8

Grażyna Koba

Lubię to!

PODRĘCZNIK DO INFORMATYKI
DLA KLASY ÓSMEJ SZKOŁY PODSTAWOWEJ

OZNACZENIA W E-PODRĘCZNIKU

Ikona na górze strony sygnalizuje zmiany, które obowiązują od 2024 r.

W ten sposób oznaczono treści, które nie są już wymagane.

nowa era

Twoje mocne strony

Droga Nowa Ero,

Nigdy bym nie publikowała publicznie książek wydawnictw, które działają na uczciwych zasadach.

Wasza firma jednak promuje masowy dodruk, całkowicie niepotrzebnych książek, które mogłyby zastąpione wersjami elektronicznymi!

Co prawda e-booki są dostępne na waszej stronie, jednak:

- W przeciwieństwie do fizycznej książki, licencja na e-book kończy się po roku. Oznacza to, że jeżeli moja córka chciałaby powtórzyć sobie całą wiedzę do matury, musiałabym jej kupić wszystkie wasze książki od nowa.
- Waszych e-booków nie da się pobrać! Wymagają one dostęp do internetu, co uniemożliwia ich użycie na naszej wsi, gdzie zasięg jest ograniczony.
- Wasze e-booki sprzedawane są po tej samej (albo wyższej) cenie co regularne książki. Cena e-booka powinna być niższa, gdyż e-booki wymagają elektronicznego czytnika (tabletu)!

Czas rozpocząć nową erę (o ironio), w której papier nie jest bezczelnie marnowany dla pieniędzy. Przedstawiam e-book, który spełnia wszystkie oczekiwania uczniów.

Dbajmy o środowisko, zróbmy to dla młodych pokoleń.

Spis treści

1. Arkusz kalkulacyjny

1.1. Formuły i adresowanie względne w arkuszu kalkulacyjnym 6
1.2. Funkcje oraz adresowanie bezwzględne i mieszane w arkuszu kalkulacyjnym 17
1.3. Przedstawianie danych na wykresie 31
1.4. Zastosowania arkusza kalkulacyjnego 43

2. Programowanie w języku Python

2.1. Wprowadzenie do programowania w języku Python 54
2.2. Piszemy programy w języku Python 69
2.3. Algorytmy na liczbach naturalnych 90
2.4. Algorytmy wyszukiwania 106
2.5. Algorymy porządkowania 120

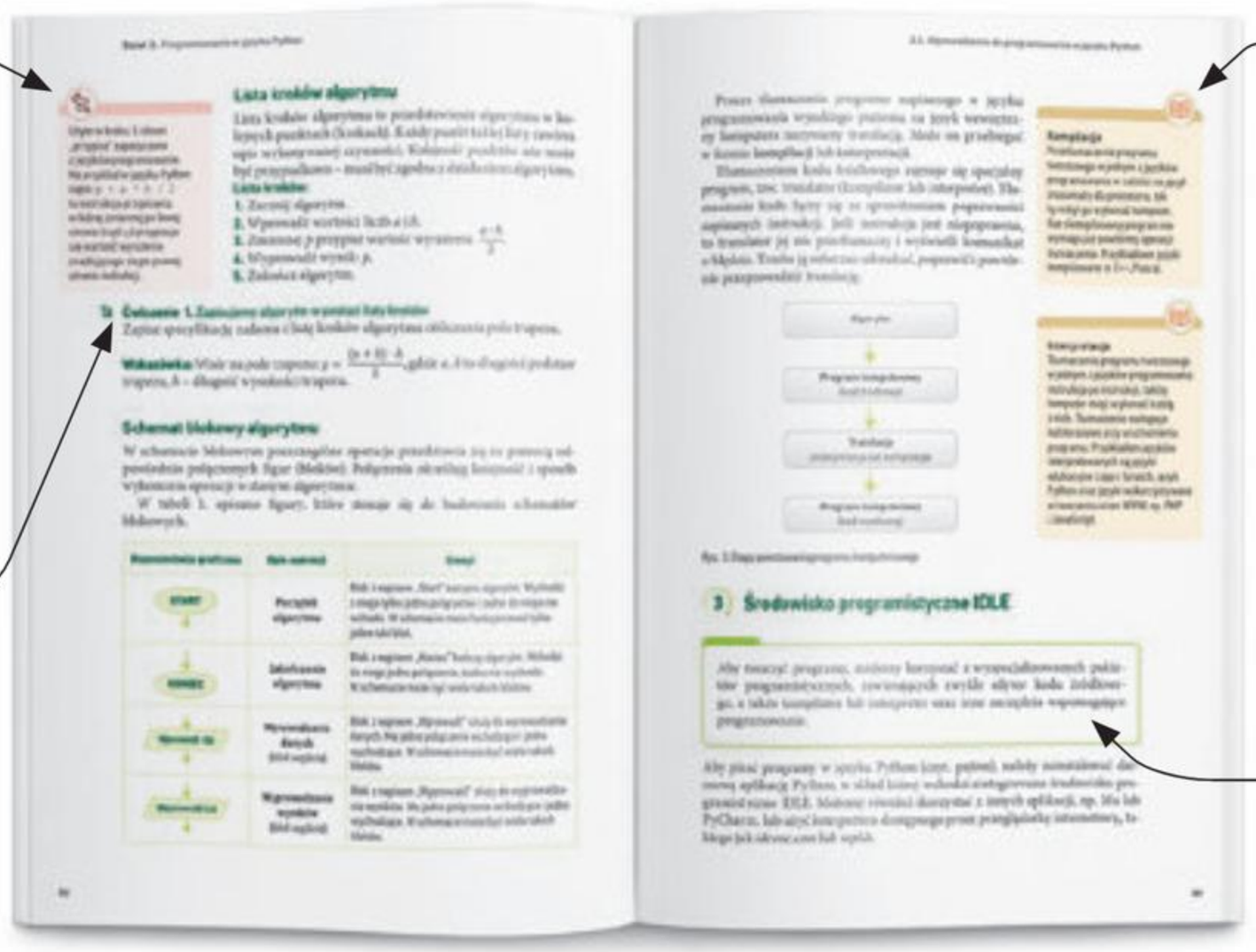
3. Programowanie w języku C++

3.1. Wprowadzenie do programowania w języku C++ 132
3.2. Piszemy programy w języku C++ 146
3.3. Algorytmy na liczbach naturalnych 165
3.4. Algorytmy wyszukiwania 183
3.5. Algorytmy porządkowania 198

4. Projekty

4.1. Dokumentacja szkolnej imprezy sportowej.....210
4.2. Sterowanie obiektem na ekranie 212
4.3. Historia i rozwój informatyki 214
4.4. Informatyka w moim przyszłym życiu 222

Poznaj swój podręcznik



Dodatkowe informacje uzupełniające lekcję o wskazówki i ciekawostki

Słowniczek zawierający wyjaśnienia ważnych terminów

Ćwiczenia pozwalające na praktyczne sprawdzenie i utrwalenie poznawanych zagadnień

Szczególnie ważne instrukcje oraz zasady

Zestawienie najważniejszych informacji podsumowujących temat

Zadania sprawdzające opanowaną wiedzę i zdobyte umiejętności

Ciekawostki z dziedziny informatyki

Zadania rozwijające zainteresowania i kreatywność

Pytania i polecenia sprawdzające zdobytą wiedzę

Arkusz kalkulacyjny

1.1. Formuły i adresowanie względne w arkuszu kalkulacyjnym

Warto powtórzyć

1. W jaki sposób tworzymy dokument w programie komputerowym?
2. Czym jest **Schowek**?
3. W jaki sposób kopiujemy fragment dokumentu (np. tekstowego) do **Schowka**?
4. Z jakich elementów składa się tabela?

Arkusz kalkulacyjny pozwala tworzyć dokumenty w postaci tabel. Umieszczamy w nich dane, na których możemy wykonywać operacje, np. obliczenia (np. finansowe, matematyczne, statystyczne) i porządkowanie (np. filtrowanie, sortowanie, analiza). W szkole arkusz kalkulacyjny wykorzystuje się do rozwiązywania zadań (w tym do zapisywania i przedstawiania na wykresach wyników pomiarów), m.in. z fizyki, matematyki, chemii i biologii. W tym temacie:

1. poznasz budowę arkusza kalkulacyjnego,
2. wykonasz obliczenia w arkuszu kalkulacyjnym,
3. dowiesz się, czym jest adresowanie względne.

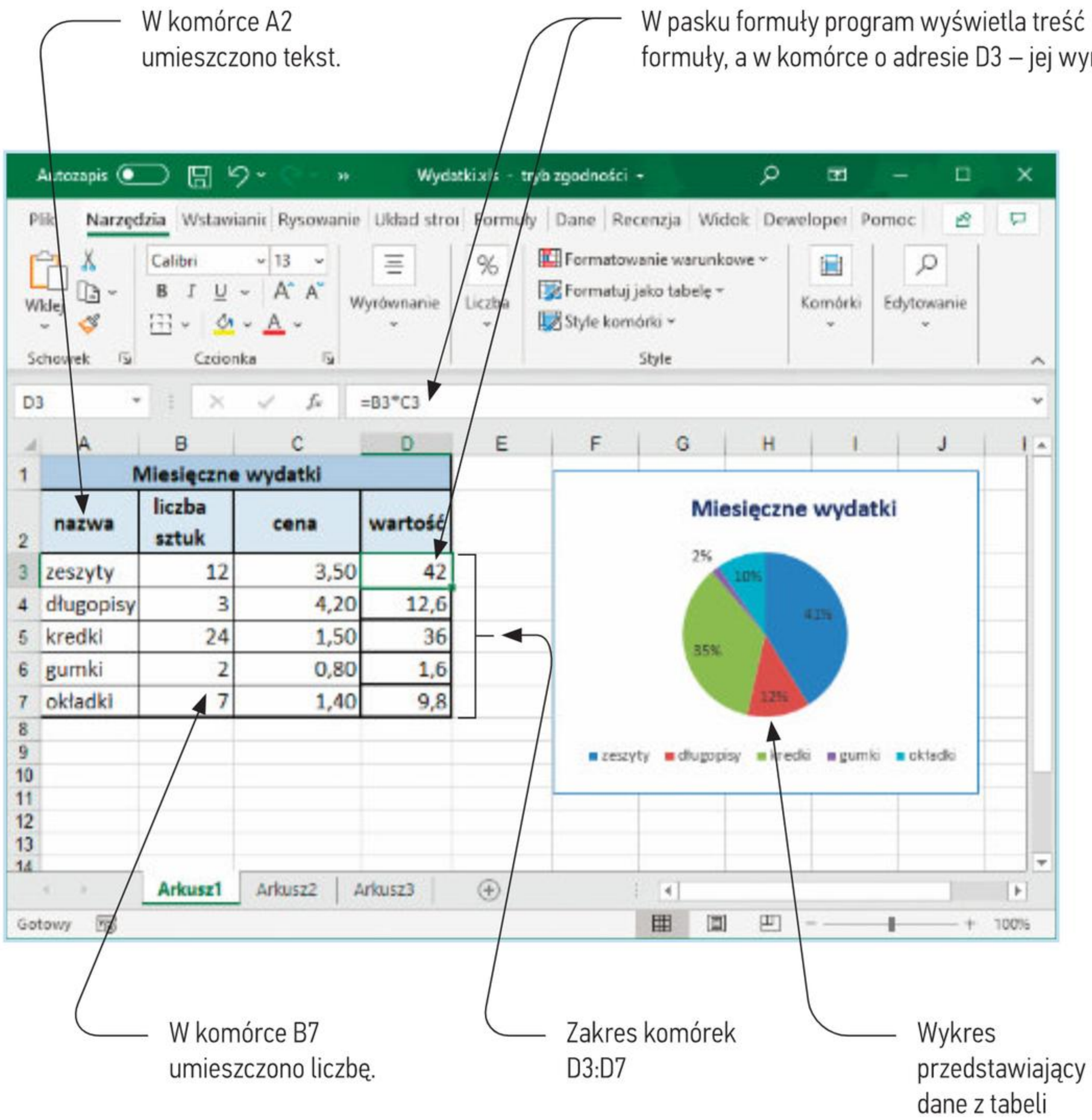
1 Budowa arkusza kalkulacyjnego

Najbardziej znanym arkuszem kalkulacyjnym jest program Microsoft Excel (rys. 1.) – powszechnie używany w wielu firmach, szkołach i różnych instytucjach. Innymi przykładami arkuszy kalkulacyjnych są programy LibreOffice Calc (rys. 2.) oraz Arkusze Google (rys. 3.).

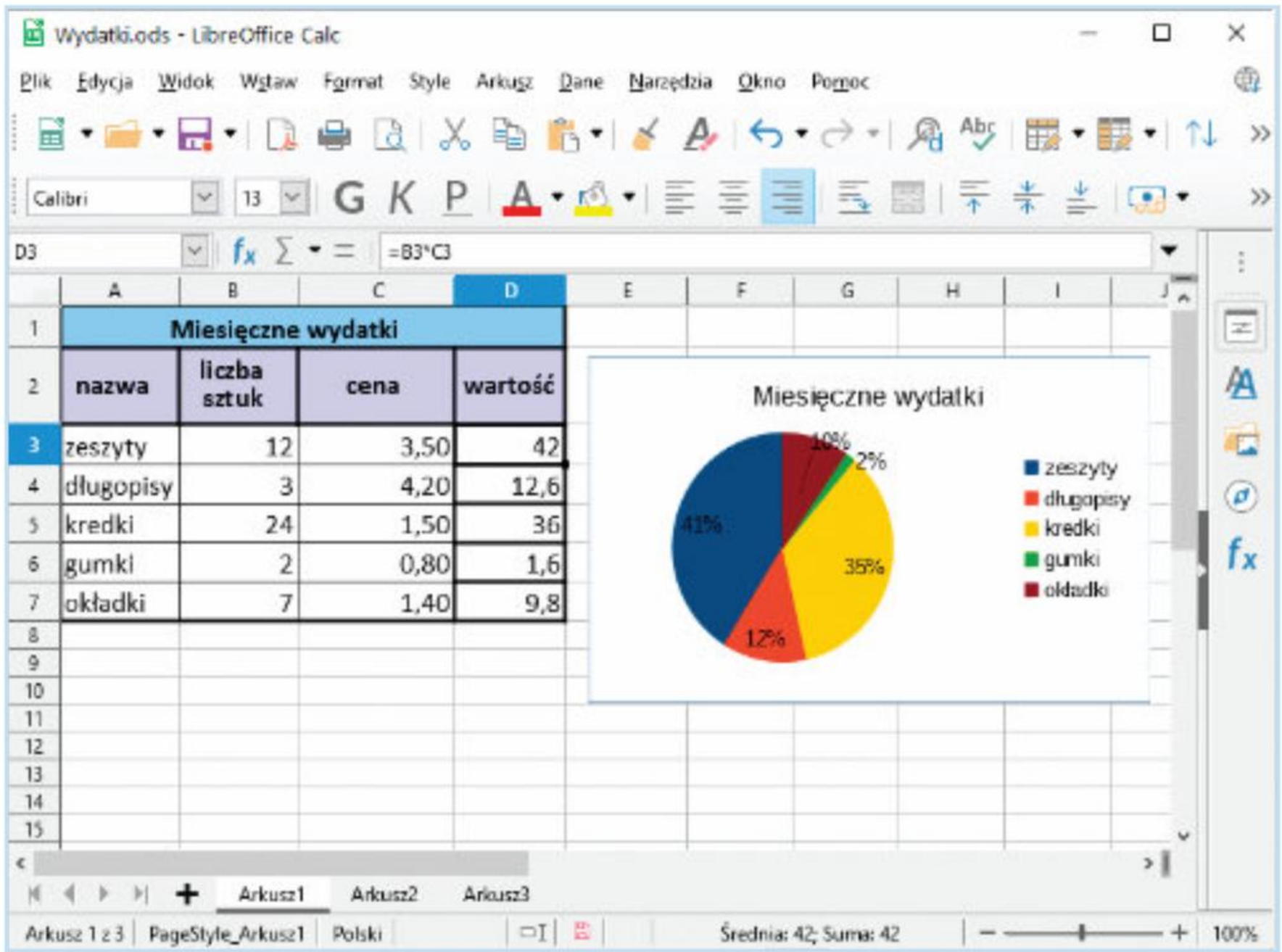
Dokument arkusza kalkulacyjnego, zwany skoroszytem, składa się z arkuszy. Każdy arkusz jest tabelą zbudowaną z wierszy i kolumn. Pojedyncze pole tabeli (powstające w miejscu przecięcia wiersza i kolumny) nazywamy komórką.

Każda komórka arkusza kalkulacyjnego ma swój adres, który określa jej położenie w dokumencie. Adres składa się z litery (ciągu liter) oznaczającej kolumnę i liczby naturalnej (większej od zera), oznaczającej numer wiersza, np. A4, F10, AB5, CF7.

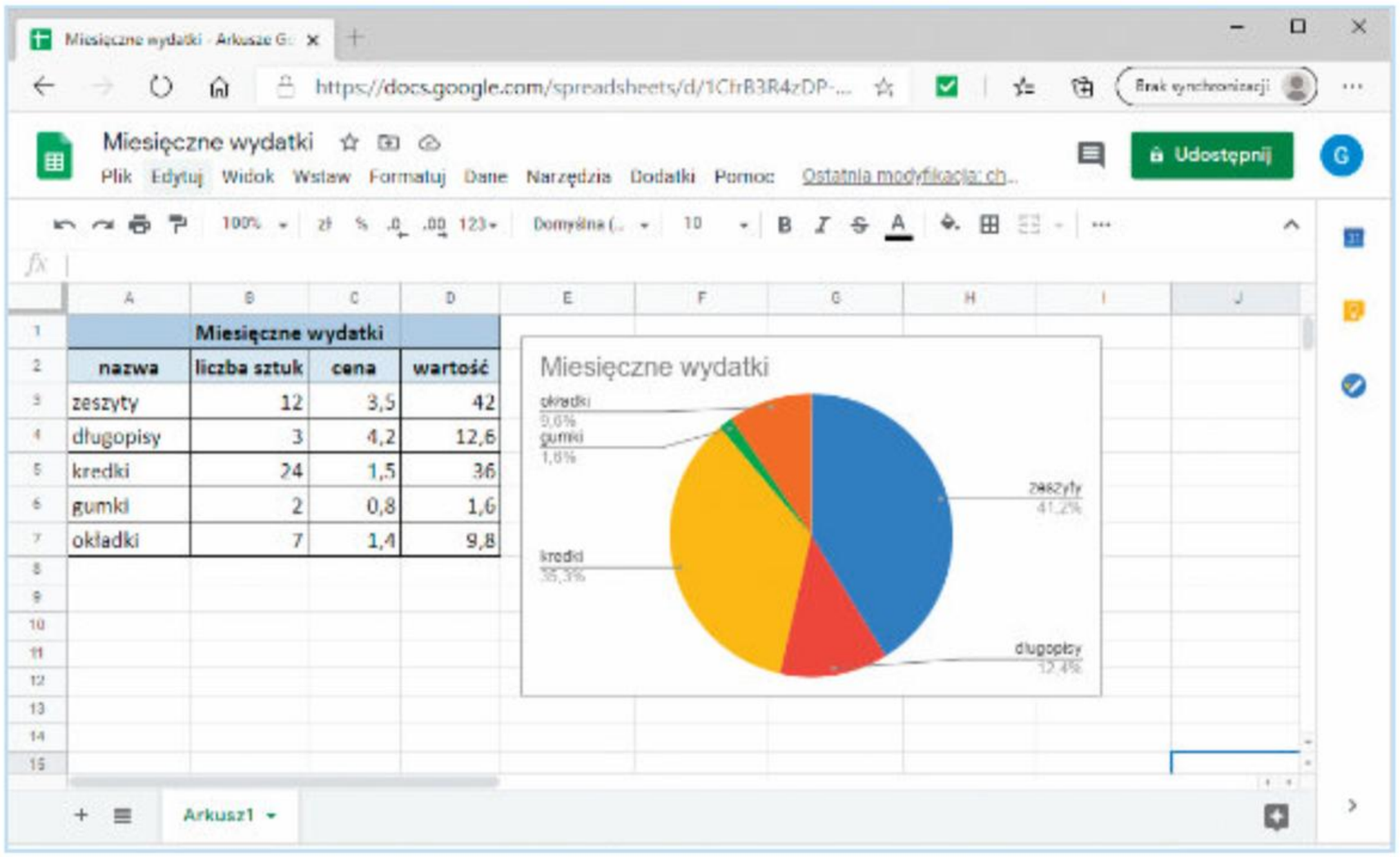
Adres A4 określa pojedynczą komórkę, natomiast adres typu A4:A20 określa zakres komórek – w tym przypadku z kolumny A od czwartego do dwudziestego wiersza. Podobnie adres B2:K2 określa zakres komórek z drugiego wiersza od kolumny B do kolumny K, a C3:F10 – zakres komórek od trzeciego wiersza kolumny C do dziesiątego wiersza kolumny F.



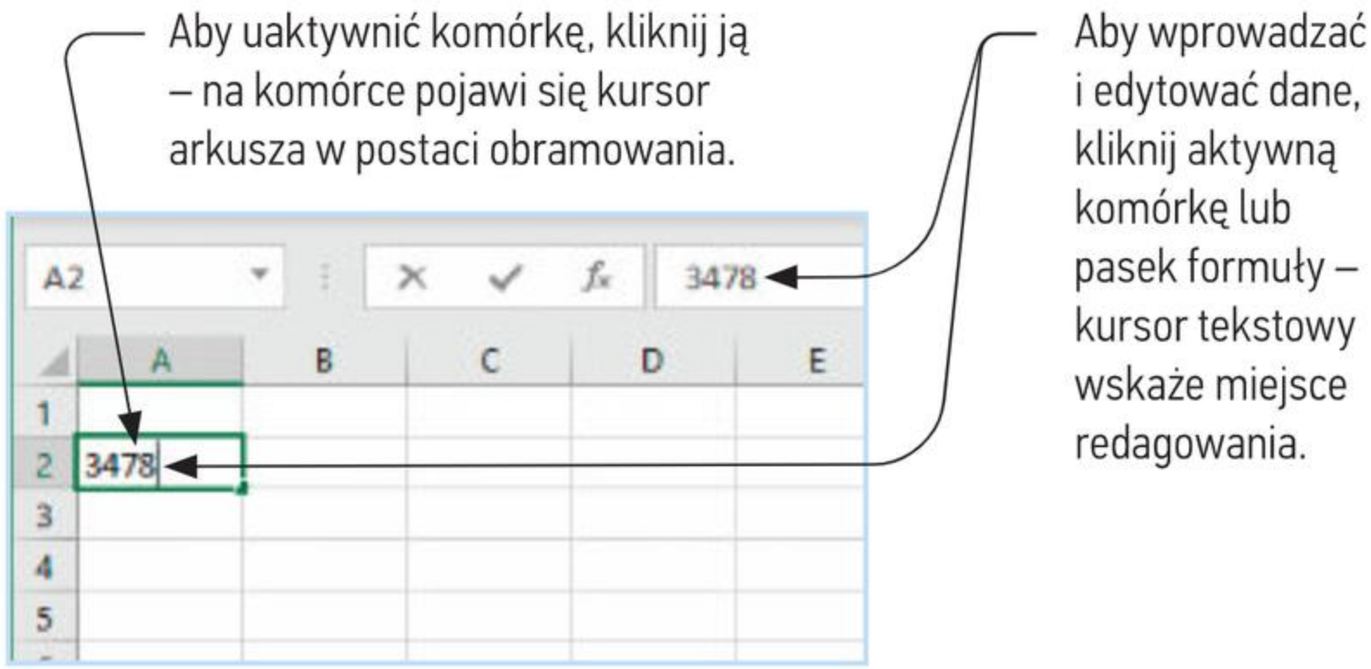
Rys. 1. Okno programu Microsoft Excel – przykładowa tabela i wykres



Rys. 2. Okno programu LibreOffice Calc – przykładowa tabela i wykres



Rys. 3. Okno aplikacji Arkusze Google – przykładowa tabela i wykres



Rys. 4. Sposoby wprowadzania danych do komórki arkusza

Aktywną komórkę możemy edytować również po naciśnięciu klawisza **F2**.

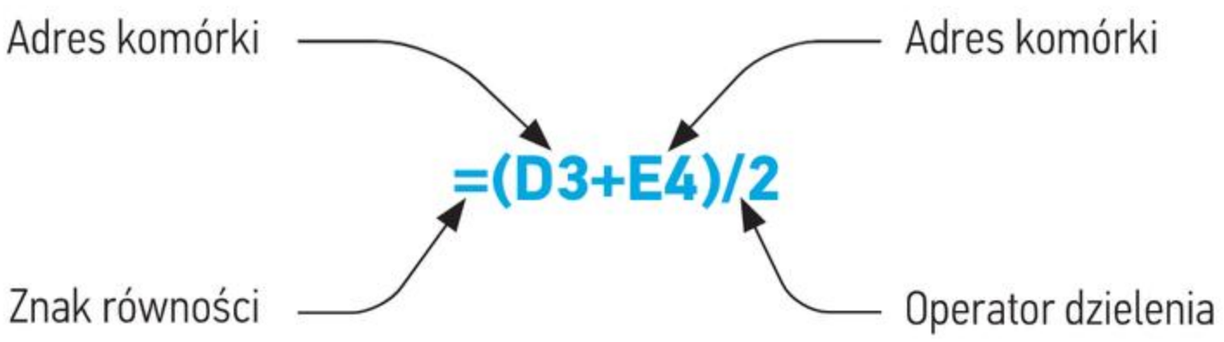
2 Obliczenia w arkuszu kalkulacyjnym

Formuła w arkuszu kalkulacyjnym to wyrażenie (np. działanie matematyczne) zapisane za pomocą operatorów i funkcji dostępnych w arkuszu kalkulacyjnym. Formuły budujemy, stosując zasady tworzenia wyrażeń algebraicznych (w tym dotyczące kolejności wykonywania działań i korzystania z nawiasów do zmiany tej kolejności), z wyjątkami podanymi w tabeli 1.

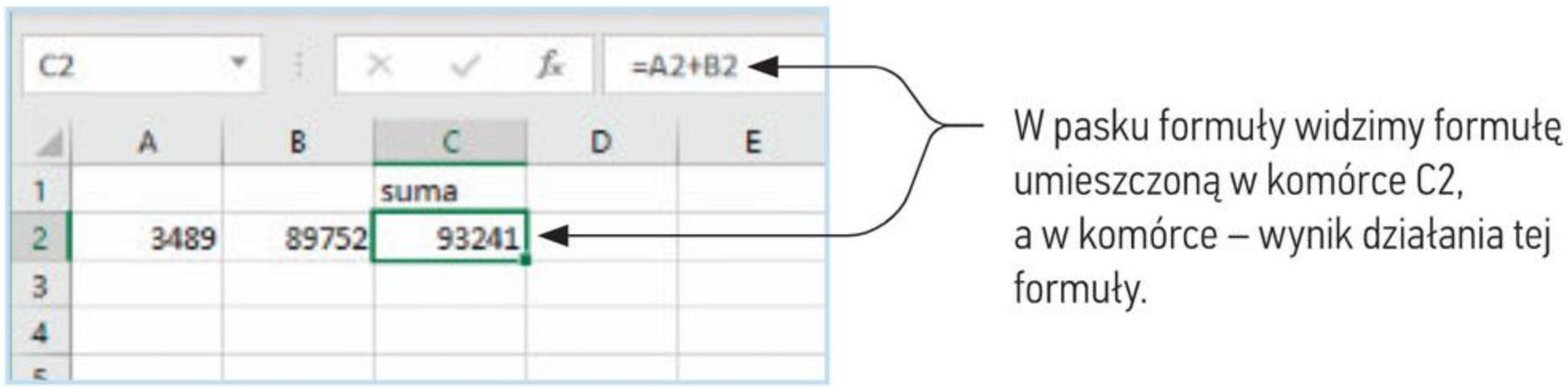
W formułach zaleca się używanie adresów komórek, a nie konkretnych, wprowadzonych do nich wartości (tak jak w matematyce lub fizyce we wzorach używa się nazw zmiennych). Użycie w formule konkretnych wartości umożliwi tylko jednokrotne obliczenie wartości wyrażenia (jak na kalkulatorze), np. =(234-189)^2 poda w komórce wynik działania $(234 - 189)^2$, czyli 2025.

Zasada	Operator	Przykłady
formułę zaczynamy od znaku równości	=	=A1+B1
używamy standardowych symboli dodawania i odejmowania	+ –	=A1+A2+1 =B8-10
symbole matematyczne mnożenia, dzielenia i potęgowania w formułach zastępujemy odpowiednio znakami	* / ^	=G5*2 =F7/3 =D6^2
stosujemy wyłącznie nawiasy okrągłe	( )	=(A1+B1)*(2+2*(C1-D1))

Tabela 1. Zasady tworzenia formuł w arkuszu kalkulacyjnym



Rys. 5. Przykładowa formuła w arkuszu kalkulacyjnym



Rys. 6. Fragment tabeli arkusza kalkulacyjnego – ćwiczenie 1.

Zmiana wartości liczbowych w komórkach arkusza kalkulacyjnego powoduje automatyczne przeliczenie wartości każdej formuły odwołującej się do tych komórek (zawierającej adresy tych komórek).

Dobrym nawykiem jest zapisywanie tworzonego dokumentu w pliku pod nazwą, która określa jego zawartość.

Ćwiczenie 1. Tworzymy formułę w arkuszu kalkulacyjnym

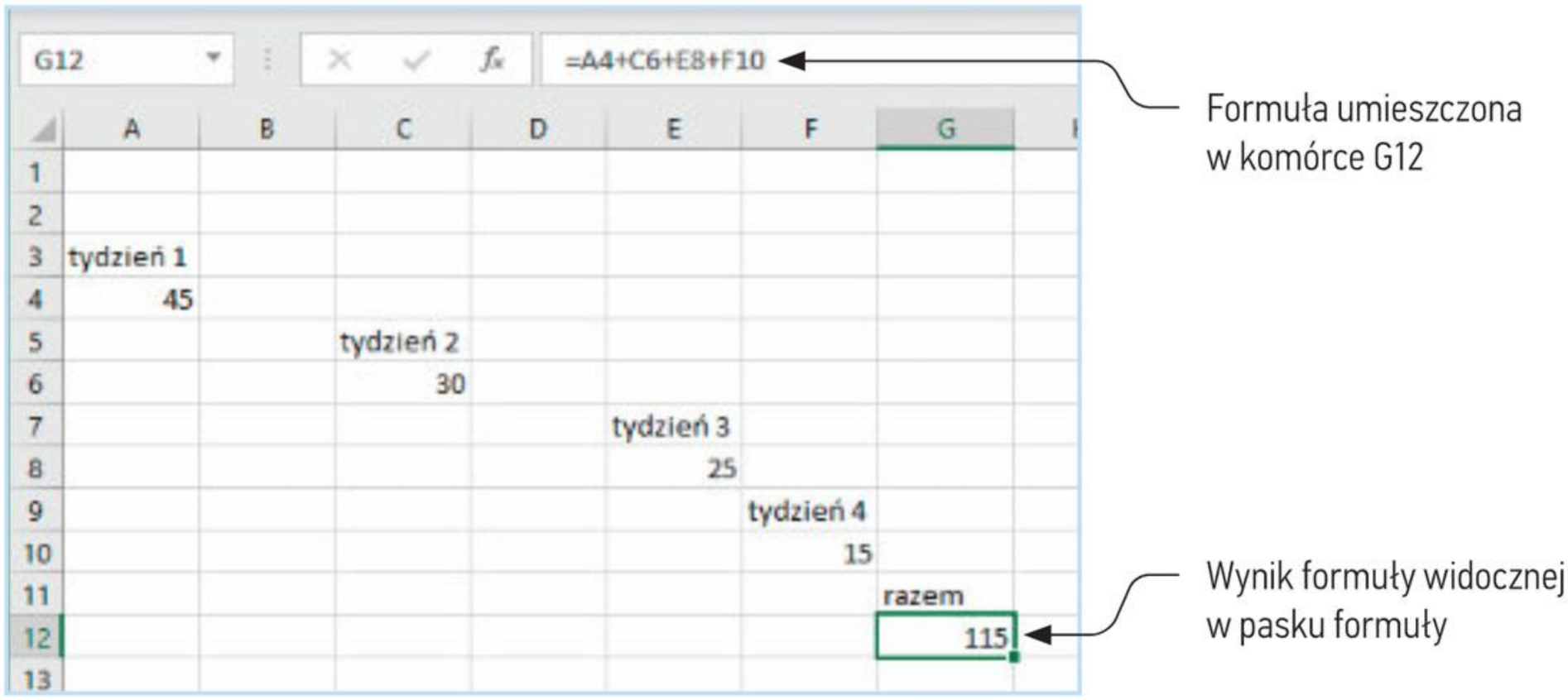
1. Otwórz nowy plik arkusza kalkulacyjnego. Wpisz do komórki A2 liczbę 3489, a do komórki B2 – liczbę 89752.
2. W komórce C2 umieść formułę obliczającą sumę tych dwóch liczb, używając adresów komórek, w których je zapisano.
3. Zmień kilka razy zawartość komórek A2 i B2. Zwróć uwagę na zmiany wartości w komórce C2. Utwórz folder *Arkusze* i zapisz w nim utworzony plik pod nazwą *Suma*.

Do jednej komórki arkusza najlepiej wprowadzać informacje jednego rodzaju, np. jedną liczbę, tekst, formułę. Liczby należy umieszczać w osobnych komórkach, jeśli program ma wykonać na nich obliczenia.

Jeśli w komórce zawierającej formułę pojawi się komunikat #ARG!, oznacza to, że program odczytał zawartość komórki o adresie wpisanym w formule jako tekst, a nie liczbę, lub że nie można wykonać obliczeń z jej udziałem.

Ćwiczenie 2. Oceniamy skutki wprowadzenia do jednej komórki wielu różnych danych

1. Otwórz nowy plik arkusza kalkulacyjnego. Wprowadź do komórek A2 i B2 po dwie liczby oddzielone spacją. W komórce A4 umieść napis „liczba 23”, a w B4 – „liczba 37”.
2. Sprawdź, jaki otrzymasz wynik, gdy wpiszesz do komórki C2 formułę na obliczenie sumy zawartości komórek A2 i B2. Następnie sprawdź, jaki będzie wynik sumowania zawartości komórek A4 i B4.
3. Wyjaśnij, dlaczego program nie wyświetlił poprawnie wyników obliczeń.



Rys. 7. Fragment tabeli arkusza kalkulacyjnego – ćwiczenie 3.

Ćwiczenie 3. Sprawdzamy, czy formułę można wpisać do dowolnej komórki

1. Twój kolega wpisał w różnych wierszach i kolumnach kwoty, jakie otrzymywał co tydzień od rodziców, i chciałby obliczyć, ile w sumie zgromadził pieniędzy. Do komórki A4 wpisał liczbę 45, do C6 – 30, do E8 – 25, a do F10 – 15.
2. Czy formułę na obliczenie sumy można wpisać do dowolnej komórki? Uzasadnij odpowiedź, a następnie umieść formułę na obliczenie sumy liczb w wybranej komórce (innej niż na rysunku 7.).
3. Dane z punktu 1. wprowadź do kolejnych komórek dwóch kolumn: w kolumnie A – opisy, w kolumnie B – liczby. W komórce B5 umieść formułę na obliczenie sumy podanych liczb.
4. Porównaj sposoby wprowadzenia danych do obliczeń zastosowane w punktach 1. i 3.
5. Zapisz plik pod nazwą *Tygodniówki*.

Formułę można umieścić w komórce o dowolnym adresie. Ważne są adresy komórek zawarte w tej formule – muszą dotyczyć danych, które chcemy wykorzystać w obliczeniach.

Ćwiczenie 4. Obliczamy wydatki

1. Oblicz swoje miesięczne wydatki. Zaprojektuj tabelę w arkuszu kalkulacyjnym. Wpisz do niej nazwy kupionych towarów, ich ceny i liczbę sztuk.
2. Oblicz wartość towarów każdego rodzaju (cena pomnożona przez liczbę sztuk).
3. Zapisz plik pod nazwą *Moje wydatki*.

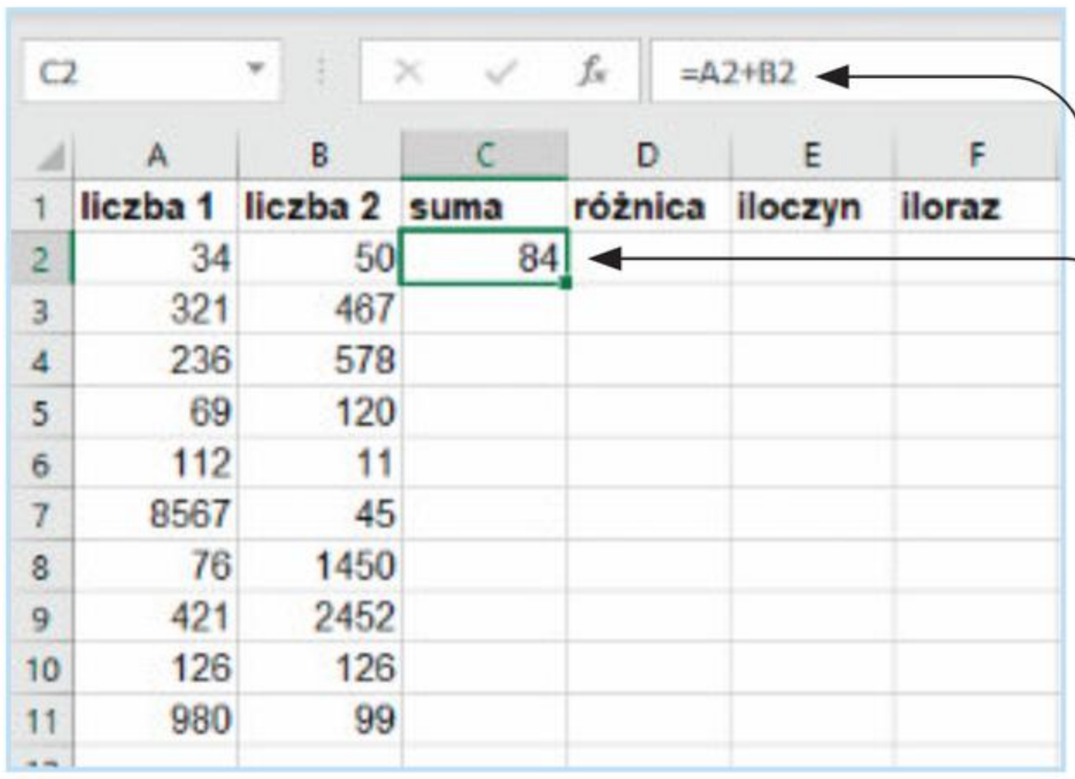
Wskazówka: Możesz skorzystać z tabeli przedstawionej na rysunku 1.

3 Adresowanie względne

Zaletą arkusza jest to, że nie trzeba przepisywać tej samej formuły wiele razy. Zapisaną w komórce formułę można skopiować, a następnie wkleić do komórek, w których chcemy wykonać te same obliczenia dla innych danych.

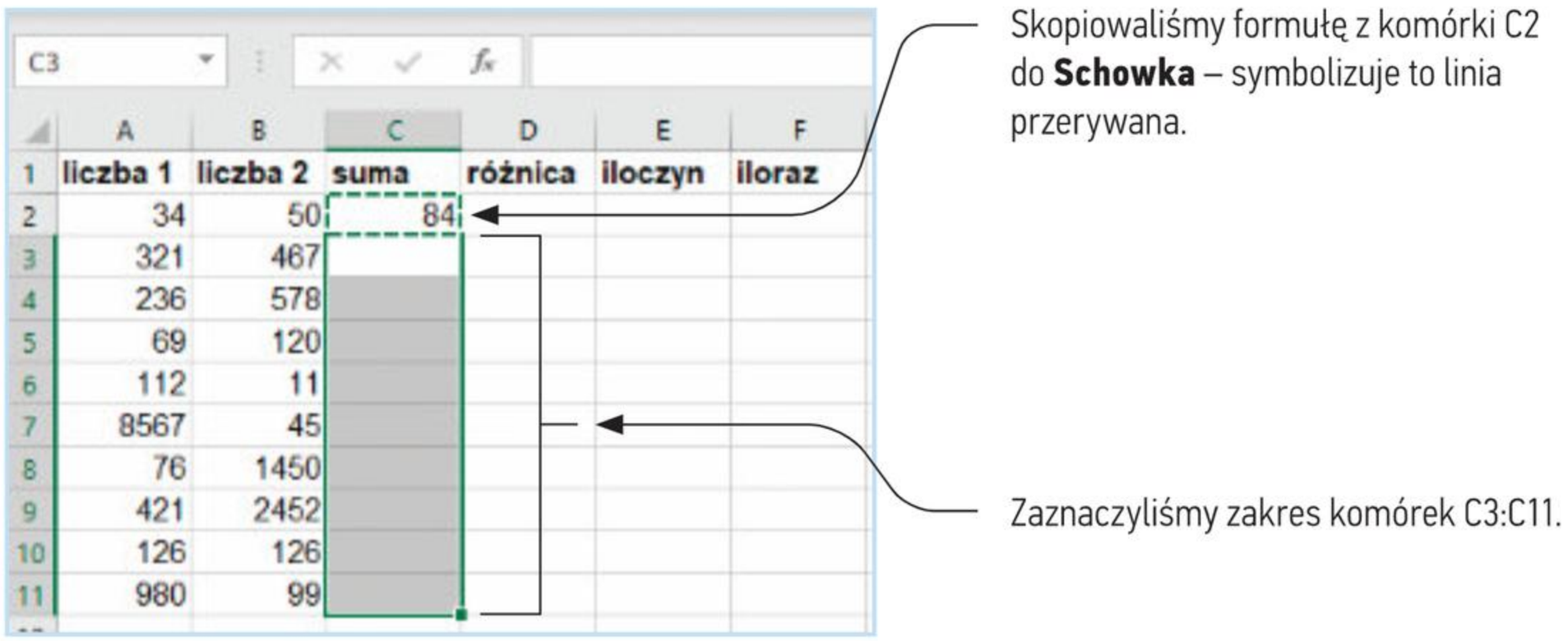
Adres względny
Adres komórki arkusza kalkulacyjnego, który zmienia się zgodnie z kierunkiem kopiowania (przy kopiowaniu w pionie zmieniają się numery wierszy, a przy kopiowaniu w poziomie – symbole kolumn).

Praca w arkuszu kalkulacyjnym opiera się na adresowaniu względnym, ułatwiającym wykonywanie powtarzających się obliczeń, ponieważ podczas kopiowania formuł automatycznie zmieniają się odwołania do wierszy i kolumn.



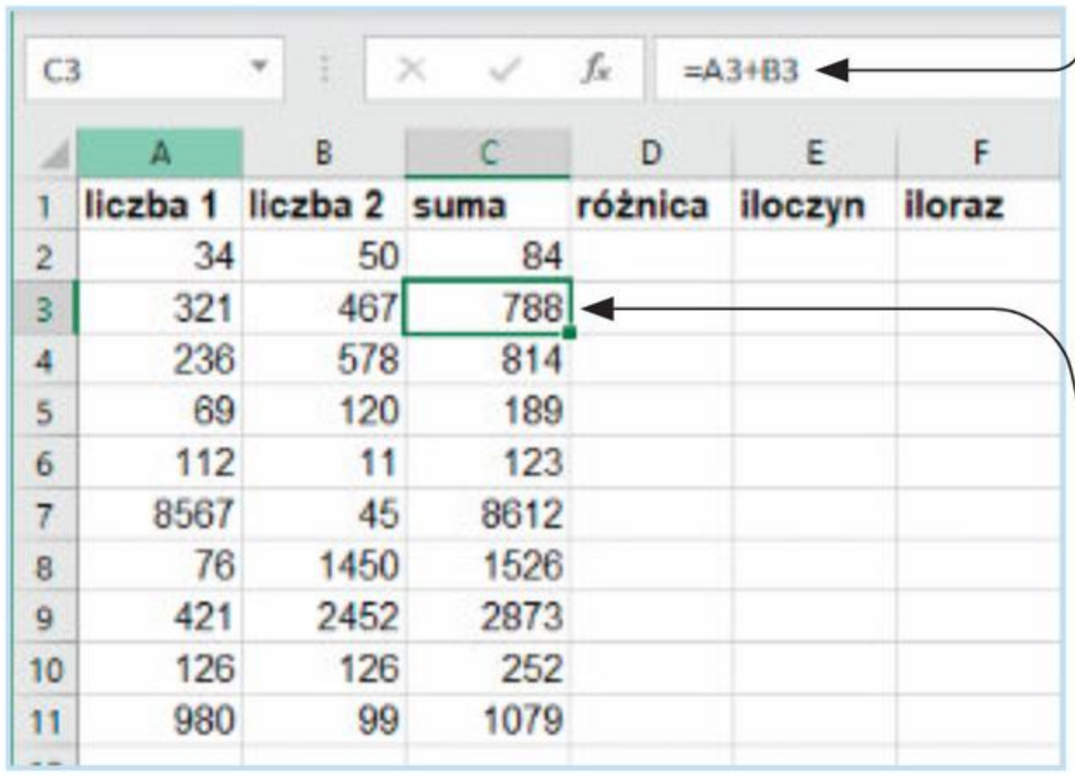
	A	B	C	D	E	F
1	liczba 1	liczba 2	suma	różnica	iloczyn	iloraz
2	34	50	84			
3	321	467				
4	236	578				
5	69	120				
6	112	11				
7	8567	45				
8	76	1450				
9	421	2452				
10	126	126				
11	980	99				

Rys. 8a. Tabela – ćwiczenie 5.



	A	B	C	D	E	F
1	liczba 1	liczba 2	suma	różnica	iloczyn	iloraz
2	34	50	84			
3	321	467				
4	236	578				
5	69	120				
6	112	11				
7	8567	45				
8	76	1450				
9	421	2452				
10	126	126				
11	980	99				

Rys. 8b. Kopiowanie formuły z komórki arkusza kalkulacyjnego do Schowka – ćwiczenie 5.



Rys. 8c. Wklejanie formuły ze Schowka do komórek arkusza kalkulacyjnego – ćwiczenie 5.

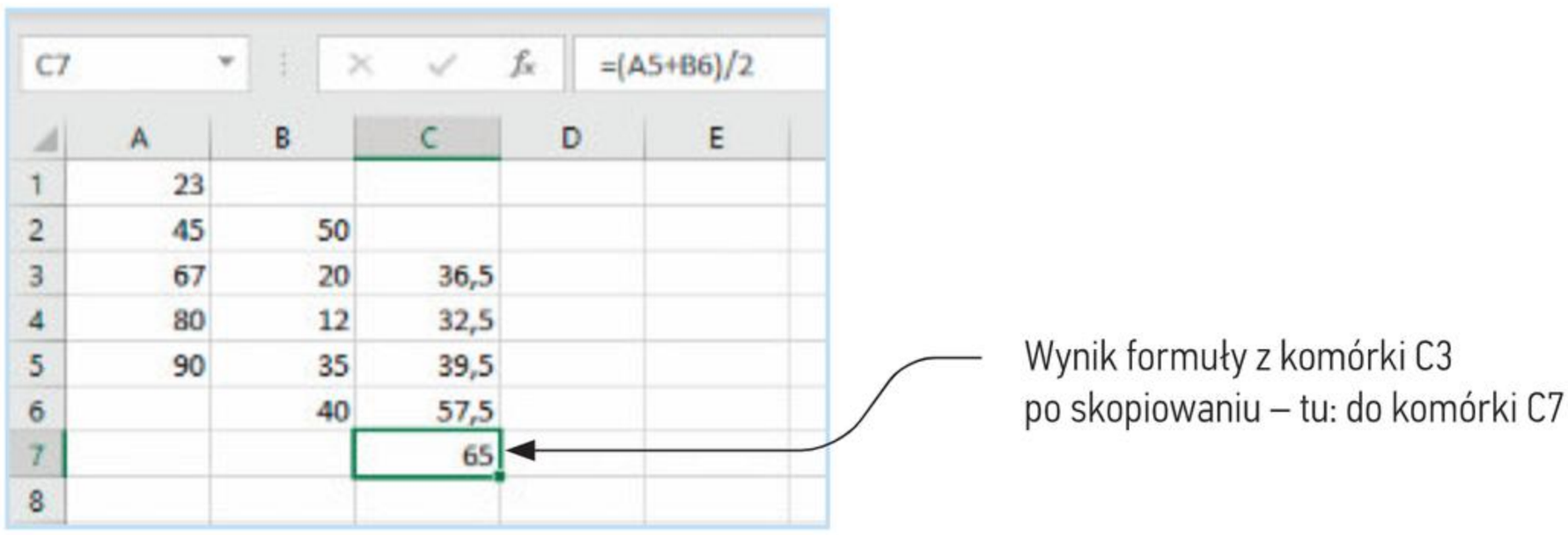
Ćwiczenie 5. Kopiujemy formuły i poznajemy zasadę adresowania względnego

1. Utwórz nowy plik arkusza kalkulacyjnego. Wpisz w kolumnach A i B po dziesięć liczb, zaczynając od komórki A2 i B2. W komórkach A1:F1 umieść napisy pokazane na rysunku 8a.
2. W komórce C2 umieść formułę obliczającą sumę liczb z komórek A2 i B2. Skopiuj ją do komórek C3:C11. Sprawdź postać formuły w komórce C8. Wyjaśnij, dlaczego jest poprawna.
3. W komórce D2 umieść formułę obliczającą różnicę liczb z komórek A2 i B2, w E2 – iloczyn, w F2 – iloraz. Oblicz odpowiednie wartości dla liczb w pozostałych wierszach: różnicę, iloczyn i iloraz. W tym celu skopiuj odpowiednie formuły.
4. Zapisz plik pod nazwą *Obliczenia*.

Wskazówka: Można skopiować jednocześnie formuły zapisane w komórkach C2, D2, E2 i F2 do komórek z zakresu C3:F11.

Po wklejeniu formuły ze **Schowka** w odpowiednich komórkach od razu wyświetlają się wyniki.

Formułę można również skopiować metodą **przeciągnij i upuść**, przeciągając zawartość komórki z formułą na komórki w tej samej kolumnie lub wierszu. Należy w tym celu wskazać kursorem myszy uchwyt wypełniania znajdujący się w prawym dolnym rogu komórki (kursor myszy zmieni się na +).



Rys. 9. Fragment tabeli – ćwiczenie 6.

Ćwiczenie 6. Stosujemy adresowanie względne

1. Utwórz nowy plik arkusza kalkulacyjnego. Do kolumn A i B wpisz po pięć liczb, zaczynając od komórki A1 i od komórki B2.
2. Do komórki C3 wpisz formułę na obliczenie średniej arytmetycznej liczb z komórek A1 i B2.
3. Skopiuj formułę z komórki C3 do komórek C4:C7.
4. Dlaczego w komórce C7 formuła ma postać =(A5+B6)/2? W wyjaśnieniu uwzględnij położenie komórek, których adresy występują w formule zawartej w komórce C3, względem tej komórki.
5. Zapisz plik pod nazwą *Średnia*.

Zapamiętaj

- Dokument arkusza kalkulacyjnego, zwany skoroszytem, składa się z arkuszy. Arkusz jest tabelą zbudowaną z wierszy i kolumn. Pojedyncze pole tabeli to komórka.
- Każda komórka ma jednoznacznie określony adres: literę (litery) określającą kolumnę i numer określający wiersz. Komórka jest miejscem przechowywania danych (liczb, tekstów, formuł) i ich edytowania.
- Tworząc formuły w arkuszu kalkulacyjnym, lepiej używać adresów komórek, w których znajdują się dane – można wówczas korzystać ze wszystkich własności arkusza, zwłaszcza z możliwości kopiowania formuł.
- Adresowanie względne polega na tym, że ogólna postać formuły po skopiowaniu nie zmienia się, natomiast odpowiednio zmieniają się adresy komórek, do których odwołuje się formuła.

Serwis dla ciekawskich

Microsoft Excel 2007 zawierał błąd powodujący niepoprawne wykonywanie niektórych obliczeń. Np. wpisanie formuły =77,1*850, której wynikiem powinna być liczba 65 535, powodowało wyświetlenie liczby 100 000. Błąd poprawiono wkrótce po jego ujawnieniu w roku 2007.

OKO W OKO Z MONITOREM

Pytania i polecenia

1. Czy w arkuszu kalkulacyjnym różne komórki mogą mieć taki sam adres? Uzasadnij odpowiedź.
2. Wyjaśnij na przykładzie, czy zmiana wartości liczbowych w komórkach arkusza kalkulacyjnego powoduje automatyczne przeliczenie wartości każdej formuły odwołującej się do tych komórek.
3. Dlaczego ważne jest, które adresy komórek umieszczamy w formule?
4. Co oznacza stwierdzenie, że adresy komórek zawarte w formule zmieniają się zgodnie z kierunkiem kopiowania?

Zadania

1. W arkuszu kalkulacyjnym, który otrzymasz od nauczyciela, znajduje się tabela z danymi dotyczącymi powierzchni i liczby mieszkańców krajów Ameryki Północnej.
   a. Oblicz gęstość zaludnienia każdego z tych krajów, dzieląc liczbę mieszkańców przez powierzchnię. Czy dla każdego kraju musisz wpisywać oddzielnie formułę obliczającą gęstość zaludnienia? Zapisz plik pod nazwą *Ameryka*.
   b. Korzystając z internetu lub innych źródeł, dodaj dane dotyczące krajów Ameryki Południowej i Środkowej. Oblicz gęstość zaludnienia każdego z tych krajów. Zapisz plik pod tą samą nazwą.
2. W arkuszu kalkulacyjnym, który otrzymasz od nauczyciela, znajduje się tabela pokazana na rysunku 10. W kolumnach A i B dodaj jeszcze po osiem wartości *a* i *b*. Wpisz w odpowiednich kolumnach drugiego wiersza formuły obliczające wartości wyrażeń zapisanych w pierwszym wierszu tabeli, gdzie *a* to liczba z komórki A2, *b* to liczba z komórki B2. Wykonaj obliczenia dla wszystkich danych w pozostałych wierszach. Jaką metodę zastosujesz? Porównaj wyniki otrzymane w kolumnie C z wynikami z kolumny D, wyniki z kolumny E – z wynikami z kolumny F oraz wyniki z kolumny G – z wynikami z kolumny H. Jakie są twoje spostrzeżenia? Zapisz plik pod nazwą *Wzory*.

Wskazówka: Wzory: $a^2 - b^2 = (a + b)(a - b)$, $(a + b)^2 = a^2 + 2ab + b^2$, $(a - b)^2 = a^2 - 2ab + b^2$, zwane są wzorami skróconego mnożenia.

Uwaga: W matematyce zapis typu *2ab* oznacza 2 · a · b.

	A	B	C	D	E	F	G	H
1	**a**	**b**	**a^2-b^2**	**(a+b)(a-b)**	**$(a+b)^2$**	**a^2+2ab+b^2**	**$(a-b)^2$**	**a^2-2ab+b^2**
2	4	5						
3	8	2						

Rys. 10. Fragment tabeli – zadanie 2.

3. Zaprojektuj w arkuszu kalkulacyjnym tabelę do obliczania pól prostokąta, trapezu i trójkąta. Wpisz odpowiednie dane dla trzech figur każdego rodzaju i przeprowadź obliczenia. Zapisz plik pod nazwą *Pola*.

4. W tabeli pokazanej na rysunku 11. zawarto fikcyjne dane dotyczące godzin nieobecności uczniów. Podobną tabelę otrzymasz od nauczyciela. Oblicz: sumę wszystkich godzin nieobecności, sumę godzin nieobecności nieusprawiedliwionych oraz jaki procent nieobecności stanowią godziny nieusprawiedliwione – za wrzesień-listopad, dla każdego ucznia i dla całej klasy. Dodaj do tabeli nowe kolumny z danymi za grudzień i styczeń. Umieść je przed kolumną RAZEM. Sprawdź poprawność utworzonych formuł. Dołącz do tabeli fikcyjne dane kolejnych pięciu uczniów. Wykonaj obliczenia dla każdego z nich. Zapisz plik pod nazwą *G_nieusp*.

	A	B	C	D	E	F	G	H	I	J	K	L
1												
2				wrzesień		październik		listopad		RAZEM		
3	Lp.	nazwisko	imię	nieobec-ności	w tym nieuspr.	nieobec-ności	w tym nieuspr.	nieobec-ności	w tym nieuspr.	nieobec-ności	w tym nieuspr.	% nieobecności nieusprawiedliw.
4	1	Bilski	Adam	1	1	6	0	8	2			
5	2	Bodycz	Hanna	2	0	9	0	12	6			
6	3	Bromska	Alina	5	0	18	0	25	6			
7	4	Broszyk	Mateusz	5	0	15	15	21	6			
8	5	Chapik	Magdalena	0	0	12	2	17	6			
9	6	Cieślak	Piotr	3	0	9	0	12	5			
10	7	Gmółka	Paweł	2	0	12	0	17	6			
11	8	Janiszak	Roman	4	0	15	0	21	6			
12	9	Januszewski	Wojciech	0	0	9	2	4	4			
13	10	Kiełaś	Zofia	2	2	9	2	12	4			
14	11	Kuchera	Elżbieta	2	0	9	0	5	5			
15	12	Kowalski	Michał	4	0	15	15	21	6			
16	13	Kozlicki	Zbigniew	2	0	9	0	12	4			
17	14	Krzysztofik	Jacek	0	0	9	0	4	4			
18	15	Majewska	Anna	4	0	15	15	21	6			
19	16	Mituł	Wojciech	4	0	15	15	6	6			
20	17	Mykowska	Beata	3	0	12	2	17	5			
21	18	Porulka	Helena	0	0	15	6	21	6			
22	19	Pyzowski	Krzysztof	4	0	18	8	25	6			
23	20	Rajewski	Przemysław	2	0	9	0	12	5			
24	21	Stapik	Arkadiusz	2	2	12	0	17	4			
25	22	Szymańska	Izabela	0	0	0	0	8	0			
26	23	Tebrus	Aleksander	0	0	9	7	12	4			
27	24	Wasilska	Grażyna	0	0	15	15	21	6			
28	25	Ząbkowski	Szczepan	4	0	12	0	5	4			
29	26	Ząbkowska	Aleksandra	4	0	12	0	12	0			
30	27	Zmudziński	Jacek	3	0	12	0	17	4			
31	28	Żuławiak	Halina	2	0	9	0	3	2			
32			RAZEM									
33												

Rys. 11. Tabela – zadanie 4.

DLA ZAINTERESOWANYCH

1. Pomóż wychowawcy w przygotowaniu semestralnej statystyki klasy. Zaprojektuj tabelę arkusza kalkulacyjnego i zapisz w pliku pod nazwą *Klasa*. Ustal z nauczycielem, jakie dane są potrzebne i jakie obliczenia należy wykonać.

1.2. Funkcje oraz adresowanie bezwzględne i mieszane w arkuszu kalkulacyjnym

Warto powtórzyć

1. Czym jest adres komórki?
2. W jaki sposób tworzymy formuły?
3. W jaki sposób kopiujemy formułę z jednej komórki do innych komórek?
4. Na czym polega adresowanie względne?

Arkusze kalkulacyjne mają wiele funkcji ułatwiających obliczenia. Aby podczas tworzenia i kopiowania formuł w arkuszu kalkulacyjnym nie pojawiły się błędy, warto dobrać właściwy rodzaj adresowania. W tym temacie:

1. zastosujesz funkcje ŚREDNIA, JEŻELI i SUMA,
2. dowiesz się, dlaczego stosujemy adresowanie bezwzględne,
3. dowiesz się, kiedy użyć adresowania mieszanego.

1 Wybrane funkcje arkusza kalkulacyjnego

Funkcje arkusza kalkulacyjnego należą do różnych kategorii, np. statystyczne (m.in. ŚREDNIA), logiczne (m.in. JEŻELI), matematyczne (m.in. SUMA), tekstowe, daty i czasu.

Rys. 1. Ogólna postać funkcji arkusza kalkulacyjnego

Jeśli funkcja rozpoczyna formułę, jej nazwa musi być poprzedzona znakiem równości.

Argumentami funkcji mogą być: odwołania do komórek, odwołania do zakresu komórek, liczby, teksty, warunki logiczne.

Jeśli stosujemy funkcję w formule, najlepiej umieścić dane do obliczeń w jednej kolumnie lub jednym wierszu.

Aby użyć funkcji w formule, można:

- kliknąć komórkę pod kolumną danych lub obok wiersza z danymi,
- wybrać potrzebną funkcję z **Biblioteki funkcji** na karcie **Formuły** (Microsoft Excel) lub w oknie **Kreator funkcji** po wybraniu opcji **Funkcja** w menu **Wstaw** (LibreOffice Calc),
- w oknie **Argumenty funkcji** (Microsoft Excel) lub **Kreatora funkcji** (LibreOffice Calc) wpisać argumenty funkcji lub sprawdzić je i ewentualnie zmodyfikować (jeśli program umieścił argumenty automatycznie).

Funkcję możemy również wpisać do komórki, umieszczając po nazwie funkcji w nawiasach okrągłych argumenty (rozdzielone średnikami, jeśli jest ich więcej).

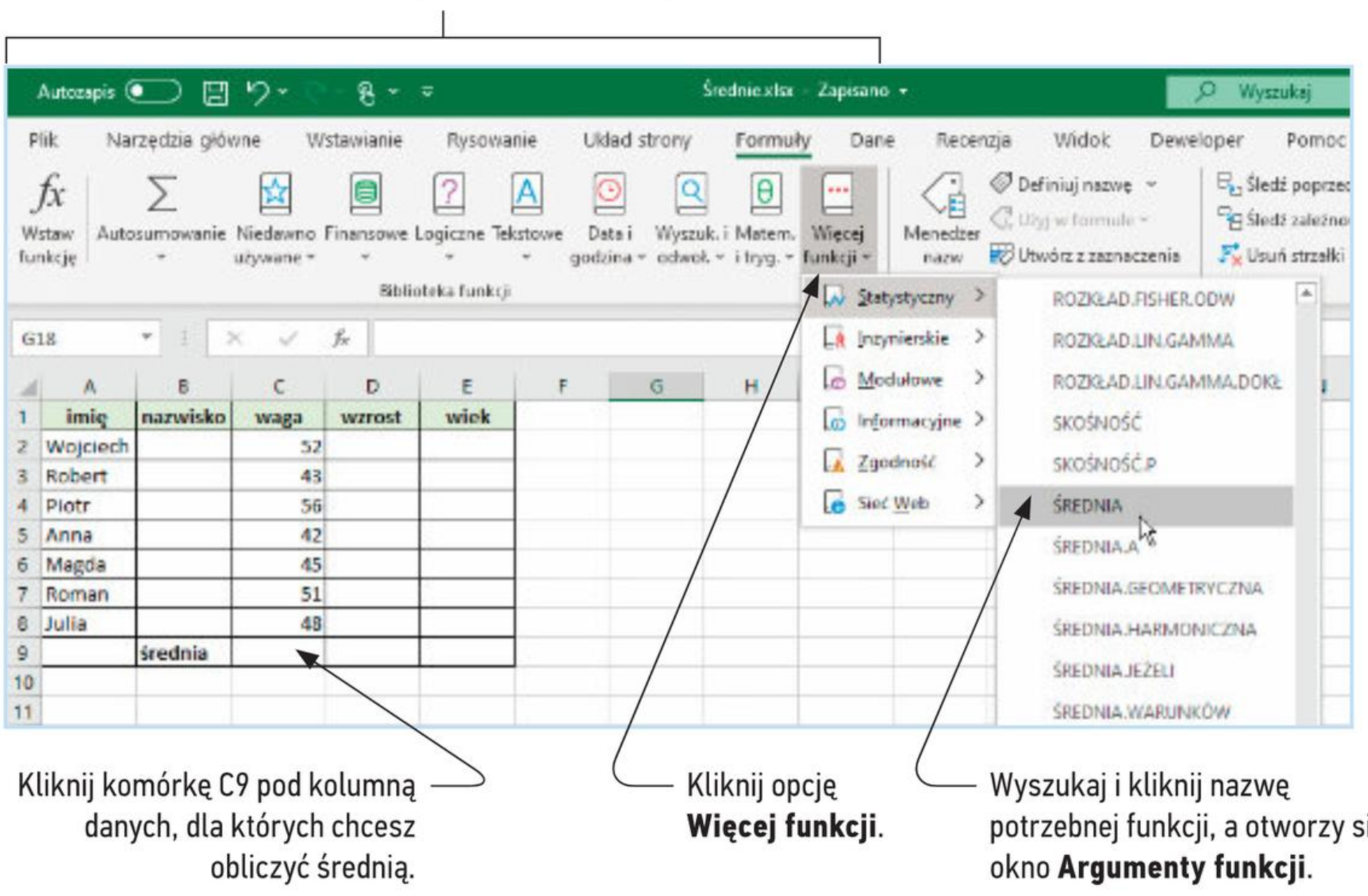
Funkcja ŚREDNIA

Funkcja ŚREDNIA należy do kategorii funkcji statystycznych i oblicza średnią arytmetyczną liczb podanych jako argumenty. Zwykle jako argumentu używa się odwołania do zakresu komórek. Argumentami mogą być również odwołania do adresów komórek lub konkretne liczby.

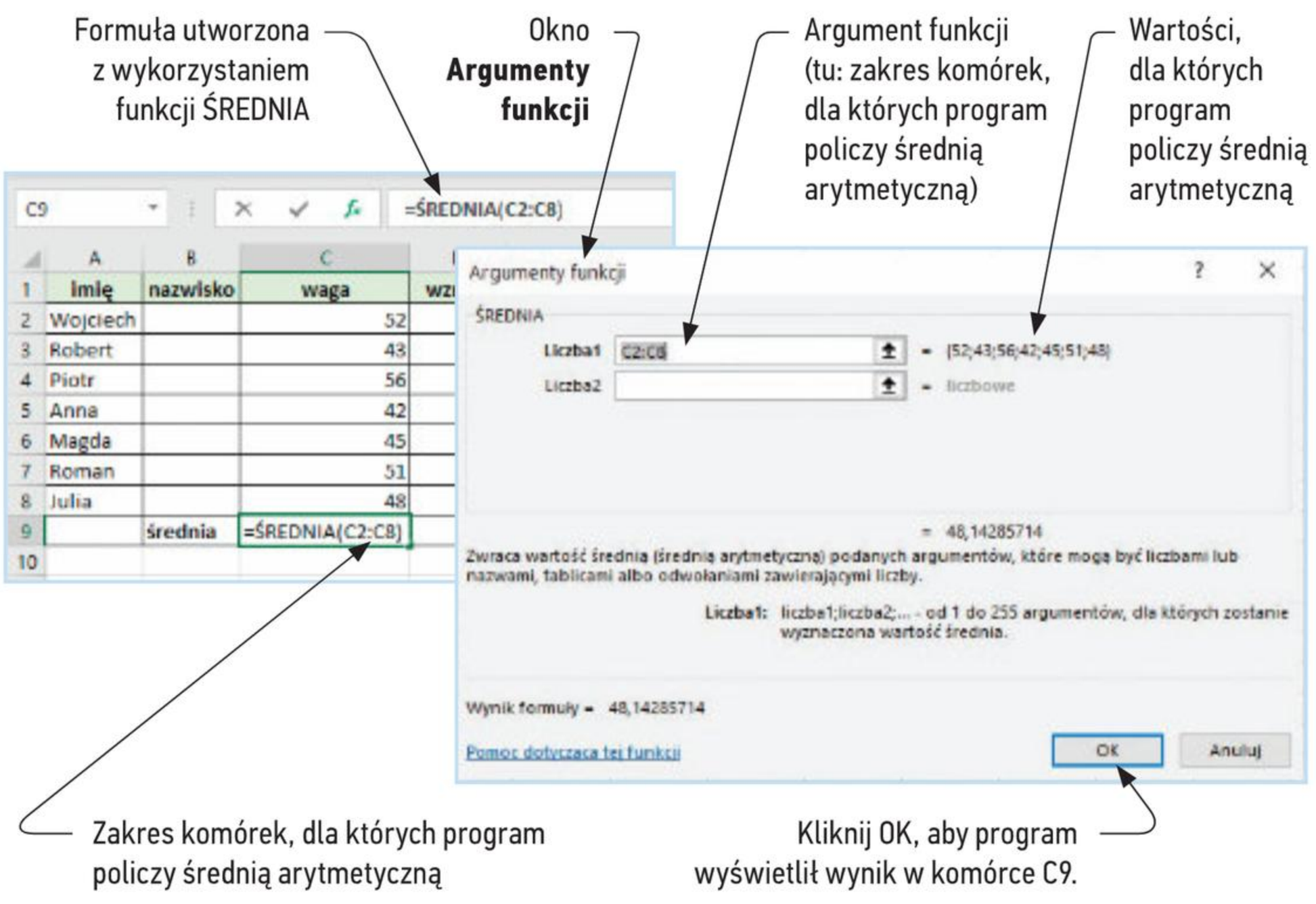
Rys. 2. Ogólna postać funkcji ŚREDNIA

Przykład	Opis działania
=ŚREDNIA(A1:A20)	średnia liczb z komórek od A1 do A20 (w zakresie komórek stosujemy dwukropek)
=ŚREDNIA(B5;C10;F12)	średnia liczb z komórek B5, C10 i F12 (argumenty rozdzielamy średnikami)
=ŚREDNIA(24;56;100)	średnia liczb 24, 56, 100

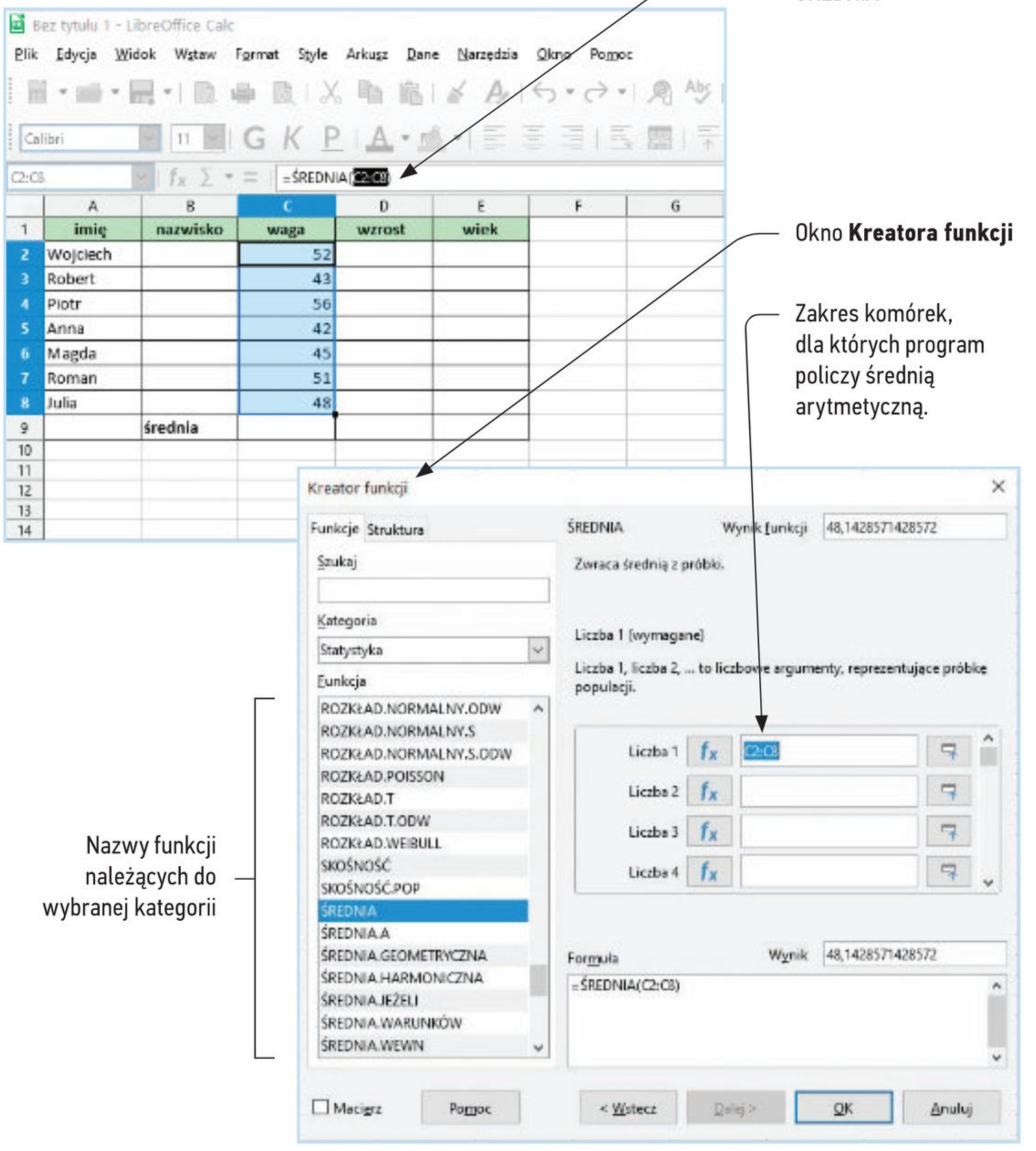
Tabela 1. Przykłady użycia funkcji ŚREDNIA w formułach



Rys. 3a. Stosowanie funkcji ŚREDNIA (Microsoft Excel) – ćwiczenie 1.



Rys. 3b. Stosowanie funkcji ŚREDNIA (Microsoft Excel) – ćwiczenie 1.



Rys. 4. Stosowanie funkcji ŚREDNIA (LibreOffice Calc) – ćwiczenie 1.

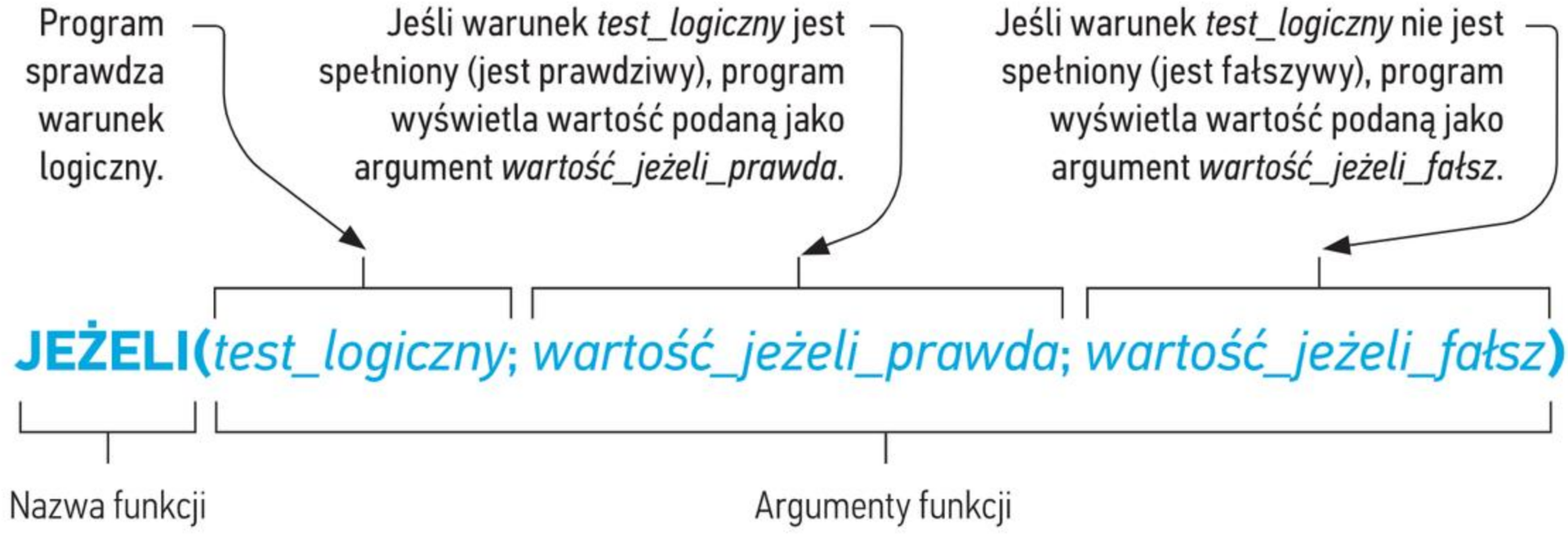
Ćwiczenie 1. Korzystamy z funkcji ŚREDNIA

1. W pliku, który otrzymasz od nauczyciela, znajduje się zestawienie wagi, wzrostu i wieku wszystkich uczniów pewnej szkoły grających w tenisa.
2. Utwórz formułę obliczającą średnią wagę zawodników.
3. Skopiuj formułę, aby policzyć pozostałe średnie. Sprawdź, jaka jest postać formuły po skopiowaniu.
4. Zapisz plik pod nazwą *Średnie*.

Funkcja JEŻELI

Funkcja JEŻELI należy do kategorii funkcji logicznych. Funkcji JEŻELI możemy użyć, aby uzależnić wyświetlanie informacji od spełnienia warunku. Za jej pomocą można sprawdzać, czy dany warunek jest spełniony czy nie, i w zależności od wyniku wyświetlać odpowiednie wartości.

Do budowania warunków używa się operatorów opisanych w tabeli 2.



Rys. 5. Ogólna postać funkcji JEŻELI

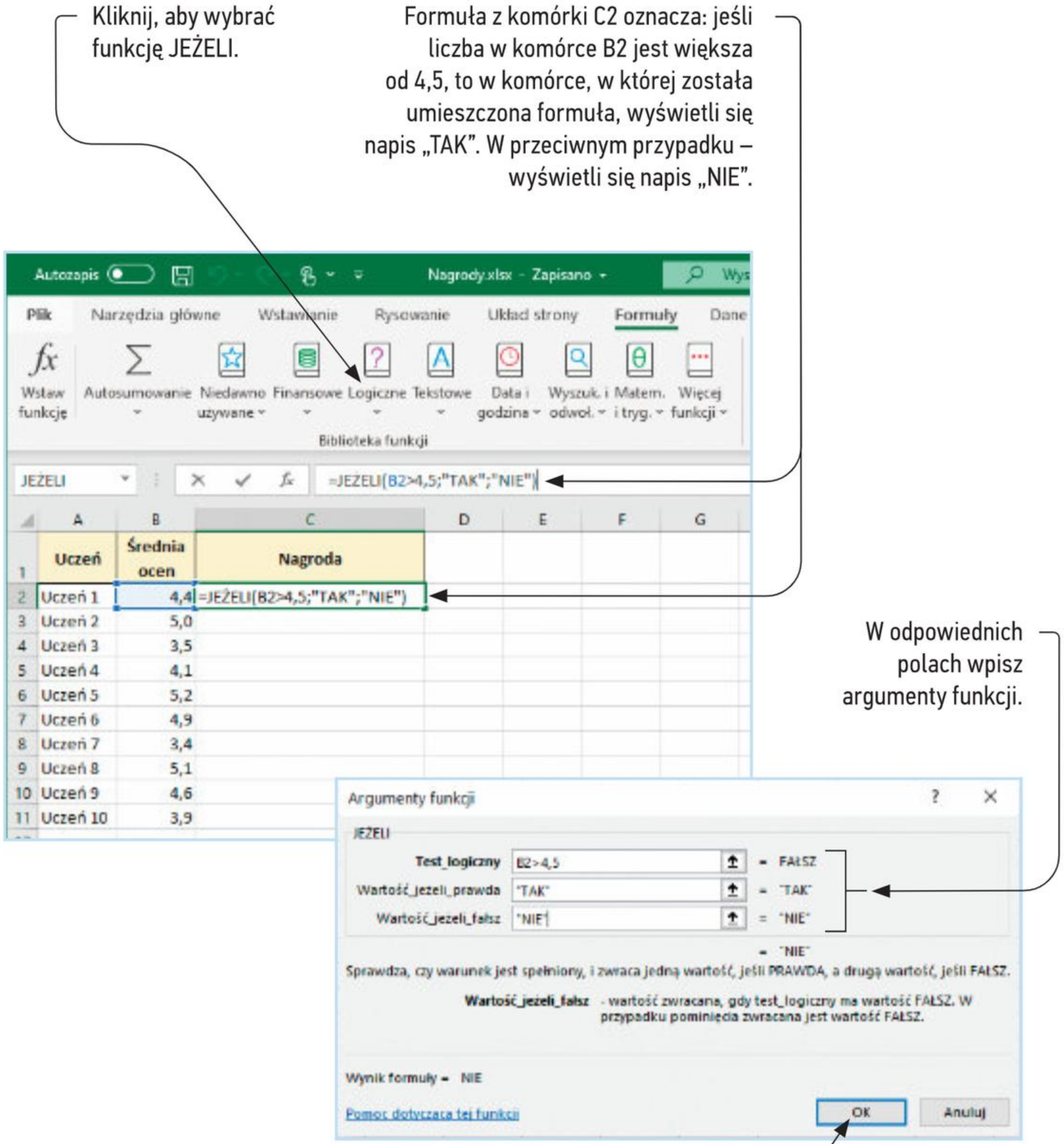
Operator	Opis	Przykład
=	równe	A1=10
<>	różne	B3<>10
>	większe niż	B4>8
<	mniejsze niż	A2<3
>=	większe niż lub równe	A2>=B2
<=	mniejsze niż lub równe	C5<=11

Tabela 2. Operatory używane do budowania warunków w funkcji JEŻELI

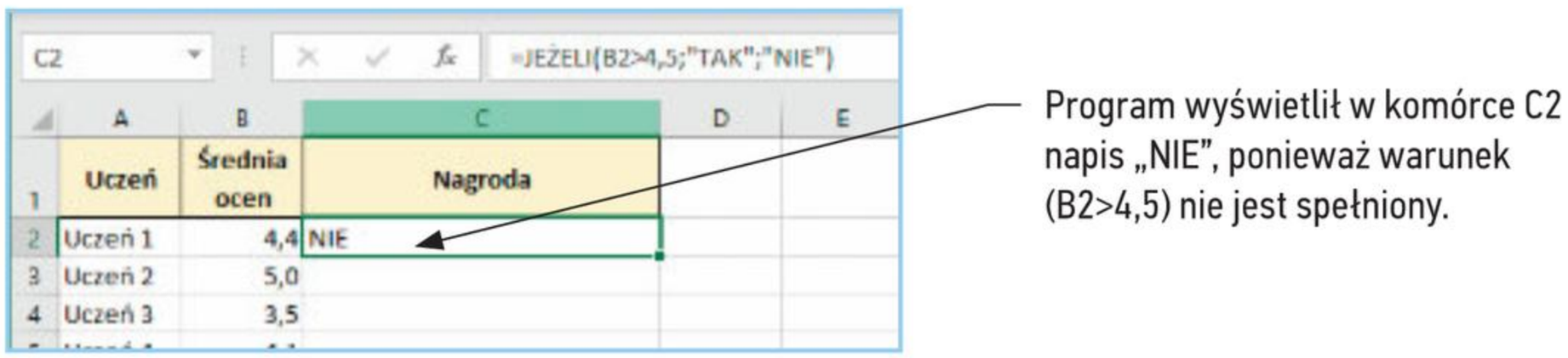
Drugim i trzecim argumentem funkcji JEŻELI może być: napis (ujęty w górne cudzysłowy ""), odwołanie do komórki, wartość liczbowa, wyrażenie zawierające operatory lub funkcje.

Przykład	Opis działania
=JEŻELI(B2>0;"dodatnia";"niedodatnia")	jeśli wartość w komórce B2 jest większa od zera, program wyświetli napis „dodatnia", w przeciwnym razie – „niedodatnia"
=JEŻELI(C5<>0;A5/C5;"nie dzielimy przez zero")	jeśli wartość w komórce C5 jest różna od zera, program wyświetli wynik dzielenia liczby z komórki A5 przez liczbę z komórki C5, w przeciwnym razie – wyświetli napis „nie dzielimy przez zero"

Tabela 3. Przykłady użycia funkcji JEŻELI w formułach



Rys. 6a. Stosowanie funkcji JEŻELI (Microsoft Excel) – ćwiczenie 2.



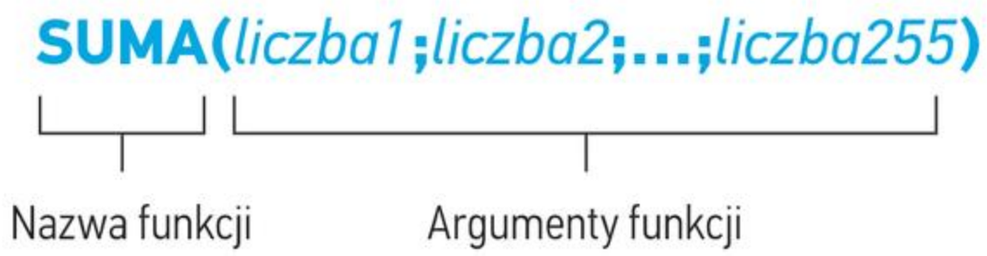
Rys. 6b. Wynik działania funkcji JEŻELI (Microsoft Excel) – ćwiczenie 2.

Ćwiczenie 2. Sprawdzamy działanie funkcji JEŻELI

1. W pliku, który otrzymasz od nauczyciela, znajdują się średnie ocen uczniów. Uczeń może otrzymać nagrodę, jeśli średnia jego ocen jest większa od 4,5; w przeciwnym razie – nie może otrzymać nagrody.
2. W zależności od wartości w kolumnie B program powinien wyświetlać w kolumnie C informację „TAK" lub „NIE".
3. Umieść odpowiednią formułę w komórce C2 i skopiuj ją do pozostałych komórek w tej kolumnie.
4. Zapisz plik pod nazwą *Nagrody*.
5. Sprawdź postać formuły w komórce C10. Wyjaśnij, dlaczego w funkcji JEŻELI warunek ma postać B10>4,5.

Funkcja SUMA

Funkcja SUMA należy do funkcji matematycznych i sumuje liczby podane jako argumenty, którymi zwykle są: odwołanie do zakresu komórek, odwołania do adresów komórek lub konkretne liczby.

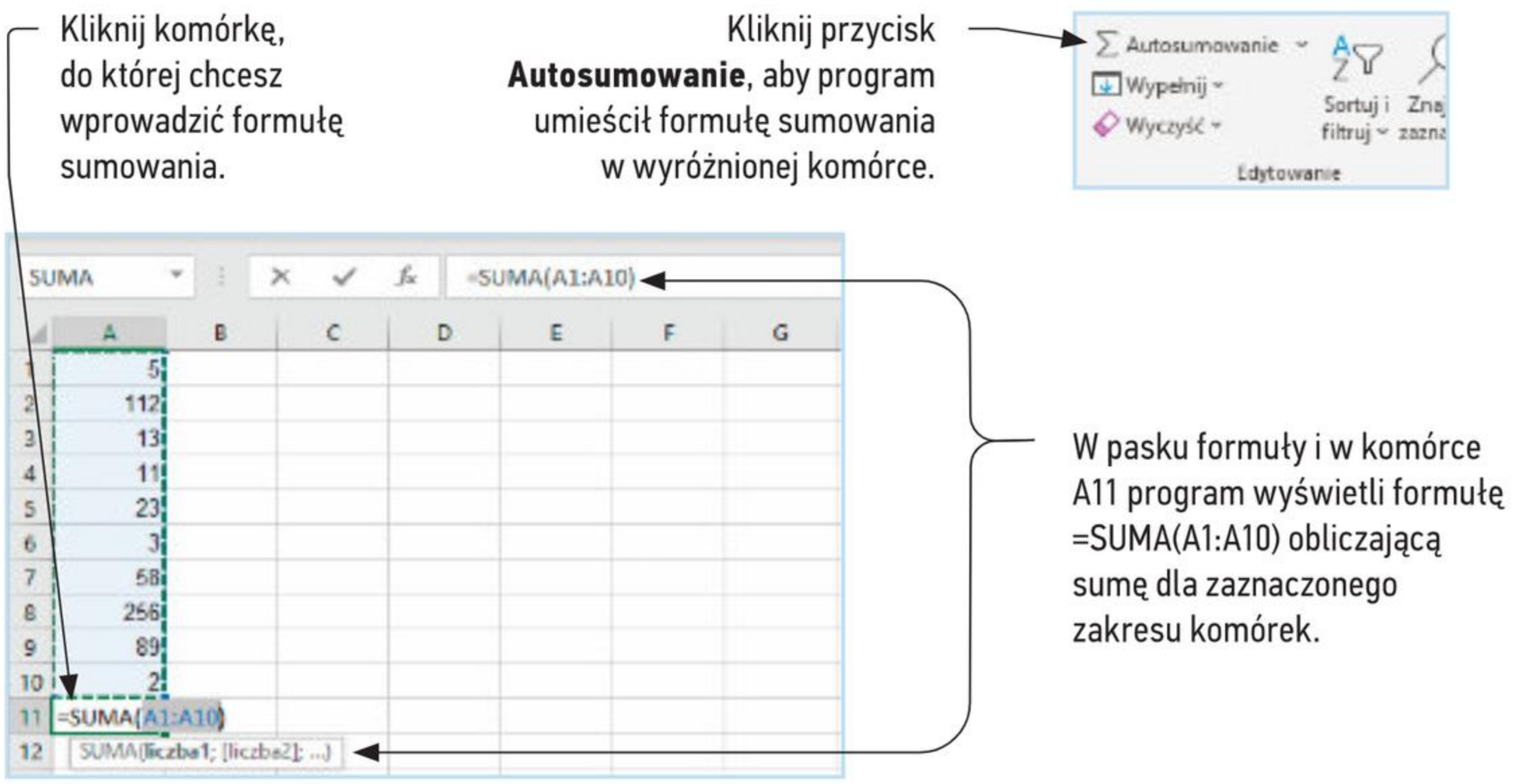


Rys. 7. Ogólna postać funkcji SUMA

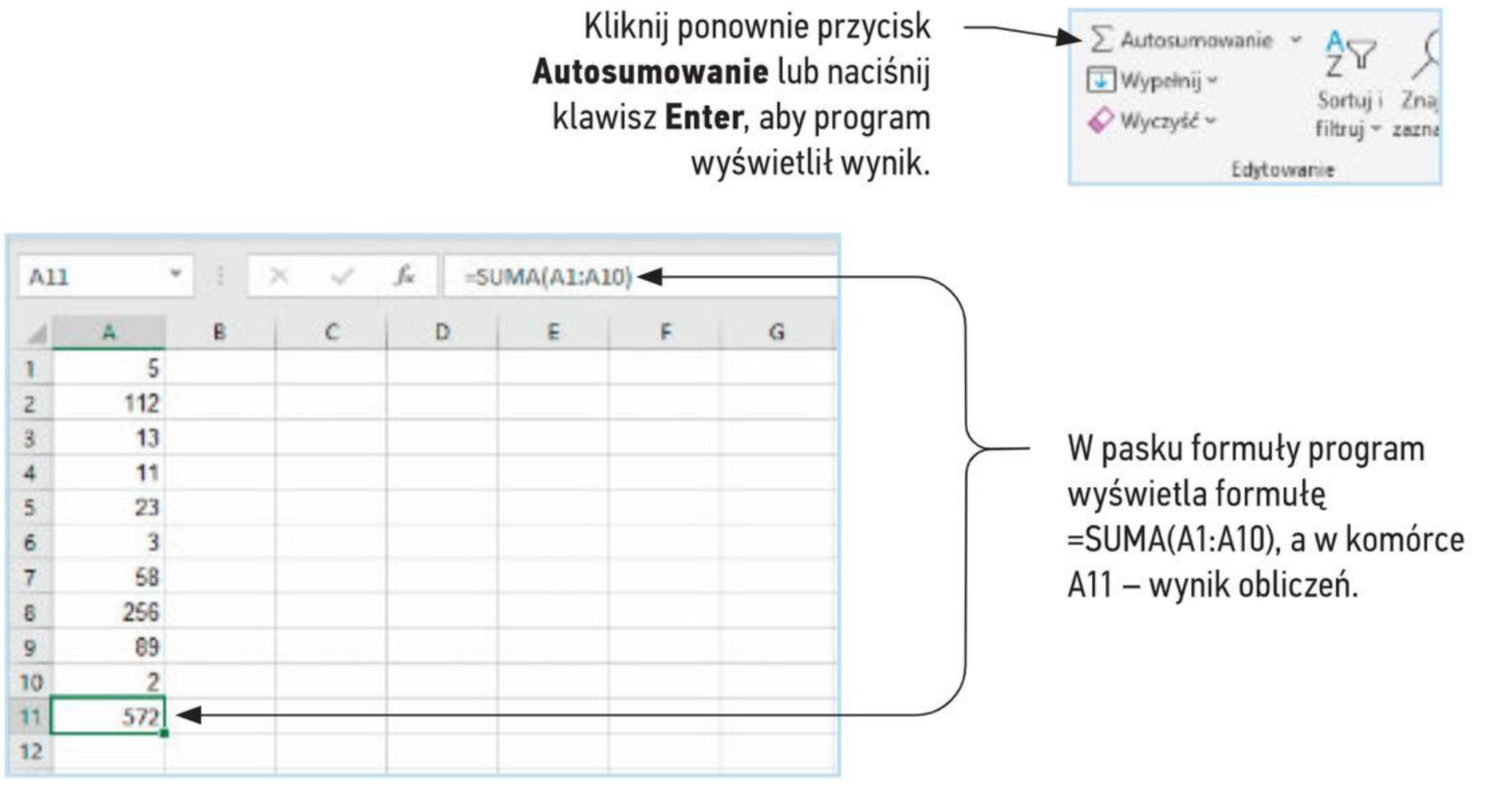
Przykład	Opis działania
=SUMA(B1:B10)	suma liczb z komórek od B1 do B10
=SUMA(A5;B7;E10)	suma liczb z komórek A5, B7 i E10
=SUMA(34;789)	suma liczb 34 i 789

Tabela 4. Przykłady użycia funkcji SUMA w formułach

Funkcja SUMA jest jedną z częściej używanych funkcji arkusza, dlatego na pasku narzędzi umieszczono reprezentujący ją przycisk. Aby zsumować dane w komórkach (w kolumnie lub w wierszu), wystarczy użyć przycisku **Autosumowanie** (rys. 8a).



Rys. 8a. Przykład zastosowania funkcji SUMA – ćwiczenie 3.



Rys. 8b. Przykład zastosowania funkcji SUMA – ćwiczenie 3.

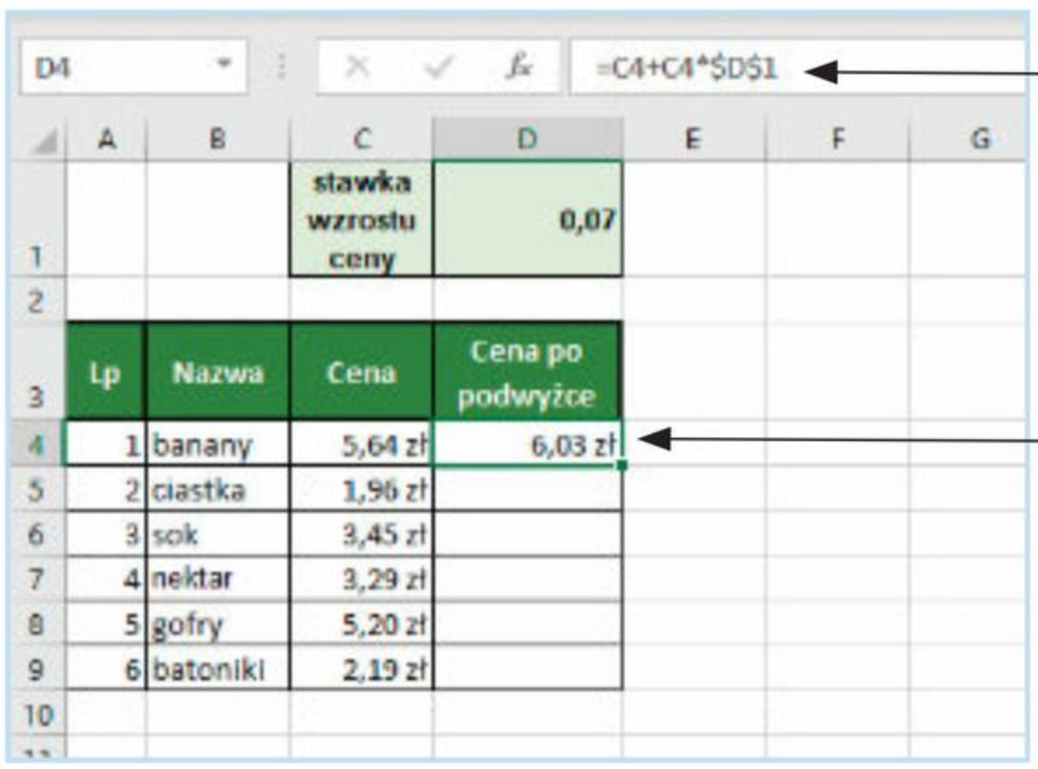
Ćwiczenie 3. Korzystamy z funkcji SUMA

1. Otwórz nowy plik arkusza kalkulacyjnego. W kolumnie A wpisz dziesięć liczb jednocyfrowych. Korzystając z funkcji SUMA, umieść w komórce A11 formułę obliczającą sumę tych liczb.
2. Dodaj w kolumnach (B, C, D i E) po dziesięć liczb. Dla każdej kolumny i każdego wiersza oblicz sumę umieszczonych tam liczb.
3. Oblicz sumę wszystkich wpisanych liczb. Wynik umieść w komórce E12. Jakie argumenty ma funkcja SUMA? Czy jest to jedyne rozwiązanie? Uzasadnij odpowiedź.
4. Zapisz plik pod nazwą *Sumowanie*.

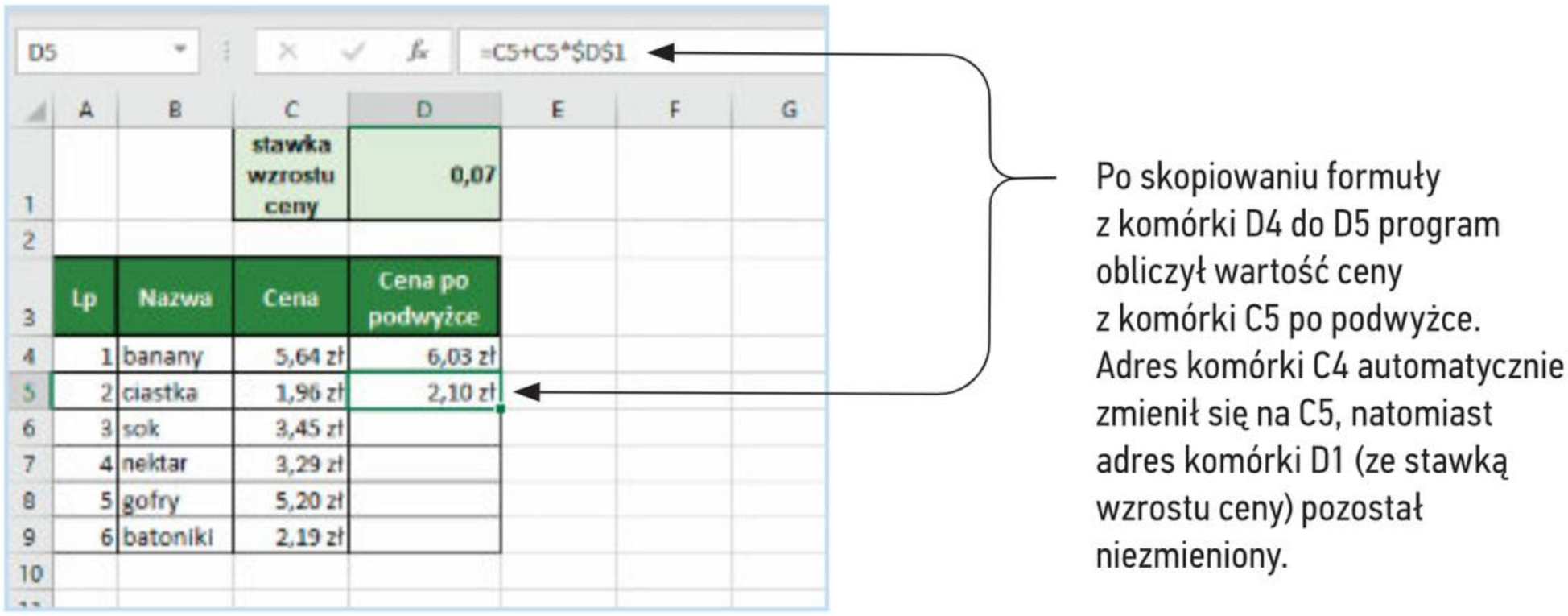
2 Adresowanie bezwzględne

Podczas kopiowania formuły zawarte w niej adresy komórek automatycznie się zmieniają – zmieniają się odwołania do wierszy i kolumn.

Aby po skopiowaniu formuły do innych komórek adres konkretnej komórki użytej w formule się nie zmieniał, stosujemy adres bezwzględny. Adres bezwzględny zawiera znak „$” przed nazwą kolumny i numerem wiersza, np. C2, F34, W12.



Rys. 9a. Stosujemy adres bezwzględny – ćwiczenie 4.



Rys. 9b. Kopiujemy formułę z adresem bezwzględnym – ćwiczenie 4.

D8 =C8+C8*D1

	A	B	C	D	E	F
1			stawka wzrostu ceny	0,07		
2						
3	Lp	Nazwa	Cena	Cena po podwyżce		
4	1	banany	5,64 zł	6,03 zł		
5	2	ciastka	1,96 zł	2,10 zł		
6	3	sok	3,45 zł	3,69 zł		
7	4	nektar	3,29 zł	3,52 zł		
8	5	gofry	5,20 zł	5,56 zł		
9	6	batoniki	2,19 zł	2,34 zł		
10						

Rys. 9c. Kopiujemy formułę z adresem bezwzględnym do pozostałych komórek – ćwiczenie 4.

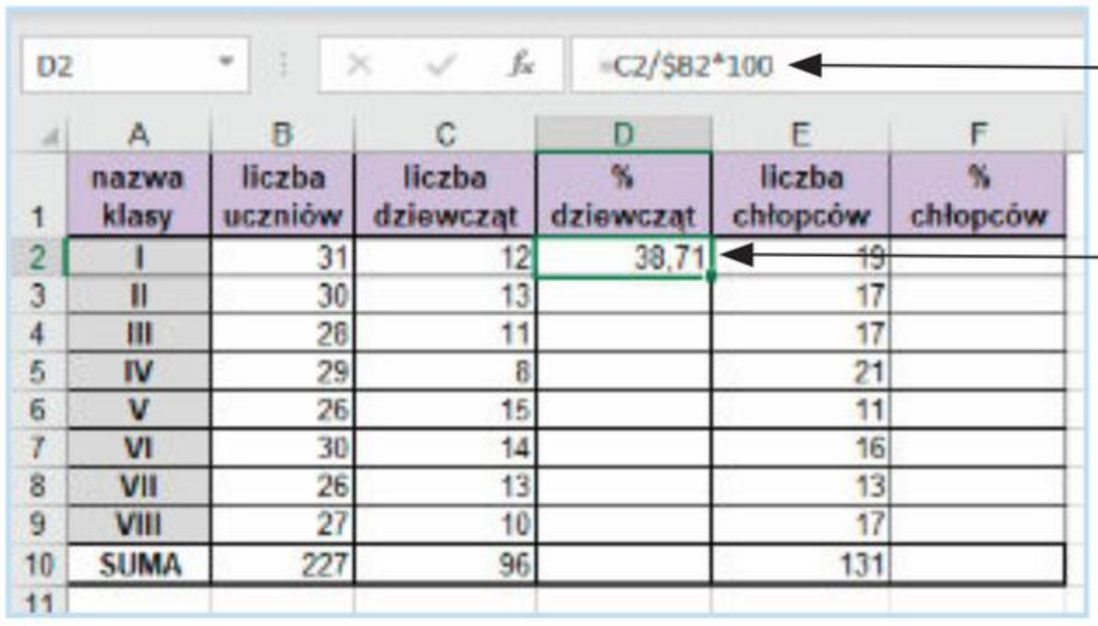
Ćwiczenie 4. Stosujemy adresowanie bezwzględne

1. W pewnym sklepie nastąpiła podwyżka cen artykułów o 7%. Oblicz ceny artykułów po podwyżce. Plik z danymi otrzymasz od nauczyciela.
2. Do komórki D4 wprowadź formułę obliczającą cenę po podwyżce. Następnie skopiuj ją do odpowiednich komórek w kolumnie D.
3. Zapisz plik pod nazwą *Podwyżka*.
4. Wyjaśnij, dlaczego w komórce D8 formuła ma postać: =C8+C8*D1.

3 Adresowanie mieszane

Przed utworzeniem formuły należy ustalić, które adresy komórek musimy w niej umieścić i jaki rodzaj adresowania zastosować, aby formuła była poprawna także po skopiowaniu i wklejeniu do innych komórek.

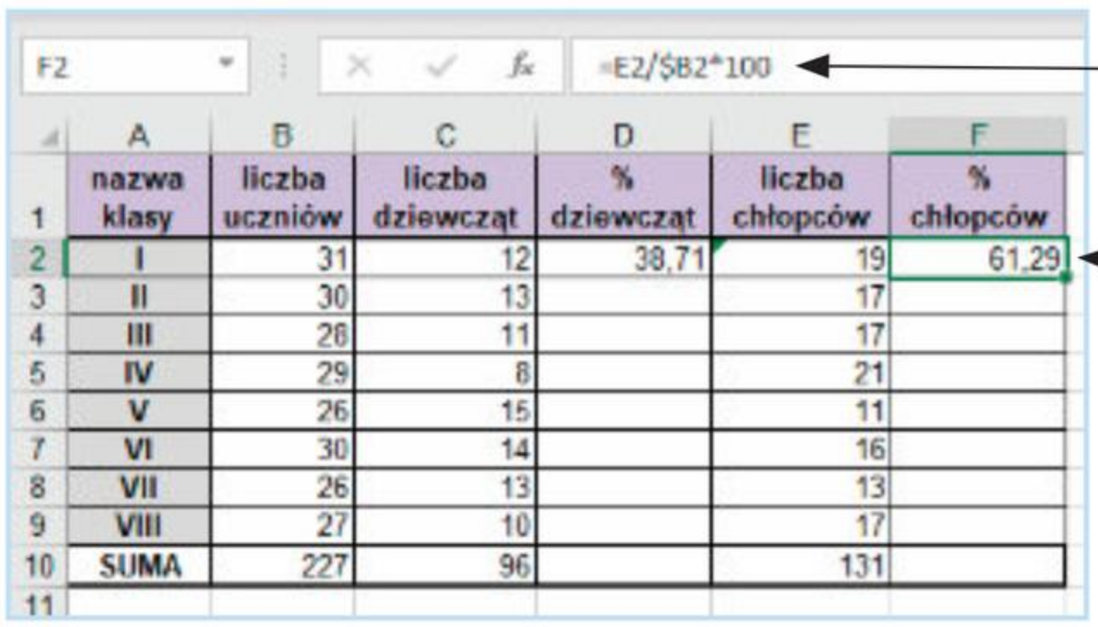
Jeśli w adresie komórki wraz ze zmianą położenia formuły zawierającej ten adres nie powinny się zmieniać tylko nazwa kolumny albo numer wiersza, należy zastosować adres mieszany.
W adresie mieszanym przed literą oznaczającą kolumnę albo liczbą oznaczającą numer wiersza należy wpisać znak „$”, np. $B2, F$18.



Formuła widoczna w pasku formuły oblicza procent dziewcząt w danej klasie. W adresie komórki, w której umieszczono liczbę uczniów klasy I, zastosowano znak „$" przed literą kolumny B, ponieważ chcemy skopiować formułę do komórki F2, aby policzyć procent chłopców.

W komórce D2 program wyświetla wynik działania formuły – procent dziewcząt w klasie I.

Rys. 10a. Stosujemy adres mieszany – ćwiczenie 5.



Po skopiowaniu formuły z komórki D2 do F2 adres komórki C2 automatycznie zmienił się na E2, natomiast w adresie komórki B2 litera kolumny pozostała niezmieniona.

W komórce F2 program wyświetla wynik działania formuły – procent chłopców w klasie I.

Rys. 10b. Kopiujemy formułę z adresem mieszanym – ćwiczenie 5.

Ćwiczenie 5. Sprawdzamy działanie adresu mieszanego

1. W pliku, którzy otrzymasz od nauczyciela, znajduje się tabela pokazana na rys. 10a.
2. Oblicz liczbę wszystkich uczniów szkoły oraz liczbę wszystkich dziewcząt i wszystkich chłopców.
3. W komórce D2 umieść formułę obliczającą, jaki jest procent dziewcząt w danej klasie. Zastosuj adres mieszany, aby formułę można było wykorzystać do obliczenia procentu chłopców w każdej klasie.
4. Skopiuj formułę do odpowiednich komórek w kolumnach D i F i ustal w tych kolumnach format wyświetlania liczb z dwoma miejscami po przecinku.
5. Zapisz plik pod nazwą *Szkoła*.
6. Wyjaśnij, dlaczego w komórce D7 formuła ma postać: =C7/$B7*100, a w komórce F7: =E7/$B7*100

Wskazówka: Nie ustalaj formatu procentowego, bo program pokaże wartości pomnożone po raz drugi przez 100.

Zapamiętaj

- Arkusz kalkulacyjny umożliwia stosowanie w formułach funkcji wbudowanych do programu, np. SUMA, ŚREDNIA, JEŻELI.
- Po nazwie funkcji w nawiasach okrągłych umieszcza się argumenty rozdzielone średnikami (jeśli jest ich więcej niż jeden).
- Adres bezwzględny umożliwia zablokowanie adresu komórki w kopiowanej formule zawierającej ten adres – nazwa kolumny i numer wiersza nie zmieniają się, niezależnie od komórki, do której skopiujemy formułę. Adres bezwzględny oznaczamy przez dodanie znaków „$” przed literą kolumny i numerem wiersza.
- Adres mieszany umożliwia zablokowanie części adresu komórki – podczas kopiowania formuły zawierającej ten adres albo zmienia się względnie nazwa kolumny, a numer wiersza pozostaje ten sam, albo zmienia się względnie numer wiersza, a nazwa kolumny pozostaje ta sama. Adres mieszany tworzymy przez dodanie znaku „$” albo przed literą kolumny, albo przed numerem wiersza.

Serwis dla ciekawskich

Aby zobaczyć wszystkie formuły w arkuszu kalkulacyjnym, wystarczy nacisnąć kombinację klawiszy **Ctrl+`** (znak ` – odwrócony apostrof, na klawiaturze znajduje się na lewo od klawisza z cyfrą 1). Aby zamknąć widok formuł, należy ponownie nacisnąć kombinację klawiszy **Ctrl+`** (Microsoft Excel, LibreOffice Calc).

OKO W OKO Z MONITOREM

Pytania i polecenia

1. Dlaczego w arkuszu kalkulacyjnym nie zaleca się wpisywania kilku liczb do jednej komórki?
2. Gdzie najlepiej umieścić liczby, na których chcemy wykonywać działania (np. sumować lub obliczyć ich średnią), by łatwiej było zastosować odpowiednie funkcje?

3. Wyjaśnij na przykładzie działanie funkcji logicznej JEŻELI.
4. Jakie są skutki wstawienia w adresie komórki znaków „$” przed literą kolumny i numerem wiersza?
5. Kiedy stosujemy adres bezwzględny? Podaj przykład.
6. Wyjaśnij na przykładzie różnicę między adresowaniem bezwzględnym a mieszanym.
7. Jakie korzyści daje stosowanie różnego rodzaju adresowania w arkuszu kalkulacyjnym?

Zadania

1. Chcesz obliczyć średnie wyniki biegów na 50, 100 i 200 metrów uzyskane przez pięciu uczniów. Plik z danymi otrzymasz od nauczyciela. W odpowiednich komórkach umieść formuły na obliczenie średnich wyników biegów na poszczególnych dystansach. Postaraj się utworzyć jak najmniejszą liczbę formuł (możesz wykorzystać kopiowanie formuł). Zapisz plik pod nazwą *Biegi*.
2. W pliku, którzy otrzymasz od nauczyciela, znajduje się tabela zawierająca fikcyjne dane dwudziestu pracowników. Oblicz sumę godzin przepracowanych przez każdego pracownika. Jeśli pracownik przepracował więcej niż 200 godzin, premia wynosi 600 zł, w przeciwnym razie – pracownik nie otrzymuje premii. Zależnie od liczby przepracowanych godzin w kolumnie I należy wyświetlać kwotę 600 zł lub 0 zł. Zapisz plik pod nazwą *Premia*.

	A	B	C	D	E	F	G	H	I
1	Lp.	Imię	Nazwisko	Liczba przepracowanych godzin				Razem liczba godzin	Premia [zł]
2				1. tydzień	2. tydzień	3. tydzień	4. tydzień		
3	1	Tomasz	Adamski	40	40	40	40	160	0 zł
4	2	Agnieszka	Grabowska	55	52	49	51	207	300 zł
5	3	Bolesław	Janik	48	40	40	43		
6	4	Krzysztof	Kaczka	53	55	49	44		
7	5	Franciszka	Kalarus	45	49	54	56		
8	6	Kacper	Kminek	40	39	41	40		
9	7	Barbara	Kogut	45	45	45	45		
10	8	Jan	Kotlarski	40	42	42	41		
11	9	Ewa	Krajewska	40	40	43	43		
12	10	Jakub	Krasicki	52	51	49	49		
13	11	Maria	Krawczyk	40	43	42	40		
14	12	Katarzyna	Leszczyńska	42	46	44	40		

Rys. 11. Fragment tabeli – zadanie 2.

3. Planujesz podróż samolotem do wybranych miast Ameryki. W pliku, którzy otrzymasz od nauczyciela, znajdują się ceny lotów w dolarach. Chcesz znać ceny lotów w złotych. Utwórz formułę przeliczającą cenę w dolarach na cenę w złotych. Znajdź w internecie aktualny kurs dolara względem złotego. Zapisz plik pod nazwą *Loty*.

4. W pliku, którzy otrzymasz od nauczyciela, znajduje się tabela z informacjami o liczbie poszczególnych ocen w klasach IV–VIII z wszystkich przedmiotów. Do tabeli dodaj formuły obliczające, ile poszczególnych ocen przypada na jednego ucznia w każdej z klas. Uwzględnij podział na oceny celujące (6), bardzo dobre (5), dobre (4), dostateczne (3), dopuszczające (2) i niedostateczne (1). Obliczenia umieść w kolejnych kolumnach. Dodaj odpowiednie nagłówki. Jaki format liczb (ile miejsc po przecinku) jest najbardziej odpowiedni dla otrzymanych wyników? Zapisz plik pod nazwą *Oceny*.

	A	B	C	D	E	F	G	H
1			Oceny z wszystkich przedmiotów na koniec roku szkolnego					
2	**klasa**	**liczba uczniów**	**liczba szóstek**	**liczba piątek**	**liczba czwórek**	**liczba trójek**	**liczba dwójek**	**liczba jedynek**
3	**IVA**	30	16	55	98	113	70	8
4	**IVB**	29	21	40	99	114	65	9
5	**IVC**	28	10	54	112	99	44	17
6	**VA**	31	15	68	97	121	51	20
7	**VB**	29	11	71	100	111	44	11
8	**VC**	29	6	69	109	113	38	13
9	**VIA**	27	9	60	89	131	27	8
10	**VIB**	26	13	79	110	88	18	4
11	**VIC**	28	5	70	130	101	23	7
12	**VIIA**	29	12	67	101	120	39	9
13	**VIIB**	25	14	72	99	90	14	11
14	**VIIC**	26	8	85	105	96	8	10
15	**VIIIA**	27	10	76	113	98	15	12
16	**VIIIB**	29	11	75	115	119	20	8
17	**VIIIC**	30	17	100	153	68	16	6
18	**razem**							

Rys. 12. Fragment tabeli – zadanie 4.

DLA ZAINTERESOWANYCH

1. Zaprojektuj tabelę symulacji średniej twoich ocen uzyskanych ze wszystkich przedmiotów na zakończenie pierwszego półrocza (lub na koniec roku szkolnego) tak, aby po zmianie oceny z dowolnego przedmiotu można było ponownie obliczyć średnią ocen. Zapisz plik pod nazwą *Oceny przewidywane*.

2. Jesteś osobą zajmującą się obliczaniem pensji pracowników. Dla celów sprawozdawczych musisz przygotować odpowiednie pismo, do którego dołączysz tabelę z zarobkami pracowników. Projektując tabelę, uwzględnij: dane osobowe, stawkę godzinową, liczbę przepracowanych godzin oraz premię dla tych pracowników, którzy przepracowali określoną liczbę godzin. Na koniec sformatuj tabelę, dodając m.in. obramowania i cieniowania. Zapisz plik pod nazwą *Płace*.

1.3. Przedstawianie danych na wykresie

Warto powtórzyć

1. Do czego służy arkusz kalkulacyjny?
2. Jak zbudowany jest dokument arkusza kalkulacyjnego?
3. Czym jest zakres komórek? Podaj przykład.
4. Jakie dane można umieszczać w komórkach arkusza? Podaj przykłady.
5. Jakie poznaliśmy sposoby adresowania w arkuszu kalkulacyjnym?

Arkusz kalkulacyjny pozwala na tworzenie wykresów, co znacznie ułatwia analizę i interpretację danych. Wykresy możemy znaleźć w wielu szkolnych podręcznikach, np. do geografii, biologii, matematyki czy fizyki. W tym temacie:

1. dowiesz się, jak utworzyć wykres w arkuszu kalkulacyjnym,
2. dowiesz się, jak przedstawić na wykresie więcej niż jedną serię danych,
3. poznasz różne typy wykresów.

1 Tworzenie wykresu w arkuszu kalkulacyjnym

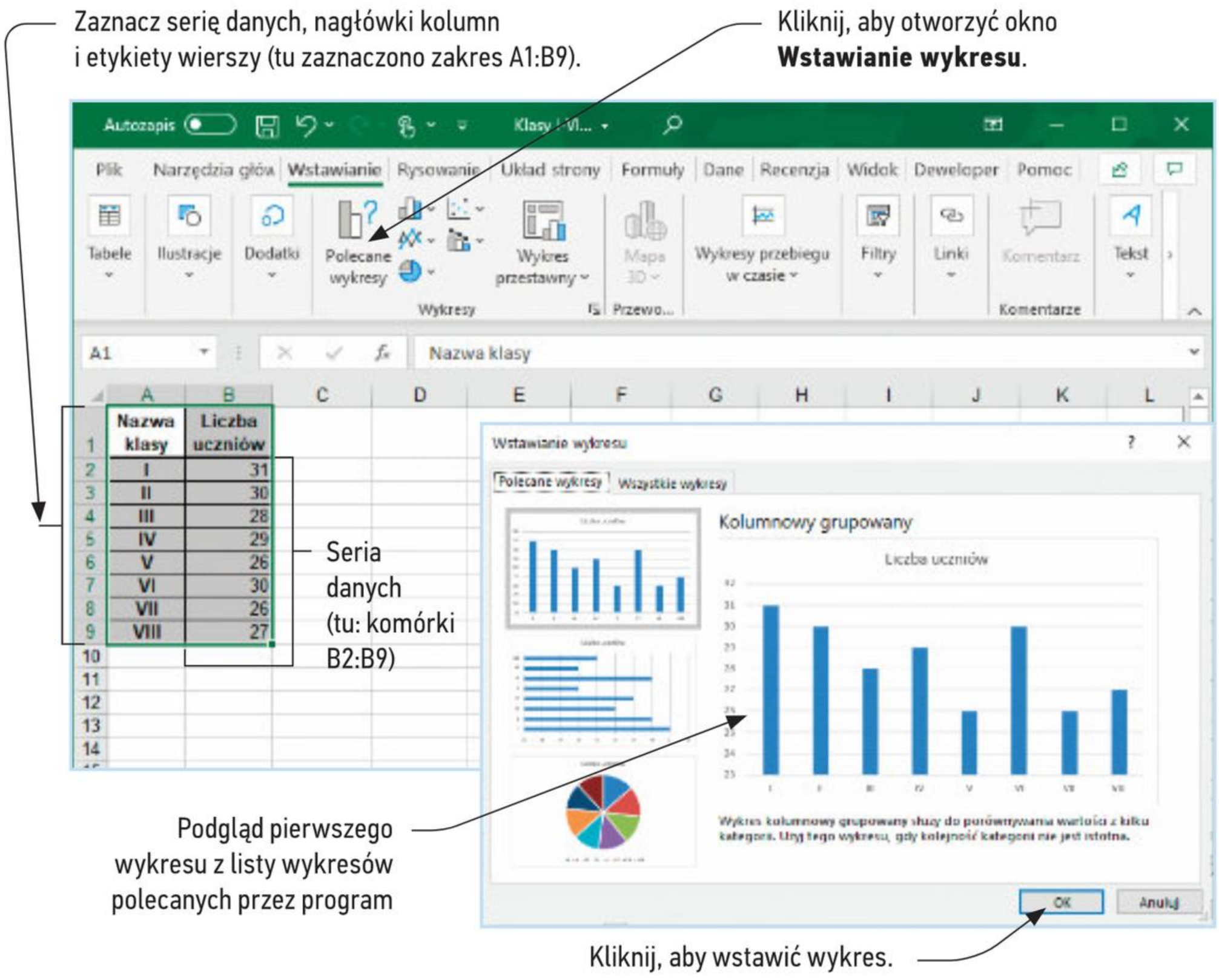
Wykresy możemy tworzyć dla jednej lub dla kilku **serii danych**. Dane prezentujemy na wykresach różnych typów. Typ wykresu określa sposób prezentacji danych, np. za pomocą słupków, linii, wycinków koła.

Aby rozpocząć tworzenie wykresu dla jednej serii danych, należy:

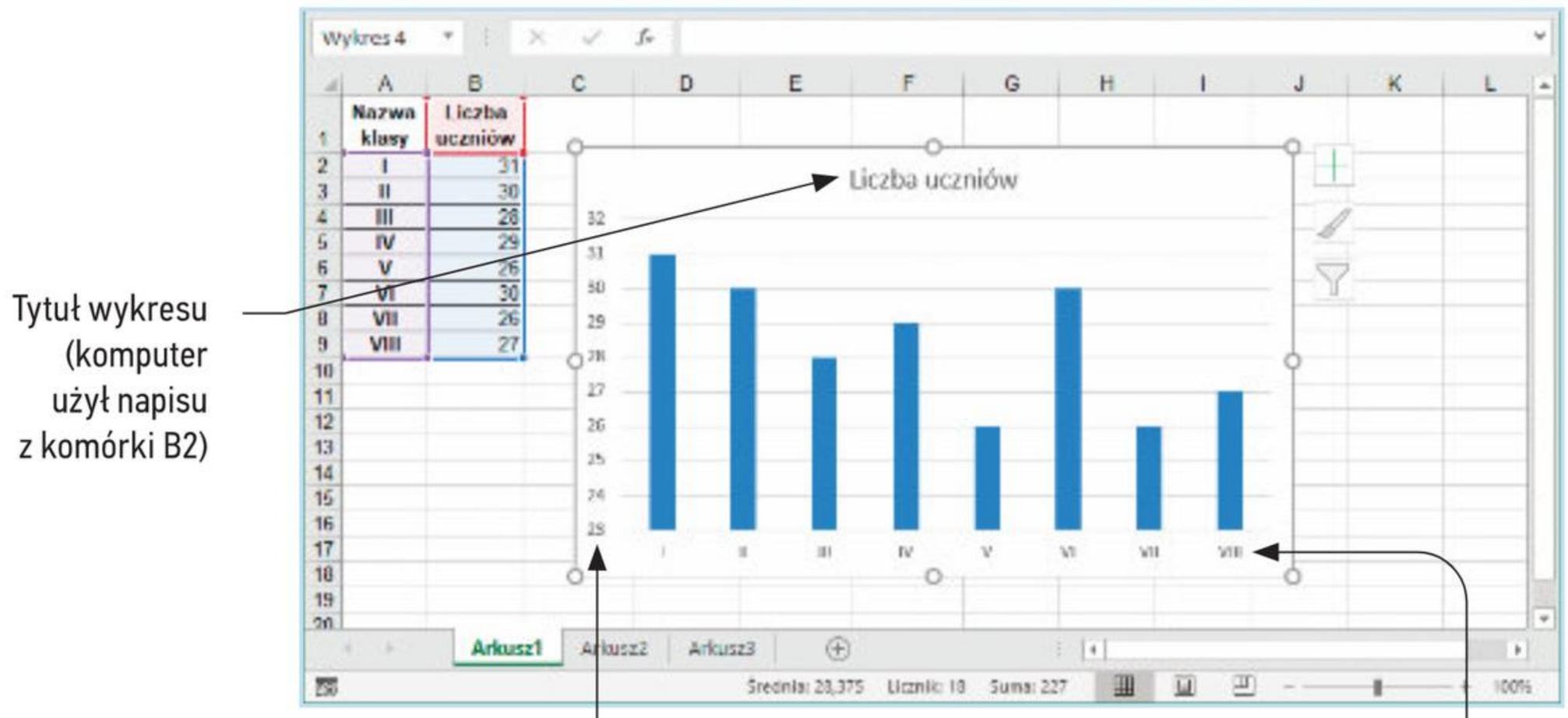
1. zaznaczyć dane, które mają się znaleźć na wykresie, oraz nagłówki kolumn lub etykiety wierszy (dane powinny znajdować się w jednej kolumnie lub w jednym wierszu),
2. wybrać opcję wstawiania wykresu na karcie **Wstawianie** (Microsoft Excel) lub w menu **Wstaw** (LibreOffice Calc),
3. wybrać typ wykresu.

Seria danych
Zakres komórek arkusza kalkulacyjnego obejmujący jeden wiersz lub jedną kolumnę, których wartości chcemy umieścić na wykresie.

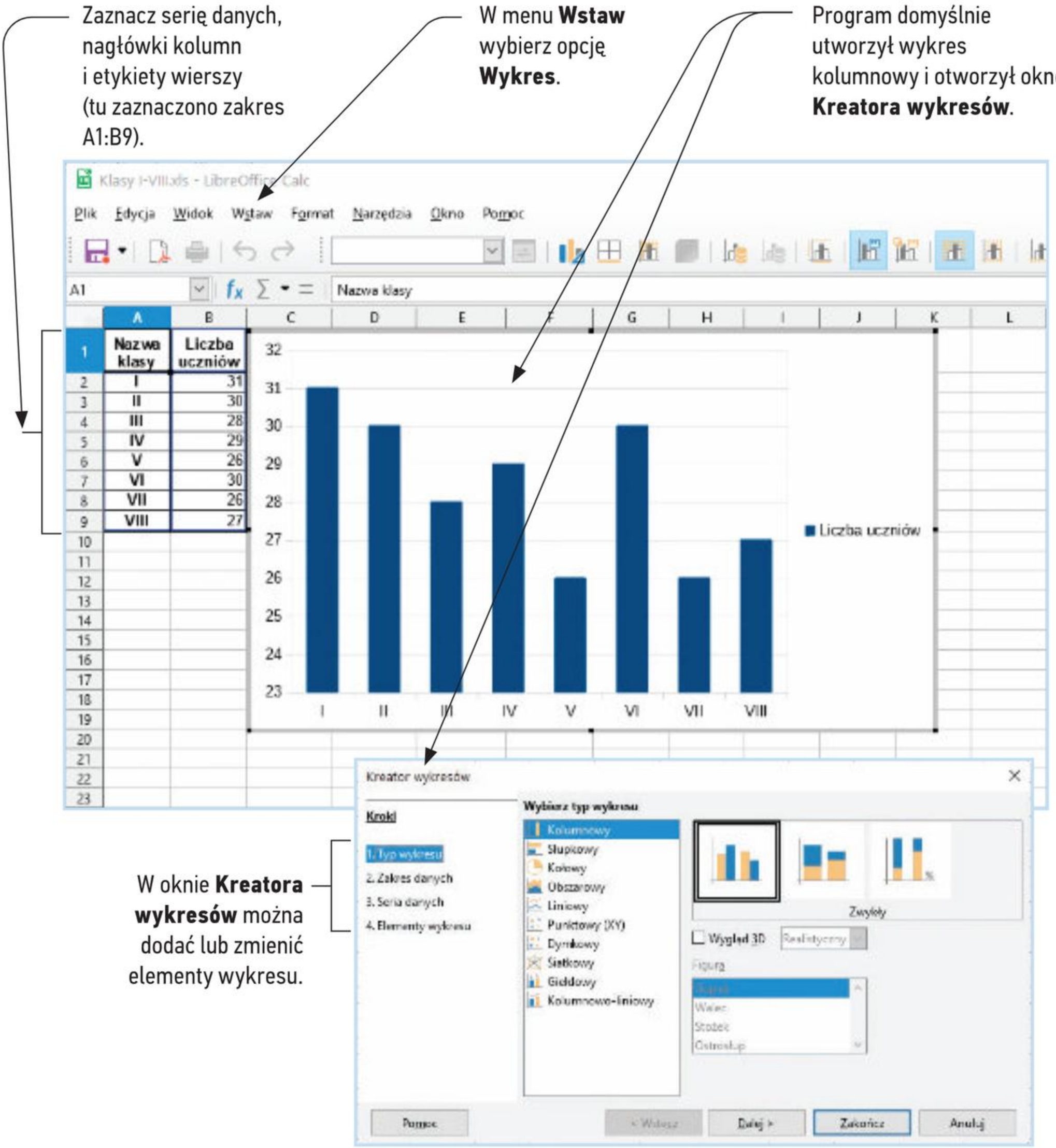
Jeśli zaznaczymy dane łącznie z nagłówkami kolumn i etykietami wierszy, wtedy nagłówki kolumn i etykiety wierszy mogą posłużyć do opisania osi, legendy oraz jako tytuł. Na przykład jeśli seria danych znajduje się w kolumnie, jej nagłówek posłuży do opisania tytułu i serii danych w legendzie, a etykiety wierszy posłużą do opisania osi poziomej.



Rys. 1a. Tworzenie wykresu kolumnowego dla jednej serii danych (Microsoft Excel) – ćwiczenie 1.



Rys. 1b. Przykład wykresu kolumnowego dla jednej serii danych (Microsoft Excel) – ćwiczenie 1.



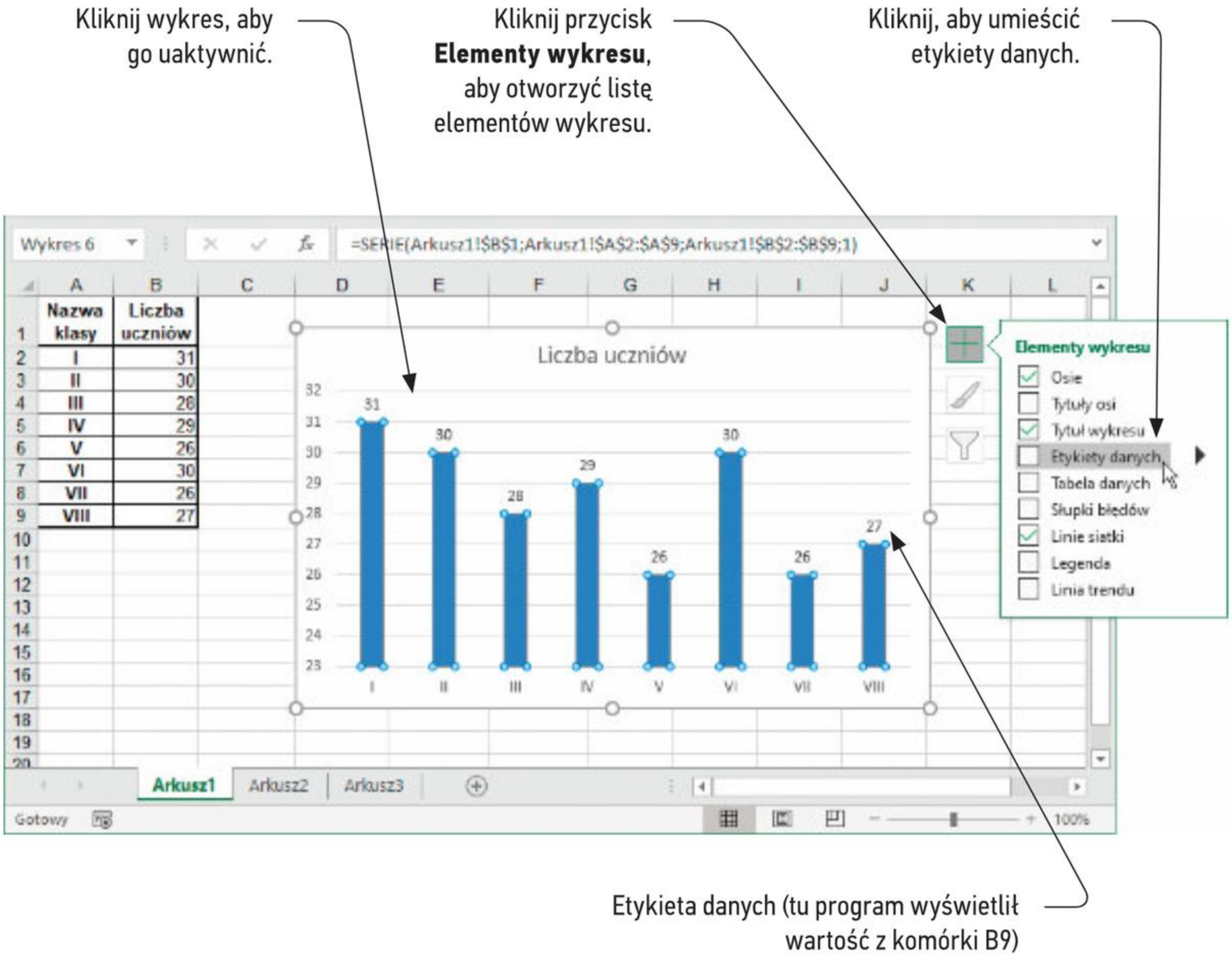
Rys. 2. Tworzenie wykresu kolumnowego dla jednej serii danych (LibreOffice Calc) – ćwiczenie 1.

Ćwiczenie 1. Tworzymy wykres dla jednej serii danych

1. W pliku, który otrzymasz od nauczyciela, umieszczono informacje o liczbie uczniów pewnej szkoły.
2. Przygotuj wykres kolumnowy, obrazujący liczbę uczniów w poszczególnych klasach.
3. Zapisz plik pod nazwą *Klasy I-VIII*.
4. Sprawdź, czy po zmianie danych w tabeli arkusza kalkulacyjnego trzeba ponownie tworzyć wykres. Zmień dane w tabeli: w klasie IV powinno być 27 uczniów, w V – 24, w VII – 32. Co można zauważyć na wykresie? Nie zapisuj zmian.

Zmiana danych w tabeli powoduje automatyczną aktualizację wykresu utworzonego na podstawie tej tabeli.

Wstawiony wykres można modyfikować, np. dodać etykiety danych (nazwy serii, nazwy kategorii lub wartości danych), zmienić zakres danych, poprawić tytuł wykresu czy opis legendy. Można również formatować poszczególne elementy wykresu. W tym celu należy najpierw uaktywnić element, w którym umieszczono daną informację.



Rys. 3. Dodawanie etykiet danych do wykresu (Microsoft Excel) – ćwiczenie 2.

Ćwiczenie 2. Modyfikujemy wykres

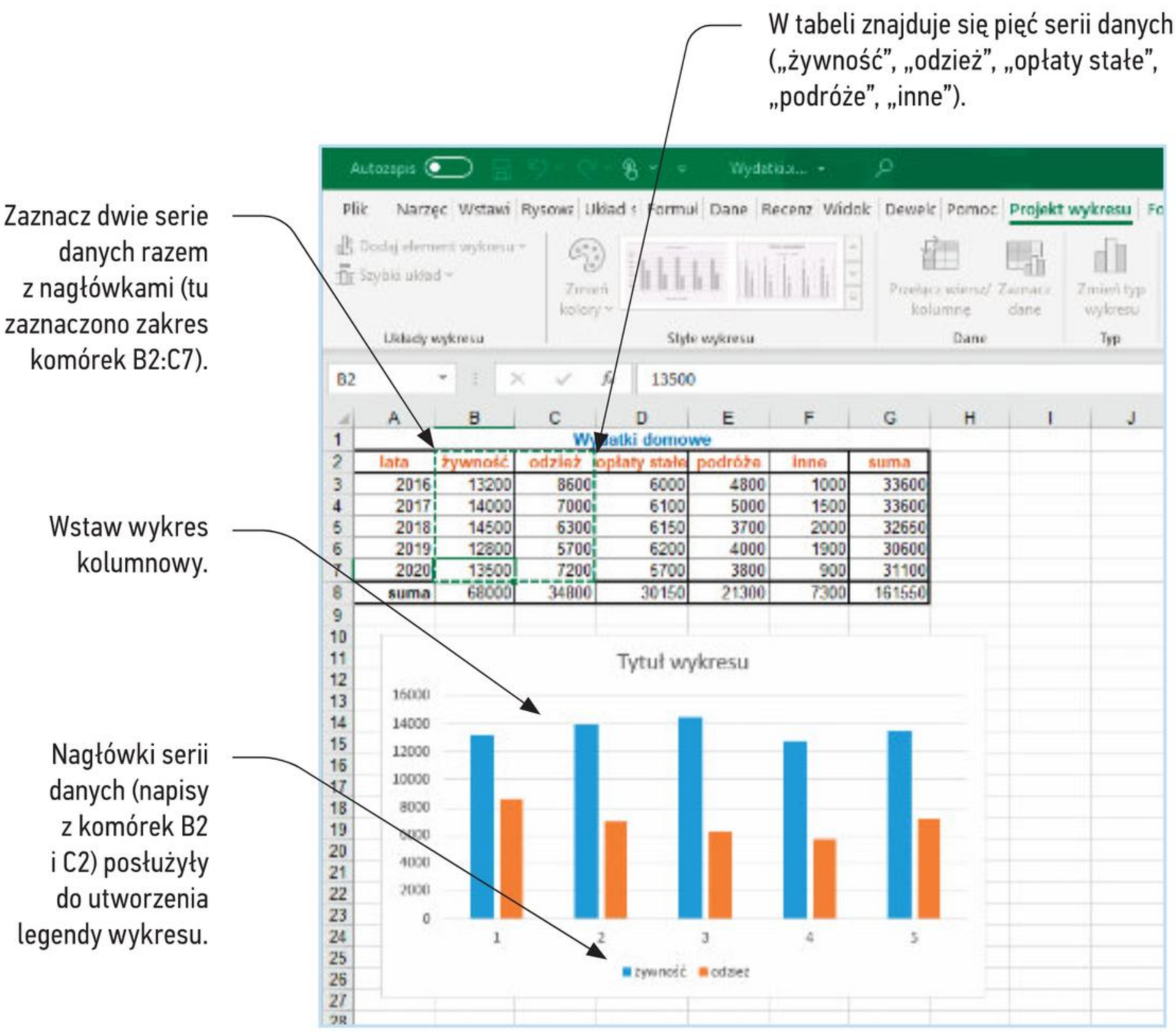
1. Dodaj etykiety danych do wykresu utworzonego w ćwiczeniu 1. Zmień kolor serii danych na inny – według uznania. Sformatuj tytuł – zastosuj pogrubienie tekstu i zmień kolor czcionki.
2. Zapisz plik pod nazwą *Etykiety danych*.

2 Tworzenie wykresu dla kilku serii danych

Aby utworzyć wykres dla kilku serii danych, postępujmy podobnie jak w przypadku wykresu dla jednej serii. Należy jednak zaznaczyć wszystkie serie danych, których wartości chcemy na nim umieścić. Jeśli zaznaczenie wszystkich serii nie jest możliwe, zaznaczamy tylko pierwszą.

Jeśli chcemy zaznaczyć serie danych zapisanych w kolumnach lub wierszach, które ze sobą nie sąsiadują, to po zaznaczeniu pierwszej serii należy zaznaczyć kolejną serię, przytrzymując wciśnięty klawisz **Ctrl**. Do utworzonego wykresu można dodać nową serię danych, wybierając opcję wykresu **Zaznacz dane** i zmieniając zakres danych w oknie **Wybieranie źródła danych**. W tym samym oknie można nadać nazwy seriom danych, które pojawią się w legendzie. Każdą serię danych program zaznacza na wykresie innym kolorem.

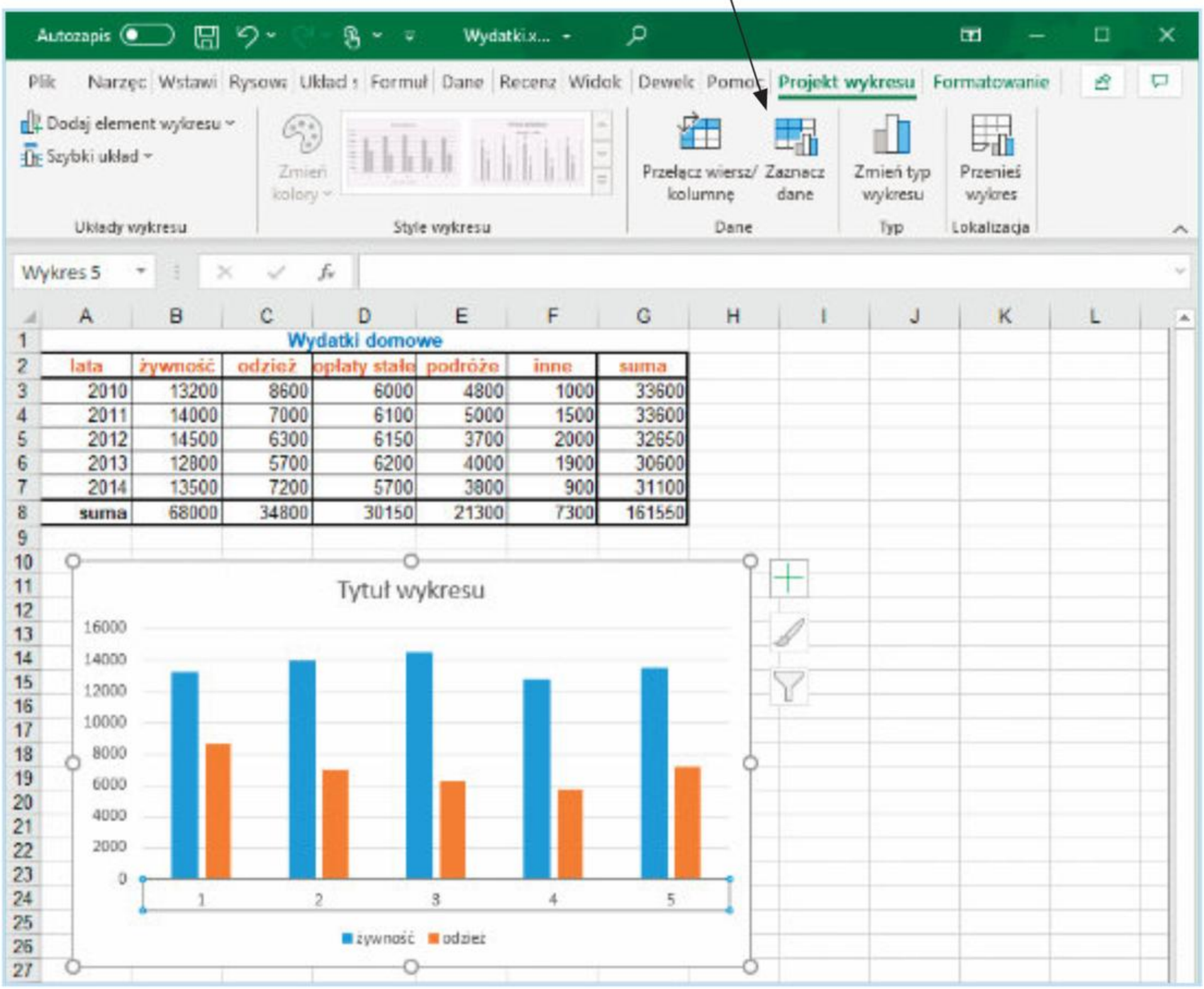
Zaznaczony zakres danych można skopiować do **Schowka** (**Ctrl**+**C**), uaktywnić wykres i wkleić w jego obszarze zaznaczoną serię danych (**Ctrl**+**V**).
Można też skorzystać z opcji **Zaznacz dane** w menu kontekstowym wykresu.

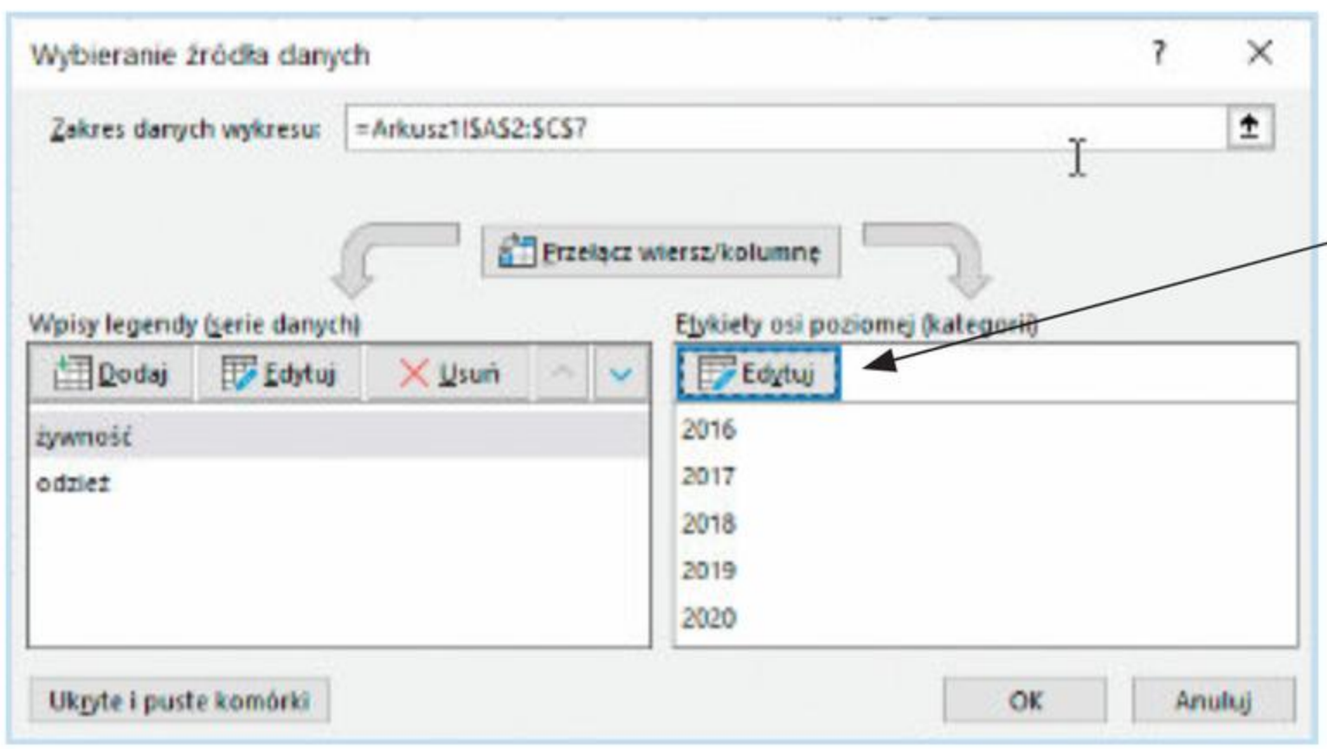


Rys. 4a. Tworzenie wykresu kolumnowego dla dwóch serii danych (Microsoft Excel) – ćwiczenie 3.

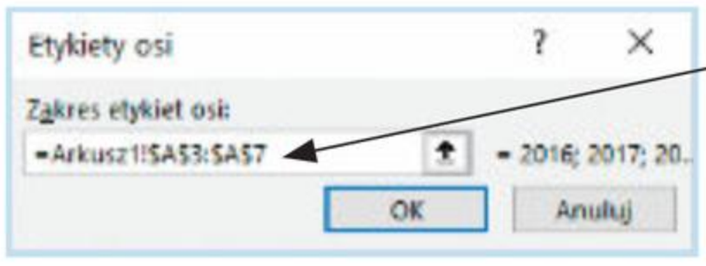
Aby dodać etykiety osi poziomej, zaznacz zakres komórek A3:A7.

Kliknij opcję **Zaznacz dane**, aby otworzyć okno **Wybieranie źródła danych**.



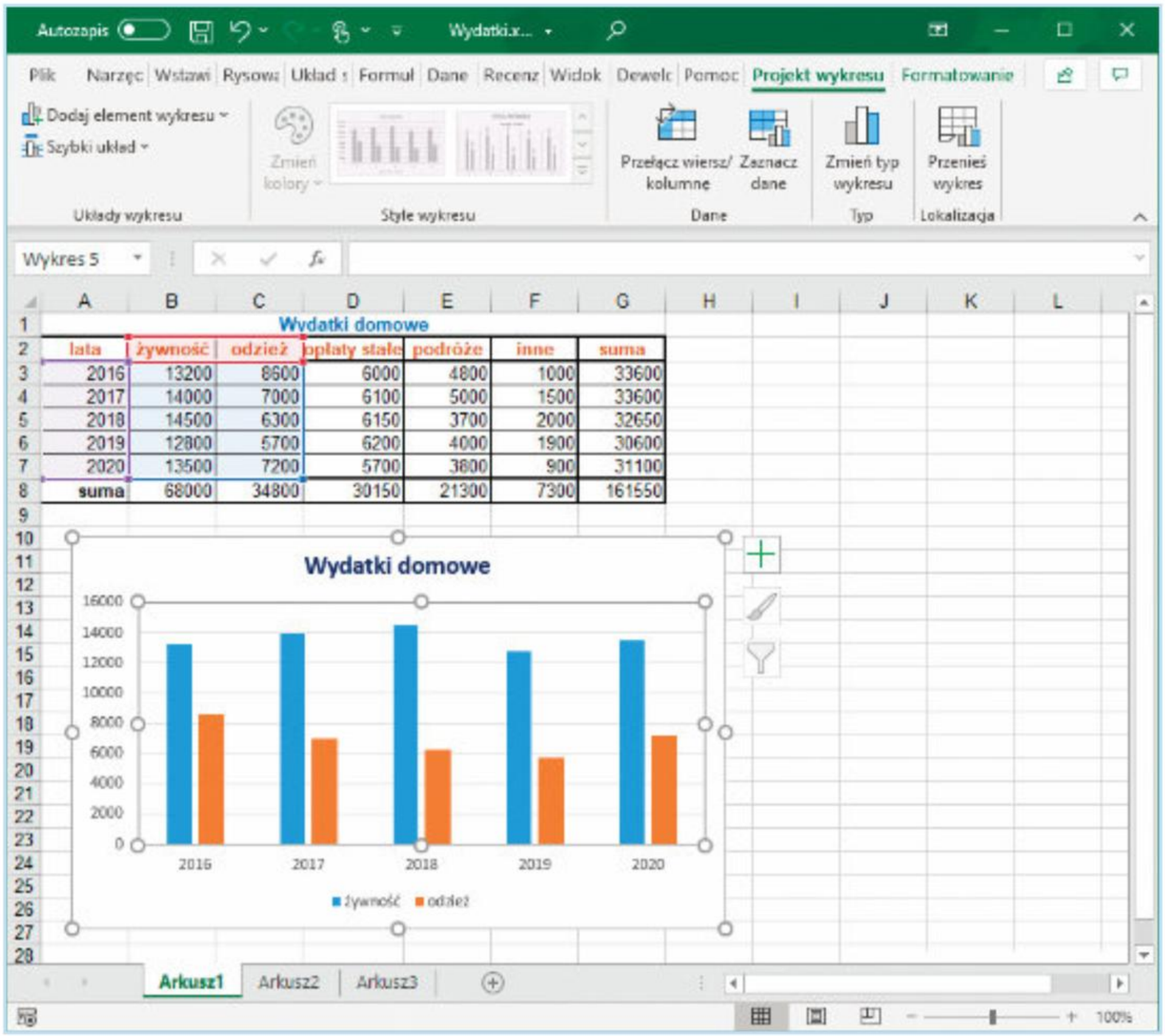


Kliknij przycisk **Edytuj**, aby zmienić etykiety osi poziomej.



Po zaznaczeniu zakresu komórek A3: A7 w tym polu pojawi się zakres etykiet osi poziomej.

Rys. 4b. Tworzenie wykresu kolumnowego dla dwóch serii danych (Microsoft Excel) – ćwiczenie 3.



Rys. 5. Przykład tabeli oraz wykresu kolumnowego dla dwóch serii danych – ćwiczenie 3.

Ćwiczenie 3. Tworzymy wykres dla dwóch serii danych

1. W pliku, który otrzymasz od nauczyciela, znajduje się tabela przedstawiająca wydatki domowe w kolejnych latach (zgodnie ze wzorem pokazanym na rysunku 5.). Zsumuj wydatki każdego rodzaju oraz wydatki w poszczególnych latach.
2. Utwórz wykres przedstawiający wydatki na żywność i odzież. Dodaj do wykresu legendę i tytuł (zgodnie ze wzorem pokazanym na rysunku 5.).
3. Zapisz plik pod nazwą *Wydatki*.

Etykiety osi poziomej można również wstawić, zaznaczając od razu dane z kolumn A, B i C, czyli zakres komórek A2:C7.

Ćwiczenie 4. Dodajemy kolejne serie danych

1. Dodaj do wykresu wykonanego w ćwiczeniu 3. pozostałe trzy serie danych. Postaraj się umieścić na wykresie od razu wszystkie serie danych łącznie z legendą. Wyjaśnij sposób postępowania.
2. Zapisz plik pod tą samą nazwą *Wydatki_całość*.

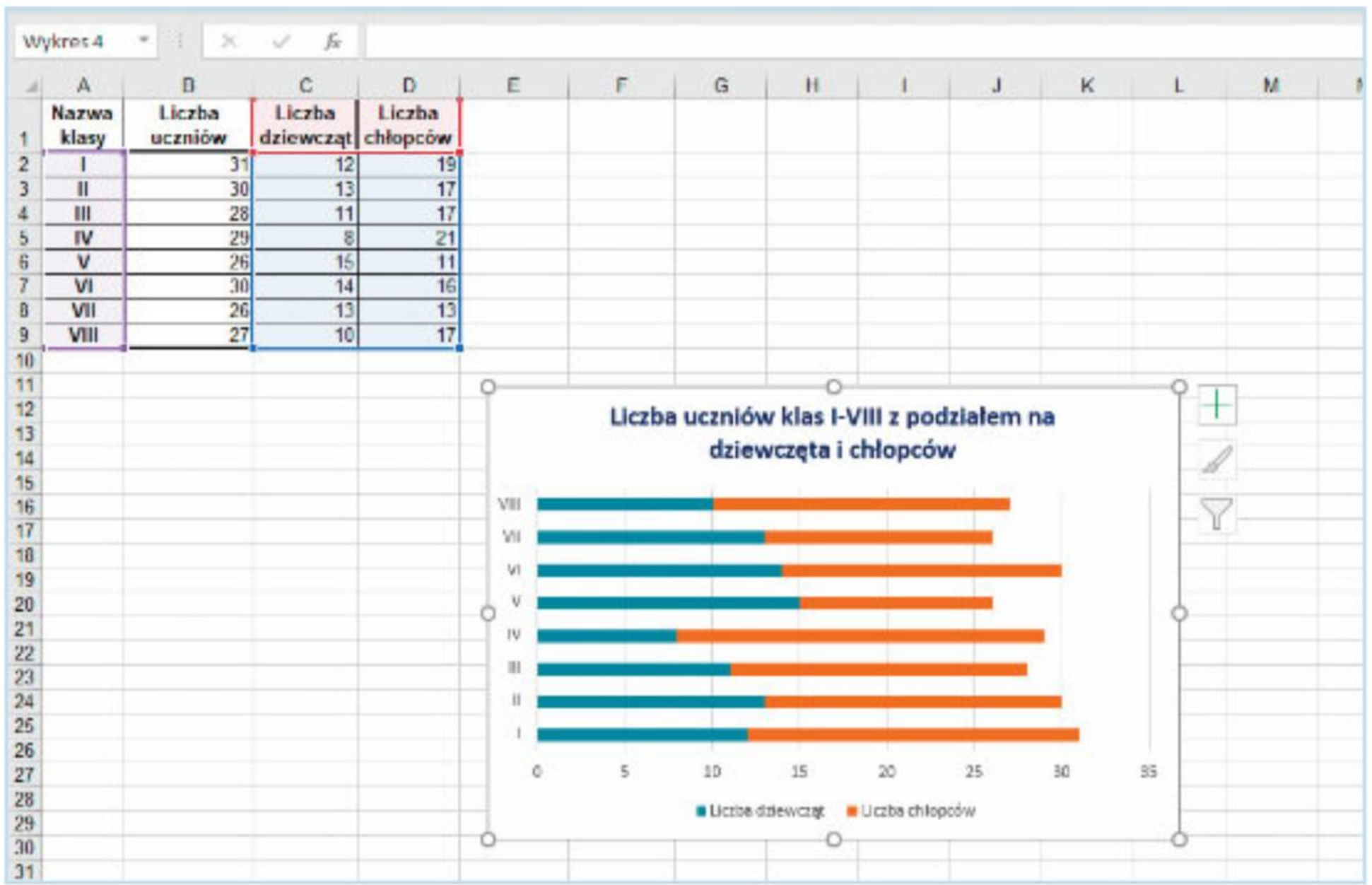
3 Przykładowe typy wykresów

Aby dane przedstawione na wykresie były czytelne, należy wybrać odpowiedni typ wykresu. Przedstawimy kilka wybranych typów wykresów.

Wykres kolumnowy i słupkowy

Wykres kolumnowy i słupkowy służą do porównywania wartości dla różnych kategorii, np. liczby ocen niedostatecznych, dopuszczających, dostatecznych, dobrych, bardzo dobrych i celujących w poszczególnych klasach, liczby urodzeń w krajach Europy w danym roku itp.

Na rysunku 6. pokazano wykres słupkowy dla dwóch serii danych porównujący liczbę dziewcząt i chłopców w każdej z klas I–VIII pewnej szkoły.



Rys. 6. Przykład wykresu słupkowego dla dwóch serii danych – ćwiczenie 5.

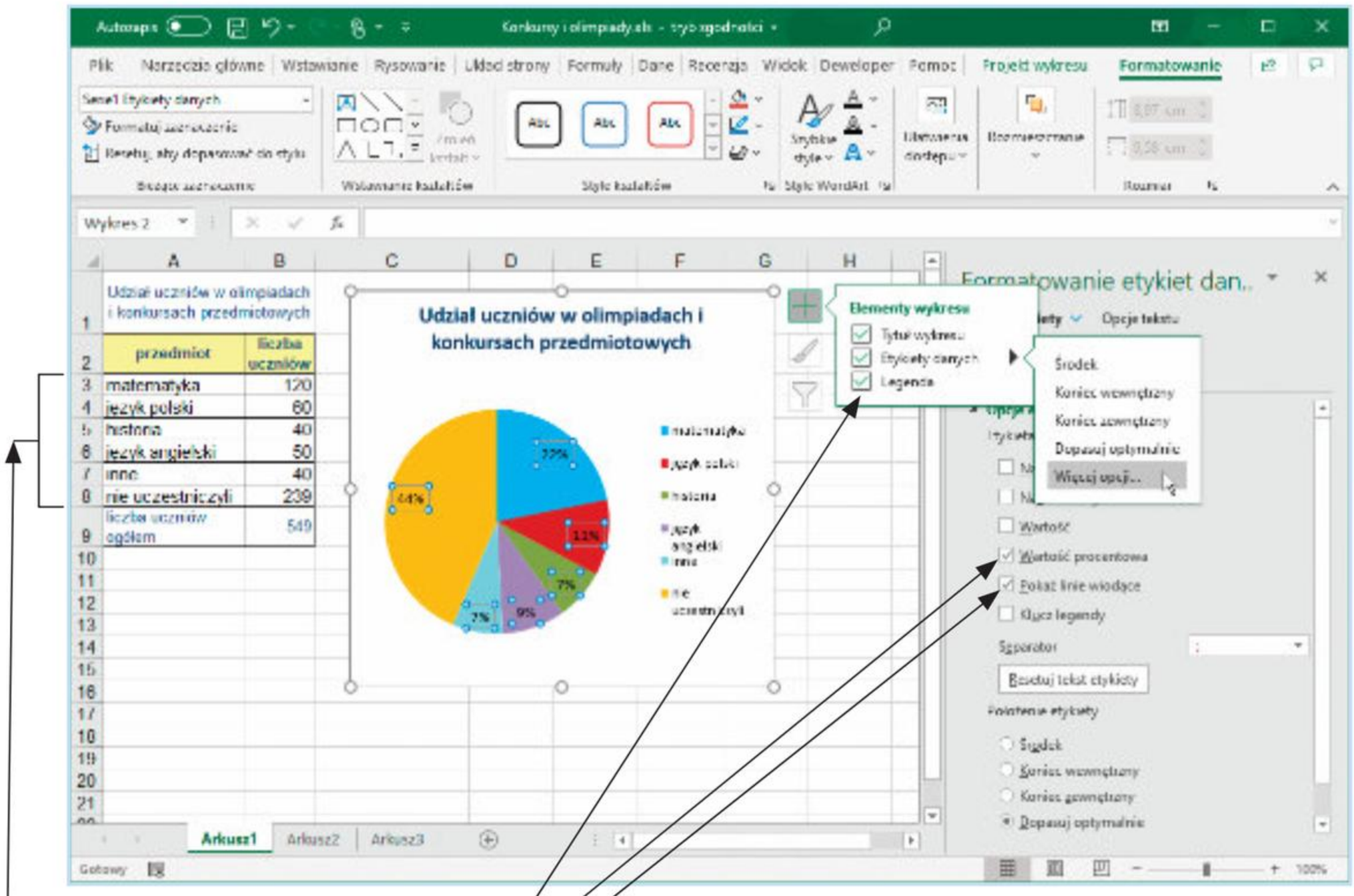
Ćwiczenie 5. Tworzymy wykres słupkowy dla dwóch serii danych

1. W pliku, który otrzymasz od nauczyciela, znajduje się tabela pokazana na rysunku 6.
2. Utwórz wykres słupkowy porównujący liczbę dziewcząt i chłopców w każdej z klas I–VIII. Dodaj do wykresu legendę i tytuł (zgodnie ze wzorem pokazanym na rysunku 6.).
3. Zapisz plik pod nazwą *Dziewczęta i chłopcy*.

Wykres kołowy

Wykresu kołowego używa się do przedstawienia proporcji, np. jaki procent uczniów szkoły uczestniczy w różnych olimpiadach przedmiotowych, jaki jest udział w rynku przedsiębiorstw produkujących wybrane marki samochodów, jaki jest procent powierzchni poszczególnych kontynentów w stosunku do powierzchni kuli ziemskiej.

Wykres kołowy wykonuje się tylko dla jednej serii danych. Pokazuje on rozmiary (czyli udziały) poszczególnych składników serii danych w porównaniu do sumy tych składników (np. jako procenty całego koła).



Zaznacz kolumny z nazwami przedmiotów i z liczbami uczniów, aby umieścić nazwy przedmiotów w legendzie.

Aby umieścić na wykresie wartości procentowe, w menu **Elementy wykresu** kliknij **Etykiety danych**, a następnie w oknie **Formatowanie etykiet danych** – polecenie **Więcej opcji**, odznacz pole **Wartość** i zaznacz **Wartość procentowa**.

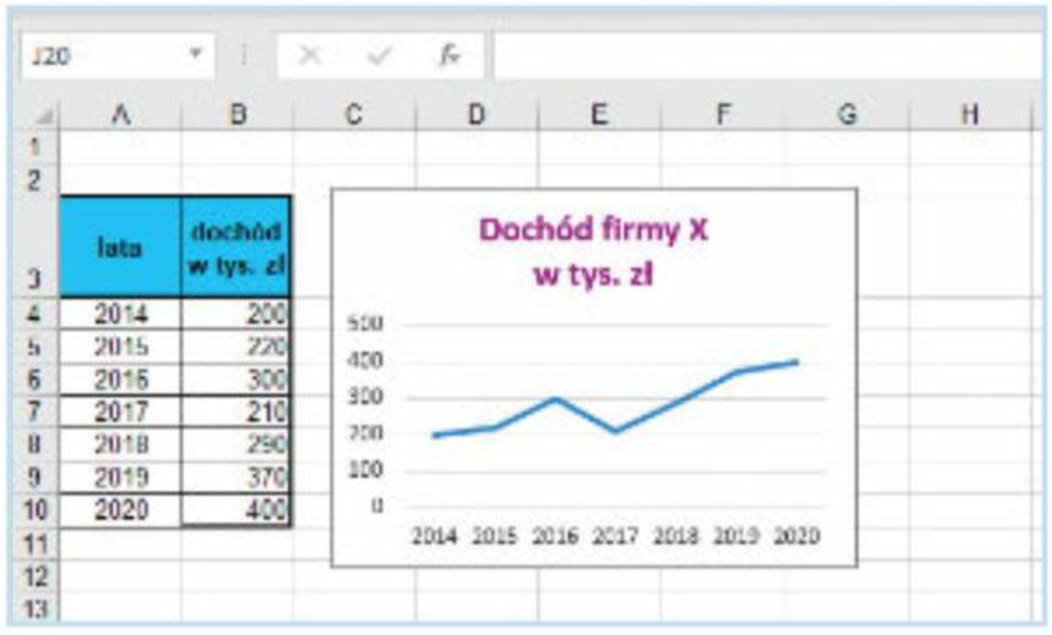
Rys. 7. Przykład wykresu kołowego – ćwiczenie 6.

Ćwiczenie 6. Tworzymy wykres kołowy

1. W pliku, który otrzymasz od nauczyciela, znajduje się tabela, zawierająca informacje o liczbie uczniów biorących udział w olimpiadach i konkursach przedmiotowych. Przedstaw dane w postaci wykresu kołowego. Umieść na wykresie etykiety danych (wartości procentowe).
2. Zapisz plik pod nazwą *Konkursy i olimpiady*.

Wykres liniowy

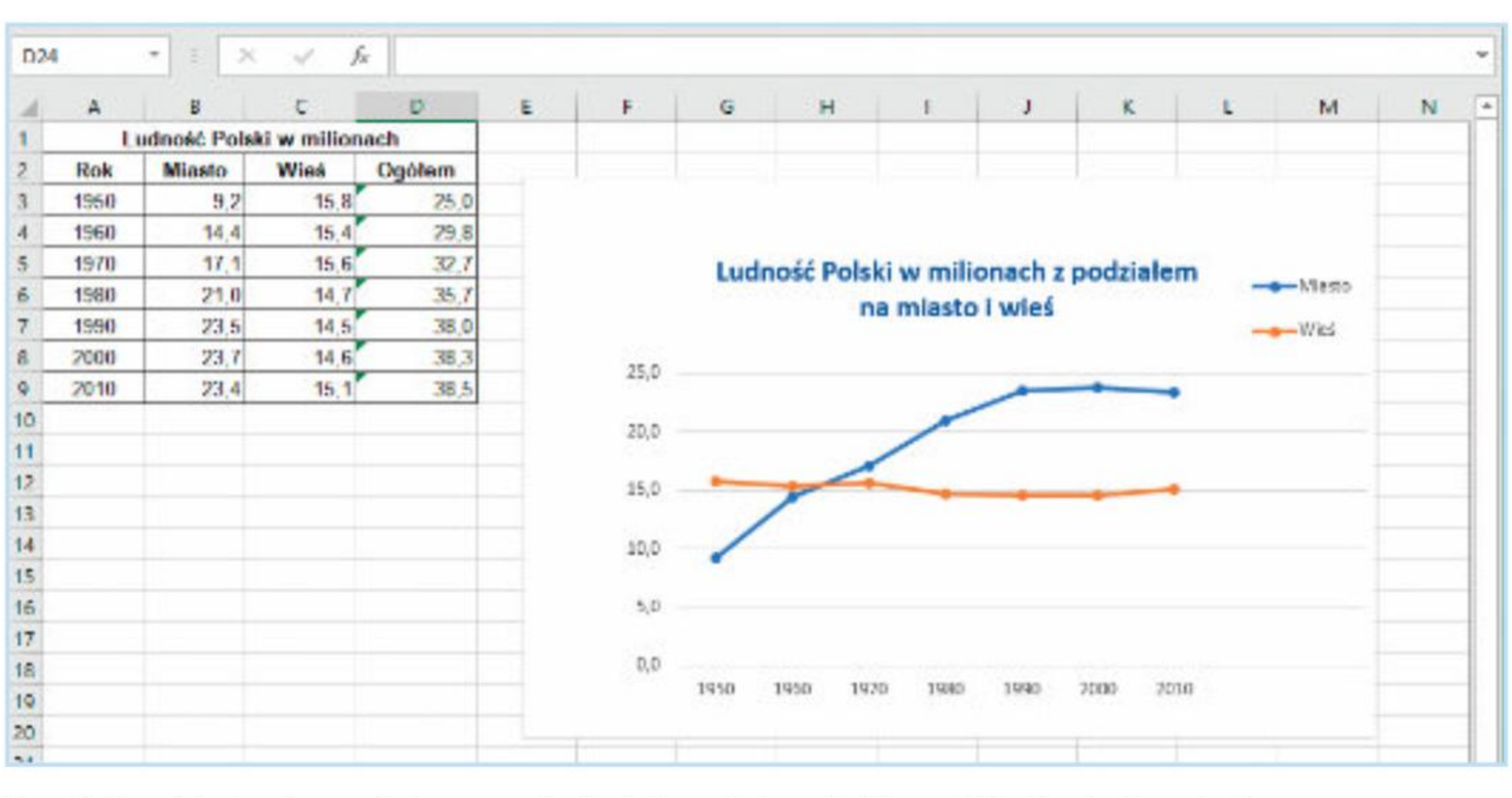
Za pomocą wykresu liniowego przedstawia się zwykle tendencje wzrostowe lub spadkowe (wzrost produkcji, spadek zatrudnienia itp.).



Rys. 8. Przykład wykresu liniowego dla jednej serii danych – ćwiczenie 7.

Ćwiczenie 7. Tworzymy wykres liniowy dla jednej serii danych

1. Przedstaw na wykresie zmiany wartości dochodów pewnej firmy w kolejnych latach. Umieść przykładowe dane w tabeli (kwoty podaj w tysiącach złotych). Dane z kolumny A mają posłużyć do opisu osi poziomej. Na rysunku 8. przedstawiono propozycję wykresu.
2. Zapisz plik pod nazwą *Dochody firmy*.



Rys. 9. Przykład wykresu liniowego dla dwóch serii danych (dane GUS-u) – ćwiczenie 8.

Ćwiczenie 8. Tworzymy wykres liniowy dla dwóch serii danych

1. Tabela na rysunku 9. przedstawia informacje o liczbie ludności Polski z podziałem na mieszkańców miast i wsi w latach 1950-2010 (co 10 lat). Utwórz podobną tabelę i wykres, ale uzupełnij je o dane z roku 2020.
2. Zapisz plik pod nazwą *Ludność Polski*.

Zapamiętaj

- Tworzenie wykresu rozpoczynamy od zaznaczenia komórek zawierających serie danych, które chcemy zilustrować.
- Warto opisać wykres – wstawić tytuł wykresu, opisy osi poziomej i pionowej, legendę (zwłaszcza gdy na wykresie przedstawiamy kilka serii danych).
- Na jednym wykresie można umieścić więcej niż jedną serię danych, ale nie powinno ich być zbyt wiele, gdyż wykres stanie się nieczytelny.
- Trzeba dostosować typ wykresu do rodzaju danych, które chcemy zobrazować. Jeśli chcemy:
  - porównać dane – najlepiej zastosować wykres słupkowy lub kolumnowy,
  - zobrazować proporcje lub udziały procentowe – najlepiej zastosować wykres kołowy,
  - przedstawić tendencje (wzrostowe lub spadkowe) – najlepiej stosować wykres liniowy.

OKO W OKO Z MONITOREM

Pytania i polecenia

1. Jakie są zalety prezentowania danych w postaci wykresów? Wymień przynajmniej trzy.
2. Jakie są kolejne kroki postępowania przy tworzeniu wykresów dla jednej serii danych?
3. Jak dodać kolejną serię danych do wykresu?
4. Na wykresie nie umieściliśmy opisu osi poziomej. Czy można uzupełnić ten opis? Wyjaśnij sposób postępowania.
5. Wyjaśnij, kiedy powinniśmy posługiwać się wykresem kolumnowym i słupkowym, kiedy kołowym, a kiedy liniowym. Podaj przykłady.
6. Pomyliliśmy się, wprowadzając dane, i już przygotowaliśmy na ich podstawie wykres. Czy po poprawieniu danych należy sporządzić wykres od nowa? Uzasadnij odpowiedź.

7. Sprawdź, jakiego typu wykresy zawiera arkusz kalkulacyjny, z którego korzystasz. Czy są one tylko dwuwymiarowe?

8. Wyjaśnij, dlaczego wykresu kołowego nie można stosować dla kilku serii danych.

Zadania

1. W pliku, który otrzymasz od nauczyciela, znajduje się tabela zawierająca liczby ocen od niedostatecznych do celujących z informatyki trzech klas ósmych. Porównaj dane na odpowiednim wykresie. Zapisz plik pod nazwą *Oceny z informatyki*.

	A	B	C	D	E	F	G
1		Oceny z informatyki					
2	Klasa	Liczba ocen niedostatecznych	Liczba ocen dopuszczających	Liczba ocen dostatecznych	Liczba ocen dobrych	Liczba ocen bardzo dobrych	Liczba ocen celujących
3	VIIIa	0	1	5	15	5	2
4	VIIIb	0	3	2	16	3	1
5	VIIIc	1	2	13	10	3	0
6							
7							
8							

Rys. 10. Tabela – zadanie 1.

2. Wielu uczniów uprawia różne dyscypliny sportu, ale są i tacy, którzy nie uprawiają żadnej (w tym zadaniu zakładamy, że dany uczeń uprawia maksymalnie jedną dyscyplinę sportu). W pliku, który otrzymasz od nauczyciela, znajduje się tabela zawierająca informacje o liczbie uczniów pewnej szkoły uprawiających konkretne dyscypliny sportu lub nieuprawiających sportu w ogóle. Utwórz wykres przedstawiający procentowy udział uczniów uprawiających poszczególne dyscypliny sportu. Zapisz plik pod nazwą *Dyscypliny sportu*.

3. Znajdź w internecie informacje o opadach deszczu w Polsce w czerwcu, lipcu i wrześniu bieżącego roku. Umieść dane w tabeli i przedstaw na wykresie. Zapisz plik pod nazwą *Opady deszczu*.

DLA ZAINTERESOWANYCH

1. Wyszukaj w internecie dane dotyczące średnich wyników egzaminu ósmoklasisty w poszczególnych województwach w poprzednim roku. Przygotuj tabelę i odpowiedni wykres. Zapisz plik pod nazwą *Egzamin*.

1.4. Zastosowania arkusza kalkulacyjnego

Warto powtórzyć

1. Jakie znasz sposoby adresowania w arkuszu kalkulacyjnym?
2. Czym jest seria danych?
3. W jaki sposób tworzy się wykresy z jedną serią danych, a w jaki – z kilkoma seriami?
4. Co należy uwzględniać podczas wybierania typu wykresu?

W arkuszu kalkulacyjnym można rozwiązywać zadania rachunkowe z różnych przedmiotów szkolnych, dziedzin nauki i życia codziennego.

W tym temacie:

1. wykorzystasz arkusz kalkulacyjny do kalkulacji wydatków,
2. wykorzystasz arkusz do rozwiązania zadań z fizyki i geografii,
3. przeprowadzisz symulację modelu rzutu sześcienną kostką do gry,
4. zastosujesz arkusz do porządkowania i filtrowania danych.

1 Kalkulacja wydatków

Załóżmy, że dysponujemy pewną kwotą, którą możemy wydać na zakup różnych artykułów. Musimy skalkulować, czy różnica między tą kwotą a wartością otrzymaną po zsumowaniu wszystkich wydatków będzie ujemna czy dodatnia. Inaczej mówiąc – określimy saldo.

W arkuszu kalkulacyjnym możemy wielokrotnie modyfikować dane, aby otrzymać saldo zerowe:

- jeśli saldo jest ujemne (czyli zabrakło pieniędzy), można zrezygnować z jakiegoś zakupu, usuwając wiersz z danym artykułem, albo zmniejszyć liczbę sztuk lub wagę danego produktu,
- jeśli saldo jest dodatnie (czyli została jeszcze pewna kwota do wydania), można zwiększyć liczbę lub wagę niektórych artykułów albo wstawić dodatkową pozycję.

W każdym przypadku formuły zostaną automatycznie przeliczone.

	A	B	C	D	E	F
1		kwota				
2		200				
3						
4	lp.	nazwa	liczba kg	cena w [zł]	wartość w [zł]	
5	1	ziemniaki	12	0,85		
6	2	cukierki	4	1,99		
7	3	jabłka	5	1,47		
8	4	mandarynki	3	7,50		
9	5	banany	7	5,64		
10	6	ciastka	5	1,96		
11				razem		
12				saldo		
13						
14						
15						
16						
17						

Rys. 1. Kalkulacja wydatków – ćwiczenie 1.

Ćwiczenie 1. Kalkulujemy wydatki

1. Dysponujesz kwotą 200 zł na przygotowanie przyjęcia urodzinowego. W pliku, który otrzymasz od nauczyciela, znajduje się tabela zawierająca planowane wydatki.
   a. Oblicz wartości artykułów (liczbę kilogramów danego artykułu pomnóż przez jego cenę jednostkową).
   b. Oblicz saldo. Jeśli jest ujemne, musisz z czegoś zrezygnować lub zmniejszyć liczbę kilogramów niektórych artykułów. Jeśli saldo jest dodatnie, możesz coś dokupić. Postaraj się wydać całą kwotę.
2. Zapisz plik pod nazwą *Saldo*.

Wskazówki:

- Skorzystaj z możliwości kopiowania formuły do obliczania wartości poszczególnych artykułów.
- W formule salda użyj adresu komórki zawierającej wartość kwoty do wydania (komórka B2). Sprawdź, jaką postać mają formuły do obliczania sumy i salda po wstawieniu lub usunięciu wiersza tabeli.

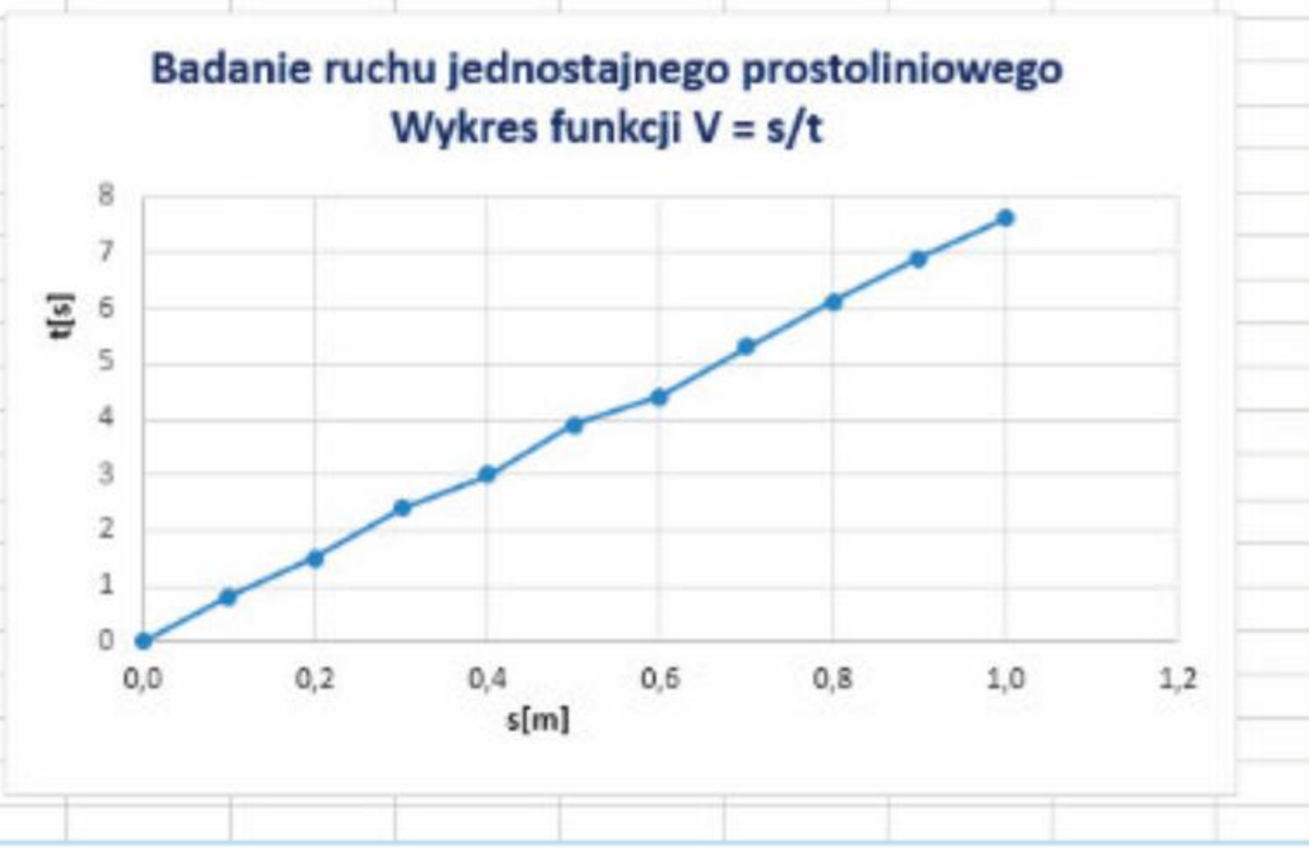
2 Rozwiązywanie zadań z fizyki i geografii

Fizyka – badanie ruchu jednostajnego prostoliniowego

Arkusz kalkulacyjny ułatwia opracowanie danych uzyskanych w doświadczeniu, np. w badaniu ruchu jednostajnego prostoliniowego. Obliczenia dla nowych danych program wykonuje automatycznie oraz przedstawia je na wykresie.

Opis doświadczenia: Obserwowano ruch pęcherzyka powietrza znajdującego się w szklanej rurce napełnionej cieczą. Rurka ma długość 100 cm. Zaznaczono na niej odcinki co 10 cm. Pęcherzyk powietrza utrzymuje się zawsze u góry rurki ustawionej pionowo. Jeśli odwrócimy rurkę, pęcherzyk znajdzie się na dole i ruchem jednostajnym zacznie przesuwać się do góry. W tabeli (rys. 2.) przedstawiono wyniki pomiaru czasu, w którym pęcherzyk powietrza przebył odcinki drogi tej samej długości. Doświadczenie można powtarzać wiele razy, a wyniki pomiarów – za każdym razem wpisywać do tabeli arkusza kalkulacyjnego. Wartości prędkości otrzymamy automatycznie, o ile poprawnie utworzymy formułę do jej obliczania.

	A	B	C
1	s [m]	t [s]	V [m/s]
2	0,0	0	
3	0,1	0,8	0,13
4	0,2	1,5	0,13
5	0,3	2,4	0,13
6	0,4	3,0	0,13
7	0,5	3,9	0,13
8	0,6	4,4	0,14
9	0,7	5,3	0,13
10	0,8	6,1	0,13
11	0,9	6,9	0,13
12	1,0	7,6	0,13



Rys. 2. Tabela i wykres do doświadczenia Badanie ruchu jednostajnego prostoliniowego – ćwiczenie 2.

Ćwiczenie 2. Badamy ruch jednostajny prostoliniowy

1. Zaprojektuj w arkuszu kalkulacyjnym tabelę do wpisywania wyników pomiarów w doświadczeniu.
2. Wpisz przykładowe wyniki. Utwórz formułę i przeprowadź obliczenia.
3. Przygotuj odpowiedni wykres.
4. Zapisz plik pod nazwą *Badanie ruchu*.

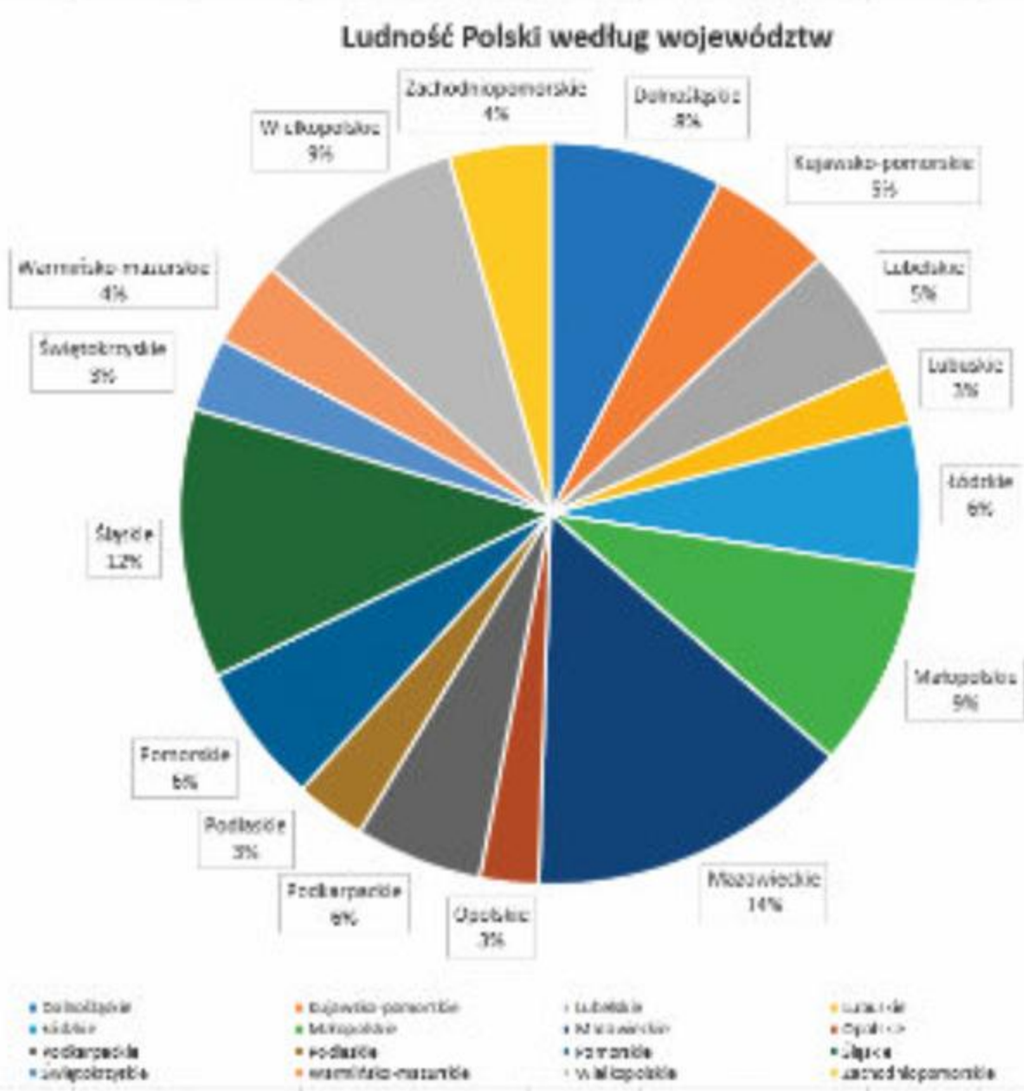
Podobne doświadczenie, niekoniecznie z pęcherzykiem powietrza, można wykonać na lekcji fizyki lub w domu, wykorzystując do obliczeń i sporządzenia wykresu arkusz kalkulacyjny.

Do przedstawiania wykresu prędkości odpowiedni jest wykres XY (punktowy), który wyświetla serie danych w postaci punktów w układzie współrzędnych. Aby połączyć punkty linią, można wybrać typ *Punktowy z prostymi liniami i znacznikami*.

Geografia – przygotowanie referatu o województwach w Polsce

Niektóre informacje z geografii można zamieszczać w tabelach lub ilustrować je za pomocą wykresów. Najczęściej są to wykresy kołowe lub kolumnowe. Tabele arkusza kalkulacyjnego i wykresy można włączać do treści referatów.

Lp.	Województwo	Ludność Polski według województw
1	Dolnośląskie	2898525
2	Kujawsko-pomorskie	2069273
3	Lubelskie	2103342
4	Lubuskie	1010177
5	Łódzkie	2448713
6	Małopolskie	3413931
7	Mazowieckie	5428031
8	Opolskie	980771
9	Podkarpackie	2125901
10	Podlaskie	1176576
11	Pomorskie	2346717
12	Śląskie	4500078
13	Świętokrzyskie	1230044
14	Warmińsko-mazurskie	1420514
15	Wielkopolskie	3500361
16	Zachodniopomorskie	1693219
	Polska	38354173



Rys. 3. Przykład wykresu kołowego obrazującego liczbę ludności Polski według województw (stan na 2020 r.; *http://demografia.stat.gov.pl/bazademografia/Tables.aspx)* – ćwiczenie 3.

Ćwiczenie 3. Wykonujemy wykres kołowy

1. W pliku, który otrzymasz od nauczyciela, znajduje się tabela zawierająca informacje o liczbie ludności Polski z podziałem na poszczególne województwa. Przedstaw dane w postaci wykresu kołowego. Umieść na wykresie etykiety danych (nazwy województw i wartości procentowe).
2. Zapisz plik pod nazwą *Ludność*.

3 Tworzenie modelu na przykładzie rzutu sześcienną kostką do gry

Wiele zjawisk obrazuje się za pomocą modeli. Dzięki obserwacji zachowania modelu wyciągamy wnioski dotyczące opisywanej rzeczywistości.

Przedstawimy prosty model rzutu sześcienną kostką do gry, który wykonamy w arkuszu kalkulacyjnym (rys. 4.).

E2 =LICZ.JEŻELI(B$2:B$7;6)

	A	B	C	D	E	F	G	H	I
1	**losowanie liczby**	**wynik rzutu kostką**		**dla 6 rzutów**			**dla 600 rzutów**		
2	0,8607478	6		liczba szóstek	1		liczba szóstek	85	
3	0,1263345	1		liczba piątek	1		liczba piątek	94	
4	0,1775813	2		liczba czwórek	0		liczba czwórek	102	
5	0,7287241	5		liczba trójek	1		liczba trójek	96	
6	0,0556075	1		liczba dwójek	1		liczba dwójek	115	
7	0,4877683	3		liczba jedynek	2		liczba jedynek	108	
8	0,4142114	3							
9	0,8136657	5		**dla 60 rzutów**			**dla 6000 rzutów**		
10	0,442708	3		liczba szóstek	8		liczba szóstek	982	
11	0,7043564	5		liczba piątek	10		liczba piątek	990	
12	0,1922782	2		liczba czwórek	7		liczba czwórek	953	
13	0,0013027	1		liczba trójek	10		liczba trójek	999	
14	0,6606808	4		liczba dwójek	13		liczba dwójek	1047	
15	0,8658768	6		liczba jedynek	12		liczba jedynek	1028	
16	0,2586107	2							
17	0,6349051	4							

Rys. 4. Przykład modelu rzutu kostką do gry zrealizowany w arkuszu kalkulacyjnym – ćwiczenie 4.

Ćwiczenie 4. Badamy model rzutu kostką do gry

1. Utwórz w arkuszu kalkulacyjnym tabelę przedstawiającą model rzutu sześcienną kostką do gry (rys. 4.).
2. Losuj kolejne wyniki rzutów (naciskając klawisz F9) i obserwuj, jak zmieniają się wartości w tabeli – ile razy wypadły jedynka, dwójka, trójka, czwórka, piątka, szóstka.
3. Zapisz plik pod nazwą *Kostka*.

Wskazówki:

- Skorzystaj z funkcji LOS(), która generuje losowe liczby rzeczywiste z przedziału ⟨0; 1). W komórce A1 umieść formułę =LOS(). Skopiuj ją do komórek A2:A6001.
- W komórce B2 umieść formułę =LICZBA.CAŁK(A2*6+1), która przekształci liczby z kolumny A na liczby całkowite z zakresu ⟨1; 6⟩, czyli możliwe wyniki rzutu kostką. Skopiuj tę formułę do pozostałych komórek w kolumnie B.
- Aby obliczyć, ile razy dla różnej liczby rzutów wypadła szóstka, piątka, czwórka itd., skorzystaj ze standardowej funkcji LICZ.JEŻELI, która we wskazanym zakresie oblicza liczbę komórek spełniających określone kryteria. W komórce E2 umieść formułę =LICZ.JEŻELI(B$2:B$7;6), skopiuj ją do komórek E10, H2 i H10 i odpowiednio zmodyfikuj zakres danych (z kolumny B), aby dostosować go do symulowanej liczby rzutów.
- Formuły z komórek E2, E10, H2 i H10 skopiuj do komórek, w których będą zliczane liczby piątek, czwórek, trójek, dwójek, jedynek i odpowiednio zmodyfikuj kryterium (wynik rzutu).

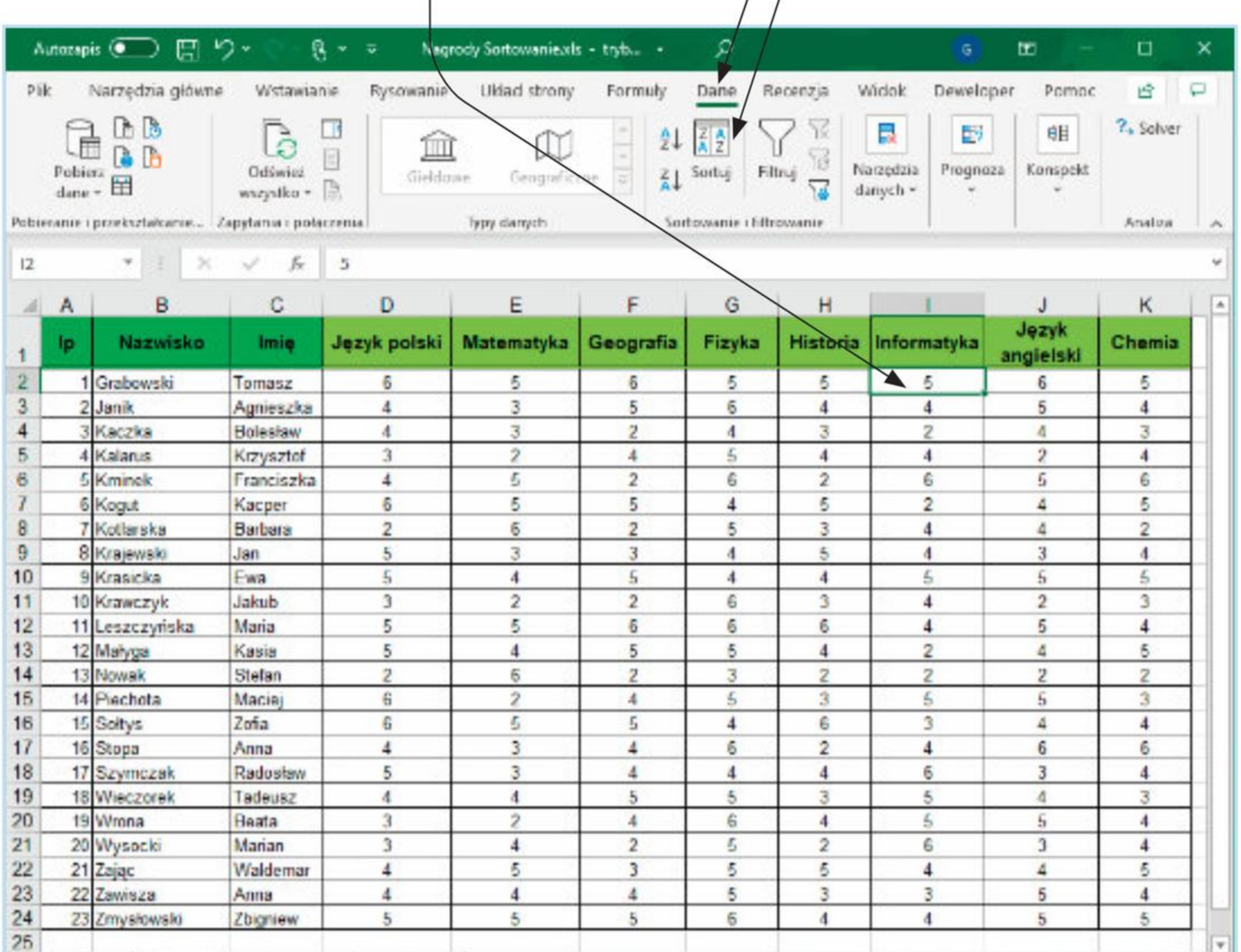
4 Porządkowanie i filtrowanie danych

W tabeli arkusza kalkulacyjnego (rys. 5.) znajdują się oceny uczniów, uporządkowane alfabetycznie według ich nazwisk. Chcemy uporządkować te dane w kolejności od najwyższej do najniższej oceny z informatyki lub znaleźć nazwiska uczniów, którzy z matematyki otrzymali szóstki. Ręczne wykonanie takich operacji, zwłaszcza dla dużej liczby uczniów, jest pracochłonne. W arkuszu kalkulacyjnym tego typu zadania możemy wykonać, stosując opcje sortowania i filtrowania. Ważne jest, aby kolumny tabeli miały nagłówki (szczególnie przy stosowaniu opcji filtrowania) – na rysunku 5. umieszczono je w pierwszym wierszu.

Po wybraniu sposobu sortowania dla określonej kolumny program uporządkuje wszystkie wiersze tabeli według wartości znajdujących się w tej kolumnie.

Kliknij dowolną komórkę w kolumnie, w której chcesz posortować dane.

Na karcie **Dane** wybierz sposób sortowania: **Sortuj od najmniejszych do największych** lub **Sortuj od największych do najmniejszych**.



	A	B	C	D	E	F	G	H	I	J	K
1	lp	Nazwisko	Imię	Język polski	Matematyka	Geografia	Fizyka	Historia	Informatyka	Język angielski	Chemia
2	1	Grabowski	Tomasz	6	5	6	5	5	5	6	5
3	2	Janik	Agnieszka	4	3	5	6	4	4	5	4
4	3	Kaczka	Bolesław	4	3	2	4	3	2	4	3
5	4	Kalarus	Krzysztof	3	2	4	5	4	4	2	4
6	5	Kminek	Franciszka	4	5	2	6	2	6	5	6
7	6	Kogut	Kacper	6	5	5	4	5	2	4	5
8	7	Kotlarska	Barbara	2	6	2	5	3	4	4	2
9	8	Krajewski	Jan	5	3	3	4	5	4	3	4
10	9	Krasicka	Ewa	5	4	5	4	4	5	5	5
11	10	Krawczyk	Jakub	3	2	2	6	3	4	2	3
12	11	Leszczyńska	Maria	5	5	6	6	6	4	5	4
13	12	Małyga	Kasia	5	4	5	5	4	2	4	5
14	13	Nowak	Stefan	2	6	2	3	2	2	2	2
15	14	Piechota	Maciej	6	2	4	5	3	5	5	3
16	15	Sołtys	Zofia	6	5	5	4	6	3	4	4
17	16	Stopa	Anna	4	3	4	6	2	4	6	6
18	17	Szymczak	Radosław	5	3	4	4	4	6	3	4
19	18	Wieczorek	Tadeusz	4	4	5	5	3	5	4	3
20	19	Wrona	Beata	3	2	4	6	4	5	5	4
21	20	Wysocki	Marian	3	4	2	5	2	6	3	4
22	21	Zając	Waldemar	4	5	3	5	5	4	4	5
23	22	Zawisza	Anna	4	4	4	5	3	3	5	4
24	23	Zmysłowski	Zbigniew	5	5	5	6	4	4	5	5
25											

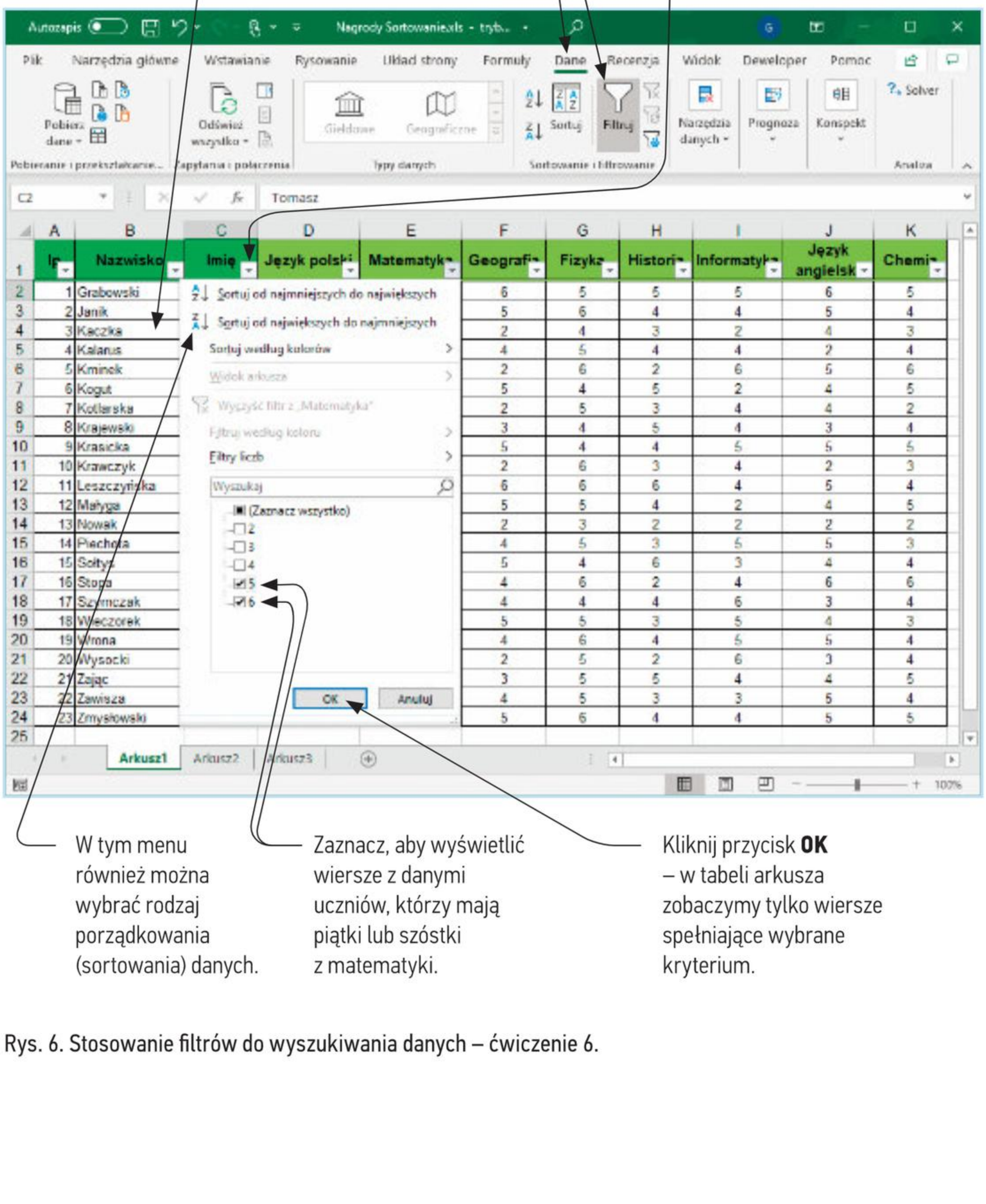
Rys. 5. Sortowanie danych w tabeli – ćwiczenie 5.

Ćwiczenie 5. Porządkujemy dane w tabeli

W pliku, który otrzymasz od nauczyciela, znajduje się tabela zawierająca fikcyjne nazwiska uczniów i ich oceny z poszczególnych przedmiotów.

1. Uporządkuj dane uczniów według ocen z informatyki – od najwyższej do najniższej.
2. Zapisz plik pod nazwą *Nagrody_sortowanie*.

W programie LibreOffice Calc opcje sortowania i filtrowania umieszczono w menu **Dane**.



Rys. 6. Stosowanie filtrów do wyszukiwania danych – ćwiczenie 6.

Po wybraniu określonej liczby w kolumnie (np. *Matematyka*) ograniczymy wyświetlanie tabeli do wierszy zawierających tę liczbę w wybranej kolumnie, ponieważ program ukrywa wiersze, które nie spełniają wybranego kryterium.

Aby wyłączyć widok filtru, należy wybrać tę samą opcję, która włącza ten widok (**Filtr/Autofiltr** lub **Filtruj**).

Aby ponownie wyświetlić wszystkie wiersze, należy przy polach, dla których ustawiono filtry, zaznaczyć opcję wyświetlenia wszystkich wierszy lub opcję wyczyszczenia filtru (np. **Wyczyść**).

Aby wybrać niestandardowe filtry (np. **Mniejsze niż**, **Mniejsze niż lub równe**), należy w oknie wyboru kryterium kliknąć polecenie **Filtry liczb**.

Ćwiczenie 6. Stosujemy filtrowanie do wyszukiwania danych

1. Otwórz plik *Nagrody_sortowanie* zapisany w ćwiczeniu 5.
2. Uporządkuj dane uczniów według liczby porządkowej (Lp.) – rosnąco (od najmniejszej do największej).
3. Wyświetl w tabeli tylko dane uczniów, którzy otrzymali z matematyki piątki lub szóstki.
4. Zapisz plik pod nazwą *Nagrody_filtrowanie*.

Zapamiętaj

- Arkusz kalkulacyjny możemy zastosować:
  - na lekcji fizyki (np. do pokazania ruchu jednostajnego w postaci wykresu),
  - na lekcji geografii (np. do zilustrowania danych w postaci tabel i wykresów)
  - do tworzenia modeli zjawisk.
- Dzięki modelowaniu możemy wyciągać wnioski dotyczące rzeczywistości opisywanej w modelach.
- Sortowanie i filtrowanie danych w arkuszu kalkulacyjnym pozwala na ich porządkowanie i wyszukiwanie według określonych kryteriów.

Serwis dla ciekawskich

W 6000 losowań Lotto (dawniej: Dużego Lotka) najczęściej wypadała liczba 34 (792 razy), a najrzadziej – liczba 43 (657 razy). Możesz się o tym przekonać samodzielnie, korzystając z arkusza kalkulacyjnego oraz informacji o wynikach losowań znalezionych w internecie.

OKO W OKO Z MONITOREM

Pytania i polecenia

1. Dlaczego do narysowania w arkuszu kalkulacyjnym wykresu prędkości w ruchu jednostajnym prostoliniowym najlepiej wybrać wykres XY (punktowy), a do przedstawienia danych dotyczących ludności Polski – wykres kołowy?
2. W jakim celu buduje się modele rzeczywistych zjawisk? Na przykładzie modelu rzutu kostką wyjaśnij, na czym polegają modelowanie i symulacja.
3. W jaki sposób można uporządkować alfabetycznie dane umieszczone w tabeli arkusza kalkulacyjnego?
4. Jak w tabeli arkusza kalkulacyjnego wyświetlić tylko te wiersze, które spełniają określone kryterium?

Zadania

1. W pliku, który otrzymasz od nauczyciela, znajduje się tabela zawierająca przykładowe dane dotyczące kalkulacji budżetu pewnej szkoły. Saldo jest ujemne. Postaraj się tak zmienić wydatki, aby saldo było zerowe. Zapisz plik pod nazwą *Budżet szkoły*.

	A	B	C	D	E	F	G	H
1	**Kwota budżetu:**	**700000**						
2		**Planowane wydatki**						
3		**nazwa**	**I kwartał**	**II kwartał**	**III kwartał**	**IV kwartał**	**suma**	
4	1	2	3	4	5	6	7	
5	**wydatki osobowe**	nauczyciele	55300	54254	57263	56224	223041	
6		pracownicy administracji i obsługi	18770	17910	17558	17920	72158	
7	**wydatki rzeczowe**	pomoce dydaktyczne	20440	10455	10500	5230	46625	
8		inne	1000	72500	15211	35426	124137	
9		środki czystości	1500	2254	2000	3241	8995	
10	**inne**	remonty	2520	2500	75825	3204	84049	
11		modernizacje	12050	24500	15612	12400	64562	
12		opał	12000	2560	2560	12000	29120	
13		energia elektryczna	18006	12485	6280	28457	65228	
14		nagrody dla uczniów	1200	1754	450	1526	4930	
15		**suma:**	142786	201172	203259	175628	722845	
16						**saldo:**	**-22845**	

Rys. 7. Budżet szkoły – zadanie 1.

2. Przygotuj kalkulację wydatków domowych w jednym miesiącu. Uwzględnij planowane dochody oraz opłaty stałe. Zapisz plik pod nazwą *Wydatki domowe*.

3. W pliku, który otrzymasz od nauczyciela, znajdują się wyniki pomiarów temperatury wody z jeziora. Bezpośrednio po przyniesieniu do laboratorium woda miała temperaturę 11°C. Po upływie czterech minut termometr wskazywał 14°C, a po następnych czterech minutach 17°C. Cztery kolejne pomiary, przeprowadzane co 4 minuty, wskazywały temperaturę 20°C. Sporządź wykres zależności temperatury próbki wody od czasu pomiaru. Zapisz plik pod nazwą *Temperatura wody*.

4. W pliku, który otrzymasz od nauczyciela, znajdują się dane dotyczące użytków rolnych w Polsce. Przedstaw te dane na wykresie. Utwórz formułę obliczającą procentowy udział danego rodzaju użytków w ogólnej powierzchni użytków rolnych. Czy wykres kołowy będzie tu odpowiedni? Uzasadnij odpowiedź. Zapisz plik pod nazwą *Użytki rolne*.

5. Przygotuj tabelę i wykres z danymi dotyczącymi struktury zatrudnienia w Polsce. Uwzględnij zatrudnienie m.in. w przemyśle, rolnictwie, budownictwie, komunikacji, handlu, oświacie, służbie zdrowia. Jaki typ wykresu najlepiej zastosować? Dane odszukaj w internecie, podręcznikach szkolnych, encyklopedii lub innych źródłach. Zapisz plik pod nazwą *Zatrudnienie*.

6. Korzystając z arkusza kalkulacyjnego, wykonaj model rzutu monetą. Zapisz plik pod nazwą *Moneta*.

7. W pliku, który otrzymasz od nauczyciela, umieszczono wykaz szkolnych lektur (imię i nazwisko autora, tytuł):
   a. uporządkuj dane alfabetycznie według nazwiska autora,
   b. wyświetl tylko wiersze z utworami dwóch twoich ulubionych autorów.

   Zapisz w pliku pod nazwą *Lektury*.

DLA ZAINTERESOWANYCH

1. Przygotuj w edytorze tekstu referat z geografii o polskim przemyśle, umieszczając w dokumencie odpowiednie tabele i wykresy.

2. Zaprojektuj doświadczenie badające ruch jednostajnie zmienny (np. staczanie się piłki lub wózka po równi pochyłej). Przygotuj w arkuszu kalkulacyjnym odpowiednie tabele i wykresy, np. wykres prędkości piłeczki na równi pochyłej, wykres położenia piłeczki (wózka) w zależności od czasu.

3. Korzystając z arkusza kalkulacyjnego, wykonaj model rzutu dwiema i trzema kostkami do gry. Jaka suma oczek wypada najczęściej przy rzucie dwiema kostkami, a jaka – przy rzucie trzema kostkami?

Programowanie w języku Python

.py

2.1. Wprowadzenie do programowania w języku Python

Na lekcjach matematyki i fizyki większość zadań rozwiązujemy według pewnych schematów: wypisujemy dane, precyzujemy, które wartości należy obliczyć, czyli określamy szukane, wybieramy metodę rozwiązania, wypisujemy wzory łączące dane z szukanymi, przeprowadzamy obliczenia, formułujemy odpowiedź. Rozwiązanie wybranego zadania przedstawiamy, używając wzorów, symboli oraz komentarzy słownych. W zadaniach na lekcjach informatyki postępujemy podobnie, przedstawiając rozwiązanie, np. graficznie lub w postaci programu komputerowego. W tym temacie:

1. dowiesz się, czym jest algorytm i jakie są sposoby jego przedstawiania,
2. dowiesz się, na czym polega programowanie,
3. poznasz środowisko programistyczne IDLE,
4. napiszesz pierwszy program w języku Python (czyt. pajton).

1 Sposoby przedstawiania algorytmów

Algorytm to uporządkowany i uściślony sposób rozwiązania danego problemu, zawierający szczegółowy opis wykonywanych czynności w skończonej liczbie kroków.

Aby rozwiązać dowolny problem (zadanie), nie tylko z informatyki, należy rozpocząć od poprawnego sformułowania problemu oraz ustalenia danych i warunków, które każda z danych musi spełniać (np. czy ma to być tekst, data, liczba). Należy też określić cel, czyli wyniki (szukane), które chcemy otrzymać, a także warunki, jakie powinny one spełniać. Następnie należy zastanowić się nad sposobem rozwiązania, czyli algorytmem, a także wyborem odpowiedniego narzędzia, np. programu komputerowego (rys. 1.).

Rys. 1. Schemat postępowania podczas rozwiązywania problemów za pomocą komputera

Etapy rozwiązywania problemów

W procesie rozwiązywania dowolnego zadania, nie tylko za pomocą komputera, można wyróżnić kilka etapów:

1. Sformułowanie zadania.
2. Określenie danych wejściowych.
3. Ustalenie celu, czyli wyniku.
4. Określenie metody rozwiązania, czyli wybór algorytmu.
5. Przedstawienie algorytmu w wybranej postaci.
6. Analiza poprawności rozwiązania.
7. Testowanie rozwiązania dla różnych danych – ocena efektywności przyjętej metody.

Algorytmy można przedstawiać w postaci:

- opisu słownego,
- listy kroków,
- schematu blokowego,
- programu w wybranym języku programowania.

Różne sposoby przedstawiania algorytmu pokażemy na przykładzie algorytmu obliczania pola trójkąta.

Specyfikacja zadania

Specyfikacja zadania (problemu) to szczegółowy opis zadania, w którym określa się dane wejściowe i oczekiwane wyniki oraz związek między danymi a wynikami (warunki, jakie muszą spełniać wyniki).

Zadanie: Oblicz pole trójkąta o podstawie a i wysokości h.

Specyfikacja zadania:

Dane: Dowolne liczby rzeczywiste dodatnie: a, h (a – długość podstawy trójkąta, h – długość wysokości trójkąta opuszczonej na podstawę).

Wynik: Wartość pola trójkąta: p.

Opis słowny algorytmu

Opis słowny algorytmu to przedstawienie go w postaci ogólnego opisu wykonywanych operacji, bez podawania szczegółów. Może zawierać przemyślenia dotyczące sposobów rozwiązania postawionego zadania.

Opis algorytmu: Pole trójkąta możemy policzyć, mnożąc długość jednego boku trójkąta przez długość wysokości opuszczonej na ten bok i dzieląc wynik przez 2. Ponieważ w naszym zadaniu mowa jest o podstawie trójkąta, przyjmiemy, że jest ona tym bokiem.

Użyte w kroku 3. słowo „przypisz" zapożyczono z języków programowania. Na przykład w języku Python zapis: `p = a * h / 2` to instrukcja przypisania, w której zmiennej po lewej stronie (czyli `p`) przypisuje się wartość wyrażenia znajdującego się po prawej stronie instrukcji.

Lista kroków algorytmu

Lista kroków algorytmu to przedstawienie algorytmu w kolejnych punktach (krokach). Każdy punkt takiej listy zawiera opis wykonywanej czynności. Kolejność punktów nie może być przypadkowa – musi być zgodna z działaniem algorytmu.

Lista kroków:

1. Zacznij algorytm.
2. Wprowadź wartości liczb a i h.
3. Zmiennej p przypisz wartość wyrażenia $\frac{a \cdot h}{2}$.
4. Wyprowadź wynik: p.
5. Zakończ algorytm.

Ćwiczenie 1. Zapisujemy algorytm w postaci listy kroków

Zapisz specyfikację zadania i listę kroków algorytmu obliczania pola trapezu.

Wskazówka: Wzór na pole trapezu: $p = \frac{(a + b) \cdot h}{2}$, gdzie a, b to długości podstaw trapezu, h – długość wysokości trapezu.

Schemat blokowy algorytmu

W schemacie blokowym poszczególne operacje przedstawia się za pomocą odpowiednio połączonych figur (bloków). Połączenia określają kolejność i sposób wykonania operacji w danym algorytmie.

W tabeli 1. opisano figury, które stosuje się do budowania schematów blokowych.

Reprezentacja graficzna	Opis operacji	Uwagi
	Początek algorytmu	Blok z napisem „Start" zaczyna algorytm. Wychodzi z niego tylko jedno połączenie i żadne do niego nie wchodzi. W schemacie może funkcjonować tylko jeden taki blok.
	Zakończenie algorytmu	Blok z napisem „Koniec" kończy algorytm. Wchodzi do niego jedno połączenie, żadne nie wychodzi. W schemacie może być wiele takich bloków.
	Wprowadzanie danych (*blok wejścia*)	Blok z napisem „Wprowadź" służy do wprowadzania danych. Ma jedno połączenie wchodzące i jedno wychodzące. W schemacie może być wiele takich bloków.
	Wyprowadzanie wyników (*blok wyjścia*)	Blok z napisem „Wyprowadź" służy do wyprowadzania wyników. Ma jedno połączenie wchodzące i jedno wychodzące. W schemacie może być wiele takich bloków.

	Wykonywanie działań (*blok operacyjny*)	Blok, w którym wykonywane są różne operacje, m.in. obliczenia. Ma jedno połączenie wchodzące i jedno wychodzące. W jednym bloku można wpisać więcej niż jedno wyrażenie. W schemacie może być wiele takich bloków.
	Sprawdzanie warunku (*blok warunkowy albo decyzyjny*)	Blok podejmowania decyzji. Wchodzi do niego jedno połączenie, wychodzą dwa: • z napisem „Tak”, gdy warunek jest spełniony; • z napisem „Nie”, gdy warunek nie jest spełniony. W schemacie może być wiele takich bloków.
	Łącznik	Łącznik stosuje się, gdy schemat blokowy rysujemy w kilku częściach, np. na dwóch stronach. Umieszczony wewnątrz numer powinien być taki sam w obu łączonych częściach.
	Połączenie	Połączenie łączy bloki. Tworzy je linia prosta bądź łamana, zakończona strzałką. Połączenie może dochodzić również do innego połączenia.

Tabela 1. Figury geometryczne stosowane w graficznym przedstawianiu algorytmów

Ćwiczenie 2. Sprawdzamy działanie algorytmu

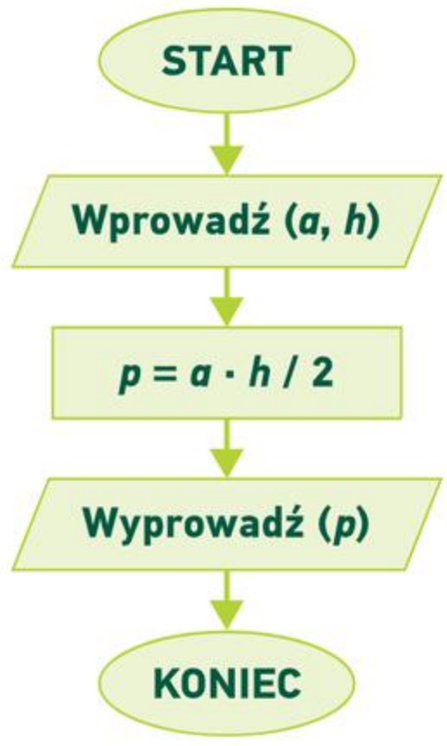
Przetestuj działanie algorytmu pokazanego na rysunku 2. dla następujących liczb: (17; 9), (5; 10), (24; 13).

Do rysowania schematów blokowych można wykorzystać **Kształty**, dostępne w programach: Microsoft Word, Microsoft PowerPoint lub Paint. Można również poszukać w internecie darmowych programów, służących do tworzenia schematów blokowych.

Ćwiczenie 3. Budujemy schemat blokowy

Narysuj schemat blokowy algorytmu na podstawie listy kroków utworzonej w ćwiczeniu 1.

Wskazówka: Jeśli wykonujesz zadanie w edytorze tekstu, korzystając z **Kształtów**, pamiętaj o możliwości grupowania wstawionych obiektów.



Rys. 2. Schemat blokowy algorytmu obliczania pola trójkąta

W algorytmach informatycznych dane wejściowe powinny być precyzyjnie określone, a wszystkie operacje – dokładnie opisane. Dla tych samych danych powinniśmy otrzymać te same wyniki.

Przykład algorytmu nieinformatycznego

W algorytmach nieinformatycznych wynik może zależeć od danych wejściowych i różnych czynników (np. od czasu gotowania). Przykładem algorytmu nieinformatycznego jest przepis na przygotowanie kisielu:

1. Z 0,5 litra zimnej wody odlać połowę.
2. Wsypać do niej zawartość torebki.
3. Dobrze wymieszać.
4. Pozostałą część wody zagotować.
5. Dodać trzy łyżki cukru.
6. Do wrzątku wlać rozrobiony proszek.
7. Gotować przez chwilę.
8. Kisiel wlać do salaterek wypłukanych zimną wodą.
9. Podawać gorący lub wystudzony – z konfiturami, sokiem lub śmietaną.

W podanej liście kroków niektóre sformułowania są mało precyzyjne, np. w punkcie 7.: „gotować przez chwilę". Zależnie od tego, ile ta chwila będzie trwała, możemy otrzymać różne wyniki (np. kisiel o różnej konsystencji).

2 Na czym polega programowanie

Aby przedstawić algorytm w postaci **programu komputerowego**, trzeba zapisać go jako ciąg instrukcji języka programowania. Powstaje wówczas tzw. program (kod) źródłowy.

Program komputerowy (kod źródłowy)
Ciąg instrukcji języka programowania, realizujący algorytm.

Język programowania jest specjalnym językiem służącym do pisania programów komputerowych. Jest on zbiorem określonych instrukcji i zasad. Każde słowo lub znak w tym języku (poza komentarzami) jest instrukcją lub jej częścią.

Każda instrukcja programu, podobnie jak figura (blok) w schemacie blokowym, odpowiada określonej operacji. Kolejność występowania w programie instrukcji decyduje o kolejności wykonywania operacji.

Języki programowania możemy podzielić na:

- języki wysokiego poziomu, np. Python, C++, Java, Visual Basic;
- języki niskiego poziomu (wewnętrzne), np. assemblery.

Program napisany w języku wysokiego poziomu (kod źródłowy) jest niezrozumiały dla komputera. Komputer potrafi wykonywać tylko instrukcje (rozkazy) zapisane w języku wewnętrznym, zrozumiałym dla procesora (kod wynikowy).

Proces tłumaczenia programu napisanego w języku programowania wysokiego poziomu na język wewnętrzny komputera nazywamy translacją. Może on przebiegać w formie **kompilacji** lub **interpretacji**.

Tłumaczeniem kodu źródłowego zajmuje się specjalny program, tzw. translator (kompilator lub interpreter). Tłumaczenie kodu łączy się ze sprawdzaniem poprawności zapisanych instrukcji. Jeśli instrukcja jest niepoprawna, to translator jej nie przetłumaczy i wyświetli komunikat o błędzie. Trzeba ją wówczas odszukać, poprawić i powtórnie przeprowadzić translację.

Rys. 3. Etapy powstawania programu komputerowego

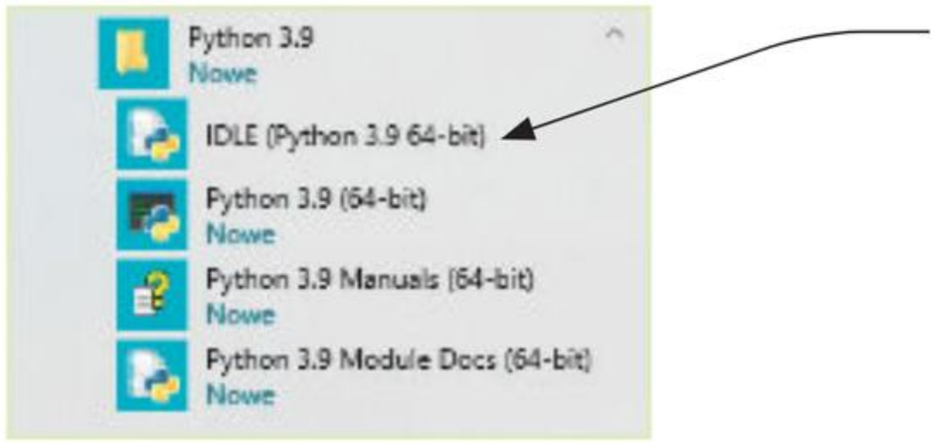
Kompilacja
Przetłumaczenie programu tworzonego w jednym z języków programowania w całości na język zrozumiały dla procesora, tak by mógł go wykonać komputer. Raz skompilowany program nie wymaga już powtórnej operacji tłumaczenia. Przykładowe języki kompilowane to C++, Pascal.

Interpretacja
Tłumaczenie programu tworzonego w jednym z języków programowania instrukcja po instrukcji, tak by komputer mógł wykonać każdą z nich. Tłumaczenie następuje każdorazowo przy uruchomieniu programu. Przykładem języków interpretowanych są języki edukacyjne Logo i Scratch, język Python oraz języki wykorzystywane w tworzeniu stron WWW, np. PHP i JavaScript.

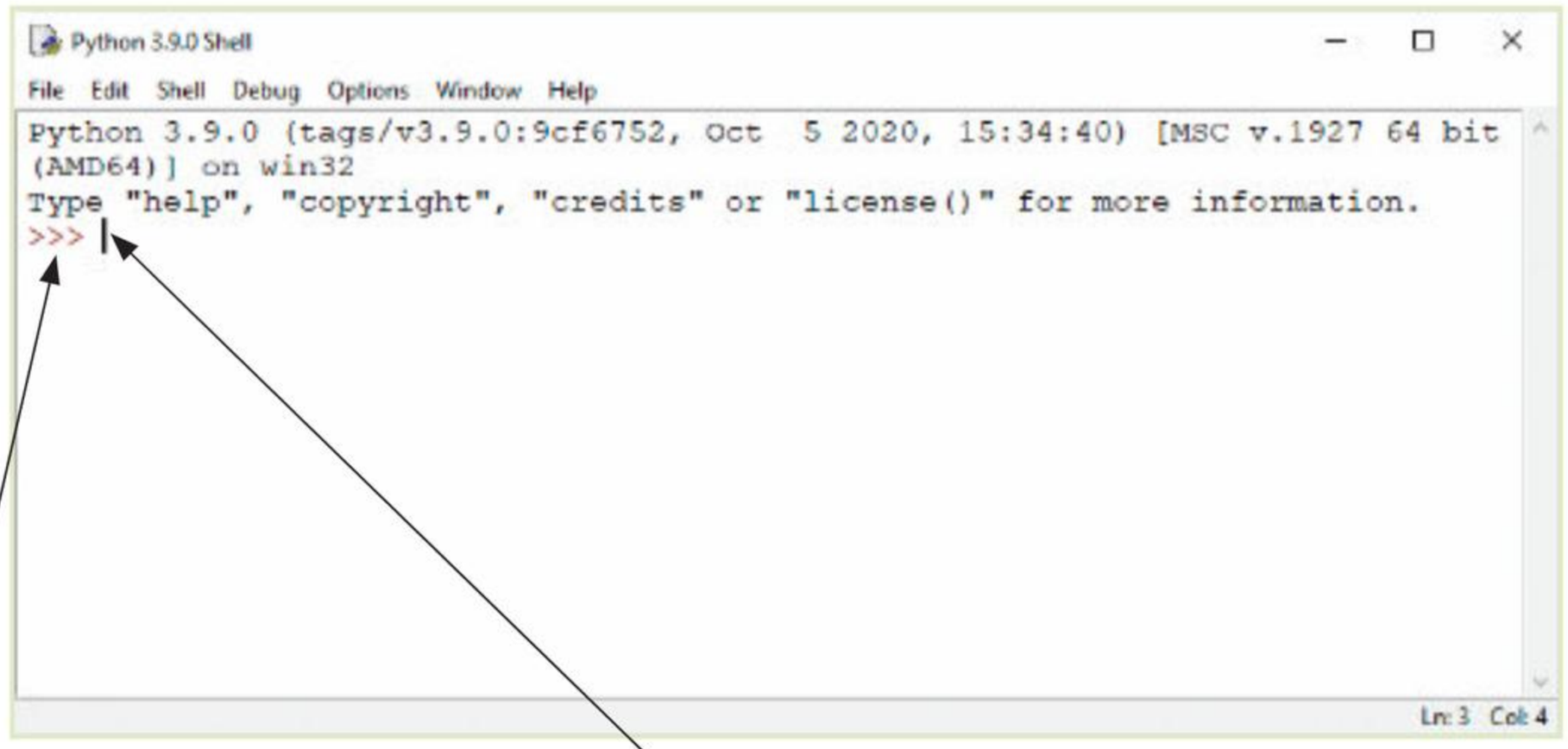
3 Środowisko programistyczne IDLE

Aby tworzyć programy, możemy korzystać z wyspecjalizowanych pakietów programistycznych, zawierających zwykle edytor kodu źródłowego, a także kompilator lub interpreter oraz inne narzędzia wspomagające programowanie.

Aby pisać programy w języku Python, należy zainstalować darmową aplikację Python, w której skład wchodzi zintegrowane środowisko programistyczne IDLE. Możemy również skorzystać z innych aplikacji, np. Mu lub PyCharm, lub użyć interpretera dostępnego przez przeglądarkę internetową, takiego jak *ideone.com* lub *repl.it*.



Rys. 4a. Uruchamianie środowiska IDLE (wersja Python 3.9 64-bitowa)



Rys. 4b. Okno powłoki Pythona (Python Shell)

Środowisko IDLE zawiera narzędzia ułatwiające pisanie programów, m.in.: edytor kodu źródłowego, interpreter i tzw. powłokę (ang. *shell*, czyt. szel).

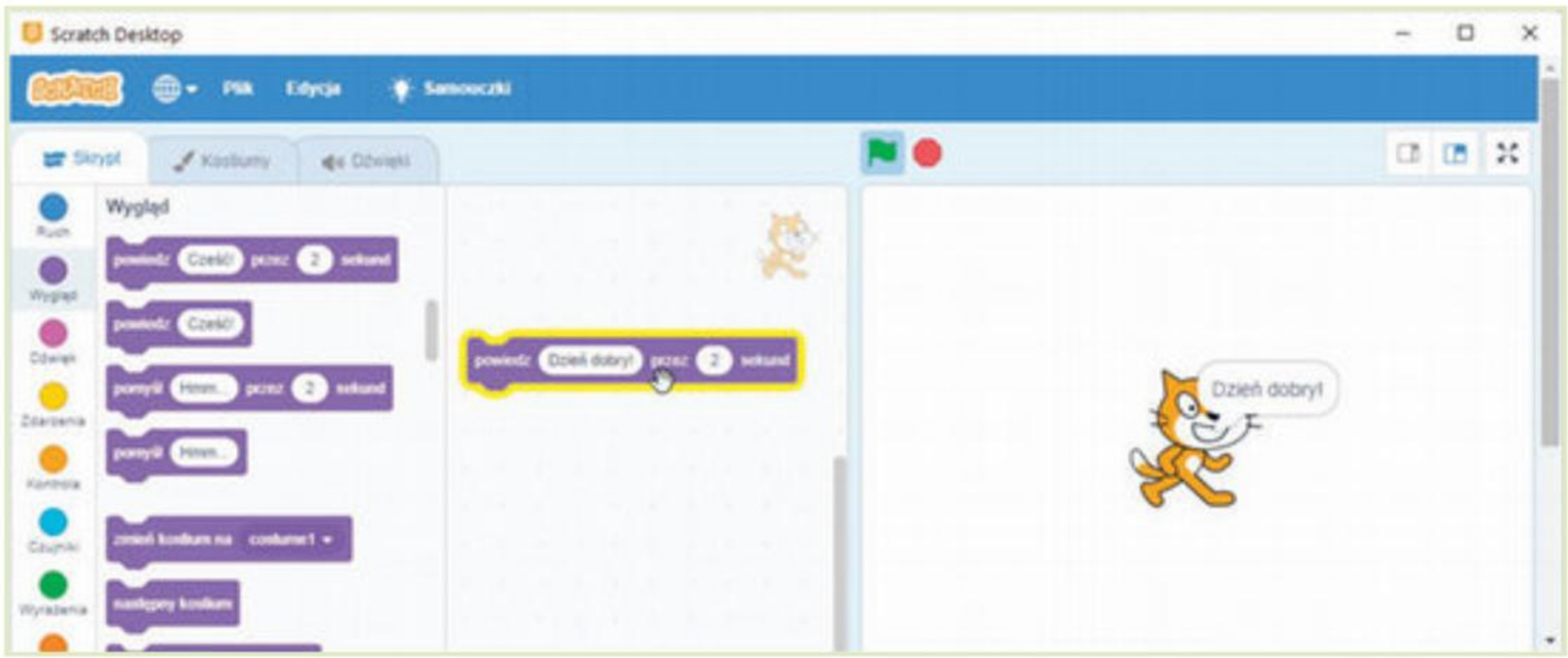
Środowisko programistyczne IDLE udostępnia dwa tryby pracy: interaktywny i skryptowy. Powłoka Pythona (rys. 4b) uruchamia się w trybie interaktywnym i umożliwia m.in. pracę w tym trybie.

Tryb interaktywny

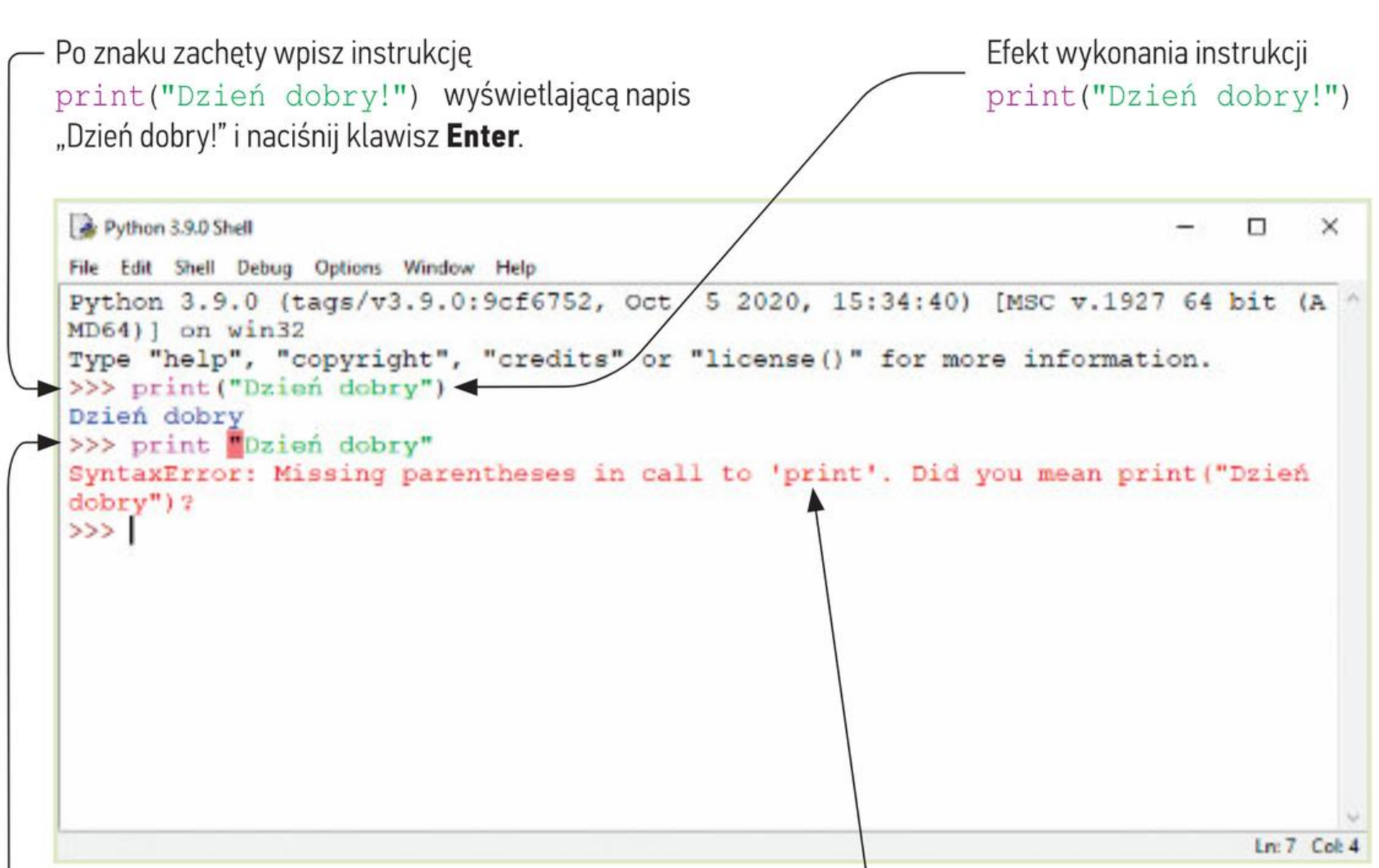
W językach interpretowanych (np. Scratch, Python) możemy napisać jedno polecenie i zatwierdzić je (klawiszem **Enter** lub kliknięciem myszy). Interpreter przetłumaczy to polecenie na język wewnętrzny komputera. Jeśli będzie poprawne, komputer je wykona. Mówimy wówczas, że pracujemy w trybie interaktywnym.

Na przykład jeśli w języku Scratch umieścimy w obszarze tworzenia programu polecenie **powiedz** wyświetlające na ekranie napis „Dzień dobry!”, a następnie klikniemy element reprezentujący to polecenie, komputer od razu je wykona. Efekt zobaczymy na scenie (rys. 5a).

Podobnie komputer wykonuje polecenia w języku Python. Jeśli w oknie powłoki napiszemy polecenie wyświetlające na ekranie napis „Dzień dobry!”: `print("Dzień dobry!")`, a następnie zatwierdzimy je klawiszem **Enter**, komputer również od razu je wykona. Efekt zobaczymy w wierszu poniżej polecenia (rys. 5b). Jeśli polecenie nie będzie poprawne, interpreter wyświetli komunikat o błędzie (rys. 5b).



Rys. 5a. Przykład wykonania polecenia w języku Scratch



Rys. 5b. Przykład pracy w trybie interaktywnym w języku Python w oknie powłoki Pythona

Ćwiczenie 4. Wyświetlamy na ekranie napisy w języku Python – w trybie interaktywnym

1. Uruchom program Python.
2. W otwartym oknie napisz polecenie wyświetlające napis „Dzień dobry!”.
3. Wpisz kolejne polecenie wyświetlające napis „Uczymy się programowania”. Wymień podobieństwa i różnice w sposobie wykonania tego zadania w językach Scratch i Python.

W trybie interaktywnym języka Python możemy sprawdzić działanie poleceń, np. gdy chcemy od razu zobaczyć wynik czy sprawdzić działanie konkretnej instrukcji (w ćwiczeniu 4. sprawdziliśmy działanie instrukcji `print()`).

4 Piszemy program w języku Python

W językach interpretowanych (np. Scratch, Python) możemy napisać program składający się z wielu poleceń (tworząc tzw. skrypt), zapisać go w pliku, a następnie uruchomić. Mówimy wówczas, że pracujemy w trybie skryptowym.

Tryb skryptowy

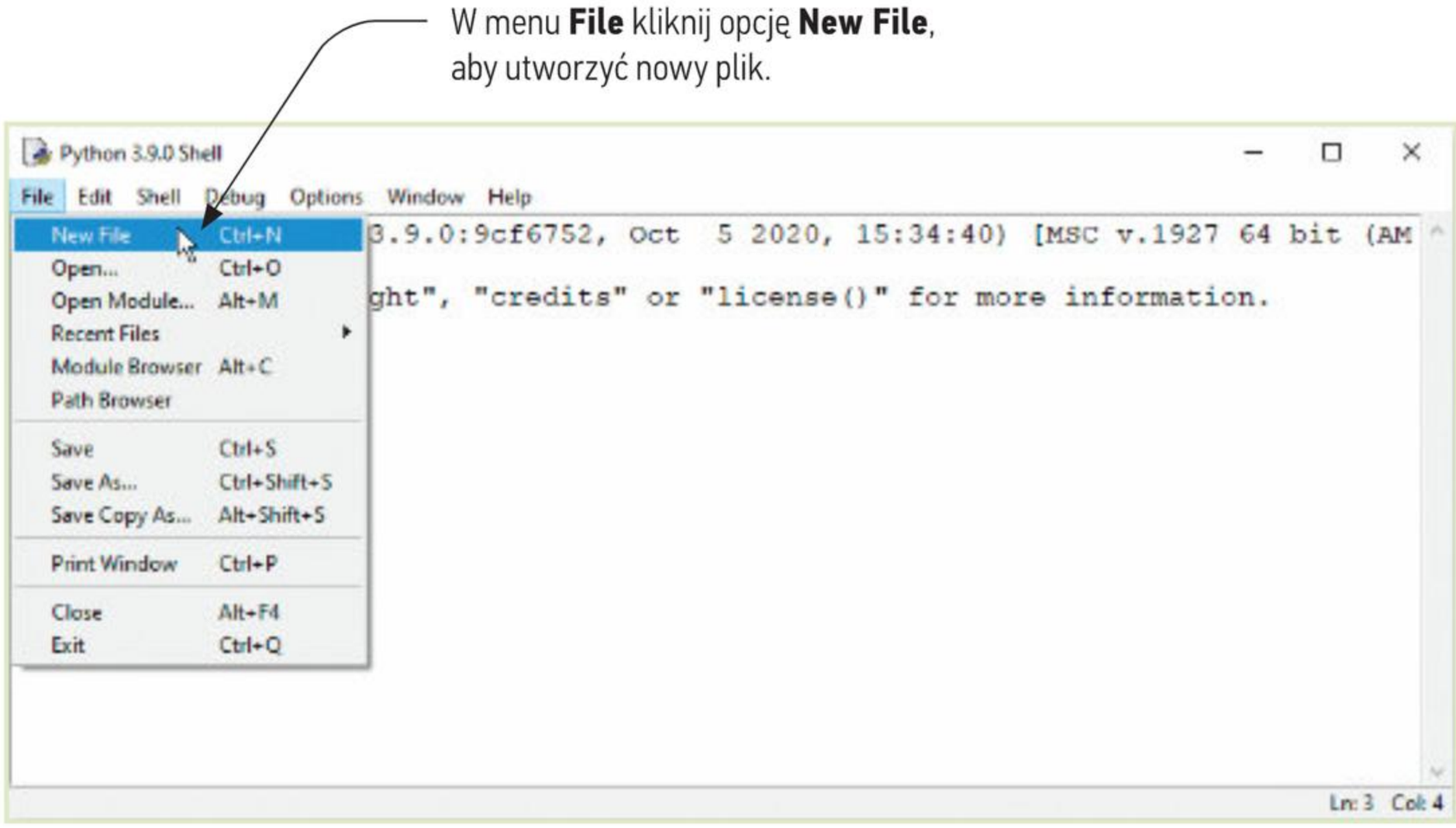
Do tworzenia programów w języku Python służy tryb skryptowy. Do trybu skryptowego możemy przejść z trybu interaktywnego, z okna powłoki Pythona.

Etapy tworzenia programu w języku Python

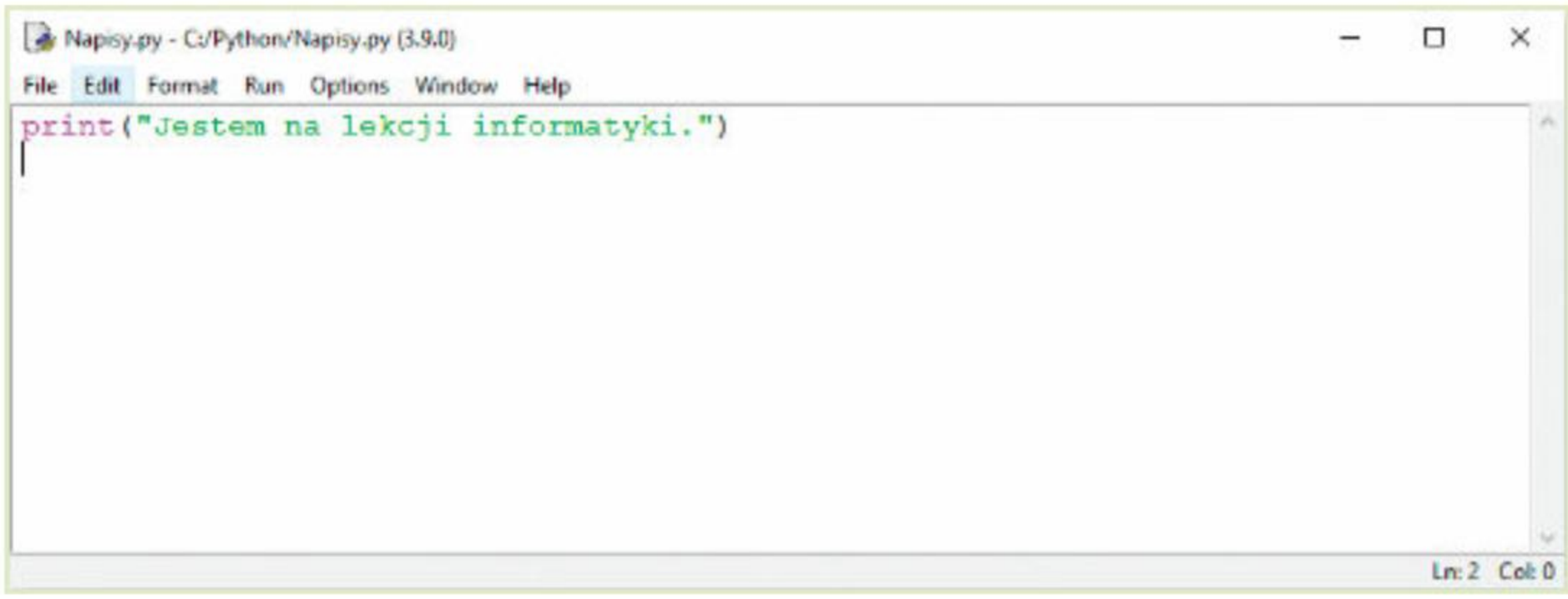
1. W oknie powłoki Pythona (rys. 6a) z menu **File** wybieramy opcję **New File** – otworzy się okno edytora kodu źródłowego.
2. W oknie edytora piszemy kod źródłowy programu (rys. 6b).
3. Zapisujemy program w pliku, wybierając w oknie edytora kodu źródłowego z menu **File** opcję **Save As**.
4. Uruchamiamy program, wybierając w oknie edytora z menu **Run** opcję **Run Module** (lub naciskając klawisz **F5**) – wynik działania programu pojawi się w oknie powłoki (rys. 6c).

Uwaga: Pliki programów w języku Python mają rozszerzenie *py*.

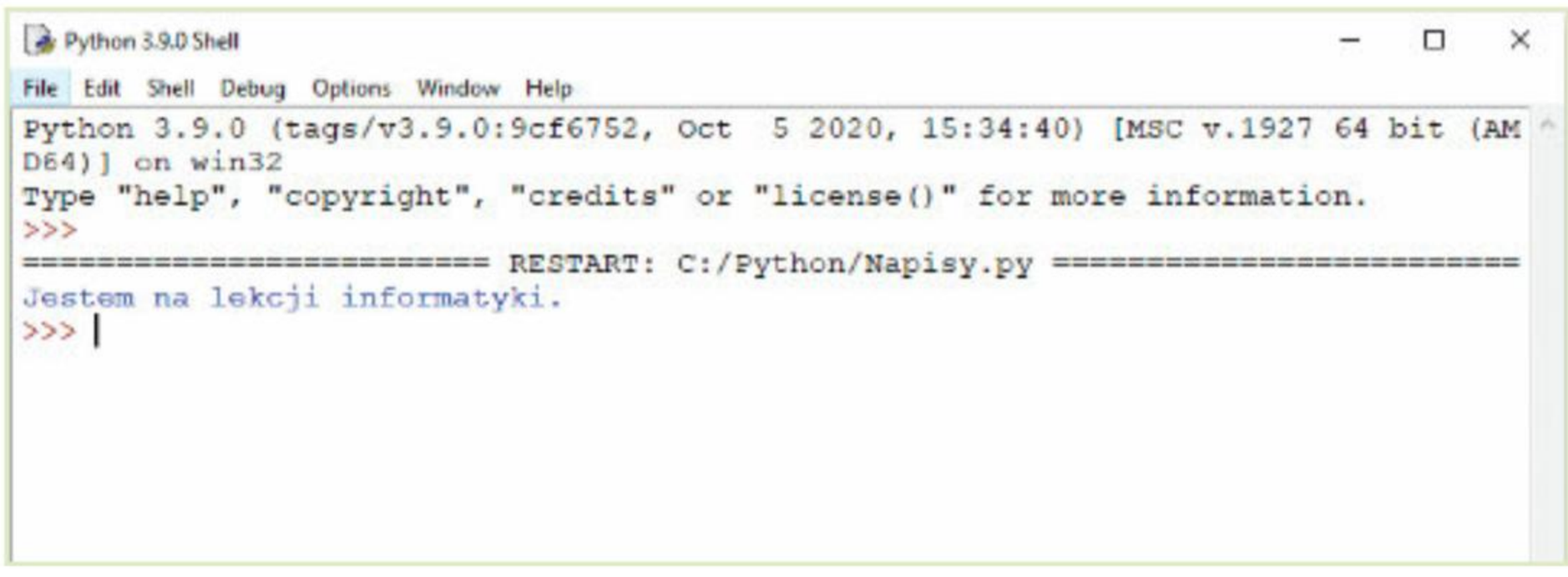
Jeśli argumentem jest wyrażenie, komputer najpierw je oblicza, a następnie wyświetla na ekranie obliczoną wartość. Tekst umieszczamy wewnątrz nawiasu w apostrofach (pojedynczych lub podwójnych). Po wyświetleniu komunikatu lub wyniku następuje przejście do nowego wiersza.



Rys. 6a. Tworzenie nowego pliku



Rys. 6b. Okno edytora kodu źródłowego środowiska IDLE



Rys. 6c. Okno powłoki Pythona z wynikiem działania programu (wyświetlonym napisem) – ćwiczenie 5. (punkt 1.)

Aby w języku Python wyświetlać na ekranie komunikaty i wyniki, stosujemy instrukcję wyjścia – funkcję `print()`. Argumentami funkcji (podanymi w nawiasach) mogą być teksty, wyrażenia arytmetyczne lub zmienne oddzielone przecinkami.

Przykłady	Wyniki
`print("Informatyka")`	komputer wyświetla na ekranie napis „Informatyka”
`print(23 + 17)`	komputer wyświetla na ekranie wartość wyrażenia 23 + 17, czyli 40
`print(wzrost)`	komputer wyświetla na ekranie wartość zmiennej `wzrost`
`print("Suma liczb 345 i 36 wynosi:", 345 + 36)`	komputer wyświetla na ekranie napis „Suma liczb 345 i 36 wynosi:” i w tym samym wierszu obliczoną wartość wyrażenia (czyli 381)

Tabela 2. Przykłady stosowania funkcji `print()`

Ćwiczenie 5. W trybie skryptowym piszemy program wyświetlający napisy na ekranie

1. Utwórz nowy plik. Napisz program wyświetlający na ekranie napis: „Jestem na lekcji informatyki.”
2. Zapisz program w pliku pod nazwą *Napisy*. Uruchom program.
3. Zmodyfikuj program tak, aby w drugim wierszu wyświetlało się zdanie „Pracuję w trybie skryptowym.”, a w trzecim – twoje imię.
4. Zapisz program pod tą samą nazwą i uruchom go ponownie.

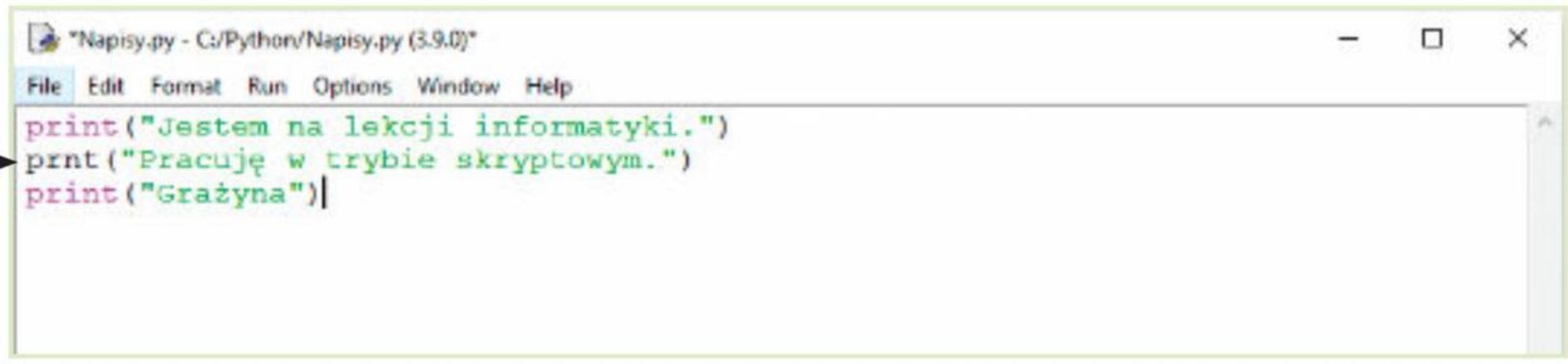
Interpreter tłumaczy program instrukcja po instrukcji. Co to oznacza? Na rysunku 7a pierwsza instrukcja jest poprawna, ale druga zawiera błąd. W takiej sytuacji (po zapisaniu i uruchomieniu programu) komputer wykona tylko pierwszą instrukcję programu, a drugiej i trzeciej – już nie. Zamiast wykonać błędną instrukcję, interpreter wyświetli odpowiedni komunikat (rys. 7b). Komunikaty interpretera mogą pojawiać się w oddzielnym oknie.

Interpreter wyświetla komunikat o pierwszym znalezionym błędzie. Należy poprawić wskazany błąd, ponownie zapisać program w pliku i uruchomić. Jeśli interpreter nie znajdzie już więcej błędów, komputer wykona cały program.

Warto zauważyć, że czcionka kodu programu ma różne kolory, np. w nazwie `print` jest fioletowa, ale w błędnym zapisie tej instrukcji – czarna (rys. 7a).

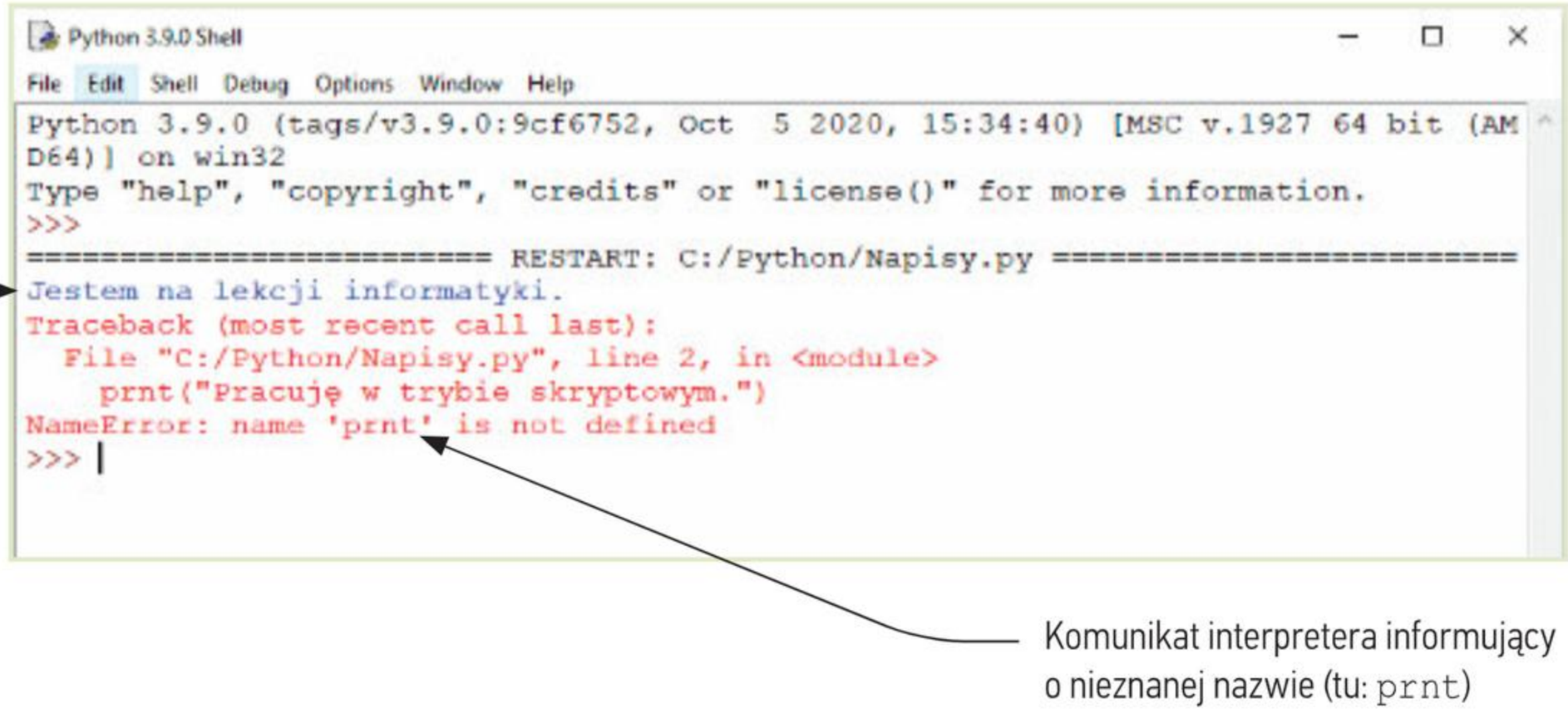
Napis wewnątrz nawiasów w instrukcji `print` jest zielony, ale po wyświetleniu na ekranie w oknie powłoki tekst jest niebieski. Komunikaty o błędach są koloru czerwonego (rys. 7b).

Błędnie zapisana instrukcja `prnt()`



Rys. 7a. Okno edytora kodu źródłowego z błędnie zapisaną instrukcją programu

Komputer wyświetli pierwszy napis.



Rys. 7b. Okno powłoki Pythona z komunikatem interpretera

Ćwiczenie 6. Modyfikujemy program

1. Zmodyfikuj program zapisany w ćwiczeniu 5. Dodaj jeszcze dwa inne napisy w dwóch kolejnych wierszach. Zapisz program pod nazwą *Kolejne_napisy*. Uruchom program.
2. Jeśli interpreter wykrył błędy, popraw je. Zapisz program i ponownie go uruchom.

Zapamiętaj

- Algorytm przedstawia krok po kroku sposób rozwiązania problemu.
- Rozwiązując dowolny problem, postępujemy według podobnego schematu. Określamy specyfikację zadania: dane wejściowe, poszukiwane wyniki oraz sposób rozwiązania.
- Poznaliśmy kilka sposobów przedstawiania algorytmów:
  - opis słowny – ogólny opis wykonywanych operacji,
  - lista kroków – każdy punkt listy zawiera opis wykonywanej czynności,
  - schemat blokowy (graficzna prezentacja algorytmu) – układ bloków i połączeń między nimi określa kolejność i sposób wykonywania kroków algorytmu,
  - program komputerowy – ciąg instrukcji języka programowania realizujący algorytm.
- Program komputerowy może występować w dwóch postaciach:
  - jako program (kod) źródłowy – program zrozumiały dla programisty, zapisany jako ciąg instrukcji tekstowych języka programowania wysokiego poziomu, np. języka Python,
  - jako program (kod) wynikowy (maszynowy) – program zrozumiały dla komputera, napisany w języku wewnętrznym komputera w postaci ciągu instrukcji dla procesora.
- Każdy język programowania ma swój zbiór instrukcji, reguły budowy zdań i właściwe słownictwo.
- Środowisko programistyczne Pythona (IDLE) udostępnia dwa tryby pracy:
  - Tryb interaktywny, w którym możemy napisać jedno polecenie i zatwierdzić je klawiszem **Enter**. Jeśli polecenie będzie poprawne, komputer je od razu wykona.
  - Tryb skryptowy, w którym możemy napisać program, zapisać go w pliku, a następnie uruchomić. Jeśli program będzie poprawny, komputer go wykona.

Serwis dla ciekawskich

Język Python zaczął powstawać w roku 1989 jako następca języka ABC.

Nazwa języka pochodzi od tytułu serialu „Latający cyrk Monty Pythona", który cieszył się dużą popularnością w Wielkiej Brytanii w latach 70. XX w. Był to ulubiony serial twórcy języka Python Holendra Guido van Rossuma (czyt: gido wan rosyma).

OKO W OKO Z MONITOREM

Pytania i polecenia

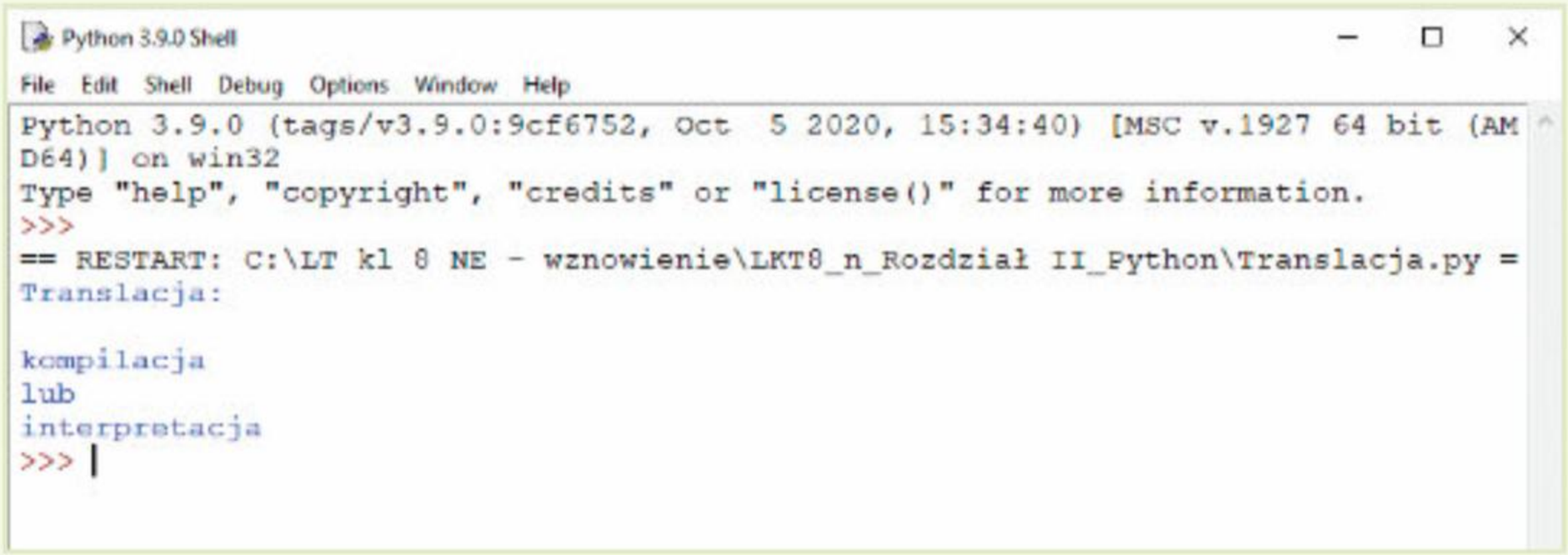
1. Co to jest algorytm?
2. Czym jest specyfikacja zadania?
3. Wymień kolejne etapy rozwiązywania problemów.
4. Czy przepis na kisiel ma cechy algorytmu informatycznego? Uzasadnij odpowiedź.
5. Wymień cztery sposoby przedstawiania algorytmów.
6. Na czym polega zapisywanie algorytmu w postaci listy kroków?
7. Jakiego bloku użyjesz w schemacie blokowym do wyprowadzenia wyników obliczeń? Uzasadnij odpowiedź.
8. Czy do każdej figury w schemacie blokowym musi dochodzić jedno połączenie? Uzasadnij odpowiedź.
9. W jaki sposób dodaje się dwa ułamki zwykłe o różnych mianownikach? Przygotuj słowny opis tego algorytmu.
10. Czym różni się program źródłowy od programu wynikowego?
11. Czym jest język programowania?
12. W jaki sposób tworzymy program w języku Python?

Zadania

1. Dane są dwie liczby: *a* i *b*. Przedstaw w postaci listy kroków i schematu blokowego algorytm obliczania ich sumy oraz wyprowadzania jej wartości.
2. Przedstaw w postaci listy kroków i schematu blokowego algorytm obliczania objętości prostopadłościanu o krawędziach podstawy *a* i *b* oraz wysokości *h*.
3. Przedstaw w postaci schematu blokowego algorytm obliczania objętości ostrosłupa o wysokości *h*. Podstawą ostrosłupa jest trójkąt prostokątny o przyprostokątnych *a* i *b*.
4. Przedstaw w postaci listy kroków algorytm gotowania budyniu.
5. Uruchom program Python. W trybie interaktywnym wpisz polecenia: `print(56+34)` i `print("56+34")`. Objaśnij otrzymane wyniki.
6. Napisz w języku Python program wyświetlający na ekranie w siedmiu wierszach nazwy dni tygodnia. Zapisz program w pliku pod nazwą *Tydzień*.

DLA ZAINTERESOWANYCH

1. Przedstaw w postaci schematu blokowego algorytm rozwiązania następującego zadania: *Znajdź cenę 1 m² pola o powierzchni 80 arów, za które zapłacono 56 000 zł.*
2. Napisz w języku Python program wyświetlający na ekranie napisy, jak pokazano na rysunku 8. Po napisie „Translacja:" powinien być umieszczony jeden pusty wiersz. Zapisz program w pliku pod nazwą *Translacja*.



Rys. 8. Wynik działania programu – zadanie 2. (dla zainteresowanych)

2.2. Piszemy programy w języku Python

Warto powtórzyć

1. Czym jest kod źródłowy programu?
2. Na czym polega interpretacja programu?
3. Jakie są etapy tworzenia programu w języku Python?
4. Które polecenie języka Python umożliwia wyświetlanie na ekranie komunikatów i wyników obliczeń?

W poprzednim temacie poznaliśmy środowisko programistyczne języka Python oraz napisaliśmy pierwszy program w tym języku. W tym temacie:

1. dowiesz się, jak stosować zmienne w programowaniu,
2. zastosujesz instrukcję warunkową do zapisywania algorytmów z warunkami,
3. zapiszesz wybrany algorytm iteracyjny z wykorzystaniem instrukcji iteracyjnej,
4. poznasz sposób definiowania i stosowania funkcji w języku Python,
5. dowiesz się, do czego służą listy w języku Python, i zastosujesz je w tworzonych programach.

1 Stosowanie zmiennych w programowaniu

Czym jest zmienna?

Zmienna w programie komputerowym to nazwana część pamięci operacyjnej komputera (RAM) o unikatowym adresie – **komórka (komórki) pamięci**. W każdym języku programowania musimy określić sposób korzystania ze zmiennych i nadawania im wartości.

Zmienna ma zawsze jakąś wartość i określone miejsce w pamięci komputera (adres komórki pamięci). Jeśli w tym samym programie zapiszemy w tej samej zmiennej nową wartość (wykonamy tzw. instrukcję przypisania), to poprzednia wartość znika.

Komórka pamięci
Część pamięci operacyjnej komputera (RAM) o unikatowym adresie. W komórkach pamięci przechowuje się dane reprezentujące instrukcje procesora lub dane dla tych instrukcji.

Na przykład po wykonaniu kolejno następujących instrukcji przypisania:

```
s = 1
s = s + 4
```

w zmiennej `s` będzie pamiętana wartość 5.

Warto zauważyć, że instrukcja przypisania `s = s + 4` nie jest równością prawdziwą w sensie matematycznym.

Zmienne w języku Python

W języku Python nie musimy wcześniej określać typu zmiennej. Interpreter określa typ zmiennej na podstawie rodzaju danych, jaki przypiszemy tej zmiennej. Nie ustanawia jednak typu zmiennej na zawsze, ponieważ przy kolejnym przypisaniu może się on zmienić. Na przykład jeśli na początku programu przypiszemy zmiennej `liczba` wartość całkowitą (34), to typ zmiennej `liczba` jest całkowity. Jeśli jednak w dalszej części programu przypiszemy jej wartość rzeczywistą (np. 45,8), to zmienna `liczba` będzie już typu rzeczywistego.

Każdy język programowania (język formalny), podobnie jak języki naturalne, którymi posługują się ludzie, posiada odpowiednie zasady składni oraz właściwe słownictwo. Nazwy zmiennych są elementami języka programowania, dlatego podlegają jego zasadom, a nie regułom języka polskiego. Z tego powodu w nazwach zmiennych mogą się pojawić m.in. znaki podkreślnika i cyfry, które nie występują w wyrazach zapisywanych w języku polskim.

W języku Python w nazwach zmiennych można używać tylko liter, znaku podkreślenia i cyfr. Nazwa nie może zaczynać się od cyfry. Przyjęte jest, że w nazwach zmiennych nie stosuje się liter z polskimi znakami diakrytycznymi. Należy nadawać nazwy określające znaczenie danej zmiennej. Małe i wielkie litery w nazwach zmiennych (także instrukcji) mają różne znaczenie, np. zmienne `Suma` i `suma` to dwie różne zmienne.

Zmiennej możemy w programie nadać konkretną wartość za pomocą instrukcji przypisania. Znak (operator) = oznacza przypisanie zmiennej zapisanej po lewej stronie znaku wartości umieszczonej po jego prawej stronie.

Przykład	Opis
`rok = 2021`	zmiennej `rok` przypisujemy wartość, która jest liczbą całkowitą, czyli zmienna `rok` jest typu całkowitego `(int)`
`wzrost = 159.5`	zmiennej `wzrost` przypisujemy wartość, która jest liczbą rzeczywistą, czyli zmienna `wzrost` jest typu rzeczywistego `(float)`
`nazwisko = "Kowalski"`	zmiennej `nazwisko` przypisujemy tekst (ciąg znaków), czyli zmienna `nazwisko` jest typu tekstowego `(str)`
`rok = rok + 1`	zmiennej `rok` przypisujemy wartość zmiennej `rok` zwiększoną o 1
`iloczyn = a * b`	zmiennej `iloczyn` przypisujemy iloczyn wartości pamiętanych w zmiennych `a` i `b`

Tabela 1. Przykłady przypisywania zmiennej wartości w języku Python

W obliczeniach z użyciem zmiennych używamy operatorów arytmetycznych zapisanych w tabeli 2.

Operator	Działanie	Przykład	Wynik
+	dodawanie	`23 + 56`	`79`
–	odejmowanie	`987 - 233`	`754`
*	mnożenie	`432 * 6`	`2592`
//	dzielenie całkowite (z obcięciem części ułamkowej)	`55 // 3`	`18`
/	dzielenie zmiennoprzecinkowe (z zachowaniem części ułamkowej)	`55 / 3`	`18.333333333333332`
%	obliczenie reszty z dzielenia dwóch liczb całkowitych	`37 % 4`	`1`

Tabela 2. Podstawowe operatory arytmetyczne w języku Python

Ćwiczenie 1. Nadajemy wartości zmiennym

1. W trybie interaktywnym sprawdź, w jaki sposób można przypisać wartość zmiennej. Umieść kolejno następujące polecenia (każde po znaku zachęty):

```
liczba = 23
liczba = liczba + 1
print(liczba)
reszta = liczba % 3
print(reszta)
```

2. Napisz jeszcze pięć instrukcji przypisania wartości (liczb lub wyrażeń) zmiennym według własnego pomysłu. Wykorzystaj operatory arytmetyczne z tabeli 2.

Zmiennej możemy nadać wartość za pomocą instrukcji przypisania, wprowadzając wartość z klawiatury po uruchomieniu programu. W tym celu stosujemy instrukcję wejścia – funkcję `input()`.

Argumentem funkcji (podanym w nawiasach) jest tekst (ciąg znaków) zapisany w apostrofach, który pojawi się na ekranie jako swego rodzaju „zaproszenie” do wpisania ciągu znaków z klawiatury. Jeśli użyjemy funkcji `input()` w instrukcji przypisania, wpisany przez użytkownika ciąg znaków komputer zapamięta w zmiennej podanej po lewej stronie:

```
nazwa_zmiennej = input("Zaproszenie do wprowadzenia danej z klawiatury: ")
```

Aby komputer mógł zapamiętać wprowadzoną za pomocą instrukcji `input()` daną jako liczbę, musimy dodać instrukcję, która zamieni ciąg znaków na liczbę: `int()` – w przypadku liczb całkowitych, `float()` – w przypadku liczb rzeczywistych.

W większości języków programowania (w tym w języku Python) część dziesiętną liczby zapisujemy po kropce (a nie po przecinku, jak w matematyce), np. 34.5, a nie 34,5.

Przykład	Opis
`nazwisko = input("Wprowadź nazwisko: ")`	jeśli wpiszemy z klawiatury nazwisko „Kowalski”, komputer w zmiennej `nazwisko` zapamięta ciąg znaków „Kowalski”
`a = input("Wprowadź liczbę: ")`	jeśli wpiszemy z klawiatury liczbę 346, komputer w zmiennej `a` zapamięta ciąg znaków „346”, a nie liczbę 346
`a = int(input("Wprowadź liczbę: "))`	jeśli wpiszemy z klawiatury liczbę 346, komputer w zmiennej `a` zapamięta liczbę całkowitą 346
`srednia = float(input("Podaj średnią ocen: "))`	jeśli wpiszemy z klawiatury liczbę 4.3, komputer w zmiennej `srednia` zapamięta 4,3

Tabela 3. Przykłady wprowadzania z klawiatury wartości zmiennej w języku Python

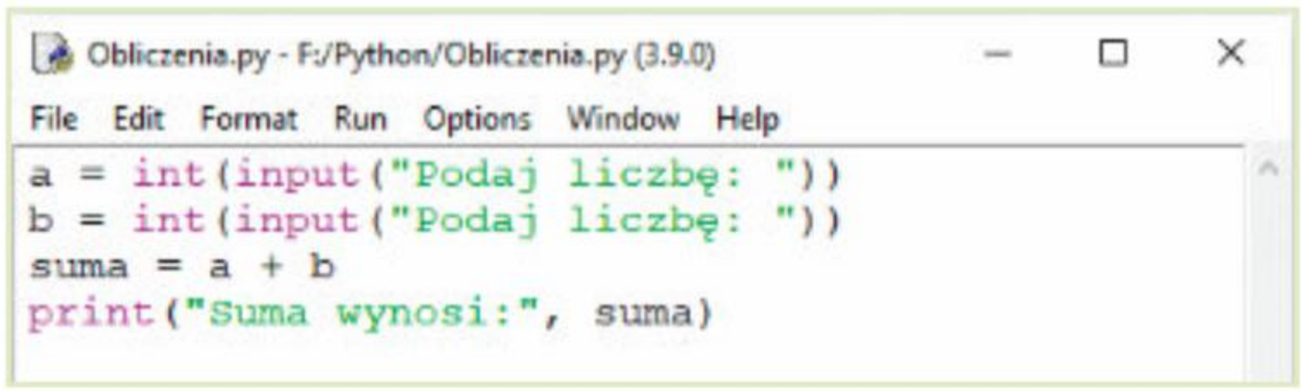
Ćwiczenie 2. Wprowadzamy wartości zmiennych z klawiatury

1. W trybie interaktywnym sprawdź użycie instrukcji `input()`. Umieść kolejno po znaku zachęty `>>>` następujące polecenia (w przypadku zmiennej `liczba` wprowadź z klawiatury za każdym razem tę samą liczbę, np. 34):

```
miasto = input("Wprowadź nazwę miasta: ")
print(miasto)
liczba = int(input("Wprowadź liczbę: "))
print(liczba + liczba)
liczba = input("Wprowadź liczbę: ")
print(liczba + liczba)
```

2. Wyjaśnij, dlaczego dla tej samej wartości zmiennej `liczba` komputer wyświetlił dwa różne wyniki po wykonaniu instrukcji `print(liczba + liczba)` z czwartego i szóstego wiersza.
3. Napisz jeszcze cztery wymyślone przez siebie instrukcje umożliwiające wprowadzenie wartości zmiennych (różnych typów). Wykorzystaj operatory arytmetyczne z tabeli 2.

W przypadku ciągów znaków operator + powoduje złączenie tych ciągów, np. gdy napiszemy instrukcję: `print("samo" + "chód")`, komputer wyświetli napis „samochód”.



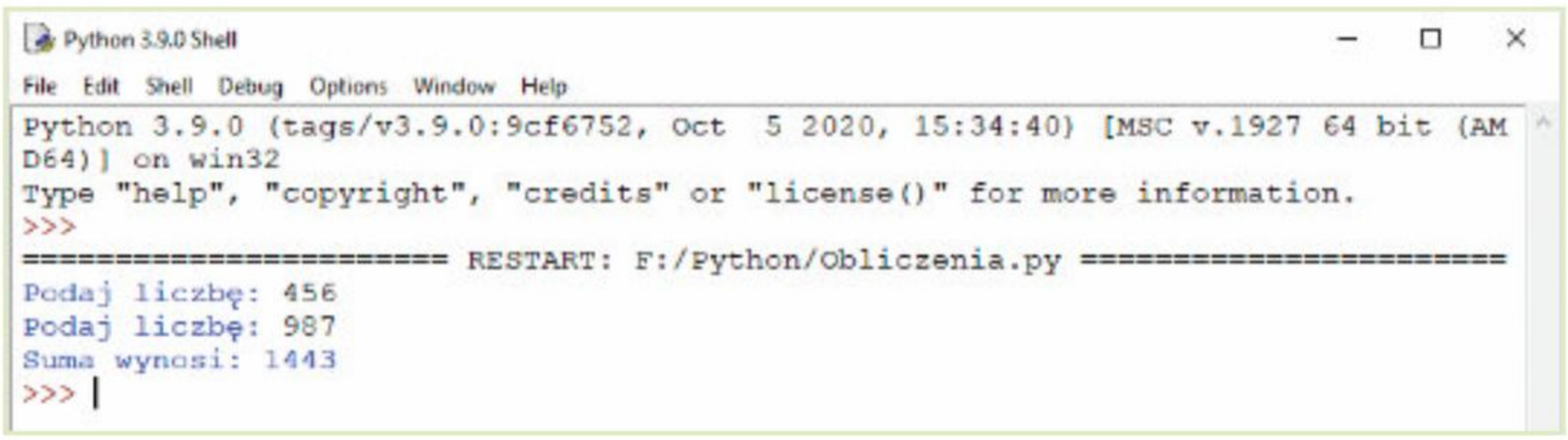
Rys. 1. Program w języku Python obliczający sumę dwóch liczb całkowitych wprowadzanych z klawiatury i wyświetlający wynik na ekranie – ćwiczenie 3. (punkt 1.)

Ćwiczenie 3. Piszemy program z użyciem zmiennych wprowadzanych z klawiatury

1. Program pokazany na rysunku 1. pozwala na wprowadzenie z klawiatury dwóch liczb całkowitych `a` i `b`, obliczenie ich sumy i wyświetlenie na ekranie wyniku (wartości zmiennej `suma`). Utwórz nowy plik źródłowy i przepisz kod tego programu.
2. Zapisz program w pliku pod nazwą *Obliczenia*. Uruchom program.
3. Dodaj do programu obliczanie iloczynu liczb `a` i `b` oraz wyświetlanie wyniku mnożenia na ekranie. Zapisz program w pliku pod tą samą nazwą. Uruchom program.

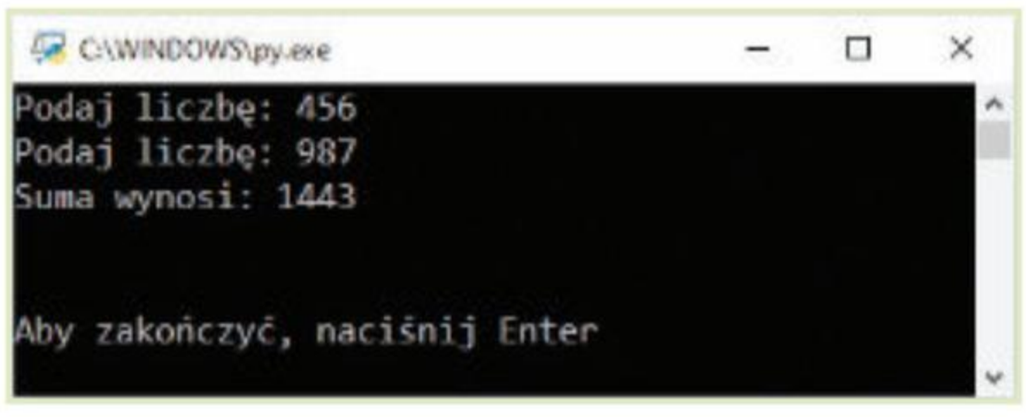
Wskazówka: Jeśli zamkniemy okno programu i chcemy ponownie uruchomić program, możemy w oknie powłoki wybrać z menu **File** opcję **Open**. Program otworzy się w oknie edytora kodu źródłowego.

Po uruchomieniu programu z wykorzystaniem opcji **Run Module** z menu **Run** w oknie edytora kodu źródłowego (rys. 1.) wynik działania programu pokaże się w oknie powłoki Pythona (rys. 2a).



Rys. 2a. Wynik działania programu obliczającego sumę dwóch liczb całkowitych wprowadzanych z klawiatury – ćwiczenie 3. (punkt 1.)

Program możemy również uruchomić, klikając dwukrotnie nazwę pliku z zapisanym programem w Eksploratorze plików – otworzy się okno, w którym zobaczymy wynik działania programu (rys. 2b). Aby uruchomić program w ten sposób, niezbędny jest interpreter Pythona.



Rys. 2b. Wynik działania programu obliczającego sumę dwóch liczb całkowitych wprowadzanych z klawiatury – ćwiczenie 3. (punkt 1.)

Aby okno pokazane na rysunku 2b nie zamknęło się automatycznie po wyświetleniu wyników obliczeń, możemy napisać na końcu programu instrukcję oczekiwania na naciśnięcie klawisza **Enter**: `input("\n\nAby zakończyć, naciśnij Enter")`. W tym przypadku wartości funkcji `input()` nie przypisujemy żadnej zmiennej.

Zapis `\n` odpowiada kodowi znaku przejścia do nowego wiersza. Umieszczenie zapisu `\n\n` oznacza wstawienie dwóch nowych wierszy.

2 Stosowanie instrukcji warunkowej do realizacji algorytmów z warunkami

Sytuacje warunkowe

Z sytuacjami warunkowymi spotykamy się w każdej dziedzinie wiedzy i życia codziennego. Na pytanie „Czy pada deszcz?" odpowiedź może brzmieć „tak" lub „nie". W zależności od tego, czy warunek jest spełniony czy nie, wybieramy sposób postępowania (rys. 3.).

Rys. 3. Przykład sytuacji warunkowej

Ćwiczenie 4. Podajemy przykłady sytuacji warunkowych
Podaj przykłady sytuacji warunkowych z życia codziennego.

Sytuacje warunkowe występują m.in. w matematyce i fizyce, gdy wykonanie obliczeń zależy od warunku, który muszą spełniać liczby (np. mają być nieujemne).

Z sytuacją warunkową spotykamy się wówczas, gdy wynik lub dalsze działanie zależy od spełnienia (lub niespełnienia) pewnego warunku. Algorytm, w którym występują sytuacje warunkowe, będziemy nazywać algorytmem z warunkami (inaczej: algorytmem z rozgałęzieniami).

Lista kroków i schemat blokowy algorytmu z warunkami

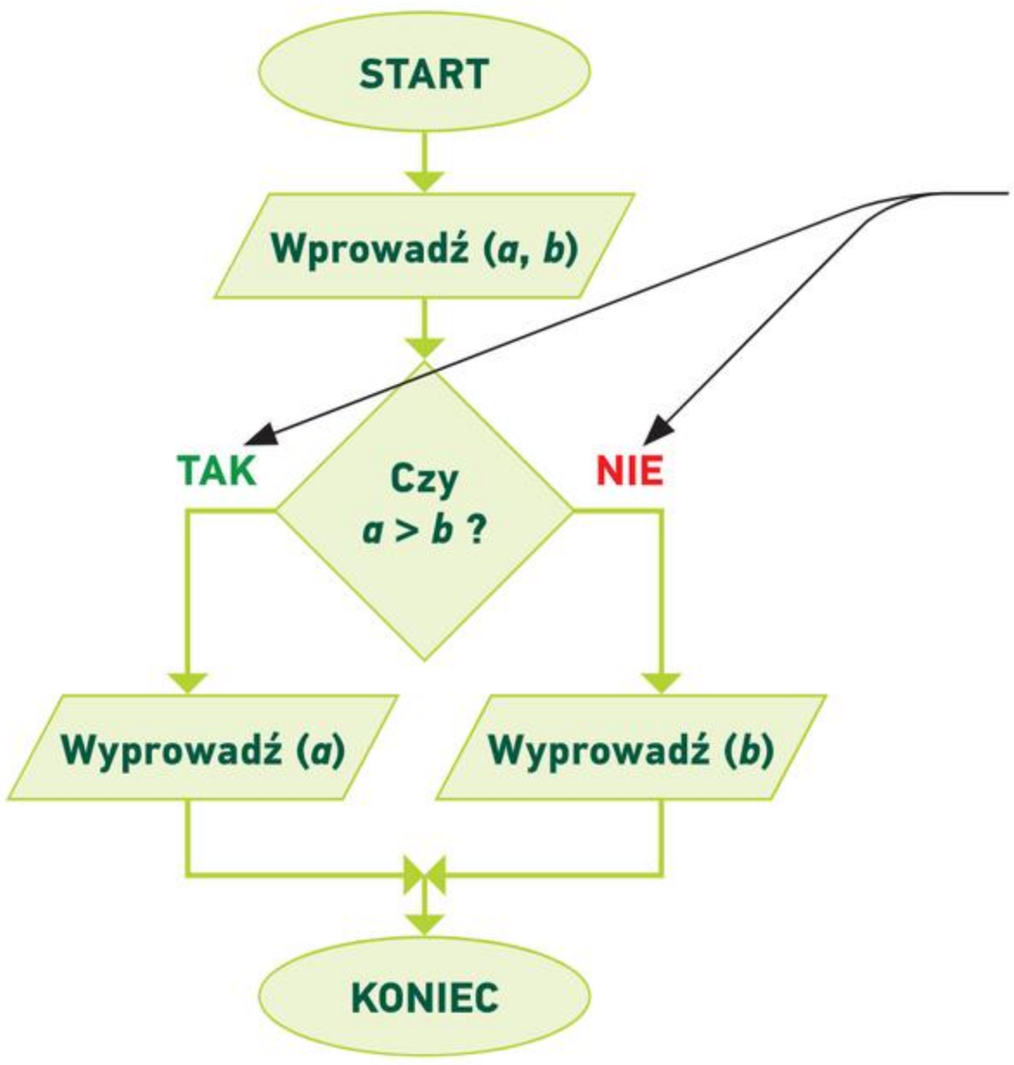
Zadanie: Sprawdź, która z dwóch nierównych sobie liczb całkowitych a i b jest większa, i wyprowadź większą z nich.
Specyfikacja zadania:
Dane: Liczby całkowite: a, b ($a \neq b$).
Wynik: Wartość większej z dwóch liczb.

Lista kroków:

1. Zacznij algorytm.
2. Wprowadź wartość liczby *a*.
3. Wprowadź wartość liczby *b*.
4. Sprawdź, czy liczba *a* jest większa od liczby *b*. Jeżeli tak, wyprowadź liczbę *a*, w przeciwnym przypadku wyprowadź liczbę *b*.
5. Zakończ algorytm.

W schemacie blokowym sytuację warunkową przedstawiamy za pomocą bloku warunkowego. W bloku tym wpisujemy warunek logiczny, stosując operatory porównań: = (równy), ≠ (różny), < (mniejszy), > (większy), <= (mniejszy lub równy), >= (większy lub równy), np. $x >= 0$, $n \neq 10$.

Można wpisywać również warunki złożone, połączone spójnikami (**i**, **lub**), np. $a < 0$ **lub** $a = 10$, $x > 0$ **i** $x < 20$.



Z bloku warunkowego wychodzą dwa połączenia: TAK i NIE. W zależności od tego, czy warunek jest spełniony czy nie, komputer wybierze odpowiednie odgałęzienie algorytmu.

Rys. 4. Schemat blokowy algorytmu z warunkiem – ćwiczenie 5.

Ćwiczenie 5. Analizujemy schemat blokowy algorytmu z warunkiem

1. Wskaż elementy związane z kolejnymi etapami rozwiązywania zadania: wprowadzanie danych, rozwiązanie zadania (wykonanie różnych operacji) i wyprowadzenie wyników.
2. Jakie rodzaje bloków zastosowano do określenia poszczególnych operacji? Zwróć uwagę na połączenia między blokami. Czy zawsze są takie same? Czy zawsze z jednego bloku wychodzi jedno połączenie? Zwróć uwagę na rozgałęzienie w schemacie blokowym.
3. Przetestuj działanie algorytmu dla różnych danych.

Zapisanie algorytmu z warunkami w języku Python

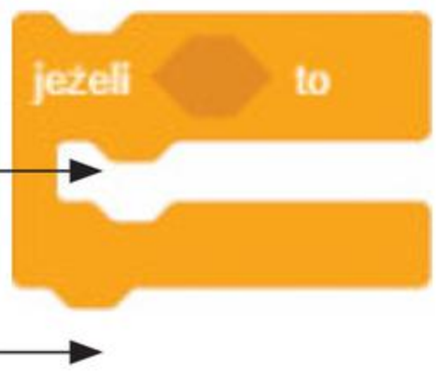
Aby w języku programowania napisać program realizujący algorytm z warunkami, stosujemy instrukcję warunkową.

W języku Python instrukcja warunkowa działa tak samo jak w języku Scratch i również występuje w wersjach pełnej i uproszczonej (rys. 5a, 5b). Jako *lista_instrukcji1* i *lista_instrukcji2* może wystąpić jedna instrukcja (w tym instrukcja warunkowa) lub więcej instrukcji. Po warunku i słowie kluczowym `else` umieszczamy dwukropek.

```
if warunek:
    lista_instrukcji1
else:
    lista_instrukcji2
kolejna_instrukcja
```

Rys. 5a. Ogólna postać instrukcji warunkowej w języku Python – po prawej odpowiadający jej element w języku Scratch

```
if warunek:
    lista_instrukcji1
kolejna_instrukcja
```



Rys. 5b. Uproszczona postać instrukcji warunkowej w języku Python – po prawej odpowiadający jej element w języku Scratch

Komputer sprawdza warunek logiczny – jeśli *warunek* jest spełniony (jest prawdziwy), wykonuje instrukcje (*lista_instrukcji1*). Jeśli *warunek* nie jest spełniony (jest fałszywy) – wykonuje instrukcje (*lista_instrukcji2*) po słowie `else` (rys. 5a), a jeśli brak tej części instrukcji – wykonuje od razu kolejną instrukcję (*kolejna_instrukcja*) po instrukcji warunkowej (rys. 5b).

W języku Python ważne są wcięcia w programie – *listę_instrukcji1* i *listę_instrukcji2* należy przesunąć w prawo przynajmniej o jedną spację. Przyjęte jest wcięcie składające się z czterech spacji. Jeśli te listy obejmują więcej instrukcji, wszystkie należy tak przesunąć. Jest to sposób wyróżnienia bloku kodu. Jeśli nie zastosujemy wcięcia, interpreter pokaże błąd.

Operatory		Określenie	Przykład wyrażenia	Interpretacja wyrażenia
Porównania	==	równy	`a == b`	`a` równe `b`
	!=	różny	`a != b`	`a` różne od `b`
	<	mniejszy	`a < b`	`a` mniejsze od `b`
	>	większy	`a > b`	`a` większe od `b`
	>=	większy lub równy	`a >= b`	`a` większe lub równe `b`
	<=	mniejszy lub równy	`a <= b`	`a` mniejsze lub równe `b`
Logiczne	or	alternatywa logiczna (lub)	`a < -5 or a > 5`	`a` mniejsze od –5 lub `a` większe od 5
	and	koniunkcja logiczna (i)	`a > 0 and a < 10`	`a` większe od 0 i `a` mniejsze od 10
	not	negacja logiczna	`not a == 5`	`a` nierówne 5

Tabela 4. Podstawowe operatory porównania i logiczne w języku Python

Większa_z_dwóch.py - C:/Python/Większa_z_dwóch.py (3.9.0)

File Edit Format Run Options Window Help

```
a = int(input("Podaj pierwszą liczbę: "))
b = int(input("Podaj drugą liczbę różną od pierwszej: "))
if a > b:
    print("Większa jest pierwsza liczba", a)
else:
    print("Większa jest druga liczba", b)


input("\n\nAby zakończyć, naciśnij Enter")
```

Rys. 6. Program w języku Python realizujący algorytm z warunkiem – ćwiczenie 6.

Ćwiczenie 6. Stosujemy instrukcję warunkową

1. Program pokazany na rysunku 6. realizuje algorytm sprawdzenia, która z dwóch różnych liczb wprowadzonych z klawiatury jest większa, i wyprowadza większą z nich. Przepisz kod pokazany na rysunku 6. Wyjaśnij znaczenie poszczególnych wierszy programu.
2. Zapisz program w pliku pod nazwą *Większa_z_dwóch*. Uruchom program.

Ćwiczenie 7. Piszemy program realizujący algorytm z warunkiem

1. Napisz program sprawdzający, czy wprowadzona z klawiatury liczba całkowita jest dodatnia. Jeśli jest dodatnia, program powinien wyświetlić komunikat „Liczba dodatnia”, jeśli nie – „Liczba niedodatnia”.
2. Zapisz program w pliku pod nazwą *Liczba*. Uruchom program.

3 Stosowanie instrukcji iteracyjnej

Czym jest iteracja?

Czasami trzeba wykonać te same operacje na wielu danych. W takich przypadkach nie musimy powtarzać opisywania takich samych działań lub rysować takich samych bloków. Wystarczy wykorzystać iterację.

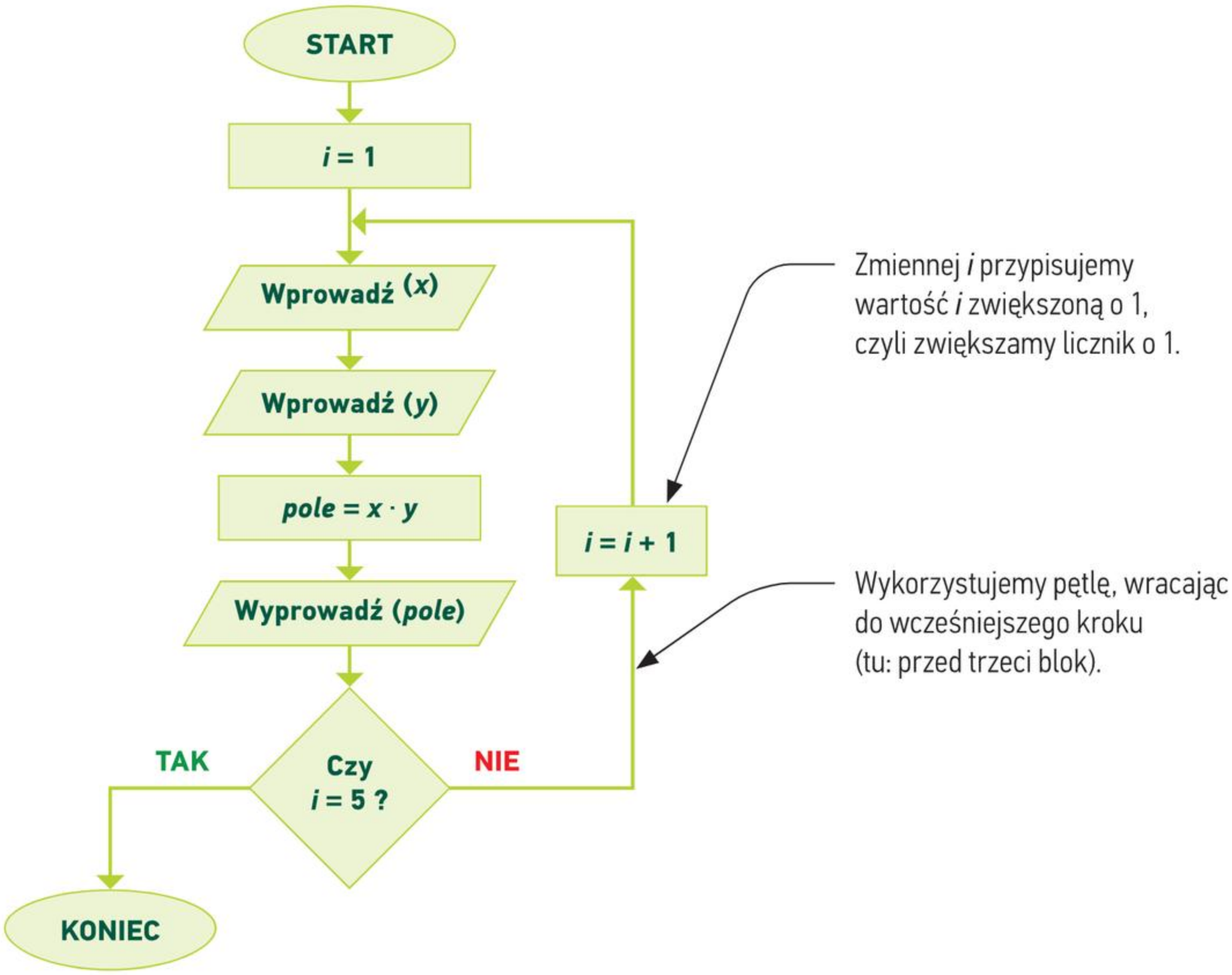
Iteracja polega na powtarzaniu tej samej operacji (ciągu operacji).
Iterację implementujemy (piszemy kod źródłowy), stosując tzw. pętlę. Z pętlą mamy do czynienia, gdy w pewnym kroku algorytmu wracamy do jednego z wcześniejszych kroków, co powoduje, że kroki te komputer wykona wiele razy.

Lista kroków i schemat blokowy algorytmu iteracyjnego

Zadanie: Policz pola pięciu prostokątnych działek o bokach x i y.
Specyfikacja zadania:
Dane: Liczby naturalne oznaczające długości boków poszczególnych działek: x, y ($x > 0$ i $y > 0$).
Wynik: Wartości pól poszczególnych działek.
Lista kroków:
1. Zacznij algorytm.
2. Zmiennej *i* przypisz wartość 1.
3. Wprowadź długość boku x.
4. Wprowadź długość boku y.
5. Zmiennej *pole* przypisz wartość wyrażenia: $x \cdot y$.
6. Wyprowadź wynik: *pole*.
7. Jeśli *i* nie równa się 5, zwiększ licznik o jeden ($i = i + 1$) i wróć do kroku 3.
8. Zakończ algorytm.

Schemat blokowy na rysunku 7. jest prezentacją graficzną algorytmu obliczenia pól pięciu prostokątnych działek. W schemacie zastosowano zmienną *i* (licznik), której przypisano wartość początkową 1.

W schemacie zastosowano blok warunkowy, w którym określono warunek zakończenia iteracji.



Rys. 7. Schemat blokowy algorytmu iteracyjnego – ćwiczenie 6.

Ćwiczenie 8. Analizujemy schemat blokowy algorytmu iteracyjnego

1. Przetestuj działanie algorytmu przedstawionego na rysunku 7. dla pięciu par różnych wartości zmiennych x i y.
2. Odpowiedz na pytania:
   a. Które operacje się powtarzają?
   b. Ile razy będzie realizowana pętla?
   c. W jaki sposób zliczamy iteracje (powtórzenia) w tym algorytmie?
   d. Kiedy kończą się działania w pętli?

Aby w języku programowania napisać program realizujący algorytm iteracyjny, korzystamy z instrukcji iteracyjnych (instrukcji pętli). W języku Python jedną z takich instrukcji jest instrukcja `for`.

Zapisanie algorytmu iteracyjnego w języku Python

```
for zmienna in lista_wartości:
    lista_instrukcji
```

Rys. 8. Ogólna postać instrukcji iteracyjnej `for` w języku Python

Jako *lista_instrukcji* może wystąpić pojedyncza instrukcja (w tym instrukcja pętli) lub więcej instrukcji (blok instrukcji). *Lista_instrukcji* musi być przesunięta w prawo przynajmniej o jedną spację. Przyjęte jest wcięcie składające się z czterech spacji.

W języku Python liczbę iteracji określa długość *listy_wartości* po słowie `in` (rys. 8.). Komputer wykona *listę_instrukcji* dla wszystkich wartości z *listy_wartości*. *Listę_wartości* możemy zapisać w różny sposób.

Przykład	Opis
`for i in [0, 1, 2, 3, 4]:` `    print(i)`	oznacza, że komputer wykona instrukcję `print(i)` dla listy wartości [0, 1, 2, 3, 4], czyli pięć razy, a zmienna `i` będzie przyjmować kolejne wartości z tej listy, czyli: 0, 1, 2, 3, 4
`for i in [1, 3, 5, 7]:` `    print(i)`	oznacza, że komputer wykona instrukcję `print(i)` dla listy wartości [1, 3, 5, 7], czyli cztery razy, a zmienna `i` będzie przyjmować kolejne wartości z tej listy, czyli: 1, 3, 5, 7

Tabela 5. Przykłady stosowania instrukcji `for` w języku Python

Nie zawsze chcemy wpisywać wszystkie wartości listy, zwłaszcza dla dużej liczby iteracji. Do utworzenia *listy_wartości* można użyć funkcji `range()`, która tworzy sekwencję wartości całkowitych. Argumentami funkcji `range()`mogą być konkretne wartości lub zmienne.

Przykład	Opis
Dla jednego argumentu: `range(koniec)`	
`for i in range(5)` `    print(i)`	oznacza, że komputer wykona instrukcję `print(i)` pięć razy; funkcja `range(koniec)` wygeneruje kolejne liczby całkowite z przedziału ⟨0, `koniec`⟩, czyli zmienna `i` będzie przyjmować kolejno wartości: 0, 1, 2, 3, 4
`for i in range(n)` `    print(i)`	oznacza, że komputer wykona instrukcję `print(i)` `n` razy dla wartości zmiennej `n` wprowadzanej z klawiatury; funkcja `range()` wygeneruje kolejne liczby całkowite z przedziału ⟨0, `n`), czyli zmienna `i` będzie przyjmować kolejno wartości: 0, 1, 2, ..., `n` – 1

Dla dwóch argumentów: `range(początek, koniec)`	
`for i in range(15, 55)` `  print(i)`	oznacza, że komputer wykona instrukcję `print(i)` czterdzieści razy; funkcja `range()` wygeneruje kolejne liczby całkowite z przedziału ⟨`początek, koniec`), czyli zmienna `i` będzie przyjmować kolejno wartości: 15, 16, 17, ..., 54 (czterdzieści kolejnych liczb naturalnych)
Dla trzech argumentów: `range(początek, koniec, krok)`	
`for i in range(1, 20, 2)` `  print(i)`	oznacza, że komputer wykona instrukcję `print(i)` dziesięć razy; funkcja `range()` wygeneruje kolejne liczby całkowite z przedziału ⟨`początek, koniec`), zmieniające się o `krok`, czyli zmienna `i` będzie przyjmować kolejno wartości: 1, 3, 5, ..., 19; trzeci argument określa tym samym, o jaką wartość zmienia się zmienna `i`

Tabela 6. Przykłady stosowania instrukcji `for` z wykorzystaniem funkcji `range()` w języku Python

Pola_działek.py - C:\Python\Pola_działek.py (3.9.0)

File Edit Format Run Options Window Help

```
for i in range(5):
    x = int(input("Podaj długość pierwszego boku: "))
    y = int(input("Podaj długość drugiego boku: "))
    pole = x * y
    print("Pole działki o boku", x, "i boku", y, "wynosi:", pole)

input("\n\nAby zakończyć, naciśnij Enter")
```

Rys. 9. Program w języku Python realizujący algorytm obliczania pól pięciu prostokątów – ćwiczenie 9.

Ćwiczenie 9. Stosujemy instrukcję iteracyjną `for`

1. Program pokazany na rysunku 9. realizuje algorytm obliczania pól pięciu prostokątów, których wymiary mają być wprowadzane z klawiatury. Przepisz kod programu. Wyjaśnij znaczenie poszczególnych wierszy programu.
2. Zapisz program w pliku pod nazwą *Pola_działek*. Uruchom program.
3. Odpowiedz na pytania:
   a. Ile razy komputer wykona instrukcję `print`?
   b. Jaka jest wartość zmiennej `i` na koniec programu?

W realizacji algorytmów iteracyjnych ważne jest prawidłowe określenie sposobu zakończenia działań. Niepoprawne określenie tych warunków może spowodować, że obliczenia nigdy się nie zakończą, czyli nastąpi „zapętlenie" algorytmu.

4 Funkcje

Kroki, które powtarzają się w algorytmie, można potraktować jako problem cząstkowy i przedstawić w postaci **podprogramu** (funkcji). Umożliwi to opracowanie każdego problemu cząstkowego oddzielnie.

Podprogram (funkcja)
Wyodrębniona część programu, mająca jednoznaczną nazwę i ustalony sposób wymiany danych z innymi częściami programu. Realizujący określone zadanie podprogram można wielokrotnie wykorzystywać w programie.

W języku Python podprogramy nazywamy funkcjami. Poznaliśmy kilka wbudowanych funkcji Pythona, m.in.: `print()`, `input()`. Podobnie jak w innych językach programowania w języku Python można definiować własne funkcje.

Aby napisać cały program, wystarczy utworzyć funkcje realizujące określone elementy programu i odpowiednio je połączyć. Połączenia podprogramu z programem głównym realizuje się poprzez instrukcję wywołania podprogramu.

Funkcje mogą zwracać wartość do programu głównego (rys. 10a) lub nic nie zwracać (rys. 10b).

W definicji funkcji określamy parametry formalne (zwane też parametrami), które w momencie wywołania funkcji komputer zastąpi parametrami aktualnymi (zwanymi też argumentami).

```
def nazwa_funkcji(lista_parametrów):
    lista_instrukcji
    return wartość
```

Rys. 10a. Ogólna postać definicji funkcji zwracającej wartość w języku Python

```
def nazwa_funkcji(lista_parametrów):
    lista_instrukcji
```

Rys. 10b. Ogólna postać definicji funkcji niezwracającej wartości w języku Python

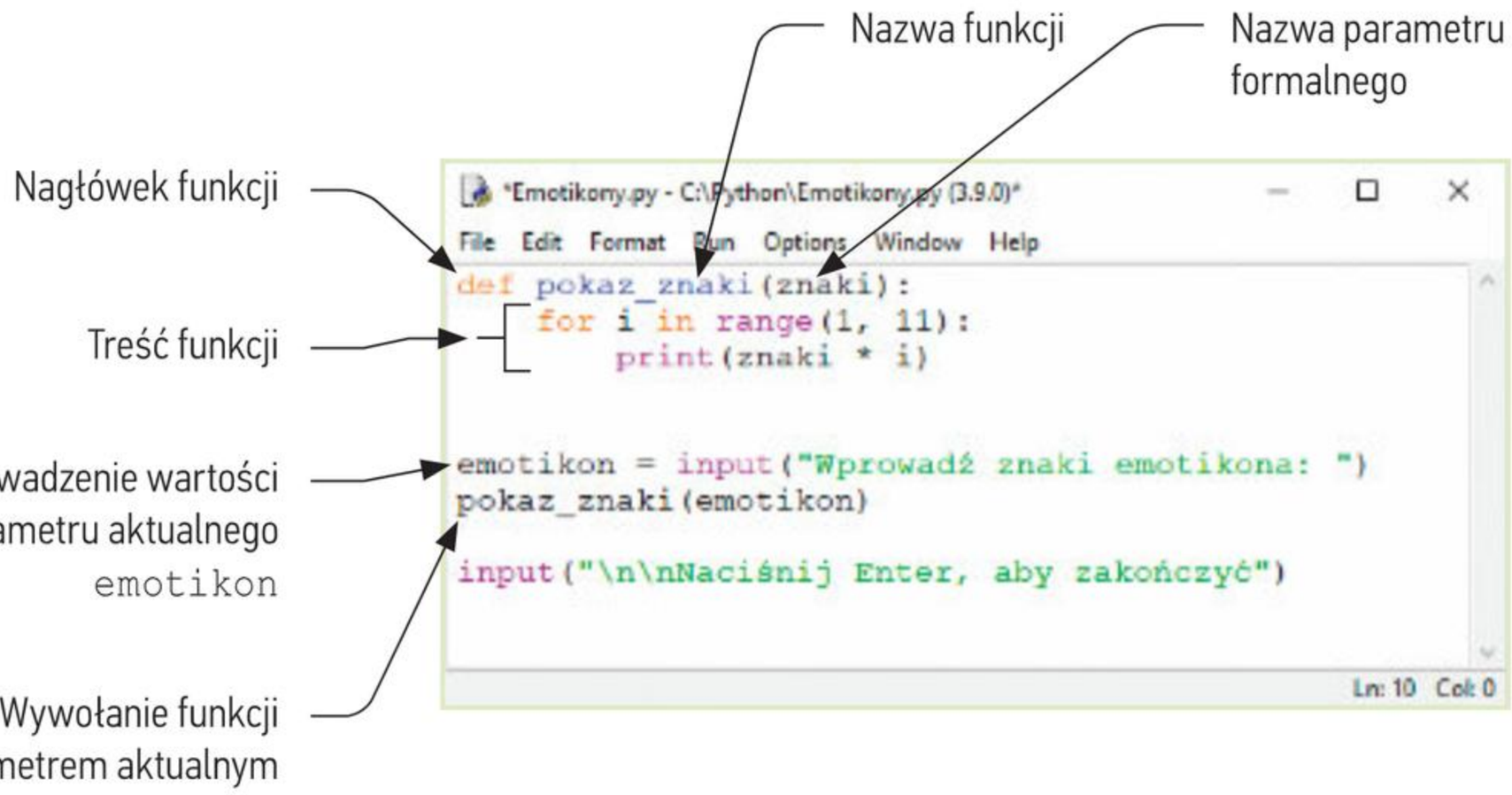
Lista_instrukcji i słowo `return` muszą być przesunięte w prawo przynajmniej o jedną spację (przyjęte jest wcięcie składające się z czterech spacji).

W języku Python nie określamy typu parametru ani typu zwracanej wartości. Podobnie jak w przypadku zmiennych interpreter automatycznie rozpoznaje typ parametru oraz zwracanej wartości. W nazwach funkcji i ich parametrów można używać tylko liter, znaków podkreślenia i cyfr. Nazwa nie może zaczynać się od cyfry.

Przyjęte jest, że w nazwach funkcji i jej parametrów nie stosuje się liter z polskimi znakami diakrytycznymi (podobnie jak w przypadku nazw zmiennych). Należy stosować nazwy opisujące działanie danej funkcji.

Funkcji niezwracającej wartości możemy użyć na przykład wtedy, gdy mamy wyświetlić dane na ekranie. Na rysunku 11. pokazano przykład funkcji wyświetlającej na ekranie znaki emotikona. W treści funkcji nie umieszczamy instrukcji `return`.

Aby wywołać funkcję niezwracającą wartości, należy umieścić nazwę funkcji (z parametrami formalnymi) w odpowiednim miejscu programu głównego (rys. 11.). W przypadku braku parametrów nawiasy po nazwie funkcji pozostawiamy puste.



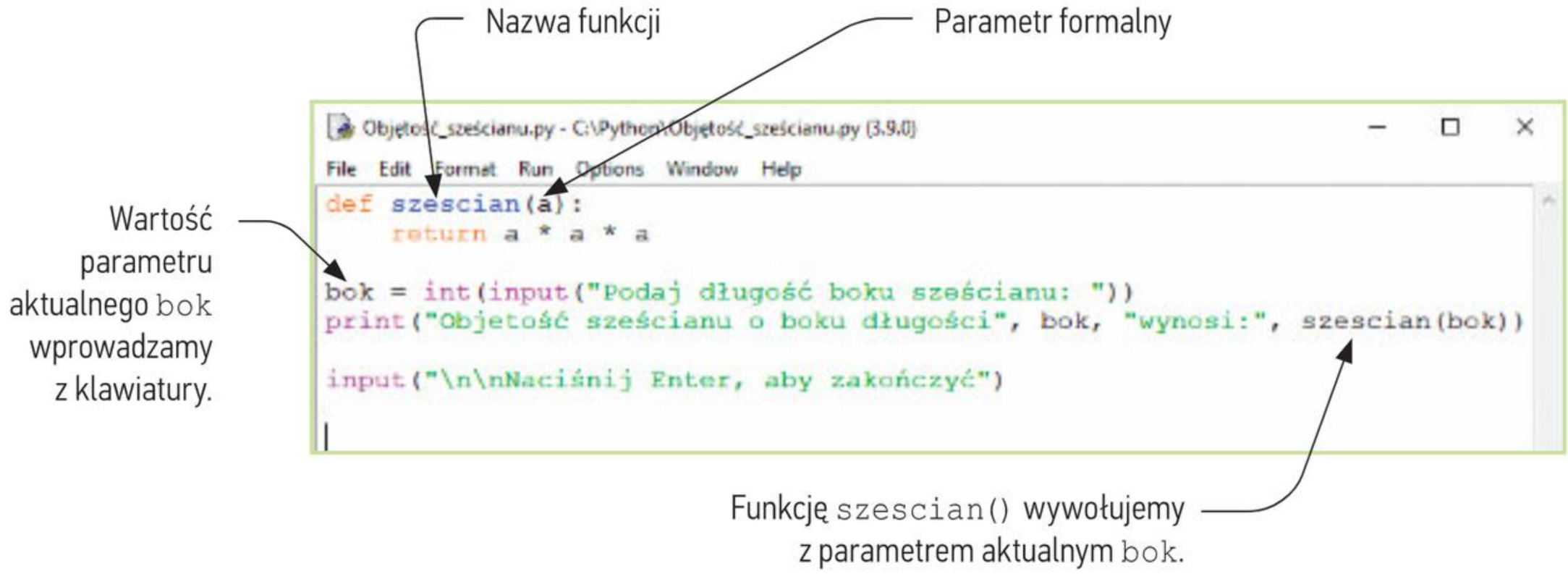
Rys. 11. Program w języku Python, w którym zdefiniowano funkcję niezwracającą wartości z jednym parametrem – ćwiczenie 10.

Ćwiczenie 10. Definiujemy w języku Python funkcję niezwracającą wartości

1. Przepisz program pokazany na rysunku 11.
2. Zapisz program w pliku pod nazwą *Emotikony*. Uruchom program. Objaśnij jego działanie.
3. Dodaj do funkcji drugi parametr (`liczba_em`), który będzie określał długość listy wartości w funkcji `range()`.
4. Zapisz program w pliku pod tą samą nazwą. Uruchom program.

Jeśli celem funkcji jest obliczenie i zwrócenie pewnej wartości do programu, stosujemy funkcję zwracającą wartość. Funkcja musi zawierać instrukcję `return` z wartością zwracaną do programu głównego.

Sposób definiowania takiej funkcji pokażemy na przykładzie funkcji z jednym parametrem o nazwie *a* obliczającej objętość sześcianu o boku *a* i zwracającej do programu głównego wynik obliczenia (rys. 12.).



Rys. 12. Program w języku Python, w którym zdefiniowano funkcję zwracającą wartość z jednym parametrem – ćwiczenie 11.

Wynik działania funkcji można również przypisać zmiennej, np. `objetosc = szescian(bok)`.

Ćwiczenie 11. Definiujemy w języku Python funkcję zwracającą wartość

1. Zdefiniuj funkcję `szescian()` z parametrem *a* typu całkowitego obliczającą objętość sześcianu i zwracającą do programu głównego wynik obliczeń (rys. 12.).
2. Funkcję `szescian()` wywołaj w programie głównym z parametrem aktualnym bok.
3. Zapisz program w pliku pod nazwą *Objętość_sześcianu*. Uruchom program.
4. Sprawdź działanie programu dla kilku różnych danych.

5 Listy

Niektóre algorytmy wymagają pamiętania wszystkich elementów zbioru zarówno w trakcie działania programu, jak i po jego zakończeniu. Przykładem jest algorytm porządkowania, ponieważ po jego zakończeniu musimy otrzymać wszystkie elementy w określonym porządku. Aby zapamiętać wszystkie dane, musimy je różnie nazwać. Moglibyśmy użyć liter alfabetu, ale w przypadku dużej liczby danych zabrakłoby liter. Dlatego wymyślono tzw. zmienne indeksowane, w których do nazwy zmiennej (np. *a*) dodaje się kolejny numer (indeks), np.: a_0, a_1, …, a_n. Wówczas dla każdej zmiennej komputer rezerwuje oddzielną komórkę pamięci (rys. 13b).

Rys. 13a. Zapisywanie w pamięci komputera wartości danych wprowadzanych z klawiatury z użyciem jednej zmiennej

Rys. 13b. Zapisywanie w pamięci komputera wartości danych wprowadzanych z klawiatury z użyciem pięciu zmiennych indeksowanych

Aby utworzyć zmienne indeksowane w języku Python, możemy zdefiniować specjalną strukturę danych – listę (rys. 14.).

```
nazwa_listy = [element1, element2, …, elementn]
```

Rys. 14. Ogólna postać definiowania listy w języku Python

Do elementów listy odwołujemy się, podając nazwę listy i indeks elementu umieszczony w nawiasach kwadratowych, np. $a[0]$, $a[1]$, …, $a[n-1]$ – dla listy n-elementowej o nazwie a. W języku Python indeks pierwszego elementu listy jest zawsze równy 0.

Lista (w odróżnieniu od tablicy w języku C++) może zawierać elementy różnego typu, dlatego w języku Python nie określamy typu elementów listy. Zarówno typ zmiennych przechowywanych w liście, jak i liczba elementów listy mogą się zmienić w trakcie działania programu.

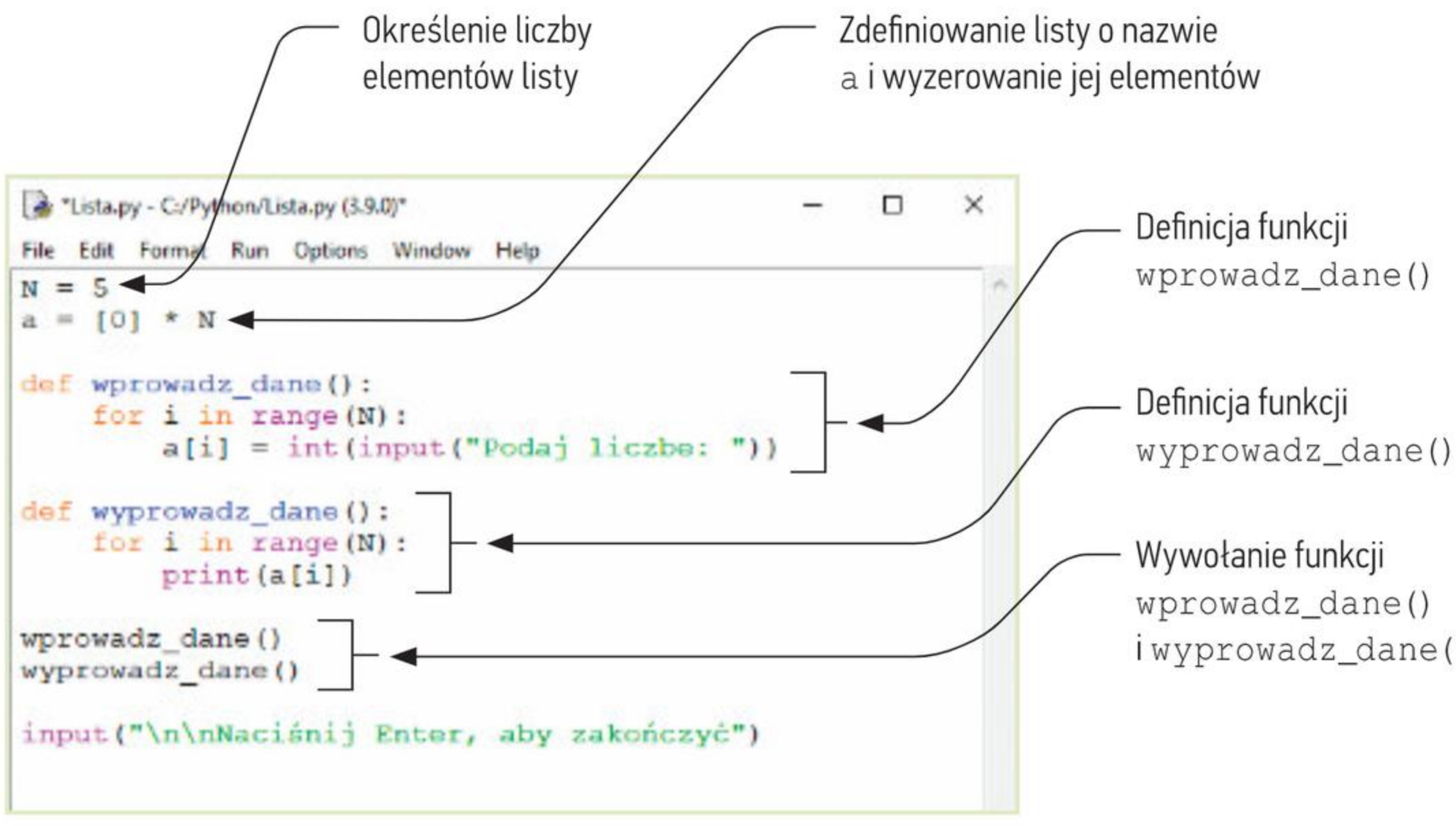
Przykłady definiowania list w języku Python (tabela 7.) ograniczyliśmy tylko do kilku, przydatnych do wykonywania dalszych ćwiczeń.

Przykład	Opis	Odwoływanie do elementów listy
wartości elementów listy możemy podać od razu w definicji listy: `L_DNI_W_MIESIACU = [31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31]`	oznacza zdefiniowanie listy o nazwie `L_DNI_W_MIESIACU` składającej się z dwunastu elementów o indeksach od 0 do 11	do elementów listy odwołujemy się przez zmienne: `L_DNI_W_MIESIACU[0]`, `L_DNI_W_MIESIACU[1]`, …, `L_DNI_W_MIESIACU[11]`
możemy przypisać od razu taką samą wartość wszystkim elementom listy: `moja_lista = [0] * 10`	oznacza zdefiniowanie listy o nazwie `moja_lista` składającej się z dziesięciu elementów, z których każdy jest równy zero (wyzerowanie wszystkich elementów listy)	do elementów listy odwołujemy się przez zmienne: `moja_lista[0]`, `moja_lista[1]`, …, `moja_lista[9]`

liczbę elementów możemy przypisać zmiennej:
`N = 100`
`a = [0] * N`
	oznacza zdefiniowanie listy o nazwie `a` składającej się ze stu elementów o wartości początkowej zero	do elementów listy odwołujemy się przez zmienne:
		`a[0], a[1], …, a[99]`

Tabela 7. Przykłady definiowania listy w języku Python

Do wprowadzania danych do listy i wyświetlania ich na ekranie zastosowaliśmy funkcje bez parametrów i niezwracające wartości.



Rys. 15. Program w języku Python, w którym zdefiniowano funkcję wprowadzającą dane z klawiatury do listy i funkcję wyświetlania danych z listy na ekranie – ćwiczenie 12.

Ćwiczenie 12. Wprowadzamy dane do listy i wyświetlamy je na ekranie

1. Utwórz nowy plik źródłowy. Zdefiniuj funkcje (rys. 15.):
   a. o nazwie `wprowadz_dane()` wprowadzającą dane do listy `a`, składającej się z pięciu liczb całkowitych,
   b. o nazwie `wyprowadz_dane()` wyświetlającą w kolejnych wierszach dane z listy na ekranie.
2. Wywołaj funkcje `wprowadz_dane()` i `wyprowadz_dane()`.
3. Zapisz program w pliku pod nazwą *Lista*. Uruchom program.

Zapamiętaj

- W języku Python typ zmiennej określa wartość, jaką przypiszemy zmiennej.
- Zmiennej możemy nadać wartość za pomocą instrukcji przypisania:
  - podając w programie konkretną wartość
    lub
  - wprowadzając tę wartość z klawiatury po uruchomieniu programu (używając funkcji `input()`).
- W języku Python do zapisywania algorytmów, w których występują sytuacje warunkowe, stosujemy instrukcję warunkową `if`.
- Jeśli w algorytmie występują powtórzenia, możemy zastosować instrukcję iteracyjną `for`. Liczba powtórzeń (iteracji) jest w niej zwykle z góry określona.
- Aby pisać przejrzyste programy, warto opracowywać oddzielnie problemy cząstkowe, stosując podprogramy (funkcje).
- Podprogram należy najpierw zdefiniować, a potem wywołać w programie głównym.
- W języku Python definiujemy funkcje niezwracające wartości i zwracające wartość.
- Możemy definiować funkcje z parametrami i bez parametrów.
- Aby zaprogramować algorytm, w którym w trakcie działania programu musimy pamiętać wszystkie elementy zbioru, możemy zapisywać dane na liście.

Serwis dla ciekawskich

Nazwa języka programowania Ada (który powstał w latach 70. XX wieku) pochodzi od imienia kobiety, którą wielu uważa za pierwszą programistkę komputerów. W pierwszej połowie XIX wieku Ada Lovelace (czyt. ejda lawlejs) współpracowała z Charlesem Babbage'em (czyt. czarlsem babydżem) przy projektowaniu pierwszej programowalnej maszyny liczącej (której jednak nigdy nie skonstruowano...). Tworzone przez lady Lovelace opisy rozwiązywania konkretnych zadań obliczeniowych uznaje się za pierwsze programy.

OKO W OKO Z MONITOREM

Pytania i polecenia

1. Jaka będzie wartość zmiennej `b` po wykonaniu programu w języku Python?

```
a = 5
b = 30
a = a + 10
b = a
```

2. Czym się różni operator `==` od `=`? Podaj przykłady użycia każdego z tych operatorów.

3. Kiedy komputer wykona instrukcje umieszczone po słowie `else` w instrukcji warunkowej?

4. Ile razy komputer wykona instrukcję `print(i)`? Dla jakich wartości zmiennej `i` komputer wykona tę instrukcję?

```
for i in range(1, 25, 3):
    print(i)
```

5. W jakim celu definiujemy podprogramy?

6. Kiedy definiujemy funkcję niezwracającą wartości, a kiedy zwracającą wartość? Wyjaśnij na przykładzie.

7. Dlaczego w funkcji zwracającej wartość musi wystąpić słowo `return`?

8. W jakim celu definiujemy listę? Podaj przykład użycia listy.

Zadania

1. Napisz program w języku Python obliczający pole trójkąta o podstawie `a` i wysokości `h`. Wartości `a` i `h` wprowadzaj z klawiatury. Wynik obliczenia i odpowiedni komunikat program powinien wyświetlać na ekranie. Zapisz program w pliku pod nazwą *Pole_trójkąta*.

2. Napisz program w języku Python obliczający sumę i średnią arytmetyczną trzech liczb wprowadzanych z klawiatury. Wyniki obliczeń i odpowiednie komunikaty program powinien wyświetlać na ekranie. Zapisz program w pliku pod nazwą *Średnia*.

3. Napisz program w języku Python obliczający iloraz liczb `a` i `b` wprowadzanych z klawiatury. Iloraz możemy obliczyć tylko dla `b` różnego od zera – jeśli ten warunek nie zostanie spełniony, program powinien wyświetlić napis „Nie dzielimy przez zero”. Zapisz program w pliku pod nazwą *Iloraz*.

4. Napisz program w języku Python obliczający pola dziesięciu trapezów o podstawach `a` i `b` oraz wysokości `h`. Wartości `a`, `b` i `h` wprowadzaj z klawiatury. Wyniki obliczeń i odpowiednie komunikaty program powinien wyświetlać na ekranie. Zapisz program w pliku pod nazwą *Trapez*.

5. Napisz program w języku Python wyświetlający na ekranie napis „Gwiazdkowe pozdrowienia”, a pod nim piętnaście gwiazdek (znaków *), w następnym wierszu – napis „Wszystkiego najlepszego”, a pod nim ponownie piętnaście gwiazdek. Zdefiniuj funkcję `Gwiazdki()` jako niezwracającą wartości i bez parametrów. Zadaniem funkcji jest wyświetlenie w jednym wierszu piętnastu gwiazdek. Funkcję wywołaj w programie głównym. Zapisz program w pliku pod nazwą *Życzenia*.

6. Napisz program w języku Python obliczający pole i obwód kwadratu. Zdefiniuj dwie funkcje `pole_kwadratu` i `obwod_kwadratu`. Każda z funkcji powinna być zdefiniowana z jednym parametrem o nazwie `a` i zwracać do programu głównego wynik obliczeń. W programie głównym wywołaj obie funkcje z parametrem aktualnym `bok_kwadratu`, którego wartość wprowadzaj z klawiatury. Zapisz program w pliku pod nazwą *Pole_i_obwód_kwadratu*.

DLA ZAINTERESOWANYCH

1. Napisz program w języku Python sprawdzający, która z trzech wprowadzonych liczb `x`, `y` i `z` jest największa. Program powinien wyświetlać na ekranie odpowiedni napis i wartość największej liczby. Zapisz program w pliku pod nazwą *Największa_z_trzech*.

2. Napisz program w języku Python wyświetlający w dwudziestu wierszach taką samą liczbę znaków „@”. Zdefiniuj funkcję niezwracającą wartości o nazwie `at` z jednym parametrem `n` określającym liczbę znaków „@”, która wyświetli n znaków „@” w jednym wierszu. Wartość parametru aktualnego `liczba_at` wprowadzaj w momencie wywołania funkcji w programie głównym. Zapisz program w pliku pod nazwą *Znaki_at*.

2.3. Algorytmy na liczbach naturalnych

Warto powtórzyć

1. Czym jest algorytm?
2. Czym jest specyfikacja problemu (zadania)?
3. Na czym polega przedstawianie algorytmu w postaci listy kroków?
4. W jaki sposób przedstawiamy na schemacie blokowym sytuację warunkową?
5. Na czym polega iteracja?
6. Jaką instrukcję stosujemy do zapisywania sytuacji warunkowej w języku Python? Wyjaśnij jej działanie.
7. Jaką instrukcję stosujemy do zapisywania iteracji w języku Python? Wyjaśnij jej działanie.

Na lekcjach informatyki w klasie ósmej poznamy podstawowe algorytmy na liczbach naturalnych stosowane w matematyce i informatyce. Niektóre z nich wykorzystujemy także na co dzień. Zapiszemy je w języku programowania Python. W tym temacie:

1. poznasz i zaprogramujesz algorytm badania podzielności liczb,
2. poznasz i zaprogramujesz algorytm Euklidesa w dwóch wersjach: z odejmowaniem i z resztą z dzielenia,
3. poznasz i zaprogramujesz algorytm wyodrębniania cyfr danej liczby.

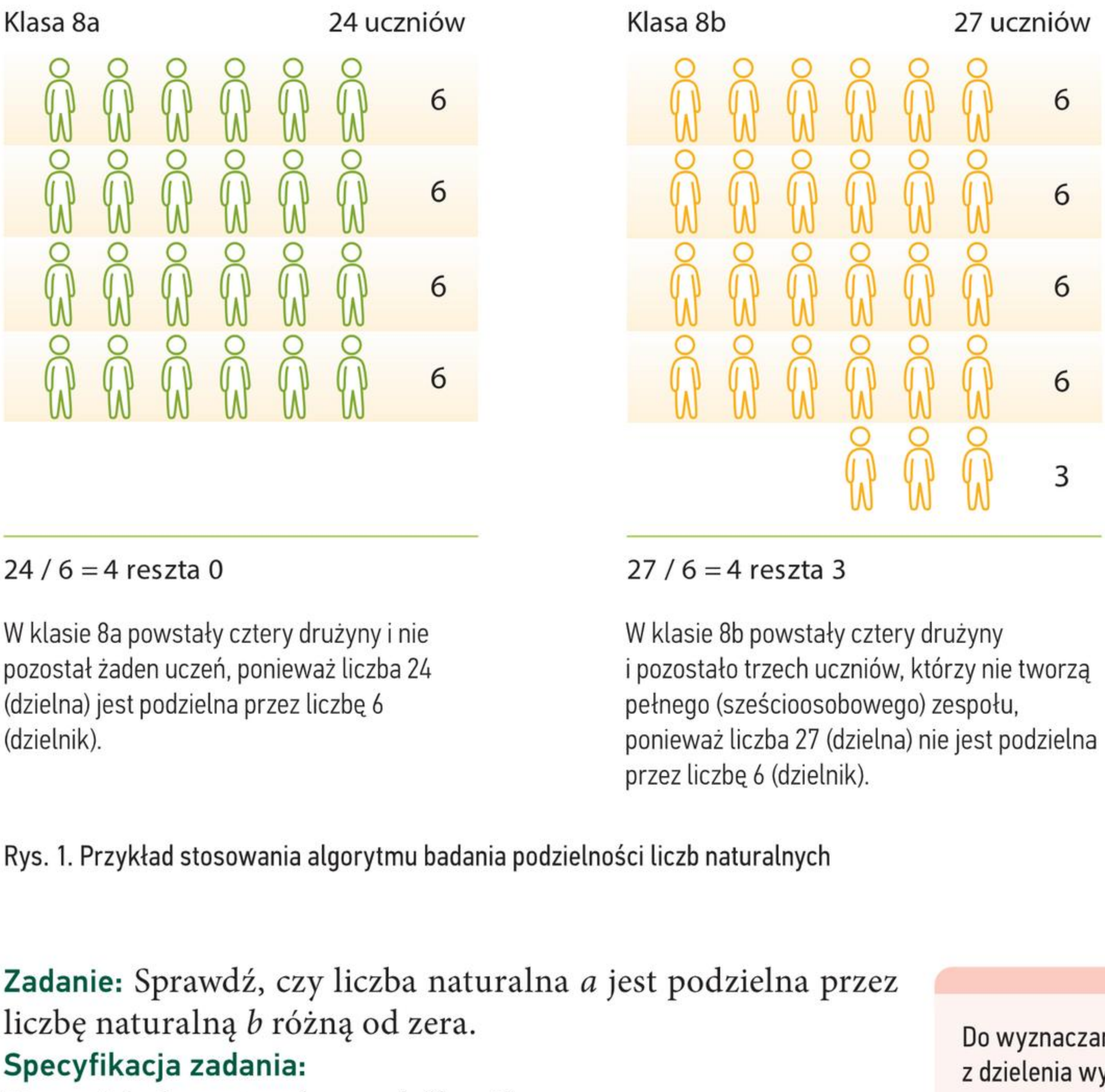
1 Badanie podzielności liczb naturalnych

W klasie 8a pewnej szkoły jest dwudziestu czterech uczniów, a w klasie 8b – dwudziestu siedmiu. W każdej klasie chcemy utworzyć sześcioosobowe zespoły do gry w siatkówkę. W której klasie można utworzyć je tak, aby wszyscy uczniowie zostali przydzieleni do jakiejś drużyny?

Aby odpowiedzieć na to pytanie, musimy sprawdzić, czy liczba uczniów danej klasy dzieli się przez sześć bez reszty. Do rozwiązania tego zadania wybierzemy algorytm badania podzielności liczb, w którym wykorzystuje się obliczanie reszty z dzielenia.

Jeśli reszta z dzielenia dwóch liczb naturalnych jest równa zero, mówimy, że pierwsza liczba (dzielna) jest podzielna przez drugą (dzielnik).

Algorytm badania podzielności liczb naturalnych



Rys. 1. Przykład stosowania algorytmu badania podzielności liczb naturalnych

Zadanie: Sprawdź, czy liczba naturalna a jest podzielna przez liczbę naturalną b różną od zera.

Specyfikacja zadania:

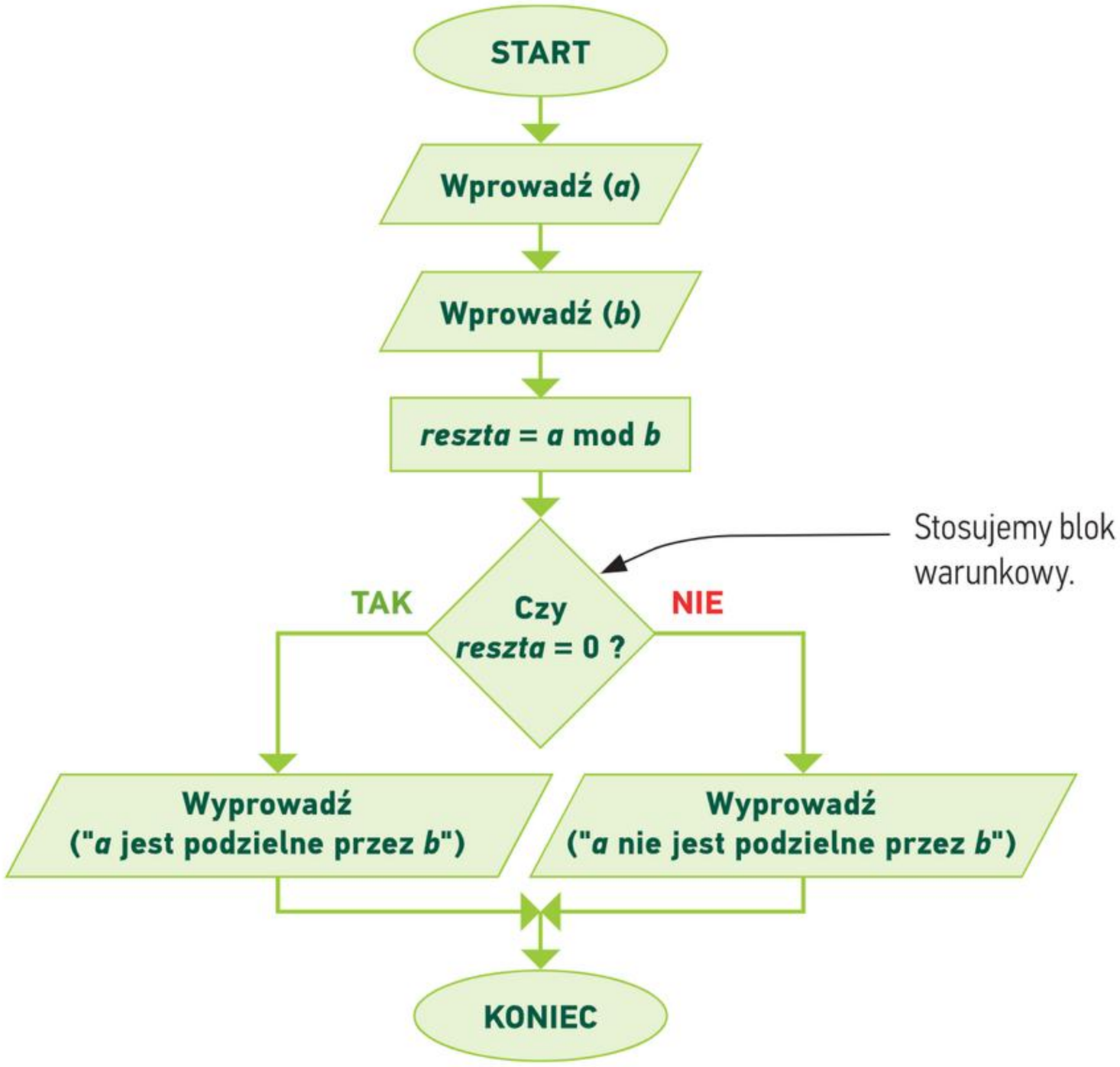
Dane: Liczby naturalne: a, b ($b \neq 0$).

Wynik: Komunikat: „b jest dzielnikiem a" lub „b nie jest dzielnikiem a" – zależnie od obliczonej reszty z dzielenia liczby a przez liczbę b.

Lista kroków:

1. Zacznij algorytm.
2. Wprowadź wartość dzielnej a.
3. Wprowadź wartość dzielnika b.
4. Zmiennej *reszta* przypisz wartość wyrażenia a **mod** b.
5. Jeżeli *reszta* jest równa zero, wyprowadź komunikat: „a jest podzielne przez b", w przeciwnym przypadku – „a nie jest podzielne przez b".
6. Zakończ algorytm.

Do wyznaczania reszty z dzielenia wykorzystamy operator modulo (**mod**). Zapis *reszta* = a **mod** b oznacza, że zmiennej *reszta* przypisano resztę z dzielenia liczby a przez liczbę b.



Rys. 2. Schemat blokowy algorytmu badania podzielności liczb

a	*b*	Obliczenie reszty	Czy *reszta* = 0?	Komunikat
21	4	*reszta* = 21 **mod** 4 = 1	**NIE**	*a* nie jest podzielne przez *b*
42	7	*reszta* = 42 **mod** 7 = 0	**TAK**	*a* jest podzielne przez *b*
245	8	*reszta* = 245 **mod** 8 = 5	**NIE**	*a* nie jest podzielne przez *b*
8	16	*reszta* = 8 **mod** 16 = 8	**NIE**	*a* nie jest podzielne przez *b*
18	6	*reszta* = 18 **mod** 6 = 0	**TAK**	*a* jest podzielne przez *b*
125	5	*reszta* = 125 **mod** 5 = 0	**TAK**	*a* jest podzielne przez *b*

Tabela 1. Testowanie algorytmu badania podzielności liczb

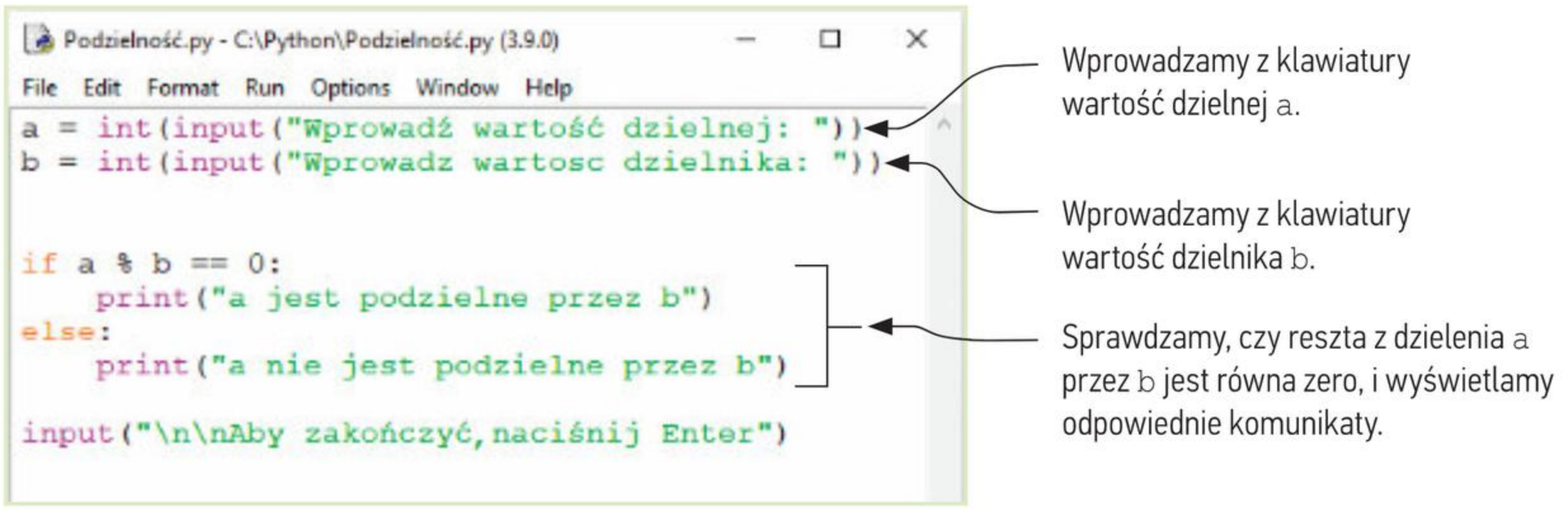
Ćwiczenie 1. Sprawdzamy działanie algorytmu badania podzielności liczb

Zbadaj podzielność czterech par liczb: $a = 34$ i $b = 9$, $a = 36$ i $b = 6$, $a = 112$ i $b = 4$, $a = 2$ i $b = 18$. Zapisz rozwiązanie w dokumencie tekstowym – wzoruj się na tabeli 1.

Zapisywanie algorytmu badania podzielności liczb naturalnych w języku Python

W algorytmie badania podzielności dwóch liczb naturalnych występuje sytuacja warunkowa, dlatego do jej zaprogramowania w języku Python wykorzystamy instrukcję warunkową `if else`.

Resztę z dzielenia wyznaczamy za pomocą operatora modulo, który w języku Python zapisujemy symbolem `%`.



Rys. 3. Program w języku Python realizujący algorytm badania podzielności liczb naturalnych

Ćwiczenie 2. Zapisujemy algorytm badania podzielności liczb naturalnych w języku Python

1. Program pokazany na rysunku 3. realizuje algorytm badania podzielności liczb naturalnych. Przepisz ten program i zapisz go w pliku pod nazwą *Podzielność*.
2. Uruchom kilkakrotnie program, testując go dla różnych wartości zmiennych `a` i `b`.
3. Objaśnij działanie programu, m.in. użycie poszczególnych instrukcji zgodnie z listą kroków tego algorytmu.

Ćwiczenie 3. Modyfikujemy program

1. Zmodyfikuj program zapisany w ćwiczeniu 2. tak, aby sprawdzał podzielność liczb tylko dla dzielnika `b` różnego od zera, a w przypadku gdy dzielnik `b` jest równy zero – wyświetlał komunikat „Wprowadzono dzielnik równy zero”.
2. Zapisz program w pliku pod nazwą *Podzielność_bez_zera*.

Ćwiczenie 4. Stosujemy algorytm badania podzielności liczb do rozwiązania przykładowego problemu

1. Napisz program sprawdzający, ile sześcioosobowych drużyn (`liczba_druzyn`) można utworzyć w każdej z pięciu klas pewnej szkoły, i wyświetlający kolejno wyniki na ekranie. Jeśli w danej klasie nie można utworzyć drużyn tak, aby wszyscy uczniowie zostali przydzieleni, program powinien wyświetlić odpowiedni komunikat. Liczbę uczniów (`liczba_uczniow`) każdej klasy wprowadzaj z klawiatury po uruchomieniu programu.
   Dane i wyniki program powinien wyświetlić w podobny sposób, jak pokazano na rysunku 4.
2. Zapisz program w pliku pod nazwą *Siatkówka*.

Wskazówka: Aby program wyświetlał komunikat przed wprowadzeniem liczby uczniów danej klasy (jak na rysunku 2.), zastosuj następujące instrukcje:

```
print("Wprowadź liczbę uczniów klasy", i + 1, end = "")
liczba_uczniow = int(input(": "))
```

```
C:\WINDOWS\py.exe
Wprowadź liczbę uczniów klasy 1: 31
Nie można utworzyć drużyn z wszystkich uczniów klasy
Wprowadź liczbę uczniów klasy 2: 30
Liczba drużyn: 5
Wprowadź liczbę uczniów klasy 3: 36
Liczba drużyn: 6
Wprowadź liczbę uczniów klasy 4: 27
Nie można utworzyć drużyn z wszystkich uczniów klasy
Wprowadź liczbę uczniów klasy 5: 24
Liczba drużyn: 4


Aby zakończyć, naciśnij Enter
```

Rys. 4. Wynik działania programu – ćwiczenie 4.

2 Algorytm Euklidesa

Opiszemy działanie algorytmu Euklidesa w dwóch wersjach iteracyjnych (z odejmowaniem i z resztą z dzielenia).

Algorytm Euklidesa służy do znajdowania największego wspólnego dzielnika (**NWD**) dwóch liczb naturalnych.

Działanie algorytmu Euklidesa w wersji z odejmowaniem

Euklides zauważył, że NWD dwóch liczb jest zawsze taki sam, jak NWD ich różnicy. Zatem od większej liczby można odjąć mniejszą tyle razy, ile ta się w niej zmieści.

Algorytm Euklidesa w wersji z odejmowaniem polega na odejmowaniu mniejszej liczby od większej i zastępowaniu większej liczby otrzymaną różnicą, dopóki większa liczba i otrzymana różnica nie będą równe.

Na przykład dla liczb $a = 10$ i $b = 25$ NWD wynosi 5. Korzystając z algorytmu Euklidesa, obliczamy NWD dla tych liczb w następujący sposób:

- od większej liczby odejmujemy mniejszą:
  $b - a = 25 - 10 = 15$,
- zastępujemy większą liczbę otrzymaną różnicą, czyli teraz:
  $a = 10$, $b = 15$,
- od większej liczby odejmujemy mniejszą:
  $b - a = 15 - 10 = 5$,
- zastępujemy większą liczbę otrzymaną różnicą, czyli teraz:
  $a = 10$, $b = 5$,
- od większej liczby odejmujemy mniejszą (teraz większą liczbą jest liczba a):
  $a - b = 10 - 5 = 5$,
- zastępujemy większą liczbę otrzymaną różnicą, czyli teraz:
  $a = 5$, $b = 5$.

Kończymy algorytm, ponieważ: $a = b$. Przyjmujemy, że NWD jest równe pierwszej z liczb (czyli liczbie a po obliczeniach): $NWD = a = 5$.

NWD
Największy wspólny dzielnik dwóch lub więcej liczb naturalnych (różnych od zera) to największa liczba naturalna, przez którą dzieli się bez reszty każda z tych liczb.

Algorytm Euklidesa – wersja z odejmowaniem

Zadanie: Znajdź największy wspólny dzielnik dwóch niezerowych liczb naturalnych. Wykorzystaj algorytm Euklidesa w wersji z odejmowaniem.

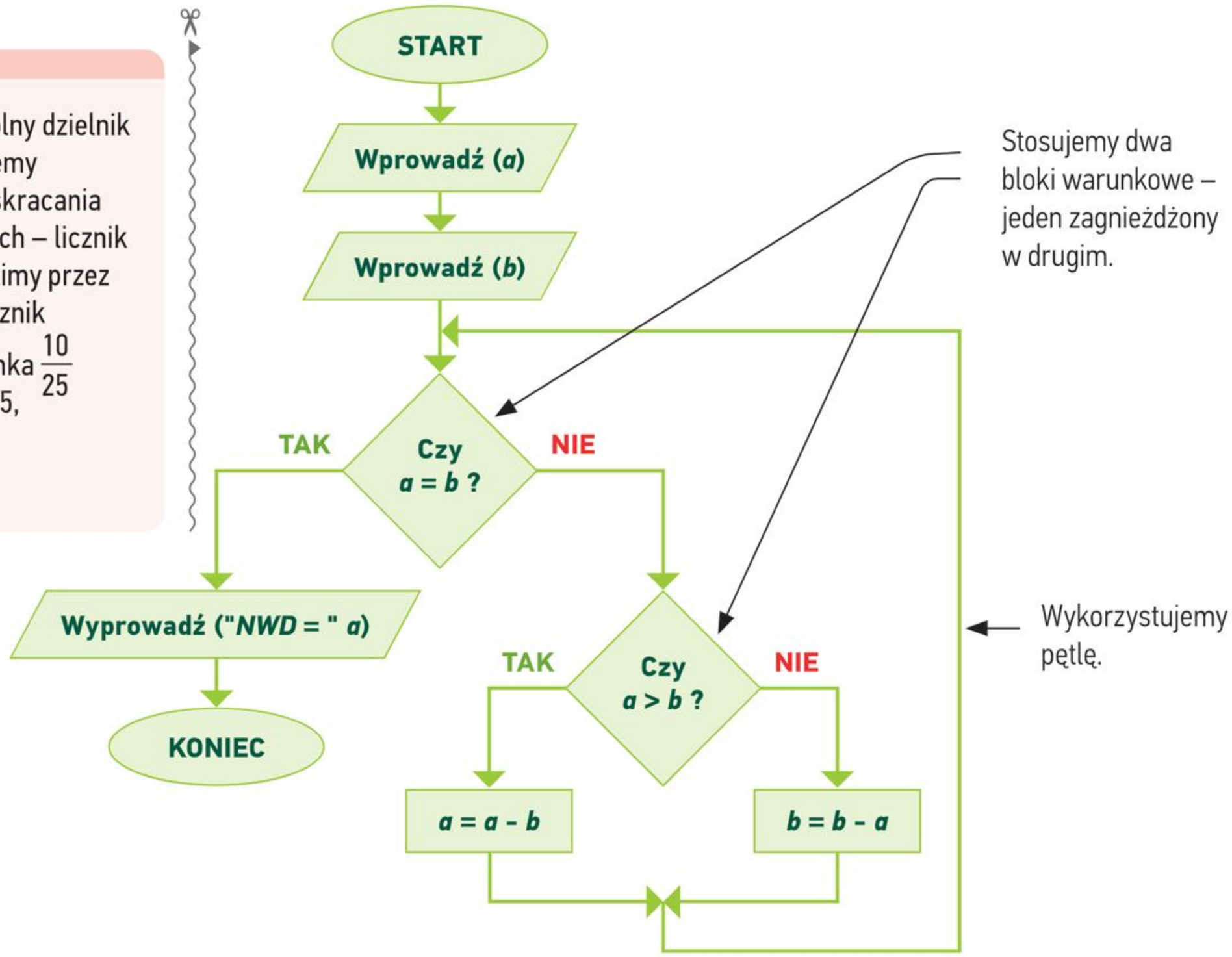
Specyfikacja zadania:

Dane: Liczby naturalne: a, b ($a \neq 0$ i $b \neq 0$).
Wynik: Wartość największego wspólnego dzielnika liczb a i b: *NWD*.

Lista kroków:

1. Zacznij algorytm.
2. Wprowadź wartość liczby a.
3. Wprowadź wartość liczby b.
4. Sprawdź, czy $a = b$: jeżeli tak, idź do kroku 7.
5. Jeżeli $a > b$, to zmiennej a przypisz wartość wyrażenia $a - b$; w przeciwnym przypadku zmiennej b przypisz wartość wyrażenia $b - a$.
6. Idź do kroku 4.
7. Wyprowadź wynik: *NWD* jest równe a.
8. Zakończ algorytm.

Największy wspólny dzielnik liczb (NWD) możemy wykorzystać do skracania ułamków zwykłych – licznik i mianownik dzielimy przez ich NWD. Jeśli licznik i mianownik ułamka $\frac{10}{25}$ podzielimy przez 5, otrzymamy $\frac{2}{5}$.



Rys. 5. Schemat blokowy algorytmu Euklidesa w wersji z odejmowaniem

a	*b*	Czy *a* = *b*?	Czy *a* > *b*?	Działania
Pierwsza para liczb: *a* = 4, *b* = 8				
4	8	NIE	NIE	$b = 8 - 4 = 4$
4	4	TAK	–	***NWD*** $= a =$ **4**
Druga para liczb: *a* = 25, *b* = 15				
25	15	NIE	TAK	$a = 25 - 15 = 10$
10	15	NIE	NIE	$b = 15 - 10 = 5$
10	5	NIE	TAK	$a = 10 - 5 = 5$
5	5	TAK	–	***NWD*** $= a =$ **5**

Tabela 2. Testowanie algorytmu Euklidesa w wersji z odejmowaniem

Ćwiczenie 5. Sprawdzamy działanie algorytmu Euklidesa w wersji z odejmowaniem

Sprawdź działanie algorytmu Euklidesa w wersji z odejmowaniem i wyznacz *NWD* dla liczb: $a = 3$ i $b = 9$ oraz $a = 16$ i $b = 12$. Zapisz rozwiązanie w dokumencie tekstowym – wzoruj się na tabeli 2.

Zapisywanie algorytmu Euklidesa w wersji z odejmowaniem w języku Python

W algorytmie Euklidesa znajdowania NWD dwóch liczb naturalnych `a` i `b` w wersji z odejmowaniem powtarzają się polecenia (iteracja), dlatego w programie zastosujemy instrukcję iteracyjną.

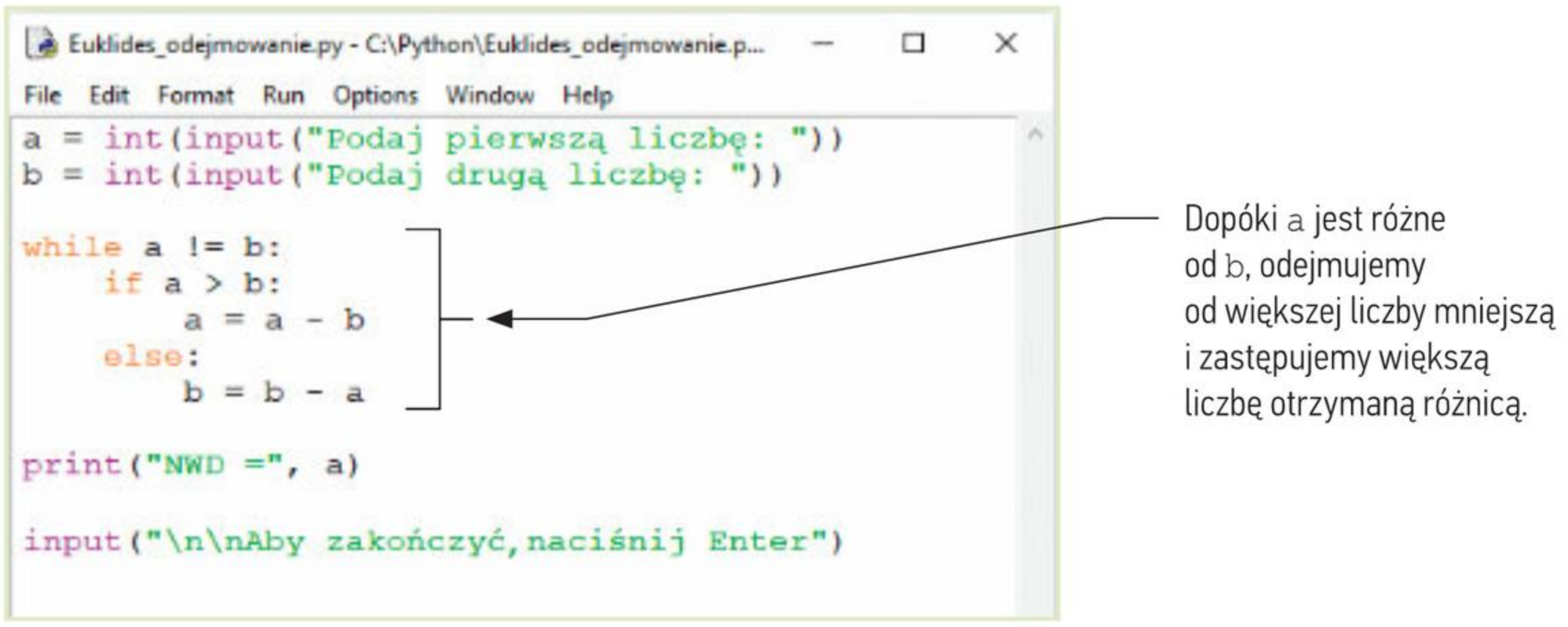
Liczba powtórzeń (iteracji) nie jest z góry określona, dlatego zastosujemy instrukcję iteracyjną `while`, w której będziemy sprawdzać, czy `a` jest różne od `b`.

Wewnątrz pętli `while` musimy też sprawdzać, która z liczb jest większa, dlatego zastosujemy instrukcję warunkową `if`.

```
while warunek:
        lista_instrukcji
kolejna_instrukcja
```

Rys. 6. Ogólna postać instrukcji iteracyjnej `while` w języku Python

Komputer wykonuje instrukcje w pętli (*lista_instrukcji*), dopóki *warunek* jest prawdziwy. Jeśli *warunek* jest fałszywy, komputer wykonuje *kolejną_instrukcję*. W bloku *lista_instrukcji* powinna być instrukcja, która zmienia wartość *warunku* – w przeciwnym wypadku pętla nigdy się nie skończy.



Rys. 7. Program w języku Python realizujący algorytm Euklidesa w wersji z odejmowaniem

Ćwiczenie 6. Zapisujemy algorytm Euklidesa w wersji z odejmowaniem w języku Python

1. Program pokazany na rysunku 7. realizuje algorytm Euklidesa w wersji z odejmowaniem. Przepisz ten program i zapisz go w pliku pod nazwą *Euklides_odejmowanie*. Uruchom kilkakrotnie program, testując go dla różnych wartości zmiennych `a` i `b`.
2. Objaśnij działanie programu, m.in. użycie poszczególnych instrukcji zgodnie z listą kroków lub schematem blokowym algorytmu.

Ćwiczenie 7. Modyfikujemy program

1. Zmodyfikuj program zapisany w ćwiczeniu 6. tak, aby szukał NWD dla dziesięciu par liczb `a` i `b` wprowadzanych kolejno z klawiatury i wyświetlał wyniki na ekranie.
2. Zapisz program w pliku pod nazwą *Euklides_odejmowanie_10*.

Działanie algorytmu Euklidesa w wersji z resztą z dzielenia

Algorytm Euklidesa w wersji z dzieleniem polega na wykonywaniu kolejnych dzieleń liczby *a* (dzielnej) przez liczbę *b* (dzielnik), zastępowaniu dzielnej dzielnikiem, a dzielnika resztą z dzielenia i wykonywaniu obliczeń, aż reszta z dzielenia osiągnie wartość 0.

Na przykład dla liczb $a = 25$ i $b = 10$ NWD obliczamy następująco:

- dzielimy liczbę *a* przez *b*:
  $a / b = 25 / 10 = 2$ reszta 5,
- liczbę *a* zastępujemy liczbą *b*, a liczbę *b* – resztą z dzielenia, czyli:
  $a = 10$, $b = 5$,
- dzielimy liczbę *a* przez *b*:
  $a / b = 10 / 5 = 2$ reszta 0,
- liczbę *a* zastępujemy liczbą *b*, a liczbę *b* – resztą z dzielenia, czyli:
  $a = 5$, $b = 0$.

Kończymy algorytm, ponieważ $b = 0$, czyli reszta z dzielenia wynosi 0. Przyjmujemy, że NWD jest równe ostatniej niezerowej reszcie, czyli $NWD = a = 5$.

Algorytm Euklidesa – wersja z resztą z dzielenia

Zadanie: Znajdź największy wspólny dzielnik dwóch niezerowych liczb naturalnych. Wykorzystaj algorytm Euklidesa w wersji z resztą z dzielenia.

Specyfikacja zadania:

Dane: Liczby naturalne: *a*, *b* ($a \neq 0$ i $b \neq 0$).

Wynik: Wartość największego wspólnego dzielnika liczb *a* i *b*: *NWD*.

Lista kroków:

1. Zacznij algorytm.
2. Wprowadź wartość liczby *a*.
3. Wprowadź wartość liczby *b*.
4. Sprawdź, czy $b = 0$: jeżeli tak, idź do kroku 9.
5. Zmiennej *dzielnik* przypisz wartość zmiennej *b*.
6. Zmiennej *b* przypisz wartość reszty z dzielenia *a* przez *b*.
7. Zmiennej *a* przypisz wartość zmiennej *dzielnik*.
8. Idź do kroku 4.
9. Wyprowadź wynik: *NWD* jest równe *a*.
10. Zakończ algorytm.

a	*b*	Czy *b* = 0?	Działania
Pierwsza para liczb: *a* = 9, *b* = 3			
9	3	**NIE**	*dzielnik* = *b* = 3 *b* = *a* **mod** *b* = 9 **mod** 3 = 0 *a* = *dzielnik* = 3
3	0	**TAK**	***NWD*** = *a* = 3
Druga para liczb: *a* = 28, *b* = 35			
28	35	**NIE**	*dzielnik* = *b* = 35 *b* = *a* **mod** *b* = 28 **mod** 35 = 28 *a* = *dzielnik* = 35
35	28	**NIE**	*dzielnik* = *b* = 28 *b* = *a* **mod** *b* = 35 **mod** 28 = 7 *a* = *dzielnik* = 28
28	7	**NIE**	*dzielnik* = *b* = 7 *b* = *a* **mod** *b* = 28 **mod** 7 = 0 *a* = *dzielnik* = 7
7	0	**TAK**	***NWD*** = *a* = 7

Tabela 3. Testowanie algorytmu Euklidesa w wersji z resztą z dzielenia

Uwaga: Jeśli pod tę samą zmienną podstawimy kolejną wartość, poprzednia wartość znika, dlatego wartość zmiennej musimy przechować w innej zmiennej, tzw. zmiennej pomocniczej. W omawianym algorytmie jest to zmienna *dzielnik*, której jest przypisany aktualny dzielnik.

Ćwiczenie 8. Sprawdzamy działanie algorytmu Euklidesa w wersji z resztą z dzielenia
Sprawdź działanie algorytmu Euklidesa w wersji z resztą z dzielenia i wyznacz *NWD* dla liczb: $a = 56$ i $b = 6$ oraz $a = 16$ i $b = 24$. Zapisz rozwiązanie w dokumencie tekstowym – wzoruj się na tabeli 3.

Zapisywanie algorytmu Euklidesa w wersji z resztą z dzielenia w języku Python

W algorytmie Euklidesa znajdowania NWD dwóch liczb naturalnych `a` i `b` w wersji z resztą z dzielenia powtarzają się polecenia (iteracja), dlatego w programie zastosujemy instrukcję iteracyjną.

Liczba iteracji (podobnie jak w wersji z odejmowaniem) nie jest z góry określona, dlatego zastosujemy instrukcję iteracyjną `while`, w której będziemy sprawdzać, czy `b` jest różne od zera.

Do wyznaczenia reszty z dzielenia liczby `a` przez liczbę `b` zastosujemy operator modulo `%`.

Euklides_dzielenie.py - C:\Python\Euklides_dzielenie.py (3.9.0)

File Edit Format Run Options Window Help

```
a = int(input("Podaj pierwszą liczbę: "))
b = int(input("Podaj drugą liczbę: "))

while b != 0:
    dzielnik = b
    b = a % b
    a = dzielnik

print("NWD =", a)

input("\n\nAby zakończyć, naciśnij Enter")
```

Dopóki `b` jest różne od 0, przypisujemy: zmiennej `dzielnik` wartość zmiennej `b`, zmiennej `b` resztę z dzielenia `a` przez `b`, zmiennej `a` wartość zmiennej `dzielnik`.

Zmienna `dzielnik` przechowuje aktualny dzielnik w trakcie wykonywania obliczeń.

Rys. 8. Program w języku Python realizujący algorytm Euklidesa w wersji z resztą z dzielenia

Ćwiczenie 9. Zapisujemy algorytm Euklidesa w wersji z resztą z dzielenia w języku Python

1. Program pokazany na rysunku 8. realizuje algorytm Euklidesa w wersji z resztą z dzielenia. Przepisz ten program i zapisz go w pliku pod nazwą *Euklides_dzielenie*. Uruchom kilkakrotnie program, testując go dla różnych wartości zmiennych `a` i `b`.
2. Objaśnij działanie programu, m.in. użycie poszczególnych instrukcji zgodnie z listą kroków algorytmu.

Ćwiczenie 10. Modyfikujemy program

1. Zmodyfikuj program zapisany w ćwiczeniu 9. tak, aby szukał NWD dla dziesięciu par liczb `a` i `b` wprowadzanych kolejno z klawiatury i wyświetlał wyniki na ekranie.
2. Zapisz program w pliku pod nazwą *Euklides_dzielenie_10*.

3 Wyodrębnianie cyfr danej liczby naturalnej

System dziesiętny opiera się na potęgach liczby 10. Liczbę naturalną w systemie dziesiętnym możemy rozpisać w następujący sposób:

$\mathbf{647} = \mathbf{6} \cdot 100 + \mathbf{4} \cdot 10 + \mathbf{7} \cdot 1 = \mathbf{6} \cdot 10^2 + \mathbf{4} \cdot 10^1 + \mathbf{7} \cdot 10^0$,

$\mathbf{92} = \mathbf{9} \cdot 10 + \mathbf{2} \cdot 1 = \mathbf{9} \cdot 10^1 + \mathbf{2} \cdot 10^0$,

$\mathbf{8} = \mathbf{8} \cdot 1 = \mathbf{8} \cdot 10^0$.

Cyfry danej liczby możemy wyodrębniać od ostatniej (dla liczby 647 jest to cyfra jedności: 7) lub od pierwszej (dla 647 to cyfra setek: 6).

Do obliczania reszty z dzielenia zastosujemy operator modulo **mod**.

Aby wyodrębnić cyfry danej liczby (od cyfry jedności), powtarzamy obliczanie reszty z dzielenia liczby przez 10 (pierwsza reszta jest cyfrą jedności), następnie od liczby odejmujemy resztę (cyfrę) i wynik dzielimy przez 10, aż do otrzymania wyniku dzielenia równego zero.

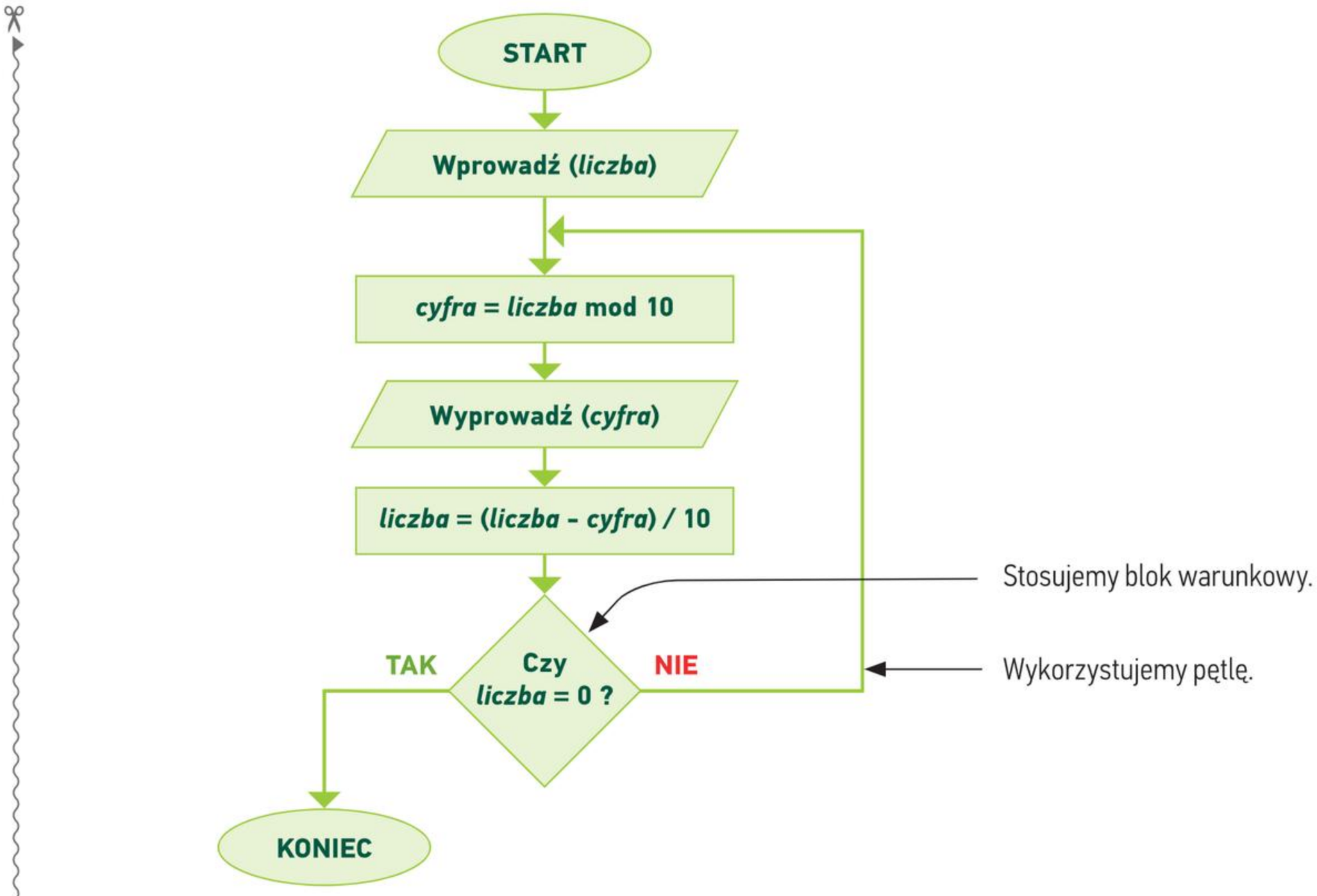
Algorytm wyodrębniania cyfr danej liczby naturalnej

Zadanie: Wyodrębnij cyfry danej liczby naturalnej różnej od 0, zaczynając od cyfry jedności.

Specyfikacja zadania:

Dane: Liczba naturalna różna od 0: *liczba*.

Wynik: Cyfry danej liczby, zaczynając od cyfry jedności.



Rys. 9. Schemat blokowy algorytmu wyodrębniania cyfr danej liczby od cyfry jedności

Uwagi:

- Zmienna *cyfra* służy do pamiętania wartości kolejnych cyfr (tu: reszt z dzielenia kolejnych wartości zmiennej *liczba* przez 10).
- Wartość zmiennej *liczba*, której cyfry wyodrębniamy, zmienia się w czasie wykonywania algorytmu.

liczba	Czy *liczba* = 0?	Działania	Cyfra
647	NIE	*cyfra* = 647 **mod** 10 = **7** *liczba* = (647 – 7) / 10 = 64	**7**
64	NIE	*cyfra* = 64 **mod** 10 = **4** *liczba* = (64 – 4) / 10 = 6	**4**
6	NIE	*cyfra* = 6 **mod** 10 = **6** *liczba* = (6 – 6) / 10 = 0	**6**
0	TAK	–	–

Tabela 4. Testowanie algorytmu wyodrębniania cyfr danej liczby

Ćwiczenie 11. Piszemy listę kroków algorytmu wyodrębniania cyfr danej liczby

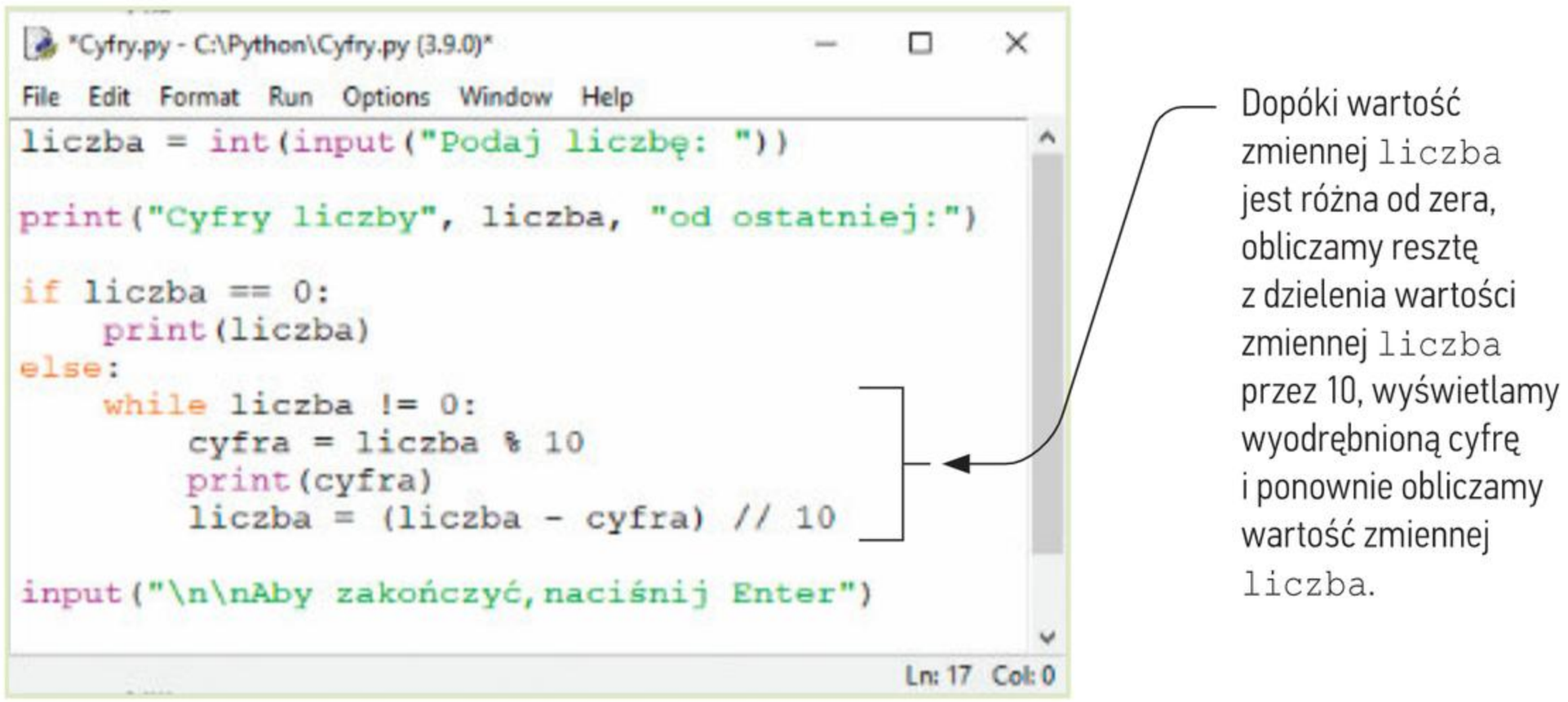
1. Zapoznaj się z opisem algorytmu wyodrębniania cyfr danej liczby.
2. Przeanalizuj schemat blokowy, korzystając z przykładu podanego w tabeli 4. Sprawdź działanie algorytmu dla liczby 8945.
3. Zapisz w dokumencie tekstowym listę kroków tego algorytmu.

Wskazówka: Wzoruj się na zapisach list kroków innych algorytmów opisanych w tym temacie.

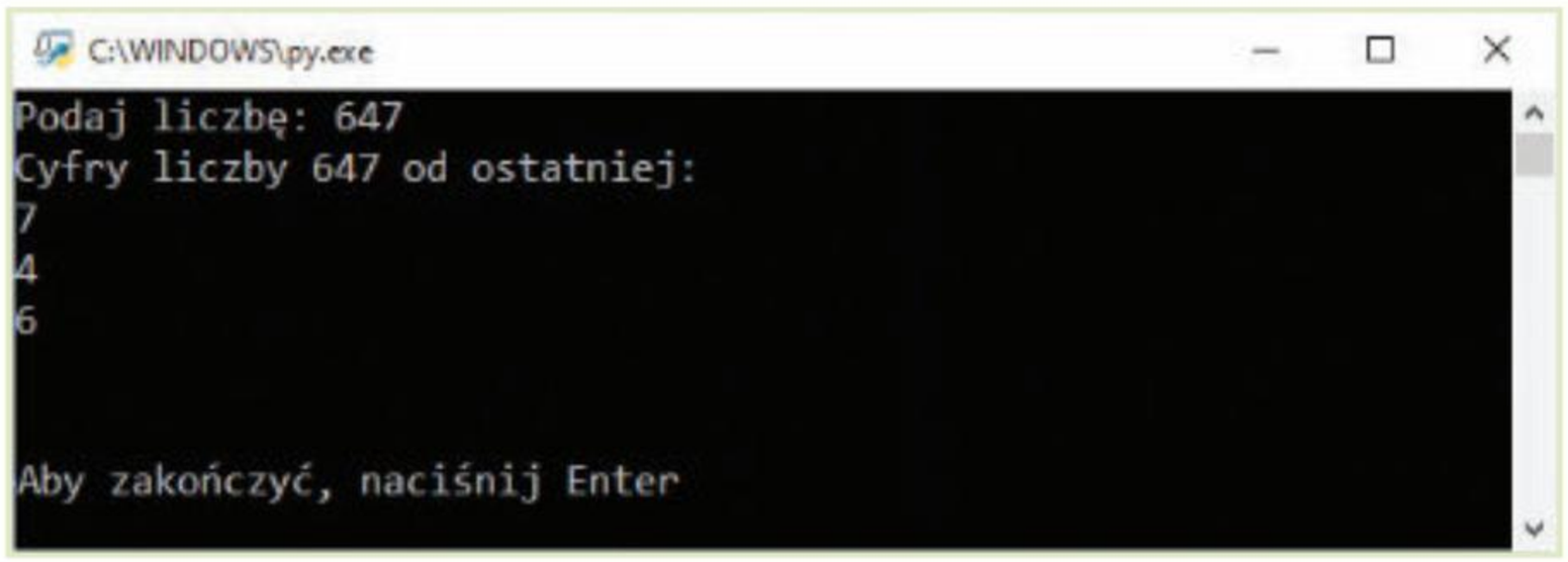
Zapisywanie algorytmu wyodrębniania cyfr danej liczby w języku Python

W algorytmie wyodrębniania cyfr danej liczby występuje powtarzanie poleceń, dlatego w programie zastosujemy instrukcję iteracyjną. Liczba iteracji (podobnie jak w algorytmie Euklidesa w wersji z odejmowaniem) nie jest z góry określona, dlatego zastosujemy instrukcję iteracyjną `while`.

Aby dla liczby zero wyświetlić cyfrę 0, dodamy instrukcję warunkową `if` sprawdzającą, czy wartość zmiennej `liczba` równa się zero.



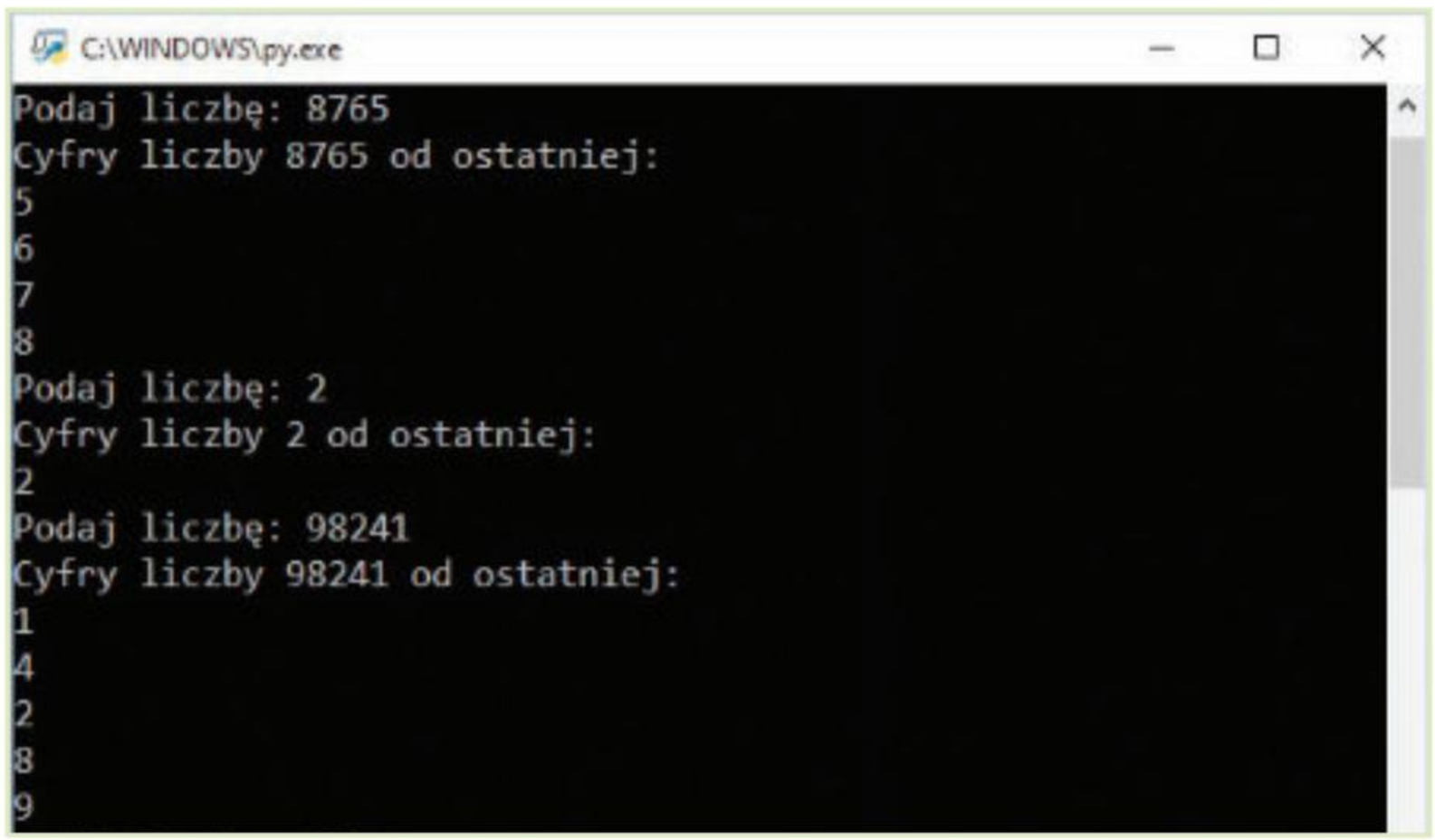
Rys. 10. Program w języku Python realizujący algorytm wyodrębniania cyfr danej liczby



Rys. 11. Wynik działania programu – ćwiczenie 12.

Ćwiczenie12. Zapisujemy algorytm wyodrębniania cyfr danej liczby w języku Python

1. Program pokazany na rysunku 10. realizuje algorytm wyodrębniania cyfr danej liczby, zaczynając od ostatniej cyfry (cyfry jedności). Przepisz ten program i zapisz go w pliku pod nazwą *Cyfry*. Uruchom kilkakrotnie program, testując go dla różnych wartości zmiennej `liczba`.
2. Objaśnij działanie programu, m.in. użycie poszczególnych instrukcji zgodnie ze schematem blokowym algorytmu.



Rys. 12. Fragment wyniku działania programu – ćwiczenie 13.

Ćwiczenie 13. Modyfikujemy program

1. Zmodyfikuj program zapisany w ćwiczeniu 12., aby wyodrębniał cyfry dziesięciu liczb wprowadzanych z klawiatury i wyświetlał kolejne wyniki na ekranie. Zapisz program w pliku pod nazwą *Cyfry_10*.
2. Odpowiedz na pytanie: *Ile razy powtarzają się polecenia w pętli* `while` *dla liczby trzycyfrowej, a ile razy – dla jednocyfrowej?* Uzasadnij odpowiedź.
3. Zamień instrukcję: `liczba = (liczba - cyfra) // 10`
   na instrukcję: `liczba = liczba // 10`.
   Czy zmieniło się działanie programu? Uzasadnij odpowiedź.

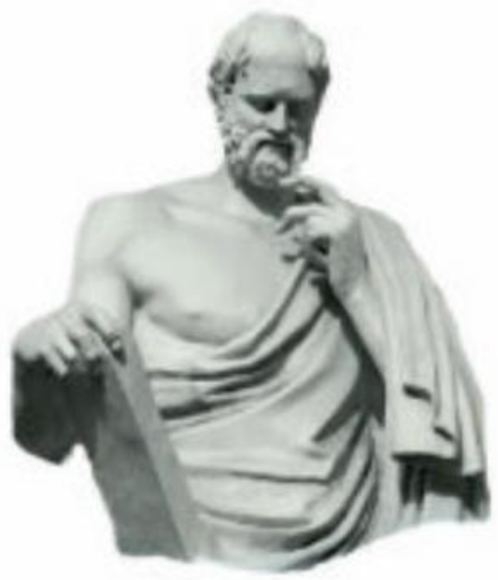
Zapamiętaj

- Liczba *a* jest podzielna przez liczbę *b*, jeśli reszta z dzielenia *a* przez *b* jest równa zero.
- W języku Python w programie badania podzielności liczb, w którym występuje sytuacja warunkowa, stosujemy instrukcję warunkową `if`.
- Największy wspólny dzielnik (NWD) dwóch liczb naturalnych możemy znaleźć, stosując algorytm Euklidesa. Algorytm ten możemy zrealizować w dwóch wersjach: z odejmowaniem i z resztą z dzielenia.
- Aby zapisać w języku Python algorytm Euklidesa w wersji z odejmowaniem i w wersji z resztą z dzielenia oraz algorytm wyodrębniania cyfr danej liczby, możemy zastosować instrukcję iteracyjną `while`. W tych algorytmach występuje powtarzanie poleceń (iteracja) – są to algorytmy iteracyjne.
- W algorytmie wyodrębniania cyfr danej liczby naturalnej wykorzystujemy reszty z dzielenia danej liczby przez dziesięć. Reszty z dzielenia są kolejnymi cyframi liczby.

Serwis dla ciekawskich

Euklides z Aleksandrii był greckim matematykiem, który ok. 300 lat p.n.e. stworzył dzieło pt. „Elementy". Opisał w nim m.in. sposób obliczania największej wspólnej miary dwóch odcinków (czyli największego wspólnego dzielnika dwóch liczb naturalnych).

„Elementy" były przez wiele wieków podstawowym podręcznikiem do nauki geometrii.



OKO W OKO Z MONITOREM

Pytania i polecenia

1. Jak sprawdzić, czy liczba *x* jest podzielna przez liczbę *y*? Przedstaw na przykładzie.
2. Dlaczego w instrukcji warunkowej programu realizującego algorytm badania podzielności liczb naturalnych warunek logiczny ma postać `a % b == 0`? Uzasadnij odpowiedź.

3. Wyjaśnij na przykładzie liczb 36 i 8, dlaczego warunkiem zakończenia algorytmu Euklidesa w wersji z odejmowaniem jest $a = b$.

4. Czym różni się wersja algorytmu Euklidesa z odejmowaniem od wersji z resztą z dzielenia? Wyjaśnij na przykładzie liczb 36 i 8.

5. Czy w programie realizującym algorytm Euklidesa w obydwu wersjach można zastosować instrukcję iteracyjną `for` zamiast `while`? Uzasadnij odpowiedź.

6. Na czym opiera się idea algorytmu wyodrębniania cyfr danej liczby?

7. Na czym polega działanie instrukcji iteracyjnej `while` w algorytmie wyodrębniania cyfr danej liczby?

Zadania

1. W dokumencie tekstowym napisz specyfikację zadania i listę kroków algorytmu sprawdzania, czy dana liczba naturalna jest parzysta. Jeśli liczba jest parzysta, wyprowadź napis „Podana liczba jest parzysta", a gdy nieparzysta – „Podana liczba jest nieparzysta".

2. Utwórz schemat blokowy algorytmu sprawdzania, czy dana liczba naturalna jest parzysta.

3. Napisz program sprawdzający, czy dana liczba naturalna jest parzysta. Zapisz program w pliku pod nazwą *Parzyste*.

4. Napisz program sprawdzający, czy liczba a wprowadzona z klawiatury jest podzielna przez 7. Program ma umożliwiać wprowadzenie i sprawdzenie siedmiu liczb. Zapisz program w pliku pod nazwą *Podzielność_przez_7*.

5. Przedstaw w postaci schematu blokowego algorytm Euklidesa w wersji z resztą z dzielenia.

DLA ZAINTERESOWANYCH

1. Zastanów się, jak sprawdzić podzielność jednej liczby przez inną bez konieczności obliczania reszty z dzielenia. Zapisz w dokumencie tekstowym listę kroków tego algorytmu lub przedstaw go w postaci schematu blokowego.

2. Napisz program sprawdzający podzielność jednej liczby przez inną bez konieczności obliczania reszty z dzielenia. Zapisz program w pliku pod nazwą *Podzielność_inaczej*.

3. W dokumencie tekstowym napisz specyfikację zadania i listę kroków oraz utwórz schemat blokowy algorytmu wyodrębniania cyfr danej liczby, zaczynając od cyfry pierwszej od lewej.

4. Napisz program służący do wyodrębniania cyfr danej liczby od cyfry pierwszej od lewej do ostatniej. Zapisz program w pliku pod nazwą *Cyfry_odwrotnie*.

2.4. Algorytmy wyszukiwania

Warto powtórzyć

1. Na czym polega iteracja?
2. W jaki sposób zapisujemy iterację w języku Python?
3. Jaka jest różnica między instrukcją `for` a instrukcją `while`? Podaj przykład wykorzystania tych poleceń.
4. W jaki sposób przypisujemy zmiennej konkretną wartość, a jak wprowadzamy wartość zmiennej z klawiatury?

Podczas rozwiązywania niektórych problemów (z różnych dziedzin nauki i życia codziennego) szukamy w danym zbiorze najmniejszego lub największego elementu, np. najmniejszej liczby, najwyższej osoby. Czasem musimy znaleźć książkę na półce, na której utwory ułożono np. alfabetycznie według nazwiska autora. Istnieją różne algorytmy wyszukiwania. W tym temacie:

1. poznasz i zaprogramujesz algorytm wyszukiwania elementu w zbiorze nieuporządkowanym,
2. poznasz i zaprogramujesz algorytm wyszukiwania elementu w zbiorze uporządkowanym.

1 Wyszukiwanie elementu w zbiorze nieuporządkowanym

Wyszukiwanie elementu w zbiorze nieuporządkowanym pokażemy na przykładzie znajdowania największego elementu w zbiorze. Wyjaśnimy również, jak sprawdzić, czy w zbiorze nieuporządkowanym istnieje dany element.

Przyjmiemy, że zbiór do przeszukania zawiera n elementów, które są liczbami całkowitymi.

Wyszukiwanie największej liczby w zbiorze nieuporządkowanym

Aby znaleźć największą spośród n liczb, porównujemy kolejne wartości liczb ze zbioru z największą znalezioną do tej pory liczbą (zapamiętaną osobno). Na początku algorytmu przyjmujemy, że pierwsza liczba ze zbioru jest największa. Porównania liczb powtarzamy, aż sprawdzimy wszystkie liczby ze zbioru.

Algorytm wyszukiwania największej liczby spośród *n* liczb

Zadanie: Znajdź największą liczbę spośród *n* liczb całkowitych.

Specyfikacja zadania:

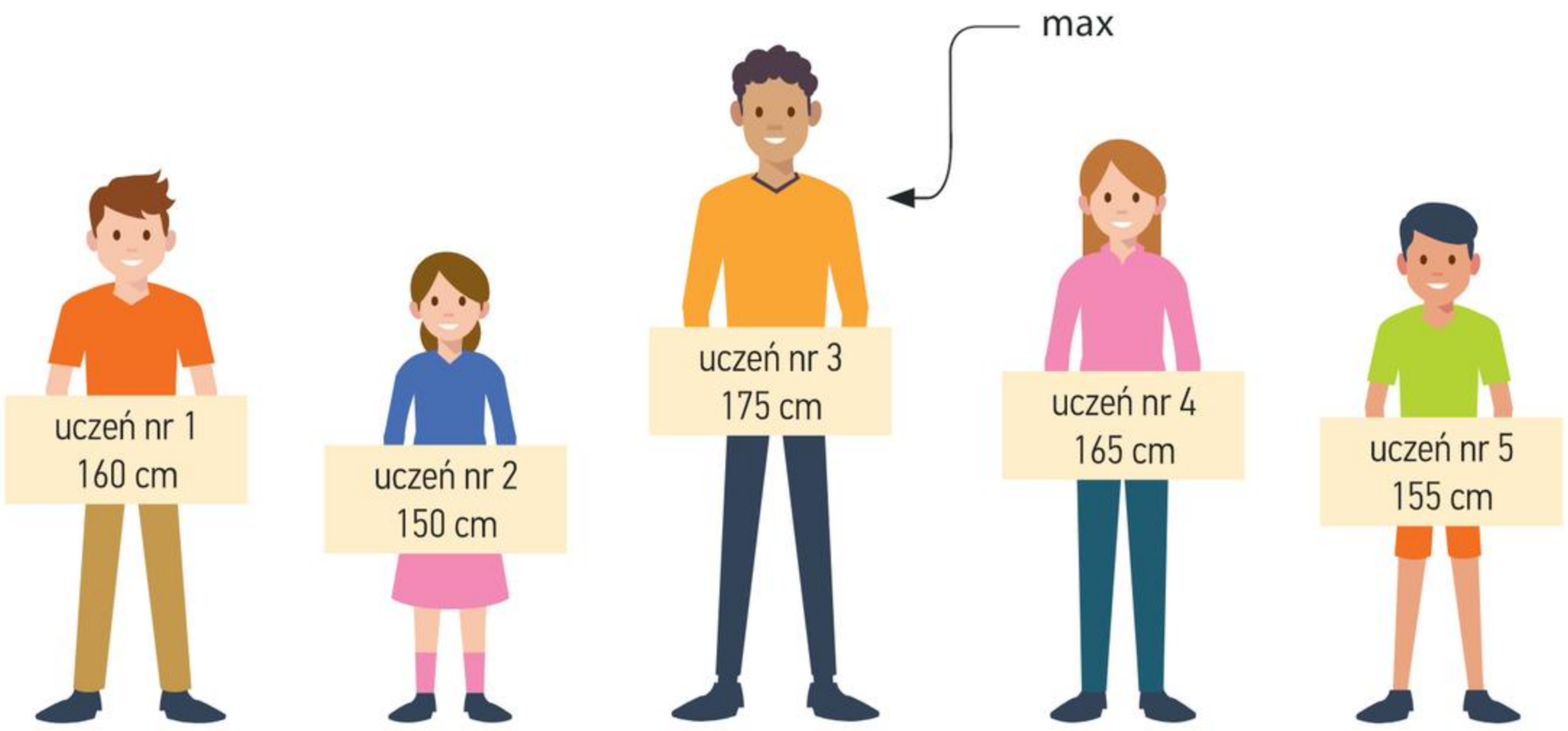
Dane: Liczba naturalna *n* oznaczająca liczbę elementów zbioru, n liczb całkowitych wprowadzanych kolejno i zapamiętywanych w zmiennej *a*.
Wynik: Wartość elementu największego: *maks*.

Lista kroków:

1. Zacznij algorytm.
2. Wprowadź liczbę elementów zbioru: *n*.
3. Wprowadź wartość pierwszej liczby: *a*.
4. Zmiennej *maks* przypisz wartość pierwszej liczby: *maks* = *a*.
5. Wprowadź wartość kolejnej liczby: *a*.
6. Porównaj kolejną liczbę ze zbioru z *maks*: czy *a* > *maks*?
7. Jeśli kolejna liczba jest większa od *maks*, to zmiennej *maks* przypisz wartość tej liczby: *maks* = *a*.
8. Jeśli nie jest to ostatnia liczba, wróć do kroku 5.
9. Wyprowadź wynik: *maks*.
10. Zakończ algorytm.

Algorytm wyszukiwania największej liczby spośród *n* liczb zastosujemy do znajdowania wzrostu najwyższego z *n* uczniów (gdzie *n* to liczba uczniów klasy). Sprawdzimy działanie algorytmu dla *n* = 5. W tabeli na rysunku 1. podaliśmy wzrost uczniów w centymetrach (czyli kolejne wartości *a*).

	uczeń nr 1	uczeń nr 2	uczeń nr 3	uczeń nr 4	uczeń nr 5
wzrost w cm	160	150	175	165	155



Rys. 1. Przykład użycia algorytmu wyszukiwania największej spośród *n* liczb

a	Czy *a* > *maks*?	Odpowiedź	Działania
wprowadź wzrost pierwszego ucznia: *a* = 160	–	–	zmiennej *maks* przypisz wzrost pierwszego ucznia: *maks* = 160
wprowadź wzrost drugiego ucznia: *a* = 150	porównaj wzrost drugiego ucznia z wartością *maks:* czy 150 > 160?	**NIE**	pozostaw *maks* bez zmian: *maks* = 160
wprowadź wzrost trzeciego ucznia: *a* = 175	porównaj wzrost trzeciego ucznia z wartością *maks*: czy 175 > 160?	**TAK**	zmiennej *maks* przypisz wzrost trzeciego ucznia: *maks* = 175
wprowadź wzrost czwartego ucznia: *a* = 165	porównaj wzrost czwartego ucznia z wartością *maks:* czy 165 > 175?	**NIE**	pozostaw *maks* bez zmian: *maks* = 175
wprowadź wzrost piątego ucznia: *a* = 155	porównaj wzrost piątego ucznia z wartością *maks*: czy 155 > 175?	**NIE**	pozostaw *maks* bez zmian: *maks* = 175
–	–	–	wzrost najwyższego ucznia to 175 cm

Tabela 1. Przykład wykonania algorytmu dla *n* = 5

Ćwiczenie 1. Sprawdzamy działanie algorytmu wyszukiwania największej liczby spośród pięciu liczb

Przedstaw wspólnie z koleżankami i kolegami z klasy algorytm wyszukiwania najwyższego ucznia spośród pięciu uczniów. Możecie przygotować odpowiednie pomoce dydaktyczne i się nimi posłużyć (rys. 1.).

Ćwiczenie 2. Sprawdzamy działanie algorytmu wyszukiwania największej liczby spośród *n* liczb

Sprawdź działanie algorytmu wyszukiwania największej liczby:

a. dla zbioru liczb {70, 160, 155, 180, 185};

b. dla zbioru liczb {8, –6, 23, 0, –50, 34, –23};

c. dla dziesięciu wybranych liczb całkowitych.

Zapisujemy w języku Python algorytm wyszukiwania największego elementu w zbiorze

Algorytm wyszukiwania elementu w zbiorze nieuporządkowanym (na przykładzie wyszukiwania największego elementu) zapiszemy w języku Python. Zdefiniujemy funkcję `maksimum()` z jednym parametrem wyszukującą największy element w `n`-elementowym zbiorze liczb całkowitych. Parametrem przekazywanym funkcji jest liczba elementów zbioru. Funkcja będzie zwracać wartość największego elementu (zapisaną w zmiennej `maks`).

Maksimum_n.py - C:/Python/Maksimum_n.py (3.9.0)

File Edit Format Run Options Window Help

```
def maksimum(n):
    for i in range(n):
        a = int(input("Podaj liczbę: "))
        if i == 0:
            maks = a
        else:
            if a > maks:
                maks = a
    return maks

liczba_elementow = int(input("Podaj liczbę elementów: "))
najwiekszy = maksimum(liczba_elementow)
print("Maksimum wynosi:", najwiekszy)

input("\n\nNaciśnij Enter, aby zakończyć")
```

Na początku przyjmujemy, że największą liczbą jest pierwsza liczba ze zbioru, dlatego pod zmienną `maks` podstawiamy wartość pierwszej wprowadzonej liczby.

Jeśli kolejna liczba jest większa od `maks`, to zmiennej `maks` przypisujemy wartość tej liczby.

Funkcja zwraca wartość największego elementu zapisaną w zmiennej `maks`.

Wywołanie funkcji `maksimum()` z parametrem aktualnym `liczba_elementow` – wynik przypisujemy zmiennej `najwiekszy`.

Rys. 2. Program w języku Python realizujący algorytm wyszukiwania największej liczby w zbiorze *n*-elementowym – ćwiczenie 3.

Ćwiczenie 3. Zapisujemy algorytm wyszukiwania największej liczby w zbiorze *n*-elementowym w języku Python

1. Program w języku Python pokazany na rysunku 2. realizuje algorytm wyszukiwania największego elementu w zbiorze *n*-elementowym. Przepisz ten program i zapisz go w pliku pod nazwą *Maksimum_n*.
2. Uruchom program. Sprawdź jego działanie dla różnych danych, w tym dla danych pokazanych na rysunku 1.
3. Objaśnij działanie programu, m.in. uzasadnij użycie poszczególnych poleceń zgodnie z podaną listą kroków.

Opisany w liście kroków algorytm wyszukiwania największego elementu w zbiorze jest przykładem metody nazywanej przeszukiwaniem liniowym. W algorytmie przeszukiwania liniowego sprawdzamy wartość elementów zbioru po kolei, według ustalonego wcześniej porządku.

Algorytm wyszukiwania danego elementu w zbiorze nieuporządkowanym

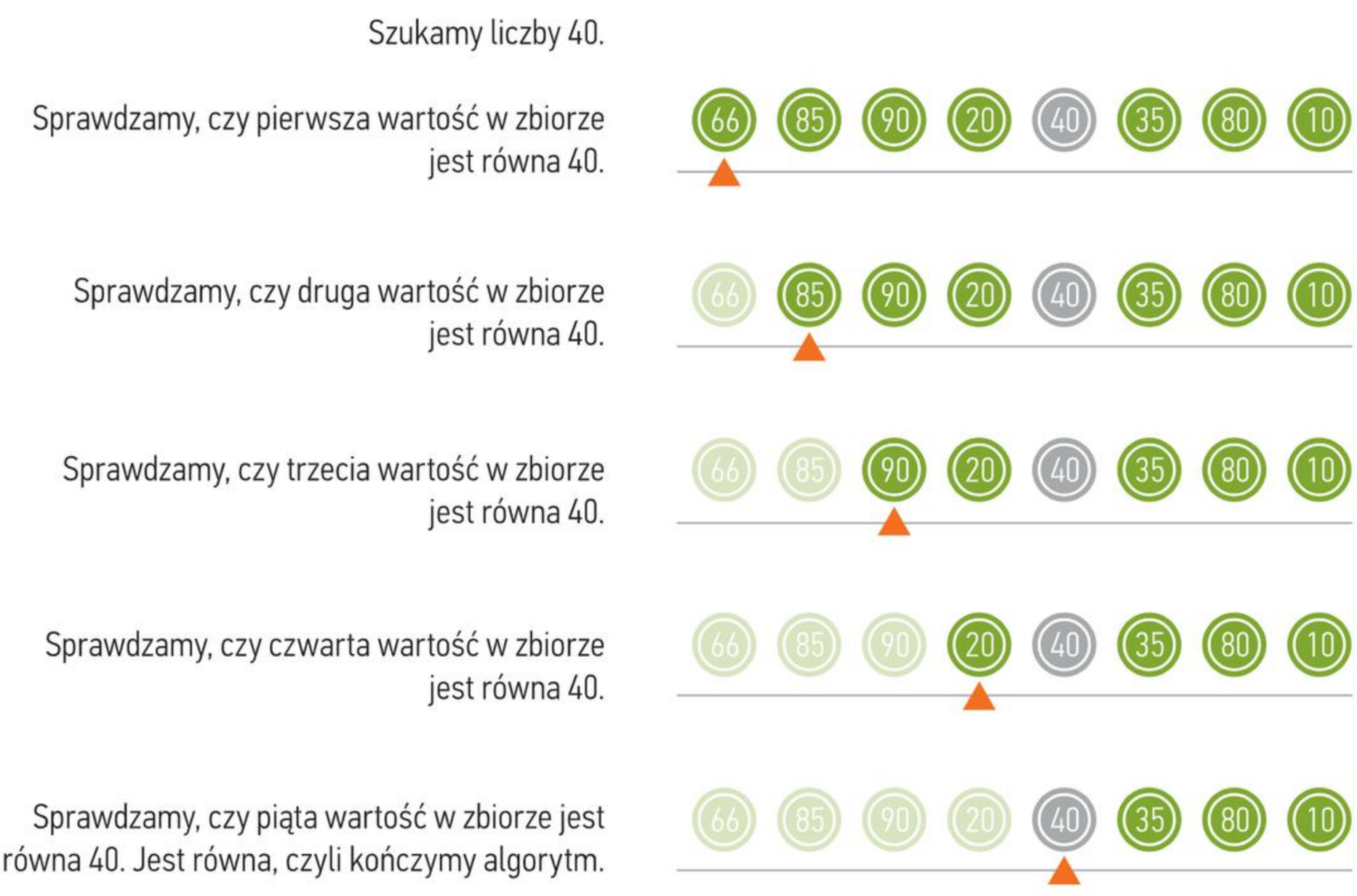
W algorytmie wyszukiwania elementu w zbiorze nieuporządkowanym szukamy danego elementu i równocześnie jego położenia w tym zbiorze.

W sytuacji, kiedy szukany element występuje w zbiorze więcej niż jeden raz, algorytm kończy działanie po natknięciu się na niego po raz pierwszy.

Aby stwierdzić, czy dany element istnieje w zbiorze n-elementowym, musimy sprawdzić każdy z elementów zbioru, ponieważ może on znajdować się w dowolnym miejscu (nawet być ostatnim elementem). W szczególnym przypadku szukanego elementu może w zbiorze w ogóle nie być.

Opis algorytmu: Dany jest nieuporządkowany zbiór ośmiu liczb (rys. 3.). Poszukiwana liczba to 40 (zaznaczona na szaro). Zakładamy, że w zbiorze jest tylko jedna taka liczba. Przeglądamy wszystkie kolejne elementy zbioru, stosując przeszukiwanie liniowe.

Zaczynamy od sprawdzenia, czy pierwsza liczba (zaznaczona pomarańczową strzałką) jest szukaną liczbą. Jeśli tak, kończymy poszukiwanie. Jeżeli nie – sprawdzamy kolejną liczbę. Algorytm kończy się, gdy znajdziemy liczbę równą szukanej, a jeśli jej nie ma – gdy dojdziemy do końca zbioru.



Rys. 3. Przykład wyszukiwania liczby w zbiorze nieuporządkowanym

Ćwiczenie 4. Sprawdzamy działanie algorytmu wyszukiwania liczby w zbiorze nieuporządkowanym

Przygotuj odpowiednie pomoce dydaktyczne (rys. 3.) i przedstaw wspólnie z koleżankami i kolegami z klasy algorytm wyszukiwania liczby w zbiorze nieuporządkowanym.

Wskazówka: Możesz zapisać liczby na kartkach i układać je na stole lub podłodze.

Zapisujemy w języku Python algorytm wyszukiwania elementu w zbiorze nieuporządkowanym

Algorytm wyszukiwania elementu w zbiorze nieuporządkowanym zapiszemy w języku Python. Elementy zbioru wprowadzimy do listy.

Program ma wyszukiwać w zbiorze nieuporządkowanym daną wprowadzoną przez użytkownika i wyświetlić na ekranie jej pozycję na liście (indeks). Przyjmujemy, że w zbiorze jest pięć elementów.

Zdefiniujemy funkcję `wyszukaj_dana()` z parametrem `wartosc` (rys. 4.).

- Jeśli element zbioru (`a[i]`) jest szukaną daną (zmienna `wartosc`), to funkcja zwraca wartość zmiennej `i`, czyli pozycję danego elementu.
- Jeśli w przeszukiwanym zbiorze nie ma szukanej liczby, funkcja `wyszukaj_dana()` zwraca wartość –1. Programiści często używają wartości niepoprawnych z punktu widzenia algorytmu (w tym przypadku indeks listy nie może być liczbą ujemną) do zasygnalizowania błędu lub niepowodzenia operacji.

Zmiennej `N` przypisujemy liczbę elementów listy.

Definiujemy listę `N`-elementową o nazwie `a` i wyzerowujemy jej elementy.

Wyszukiwanie_nieuporządkowany.py - C:\Python\Wyszukiwanie_nieuporządkowany.py (3.9.0)

File Edit Format Run Options Window Help

```
N = 5
a = [0] * N
def wprowadz_dane():
    for i in range(N):
        a[i] = int(input("Podaj liczbę: "))
def wyprowadz_dane():
    for i in range(N):
        print(a[i])

def wyszukaj_dana(wartosc):
    for i in range(N):
        if a[i] == wartosc:
            return i
    return -1

wprowadz_dane()
print("Wprowadzone dane:")
wyprowadz_dane()
wartosc = int(input("Podaj daną do wyszukania: "))
pozycja = wyszukaj_dana(wartosc)
if pozycja >= 0:
    print("Znaleziono daną", wartosc, "na pozycji", pozycja)
    print("Pozycje liczone są od zera")
else:
    print("Nie znaleziono danej", wartosc)


input("\n\nNaciśnij Enter, aby zakończyć")
```

Funkcja `wprowadz_dane()` wprowadza dane do listy.

Funkcja `wyprowadz_dane()` wyświetla na ekranie dane zapisane w liście.

Funkcja `wyszukaj_dana()` zwraca pozycję szukanego elementu w zbiorze (indeks w liście `a`).

Wywołujemy funkcje `wprowadz_dane()` i `wyprowadz_dane()`.

Wywołujemy funkcję `wyszukaj_dana()`, przypisując wynik zmiennej `pozycja`.

Uwzględniamy również sytuację, gdy podanej wartości nie ma w zbiorze.

Rys. 4. Program w języku Python realizujący algorytm wyszukiwania elementu w zbiorze nieuporządkowanym – ćwiczenie 5.

Ćwiczenie 5. Zapisujemy algorytm wyszukiwania elementu w zbiorze nieuporządkowanym w języku Python

1. Program w języku Python pokazany na rysunku 4. realizuje algorytm wyszukiwania elementu w zbiorze nieuporządkowanym. Przepisz ten program i zapisz go w pliku pod nazwą *Wyszukiwanie_nieuporządkowany*.
2. Uruchom kilkakrotnie program.
3. Sprawdź działanie programu dla różnych danych.

2 Wyszukiwanie elementu w zbiorze uporządkowanym metodą połowienia

Algorytm wyszukiwania przez połowienie jest przykładem metody „dziel i zwyciężaj". Polega ona na dzieleniu przeszukiwanego zbioru na dwie części i zawężeniu przeszukiwania do jednej z tych części. Metoda podziału pomaga szybko znaleźć poszukiwany element, czyli zwyciężyć.

Gra w zgadywanie liczby

Algorytm poszukiwania przez połowienie wykorzystuje się w zabawie w zgadywanie liczby. Zabawa polega na odgadywaniu liczby wybranej z określonego zakresu. Osoba zgadująca podaje liczbę, a druga osoba udziela odpowiedzi: „za mała" lub „za duża". Osoba zgadująca dzieli zbiór liczb, odrzucając część, w której są liczby większe lub mniejsze od ostatnio podanej, aż odgadnie właściwą liczbę.

Ćwiczenie 6. Gramy w zgadywanie liczby

Zagraj z koleżanką lub z kolegą z klasy w zgadywanie liczby. Przyjmijcie zbiór liczb naturalnych z zakresu od 1 do 100.

Wskazówka: Jeśli jesteś osobą zgadującą liczbę, najlepiej na początku podać liczbę 50. Jak sądzisz, dlaczego?

Algorytm gry w zgadywanie liczby

Zadanie: Odgadnij liczbę w zbiorze liczb naturalnych z zakresu od 1 do 100, stosując algorytm gry w zgadywanie liczby.

Specyfikacja zadania:

Dane: Liczba do odgadnięcia pamiętana w zmiennej *szukana*, liczby naturalne z zakresu od 1 do 100 podawane przez gracza i zapamiętywane kolejno w zmiennej *liczba*.

Wynik: Odgadnięta liczba: *szukana*.

Lista kroków:

1. Zacznij algorytm.
2. Zmiennej *szukana* przypisz liczbę, którą ma odgadnąć gracz.
3. Wprowadź liczbę podawaną przez gracza do zmiennej *liczba*.
4. Jeżeli *liczba* = *szukana*, idź do kroku 7.
5. Jeżeli *liczba* > *szukana*, wyświetl na ekranie napis „Za duża”; w przeciwnym przypadku wyświetl na ekranie napis „Za mała”.
6. Idź do kroku 3.
7. Wyprowadź komunikat „Gratulacje! Odgadnięta liczba to: ” i wartość zmiennej *szukana*.
8. Zakończ algorytm.

Zapisujemy w języku Python algorytm gry w zgadywanie liczby

Algorytm gry w zgadywanie liczby zapiszemy w języku Python. Wartość zmiennej `szukana` możemy wprowadzić z klawiatury (stosując funkcję `input()`) lub wygenerować losowo (stosując funkcję `randint()`. W pierwszym przypadku gramy z wybraną osobą, w drugim – z komputerem.

Funkcja `randint(x, y)` generuje całkowitą liczbę losową należącą do przedziału obustronnie domkniętego, wyznaczonego przez dwie podane wartości: ⟨`x, y`⟩.

Na przykład:
`randint(-20, 20)` – wygeneruje losowo liczbę całkowitą z przedziału ⟨-20, 20⟩,
`randint(1, 100)` – wygeneruje losowo liczbę całkowitą z przedziału ⟨1, 100⟩.

Zgadnij_liczbę.py - C:\Python\Zgadnij_liczbę.py (3.9.0)

File Edit Format Run Options Window Help

```
from random import randint

szukana = randint(1, 100)

liczba = int(input("Wprowadź liczbę od 1 do 100: "))

while liczba != szukana:
    if liczba > szukana:
        print("Za duża")
    else:
        print("Za mała")
    liczba = int(input("Wprowadź liczbę od 1 do 100: "))

print("Gratulacje! Odgadnięta liczba to:", szukana)
```

Dołączamy bibliotekę, w której zdefiniowano funkcję `randint()`.

Zmiennej `szukana` przypisujemy liczbę z przedziału ⟨1, 100⟩ wygenerowaną losowo przez komputer.

Komputer wykonuje instrukcje w pętli, dopóki wartość zmiennej liczba jest różna od wartości zmiennej `szukana`.

Rys. 5. Program w języku Python realizujący algorytm gry w zgadywanie liczby – ćwiczenie 7.

W programie pokazanym na rysunku 5. zastosowaliśmy losowe generowanie wartości zmiennej `szukana`. Do zapisania iteracji użyliśmy instrukcji `while`, ponieważ szukamy liczby, dopóki wprowadzana przez nas z klawiatury liczba nie będzie równa liczbie wygenerowanej przez komputer.

Ćwiczenie 7. Tworzymy w języku Python grę w zgadywanie liczby

1. Program w języku Python pokazany na rysunku 5. realizuje algorytm gry w zgadywanie liczby w zbiorze liczb naturalnych z zakresu ⟨1; 100⟩. Przepisz ten program i zapisz go w pliku pod nazwą *Zgadnij_liczbę*.
2. Uruchom kilkakrotnie program, grając w grę w zgadywanie liczby z komputerem.
3. Objaśnij działanie programu, m.in. uzasadnij użycie poszczególnych instrukcji zgodnie z podaną listą kroków.

Ćwiczenie 8. Modyfikujemy program

1. W programie zapisanym w ćwiczeniu 7. zmień wprowadzanie zmiennej `szukana` tak, aby wartość tej zmiennej była wprowadzana z klawiatury.
2. Zapisz program w pliku pod nazwą *Zgadnij_liczbę_klawiatura*.
3. Uruchom kilkakrotnie program, grając w grę w zgadywanie liczby z drugą osobą.
4. Oceń grę z użyciem każdego z programów. Uzasadnij odpowiedź.

Algorytm wyszukiwania elementu w zbiorze uporządkowanym metodą połowienia

W algorytmie wyszukiwania elementu w zbiorze uporządkowanym szukamy danego elementu i równocześnie jego położenia w tym zbiorze.

Opis algorytmu: Dany jest uporządkowany w kolejności od najmniejszej do największej zbiór dziesięciu liczb (rys. 6.). Poszukiwana liczba to 40 (zaznaczona na szaro). Jeżeli w zbiorze jest kilka takich samych wartości, algorytm znajdzie pozycję jednego z tych elementów (nie jest określone którego).

Dzielimy zbiór liczb na dwie części, rozdzielone środkowym elementem. Zaczynamy od sprawdzenia tego elementu (liczby zaznaczonej na rys. 6. pomarańczową strzałką). W przypadku, gdy liczba elementów zbioru jest parzysta, spośród dwóch środkowych elementów (w naszym przypadku liczb 57 i 66) jako „środkowy" wybieramy ten, który jest pierwszy w kolejności (czyli 57).

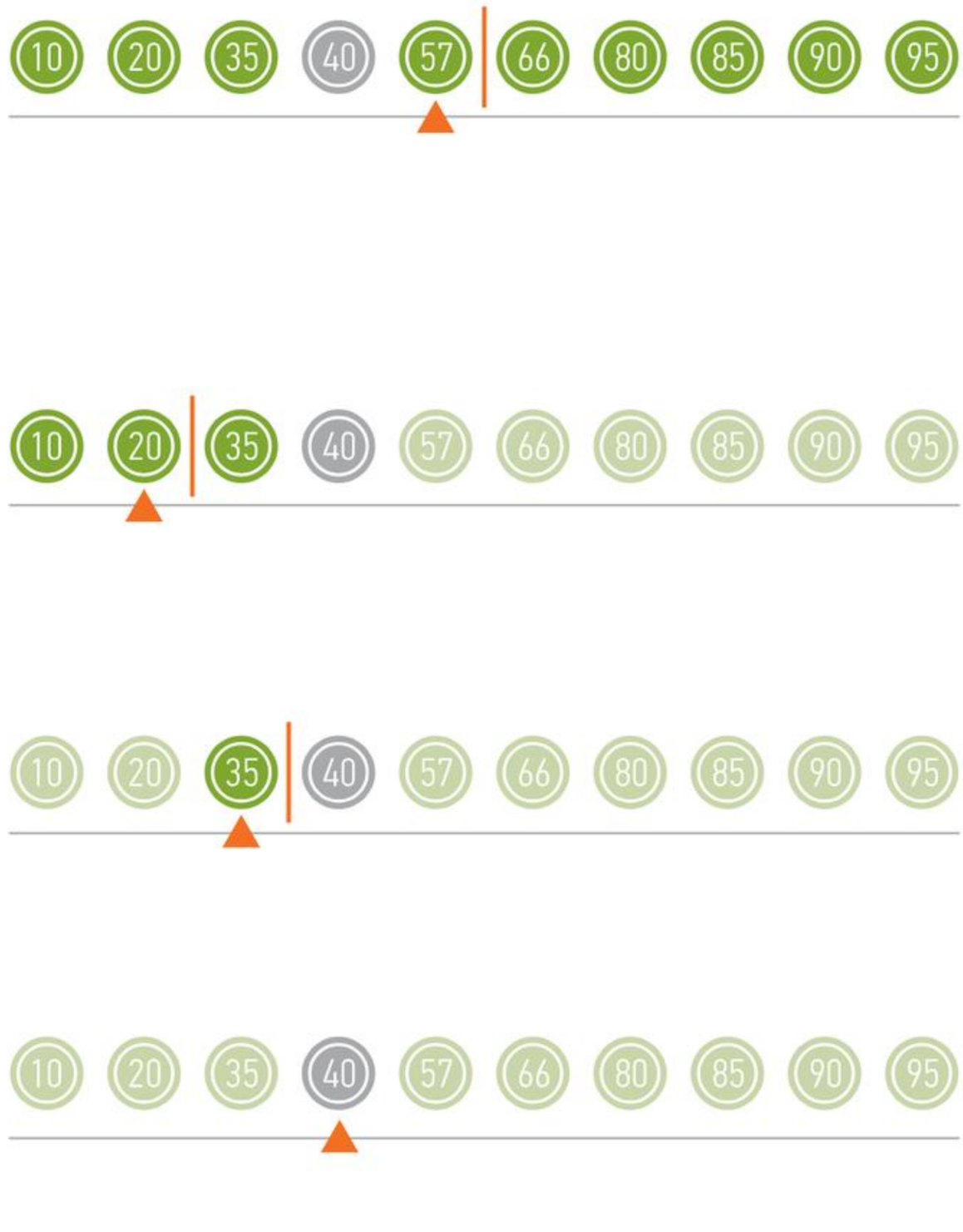
Jeśli środkowy element jest szukaną liczbą, kończymy wyszukiwanie. Jeśli środkowy element jest większy od poszukiwanej liczby, przeszukujemy podzbiór na lewo od elementu środkowego. Jeśli środkowy element jest mniejszy od poszukiwanej liczby, przeszukujemy podzbiór na prawo od elementu środkowego.

Szukamy liczby 40. „Środkowym" elementem zbioru jest liczba 57 i jest ona większa od 40. Dalej przeszukujemy tylko lewą część zbioru, a odrzucamy prawą oraz element środkowy 57.

Liczba elementów w przeszukiwanej części zbioru jest również parzysta (wynosi 4), więc jako „środkowy" wybieramy liczbę 20. Liczba 20 jest mniejsza od 40, więc odrzucamy lewą część zbioru razem z liczbą 20.

Pozostały dwie liczby. „Środkowym" elementem jest liczba 35. Jest ona mniejsza od 40, czyli ją odrzucamy.

Pozostał nam jeden element i jest to szukana liczba 40, która jest czwartą wartością w zbiorze.



Rys. 6. Przykład wyszukiwania liczby w zbiorze uporządkowanym metodą połowienia

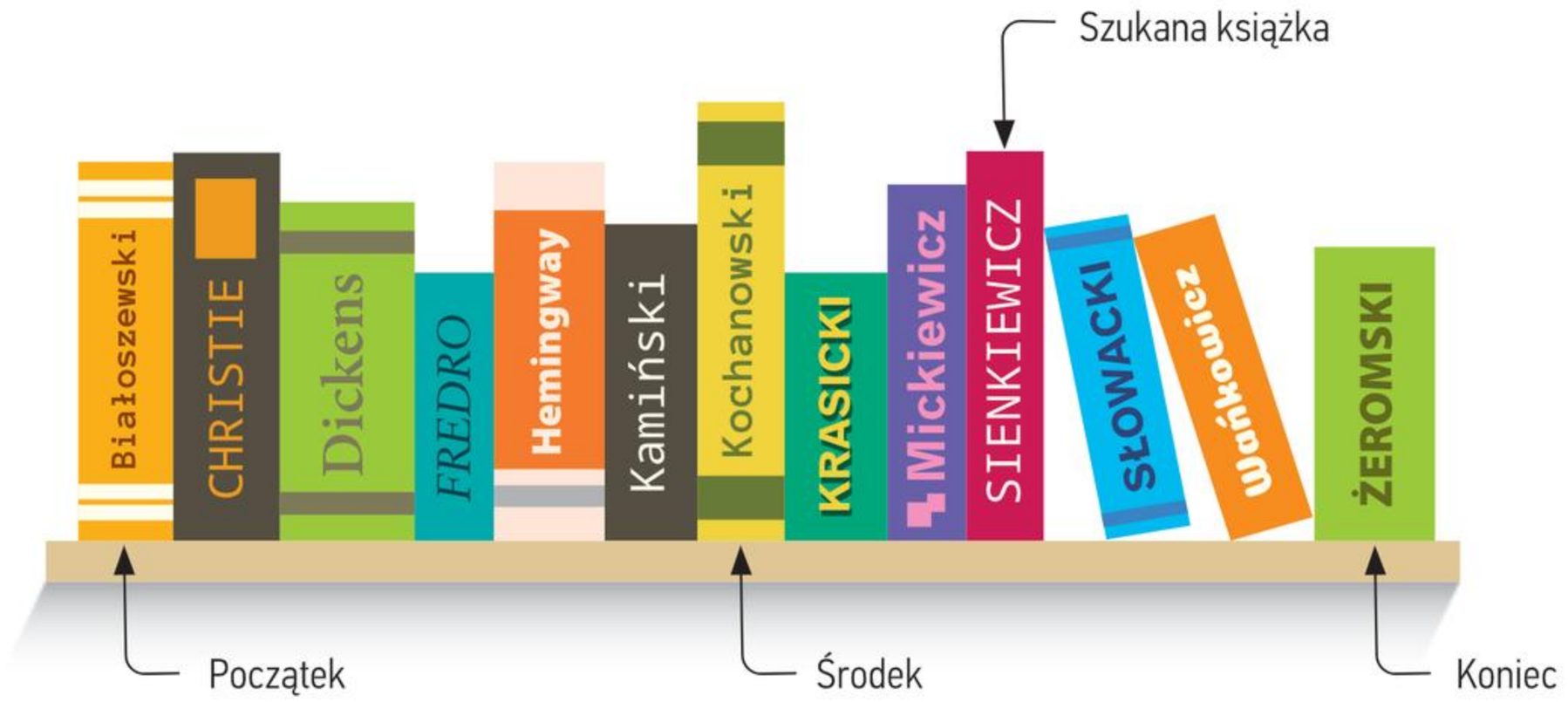
Ćwiczenie 9. Sprawdzamy działanie algorytmu wyszukiwania liczby w zbiorze uporządkowanym metodą połowienia

Przygotuj odpowiednie pomoce dydaktyczne (rys. 6.) i przedstaw wspólnie z koleżankami i kolegami z klasy algorytm wyszukiwania liczby w zbiorze uporządkowanym metodą połowienia.

Wskazówka: Możesz zapisać liczby na kartkach i układać je na stole lub podłodze.

Z wyszukiwaniem przez połowienie spotykamy się w życiu codziennym, np. gdy szukamy jakiejś rzeczy w zbiorze, w którym panuje ustalony wcześniej porządek.

Wyobraźmy sobie półkę z książkami, które ułożono alfabetycznie według nazwisk autorów, zaczynając od lewej strony (rys. 7.). Chcemy znaleźć książkę autora, którego nazwisko zaczyna się na literę „S". Czy przeglądamy półkę od samego początku? Raczej nie. Zwykle odruchowo postępujemy zgodnie z algorytmem wyszukiwania przez połowienie – sprawdzamy, czy szukanej książki nie ma pośrodku. Jeśli trafimy na inną książkę, np. autora na literę „K", to nie przeszukujemy lewej połowy półki, tylko będziemy szukać po prawej stronie.



Rys. 7. Zastosowanie algorytmu wyszukiwania przez połowienie – szukanie książki na półce

Zapisujemy w języku Python algorytm wyszukiwania metodą połowienia

Algorytm wyszukiwania elementu w zbiorze uporządkowanym metodą połowienia zapiszemy w języku Python. Elementy zbioru wprowadzimy do listy. Zakładamy, że wprowadzamy dane od najmniejszej do największej (inaczej program nie będzie działał poprawnie). Bez takiego założenia należałoby dodać również funkcję sortowania danych (ustawiania w określonym porządku).

Program ma wyszukiwać w zbiorze uporządkowanym daną podaną przez użytkownika i wyświetlić na ekranie jej pozycję na liście (indeks). Przyjmujemy, że w zbiorze jest pięć elementów.

Zdefiniujemy funkcję `znajdz_dana()` z parametrem `wartosc` (rys. 8.).

- W funkcji wyznaczamy pozycję elementu środkowego `srodek = (początek + koniec) // 2`.
- Jeśli element środkowy (`a[srodek]`) jest szukaną daną (zmienna `wartosc`), to funkcja zwraca wartość zmiennej `srodek`, czyli pozycję danego elementu;
  w przeciwnym wypadku:
  jeśli szukana dana jest mniejsza od danej środkowej, odrzucamy prawą połowę zbioru (liczby większe od elementu środkowego lub mu równe) – ustalamy nowy koniec: `koniec = srodek - 1`;
  w przeciwnym wypadku odrzucamy lewą połowę zbioru (liczby mniejsze od elementu środkowego lub mu równe) i ustalamy nowy początek: `poczatek = srodek + 1`.
- Połowienie powtarzamy, dopóki początek zbioru jest mniejszy od końca zbioru lub mu równy (`poczatek <= koniec`) – o ile wcześniej nie znajdziemy szukanego elementu (wtedy komputer wykonana instrukcję `return srodek`).
- Jeśli w przeszukiwanym zbiorze nie ma szukanej liczby, funkcja `znajdz_dana()` zwraca wartość –1.

Wyszukiwanie_uporządkowany.py - C:\Python\Wyszukiwanie_uporządkowany.py (3.9.0)

File Edit Format Run Options Window Help

```
N = 5
a = [0] * N
def wprowadz_dane():
    for i in range(N):
        a[i] = int(input("Podaj liczbę: "))

def znajdz_dana(wartosc):
    poczatek = 0
    koniec = N - 1
    while poczatek <= koniec:
        srodek = (poczatek + koniec) // 2
        if a[srodek] == wartosc:
            return srodek
        else:
            if wartosc < a[srodek]:
                koniec = srodek - 1
            else:
                poczatek = srodek + 1
    return -1

wprowadz_dane()
wartosc = int(input("Podaj daną do wyszukania: "))
pozycja = znajdz_dana(wartosc)
if pozycja >= 0:
    print("Znaleziono wartość", wartosc, "na pozycji", pozycja)
    print("Pozycje liczone są od zera")
else:
    print("Nie znaleziono wartości", wartosc)

input("\n\nNaciśnij Enter, aby zakończyć")
```

Zmiennej N przypisujemy liczbę elementów listy.

Definiujemy listę N-elementową o nazwie a i wyzerowujemy jej elementy.

Wykorzystujemy utworzoną wcześniej funkcję wprowadz_dane() do wprowadzenia danych do listy.

Funkcja znajdz_dana(), która zwraca pozycję szukanego elementu w zbiorze (indeks na liście a).

Wywołujemy funkcję wprowadzania danych wprowadz_dane().

Wywołujemy funkcję znajdz_dana(), przypisując wynik zmiennej pozycja.

Uwzględnimy również sytuację, gdy podanej wartości nie ma w zbiorze.

Rys. 8. Program w języku Python realizujący algorytm wyszukiwania elementu w zbiorze uporządkowanym – ćwiczenie 10.

Ćwiczenie 10. Zapisujemy w języku Python algorytm wyszukiwania elementu w zbiorze uporządkowanym

1. Program w języku Python pokazany na rysunku 8. realizuje algorytm wyszukiwania elementu w zbiorze uporządkowanym. Przepisz ten program i zapisz go w pliku pod nazwą *Wyszukiwanie_uporządkowany*.
2. Uruchom kilkakrotnie program.
3. Sprawdź działanie programu dla różnych danych (pamiętaj o wprowadzaniu danych uporządkowanych – od najmniejszej do największej).

Zapamiętaj

- Wybrany element w zbiorze nieuporządkowanym możemy znaleźć, korzystając z algorytmu przeszukiwania liniowego.
- Wybrany element w zbiorze uporządkowanym możemy znaleźć, korzystając z algorytmu poszukiwania przez połowienie, który jest przykładem realizacji metody „dziel i zwyciężaj”.
- Definiowanie funkcji i stosowanie list możemy wykorzystać do zapisania w języku Python algorytmu wyszukiwania elementu w zbiorze nieuporządkowanym i algorytmu wyszukiwania elementu w zbiorze uporządkowanym.

Serwis dla ciekawskich

Angielski matematyk Alan Turing opracował w 1936 roku model teoretyczny maszyny do wykonywania algorytmów. Maszyna składała się z nieskończenie długiej taśmy, zawierającej komórki z przetwarzanymi symbolami, głowicy odczytująco-zapisującej i układu sterowania nią. Obliczenia wykonywane za pomocą tej maszyny zależały od układu symboli wpisanych na taśmie oraz od przyjętego zestawu instrukcji.

Podobnie działają dzisiejsze komputery – wyniki obliczeń zależą od danych zapisanych w pamięci komputera i od zestawu wykonanych instrukcji.

OKO W OKO Z MONITOREM

Pytania i polecenia

1. Wyjaśnij, w jaki sposób znajduje się największą spośród siedmiu liczb w zbiorze nieuporządkowanym.
2. Dlaczego w programie na rysunku 2. wprowadzaną wartość przypisujemy najpierw zmiennej `maks`?
3. Jakie może być zastosowanie algorytmu wyszukiwania przez połowienie? Pokaż na przykładzie.
4. Dlaczego w programie realizującym algorytm wyszukiwania elementu w zbiorze uporządkowanym najpierw sprawdzamy, czy środkowy element jest szukaną liczbą?

Zadania

1. Na podstawie listy kroków pokazanej w punkcie 1. tego tematu opracuj listę kroków algorytmu znajdowania najmniejszej liczby spośród dziesięciu liczb. Sprawdź działanie algorytmu dla danych: 5, 78, 15, 0, 1, 100, 155, 50, 22, 30.
2. Zdefiniuj funkcję `minimum()` wyszukującą najmniejszy element w zbiorze `n`-elementowym. Wywołaj funkcję w programie głównym. Zapisz program w pliku pod nazwą *Minimum_n*.
3. Dany jest uporządkowany zbiór liczb: 5, 8, 15, 20, 25, 30, 35, 70, 88, 90, 100, 120. Poszukiwana liczba to 70. Przedstaw na rysunku (podobnym do rys. 6.) sposób znalezienia liczby 70.

DLA ZAINTERESOWANYCH

1. Opracuj listę kroków algorytmu wyszukiwania przez połowienie. Skorzystaj z opisu metody podanego w punkcie 2. tego tematu.
2. Zdefiniuj funkcję `maksimum_lista()` tak, aby znajdowała maksymalną wartość na liście liczb. Zapisz program w pliku pod nazwą *Liczba_maksymalna*.

2.5. Algorytmy porządkowania

Warto powtórzyć

1. Na czym polega algorytm wyszukiwania największego elementu w zbiorze nieuporządkowanym? Wyjaśnij na przykładzie pięciu elementów.
2. W jaki sposób definiujemy listę w języku Python?
3. Jakie instrukcje stosujemy do zapisywania iteracji w języku Python?

Czasem porządkujemy (w terminologii informatycznej – sortujemy) zbiory przedmiotów, np. ustawiamy książki na półce – od największej do najmniejszej lub alfabetycznie według nazwisk autorów, ustalamy kolejność wyświetlania plików według nazw lub dat utworzenia. Na danych uporządkowanych (posortowanych) szybciej wykonuje się różne operacje (np. wyszukiwanie). Istnieją różne algorytmy porządkowania. Zapiszemy je w języku programowania Python.

W tym temacie:

1. poznasz i zaprogramujesz algorytm porządkowania elementów zbioru metodą przez wybieranie,
2. poznasz i zaprogramujesz algorytm porządkowania elementów zbioru metodą przez zliczanie.

1 Porządkowanie elementów zbioru metodą przez wybieranie

Porządkowanie zbioru liczb metodą przez wybieranie polega na wyszukiwaniu w zbiorze liczby największej (dla porządku malejącego) lub najmniejszej (dla porządku rosnącego), zamienieniu jej miejscami z liczbą, która jest aktualnie na początku zbioru, a następnie powtarzaniu tych czynności z pominięciem liczb już uporządkowanych.

Aby wyszukać największą lub najmniejszą liczbę w zbiorze, stosujemy odpowiednio algorytm wyszukiwania największej lub najmniejszej liczby ze zbioru.

Rys. 1. Przykład porządkowania liczb metodą przez wybieranie – od największej do najmniejszej

Algorytm porządkowania metodą przez wybieranie wykorzystamy do ustawienia uczniów od najwyższego do najniższego.

Algorytm porządkowania elementów zbioru metodą przez wybieranie

Zadanie: Uporządkuj pięcioro uczniów (koleżanek i kolegów z klasy) według wzrostu (od ucznia najwyższego do najniższego), stosując sortowanie przez wybieranie.

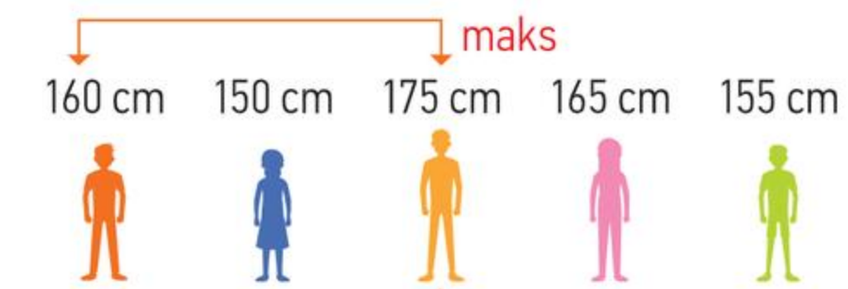
Specyfikacja zadania:

Dane: Pięcioro uczniów ustawionych w przypadkowej kolejności.

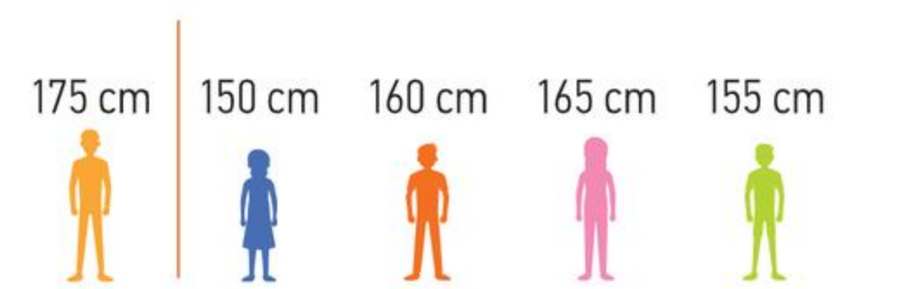
Wynik: Pięcioro uczniów ustawionych w kolejności od najwyższego do najniższego.

Lista kroków:

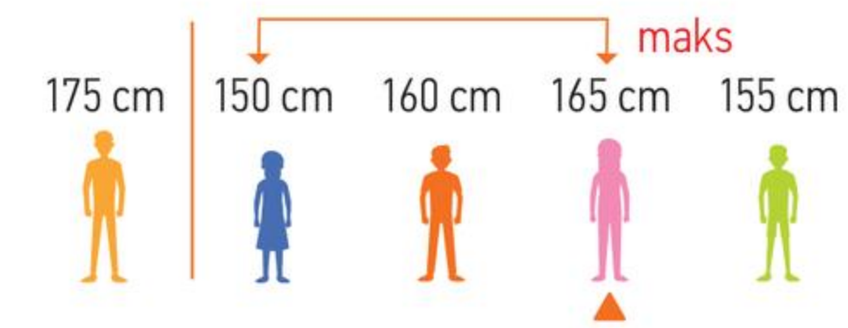
1. Zacznij algorytm.
2. Znajdź najwyższego ucznia. Zapamiętaj, na którym miejscu stoi.
3. Zamień miejscami ucznia najwyższego z uczniem stojącym na pierwszym miejscu.
4. Powtarzaj kroki 2. i 3. (z pominięciem uczniów już ustawionych na miejscach początkowych), aż uporządkujesz wszystkich uczniów.
5. Zakończ algorytm.



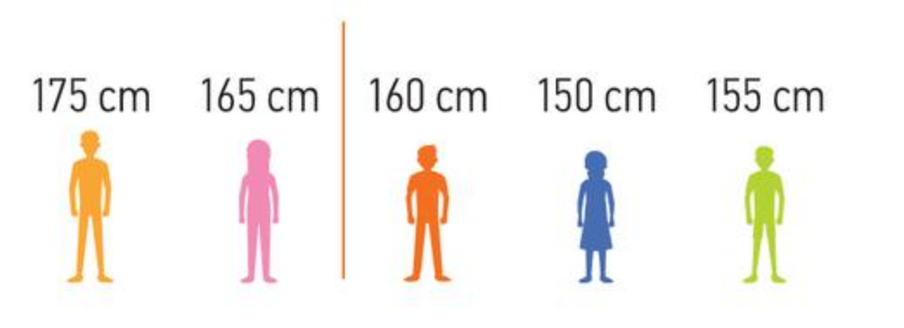
Znaleziono najwyższego ucznia wśród pięciu uczniów.



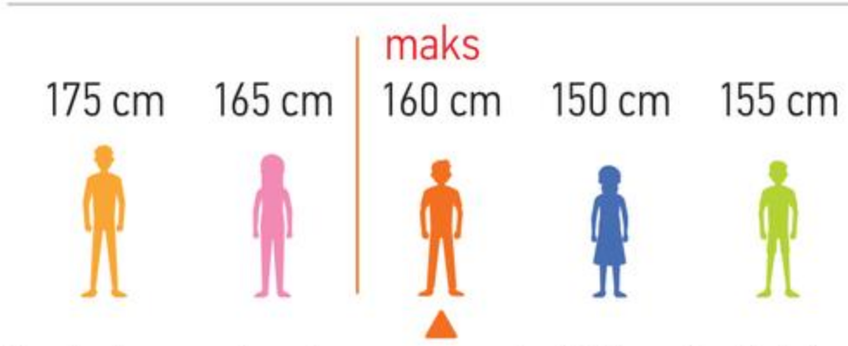
Najwyższy uczeń (175 cm) zamienił się miejscami ze stojącym na pierwszej pozycji (160 cm).



Znaleziono najwyższego ucznia (165 cm) wśród czterech pozostałych uczniów.

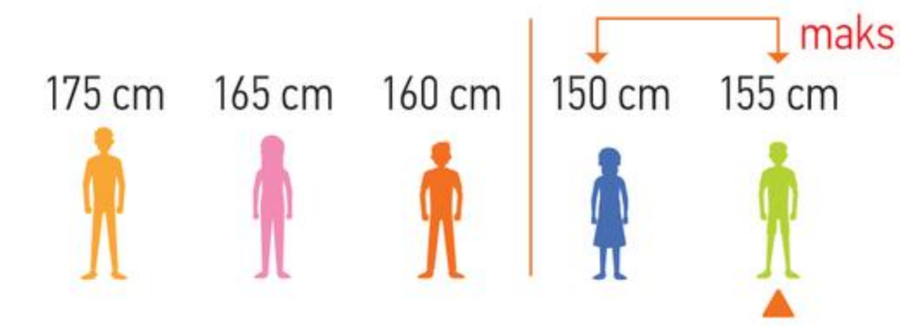


Najwyższy uczeń (165 cm) zamienił się miejscami ze stojącym na pierwszej pozycji (150 cm) wśród czterech pozostałych.



Znaleziono najwyższego ucznia (160 cm) wśród trzech pozostałych uczniów.

Najwyższy uczeń (160 cm) nie musi zmieniać miejsca, ponieważ stoi na właściwym.



Znaleziono najwyższego ucznia (155 cm) wśród dwóch pozostałych uczniów.

Najwyższy uczeń (155 cm) zamienił się miejscami ze stojącym na pierwszej pozycji (150 cm) wśród dwóch pozostałych.

Został ostatni uczeń (150 cm) , który stoi na właściwym miejscu. Wszyscy uczniowie stoją w kolejności od najwyższego do najniższego.

Rys. 2. Porządkowanie pięciorga uczniów według wzrostu metodą przez wybieranie – od najwyższego do najniższego

Ćwiczenie 1. Porządkujemy uczniów według wzrostu metodą przez wybieranie
Wybierz z klasy pięcioro uczniów. Ustaw wybranych uczniów według wzrostu (od najwyższego do najniższego). Postępuj zgodnie z podaną listą kroków.

Ćwiczenie 2. Porządkujemy książki metodą przez wybieranie
Opracuj listę kroków algorytmu porządkowania książek od najgrubszej do najcieńszej.

Zapisujemy algorytm porządkowania elementów zbioru metodą przez wybieranie w języku Python

Zapiszemy w języku Python algorytm porządkowania liczb metodą przez wybieranie od największej do najmniejszej. Przyjmujemy, że jest pięć liczb do uporządkowania.

W programie zdefiniujemy funkcje:

- `wprowadz_dane()` do wprowadzenia danych do listy,
- `wyprowadz_dane()` do wyświetlenia na ekranie uporządkowanych danych,
- `maks_wybor()` (z jednym parametrem `starti`) wyszukującą największy element (`maks`) w zbiorze, począwszy od elementu o indeksie `starti`, i zapamiętującą jego pozycję w zmiennej `maksi` (zwracaną na końcu funkcji),
- `sort_wybor()` (niezwracającą wartości i bez parametrów), w której wywołujemy w pętli funkcję `maks_wybor()` z parametrem aktualnym `i` oraz zamieniamy miejscami znaleziony największy element `maks` z aktualnie pierwszym elementem zbioru. Po każdym przejściu pętli liczba nieuporządkowanych elementów zbioru jest o 1 mniejsza, ponieważ nie sprawdzamy już elementów ustawionych na właściwych miejscach (uporządkowanych).

W programie głównym wywołamy trzy funkcje: wprowadzania danych (`wprowadz_dane()`), porządkowania (`sort_wybor()`) i wyświetlania danych na ekranie (`wyprowadz_dane()`).

Ćwiczenie 3. Zapisujemy w języku Python algorytm porządkowania elementów zbioru metodą przez wybieranie

1. Program w języku Python pokazany na rysunku 3. realizuje algorytm porządkowania zbioru liczb metodą przez wybieranie od największej do najmniejszej dla zbioru pięcioelementowego. Przepisz ten program i zapisz go w pliku pod nazwą *Wybieranie*.
2. Uruchom program. Sprawdź jego działanie dla różnych danych, w tym dla danych pokazanych na rysunkach 1. i 2. Objaśnij działanie programu.

Funkcja wprowadzająca dane do listy

Funkcja wyświetlająca dane na ekranie

Zmiennej `maksi` przypisujemy numer pierwszego elementu nieuporządkowanego zbioru.

Do początkowej wartości zmiennej sterującej `starti` dodajemy 1, ponieważ wartość pierwszego elementu nieuporządkowanego zbioru już wprowadziliśmy do zmiennej `maks`.

W pętli szukamy kolejnych liczb większych od aktualnie największej i zapamiętujemy pozycję największej liczby na liście `maksi = i`. Polecenia w pętli powtarzamy `N - (starti + 1)` razy.

Funkcja zwraca wartość największego elementu `maksi`.

Wywołujemy funkcję `maks_wybor()` z parametrem aktualnym `i`, wynik przypisując zmiennej `maksi`.

Zmienną pomocniczą `t` stosujemy do zamiany miejscami znalezionego największego elementu z aktualnie pierwszym.

Wywołujemy w programie głównym funkcje wprowadzania danych, porządkowania i wyświetlania danych na ekranie.

Wybieranie.py - C:\Python\Wybieranie.py (3.9.0)

File Edit Format Run Options Window Help

```
N = 5
a = [0] * N

def wprowadz_dane():
    for i in range(N):
        a[i] = int(input("Podaj liczbę: "))

def wyprowadz_dane():
    for i in range(N):
        print(a[i])

def maks_wybor(starti):
    maksi = starti
    maks = a[maksi]
    for i in range(starti + 1, N):
        if a[i] > maks:
            maksi = i
            maks = a[maksi]
    return maksi

def sort_wybor():
    for i in range(N-1):
        maksi = maks_wybor(i)
        t = a[i]
        a[i] = a[maksi]
        a[maksi] = t

wprowadz_dane()
sort_wybor()
print("Dane posortowane metodą przez wybieranie")
wyprowadz_dane()

input("\n\nNaciśnij Enter, aby zakończyć")
```

Rys. 3. Program w języku Python realizujący algorytm porządkowania metodą przez wybieranie – ćwiczenie 3.

2 Porządkowanie elementów zbioru metodą przez zliczanie

Metoda porządkowania przez zliczanie polega na zliczaniu wystąpień poszczególnych wartości w zbiorze, np. ile mamy jedynek, dwójek, trójek, czwórek, piątek, …, dwudziestek – w przypadku zbioru liczb całkowitych zawierającego wartości od 1 do 20. Następnie wypisujemy daną wartość tyle razy, ile wynosi liczba jej wystąpień w zbiorze. W ten sposób otrzymujemy uporządkowany zbiór.

Metodę porządkowania przez zliczanie pokażemy na przykładzie porządkowania liczb całkowitych od najmniejszej do największej. Przed rozpoczęciem porządkowania musimy znać najmniejszą i największą wartość w zbiorze. Wartości tych szukamy, stosując algorytm wyszukiwania elementu najmniejszego i największego w zbiorze nieuporządkowanym.

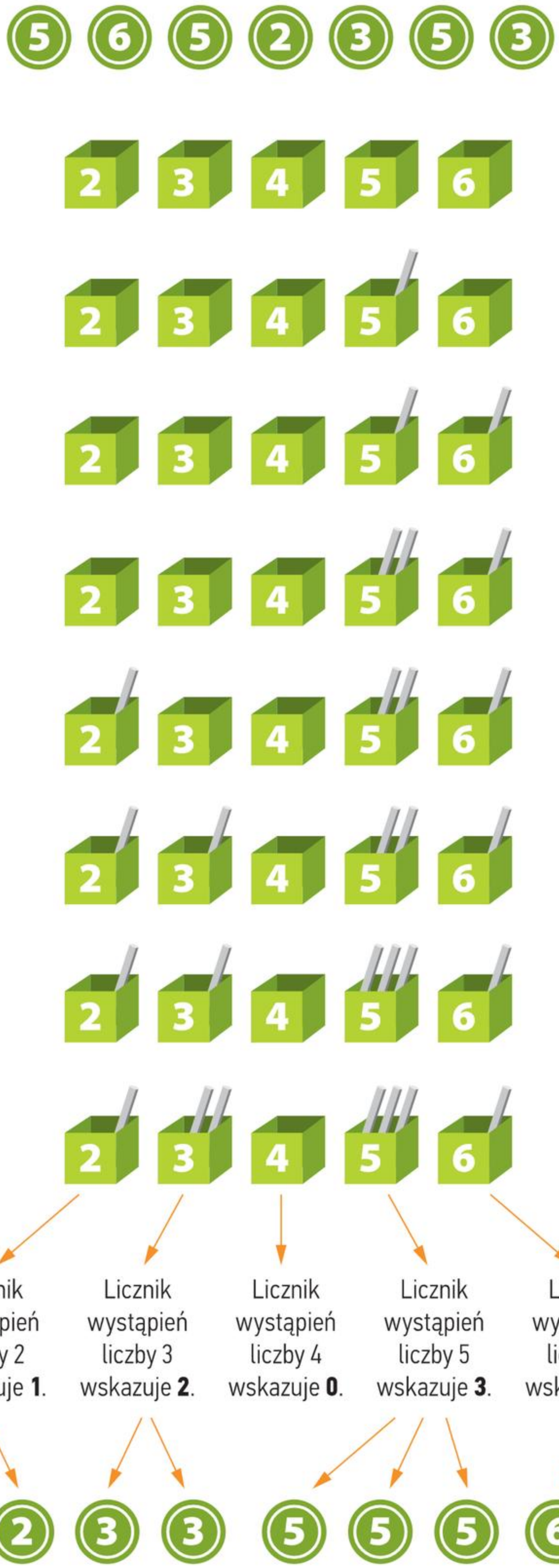
Algorytm porządkowania elementów zbioru metodą przez zliczanie (na przykładzie siedmiu liczb)

Opis algorytmu: Dany jest zbiór liczb {5, 6, 5, 2, 3, 5, 3}, który zawiera liczby całkowite z przedziału od 2 do 6 (czyli najmniejszą liczbą jest 2, a największą – 6).

Do zliczania wystąpień wszystkich liczb z tego zakresu musimy przygotować pięć liczników do zliczania wystąpień liczb: 2, 3, 4, 5 i 6. Na rysunkach 4. i 5. liczniki mają formę pudełek. Na początku wszystkie pudełka liczników są puste (mają wartość 0), ale gdy zaczniemy przeglądać zbiór, wartości liczników będą się zmieniać.

Liczniki przygotowujemy dla wszystkich liczb z przedziału od 2 do 6, również dla tych, których nie ma w zbiorze.

Rys. 4. Pudełka pełniące rolę liczników



Zbiór siedmiu elementów do uporządkowania

Pięć wyzerowanych liczników (tu pustych pudełek), oznaczonych liczbami od 2 do 6.

Pierwszym elementem zbioru jest liczba 5, czyli licznik piątek zwiększamy o jeden.

Drugim elementem zbioru jest liczba 6, czyli licznik szóstek zwiększamy o jeden.

Trzecim elementem zbioru jest liczba 5, czyli licznik piątek zwiększamy o jeden.

Czwartym elementem zbioru jest liczba 2, czyli licznik dwójek zwiększamy o jeden

Piątym elementem zbioru jest liczba 3, czyli licznik trójek zwiększamy o jeden.

Szóstym elementem zbioru jest liczba 5, czyli licznik piątek zwiększamy o jeden.

Siódmym elementem zbioru jest liczba 3, czyli licznik trójek zwiększamy o jeden.

Otrzymujemy następujące wartości kolejnych liczników.

Wypisujemy daną liczbę tyle razy, ile wskazuje wartość jej licznika i otrzymujemy zbiór liczb uporządkowany od najmniejszej do największej.

Rys. 5. Przykład porządkowania liczb metodą przez zliczanie

Ćwiczenie 4. Porządkujemy liczby od najmniejszej do największej metodą przez zliczanie

Przygotuj odpowiednie pomoce dydaktyczne (pudełka lub kubeczki i patyczki lub ołówki – rys. 5.), a następnie wykonaj wspólnie z koleżankami i kolegami z klasy algorytm porządkowania liczb metodą przez zliczanie dla zbioru składającego się z dziesięciu elementów: {–2, 5, 2, 0, 4, –2, 4, –1, –2, 5}. Liczby uporządkuj od najmniejszej do największej.

Wskazówki:

- Pudełek (kubeczków) ma być tyle, ile jest liczb całkowitych z zakresu ⟨*najmniejszy*, *największy*⟩, gdzie *najmniejszy* to najmniejsza liczba w zbiorze, a *największy* – największa.
- Zbiór zaczynamy przeglądać od pierwszego elementu i kolejno wrzucamy patyczki (ołówki) do kubeczków oznaczonych daną liczbą (jak pokazano na rysunku 5.).

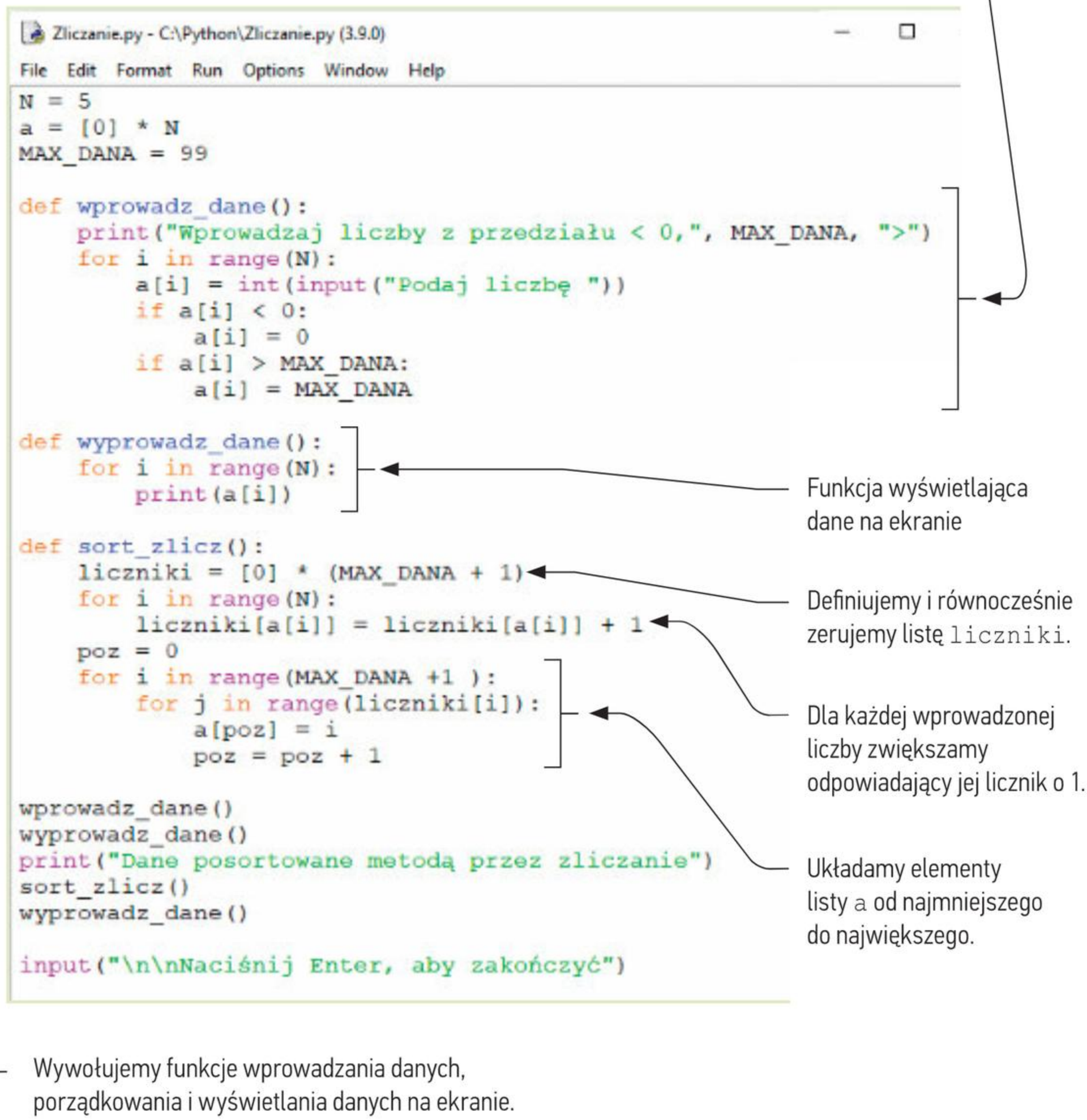
Aby zbiór liczb był uporządkowany od największej do najmniejszej, wystarczy podczas wypisywania zbioru wartości liczników przejrzeć go w odwrotnej kolejności – z prawej strony do lewej.

Zapisujemy algorytm porządkowania elementów zbioru metodą przez zliczanie w języku Python

Zapiszemy w języku Python algorytm porządkowania liczb metodą przez zliczanie od najmniejszej do największej. Przyjmujemy, że jest siedem liczb do uporządkowania.

W programie wykorzystamy funkcje:

- `wprowadz_dane()` do wprowadzenia danych do listy. Dodamy w funkcji sprawdzanie, czy liczba należy do określonego zakresu. Ustalimy zakres ⟨`0, MAX_DANA`⟩. Zmiennej `MAX_DANA` przypiszemy stałą wartość, np. 99. Dzięki temu będziemy mogli zdefiniować listę, która będzie przechowywała liczbę wystąpień każdej liczby z przedziału od 0 do 99,
- `wyprowadz_dane()` do wyświetlenia danych na ekranie po uporządkowaniu,
- `sort_zlicz()`, w której będziemy wykorzystywać listę `liczniki[MAX_DANA + 1]`, służącą do zliczania wystąpień elementów zbioru. Liczba liczników, zgodnie z algorytmem, musi umożliwić zliczenie każdej liczby z danego zakresu, więc przygotowujemy 100 liczników (dla liczb od 0 do 99).



Rys. 6. Program w języku Python realizujący algorytm porządkowania metodą przez zliczanie – ćwiczenie 5.

W pętli `for` (rys. 6., wiersze 20–21) dla każdej liczby wprowadzonej przez użytkownika zwiększamy odpowiadający jej licznik o 1.

W pętli `for` (rys. 6., wiersze 24–26) przeglądamy listę `liczniki` i wpisujemy do listy z danymi (`a`) daną wartość tyle razy, ile razy pojawiła się w zbiorze (liczbie wystąpień odpowiada element listy `liczniki`).

Pętlę z wierszy 24–26 komputer wykonuje dla każdej możliwej wartości danych wejściowych (rys. 6., pętla `for` – wiersz 23.), mamy tu zatem do czynienia z zagnieżdżeniem pętli. Zmienna `poz` określa aktualną pozycję na liście `a`.

W programie głównym wywołamy trzy funkcje: wprowadzania danych (`wprowadz_dane()`), porządkowania (`sort_zlicz()`) i wyświetlania danych na ekranie (`wyprowadz_dane()`).

Na rysunku 7. pokazano stan listy `liczniki` dla wprowadzonych liczb {1, 1, 2, 4, 5}.

liczniki[0]	liczniki[1]	liczniki[2]	liczniki[3]	liczniki[4]	liczniki[5]
0	2	1	0	1	1

Rys. 7. Wartości elementów listy liczniki dla liczb {1, 1, 2, 4, 5}

Ćwiczenie 5. Zapisujemy w języku Python algorytm porządkowania elementów zbioru metodą przez zliczanie

1. Program w języku Python pokazany na rysunku 6. realizuje algorytm porządkowania liczb metodą przez zliczanie od najmniejszej do największej. Przepisz ten program i zapisz go w pliku pod nazwą *Zliczanie*.
2. Uruchom program. Sprawdź jego działanie dla różnych danych, w tym dla danych pokazanych na rysunku 5. i podanych w ćwiczeniu 4.
3. Objaśnij działanie programu.

Zapamiętaj

- Zbiór elementów możemy uporządkować, stosując algorytmy porządkowania, np. metodą przez wybieranie lub zliczanie.
- W porządkowaniu metodą przez wybieranie wykorzystujemy algorytm wyszukiwania elementu (najmniejszego lub największego) w zbiorze nieuporządkowanym.
- Definiowanie funkcji i stosowanie list możemy wykorzystać do zapisania w języku Python algorytmów porządkowania metodą przez wybieranie i porządkowania metodą przez zliczanie.

OKO W OKO Z MONITOREM

Pytania i polecenia

1. Wskaż przynajmniej dwie różnice między porządkowaniem metodą przez wybieranie a metodą przez zliczanie.
2. Dlaczego w programie realizującym algorytm porządkowania metodą przez wybieranie musimy zapamiętywać pozycję znalezionego największego elementu w oddzielnej zmiennej?
3. Dlaczego w porządkowaniu metodą przez zliczanie należy przygotować liczniki dla wszystkich liczb z danego zakresu, nawet jeśli niektóre liczniki pozostaną zerowe?
4. Po co potrzebna jest lista `liczniki` w programie realizującym algorytm porządkowania metodą przez zliczanie?

Zadania

1. Napisz listę kroków sortowania metodą przez wybieranie dla *n* elementów.
2. Dany jest zbiór liczb: {12, 3, 8, 2,12, 5, 8, 4, 10, 4}. Przedstaw na rysunku (podobnym do rys. 5.) uporządkowanie tego zbioru metodą przez zliczanie. Ile liczników musisz przygotować? Ile liczników pozostało pustych?
3. Zmodyfikuj program *Wybieranie* (zapisany w ćwiczeniu 3.), aby umożliwił wprowadzenie dziesięciu elementów. Zapisz program w pliku pod nazwą *Wybieranie_10*.

DLA ZAINTERESOWANYCH

1. Zmodyfikuj program *Wybieranie* (zapisany w ćwiczeniu 3.), aby porządkował wprowadzone dane od najmniejszej do największej. Zapisz program w pliku pod nazwą *Wybieranie_od_najmniejszej*.
2. Zmodyfikuj program *Zliczanie* (zapisany w ćwiczeniu 5.), aby porządkował wprowadzone dane od największej do najmniejszej. Zapisz program w pliku pod nazwą *Zliczanie_od_największej*.

Programowanie w języku C++

.cpp

3.1. Wprowadzenie do programowania w języku C++

Na lekcjach matematyki i fizyki większość zadań rozwiązujemy według pewnych schematów: wypisujemy dane, precyzujemy, które wartości należy obliczyć, czyli określamy szukane, wybieramy metodę rozwiązania, wypisujemy wzory łączące dane z szukanymi, przeprowadzamy obliczenia, formułujemy odpowiedź. Rozwiązanie wybranego zadania przedstawiamy, używając wzorów, symboli oraz komentarzy słownych. W zadaniach na lekcjach informatyki postępujemy podobnie, przedstawiając rozwiązanie np. graficznie lub w postaci programu komputerowego. W tym temacie:

1. dowiesz się, czym jest algorytm i poznasz sposoby jego przedstawiania,
2. dowiesz się, na czym polega programowanie,
3. poznasz środowisko programistyczne Dev-C++,
4. napiszesz pierwszy program w języku C++.

1 Sposoby przedstawiania algorytmów

Algorytm to uporządkowany i uściślony sposób rozwiązania danego problemu, zawierający szczegółowy opis wykonywanych czynności w skończonej liczbie kroków.

Aby rozwiązać dowolny problem (zadanie), nie tylko z informatyki, należy rozpocząć od poprawnego sformułowania problemu oraz ustalenia danych i warunków, które każda z danych musi spełniać (np. czy ma to być tekst, data, liczba). Należy też określić cel, czyli wyniki (szukane), które chcemy otrzymać, a także warunki, jakie powinny one spełniać. Następnie należy zastanowić się nad sposobem rozwiązania, czyli algorytmem, a także wyborem odpowiedniego narzędzia, np. programu komputerowego (rys. 1.).

Rys. 1. Schemat postępowania podczas rozwiązywania problemów za pomocą komputera

Etapy rozwiązywania problemów

W procesie rozwiązywania dowolnego zadania, nie tylko za pomocą komputera, można wyróżnić kilka etapów:

1. Sformułowanie zadania.
2. Określenie danych wejściowych.
3. Ustalenie celu, czyli wyniku.
4. Określenie metody rozwiązania, czyli wybór algorytmu.
5. Przedstawienie algorytmu w wybranej postaci.
6. Analiza poprawności rozwiązania.
7. Testowanie rozwiązania dla różnych danych – ocena efektywności przyjętej metody.

Algorytmy można przedstawiać w postaci:

- opisu słownego,
- listy kroków,
- schematu blokowego,
- programu w wybranym języku programowania.

Różne sposoby przedstawiania algorytmu pokażemy na przykładzie algorytmu obliczania pola trójkąta.

Specyfikacja zadania

Specyfikacja zadania (problemu) to szczegółowy opis zadania, w którym określa się dane wejściowe i oczekiwane wyniki oraz związek między danymi a wynikami (warunki, jakie muszą spełniać wyniki).

Zadanie: Oblicz pole trójkąta o podstawie a i wysokości h.
Specyfikacja zadania:
Dane: Dowolne liczby rzeczywiste dodatnie: a, h (a – długość podstawy trójkąta, h – długość wysokości trójkąta opuszczonej na podstawę).
Wynik: Wartość pola trójkąta: p.

Opis słowny algorytmu

Opis słowny algorytmu to przedstawienie go w postaci ogólnego opisu wykonywanych operacji, bez podawania szczegółów. Może zawierać przemyślenia dotyczące sposobów rozwiązania postawionego zadania.

Opis algorytmu: Pole trójkąta możemy policzyć, mnożąc długość jednego boku trójkąta przez długość wysokości opuszczonej na ten bok i dzieląc wynik przez 2. Ponieważ w naszym zadaniu mowa jest o podstawie trójkąta, przyjmiemy, że jest ona tym bokiem.

Użyte w kroku 3. słowo „przypisz" zapożyczono z języków programowania. Na przykład w języku C++ zapis: `p = a * h / 2` to instrukcja przypisania, w której zmiennej po lewej stronie (czyli `p`) przypisuje się wartość wyrażenia znajdującego się po prawej stronie instrukcji.

Lista kroków algorytmu

Lista kroków algorytmu to przedstawienie algorytmu w kolejnych punktach (krokach). Każdy punkt takiej listy zawiera opis wykonywanej czynności. Kolejność punktów nie może być przypadkowa – musi być zgodna z działaniem algorytmu.

Lista kroków:

1. Zacznij algorytm.
2. Wprowadź wartości liczb a i h.
3. Zmiennej p przypisz wartość wyrażenia $\frac{a \cdot h}{2}$.
4. Wyprowadź wynik: p.
5. Zakończ algorytm.

Ćwiczenie 1. Zapisujemy algorytm w postaci listy kroków

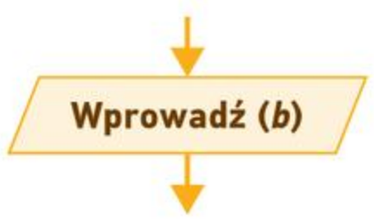
Zapisz specyfikację zadania i listę kroków algorytmu obliczania pola trapezu.

Wskazówka: Wzór na pole trapezu: $p = \frac{(a + b) \cdot h}{2}$, gdzie a, b to długości podstaw trapezu, h – długość wysokości trapezu.

Schemat blokowy algorytmu

W schemacie blokowym poszczególne operacje przedstawia się za pomocą odpowiednio połączonych figur (bloków). Połączenia określają kolejność i sposób wykonania operacji w danym algorytmie.

W tabeli 1. opisano figury, które stosuje się do budowania schematów blokowych.

Reprezentacja graficzna	Opis operacji	Uwagi
	Początek algorytmu	Blok z napisem „Start" zaczyna algorytm. Wychodzi z niego tylko jedno połączenie i żadne do niego nie wchodzi. W schemacie może funkcjonować tylko jeden taki blok.
	Zakończenie algorytmu	Blok z napisem „Koniec" kończy algorytm. Wchodzi do niego jedno połączenie, żadne nie wychodzi. W schemacie może być wiele takich bloków.
	**Wprowadzanie danych** (*blok wejścia*)	Blok z napisem „Wprowadź" służy do wprowadzania danych. Ma jedno połączenie wchodzące i jedno wychodzące. W schemacie może być wiele takich bloków.
	Wyprowadzanie wyników (*blok wyjścia*)	Blok z napisem „Wyprowadź" służy do wyprowadzania wyników. Ma jedno połączenie wchodzące i jedno wychodzące. W schemacie może być wiele takich bloków.

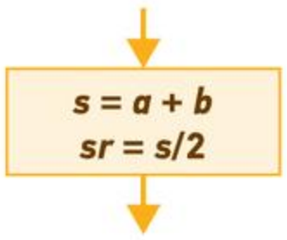
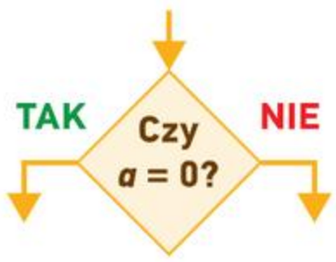
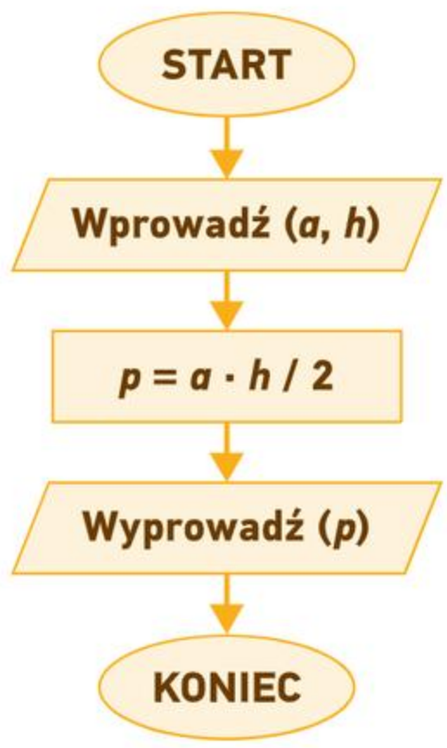
	**Wykonywanie działań** (*blok operacyjny*)	Blok, w którym wykonywane są różne operacje, m.in. obliczenia. Ma jedno połączenie wchodzące i jedno wychodzące. W jednym bloku można wpisać więcej niż jedno wyrażenie. W schemacie może być wiele takich bloków.
	**Sprawdzanie warunku** (*blok warunkowy albo decyzyjny*)	Blok podejmowania decyzji. Wchodzi do niego jedno połączenie, wychodzą dwa: • z napisem „Tak”, gdy warunek jest spełniony; • z napisem „Nie”, gdy warunek nie jest spełniony. W schemacie może być wiele takich bloków.
	Łącznik	Łącznik stosuje się, gdy schemat blokowy rysujemy w kilku częściach, np. na dwóch stronach. Umieszczony wewnątrz numer powinien być taki sam w obu łączonych częściach.
	Połączenie	Połączenie łączy bloki. Tworzy je linia prosta bądź łamana, zakończona strzałką. Połączenie może dochodzić również do innego połączenia.

Tabela 1. Figury geometryczne stosowane w graficznym przedstawianiu algorytmów

Ćwiczenie 2. Sprawdzamy działanie algorytmu

Przetestuj działanie algorytmu pokazanego na rysunku 2. dla następujących liczb: (17; 9), (5; 10), (24; 13).



Rys. 2. Schemat blokowy algorytmu obliczania pola trójkąta

Do rysowania schematów blokowych można wykorzystać **Kształty**, dostępne w programach: Microsoft Word, Microsoft PowerPoint lub Paint. Można również poszukać w internecie darmowych programów, służących do tworzenia schematów blokowych.

Ćwiczenie 3. Budujemy schemat blokowy

Narysuj schemat blokowy algorytmu na podstawie listy kroków utworzonej w ćwiczeniu 1.

Wskazówka: Jeśli wykonujesz zadanie w edytorze tekstu, korzystając z **Kształtów**, pamiętaj o możliwości grupowania wstawionych obiektów.

W algorytmach informatycznych dane wejściowe powinny być precyzyjnie określone, a wszystkie operacje – dokładnie opisane. Dla tych samych danych powinniśmy otrzymać te same wyniki.

Przykład algorytmu nieinformatycznego

W algorytmach nieinformatycznych wynik może zależeć od danych wejściowych i różnych czynników (np. od czasu gotowania). Przykładem algorytmu nieinformatycznego jest przepis na przygotowanie kisielu:

1. Z 0,5 litra zimnej wody odlać połowę do naczynia (np. kubka).
2. Do naczynia z wodą wsypać zawartość torebki.
3. Dobrze wymieszać.
4. Pozostałą część wody zagotować.
5. Dodać trzy łyżki cukru.
6. Do wrzątku wlać rozrobiony proszek.
7. Gotować przez chwilę.
8. Kisiel wlać do salaterek wypłukanych zimną wodą.
9. Podawać gorący lub wystudzony – z konfiturami, sokiem lub śmietaną.

W podanej liście kroków niektóre sformułowania są mało precyzyjne, np. w punkcie 7.: „gotować przez chwilę". Zależnie od tego, ile ta chwila będzie trwała, możemy otrzymać różne wyniki (np. kisiel o różnej konsystencji).

2 Na czym polega programowanie

Aby przedstawić algorytm w postaci **programu komputerowego**, trzeba zapisać go jako ciąg instrukcji języka programowania. Powstaje wówczas tzw. program (kod) źródłowy.

Program komputerowy (kod źródłowy)
Ciąg instrukcji języka programowania, realizujący algorytm.

Język programowania jest specjalnym językiem służącym do pisania programów komputerowych. Jest on zbiorem określonych instrukcji i zasad. Każde słowo lub znak w tym języku (poza komentarzami) jest instrukcją lub jej częścią.

Każda instrukcja programu, podobnie jak figura (blok) w schemacie blokowym, odpowiada określonej operacji. Kolejność występowania w programie instrukcji decyduje o kolejności wykonywania operacji.

Języki programowania możemy podzielić na:

- języki wysokiego poziomu, np. Python, C++, Java, Visual Basic;
- języki niskiego poziomu (wewnętrzne), np. assemblery.

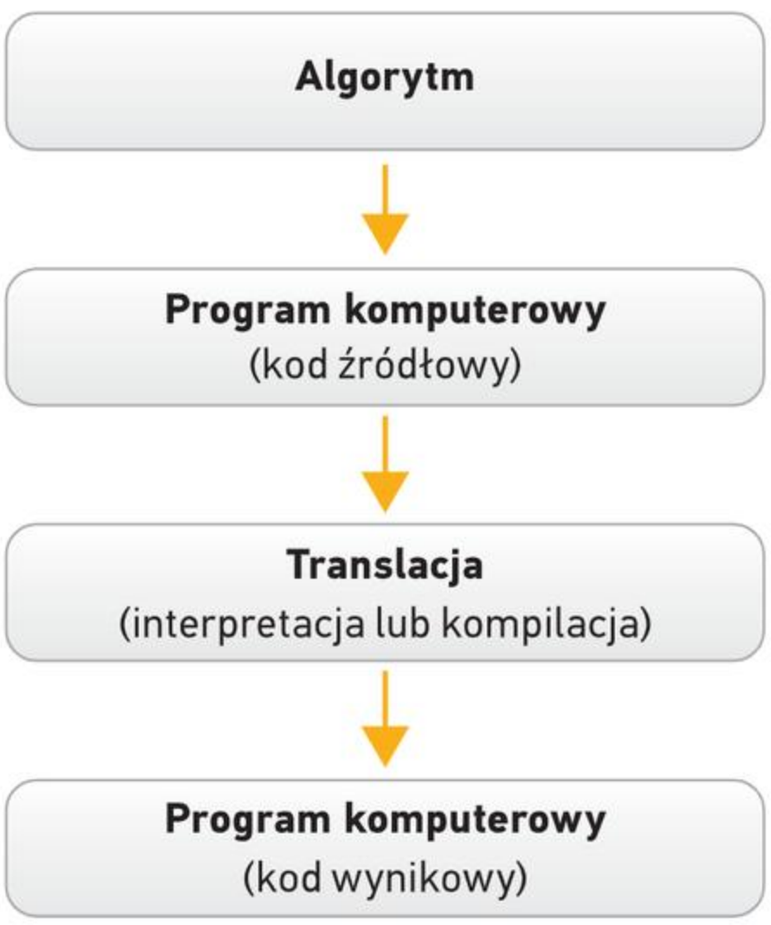
Program napisany w języku wysokiego poziomu (kod źródłowy) jest niezrozumiały dla komputera. Komputer potrafi wykonywać tylko instrukcje (rozkazy) zapisane w języku wewnętrznym, zrozumiałym dla procesora (kod wynikowy).

Proces tłumaczenia programu napisanego w języku programowania wysokiego poziomu na język wewnętrzny komputera nazywamy translacją. Może on przebiegać w formie **kompilacji** lub **interpretacji**.

Tłumaczeniem kodu źródłowego zajmuje się specjalny program, tzw. translator (kompilator lub interpreter). Tłumaczenie kodu łączy się ze sprawdzaniem poprawności zapisanych instrukcji. Jeśli instrukcja jest niepoprawna, to translator jej nie przetłumaczy i wyświetli komunikat o błędzie. Trzeba ją wówczas odszukać, poprawić i powtórnie przeprowadzić translację.

Kompilacja
Przetłumaczenie programu tworzonego w jednym z języków programowania w całości na język zrozumiały dla procesora, by mógł go wykonać komputer. Raz skompilowany program nie wymaga już powtórnej operacji tłumaczenia. Przykładowe języki kompilowane to C++, Pascal.

Interpretacja
Tłumaczenie programu tworzonego w jednym z języków programowania instrukcja po instrukcji, by komputer mógł wykonać każdą z nich. Tłumaczenie następuje każdorazowo przy uruchomieniu programu. Przykładem języków interpretowanych są języki edukacyjne Logo i Scratch, język Python oraz języki wykorzystywane w tworzeniu stron WWW, np. PHP i JavaScript.



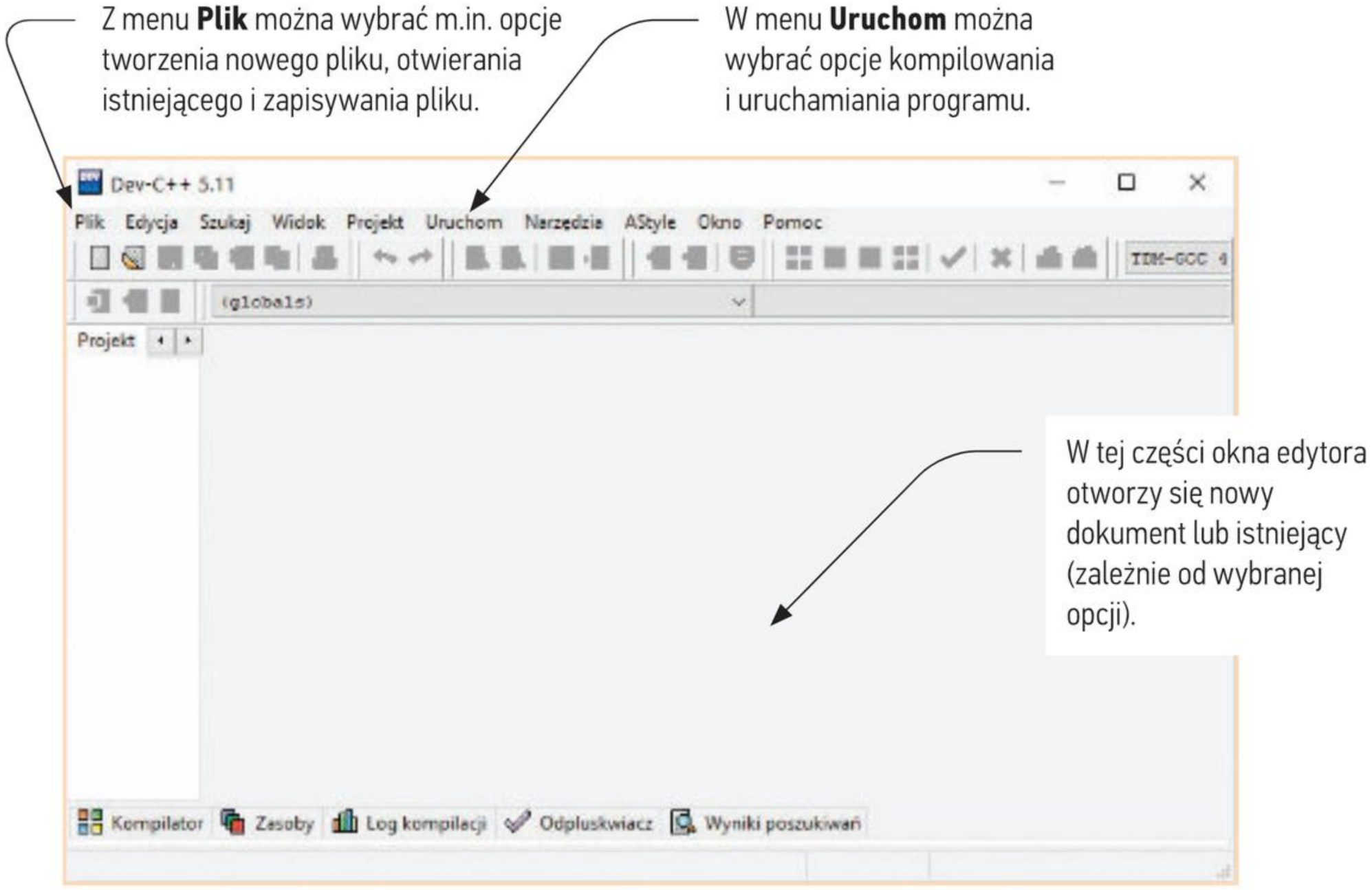
Rys. 3. Etapy powstawania programu komputerowego

3 Środowisko programistyczne języka C++

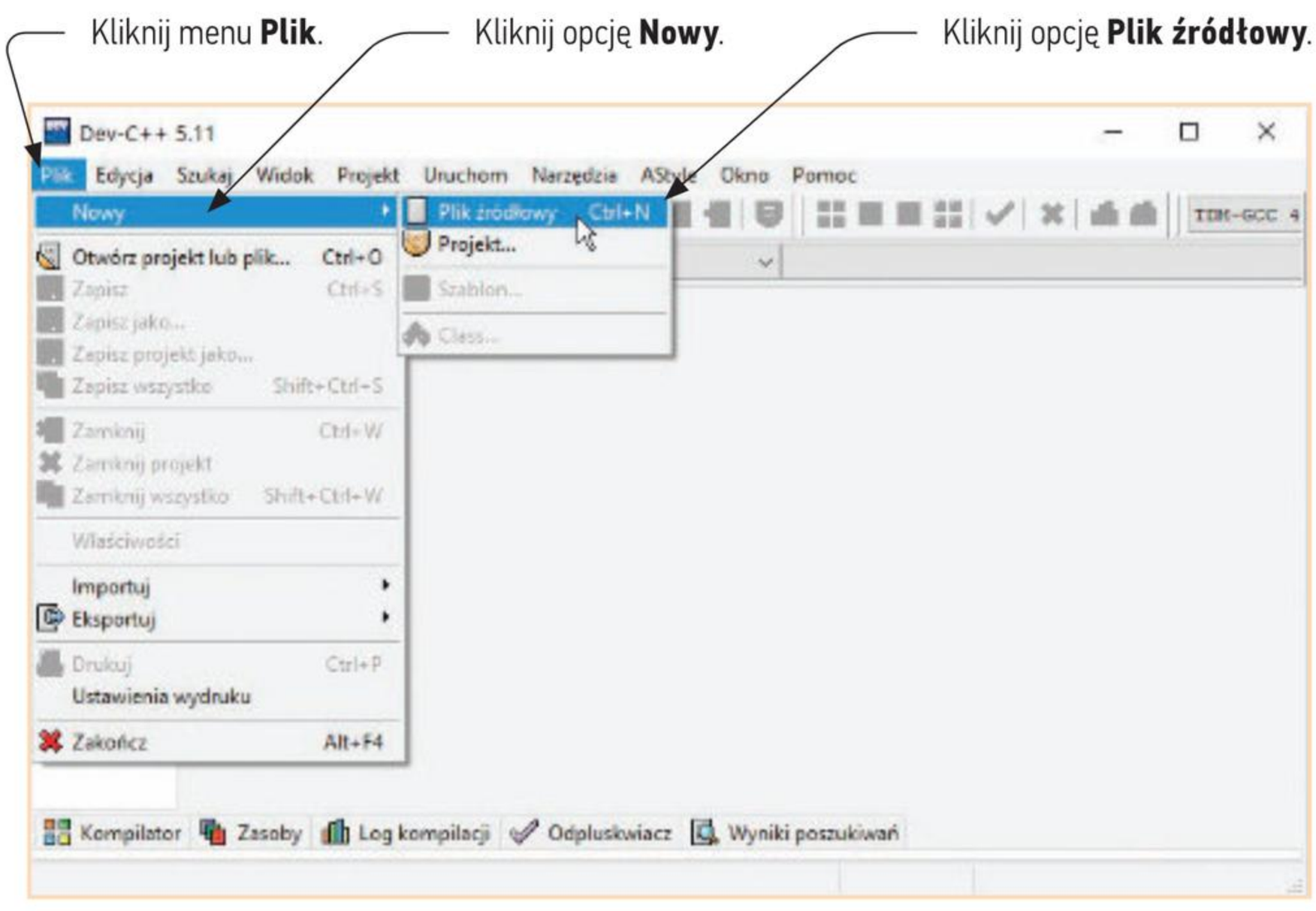
Aby tworzyć programy, możemy korzystać z wyspecjalizowanych pakietów programistycznych, zawierających zwykle edytor kodu źródłowego, a także kompilator lub interpreter oraz inne narzędzia wspomagające programowanie.

Aby pisać programy w języku C++, należy zainstalować wybraną aplikację zawierającą edytor i kompilator języka C++, np. Microsoft Visual Studio, Code::Blocks, Dev-C++. Są one dostępne bezpłatnie w internecie.

Wyspecjalizowany pakiet programistyczny zawiera edytor kodu źródłowego, kompilator i inne narzędzia wspomagające programowanie. Taką aplikację (lub zespół aplikacji) określa się też jako zintegrowane środowisko programistyczne.



Rys. 4. Okno programu Dev-C++



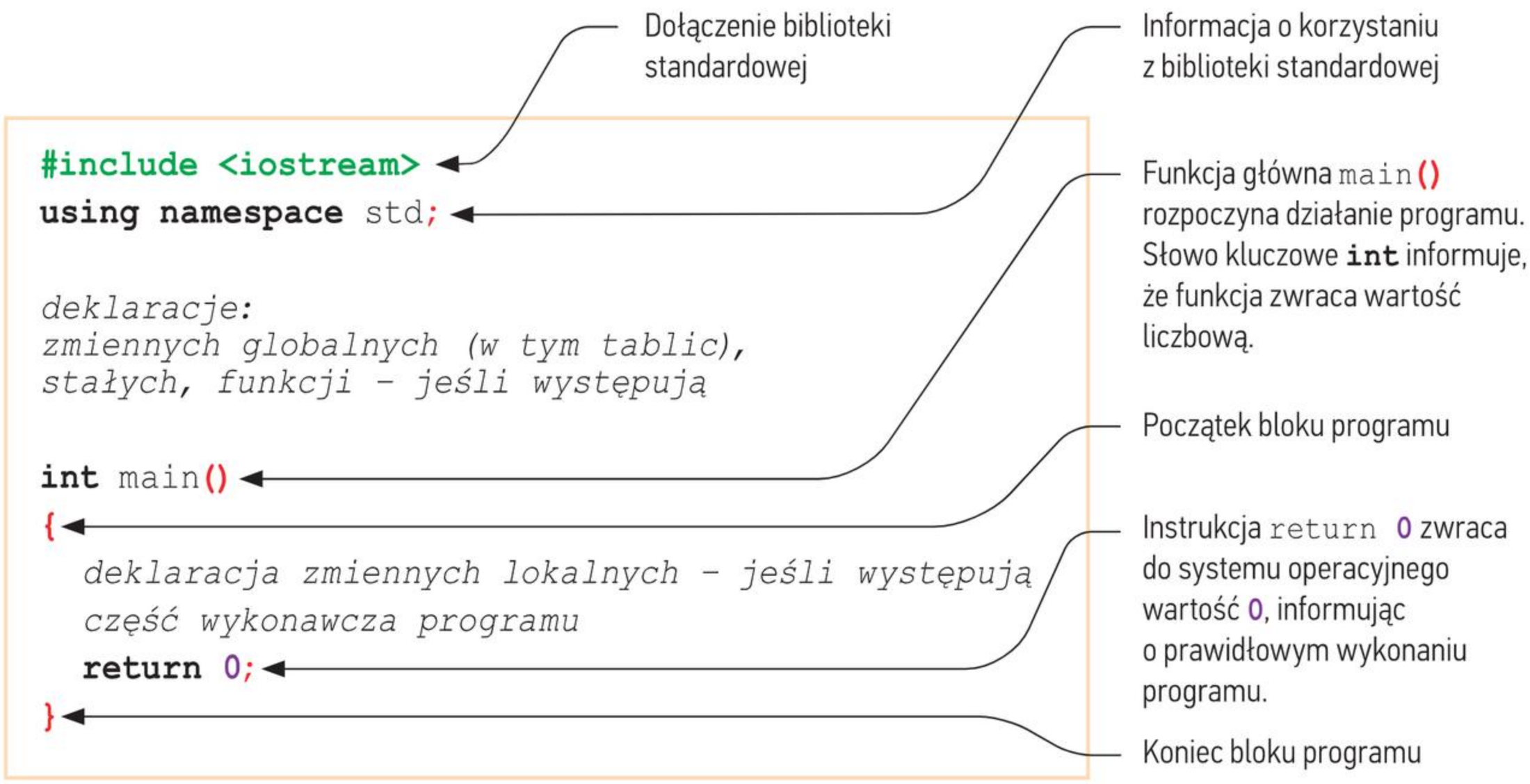
Rys. 5. Tworzenie nowego pliku źródłowego w programie Dev-C++

4 Piszemy program w języku C++

Każdy język programowania ma swój zbiór instrukcji. Postać instrukcji jest inna dla różnych języków. Języki programowania posiadają również odpowiednie reguły budowy zdań oraz właściwe słownictwo podobnie jak języki naturalne, którymi posługują się ludzie.

Instrukcje języka programowania realizują czynności takie, jak: wprowadzanie danych, wyprowadzanie wyników, wykonywanie obliczeń, określanie warunków czy powtarzanie operacji.

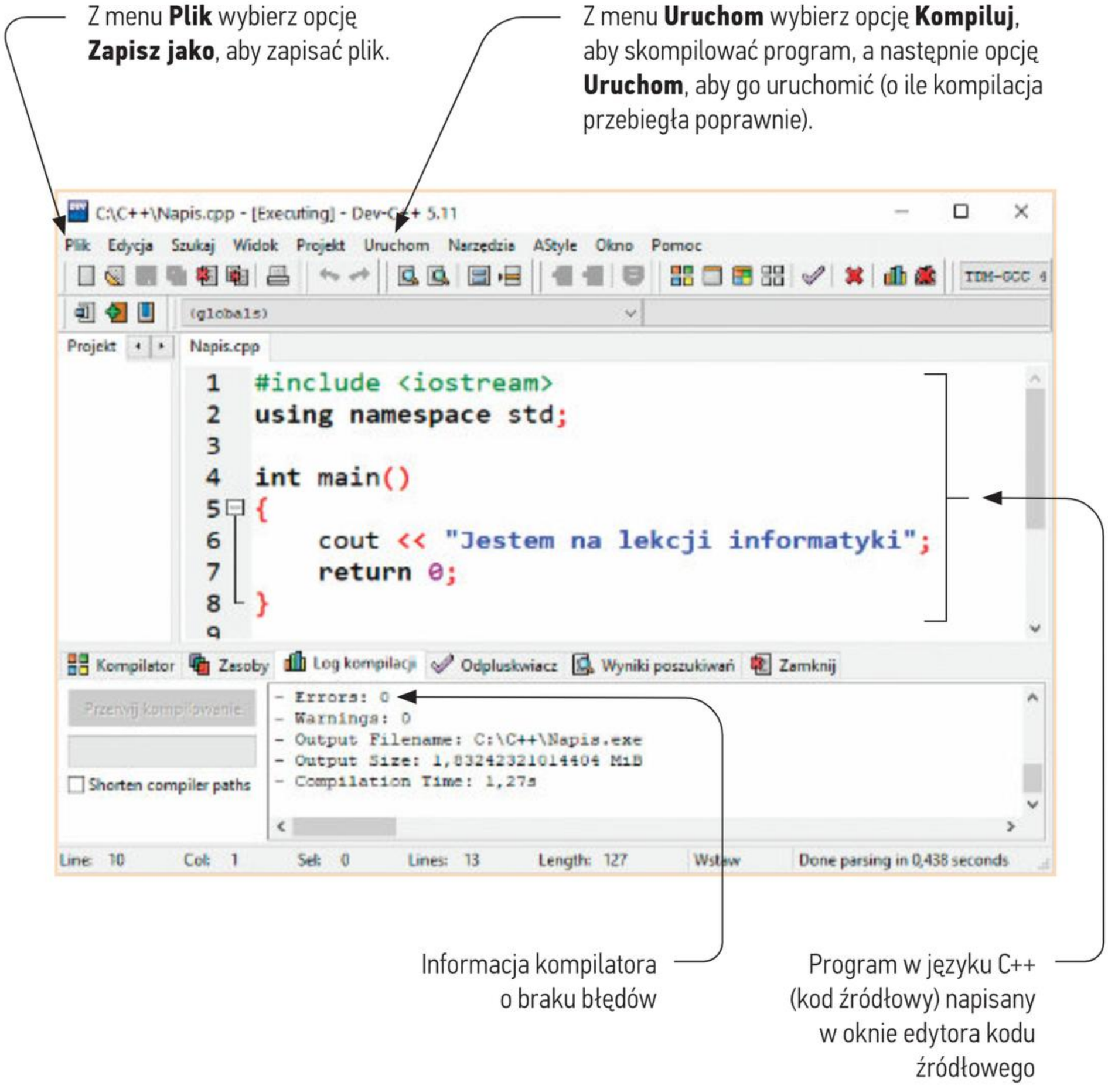
Każda instrukcja języka C++ musi być zakończona średnikiem.



Rys. 6. Ogólna struktura programu w języku C++

Etapy tworzenia programu w języku C++

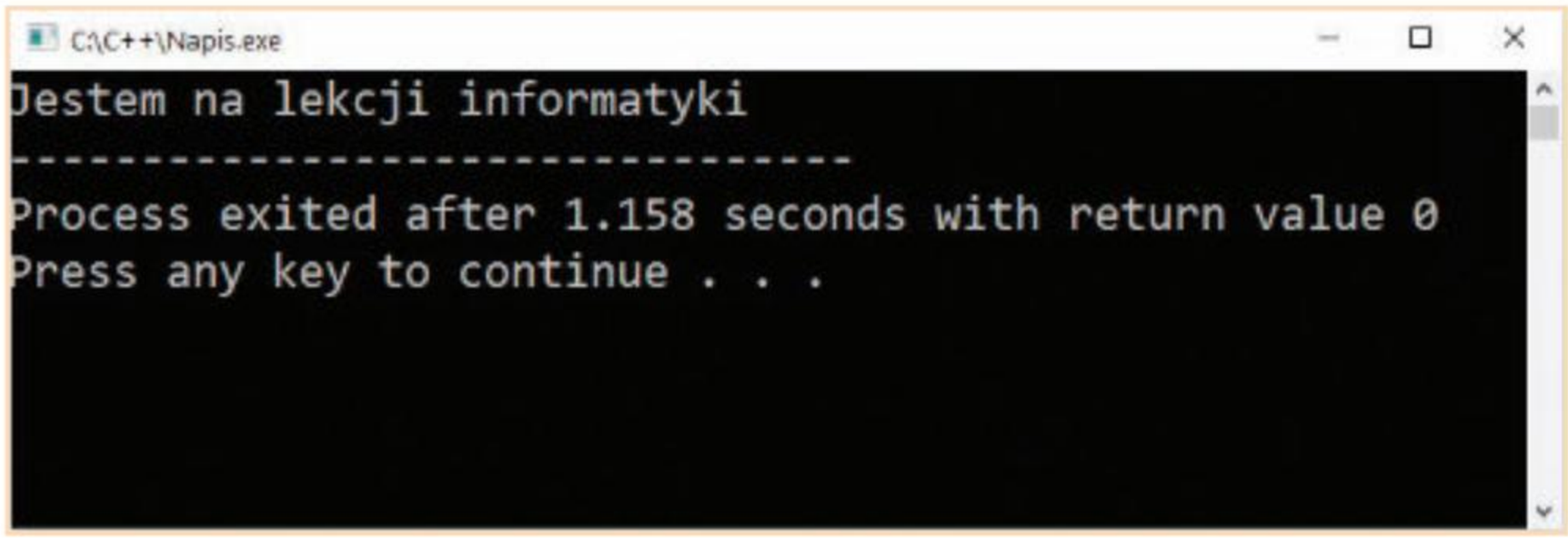
1. W edytorze wbudowanym do środowiska programistycznego języka C++ piszemy kod źródłowy programu (rys. 7.).
2. Zapisujemy program w pliku (pliki programów w języku C++ mają rozszerzenie *cpp*).
3. Kompilujemy program, korzystając z opcji **Kompiluj** z menu **Uruchom**. Przebieg kompilacji obserwujemy w dolnej części okna programu (rys. 7.). Efektem poprawnie przeprowadzonej kompilacji jest plik z rozszerzeniem *exe*.
4. Jeśli program został skompilowany, uruchamiamy go, zazwyczaj korzystając z opcji **Uruchom** z menu **Uruchom**. Wynik działania programu pokaże się w osobnym oknie (rys. 8.).



Rys. 7. Okno programu Dev-C++ – kompilacja programu napisanego w języku C++ przebiegła prawidłowo

Po wykonaniu programu wyniki pojawią się na ekranie monitora w osobnym oknie (oknie wykonania programu) – rys. 8.

Aby w języku C++ wyświetlać na ekranie komunikaty i wyniki, możemy zastosować polecenie `cout <<`.
Operator `<<` oznacza wyprowadzenie wartości podanej po prawej stronie. Jeśli chcemy wyprowadzić więcej wartości, należy użyć operatora `<<` wielokrotnie.
Napisy wyświetlane na ekranie muszą być ujęte w programie w górne cudzysłowy `""`.



Rys. 8. Wynik działania programu – ćwiczenie 4.

Przykłady	Wyniki
`cout << "informatyka";`	komputer wyświetla na ekranie napis „informatyka”
`cout << "Pole wynosi: " << p;`	komputer wyświetla na ekranie napis „Pole wynosi: ”, a następnie wartość zmiennej `p`
`cout << a + b;`	komputer wyświetla na ekranie wynik wyrażenia `a + b`

Tabela 2. Przykłady zastosowania polecenia `cout <<`

Ćwiczenie 4. Piszemy program wyświetlający napis na ekranie

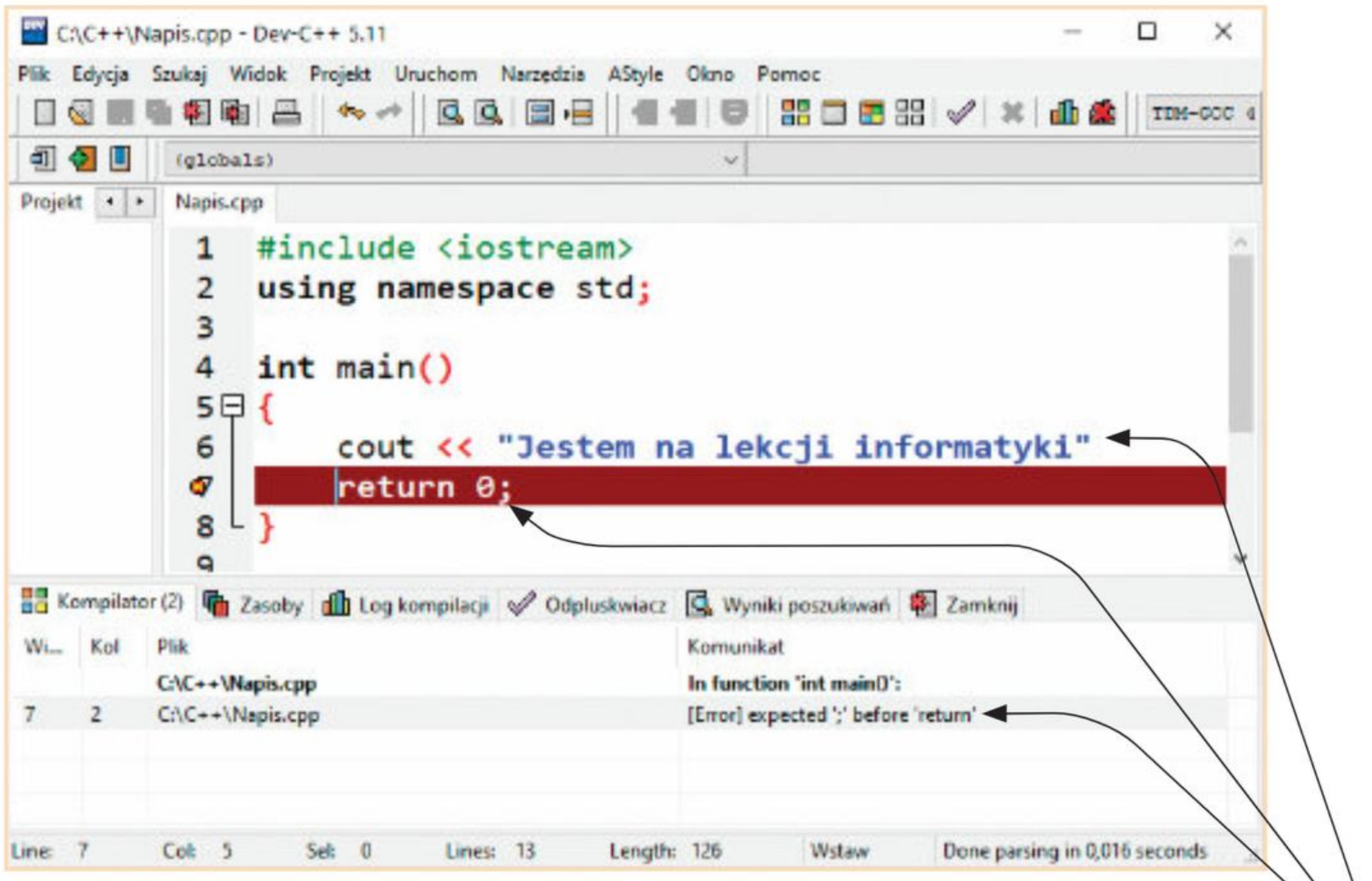
1. Utwórz nowy plik. Napisz program wyświetlający na ekranie napis „Jestem na lekcji informatyki”.
2. Zapisz program w pliku pod nazwą *Napis*.
3. Skompiluj program.
4. Jeśli kompilator wykrył błędy, popraw je. Zapisz program w pliku pod tą samą nazwą. Ponownie skompiluj program.
5. Jeżeli kompilacja przebiegła poprawnie, uruchom program.

Uruchomić i wykonać można tylko program poprawnie skompilowany. W trakcie wykonywania programu procesor rozpoznaje i wykonuje swoje wewnętrzne instrukcje.

Poprawnie skompilowany program nie wymaga już powtórnego tłumaczenia na kod maszynowy.

Aby zmienić rozmiar czcionki w oknie wykonania programu, należy kliknąć przycisk (w lewym górnym rogu okna) i wybrać polecenie **Właściwości/ Czcionka**.

Możesz stosować w edytorze kodu źródłowego operacje kopiowania, wycinania i wklejania fragmentu tekstu z wykorzystaniem **Schowka**, tak jak w każdym edytorze tekstu.



Komunikat kompilatora o błędzie (w tym przypadku braku średnika przed instrukcją **`return`** – średnik powinien kończyć instrukcję w wierszu 6.)

Rys. 9. Okno programu Dev-C++ – kompilator wykrył błąd w programie napisanym w języku C++

Ćwiczenie 5. Modyfikujemy program w języku C++

1. Zmodyfikuj program zapisany w ćwiczeniu 4., aby dodatkowo w drugim wierszu wyświetlał twoje imię.
2. Zapisz program w pliku pod nazwą *Imię*. Skompiluj i uruchom program.

Wskazówka: Aby po wyświetleniu napisu kursor przeszedł do drugiego wiersza, należy umieścić zapis `\n` po pierwszej części napisu. Można to zapisać na kilka sposobów:

```
cout << "Jestem na lekcji informatyki\n";
cout << "Tomek";
```

lub krócej:

```
cout << "Jestem na lekcji informatyki\n" << "Tomek";
```

lub jeszcze krócej:

```
cout << "Jestem na lekcji informatyki\nTomek";
```

Zapis `\n` odpowiada kodowi znaku przejścia do nowego wiersza.

Zapamiętaj

- Algorytm przedstawia krok po kroku sposób rozwiązania problemu.
- Rozwiązując dowolny problem, postępujemy według podobnego schematu. Określamy specyfikację zadania: dane wejściowe, poszukiwane wyniki oraz sposób rozwiązania.
- Poznaliśmy kilka sposobów przedstawiania algorytmów:
  - opis słowny – ogólny opis wykonywanych operacji,
  - lista kroków – każdy punkt listy zawiera opis wykonywanej czynności,
  - schemat blokowy (graficzna prezentacja algorytmu) – układ bloków i połączeń między nimi określa kolejność i sposób wykonywania kroków algorytmu,
  - program komputerowy – ciąg instrukcji języka programowania realizujący algorytm.
- Program komputerowy może występować w dwóch postaciach:
  - jako program (kod) źródłowy – program zrozumiały dla programisty, zapisany jako ciąg instrukcji tekstowych języka programowania wysokiego poziomu, np. języka C++,
  - jako program (kod) wynikowy (maszynowy) – program zrozumiały dla komputera, napisany w języku wewnętrznym komputera w postaci ciągu instrukcji dla procesora.
- Każdy język programowania ma swój zbiór instrukcji, reguły budowy zdań i właściwe słownictwo.
- Aby utworzyć program w języku C++, piszemy kod źródłowy w odpowiednim edytorze, zapisujemy go w pliku i kompilujemy. Jeśli kompilacja przebiegła prawidłowo, uruchamiamy program. Efektem poprawnie przeprowadzonej kompilacji jest plik z rozszerzeniem *exe*, który uruchamia się w oddzielnym oknie.

Serwis dla ciekawskich

Język C++ jest rozwinięciem języka C, wymyślonego na początku lat 70. XX wieku. Język C powstał z kolei na bazie języków BCPL i B. Następna litera w alfabecie („D") również jest nazwą języka programowania, wykorzystującego cechy języka C++. Istnieją także języki C# i Objective-C, wykorzystujące niektóre koncepcje języków C i C++. Na świecie istnieje obecnie ponad 1000 różnych języków programowania!

OKO W OKO Z MONITOREM

Pytania i polecenia

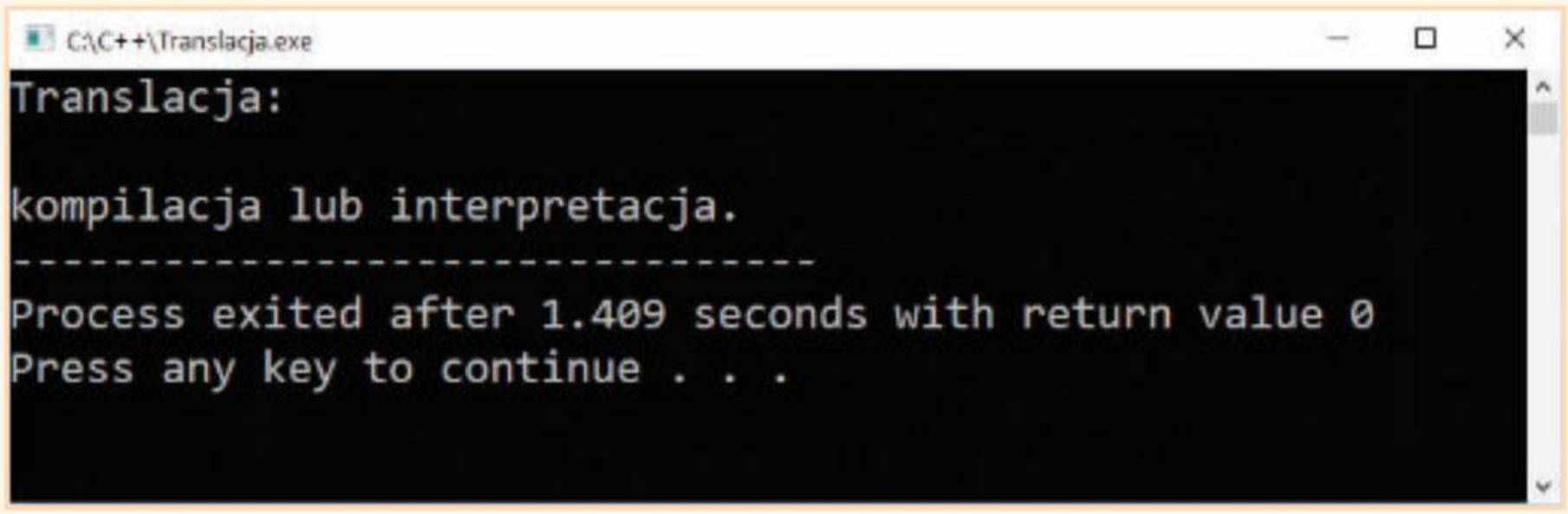
1. Co to jest algorytm?
2. Czym jest specyfikacja zadania?
3. Wymień kolejne etapy rozwiązywania problemów.
4. Czy przepis na kisiel ma cechy algorytmu informatycznego? Uzasadnij odpowiedź.
5. Wymień cztery sposoby przedstawiania algorytmów.
6. Na czym polega zapisywanie algorytmu w postaci listy kroków?
7. Jakiego bloku użyjesz w schemacie blokowym do wyprowadzenia wyników obliczeń?
8. Czy do każdej figury w schemacie blokowym musi dochodzić jedno połączenie? Uzasadnij odpowiedź.
9. W jaki sposób dodaje się dwa ułamki zwykłe o różnych mianownikach? Przygotuj słowny opis tego algorytmu.
10. Czym różni się program źródłowy od programu wynikowego?
11. Czym jest język programowania?
12. W jaki sposób tworzymy program w języku C++?

Zadania

1. Dane są dwie liczby: *a* i *b*. Przedstaw w postaci listy kroków i schematu blokowego algorytm obliczania ich sumy oraz wyprowadzania jej wartości.
2. Przedstaw w postaci listy kroków i schematu blokowego algorytm obliczania objętości prostopadłościanu o krawędziach podstawy *a* i *b* oraz wysokości *h*.
3. Przedstaw w postaci schematu blokowego algorytm obliczania objętości ostrosłupa o wysokości *h*. Podstawą ostrosłupa jest trójkąt prostokątny o przyprostokątnych *a* i *b*.
4. Napisz w języku C++ program wyświetlający na ekranie w siedmiu wierszach nazwy dni tygodnia. Zapisz program w pliku pod nazwą *Tydzień*.

DLA ZAINTERESOWANYCH

1. Przedstaw w postaci schematu blokowego algorytm rozwiązania następującego zadania: *Znajdź cenę 1 m² pola o powierzchni 80 arów, za które zapłacono 56 000 zł.*
2. Napisz w języku C++ program wyświetlający na ekranie napisy, jak pokazano na rysunku 10. Po napisie „Translacja:" powinien być umieszczony jeden pusty wiersz. Zapisz program w pliku pod nazwą *Translacja*.



Rys. 10. Wynik działania programu – zadanie 2. (dla zainteresowanych)

3.2. Piszemy programy w języku C++

Warto powtórzyć

1. Czym jest kod źródłowy programu?
2. Na czym polega kompilacja programu?
3. Jakie są etapy tworzenia programu w języku C++?
4. Które polecenie języka C++ umożliwia wyświetlanie na ekranie komunikatów i wyników obliczeń?

W poprzednim temacie poznaliśmy środowisko programistyczne języka C++ oraz napisaliśmy pierwszy program w tym języku. W tym temacie:

1. dowiesz się, jak stosować zmienne w programowaniu,
2. zastosujesz instrukcję warunkową do zapisywania algorytmów z warunkami,
3. zapiszesz wybrany algorytm iteracyjny z wykorzystaniem instrukcji iteracyjnej,
4. poznasz sposób definiowania i stosowania funkcji w języku C++,
5. dowiesz się, do czego służą tablice, i zastosujesz je w tworzonych programach.

1 Stosowanie zmiennych w programowaniu

Czym jest zmienna?

Zmienna w programie komputerowym to nazwana część pamięci operacyjnej komputera (RAM) o unikatowym adresie – **komórka (komórki) pamięci**. W każdym języku programowania musimy określić sposób korzystania ze zmiennych i nadawania im wartości.

Komórka pamięci
Część pamięci operacyjnej komputera (RAM) o unikatowym adresie. W komórkach pamięci przechowuje się dane reprezentujące instrukcje procesora lub dane dla tych instrukcji.

W wielu językach programowania (np. Pascal i C++) zmienne przed użyciem należy zadeklarować (co w środowiskach Scratch i Logomocja nazywamy tworzeniem zmiennych). Dzięki deklaracji kompilator może przydzielić pamięć operacyjną dla zmiennej.

Zmienna ma zawsze jakąś wartość i określone miejsce w pamięci komputera (adres komórki pamięci). Jeśli w tym samym programie zapiszemy w tej samej zmiennej nową wartość (wykonamy tzw. **instrukcję przypisania**), to poprzednia wartość znika.

Na przykład po wykonaniu kolejno następujących instrukcji przypisania:

```
s = 1;
s = s + 4;
```

w zmiennej `s` będzie pamiętana wartość 5.

Warto zauważyć, że instrukcja przypisania `s = s + 4` nie jest równością prawdziwą w sensie matematycznym.

Zmienne w języku C++

W każdym języku programowania musimy określić sposób korzystania ze zmiennych i nadawania im wartości. W języku C++ każdą zmienną używaną w programie należy zadeklarować, czyli określić jej nazwę i typ. Typ zmiennej wyznacza zbiór wartości, które może ona przyjmować (np. liczby całkowite, liczby rzeczywiste, znaki, wartości logiczne).

Zmienną deklarujemy w dowolnym miejscu programu, ale zawsze przed jej pierwszym użyciem.

Przykład	Opis
`int liczba;`	deklaracja zmiennej `liczba` typu całkowitego
`int a, b;`	deklaracja zmiennych `a` i `b` typu całkowitego (w przypadku zmiennych tego samego typu można ich nazwy wymienić po przecinku)
`float wzrost, wynik;`	deklaracja zmiennych `wzrost` i `wynik` typu rzeczywistego

Tabela 1. Przykłady deklarowania zmiennych w języku C++

Każdy język programowania (język formalny), podobnie jak języki naturalne, którymi posługują się ludzie, posiada odpowiednie zasady składni oraz właściwe słownictwo. Nazwy zmiennych są elementami języka programowania, dlatego podlegają jego zasadom, a nie regułom języka polskiego. Z tego powodu w nazwach zmiennych mogą się pojawić m.in. znaki podkreślnika i cyfry, które nie występują w wyrazach zapisywanych w języku polskim.

W języku C++ w nazwach zmiennych wolno stosować litery (bez polskich znaków diakrytycznych), cyfry i znak podkreślenia. Małe i wielkie litery w nazwach instrukcji, zmiennych itp. mają różne znaczenie – np. `liczba` i `Liczba` to dwie różne zmienne. Nazwa nie może zaczynać się od cyfry. Należy stosować nazwy zmiennych określające ich znaczenie.

Zmiennej można nadać wartość za pomocą instrukcji przypisania. Znak (operator) **=** oznacza przypisanie zmiennej zapisanej po lewej stronie znaku wartości umieszczonej po jego prawej stronie.

W większości języków programowania (w tym w języku C++) część dziesiętną liczby zapisujemy po kropce (a nie po przecinku, jak w matematyce), np. 34.5, a nie 34,5.

W języku C++ można również w ramach jednej instrukcji zadeklarować zmienną i nadać jej wartość.

Przykład	Opis
`liczba = 30;`	zmiennej `liczba` przypisujemy wartość 30
`int liczba = 20;`	deklarujemy zmienną `liczba` typu całkowitego i przypisujemy jej wartość 20
`wynik = 12.3 + 14.9;`	zmiennej `wynik` przypisujemy wartość wyrażenia 12,3 + 14,9
`wzrost = 165.5;`	zmiennej `wzrost` przypisujemy wartość 165,5
`float wzrost = 165.5;`	deklarujemy zmienną `wzrost` typu rzeczywistego i przypisujemy jej wartość 165,5

Tabela 2. Przykłady przypisywania zmiennej wartości w języku C++

Wartość zmiennej można również wprowadzać z klawiatury po uruchomieniu programu. W tym celu stosujemy polecenie `cin >>`. Operator `>>` oznacza wprowadzenie danych do odpowiedniej zmiennej, podanej po prawej stronie operatora. Jeśli chcemy wprowadzić dane do kilku zmiennych, użyjemy operatora `>>` wielokrotnie.

Przykład	Opis
`cin >> wzrost;`	po wprowadzeniu liczby z klawiatury komputer zapamiętuje ją w zmiennej `wzrost`
`cin >> a >> b;`	po wprowadzeniu dwóch liczb z klawiatury komputer zapamiętuje je kolejno w zmiennych `a` i `b`

Tabela 3. Przykłady wprowadzania z klawiatury wartości zmiennej w języku C++

Obliczenia z użyciem zmiennych

W obliczeniach z użyciem zmiennych używamy operatorów arytmetycznych zapisanych w tabeli 4.

Operator	Działanie	Przykład instrukcji przypisania
`+`	dodawanie	`suma = a + b;`
`-`	odejmowanie	`roznica = a - b;`
`*`	mnożenie	`iloczyn = a * b;`

`/`	dzielenie w zależności od typu danych: • całkowite (bez części ułamkowej) – dla zmiennych typu całkowitego • zmiennoprzecinkowe (z zachowaniem części ułamkowej) – jeśli przynajmniej jedna zmienna jest zmienną typu rzeczywistego	`iloraz = a / b;`
`%`	obliczenie reszty z dzielenia dwóch zmiennych typu całkowitego	`reszta = a % b;`

Tabela 4. Podstawowe operatory arytmetyczne w języku C++

Obliczenia.cpp

```
#include <iostream>
using namespace std;
int main()
{
    int a, b, suma;

    cout << "Pierwsza liczba: ";
    cin >> a;
    cout << "Druga liczba: ";
    cin >> b;
    suma = a + b;
    cout <<"Suma wynosi: "<< suma;
    return 0;
}
```

Rys. 1. Program w języku C++ obliczający sumę dwóch liczb całkowitych wprowadzanych z klawiatury i wyświetlający wynik na ekranie – ćwiczenie 1. (punkt 1.)

```
C:\C++\Obliczenia.exe
Pierwsza liczba: 342
Druga liczba: 91
Suma wynosi: 433
--------------------------------
Process exited after 22.15 seconds with return value 0
Press any key to continue . . .
```

Rys. 2. Wynik działania programu obliczającego sumę dwóch liczb całkowitych wprowadzanych z klawiatury – ćwiczenie 1. (punkt 1.)

Ćwiczenie 1. Deklarujemy zmienne w programie i wykonujemy na nich obliczenia

1. Program pokazany na rysunku 1. pozwala na wprowadzenie dwóch liczb całkowitych z klawiatury, obliczenie ich sumy i wyświetlenie wyniku na ekranie. Przepisz kod programu pokazanego na rysunku 1. Zapisz program w pliku pod nazwą *Obliczenia*. Skompiluj i uruchom program.
2. Zadeklaruj nową zmienną typu całkowitego o nazwie `iloczyn`. Dodaj do programu obliczanie iloczynu liczb `a` i `b` oraz wyświetlenie wyniku mnożenia na ekranie. Zapisz plik pod tą samą nazwą. Skompiluj i uruchom program.

2 Stosowanie instrukcji warunkowej do realizacji algorytmów z warunkami

Sytuacje warunkowe

Z sytuacjami warunkowymi spotykamy się w każdej dziedzinie wiedzy i życia codziennego. Na pytanie „Czy pada deszcz?" odpowiedź może brzmieć „tak" lub „nie". W zależności od tego, czy warunek jest spełniony czy nie, wybieramy sposób postępowania (rys. 3.).

Rys. 3. Przykład sytuacji warunkowej

Ćwiczenie 2. Podajemy przykłady sytuacji warunkowych
Podaj przykłady sytuacji warunkowych z życia codziennego.

Sytuacje warunkowe wystepują m.in. w matematyce i fizyce, gdy wykonanie obliczeń zależy od warunku, który muszą spełniać liczby (np. mają być nieujemne).

Z sytuacją warunkową spotykamy się wówczas, gdy wynik lub dalsze działanie zależy od spełnienia (lub niespełnienia) pewnego warunku. Algorytm, w którym występują sytuacje warunkowe, będziemy nazywać algorytmem z warunkami (inaczej: algorytmem z rozgałęzieniami).

Lista kroków i schemat blokowy algorytmu z warunkami

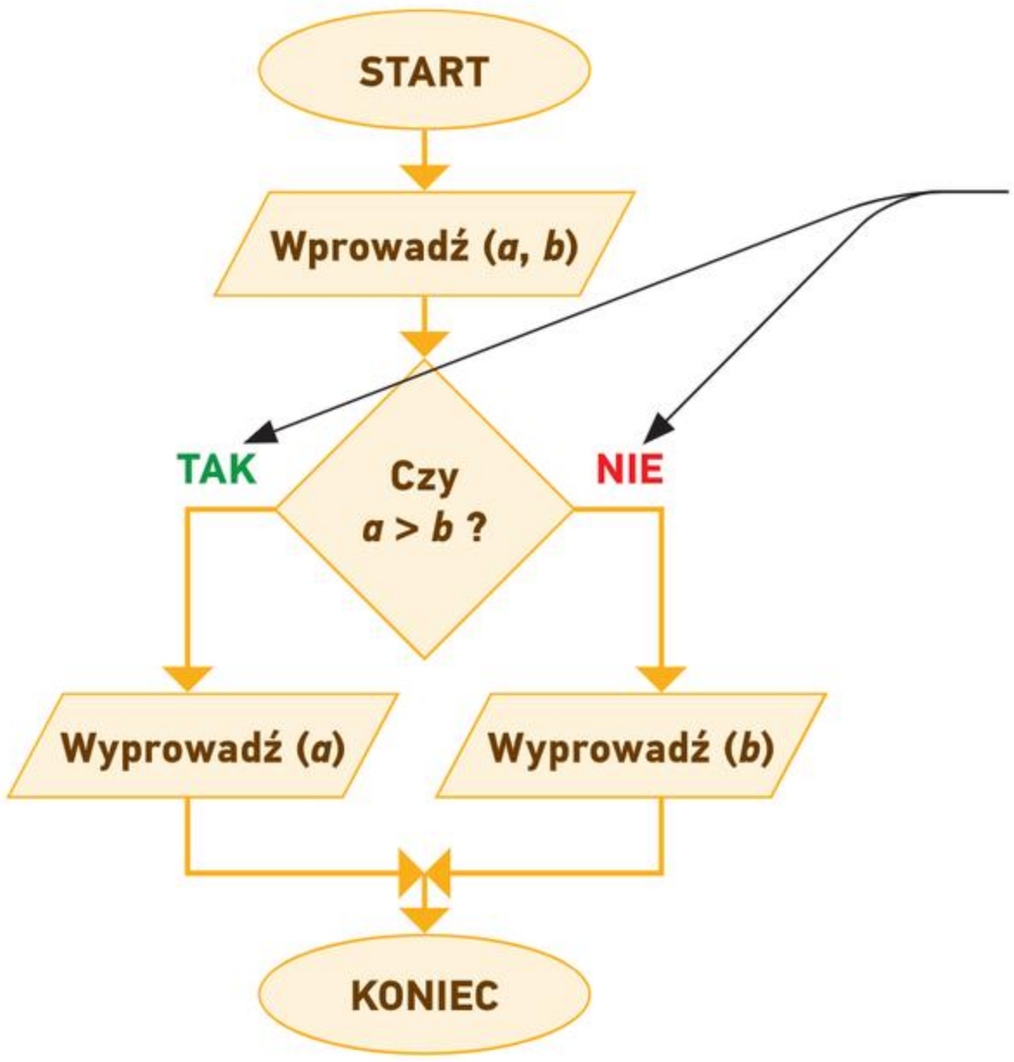
Zadanie: Sprawdź, która z dwóch nierównych sobie liczb całkowitych a i b jest większa, i wyprowadź większą z nich.
Specyfikacja zadania:
Dane: Liczby całkowite: a, b ($a \neq b$).
Wynik: Wartość większej z dwóch liczb.

Lista kroków:

1. Zacznij algorytm.
2. Wprowadź wartość liczby *a*.
3. Wprowadź wartość liczby *b*.
4. Sprawdź, czy liczba *a* jest większa od liczby *b*. Jeżeli tak, wyprowadź liczbę *a*, w przeciwnym przypadku wyprowadź liczbę *b*.
5. Zakończ algorytm.

W schemacie blokowym sytuację warunkową przedstawiamy za pomocą bloku warunkowego. W bloku tym wpisujemy warunek logiczny, stosując operatory porównań: = (równy), ≠ (różny), < (mniejszy), > (większy), <= (mniejszy lub równy), >= (większy lub równy), np. $x >= 0$, $n \neq 10$.

Można wpisywać również warunki złożone, połączone spójnikami (**i**, **lub**), np. $a < 0$ **lub** $a = 10$, $x > 0$ **i** $x < 20$.



Z bloku warunkowego wychodzą dwa połączenia: TAK i NIE. W zależności od tego, czy warunek jest spełniony czy nie, komputer wybierze odpowiednie odgałęzienie algorytmu.

Rys. 4. Schemat blokowy algorytmu z warunkiem – ćwiczenie 3.

Ćwiczenie 3. Analizujemy schemat blokowy algorytmu z warunkiem

1. Przeanalizuj schemat blokowy algorytmu z warunkiem pokazany na rysunku 4. Wskaż elementy związane z kolejnymi etapami rozwiązywania zadania: wprowadzanie danych, rozwiązanie zadania (wykonanie różnych operacji) i wyprowadzenie wyników.
2. Jakie rodzaje bloków zastosowano do określenia poszczególnych operacji? Zwróć uwagę na połączenia między blokami. Czy zawsze są takie same? Czy zawsze z jednego bloku „wychodzi" jedno połączenie? Zwróć uwagę na rozgałęzienie w schemacie blokowym.
3. Przetestuj działanie algorytmu dla różnych danych.

Zapisanie algorytmu z warunkami w języku C++

Aby w języku programowania napisać program realizujący algorytm z warunkami, stosujemy instrukcję warunkowa.

W języku C++ instrukcja warunkowa **`if`** występuje w wersji ogólnej (rys. 5a) i uproszczonej (rys. 5b). Jako *lista_instrukcji* może wystąpić jedna instrukcja (w tym instrukcja warunkowa) lub więcej instrukcji ujętych w blok **{}**.

Warunek musimy ująć w nawiasy okrągłe.

```
if(warunek)
  lista_instrukcji1;
else
  lista_instrukcji2;
kolejna_instrukcja;
```

Rys. 5a. Ogólna postać instrukcji warunkowej w języku C++

```
if(warunek)
  lista_instrukcji1;
kolejna_instrukcja;
```

Rys. 5b. Uproszczona postać instrukcji warunkowej w języku C++

W języku C++ wartość równą 0 uważa się za logiczny fałsz, a wartość różną od 0 – za logiczną prawdę.

Komputer sprawdza warunek logiczny – jeśli *warunek* jest spełniony (jest prawdziwy), wykonuje instrukcje (*lista_instrukcji1*). Jeśli *warunek* nie jest spełniony (jest fałszywy) – wykonuje instrukcje (*lista_instrukcji2*) po słowie **`else`** (rys. 5a), a jeśli brak tej części instrukcji – wykonuje od razu kolejną instrukcję (*kolejna_instrukcja*) po instrukcji warunkowej (rys. 5b).

Operatory		Określenie	Przykład wyrażenia	Interpretacja wyrażenia
Porównania	==	równy	a == b	a równe b
	!=	różny	a != b	a różne od b
	<	mniejszy	a < b	a mniejsze od b
	>	większy	a > b	a większe od b
	>=	większy lub równy	a >= b	a większe lub równe b
	<=	mniejszy lub równy	a <= b	a mniejsze lub równe b
Logiczne	||	alternatywa logiczna (lub)	a < -5 || a > 5	a mniejsze od –5 lub a większe od 5
	&&	koniunkcja logiczna (i)	a > 0 && a < 10	a większe od 0 i a mniejsze od 10
	!	negacja logiczna	! (a == 5)	a nierówne 5

Tabela 5. Podstawowe operatory porównania i logiczne w języku C++

Większa_z_dwóch.cpp

```
#include <iostream>
using namespace std;

int main()
{
    int a, b;
    cout << "Podaj pierwszą liczbę: ";
    cin >> a;
    cout << "Podaj drugą liczbę różną od pierwszej: ";
    cin >> b;
    if (a > b)
        cout << "Większa jest pierwsza liczba: " << a;
    else
        cout << "Większa jest druga liczba: " << b;
    return 0;
}
```

Rys. 6. Program w języku C++ realizujący algorytm z warunkiem – ćwiczenie 4.

Ćwiczenie 4. Stosujemy instrukcję warunkową

1. Program pokazany na rysunku 6. realizuje algorytm sprawdzenia, która z dwóch wprowadzonych z klawiatury różnych liczb jest większa i wyprowadza większą z nich. Przepisz kod programu pokazanego na rysunku 6. Wyjaśnij znaczenie poszczególnych wierszy programu.
2. Zapisz program w pliku pod nazwą *Większa_z_dwóch*. Skompiluj i uruchom program.

Ćwiczenie 5. Piszemy program realizujący algorytm z warunkiem

1. Napisz program sprawdzający, czy wprowadzona z klawiatury liczba całkowita jest dodatnia. Jeśli jest dodatnia, program powinien wyświetlić komunikat „Liczba dodatnia”, jeśli nie – „Liczba niedodatnia”.
2. Zapisz program w pliku pod nazwą *Liczba*. Skompiluj i uruchom program.

3 Stosowanie instrukcji iteracyjnej

Czym jest iteracja?

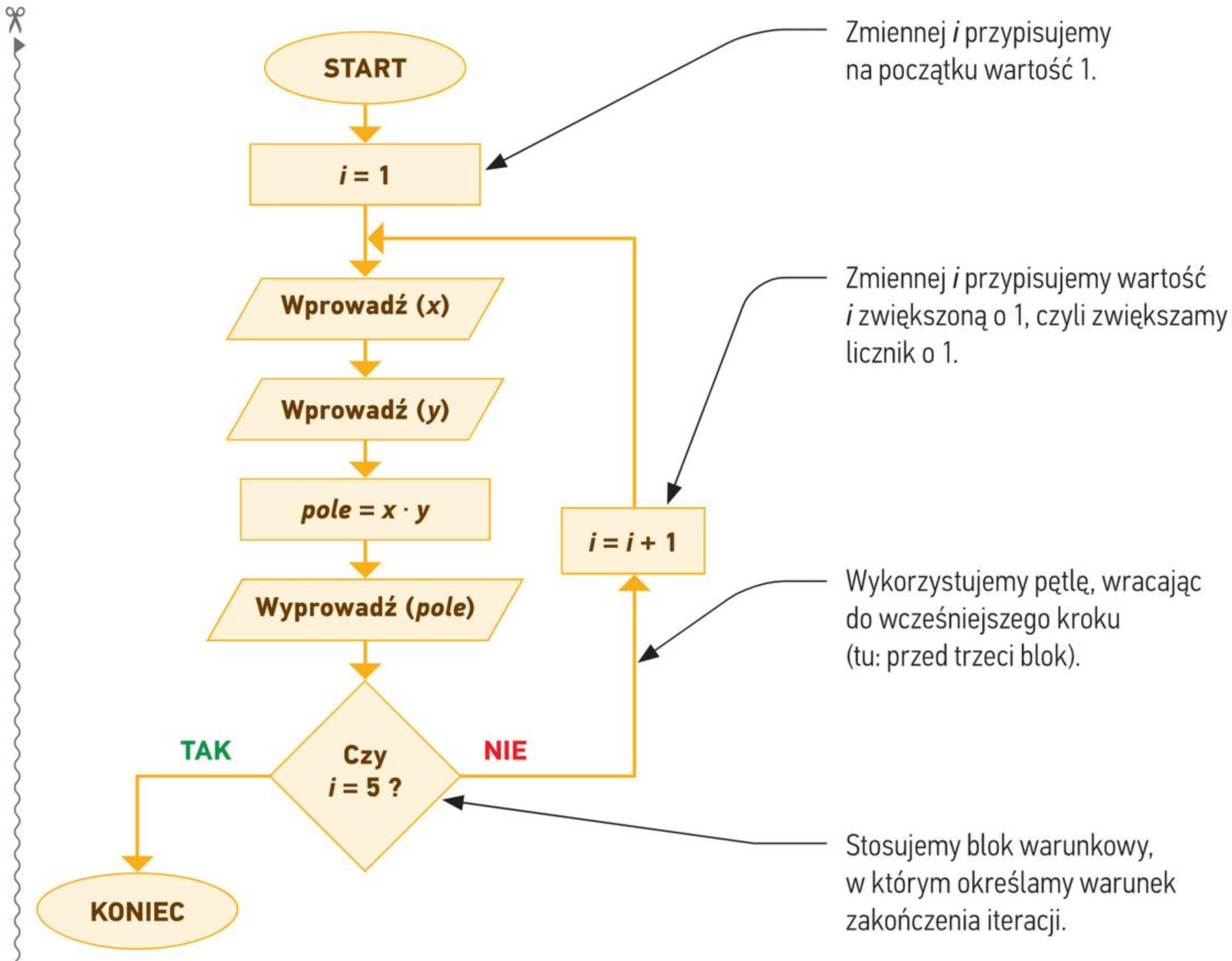
Czasami trzeba wykonać te same operacje na wielu danych. W takich przypadkach nie musimy powtarzać opisywania takich samych działań lub rysować takich samych bloków. Wystarczy wykorzystać iterację.

Iteracja polega na powtarzaniu tej samej operacji (ciągu operacji). Iterację implementujemy (piszemy kod źródłowy), stosując tzw. pętlę. Z pętlą mamy do czynienia, gdy w pewnym kroku algorytmu wracamy do jednego z wcześniejszych kroków, co powoduje, że kroki te komputer wykona wiele razy.

Lista kroków i schemat blokowy algorytmu iteracyjnego

Zadanie: Policz pola pięciu prostokątnych działek o bokach x i y.
Specyfikacja zadania:
Dane: Liczby naturalne oznaczające długości boków poszczególnych działek: x, y ($x > 0$ i $y > 0$).
Wynik: Wartości pól poszczególnych działek.
Lista kroków:

1. Zacznij algorytm.
2. Zmiennej *i* przypisz wartość 1.
3. Wprowadź długość boku x.
4. Wprowadź długość boku y.
5. Zmiennej *pole* przypisz wartość wyrażenia: $x \cdot y$.
6. Wyprowadź wynik: *pole*.
7. Jeśli *i* nie równa się 5, zwiększ licznik o jeden ($i = i + 1$) i wróć do kroku 3.
8. Zakończ algorytm.



Rys. 7. Schemat blokowy algorytmu iteracyjnego – ćwiczenie 6.

Ćwiczenie 6. Analizujemy schemat blokowy algorytmu iteracyjnego

1. Schemat blokowy na rysunku 7. jest prezentacją graficzną algorytmu obliczenia pól pięciu prostokątnych działek. Zwróć uwagę na wprowadzony w schemacie licznik (zmienna sterująca *i*) oraz przypisaną mu wartość początkową.
2. Przetestuj działanie algorytmu dla pięciu par różnych wartości zmiennych *x* i *y*.
3. Odpowiedz na pytania:
   a. Które operacje się powtarzają?
   b. Ile razy będzie realizowana pętla?
   c. W jaki sposób zliczamy iteracje (powtórzenia) w tym algorytmie?
   d. Kiedy kończą się działania w pętli?

Zapisanie algorytmu iteracyjnego w języku C++

Aby w języku programowania napisać program realizujący algorytm iteracyjny, korzystamy z instrukcji iteracyjnych (instrukcji pętli).

W języku C++ jedną z instrukcji iteracyjnych jest instrukcja **`for`**. Do określenia liczby kroków iteracji (powtórzeń) stosujemy tzw. zmienną sterującą.

```
for (wyrażenie_początkowe; warunek; wyrażenie_pętli)
  lista_instrukcji;
```

Rys. 8. Ogólna postać instrukcji iteracyjnej **`for`** w języku C++

Komputer najpierw wykonuje *wyrażenie_początkowe*, zwykle ustalające wartość początkową zmiennej sterującej (np. `i = 0`). Następnie, dopóki spełniony jest *warunek*, komputer wykonuje *listę_instrukcji*, a potem wykonuje *wyrażenie_pętli*, służące zazwyczaj do zwiększenia (lub zmniejszenia) wartości zmiennej sterującej. Jako *lista_instrukcji* może wystąpić pojedyncza instrukcja (w tym inna instrukcja pętli) lub więcej instrukcji ujętych w blok `{}`.

Na przykład instrukcja:

```
for(i = 0; i < 5; i++)
{
  cout << "Podaj liczbę: ";
  cin >> a;
}
```

oznacza, że polecenia ujęte w blok `{}` program wykona 5 razy. Zmienna sterująca o nazwie `i` na początku ma wartość 0. Komputer wykonuje polecenia w bloku `{}`, gdy zmienna `i` jest mniejsza od 5 (czyli dla zmiennej `i` równej kolejno: 0, 1, 2, 3, 4).

Wyrażenie `i++` oznacza „zmiennej `i` przypisz wartość i zwiększoną o 1”. Jest to skrót instrukcji przypisania: `i = i + 1`.

Pola_działek.cpp

```
#include <iostream>
using namespace std;

int main()
{
    int x, y, i, pole;

    for(i = 0; i < 5; i++)
    {
        cout << "Podaj długość pierwszego boku: ";
        cin >> x;
        cout << "Podaj długość drugiego boku: ";
        cin >> y;
        pole = x * y;
        cout << "Pole działki o boku" << x << " i boku " << y << " wynosi: " << pole << endl;
    }
    return 0;
}
```

Rys. 9. Program w języku C++ realizujący algorytm obliczania pól pięciu prostokątów – ćwiczenie 7.

Ćwiczenie 7. Stosujemy instrukcję iteracyjną `for`

1. Program pokazany na rysunku 9. realizuje algorytm obliczania pól pięciu prostokątów, których wymiary mają być wprowadzane z klawiatury. Przepisz kod programu pokazany na rysunku 9. Wyjaśnij znaczenie poszczególnych wierszy programu.
2. Zapisz program w pliku pod nazwą *Pola_działek*. Skompiluj i uruchom program.
3. Odpowiedz na pytania:
   a. Ile razy komputer wykona instrukcję `cin >> y`?
   b. Jaka jest wartość zmiennej `i` na koniec programu?

Wskazówka: Zapis `endl` oznacza początek nowego wiersza. Wstawienie `endl` na końcu instrukcji `cout <<` oznacza, że kursor przejdzie do następnego wiersza.

W realizacji algorytmów iteracyjnych ważne jest prawidłowe określenie sposobu zakończenia działań. Niepoprawne określenie tych warunków może spowodować, że obliczenia nigdy się nie zakończą, czyli nastąpi „zapętlenie" algorytmu lub programu.

4 Funkcje

Kroki, które powtarzają się w algorytmie, można potraktować jako problem cząstkowy i przedstawić w postaci **podprogramu** (funkcji). Umożliwi to opracowanie każdego problemu cząstkowego oddzielnie.

W języku C++ wszystkie podprogramy nazywamy funkcjami. Aby napisać cały program, wystarczy utworzyć funkcje realizujące określone elementy

programu i odpowiednio je połączyć. Połączenia funkcji z programem głównym realizuje się poprzez instrukcję wywołania podprogramu.

Program główny (w języku C++ to funkcja główna `main()`) komunikuje się z wywoływaną funkcją poprzez przekazywanie parametrów. W definicji funkcji określamy parametry formalne (zwane też parametrami), które w momencie wywołania funkcji komputer zastąpi parametrami aktualnymi (zwanymi też argumentami).

Podprogram (funkcja)
Wyodrębniona część programu, mająca jednoznaczną nazwę i ustalony sposób wymiany danych z innymi częściami programu. Realizujący określone zadanie podprogram można wielokrotnie wykorzystywać w programie.

W nazwach funkcji i jej parametrów można używać tylko liter (bez polskich znaków diakrytycznych, podobnie jak w przypadku nazw zmiennych), znaków podkreślenia i cyfr. Nazwa nie może zaczynać się od cyfry. Należy stosować nazwy opisujące działanie danej funkcji.

```
void nazwa_funkcji(lista_parametrów_formalnych)
{
  lista_instrukcji;
}
```

Rys. 10a. Ogólna postać definicji funkcji niezwracającej wartości w języku C++

```
opis_typu nazwa_funkcji(lista_parametrów_formalnych)
{
  lista_instrukcji;
  return wartość;
}
```

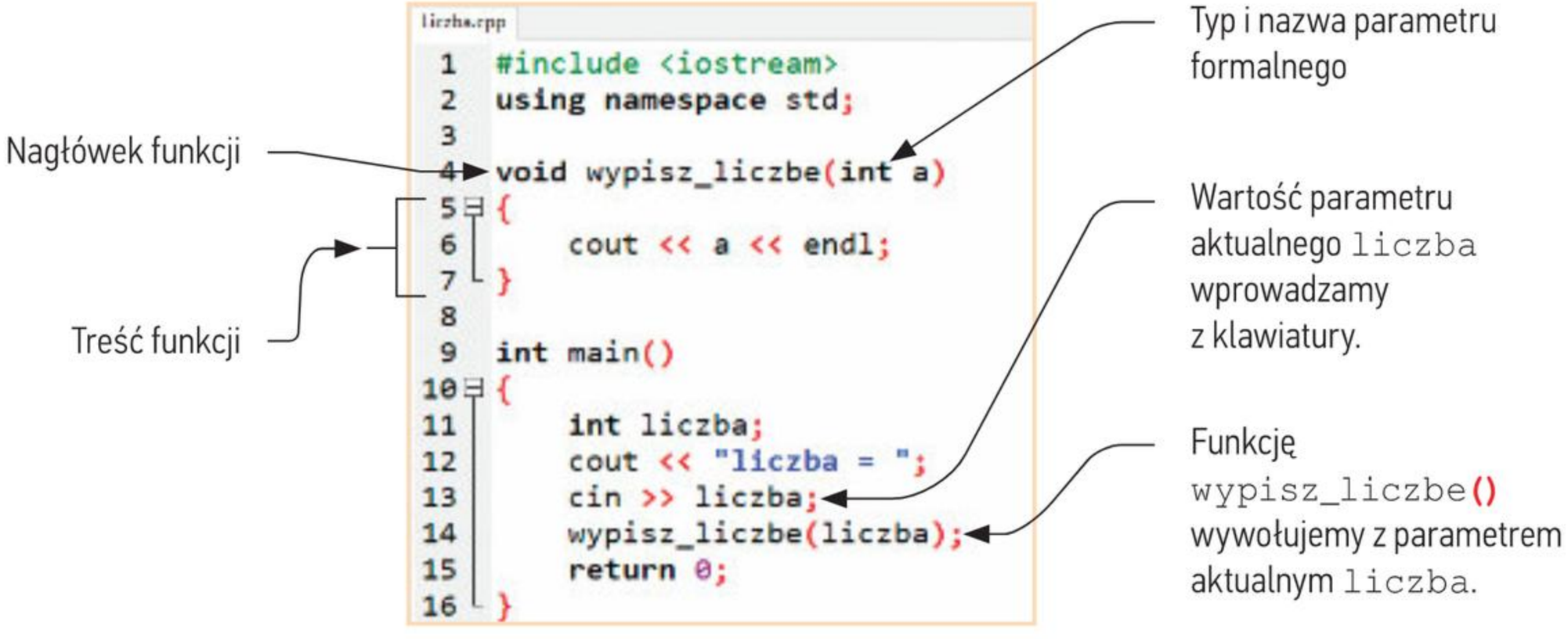
Rys. 10b. Ogólna postać definicji funkcji zwracającej wartość w języku C++

Funkcje w języku C++ dzielimy na dwie grupy: funkcje niezwracające wartości (rys. 10a) i funkcje zwracające wartość (rys. 10b).

Funkcję niezwracającą wartości stosujemy, gdy mamy wykonać pewne czynności, np. wprowadzenie danych, wyświetlenie danych na ekranie czy obliczenie wielu wartości. Funkcja taka nie ma typu, dlatego w części nagłówkowej przed nazwą funkcji musi być umieszczone słowo **`void`**. Ponieważ funkcja nie zwraca wartości, instrukcja **`return`** nie musi wystąpić (a jeśli występuje, to nie może określać *wartości*). Na rysunku 11. pokazano przykład funkcji wyświetlającej na ekranie wprowadzoną liczbę.

W przypadku funkcji bez parametrów (niezwracającej wartości i zwracającej wartość) nawiasy po nazwie funkcji pozostawiamy puste, tak jak w funkcji głównej `main()`.

Aby wywołać funkcję niezwracającą wartości, należy umieścić nazwę funkcji w odpowiednim miejscu programu głównego (w funkcji głównej `main()`).



Rys. 11. Przykład definicji funkcji niezwracającej wartości w języku C++ – ćwiczenie 8.

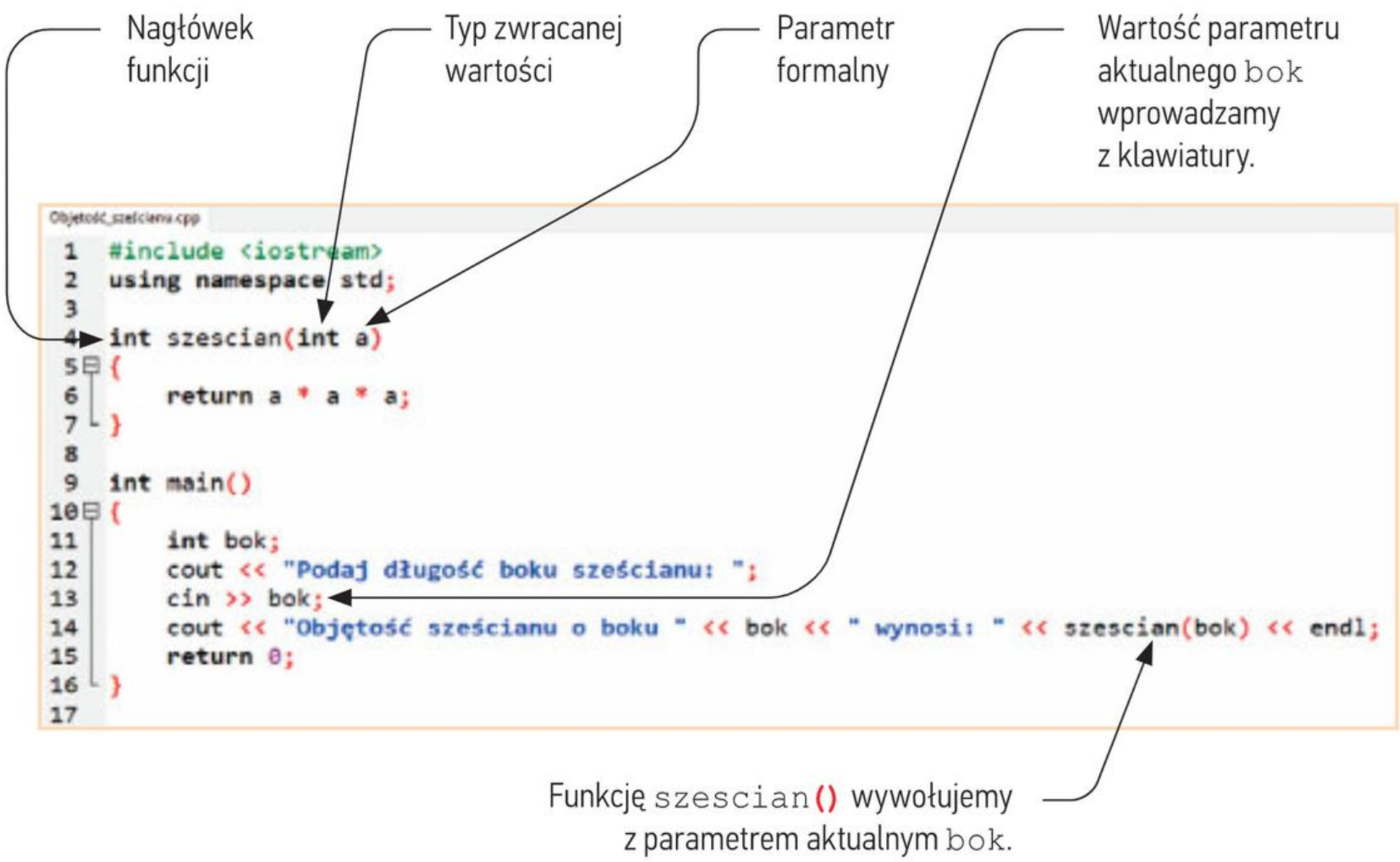
Ćwiczenie 8. Definiujemy w języku C++ funkcję niezwracającą wartości

1. Przepisz program pokazany na rysunku 11. Zapisz program w pliku pod nazwą *Liczba*. Skompiluj i uruchom program.
2. Dodaj w funkcji polecenie wyświetlania napisu „Wprowadzona liczba to: ”.
3. Zapisz program pod tą samą nazwą. Skompiluj go i uruchom.

Funkcję zwracającą wartość stosujemy, gdy celem funkcji jest obliczenie i zwrócenie pewnej wartości do funkcji wywołującej. Treść funkcji powinna się znaleźć w bloku `{}`. Funkcja zwracająca wartość musi zawierać instrukcję **`return`** z wartością zwracaną do funkcji wywołującej.

Sposób definiowania takiej funkcji pokażemy na przykładzie funkcji z jednym parametrem typu całkowitego o nazwie `a` obliczającej objętość sześcianu o boku `a` i zwracającej do programu głównego wynik obliczenia.

Aby wywołać funkcję zwracającą wartość z parametrami, należy umieścić nazwę funkcji z parametrami aktualnymi w odpowiednim miejscu programu głównego (w funkcji głównej `main()`).



Rys. 12. Definiowanie i wywołanie funkcji zwracającej wartość z jednym parametrem w języku C++ – ćwiczenie 9.

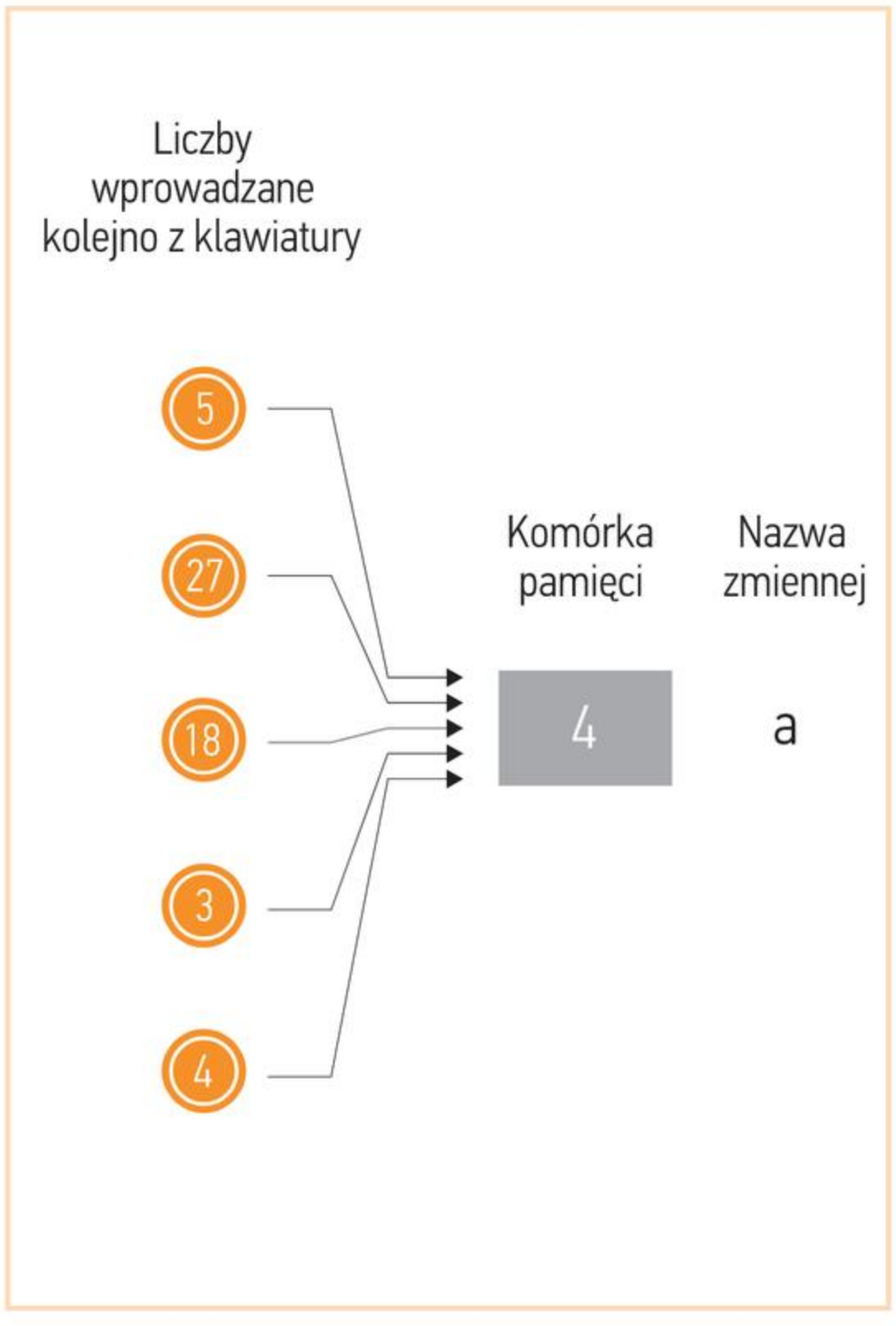
Ćwiczenie 9. Definiujemy w języku C++ funkcję zwracającą wartość

1. Zdefiniuj funkcję `szescian()` z parametrem `a` typu całkowitego obliczającą objętość sześcianu i zwracającą do programu głównego wynik obliczeń (rys. 12.).
2. Funkcję `szescian()` wywołaj w funkcji głównej `main()` z parametrem aktualnym `bok`.
3. Zapisz program w pliku pod nazwą *Objętość_sześcianu*. Skompiluj i uruchom program.
4. Sprawdź działanie programu dla kilku różnych danych.

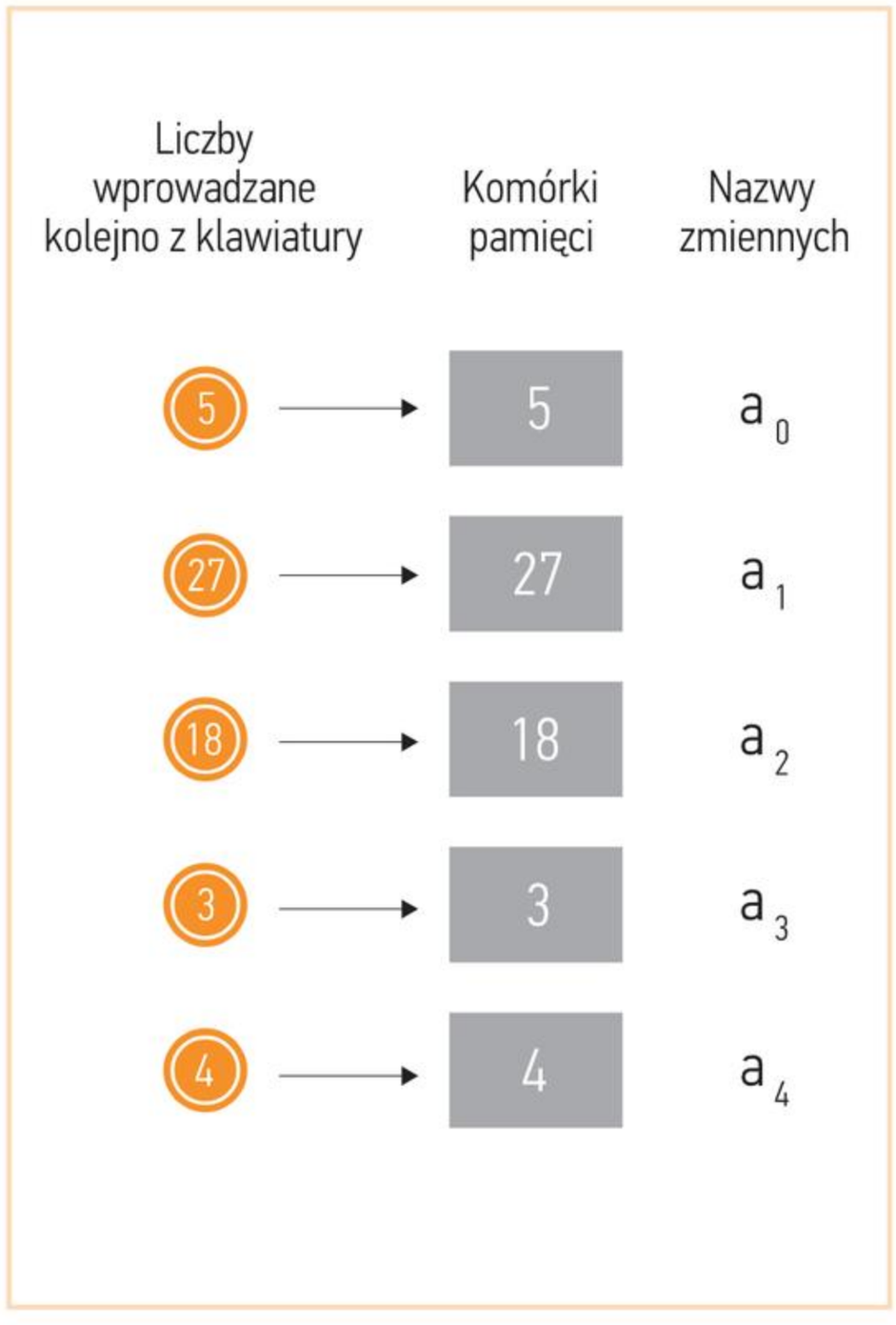
Wynik działania funkcji można również przypisać zmiennej, np. `objetosc = szescian(bok);`, przy czym zmienną `objetosc` należy wcześniej zadeklarować jako liczbę całkowitą (**`int`**).

5 Tablice

Niektóre algorytmy wymagają pamiętania wszystkich elementów zbioru zarówno w trakcie działania programu, jak i po jego zakończeniu. Przykładem jest algorytm porządkowania, ponieważ po jego zakończeniu musimy otrzymać wszystkie elementy w określonym porządku. Aby zapamiętać wszystkie dane, musimy je różnie nazwać. Moglibyśmy użyć liter alfabetu, ale w przypadku dużej liczby danych zabrakłoby liter. Dlatego wymyślono tzw. zmienne indeksowane, w których do nazwy zmiennej (np. *a*) dodaje się kolejny numer (indeks), np.: a_0, a_1, …, a_n. Wówczas dla każdej zmiennej komputer rezerwuje oddzielną komórkę pamięci (rys. 13b).



Rys. 13a. Zapisywanie w pamięci komputera wartości danych wprowadzanych z klawiatury z użyciem jednej zmiennej



Rys. 13b. Zapisywanie w pamięci komputera wartości danych wprowadzanych z klawiatury z użyciem pięciu zmiennych indeksowanych

Aby utworzyć zmienne indeksowane w języku C++, musimy zadeklarować specjalną strukturę danych – tablicę (rys. 14.).

```
opis_typu nazwa_tablicy [liczba_elementów_tablicy];
```

Rys. 14. Ogólna postać deklarowania tablicy w języku C++

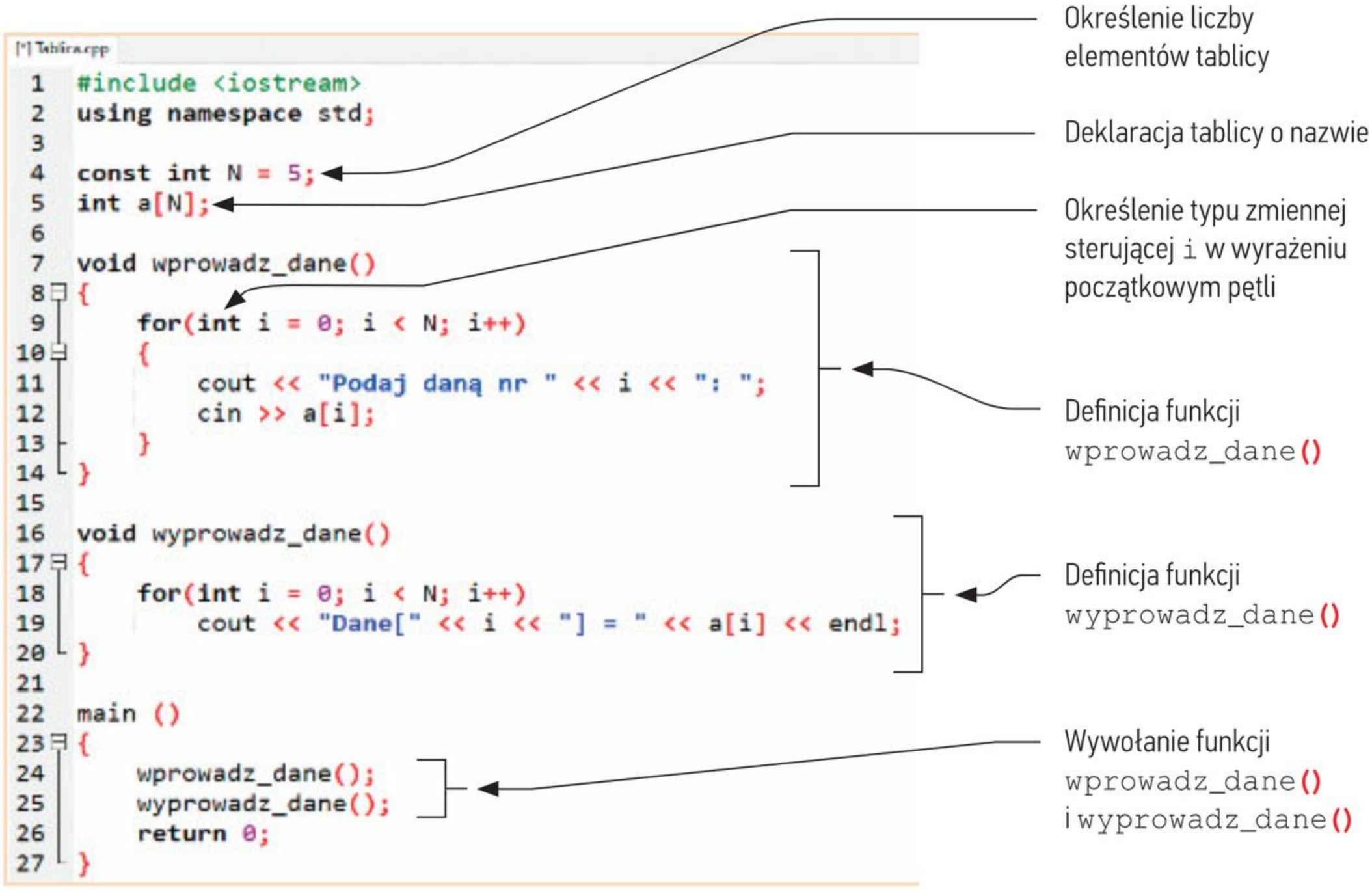
Do elementów tablicy odwołujemy się, podając nazwę tablicy i indeks elementu umieszczony w nawiasach kwadratowych, np.: `a[0]`, `a[1]`, …, `a[n - 1]` – dla tablicy `n`-elementowej o nazwie `a`. W języku C++ indeks pierwszego elementu tablicy jest zawsze równy 0.

Liczba elementów tablicy musi być z góry określona, np. wpisana w deklaracji tablicy jako konkretna wartość lub podana za pomocą wcześniej zdefiniowanej stałej (`const`).

Przykład	Opis	Odwoływanie do elementów tablicy
`int dane[50];`	oznacza zadeklarowanie tablicy o nazwie `dane`, składającej się z pięćdziesięciu elementów typu całkowitego (**`int`**) o indeksach od 0 do 49	do elementów tablicy odwołujemy się przez zmienne: `dane[0]`, `dane[1]`, …, `dane[49]`
`const int N = 10;` `int a[N];`	oznacza zadeklarowanie tablicy o nazwie `a`, składającej się z dziesięciu elementów typu całkowitego (**`int`**)	do elementów tablicy odwołujemy się przez zmienne: `a[0]`, `a[1]`, …, `a[9]`
`const int LICZBA_EL = 100;` `float tab[LICZBA_EL];`	oznacza utworzenie tablicy o nazwie `tab`, składającej się ze stu elementów typu rzeczywistego (**`float`**)	do elementów tablicy odwołujemy się przez zmienne: `tab[0]`, `tab[1]`, …, `tab[99]`

Tabela 6. Przykłady deklaracji tablicy w języku C++

Do wprowadzania danych do tablicy i do ich wyprowadzania zastosujemy funkcje bez parametrów i niezwracające wartości.



Rys. 15. Definicja funkcji wprowadzającej dane z klawiatury do tablicy i wyprowadzającej dane z tablicy na ekran – ćwiczenie 10.

Ćwiczenie 10. Wprowadzamy dane do tablicy i wyprowadzamy je z tablicy na ekran

1. Utwórz nowy plik źródłowy. Zdefiniuj funkcje (rys. 15.):
   a. o nazwie `wprowadz_dane()`, wprowadzającą liczby do tablicy `a`, składającej się z pięciu liczb całkowitych,
   b. o nazwie `wyprowadz_dane()`, wyświetlającą na ekranie w kolejnych wierszach liczby z tablicy `a`.
2. Wywołaj funkcje `wprowadz_dane()` i `wyprowadz_dane()` w funkcji głównej `main()`.
3. Zapisz program w pliku pod nazwą *Tablica*.
4. Skompiluj i uruchom program.

Zapamiętaj

- Zmienną używaną w programie należy zadeklarować przed pierwszym użyciem: określić jej nazwę i typ.
- Zmiennej nadajemy wartość za pomocą instrukcji przypisania.
- W języku C++ do zapisywania algorytmów, w których występują sytuacje warunkowe, stosujemy instrukcję warunkową **`if`**.
- Jeśli w algorytmie występują powtórzenia, możemy zastosować instrukcję iteracyjną **`for`**. Liczba powtórzeń (iteracji) jest w niej zwykle z góry określona.
- Aby pisać przejrzyste programy, należy opracowywać oddzielnie problemy cząstkowe, stosując podprogramy (funkcje).
- Podprogram należy najpierw zdefiniować, a potem wywołać w programie głównym.
- W języku C++ definiujemy funkcje niezwracające wartości i zwracające wartość. Możemy definiować funkcje z parametrami i bez parametrów.
- Aby zaprogramować algorytm, w którym w trakcie działania programu musimy pamiętać wszystkie elementy zbioru, możemy zapisywać dane w tablicy.
- W języku C++ liczba elementów (rozmiar) tablicy musi być z góry określona, np. jako zdefiniowana wcześniej stała. Należy również określić typ elementów.

Serwis dla ciekawskich

Nazwa języka programowania Ada (który powstał w latach 70. XX wieku) pochodzi od imienia kobiety, którą wielu uważa za pierwszą programistkę komputerów. W pierwszej połowie XIX wieku Ada Lovelace (czyt. ejda lawlejs) współpracowała z Charlesem Babbage'em (czyt. czarlsem babydżem) przy projektowaniu pierwszej programowalnej maszyny liczącej (której jednak nigdy nie skonstruowano...). Tworzone przez lady Lovelace opisy rozwiązywania konkretnych zadań obliczeniowych uznaje się za pierwsze programy.

OKO W OKO Z MONITOREM

Pytania i polecenia

1. Na co pozwala kompilatorowi deklaracja zmiennej?
2. Czym się różni operator `==` od `=`? Podaj przykłady użycia każdego z tych operatorów.
3. Wyjaśnij zapisy:
   a. `int liczba;`
   b. `liczba = 20;`
   c. `int liczba = 20;`
4. Kiedy komputer wykona instrukcje umieszczone po słowie `else` w instrukcji warunkowej?
5. Dlaczego zmienną `i` w pętli

```
for (i = 0; i < 5; i++)
   cout << i;
```

   nazywamy zmienną sterującą?
6. W jakim celu definiujemy podprogramy?
7. Kiedy definiujemy funkcję niezwracającą wartości, a kiedy zwracającą wartość? Wyjaśnij na przykładzie.
8. W jakim celu definiujemy tablicę? Podaj przykład użycia tablicy.

Zadania

1. Napisz program w języku C++ obliczający pole trójkąta o podstawie `a` i wysokości `h` dla wartości `a` i `h` wprowadzanych z klawiatury. Wynik obliczenia i odpowiedni komunikat program powinien wyświetlać na ekranie. Zapisz program w pliku pod nazwą *Pole_trójkąta*.

2. Napisz program w języku C++ obliczający sumę i średnią arytmetyczną trzech liczb rzeczywistych (**`float`**) wprowadzanych z klawiatury. Wyniki obliczeń i odpowiednie komunikaty program powinien wyświetlać na ekranie. Zapisz program w pliku pod nazwą *Średnia*.

3. Napisz program w języku C++ obliczający iloraz liczb rzeczywistych (**`float`**) `a` i `b` wprowadzanych z klawiatury. Dla `b` równego zero program powinien wyświetlić napis „Nie dzielimy przez zero”. Zapisz program w pliku pod nazwą *Iloraz*.

4. Napisz program w języku C++ obliczający pola dziesięciu trapezów o podstawach `a` i `b` oraz wysokości `h` będących liczbami całkowitymi wprowadzanymi z klawiatury. Wyniki obliczeń i odpowiednie komunikaty program powinien wyświetać na ekranie. Zapisz program w pliku pod nazwą *Trapez*.

5. Napisz program w języku C++ wyświetlający na ekranie napis „Gwiazdkowe pozdrowienia”, a pod nim piętnaście gwiazdek (znaków *), w następnym wierszu – napis „Wszystkiego najlepszego”, a pod nim ponownie piętnaście gwiazdek. Zdefiniuj funkcję `gwiazdki()` jako niezwracającą wartości (czyli typu **`void`**) i bez parametrów. Zadaniem funkcji jest wyświetlenie w jednym wierszu piętnastu gwiazdek. Funkcję wywołaj w programie głównym. Zapisz program w pliku pod nazwą *Życzenia*.

6. Napisz program w języku C++ obliczający pole i obwód kwadratu. Zdefiniuj dwie funkcje `pole_kwadratu()` i `obwod_kwadratu()`. Każda z funkcji powinna być typu całkowitego, z jednym parametrem typu całkowitego o nazwie `a` i powinna zwracać do programu głównego wynik obliczeń. W programie głównym wywołaj obie funkcje z parametrem aktualnym `bok_kwadratu`, którego wartość wprowadzaj z klawiatury. Zapisz program w pliku pod nazwą *Pole_i_obwód_kwadratu*.

DLA ZAINTERESOWANYCH

1. Napisz program sprawdzający, która z wprowadzonych z klawiatury trzech liczb całkowitych `x`, `y` i `z` jest największa. Wyświetlaj na ekranie odpowiedni napis i wartość największej liczby. Zapisz program w pliku pod nazwą *Największa_z_trzech*.

2. Napisz program wyświetlający w dwudziestu wierszach taką samą liczbę znaków „@”. Zdefiniuj funkcję typu **`void`** o nazwie `at()` z jednym parametrem `n` typu całkowitego określającym liczbę znaków „@”, która wyświetli `n` znaków „@” w jednym wierszu. Wartość parametru aktualnego `liczba_at` wprowadzaj w momencie wywołania funkcji w programie głównym. Zapisz program w pliku pod nazwą *Znaki_at*.

3.3. Algorytmy na liczbach naturalnych

Warto powtórzyć

1. Czym jest algorytm?
2. Czym jest specyfikacja problemu (zadania)?
3. Na czym polega przedstawianie algorytmu w postaci listy kroków?
4. W jaki sposób przedstawiamy na schemacie blokowym sytuację warunkową?
5. Na czym polega iteracja?
6. Jaką instrukcję stosujemy do zapisywania sytuacji warunkowej w języku C++? Wyjaśnij jej działanie.
7. Jaką instrukcję stosujemy do zapisywania iteracji w języku C++? Wyjaśnij jej działanie.

Na lekcjach informatyki w klasie ósmej poznamy podstawowe algorytmy na liczbach naturalnych stosowane w matematyce i informatyce. Niektóre z nich wykorzystujemy także na co dzień. Algorytmy te zapiszemy w języku programowania C++. W tym temacie:

1. poznasz i zaprogramujesz algorytm badania podzielności liczb,
2. poznasz i zaprogramujesz algorytm Euklidesa w dwóch wersjach: z odejmowaniem i z resztą z dzielenia,
3. poznasz i zaprogramujesz algorytm wyodrębniania cyfr danej liczby.

1 Badanie podzielności liczb naturalnych

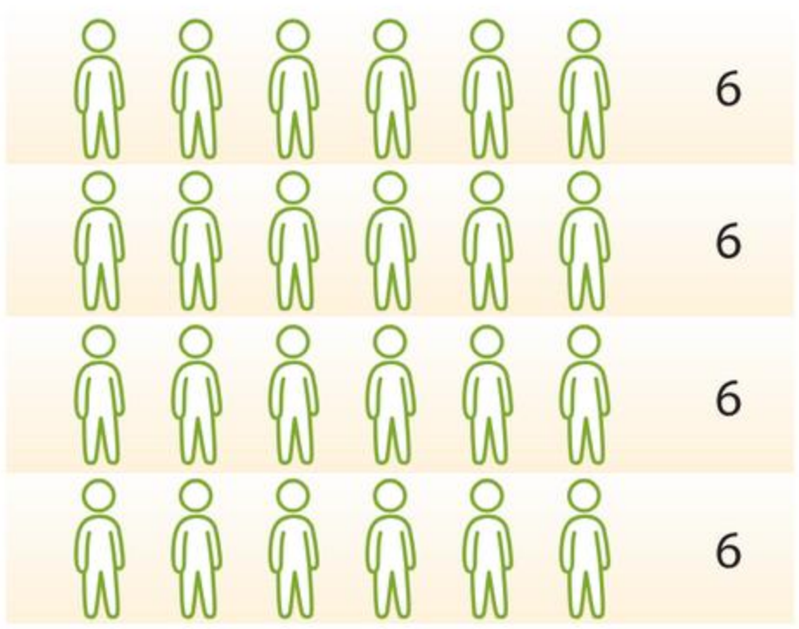
W klasie 8a pewnej szkoły jest dwudziestu czterech uczniów, a w klasie 8b – dwudziestu siedmiu. W każdej klasie chcemy utworzyć sześcioosobowe zespoły do gry w siatkówkę. W której klasie można utworzyć je tak, aby wszyscy uczniowie zostali przydzieleni do jakiejś drużyny?

Aby odpowiedzieć na to pytanie, musimy sprawdzić, czy liczba uczniów danej klasy dzieli się przez sześć bez reszty. Do rozwiązania tego zadania wybierzemy algorytm badania podzielności liczb, w którym wykorzystuje się obliczanie reszty z dzielenia.

Jeśli reszta z dzielenia dwóch liczb naturalnych jest równa zero, mówimy, że pierwsza liczba (dzielna) jest podzielna przez drugą (dzielnik).

Algorytm badania podzielności liczb naturalnych

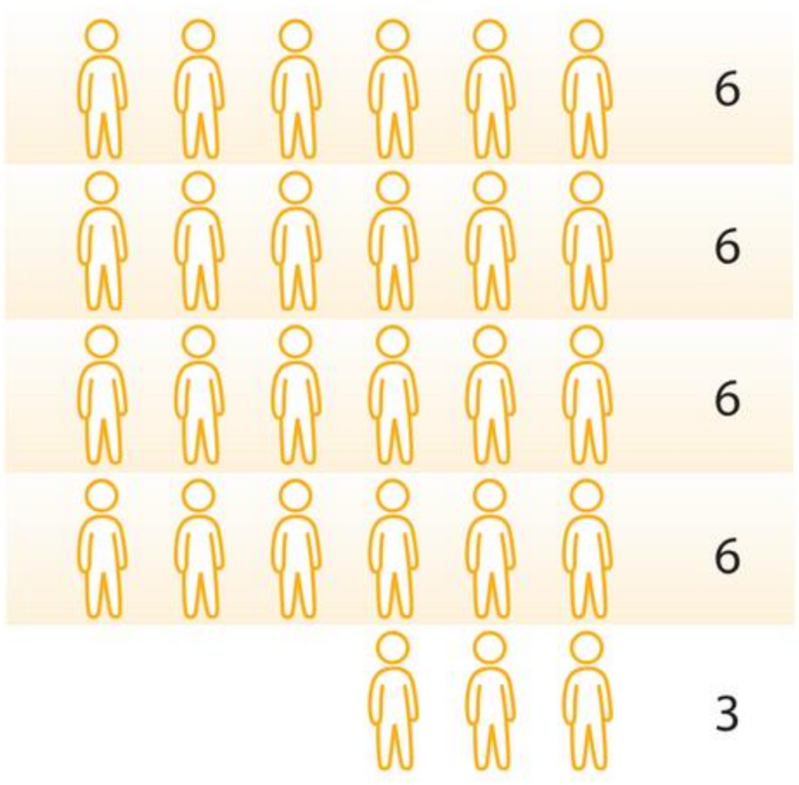
Klasa 8a — 24 uczniów



24 / 6 = 4 reszta 0

W klasie 8a powstały cztery drużyny i nie pozostał żaden uczeń, ponieważ liczba 24 (dzielna) jest podzielna przez liczbę 6 (dzielnik).

Klasa 8b — 27 uczniów



27 / 6 = 4 reszta 3

W klasie 8b powstały cztery drużyny i pozostało trzech uczniów, którzy nie tworzą pełnego (sześcioosobowego) zespołu, ponieważ liczba 27 (dzielna) nie jest podzielna przez liczbę 6 (dzielnik).

Rys. 1. Przykład stosowania algorytmu badania podzielności liczb naturalnych

Zadanie: Sprawdź, czy liczba naturalna a jest podzielna przez liczbę naturalną b różną od zera.

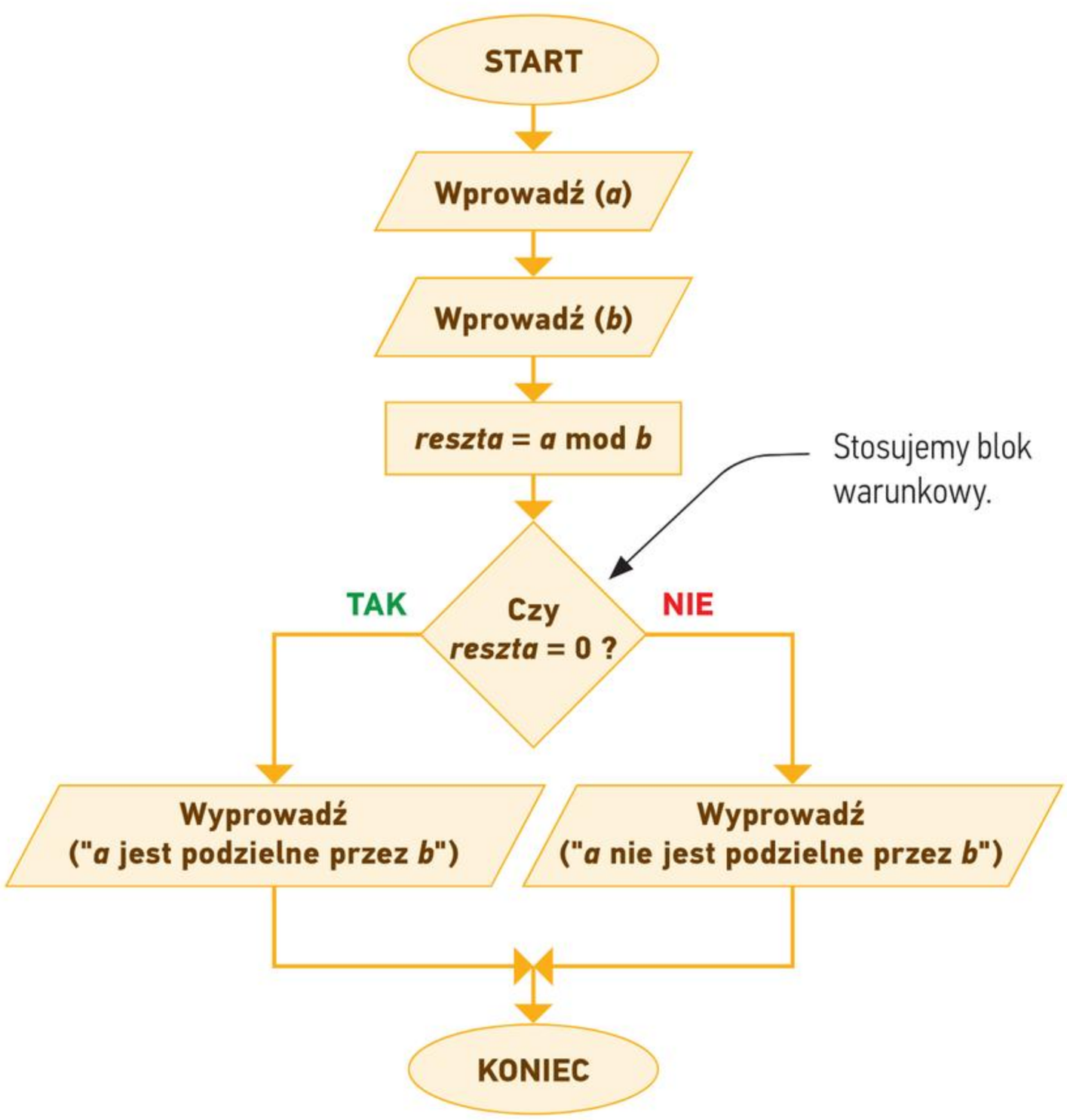
Specyfikacja zadania:

Dane: Liczby naturalne: a, b ($b \neq 0$).

Wynik: Komunikat: „a jest podzielne przez b" lub „a nie jest podzielne przez b" – zależnie od obliczonej reszty z dzielenia liczby a przez liczbę b.

Lista kroków:

1. Zacznij algorytm.
2. Wprowadź wartość dzielnej a.
3. Wprowadź wartość dzielnika b.
4. Zmiennej *reszta* przypisz wartość wyrażenia a **mod** b.
5. Jeżeli wartość zmiennej *reszta* jest równa zero, wyprowadź komunikat: „a jest podzielne przez b", w przeciwnym przypadku – „a nie jest podzielne przez b".
6. Zakończ algorytm.



Rys. 2. Schemat blokowy algorytmu badania podzielności liczb

Do wyznaczania reszty z dzielenia wykorzystamy operator modulo (**mod**).
Zapis *reszta* = *a* **mod** *b* oznacza, że zmiennej *reszta* przypisano resztę z dzielenia liczby *a* przez liczbę *b*.

a	*b*	Obliczenie reszty	Czy *reszta* = 0?	Komunikat
21	4	*reszta* = 21 **mod** 4 = 1	NIE	*a* nie jest podzielne przez *b*
42	7	*reszta* = 42 **mod** 7 = 0	TAK	*a* jest podzielne przez *b*
245	8	*reszta* = 245 **mod** 8 = 5	NIE	*a* nie jest podzielne przez *b*
8	16	*reszta* = 8 **mod** 16 = 8	NIE	*a* nie jest podzielne przez *b*
18	6	*reszta* = 18 **mod** 6 = 0	TAK	*a* jest podzielne przez *b*
125	5	*reszta* = 125 **mod** 5 = 0	TAK	*a* jest podzielne przez *b*

Tabela 1. Testowanie algorytmu badania podzielności liczb

Ćwiczenie 1. Sprawdzamy działanie algorytmu badania podzielności liczb
Zbadaj podzielność czterech par liczb: $a = 34$ i $b = 9$, $a = 36$ i $b = 6$, $a = 112$ i $b = 4$, $a = 2$ i $b = 18$. Zapisz rozwiązanie w dokumencie tekstowym – wzoruj się na tabeli 1.

Zapisywanie algorytmu badania podzielności liczb naturalnych w języku C++

W algorytmie badania podzielności dwóch liczb naturalnych występuje sytuacja warunkowa, dlatego do jej zaprogramowania w języku C++ wykorzystamy instrukcję warunkową **`if else`**.

Resztę z dzielenia wyznaczamy za pomocą operatora modulo, który w języku C++ zapisujemy symbolem `%`.

Typ danych **`unsigned int`** określa nieujemne liczby całkowite (liczby naturalne).

Wprowadzamy z klawiatury wartość dzielnej `a`.

Wprowadzamy z klawiatury wartość dzielnika `b`.

Sprawdzamy, czy reszta z dzielenia `a` przez `b` jest równa zero i wyświetlamy odpowiednie komunikaty.

(globalny)

[*] Podzielność.cpp

```
1  #include <iostream>
2  using namespace std;
3
4  int main()
5  {
6      unsigned int a, b;
7      cout << "Wprowadź wartość dzielnej a: ";
8      cin >> a;
9      cout << "Wprowadź wartość dzielnika b: ";
10     cin >> b;
11     if (a % b == 0)
12         cout << "a jest podzielne przez b";
13     else
14         cout << "a nie jest podzielne przez b";
15     return 0;
16 }
```

Rys. 3. Program w języku C++ realizujący algorytm badania podzielności liczb naturalnych

Ćwiczenie 2. Zapisujemy algorytm badania podzielności liczb naturalnych w języku C++

1. Program pokazany na rysunku 3. realizuje algorytm badania podzielności liczb naturalnych. Przepisz ten program i zapisz go w pliku pod nazwą *Podzielność*.
2. Skompiluj i uruchom kilkakrotnie program, testując go dla różnych wartości zmiennych `a` i `b`.
3. Objaśnij działanie programu, m.in. użycie poszczególnych instrukcji zgodnie z listą kroków tego algorytmu.

Ćwiczenie 3. Modyfikujemy program

1. Zmodyfikuj program zapisany w ćwiczeniu 2. tak, aby w przypadku gdy dzielnik `b` jest równy zero – wyświetlał komunikat „Wprowadzono dzielnik równy zero".
2. Zapisz program w pliku pod nazwą *Podzielność_bez_zera*.

Ćwiczenie 4. Stosujemy algorytm badania podzielności liczb do rozwiązania przykładowego problemu

1. Napisz program sprawdzający, ile sześcioosobowych grup zawodników do gry w siatkówkę (`liczba_g`) można utworzyć w każdej z pięciu klas pewnej szkoły. Wyświetlaj na ekranie kolejno wyniki. Jeśli w danej klasie nie można utworzyć grup tak, aby wszyscy uczniowie zostali przydzieleni, program powinien wyświetlić odpowiedni komunikat. Liczbę uczniów (`liczba_u`) każdej klasy wprowadzaj z klawiatury po uruchomieniu programu. Dane i wyniki program powinien wyświetlić w podobny sposób, jak pokazano na rysunku 4.
2. Zapisz program w pliku pod nazwą *Siatkówka*.

Wskazówka: Aby program wyświetlał komunikat przed wprowadzeniem liczby uczniów danej klasy (jak na rysunku 4.), można zastosować następujące instrukcje:

```
cout << "Uczniowie klasy " << i + 1 << ": ";
cin >> liczba_u;
```

gdzie zmienna `i` jest zmienną sterującą w pętli **`for`**.

```
C:\C++\Siatkówka.exe
Uczniowie klasy 1: 31
Nie utworzono grup z wszystkich graczy
Uczniowie klasy 2: 30
Liczba grup: 5
Uczniowie klasy 3: 36
Liczba grup: 6
Uczniowie klasy 4: 27
Nie utworzono grup z wszystkich graczy
Uczniowie klasy 5: 24
Liczba grup: 4

--------------------------------
Process exited after 24.25 seconds with return value 0
Press any key to continue . . .
```

Rys. 4. Wynik działania programu – ćwiczenie 4.

2 Algorytm Euklidesa

Opiszemy działanie algorytmu Euklidesa w dwóch wersjach iteracyjnych: z odejmowaniem i z resztą z dzielenia.

Algorytm Euklidesa służy do znajdowania największego wspólnego dzielnika (**NWD**) dwóch liczb naturalnych.

Działanie algorytmu Euklidesa w wersji z odejmowaniem

Euklides zauważył, że NWD dwóch liczb jest zawsze taki sam, jak NWD ich różnicy. Zatem od większej liczby można odjąć mniejszą tyle razy, ile ta się w niej zmieści.

Algorytm Euklidesa w wersji z odejmowaniem polega na odejmowaniu mniejszej liczby od większej i zastępowaniu większej liczby otrzymaną różnicą, dopóki większa liczba i otrzymana różnica nie będą równe.

Na przykład dla liczb $a = 10$ i $b = 25$ NWD wynosi 5. Korzystając z algorytmu Euklidesa, obliczamy NWD dla tych liczb w następujący sposób:

- od większej liczby odejmujemy mniejszą:
  $b - a = 25 - 10 = 15$,
- zastępujemy większą liczbę otrzymaną różnicą, czyli teraz:
  $a = 10$, $b = 15$,
- od większej liczby odejmujemy mniejszą:
  $b - a = 15 - 10 = 5$,
- zastępujemy większą liczbę otrzymaną różnicą, czyli teraz:
  $a = 10$, $b = 5$,
- od większej liczby odejmujemy mniejszą (teraz większą liczbą jest liczba a):
  $a - b = 10 - 5 = 5$,
- zastępujemy większą liczbę otrzymaną różnicą, czyli teraz:
  $a = 5$, $b = 5$.

Kończymy algorytm, ponieważ: $a = b$. Przyjmujemy, że NWD jest równe pierwszej z liczb (czyli liczbie a po obliczeniach): $\text{NWD} = a = 5$.

NWD

Największy wspólny dzielnik dwóch lub więcej liczb naturalnych (różnych od zera) to największa liczba naturalna, przez którą dzieli się bez reszty każda z tych liczb.

Największy wspólny dzielnik liczb (NWD) możemy wykorzystać do skracania ułamków zwykłych – licznik i mianownik dzielimy przez ich NWD. Jeśli licznik i mianownik ułamka $\frac{10}{25}$ podzielimy przez 5, otrzymamy $\frac{2}{5}$.

Algorytm Euklidesa – wersja z odejmowaniem

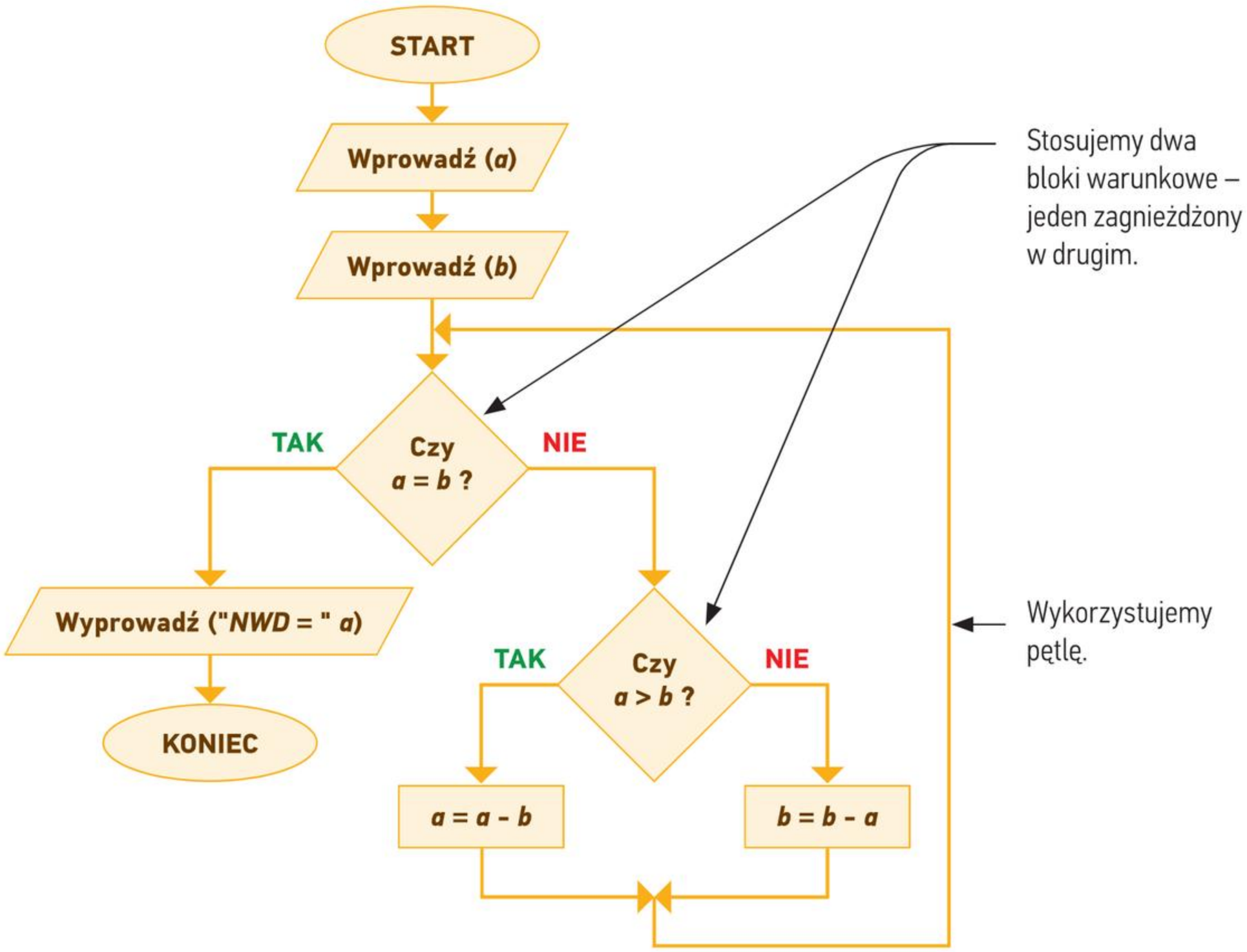
Zadanie: Znajdź największy wspólny dzielnik dwóch niezerowych liczb naturalnych.
Wykorzystaj algorytm Euklidesa w wersji z odejmowaniem.

Specyfikacja zadania:

Dane: Liczby naturalne: a, b ($a \neq 0$ i $b \neq 0$).
Wynik: Wartość największego wspólnego dzielnika liczb a i b: NWD.

Lista kroków:

1. Zacznij algorytm.
2. Wprowadź wartość liczby a.
3. Wprowadź wartość liczby b.
4. Sprawdź, czy $a = b$: jeżeli tak, idź do kroku 7.
5. Jeżeli $a > b$, to zmiennej a przypisz wartość wyrażenia $a - b$; w przeciwnym przypadku zmiennej b przypisz wartość wyrażenia $b - a$.
6. Idź do kroku 4.
7. Wyprowadź wynik: NWD jest równe a.
8. Zakończ algorytm.



Rys. 5. Schemat blokowy algorytmu Euklidesa w wersji z odejmowaniem

a	*b*	Czy *a* = *b*?	Czy *a* > *b*?	Działania
Pierwsza para liczb: *a* = 4, *b* = 8				
4	8	**NIE**	**NIE**	$b = 8 - 4 = 4$
4	4	**TAK**	–	***NWD*** $= a =$ **4**
Druga para liczb: *a* = 25, *b* = 15				
25	15	**NIE**	**TAK**	$a = 25 - 15 = 10$
10	15	**NIE**	**NIE**	$b = 15 - 10 = 5$
10	5	**NIE**	**TAK**	$a = 10 - 5 = 5$
5	5	**TAK**	–	***NWD*** $= a =$ **5**

Tabela 2. Testowanie algorytmu Euklidesa w wersji z odejmowaniem

Ćwiczenie 5. Sprawdzamy działanie algorytmu Euklidesa w wersji z odejmowaniem
Sprawdź działanie algorytmu Euklidesa w wersji z odejmowaniem i wyznacz *NWD* dla liczb: $a = 3$ i $b = 9$ oraz $a = 16$ i $b = 12$. Zapisz rozwiązanie w dokumencie tekstowym – wzoruj się na tabeli 2.

Zapisywanie algorytmu Euklidesa w wersji z odejmowaniem w języku C++

W algorytmie Euklidesa znajdowania NWD dwóch liczb naturalnych `a` i `b` w wersji z odejmowaniem występuje powtarzanie poleceń (iteracja), dlatego w programie zastosujemy instrukcję iteracyjną.

Liczba powtórzeń (iteracji) nie jest z góry określona, dlatego zastosujemy instrukcję iteracyjną **`while`**, w której będziemy sprawdzać, czy `a` jest różne od `b`.

Wewnątrz pętli **`while`** musimy też sprawdzać, która z liczb jest większa, dlatego zastosujemy instrukcję warunkową **`if`**.

```
while(warunek)
   lista_instrukcji;
kolejna_instrukcja;
```

Rys. 6. Ogólna postać instrukcji iteracyjnej **`while`** w języku C++

Komputer wykonuje instrukcje w pętli (*lista_instrukcji*), dopóki *warunek* jest prawdziwy. Jeśli *warunek* jest fałszywy, wykonuje *kolejną_instrukcję*. W bloku *lista_instrukcji* powinna być instrukcja, która zmienia wartość *warunku* – w przeciwnym wypadku pętla nigdy się nie skończy.

[*] Euklides_odejmowanie.cpp

```
#include <iostream>
using namespace std;

int main()
{
    unsigned int a, b;

    cout << "Podaj dwie liczby: " << endl;
    cin >> a;
    cin >> b;

    while (a != b)
    {
        if (a > b)
            a = a - b;
        else
            b = b - a;
    }

    cout << "NWD = " << a << endl;
    return 0;
}
```

Dopóki a jest różne od b, odejmujemy od większej liczby mniejszą i zastępujemy większą liczbę otrzymaną różnicą.

Rys. 7. Program w języku C++ realizujący algorytm Euklidesa w wersji z odejmowaniem

Ćwiczenie 6. Zapisujemy algorytm Euklidesa w wersji z odejmowaniem w języku C++

1. Program pokazany na rysunku 7. realizuje algorytm Euklidesa w wersji z odejmowaniem. Przepisz ten program i zapisz go w pliku pod nazwą *Euklides_odejmowanie*. Skompiluj i uruchom kilkakrotnie program, testując go dla różnych wartości zmiennych a i b.
2. Objaśnij działanie programu, m.in. użycie poszczególnych instrukcji zgodnie z listą kroków lub schematem blokowym algorytmu.

Ćwiczenie 7. Modyfikujemy program

1. Zmodyfikuj program zapisany w ćwiczeniu 6. tak, aby szukał NWD dla dziesięciu par liczb a i b wprowadzanych kolejno z klawiatury i wyświetlał wyniki na ekranie.
2. Zapisz program w pliku pod nazwą *Euklides_odejmowanie_10*.

Działanie algorytmu Euklidesa w wersji z resztą z dzielenia

Algorytm Euklidesa w wersji z dzieleniem polega na wykonywaniu kolejnych dzieleń liczby *a* (dzielnej) przez liczbę *b* (dzielnik), zastępowaniu dzielnej dzielnikiem, a dzielnika resztą z dzielenia i wykonywaniu obliczeń aż reszta z dzielenia osiągnie wartość 0.

Na przykład dla liczb $a = 25$ i $b = 10$ NWD obliczamy następująco:

- dzielimy liczbę a przez b:
  $a / b = 25 / 10 = 2$ reszta 5,
- liczbę a zastępujemy liczbą b, a liczbę b – resztą z dzielenia, czyli:
  $a = 10$, $b = 5$,
- dzielimy liczbę a przez b:
  $a / b = 10 / 5 = 2$ reszta 0,
- liczbę a zastępujemy liczbą b, a liczbę b – resztą z dzielenia, czyli:
  $a = 5$, $b = 0$.

Kończymy algorytm, ponieważ $b = 0$, czyli reszta z dzielenia wynosi 0. Przyjmujemy, że NWD jest równe ostatniej niezerowej reszcie, czyli $NWD = a = 5$.

Algorytm Euklidesa – wersja z resztą z dzielenia

Zadanie: Znajdź największy wspólny dzielnik dwóch niezerowych liczb naturalnych. Wykorzystaj algorytm Euklidesa w wersji z resztą z dzielenia.

Specyfikacja zadania:

Dane: Liczby naturalne: a, b ($a \neq 0$ i $b \neq 0$).

Wynik: Wartość największego wspólnego dzielnika liczb a i b: *NWD*.

Lista kroków:

1. Zacznij algorytm.
2. Wprowadź wartość liczby a.
3. Wprowadź wartość liczby b.
4. Sprawdź, czy $b = 0$: jeżeli tak, idź do kroku 9.
5. Zmiennej *dzielnik* przypisz wartość zmiennej b.
6. Zmiennej b przypisz wartość reszty z dzielenia a przez b.
7. Zmiennej a przypisz wartość zmiennej *dzielnik*.
8. Idź do kroku 4.
9. Wyprowadź wynik: *NWD* jest równe a.
10. Zakończ algorytm.

a	b	Czy $b = 0$?	Działania
Pierwsza para liczb: $a = 9$, $b = 3$			
9	3	**NIE**	$dzielnik = b = 3$ $b = a$ **mod** $b = 9$ **mod** $3 = 0$ $a = dzielnik = 3$
3	0	**TAK**	***NWD*** $= a = 3$

Druga para liczb: $a = 28$, $b = 35$			
28	35	**NIE**	*dzielnik* = $b = 35$ $b = a$ **mod** $b = 28$ **mod** $35 = 28$ $a = dzielnik = 35$
35	28	**NIE**	*dzielnik* = $b = 28$ $b = a$ **mod** $b = 35$ **mod** $28 = 7$ $a = dzielnik = 28$
28	7	**NIE**	*dzielnik* = $b = 7$ $b = a$ **mod** $b = 28$ **mod** $7 = 0$ $a = dzielnik = 7$
7	0	**TAK**	***NWD*** $= a = 7$

Tabela 3. Testowanie algorytmu Euklidesa w wersji z resztą z dzielenia

Uwaga: Jeśli pod tę samą zmienną podstawimy kolejną wartość, poprzednia wartość znika. Dlatego musimy wartość zmiennej *b* przechować w innej zmiennej, tzw. zmiennej pomocniczej. W omawianym algorytmie jest to zmienna *dzielnik*, której jest przypisany aktualny dzielnik.

Ćwiczenie 8. Sprawdzamy działanie algorytmu Euklidesa w wersji z resztą z dzielenia

Sprawdź działanie algorytmu Euklidesa w wersji z resztą z dzielenia i wyznacz *NWD* dla liczb: $a = 56$ i $b = 6$ oraz $a = 16$ i $b = 24$. Zapisz rozwiązanie w dokumencie tekstowym – wzoruj się na tabeli 3.

Zapisywanie algorytmu Euklidesa w wersji z resztą z dzielenia w języku C++

W algorytmie Euklidesa znajdowania NWD dwóch liczb naturalnych `a` i `b` w wersji z resztą z dzielenia występuje powtarzanie poleceń (iteracja), dlatego w programie zastosujemy instrukcję iteracyjną.

Liczba iteracji (podobnie jak w wersji z odejmowaniem) nie jest z góry określona, dlatego zastosujemy instrukcję iteracyjną **`while`**, w której będziemy sprawdzać, czy `b` jest różne od zera.

Do wyznaczenia reszty z dzielenia liczby `a` przez liczbę `b` zastosujemy operator modulo `%`.

Euklides_dzielenie.cpp

```
1  #include <iostream>
2  using namespace std;
3
4  int main()
5  {
6      unsigned int a, b, dzielnik;
7
8      cout << "Podaj dwie liczby: " << endl;
9      cin >> a;
10     cin >> b;
11
12     while( b != 0 )
13     {
14         dzielnik = b;
15         b = a % b;
16         a = dzielnik;
17     }
18
19     cout << "NWD = " << a << endl;
20
21     return 0;
22 }
```

W trakcie wykonywania obliczeń zmienna `dzielnik` przechowuje aktualny dzielnik.

Dopóki `b` jest różne od 0, przypisujemy: zmiennej `dzielnik` wartość zmiennej `b`, zmiennej `b` resztę z dzielenia `a` przez `b`, zmiennej `a` wartość zmiennej `dzielnik`.

Rys. 8. Program w języku C++ realizujący algorytm Euklidesa w wersji z resztą z dzielenia

Ćwiczenie 9. Zapisujemy algorytm Euklidesa w wersji z resztą z dzielenia w języku C++

1. Program, którego kod źródłowy pokazano na rysunku 8., realizuje algorytm Euklidesa w wersji z resztą z dzielenia. Przepisz ten program i zapisz go w pliku pod nazwą *Euklides_dzielenie*. Skompiluj i uruchom kilkakrotnie program, testując go dla różnych wartości zmiennych `a` i `b`.
2. Objaśnij działanie programu, m.in. użycie poszczególnych instrukcji zgodnie z listą kroków algorytmu.

Ćwiczenie 10. Modyfikujemy program

1. Zmodyfikuj program zapisany w ćwiczeniu 9. tak, aby szukał NWD dla dziesięciu par liczb `a` i `b` wprowadzanych kolejno z klawiatury i wyświetlał wyniki na ekranie.
2. Zapisz program w pliku pod nazwą *Euklides_dzielenie_10*.

3 Wyodrębnianie cyfr danej liczby naturalnej

System dziesiętny opiera się na potęgach liczby 10. Liczbę naturalną w systemie dziesiętnym możemy rozpisać w następujący sposób:
$\mathbf{647} = \mathbf{6} \cdot 100 + \mathbf{4} \cdot 10 + \mathbf{7} \cdot 1 = \mathbf{6} \cdot 10^2 + \mathbf{4} \cdot 10^1 + \mathbf{7} \cdot 10^0$,
$\mathbf{92} = \mathbf{9} \cdot 10 + \mathbf{2} \cdot 1 = \mathbf{9} \cdot 10^1 + \mathbf{2} \cdot 10^0$,
$\mathbf{8} = \mathbf{8} \cdot 1 = \mathbf{8} \cdot 10^0$.

Aby wyodrębnić cyfry danej liczby (od cyfry jedności), powtarzamy obliczanie reszty z dzielenia liczby przez 10 (pierwsza reszta jest cyfrą jedności), następnie od liczby odejmujemy resztę (cyfrę) i wynik dzielimy przez 10, aż do otrzymania wyniku dzielenia równego zero.

Cyfry danej liczby możemy wyodrębniać od ostatniej (dla liczby 647 jest to cyfra jedności: 7) lub od pierwszej (dla 647 to cyfra setek: 6).

Do obliczania reszty z dzielenia zastosujemy operator modulo **mod**.

Algorytm wyodrębniania cyfr danej liczby naturalnej

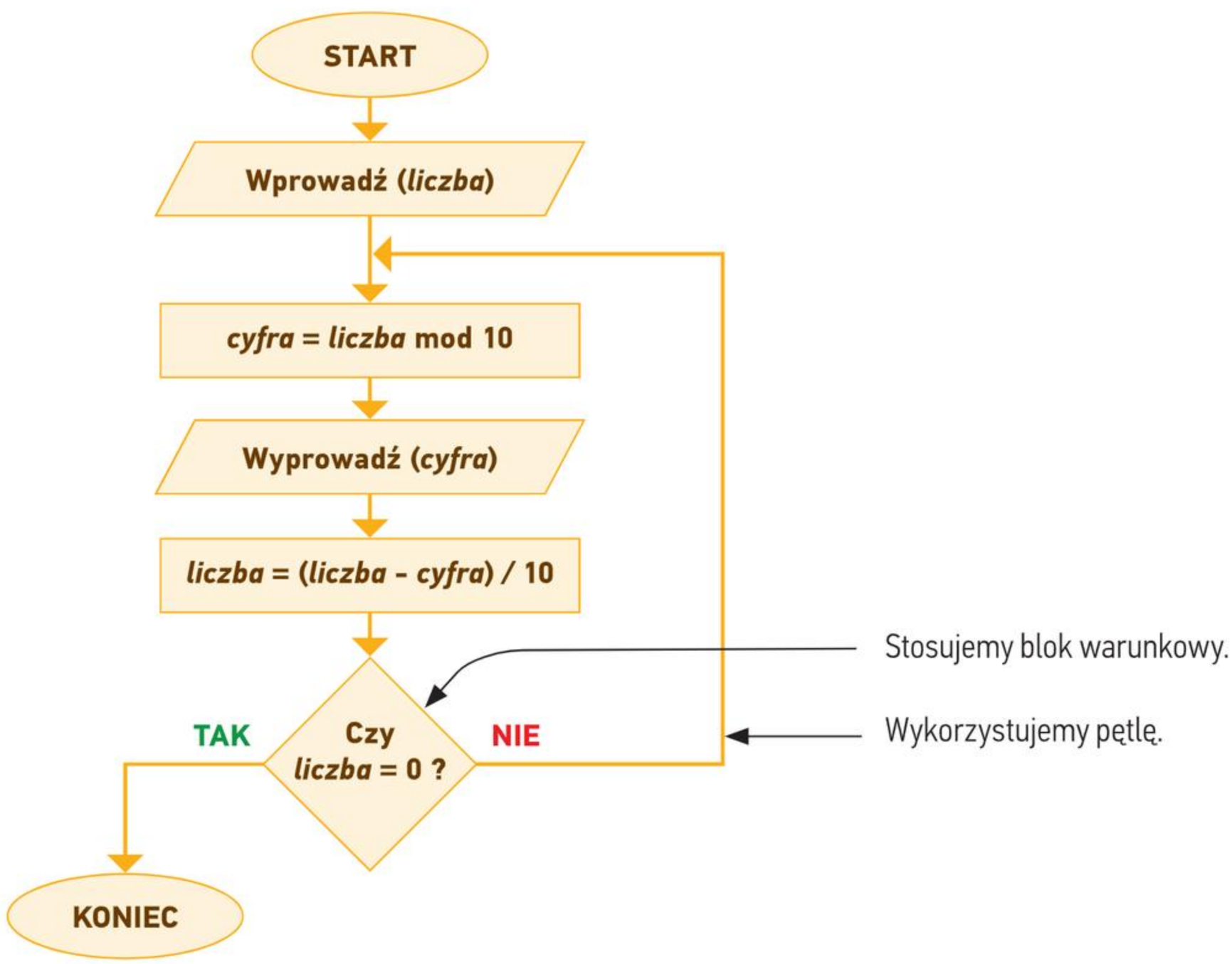
Zadanie: Wyodrębnij cyfry danej liczby naturalnej różnej od 0, zaczynając od cyfry jedności.

Specyfikacja zadania:

Dane: Liczba naturalna różna od 0: *liczba*.

Wynik: Cyfry danej liczby, zaczynając od cyfry jedności.

Uwagi: Zmienna *cyfra* służy do pamiętania wartości kolejnych cyfr (tu: reszt z dzielenia kolejnych wartości zmiennej *liczba* przez 10).Wartość zmiennej *liczba*, której cyfry wyodrębniamy, zmienia się w czasie wykonywania algorytmu.



Rys. 9. Schemat blokowy algorytmu wyodrębniania cyfr danej liczby od cyfry jedności

liczba	Czy *liczba* = 0?	Działania	Cyfra
647	NIE	*cyfra* = 647 **mod** 10 = **7** *liczba* = (647 – 7) / 10 = 64	**7**
64	NIE	*cyfra* = 64 **mod** 10 = **4** *liczba* = (64 – 4) / 10 = 6	**4**
6	NIE	*cyfra* = 6 **mod** 10 = **6** *liczba* = (6 – 6) / 10 = 0	**6**
0	TAK	–	–

Tabela 4. Testowanie algorytmu wyodrębniania cyfr danej liczby

Ćwiczenie 11. Piszemy listę kroków algorytmu wyodrębniania cyfr danej liczby

1. Zapoznaj się z opisem algorytmu wyodrębniania cyfr danej liczby.
2. Przeanalizuj schemat blokowy algorytmu, korzystając z przykładu podanego w tabeli 4. Sprawdź działanie algorytmu dla liczby 8945.
3. Zapisz w dokumencie tekstowym listę kroków tego algorytmu.

Wskazówka: Wzoruj się na zapisach list kroków innych algorytmów opisanych w tym temacie.

Zapisywanie algorytmu wyodrębniania cyfr danej liczby w języku C++

W algorytmie wyodrębniania cyfr danej liczby występuje powtarzanie poleceń, dlatego w programie zastosujemy instrukcję iteracyjną.

Liczba iteracji nie jest z góry określona, dlatego zastosujemy instrukcję iteracyjną **do** ... **while**, w której warunek (czy `liczba` jest różna od 0?) sprawdzamy na końcu. W ten sposób instrukcje umieszczone w bloku instrukcji **do** ... **while** komputer wykona przynajmniej raz.

```
do
   lista_instrukcji;
while(warunek);
kolejna_instrukcja;
```

Rys. 10. Ogólna postać działania instrukcji **do** ... **while** w języku C++

Komputer wykonuje instrukcje w pętli (*listę_instrukcji*), dopóki warunek jest prawdziwy. Jeśli warunek jest fałszywy – komputer wykonuje *kolejną_ instrukcję*. Niezależnie od wartości początkowej warunku komputer wykona *listę_instrukcji* przynajmniej raz. Jako *lista_instrukcji* może wystąpić pojedyncza instrukcja (w tym instrukcja **do** ... **while**) lub więcej instrukcji ujętych w blok {}.

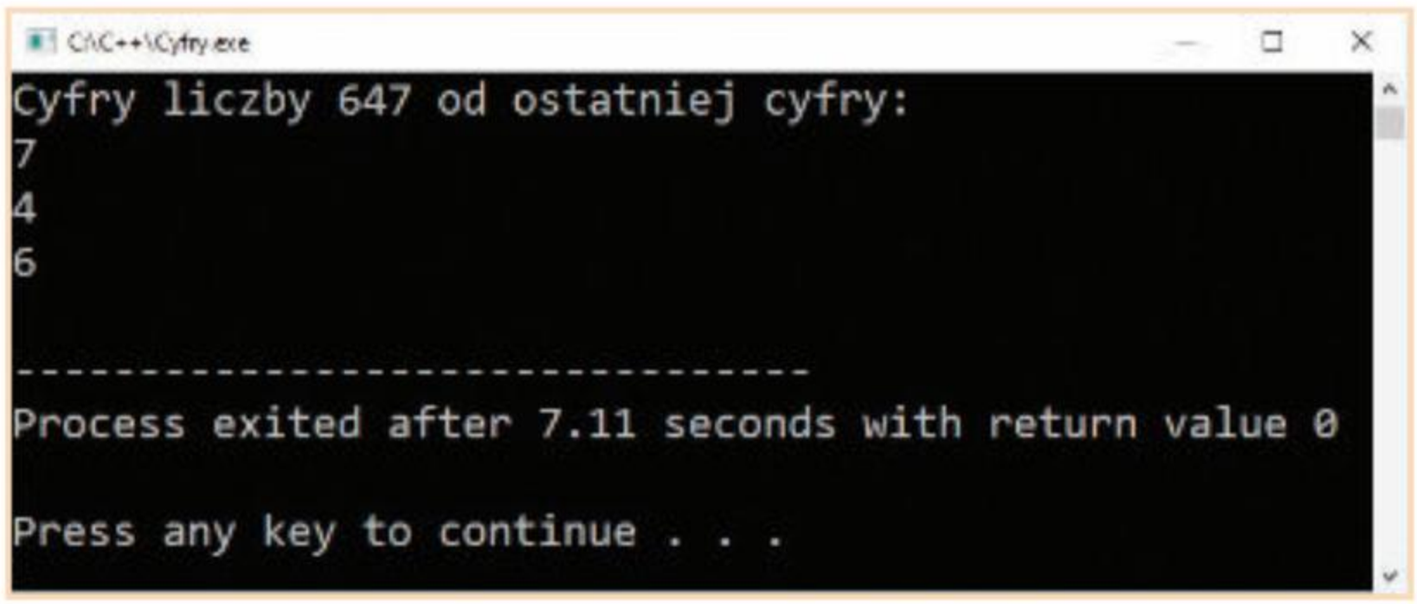
Cyfry.cpp

```
1  #include <iostream>
2  using namespace std;
3
4  int main()
5  {
6      unsigned int liczba, cyfra;
7
8      cout << "Podaj liczbę: ";
9      cin >> liczba;
10     cout << "Cyfry liczby " << liczba << " od ostatniej cyfry:" << endl;
11
12     do
13     {
14         cyfra = liczba % 10;
15         cout << cyfra << endl;
16         liczba = (liczba - cyfra) / 10;
17     }
18     while (liczba != 0 );
19
20     return 0;
21 }
```

Typ zmiennych `liczba` i `cyfry` określamy jako **`unsigned int`** – liczby naturalne.

Dopóki wartość zmiennej `liczba` jest różna od zera, obliczamy resztę z dzielenia wartości zmiennej `liczba` przez 10, wyświetlamy wyodrębnioną cyfrę i ponownie obliczamy wartość zmiennej `liczba`.

Rys. 11. Program w języku C++ realizujący algorytm wyodrębniania cyfr danej liczby



Rys. 12. Wynik działania programu – ćwiczenie 12.

Ćwiczenie 12. Zapisujemy algorytm wyodrębniania cyfr danej liczby w języku C++

1. Program pokazany na rysunku 11. realizuje algorytm wyodrębniania cyfr danej liczby, zaczynając od ostatniej cyfry (cyfry jedności). Przepisz ten program i zapisz go w pliku pod nazwą *Cyfry*. Skompiluj i uruchom kilkakrotnie program, testując go dla różnych wartości zmiennej `liczba`.
2. Objaśnij działanie programu, m.in. użycie poszczególnych instrukcji.

Ćwiczenie 13. Modyfikujemy program

1. Zmodyfikuj program zapisany w ćwiczeniu 12, aby wyodrębniał cyfry dziesięciu liczb wprowadzanych z klawiatury i wyświetlał kolejne wyniki na ekranie. Zapisz program w pliku pod nazwą *Cyfry_10*.
2. Odpowiedz na pytanie: *Ile razy powtarzają się polecenia w pętli* **`do ... while`** *dla liczby trzycyfrowej, a ile razy – dla jednocyfrowej?* Uzasadnij odpowiedź.
3. Zamień instrukcję: `liczba = (liczba - cyfra) / 10`
   na instrukcję: `liczba = liczba / 10`.
   Czy zmieniło się działanie programu? Uzasadnij odpowiedź.

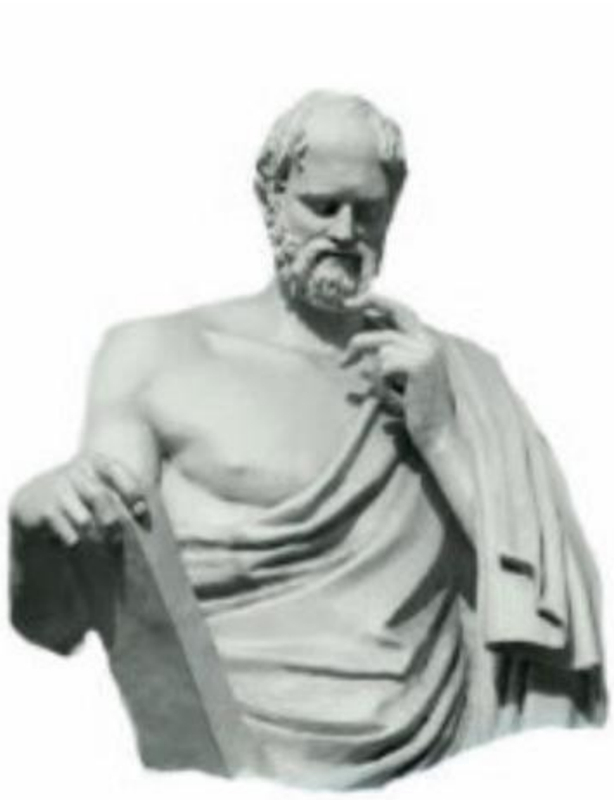
Zapamiętaj

- Liczba *a* jest podzielna przez liczbę *b*, jeśli reszta z dzielenia *a* przez *b* jest równa zero.
- W języku C ++ w programie badania podzielności liczb, w którym występuje sytuacja warunkowa, stosujemy instrukcję warunkową **`if`**.
- Największy wspólny dzielnik (NWD) dwóch liczb naturalnych możemy znaleźć, stosując algorytm Euklidesa. Algorytm ten możemy zrealizować w dwóch wersjach: z odejmowaniem i z resztą z dzielenia.
- Aby zapisać w języku C++ algorytm Euklidesa w wersji z odejmowaniem i w wersji z resztą z dzielenia, możemy zastosować instrukcję iteracyjną **`while`**. W obydwu algorytmach występuje powtarzanie poleceń (iteracja) – są to algorytmy iteracyjne.
- W algorytmie wyodrębniania cyfr danej liczby naturalnej wykorzystujemy reszty z dzielenia danej liczby przez dziesięć. Reszty z dzielenia są kolejnymi cyframi liczby.
- Gdy zapisujemy algorytm wyodrębniania cyfr danej liczby naturalnej w języku C++, możemy zastosować instrukcję iteracyjną **`do ... while`**. W algorytmie występuje powtarzanie poleceń (iteracja) – jest to algorytm iteracyjny.

Serwis dla ciekawskich

Euklides z Aleksandrii był greckim matematykiem, który ok. 300 lat p.n.e. stworzył dzieło pt. „Elementy”. Opisał w nim m.in. sposób obliczania największej wspólnej miary dwóch odcinków (czyli największego wspólnego dzielnika dwóch liczb naturalnych).

„Elementy” były przez wiele wieków podstawowym podręcznikiem do nauki geometrii.



OKO W OKO Z MONITOREM

Pytania i polecenia

1. Jak sprawdzić, czy liczba *x* jest podzielna przez liczbę *y*? Przedstaw na przykładzie.
2. Dlaczego w instrukcji warunkowej programu realizującego algorytm badania podzielności liczb naturalnych warunek logiczny ma postać `a % b == 0`? Uzasadnij odpowiedź.
3. Wyjaśnij na przykładzie liczb 36 i 8, dlaczego warunkiem zakończenia algorytmu Euklidesa w wersji z odejmowaniem jest `a = b`.
4. Czym różni się wersja algorytmu Euklidesa z odejmowaniem od wersji z resztą z dzielenia? Wyjaśnij na przykładzie liczb 36 i 8.
5. Czy w programie realizującym algorytm Euklidesa w obydwu wersjach można zastosować instrukcję iteracyjną **`for`** zamiast **`while`**? Uzasadnij odpowiedź.
6. Na czym opiera się idea algorytmu wyodrębniania cyfr danej liczby?
7. Na czym polega działanie instrukcji iteracyjnej **`do`** ... **`while`** w algorytmie wyodrębniania cyfr danej liczby?

Zadania

1. W dokumencie tekstowym napisz specyfikację zadania i listę kroków algorytmu sprawdzania, czy dana liczba naturalna jest parzysta. Jeśli liczba jest parzysta, wyprowadź napis „Podana liczba jest parzysta", a gdy nieparzysta – „Podana liczba jest nieparzysta".
2. Utwórz schemat blokowy algorytmu sprawdzania, czy dana liczba naturalna jest parzysta.
3. Napisz program sprawdzający, czy dana liczba naturalna jest parzysta. Zapisz program w pliku pod nazwą *Parzyste*.
4. Napisz program sprawdzający, czy liczba `a` wprowadzona z klawiatury jest podzielna przez 7. Program ma umożliwiać wprowadzenie i sprawdzenie dziesięciu liczb. Zapisz program w pliku pod nazwą *Podzielność_przez_7*.
5. Przedstaw w postaci schematu blokowego algorytm Euklidesa w wersji z resztą z dzielenia.

DLA ZAINTERESOWANYCH

1. Zastanów się, jak sprawdzić podzielność jednej liczby przez inną bez konieczności obliczania reszty z dzielenia. Zapisz w dokumencie tekstowym listę kroków tego algorytmu lub przedstaw go w postaci schematu blokowego.

2. Napisz program sprawdzający podzielność jednej liczby przez inną bez konieczności obliczania reszty z dzielenia. Zapisz program w pliku pod nazwą *Podzielność_inaczej*.

3. W dokumencie tekstowym napisz specyfikację zadania i listę kroków oraz utwórz schemat blokowy algorytmu wyodrębniania cyfr danej liczby, zaczynając od cyfry pierwszej od lewej.

4. Napisz program służący do wyodrębniania cyfr danej liczby od cyfry pierwszej od lewej do ostatniej. Zapisz program w pliku pod nazwą *Cyfry_odwrotnie*.

3.4. Algorytmy wyszukiwania

Warto powtórzyć

1. Na czym polega iteracja?
2. W jaki sposób zapisujemy iterację w języku C++?
3. Jaka jest różnica między instrukcją **`for`** a instrukcją **`do ... while`**?
4. W jaki sposób nadajemy wartość zmiennej w języku C++?

Podczas rozwiązywania niektórych problemów (z różnych dziedzin nauki i życia codziennego) szukamy w danym zbiorze konkretnego elementu, np. najwyższej osoby, książki danego autora. Istnieją różne algorytmy wyszukiwania.

W tym temacie:

1. poznasz i zaprogramujesz algorytm wyszukiwania elementu w zbiorze nieuporządkowanym,
2. poznasz i zaprogramujesz algorytm wyszukiwania elementu w zbiorze uporządkowanym.

1 Wyszukiwanie elementu w zbiorze nieuporządkowanym

Wyszukiwanie elementu w zbiorze nieuporządkowanym pokażemy na przykładzie znajdowania największego elementu w zbiorze. Wyjaśnimy również, jak sprawdzić, czy w zbiorze nieuporządkowanym istnieje dany element.

Przyjmiemy, że zbiór do przeszukania zawiera n elementów, które są liczbami całkowitymi.

Wyszukiwanie największej liczby w zbiorze nieuporządkowanym

Aby znaleźć największą spośród n liczb, porównujemy kolejne wartości liczb ze zbioru z największą znalezioną do tej pory liczbą (zapamiętaną osobno). Na początku algorytmu przyjmujemy, że pierwsza liczba ze zbioru jest największa. Porównania liczb powtarzamy, aż sprawdzimy wszystkie liczby ze zbioru.

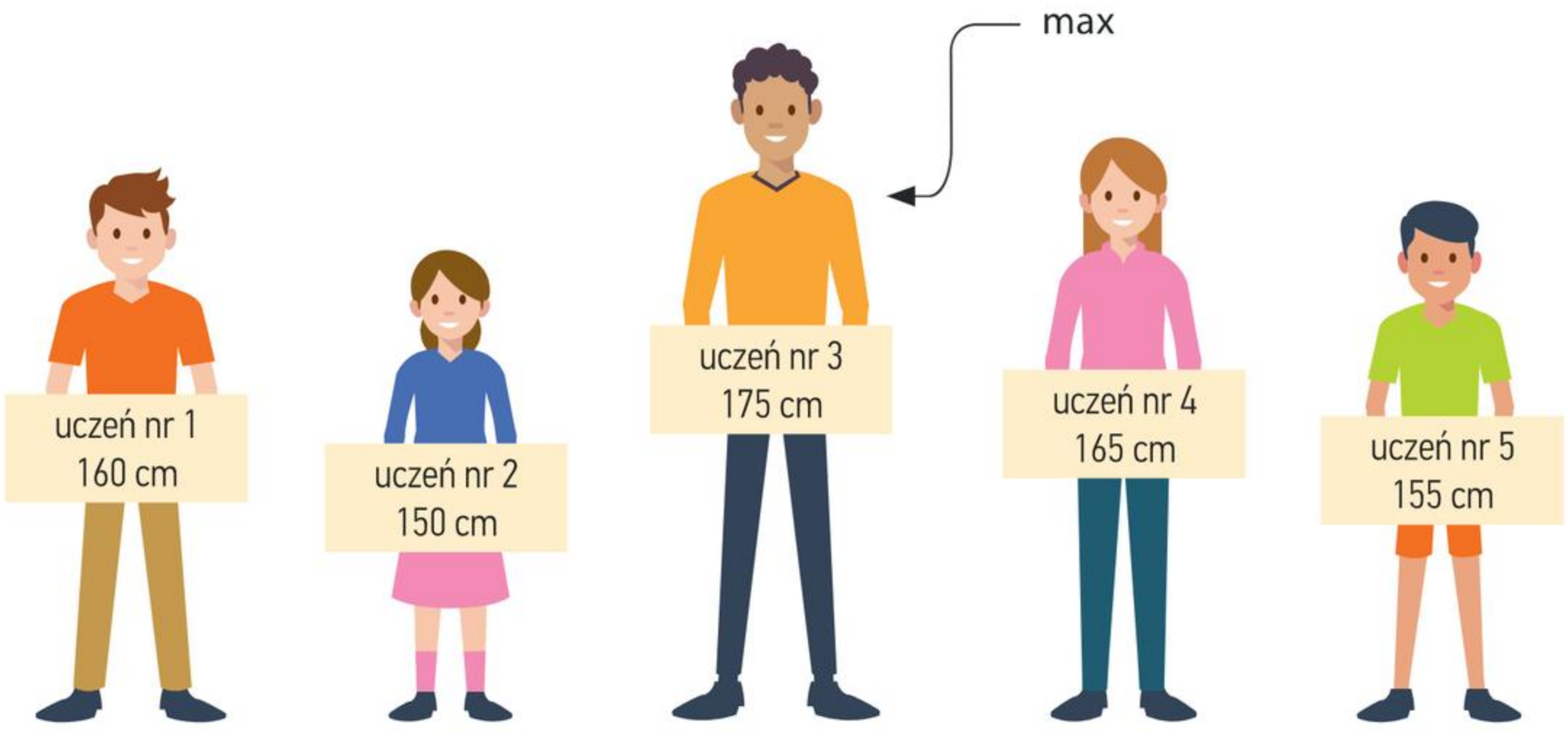
Algorytm wyszukiwania największej liczby spośród *n* liczb

Zadanie: Znajdź największą liczbę spośród *n* liczb całkowitych.
Specyfikacja zadania:
Dane: Liczba naturalna *n* oznaczająca liczbę elementów zbioru, *n* liczb całkowitych wprowadzanych kolejno i zapamiętywanych w zmiennej *a*.
Wynik: Wartość elementu największego: *maks*.
Lista kroków:

1. Zacznij algorytm.
2. Wprowadź liczbę elementów zbioru: *n*.
3. Wprowadź wartość pierwszej liczby: *a*.
4. Zmiennej *maks* przypisz wartość pierwszej liczby: *maks* = *a*.
5. Wprowadź wartość kolejnej liczby: *a*.
6. Porównaj kolejną liczbę ze zbioru z *maks*: czy *a* > *maks*?
7. Jeśli kolejna liczba jest większa od *maks*, to zmiennej *maks* przypisz wartość tej liczby: *maks* = *a*.
8. Jeśli nie jest to ostatnia liczba, wróć do kroku 5.
9. Wyprowadź wynik: *maks*.
10. Zakończ algorytm.

Algorytm wyszukiwania największej liczby spośród *n* liczb zastosujemy do znajdowania wzrostu najwyższego z *n* uczniów (gdzie *n* to liczba uczniów klasy). Sprawdzimy działanie algorytmu dla *n* = 5. W tabeli na rysunku 1. podano wzrost uczniów w centymetrach (czyli kolejne wartości *a*).

	uczeń nr 1	uczeń nr 2	uczeń nr 3	uczeń nr 4	uczeń nr 5
wzrost w cm	160	150	175	165	155



Rys. 1. Przykład użycia algorytmu wyszukiwania największej spośród *n* liczb

a	czy *a* > *maks*?	odpowiedź	działania
wprowadź wzrost pierwszego ucznia: *a* = 160	–	–	zmiennej *maks* przypisz wzrost pierwszego ucznia: *maks* = 160
wprowadź wzrost drugiego ucznia: *a* = 150	porównaj wzrost drugiego ucznia z wartością *maks*: czy 150 > 160?	**NIE**	pozostaw *maks* bez zmian: *maks* = 160
wprowadź wzrost trzeciego ucznia: *a* = 175	porównaj wzrost trzeciego ucznia z wartością *maks*: czy 175 > 160?	**TAK**	zmiennej *maks* przypisz wzrost trzeciego ucznia: *maks* = 175
wprowadź wzrost czwartego ucznia: *a* = 165	porównaj wzrost czwartego ucznia z wartością *maks*: czy 165 > 175?	**NIE**	pozostaw *maks* bez zmian: *maks* = 175
wprowadź wzrost piątego ucznia: *a* = 155	porównaj wzrost piątego ucznia z wartością *maks*: czy 155 > 175?	**NIE**	pozostaw *maks* bez zmian: *maks* = 175
–	–	–	wzrost najwyższego ucznia to 175 cm

Tabela 1. Przykład wykonania algorytmu dla *n* = 5

Ćwiczenie 1. Sprawdzamy działanie algorytmu wyszukiwania największej liczby spośród pięciu liczb

Przedstaw wspólnie z koleżankami i kolegami z klasy algorytm wyszukiwania najwyższego ucznia spośród pięciu uczniów. Możecie przygotować odpowiednie pomoce dydaktyczne i się nimi posłużyć (rys. 1.).

Ćwiczenie 2. Sprawdzamy działanie algorytmu wyszukiwania największej liczby spośród *n* liczb

Sprawdź działanie algorytmu wyszukiwania największej liczby:

a. dla zbioru liczb {70, 160, 155, 180, 185};

b. dla zbioru liczb {8, –6, 23, 0, –50, 34, –23};

c. dla dziesięciu wybranych liczb całkowitych.

Zapisujemy w języku C++ algorytm wyszukiwania największego elementu w zbiorze

Algorytm wyszukiwania największego elementu w zbiorze zapiszemy w języku C++. Zdefiniujemy funkcję `maksimum()` z jednym parametrem wyszukującą największy element w *n*-elementowym zbiorze liczb całkowitych. Parametrem przekazywanym funkcji jest liczba elementów zbioru. Funkcja będzie zwracać wartość największego elementu (zapisaną w zmiennej `maks`).

[*] Maksimum_n.cpp

```
1  #include <iostream>
2  using namespace std;
3
4  int maksimum(int n)
5  {
6      int i, a, maks;
7
8      for(i = 0; i < n; i++)
9      {
10         cout << "Podaj liczbę: ";
11         cin >> a;
12         if(i == 0)
13             maks = a;
14         else
15             if(a > maks)
16                 maks = a;
17     }
18     return maks;
19 }
20
21 int main()
22 {
23     int liczba_elementow, najwiekszy;
24
25     cout << "Podaj liczbę elementów: ";
26     cin >> liczba_elementow;
27     najwiekszy = maksimum(liczba_elementow);
28     cout << "Maksimum to: " << najwiekszy;
29
```

Na początku przyjmujemy, że największą liczbą jest pierwsza liczba ze zbioru, dlatego pod zmienną `maks` podstawiamy wartość pierwszej wprowadzonej liczby.

Jeśli kolejna liczba jest większa od `maks`, to zmiennej `maks` przypisujemy wartość tej liczby.

Funkcja zwraca wartość największego elementu zapisaną w zmiennej `maks`.

Wywołanie funkcji `maksimum()` z parametrem aktualnym `liczba_elementow` – wynik przypisujemy zmiennej `najwiekszy`.

Rys. 2. Program w języku C++ realizujący algorytm wyszukiwania największej liczby w zbiorze *n*-elementowym – ćwiczenie 3.

Ćwiczenie 3. Zapisujemy w języku C++ algorytm wyszukiwania największej liczby w zbiorze *n*-elementowym

1. Program w języku C++ pokazany na rysunku 2. realizuje algorytm wyszukiwania największego elementu w zbiorze *n*-elementowym. Przepisz ten program i zapisz go w pliku pod nazwą *Maksimum_n*.
2. Skompiluj i uruchom program. Sprawdź jego działanie dla różnych danych, w tym dla danych pokazanych na rysunku 1.
3. Objaśnij działanie programu, m.in. uzasadnij użycie poszczególnych poleceń zgodnie z podaną listą kroków.

Opisany w liście kroków algorytm znajdowania największego elementu w zbiorze jest przykładem metody nazywanej przeszukiwaniem liniowym.

W algorytmie przeszukiwania liniowego sprawdzamy wartość elementów zbioru po kolei, według ustalonego wcześniej porządku.

Algorytm wyszukiwania danego elementu w zbiorze nieuporządkowanym

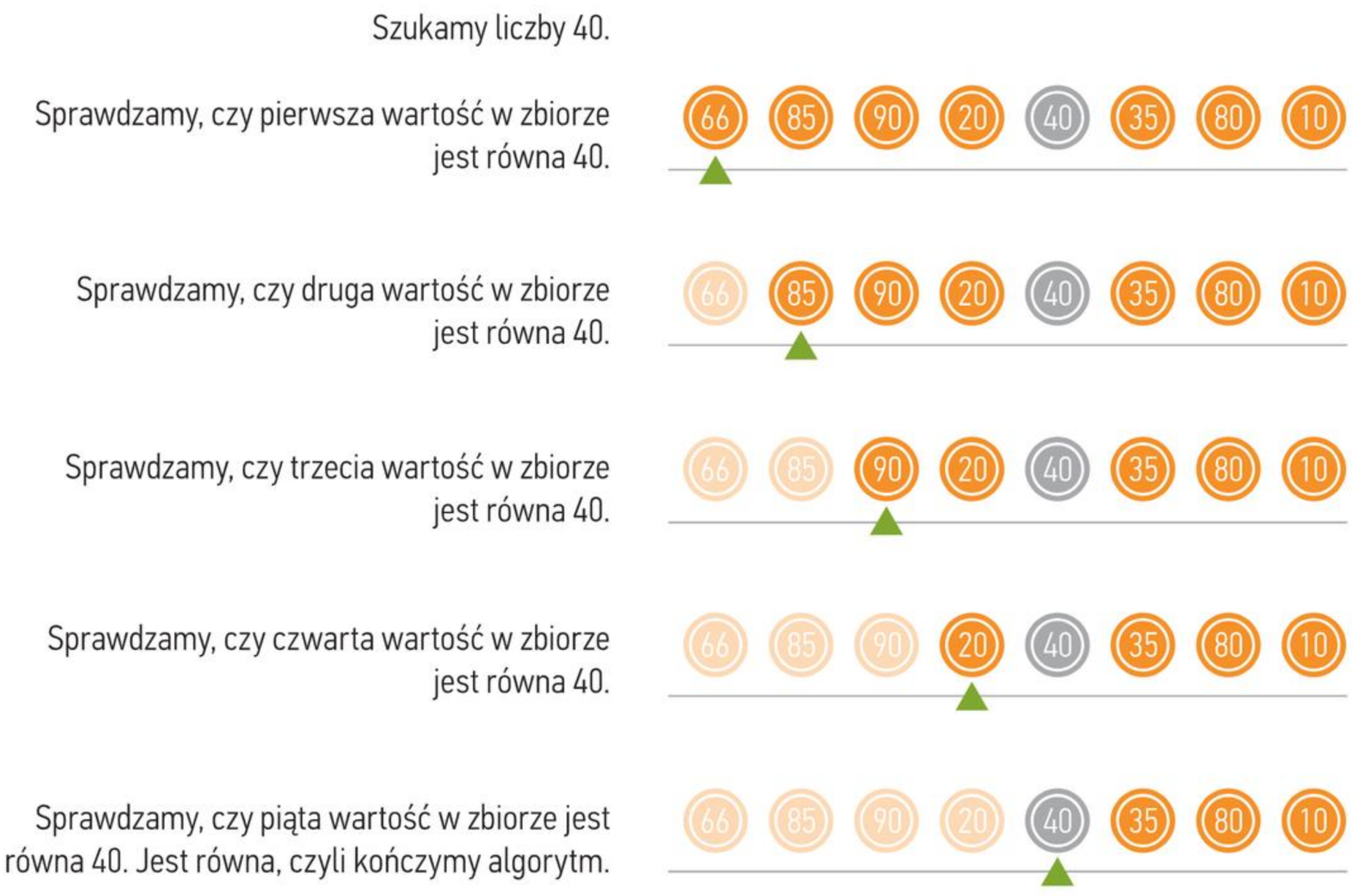
W algorytmie wyszukiwania elementu w zbiorze nieuporządkowanym szukamy danego elementu i równocześnie jego położenia w tym zbiorze.

W sytuacji, kiedy szukany element występuje w zbiorze więcej niż jeden raz, algorytm kończy działanie po natknięciu się na niego po raz pierwszy.

Aby stwierdzić, czy dany element istnieje w zbiorze n-elementowym, musimy sprawdzić każdy z elementów zbioru, ponieważ może on znajdować się w dowolnym miejscu (nawet być ostatnim elementem). W szczególnym przypadku szukanego elementu może w zbiorze w ogóle nie być.

Opis algorytmu: Dany jest nieuporządkowany zbiór ośmiu liczb (rys. 3.). Poszukiwana liczba to 40 (zaznaczona na szaro). Zakładamy, że w zbiorze jest tylko jedna taka liczba. Przeglądamy wszystkie kolejne elementy zbioru, stosując przeszukiwanie liniowe.

Zaczynamy od sprawdzenia, czy pierwsza liczba (zaznaczona zieloną strzałką) jest szukaną liczbą. Jeśli tak, kończymy poszukiwanie. Jeżeli nie – sprawdzamy kolejną liczbę. Algorytm kończy się, gdy znajdziemy liczbę równą szukanej, a jeśli jej nie ma – gdy dojdziemy do końca zbioru.



Rys. 3. Przykład wyszukiwania liczby w zbiorze nieuporządkowanym

Ćwiczenie 4. Sprawdzamy działanie algorytmu wyszukiwania liczby w zbiorze nieuporządkowanym

Przygotuj odpowiednie pomoce dydaktyczne (rys. 3.) i przedstaw wspólnie z koleżankami i kolegami z klasy algorytm wyszukiwania liczby w zbiorze nieuporządkowanym.

Wskazówka: Możesz zapisać liczby na kartkach i układać je na stole lub podłodze.

Zapisujemy w języku C++ algorytm wyszukiwania elementu w zbiorze nieuporządkowanym

Algorytm wyszukiwania elementu w zbiorze nieuporządkowanym zapiszemy w języku C++. Elementy zbioru wprowadzimy do tablicy.

Program ma wyszukiwać w zbiorze nieuporządkowanym daną wprowadzoną przez użytkownika i podawać jej pozycję w tablicy (indeks). Przyjmujemy, że w zbiorze jest pięć elementów.

Zdefiniujemy funkcję `wyszukaj _ dana()` z parametrem `wartosc` (rys. 4.):

- Jeśli element zbioru (`a[i]`) jest szukaną daną (zmienna `wartosc`), to funkcja zwraca wartość zmiennej `i`, czyli pozycję danego elementu.
- Jeśli w przeszukiwanym zbiorze nie ma szukanej liczby, funkcja `wyszukaj _ dana()` zwraca wartość –1.

Programiści często używają wartości niepoprawnych z punktu widzenia algorytmu (w tym przypadku indeks tablicy nie może być liczbą ujemną) do zasygnalizowania błędu lub niepowodzenia operacji.

Ćwiczenie 5. Zapisujemy w języku C++ algorytm wyszukiwania elementu w zbiorze nieuporządkowanym

1. Program pokazany na rysunku 4. realizuje algorytm wyszukiwania elementu w zbiorze nieuporządkowanym. Przepisz ten program i zapisz go w pliku pod nazwą *Wyszukiwanie_nieuporządkowany*.
2. Skompiluj i uruchom kilkakrotnie program.
3. Sprawdź działanie programu dla różnych danych.

Wyszukiwanie_nieuporządkowany.cpp

```
#include <iostream>
using namespace std;

const  int N = 5;
int a[N];

void wprowadz_dane()
{
    for(int i = 0; i < N; i++)
    {
        cout << "Podaj daną nr " << i << ": ";
        cin >> a[i];
    }
}

void wyprowadz_dane()
{
    for(int i = 0; i < N; i++)
    {
        cout << "a[" << i << "] = " << a[i] << endl;
    }
}

int wyszukaj_dana(int wartosc)
{
    for(int i = 0; i < N; i++)
        if(a[i] == wartosc)
            return i;
    return -1;
}

int main ()
{
    int wartosc, pozycja;
    wprowadz_dane();
    cout << endl << "Wprowadzone dane:" << endl;
    wyprowadz_dane();
    cout << endl << "Liczba do wyszukania: ";
    cin >> wartosc;
    pozycja = wyszukaj_dana(wartosc);
    if(pozycja >= 0)
        cout << "Znaleziono liczbę: " << wartosc << " na pozycji: "
        << pozycja << " (pozycje liczone są od zera)" << endl;

    else
        cout << "Nie znaleziono liczby: " << wartosc << endl;

    return 0;
}
```

Funkcja `wprowadz_dane()` wprowadza dane do tablicy.

Funkcja `wyprowadz_dane()` wyprowadza na ekran dane zapisane w tablicy.

Funkcja `wyszukaj_dana()` zwraca pozycję szukanego elementu w zbiorze (indeks w tablicy `a`).

Wywołujemy funkcje `wprowadz_dane()` i `wyprowadz_dane()`.

Wywołujemy funkcję `wyszukaj_dana()`, przypisując wynik zmiennej `pozycja`.

Uwzględnimy również sytuację, gdy podanej wartości nie ma w zbiorze.

Rys. 4. Program realizujący algorytm wyszukiwania elementu w zbiorze nieuporządkowanym w języku C++ – ćwiczenie 5.

2 Wyszukiwanie elementu w zbiorze uporządkowanym metodą połowienia

Algorytm wyszukiwania przez połowienie jest przykładem metody „dziel i zwyciężaj". Polega on na dzieleniu przeszukiwanego zbioru na dwie części i zawężeniu przeszukiwania do jednej z tych części. Metoda podziału zbioru na mniejsze części pomaga szybko znaleźć poszukiwany element, czyli zwyciężyć.

Gra w zgadywanie liczby

Algorytm poszukiwania przez połowienie wykorzystuje się w zabawie w zgadywanie liczby. Zabawa polega na odgadywaniu liczby wybranej z określonego zakresu. Osoba zgadująca podaje liczbę, a druga osoba udziela odpowiedzi: „za mała" lub „za duża". Osoba zgadująca dzieli zbiór liczb, odrzucając część, w której są liczby większe lub mniejsze od ostatnio podanej, aż odgadnie właściwą liczbę.

Ćwiczenie 6. Gramy w zgadywanie liczby

Zagraj z koleżanką lub z kolegą z klasy w zgadywanie liczby. Przyjmijcie zbiór liczb naturalnych z zakresu od 1 do 100.

Wskazówka: Jeśli jesteś osobą zgadującą liczbę, najlepiej na początku podać liczbę 50. Jak sądzisz, dlaczego?

Algorytm gry w zgadywanie liczby

Zadanie: Odgadnij liczbę w zbiorze liczb naturalnych z zakresu od 1 do 100, stosując algorytm gry w zgadywanie liczby.

Specyfikacja zadania:

Dane: Zgadywana liczba pamiętana w zmiennej *szukana*, liczby naturalne z zakresu $\langle 1, 100\rangle$ podawane przez gracza i zapamiętywane kolejno w zmiennej *liczba*.
Wynik: Odgadnięta liczba: *szukana*.

Lista kroków:

1. Zacznij algorytm.
2. Zmiennej *szukana* przypisz liczbę, którą ma odgadnąć gracz.
3. Wprowadź liczbę podawaną przez gracza do zmiennej *liczba*.
4. Jeżeli *liczba* = *szukana*, idź do kroku 7.
5. Jeżeli *liczba* > *szukana*, wyświetl na ekranie napis „Za duża";
   w przeciwnym przypadku wyświetl na ekranie napis „Za mała".
6. Idź do kroku 3.
7. Wyprowadź komunikat „Odgadnięta liczba to: " i wartość zmiennej *szukana*.
8. Zakończ algorytm.

Zapisujemy algorytm gry w zgadywanie liczby w języku C++

Algorytm gry w zgadywanie liczby zapiszemy w języku C++. Wartość zmiennej `szukana` możemy wprowadzić z klawiatury (stosując polecenie `cin >>`) lub wygenerować losowo (stosując funkcję `rand()`). W pierwszym przypadku gramy z wybraną osobą, w drugim – z komputerem.

> Funkcja `rand()` generuje liczby losowe z zakresu ⟨0, RAND_MAX⟩, gdzie RAND_MAX jest stałą określającą maksymalną wartość, jaką może wygenerować funkcja `rand()`.
> Aby uzyskać liczbę z przedziału ⟨1, `n`⟩, należy obliczyć resztę z dzielenia wartości uzyskanej za pomocą funkcji `rand()` przez `n` i do wyniku dodać 1.

Na przykład:
`rand() % 6` – wygeneruje losowo liczbę naturalną z przedziału ⟨0, 5⟩,
`rand() % 100 + 1` – wygeneruje losowo liczbę naturalną z przedziału ⟨1, 100⟩.

W programie pokazanym na rysunku 5. zastosowaliśmy losowe generowanie wartości zmiennej `szukana`. Do zapisania iteracji użyliśmy instrukcji **`while`**, ponieważ szukamy liczby dopóki wprowadzana przez nas z klawiatury liczba nie będzie równa liczbie wygenerowanej przez komputer.

Zgadnij_liczbę.cpp

```
1  #include <iostream>
2  #include <stdlib.h>
3  using namespace std;
4
5  int main()
6  {
7      int liczba, szukana;
8      szukana = rand() % 100 + 1;
9      cout << "Wprowadź liczbę od 1 do 100: ";
10     cin >> liczba;
11     while (liczba != szukana)
12     {
13         if (liczba > szukana)
14             cout << "Za duża" << endl;
15         else
16             cout << "Za mała" << endl;
17         cout << "Wprowadź liczbę od 1 do 100: ";
18         cin >> liczba;
19     }
20     cout << "Gratulacje! Odgadnięta liczba to: " << szuk
21
22     return 0;
23 }
```

Dołączamy bibliotekę, w której znajduje się deklaracja funkcji `rand()`.

Zmiennej `szukana` przypisujemy liczbę z przedziału ⟨1, 100⟩ wygenerowaną losowo przez komputer.

Komputer wykonuje instrukcje w pętli dopóki wartość zmiennej `liczba` jest różna od wartości zmiennej `szukana`.

Rys. 5. Program realizujący algorytm gry w zgadywanie liczby w języku C++ – ćwiczenie 7.

Ćwiczenie 7. Tworzymy w języku C++ grę w zgadywanie liczby

1. Program pokazany na rysunku 5. realizuje algorytm gry w zgadywanie liczby w zbiorze liczb naturalnych z zakresu ⟨1, 100⟩. Przepisz ten program i zapisz go w pliku pod nazwą *Zgadnij_liczbę*.
2. Skompiluj i uruchom kilkakrotnie program, grając w grę w zgadywanie liczby z komputerem.
3. Objaśnij działanie programu, m.in. uzasadnij użycie poszczególnych instrukcji zgodnie z podaną listą kroków.

Ćwiczenie 8. Modyfikujemy program

1. W programie zapisanym w ćwiczeniu 7. zmień wprowadzanie zmiennej `szukana` tak, aby wartość tej zmiennej była wprowadzana z klawiatury.
2. Zapisz program w pliku pod nazwą *Zgadnij_liczbę_klawiatura*.
3. Skompiluj i uruchom kilkakrotnie program, grając w grę w zgadywanie liczby z drugą osobą.
4. Oceń grę z użyciem każdego z programów. Uzasadnij odpowiedź.

Algorytm wyszukiwania elementu w zbiorze uporządkowanym metodą połowienia

W algorytmie wyszukiwania elementu w zbiorze uporządkowanym szukamy danego elementu i równocześnie jego położenia w tym zbiorze.

Opis algorytmu: Dany jest uporządkowany w kolejności od najmniejszej do największej zbiór dziesięciu liczb (rys. 6.). Poszukiwana liczba to 40 (zaznaczona na szaro). Jeżeli w zbiorze jest kilka takich samych wartości, algorytm znajdzie pozycję jednego z tych elementów (nie jest określone którego).

Dzielimy zbiór liczb na dwie części, rozdzielone środkowym elementem. Zaczynamy od sprawdzenia tego elementu (liczby zaznaczonej na rys. 6. zieloną strzałką). W przypadku, gdy liczba elementów zbioru jest parzysta, spośród dwóch środkowych elementów (w naszym przypadku liczb 57 i 66) jako „środkowy" wybieramy ten, który jest pierwszy w kolejności (czyli 57).

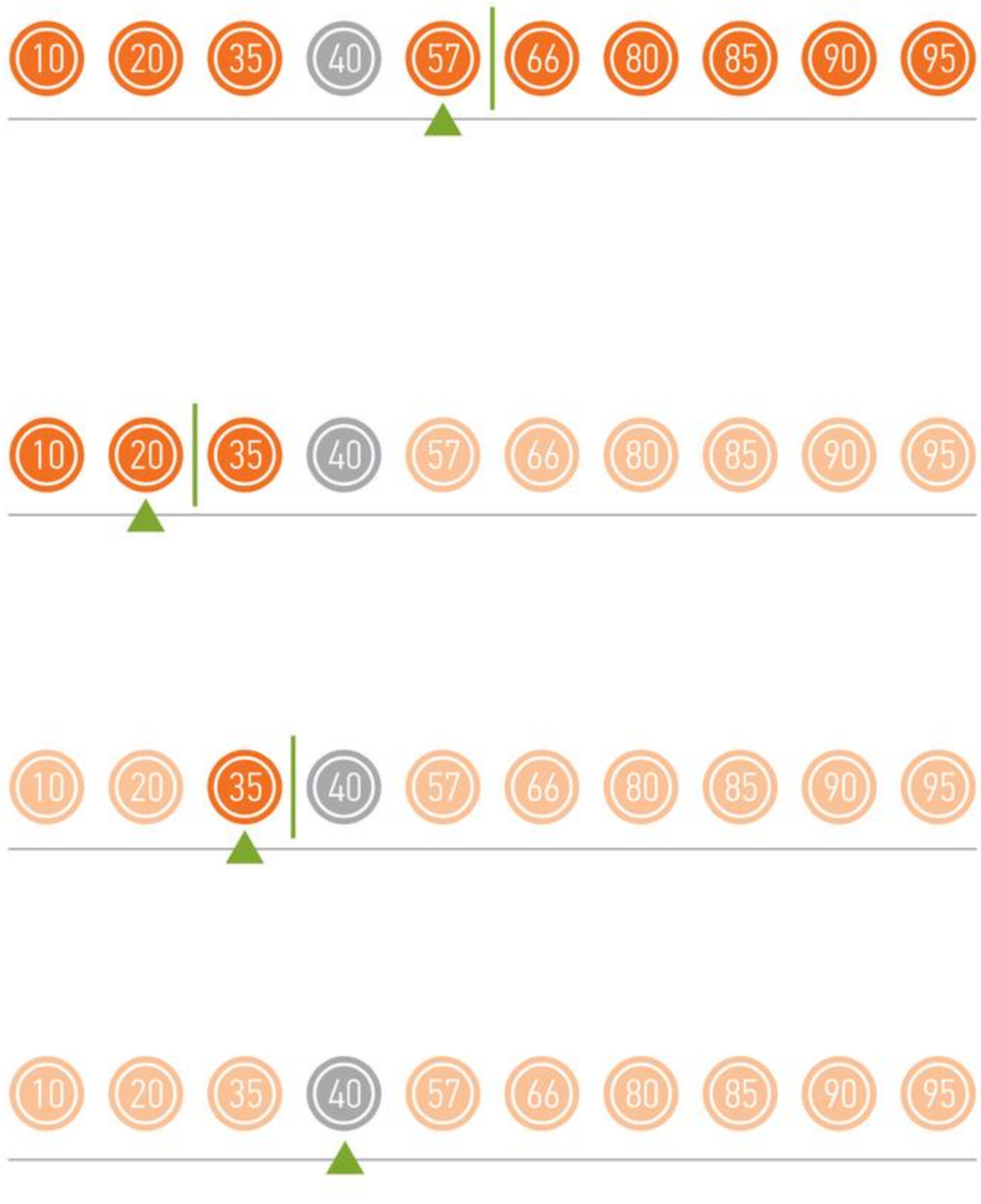
Jeśli środkowy element jest szukaną liczbą, kończymy wyszukiwanie. Jeśli środkowy element jest większy od poszukiwanej liczby, przeszukujemy podzbiór na lewo od elementu środkowego. Jeśli środkowy element jest mniejszy od poszukiwanej liczby, przeszukujemy podzbiór na prawo od elementu środkowego.

Szukamy liczby 40. „Środkowym" elementem zbioru jest liczba 57 i jest ona większa od 40. Dalej przeszukujemy tylko lewą część zbioru, a odrzucamy prawą oraz element środkowy 57.

Liczba elementów w przeszukiwanej części zbioru jest również parzysta (wynosi 4), więc jako „środkowy" wybieramy liczbę 20. Liczba 20 jest mniejsza od 40, więc odrzucamy lewą część zbioru razem z liczbą 20.

Pozostały dwie liczby. „Środkowym" elementem jest liczba 35. Jest ona mniejsza od 40, czyli ją odrzucamy.

Pozostał nam jeden element i jest to szukana liczba 40, która jest czwartą wartością w zbiorze.



Rys. 6. Przykład wyszukiwania liczby w zbiorze uporządkowanym metodą połowienia

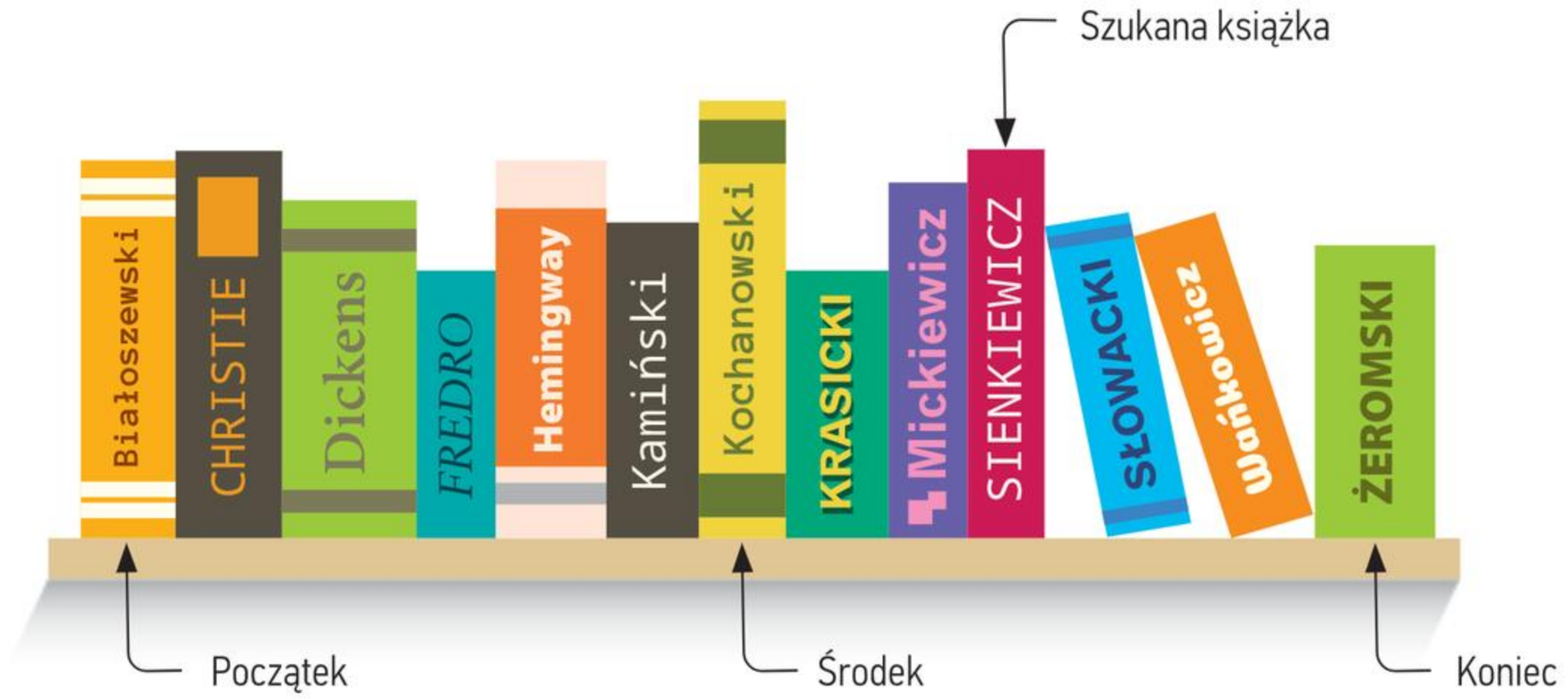
Ćwiczenie 9. Sprawdzamy działanie algorytmu wyszukiwania liczby w zbiorze uporządkowanym metodą połowienia

Przygotuj odpowiednie pomoce dydaktyczne (rys. 6.) i przedstaw wspólnie z koleżankami i kolegami z klasy algorytm wyszukiwania liczby w zbiorze uporządkowanym metodą połowienia.

Wskazówka: Możesz zapisać liczby na kartkach i układać je na stole lub podłodze.

Z wyszukiwaniem przez połowienie spotykamy się w życiu codziennym, np. gdy szukamy jakiejś rzeczy w zbiorze, w którym panuje ustalony wcześniej porządek.

Wyobraźmy sobie półkę z książkami, które ułożono alfabetycznie według nazwisk autorów, zaczynając od lewej strony (rys. 7.). Chcemy znaleźć książkę autora, którego nazwisko zaczyna się na literę „S". Czy przeglądamy półkę od samego początku? Raczej nie. Zwykle odruchowo postępujemy zgodnie z algorytmem wyszukiwania przez połowienie – sprawdzamy, czy szukanej książki nie ma pośrodku. Jeśli trafimy na inną książkę, np. autora na literę „K", to nie przeszukujemy lewej połowy półki, tylko będziemy szukać po prawej stronie.



Rys. 7. Zastosowanie algorytmu wyszukiwania przez połowienie – szukanie książki na półce

Zapisujemy w języku C++ algorytm wyszukiwania metodą połowienia

Algorytm wyszukiwania elementu w zbiorze uporządkowanym metodą połowienia zapiszemy w języku C++. Elementy zbioru wprowadzimy do tablicy. Zakładamy, że wprowadzamy dane od najmniejszej do największej (inaczej program nie będzie działał poprawnie). Bez takiego założenia należałoby dodać również funkcję sortowania danych (ustawiania w określonym porządku).

Program ma wyszukiwać w zbiorze uporządkowanym daną podaną przez użytkownika i podawać jej pozycję w tablicy (indeks). Przyjmujemy, że w zbiorze jest pięć elementów.

Zdefiniujemy funkcję `znajdz_dana()` z parametrem `wartosc` (rys. 8.):

- W funkcji wyznaczamy pozycję elementu środkowego (`srodek = poczatek + koniec / 2`).
- Jeśli element środkowy (`a[srodek]`) jest szukaną daną (zmienna `wartosc`), to funkcja zwraca wartość zmiennej `srodek`, czyli pozycję danego elementu; w przeciwnym wypadku:
  - jeśli szukana dana jest mniejsza od danej środkowej, odrzucamy prawą połowę zbioru (liczby większe od elementu środkowego) – ustalamy nowy koniec: `koniec = srodek - 1;`
  - w przeciwnym wypadku odrzucamy lewą połowę zbioru (liczby mniejsze od elementu środkowego) i ustalamy nowy początek: `poczatek = srodek + 1`.
- Połowienie powtarzamy, dopóki początek zbioru jest mniejszy od końca zbioru lub mu równy (`poczatek <= koniec`) – o ile wcześniej nie znajdziemy szukanego elementu (wtedy komputer wykonana instrukcję **`return`** `srodek`).
- Jeśli w przeszukiwanym zbiorze nie ma szukanej liczby, funkcja `znajdz_dana()` zwraca wartość –1.

[*] Wyszukiwanie_uporządkowany.cpp

```
#include <iostream>
using namespace std;

const int N = 5;
int a[N];

void wprowadz_dane()
{
    for(int i = 0; i < N; i++)
    {
        cout << "Podaj daną nr " << i << ": ";
        cin >> a[i];
    }
}

int znajdz_dana(int wartosc)
{
    int poczatek, koniec, srodek;
    poczatek = 0;
    koniec = N - 1;
    do
    {
        srodek = (poczatek + koniec) / 2;
        if(a[srodek] == wartosc)
            return srodek;
        else
            if(wartosc < a[srodek])
                koniec = srodek - 1;
            else
                poczatek = srodek + 1;
    }   while(poczatek <= koniec);

    return -1;
}

int main ()
{
    int wartosc, pozycja;
    wprowadz_dane();
    cout << "Podaj daną do wyszukania: ";
    cin >> wartosc;
    pozycja = znajdz_dana(wartosc);
    if(pozycja >= 0)
        cout << "Znaleziono wartość " << wartosc << " na pozycji "
        << pozycja << " (pozycje liczone są od 0)" << endl;
    else
        cout << "Nie znaleziono wartości " << wartosc << endl;

    return 0;
}
```

Wykorzystujemy utworzoną wcześniej funkcję `wprowadz_dane()` do wprowadzenia danych do tablicy.

Funkcja `znajdz_dana()`, która zwraca pozycję szukanego elementu w zbiorze (indeks w tablicy `a`).

Wywołujemy funkcję wprowadzania danych `wprowadz_dane()`.

Wywołujemy funkcję `znajdz_dana()`, przypisując wynik zmiennej pozycja.

Uwzględnimy również sytuację, gdy podanej wartości nie ma w zbiorze.

Rys. 8. Program w języku C++ realizujący algorytm wyszukiwania elementu w zbiorze uporządkowanym – ćwiczenie 10.

Ćwiczenie 10. Zapisujemy w języku C++ algorytm wyszukiwania elementu w zbiorze uporządkowanym

1. Program pokazany na rysunku 8. realizuje algorytm wyszukiwania elementu w zbiorze uporządkowanym. Przepisz ten program i zapisz go w pliku pod nazwą *Wyszukiwanie_uporządkowany*.
2. Skompiluj i uruchom program. Sprawdź działanie programu dla różnych danych.

Zapamiętaj

- Wybrany element w zbiorze nieuporządkowanym możemy znaleźć, korzystając z algorytmu przeszukiwania liniowego.
- Wybrany element w zbiorze uporządkowanym możemy znaleźć, korzystając z algorytmu poszukiwania przez połowienie, który jest przykładem realizacji metody „dziel i zwyciężaj”.
- Definiowanie funkcji i stosowanie tabel możemy wykorzystać do zapisania w języku C++ algorytmu wyszukiwania elementu w zbiorze nieuporządkowanym i algorytmu wyszukiwania elementu w zbiorze uporządkowanym.

Serwis dla ciekawskich

Angielski matematyk Alan Turing opracował w 1936 roku model teoretyczny maszyny do wykonywania algorytmów. Maszyna składała się z nieskończenie długiej taśmy, zawierającej komórki z przetwarzanymi symbolami, głowicy odczytująco-zapisującej i układu sterowania nią. Obliczenia wykonywane za pomocą tej maszyny zależały od układu symboli wpisanych na taśmie oraz od przyjętego zestawu instrukcji.

Podobnie działają dzisiejsze komputery – wyniki obliczeń zależą od danych zapisanych w pamięci komputera i od zestawu wykonanych instrukcji.

OKO W OKO Z MONITOREM

Pytania i polecenia

1. Wyjaśnij, w jaki sposób znajduje się największą spośród siedmiu liczb w zbiorze nieuporządkowanym.
2. Dlaczego w programie na rysunku 2. wprowadzaną wartość przypisujemy najpierw zmiennej `maks`?
3. Jaką liczbę najlepiej podać jako pierwszą w grze w zgadywanie liczby w przypadku zbioru liczb od 101 do 200? Uzasadnij odpowiedź.
4. Jakie może być zastosowanie algorytmu wyszukiwania przez połowienie? Pokaż na przykładzie.
5. Dlaczego program realizujący algorytm wyszukiwania elementu w zbiorze uporządkowanym zaczyna się od sprawdzenia, czy środkowy element jest szukaną liczbą?

Zadania

1. Na podstawie listy kroków pokazanej w punkcie 1. tego tematu opracuj listę kroków algorytmu znajdowania najmniejszej liczby spośród dziesięciu liczb. Sprawdź działanie algorytmu dla danych: 5, 78, 15, 0, 1, 100, 155, 50, 22, 30.
2. Zdefiniuj funkcję `minimum()` wyszukującą najmniejszy element ze zbioru `n`-elementowego. Wywołaj funkcję w programie głównym. Zapisz program w pliku pod nazwą *Minimum_n*.
3. Dany jest uporządkowany zbiór liczb: 5, 8, 15, 20, 25, 30, 35, 70, 88, 90, 100, 120. Poszukiwana liczba to 70. Przedstaw na rysunku (podobnym do rys. 6.) sposób znalezienia liczby 70.

DLA ZAINTERESOWANYCH

1. Opracuj listę kroków algorytmu poszukiwania przez połowienie. Skorzystaj z opisu metody podanego w punkcie 2. tego tematu.
2. Zdefiniuj funkcję `maksimum_tablica()` tak, aby znajdowała maksymalną wartość w tablicy liczb. Zapisz program w pliku pod nazwą *Liczba_maksymalna.*

3.5. Algorytmy porządkowania

Warto powtórzyć

1. Na czym polega algorytm wyszukiwania największego elementu w zbiorze nieuporządkowanym? Wyjaśnij na przykładzie pięciu elementów.
2. W jaki sposób definiuje się tablicę w języku C++?
3. Jakie instrukcje stosujemy do zapisywania iteracji w języku C++?

Czasem porządkujemy (w terminologii informatycznej – sortujemy) zbiory przedmiotów, np. ustawiamy książki na półce – od największej do najmniejszej lub alfabetycznie według nazwisk autorów, ustalamy kolejność wyświetlania plików według nazw lub dat utworzenia. Na danych uporządkowanych (posortowanych) szybciej wykonuje się różne operacje (np. wyszukiwanie). Istnieją różne algorytmy porządkowania. Zapiszemy je w języku programowania C++.

W tym temacie:

1. poznasz i zaprogramujesz algorytm porządkowania elementów zbioru metodą przez wybieranie,
2. poznasz i zaprogramujesz algorytm porządkowania elementów zbioru metodą przez zliczanie.

1 Porządkowanie elementów zbioru metodą przez wybieranie

Porządkowanie zbioru liczb metodą przez wybieranie polega na wyszukiwaniu w zbiorze liczby największej (dla porządku malejącego) lub najmniejszej (dla porządku rosnącego), zamienieniu jej miejscami z liczbą, która jest aktualnie na początku zbioru, a następnie powtarzaniu tych czynności z pominięciem już uporządkowanych liczb.

Aby wyszukać największą lub najmniejszą liczbę w zbiorze, stosujemy odpowiednio algorytm wyszukiwania największej lub najmniejszej liczby ze zbioru.

Rys. 1. Przykład porządkowania liczb metodą przez wybieranie – od największej do najmniejszej

Algorytm porządkowania metodą przez wybieranie wykorzystamy do ustawienia uczniów od najwyższego do najniższego.

Algorytm porządkowania elementów zbioru metodą przez wybieranie

Zadanie: Ustaw pięcioro uczniów (koleżanek i kolegów z klasy) według wzrostu (od ucznia najwyższego do najniższego), stosując sortowanie przez wybieranie.

Specyfikacja zadania:

Dane: Pięcioro uczniów ustawionych w przypadkowej kolejności.

Wynik: Pięcioro uczniów ustawionych w kolejności od najwyższego do najniższego.

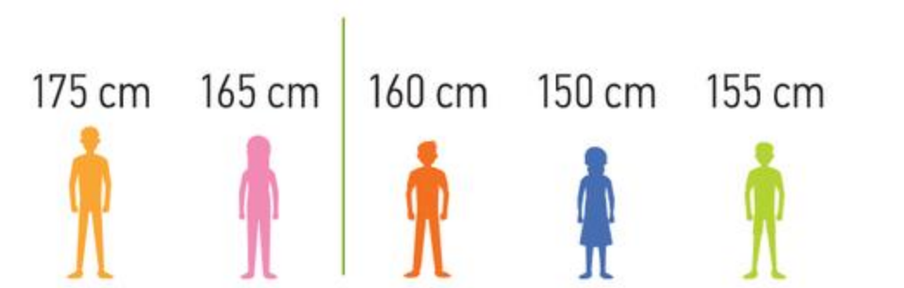
Lista kroków:

1. Zacznij algorytm.
2. Znajdź najwyższego ucznia. Zapamiętaj, na którym miejscu stoi.
3. Zamień miejscami ucznia najwyższego z uczniem stojącym na pierwszym miejscu.
4. Powtarzaj kroki 2. i 3. (z pominięciem uczniów już ustawionych na miejscach początkowych), aż ustawisz wszystkich uczniów.
5. Zakończ algorytm.

Znaleziono najwyższego ucznia wśród pięciu uczniów.

Najwyższy uczeń (175 cm) zamienił się miejscami ze stojącym na pierwszej pozycji (160 cm).

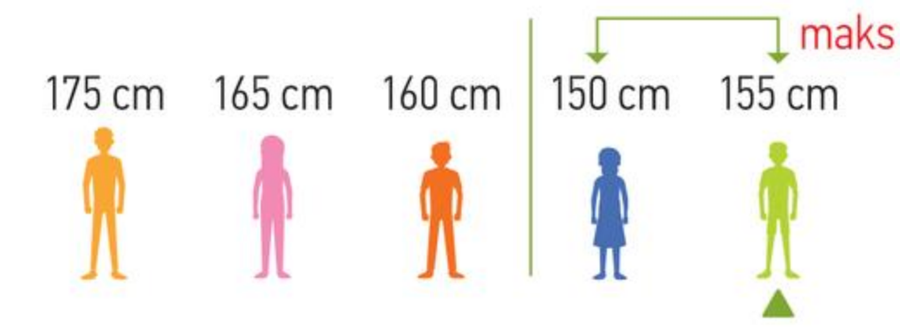
Znaleziono najwyższego ucznia (165 cm) wśród czterech pozostałych uczniów.



Najwyższy uczeń (165 cm) zamienił się miejscami ze stojącym na pierwszej pozycji (150 cm) wśród czterech pozostałych.

Znaleziono najwyższego ucznia (160 cm) wśród trzech pozostałych uczniów.

Najwyższy uczeń (160 cm) nie musi zmieniać miejsca, ponieważ stoi na właściwym.



Znaleziono najwyższego ucznia (155 cm) wśród dwóch pozostałych uczniów.

Najwyższy uczeń (155 cm) zamienił się miejscami ze stojącym na pierwszej pozycji (150 cm) wśród dwóch pozostałych.

Został ostatni uczeń (150 cm) , który stoi na właściwym miejscu. Wszyscy uczniowie stoją w kolejności od najwyższego do najniższego.

Rys. 2. Porządkowanie pięciorga uczniów według wzrostu metodą przez wybieranie – od najwyższego do najniższego

Ćwiczenie 1. Ustawiamy uczniów według wzrostu metodą przez wybieranie
Wybierz z klasy pięcioro uczniów. Ustaw wybranych uczniów według wzrostu (od najwyższego do najniższego). Postępuj zgodnie z podaną listą kroków.

Ćwiczenie 2. Porządkujemy książki metodą przez wybieranie
Opracuj listę kroków algorytmu porządkowania książek od najgrubszej do najcieńszej.

Zapisujemy algorytm porządkowania metodą przez wybieranie w języku C++

Zapiszemy w języku C++ algorytm porządkowania liczb metodą przez wybieranie od największej do najmniejszej. Przyjmujemy, że jest pięć liczb do uporządkowania.

W programie zdefiniujemy funkcje:

- `wprowadz_dane()` do wprowadzenia danych do tablicy,
- `wyprowadz_dane()` do wyświetlenia na ekranie uporządkowanych danych,
- `maks_wybor()` (z jednym parametrem `starti`) wyszukującą największy element (`maks`) w zbiorze, począwszy od elementu o indeksie `starti`, i zapamiętującą jego pozycję w zmiennej `maksi` (zwracaną na końcu funkcji),
- `sort_wybor()` (niezwracającą wartości i bez parametrów), w której wywołujemy w pętli funkcję `maks_wybor` z parametrem aktualnym `i` oraz zamieniamy miejscami znaleziony największy element `maks` z aktualnie pierwszym elementem zbioru. Po każdym przejściu pętli liczba nieuporządkowanych elementów zbioru jest o 1 mniejsza, ponieważ nie sprawdzamy już elementów ustawionych na właściwych miejscach (uporządkowanych).

W funkcji głównej `main()` wywołamy trzy funkcje: wprowadzania danych (`wprowadz_dane()`), porządkowania (`sort_wybor()`) i wyświetlania danych na ekranie (`wyprowadz_dane()`).

Ćwiczenie 3. Zapisujemy w języku C++ algorytm porządkowania metodą przez wybieranie

1. Program w języku C++ pokazany na rysunku 3. realizuje algorytm porządkowania zbioru liczb metodą przez wybieranie. Przepisz ten program i zapisz go w pliku pod nazwą *Wybieranie*.
2. Skompiluj i uruchom program. Sprawdź jego działanie dla różnych danych, w tym dla danych pokazanych na rysunkach 1. i 2. Objaśnij działanie programu.

[*] Wybieranie.cpp

```
#include <iostream>
using namespace std;

const int N = 5;
int a[N];

void wprowadz_dane()
{
    for(int i = 0; i < N; i++)
    {
        cout << "Podaj daną nr " << i << ": ";
        cin >> a[i];
    }
}

void wyprowadz_dane()
{
    for(int i = 0; i < N; i++)
        cout << "Dana nr " << i << ": " << a[i] << endl;
}

int maks_wybor(int starti)
{
    int maksi = starti;
    int maks = a[maksi];
    for(int i = starti + 1; i < N; i++)
        if(a[i] > maks)
        {
            maksi = i;
            maks = a[maksi];
        }
    return maksi;
}

void sort_wybor()
{
    for(int i = 0; i < N - 1; i++)
    {
        int maksi = maks_wybor(i);
        int t = a[i];
        a[i] = a[maksi];
        a[maksi] = t;
    }
}

int main()
{
    wprowadz_dane();
    sort_wybor();
    wyprowadz_dane();
    return 0;
}
```

Funkcja wprowadzająca dane do tablicy

Funkcja wyświetlająca dane na ekranie

Deklarujemy zmienną `maksi` jako liczbę całkowitą i równocześnie przypisujemy jej numer pierwszego elementu nieuporządkowanego zbioru.

W pętli szukamy kolejnych liczb większych od aktualnie największej i zapamiętujemy pozycję największej liczby w tablicy `maksi = i`. Polecenia w pętli powtarzamy `N - (starti + 1)` razy.

Do początkowej wartości zmiennej sterującej `starti` dodajemy 1, ponieważ wartość pierwszego elementu nieuporządkowanego zbioru już wprowadziliśmy do zmiennej `maks`.

Funkcja zwraca wartość największego elementu `maksi`.

Wywołujemy funkcję `maks_wybor` z parametrem aktualnym `i`, wynik przypisując zmiennej `maksi`, którą równocześnie deklarujemy jako **`int`**.

Zmienną pomocniczą `t` stosujemy do zamiany miejscami znalezionego największego elementu z aktualnie pierwszym.

Wywołujemy w programie głównym funkcje wprowadzania danych, porządkowania i wyświetlania danych na ekranie.

Rys. 3. Program w języku C++ realizujący algorytm porządkowania metodą przez wybieranie w języku C++ – ćwiczenie 3.

2 Porządkowanie elementów zbioru metodą przez zliczanie

Metoda porządkowania przez zliczanie polega na zliczaniu wystąpień poszczególnych wartości w zbiorze, np. ile mamy jedynek, dwójek, trójek, czwórek, piątek, …, dwudziestek – w przypadku zbioru liczb całkowitych zawierającego wartości od 1 do 20. Następnie wypisujemy daną wartość tyle razy, ile wynosi liczba jej wystąpień w zbiorze. W ten sposób otrzymujemy uporządkowany zbiór.

Metodę porządkowania przez zliczanie pokażemy na przykładzie porządkowania liczb całkowitych od najmniejszej do największej. Przed rozpoczęciem porządkowania musimy znać najmniejszą i największą wartość w zbiorze. Wartości tych szukamy, stosując algorytm wyszukiwania elementu najmniejszego i największego w zbiorze nieuporządkowanym.

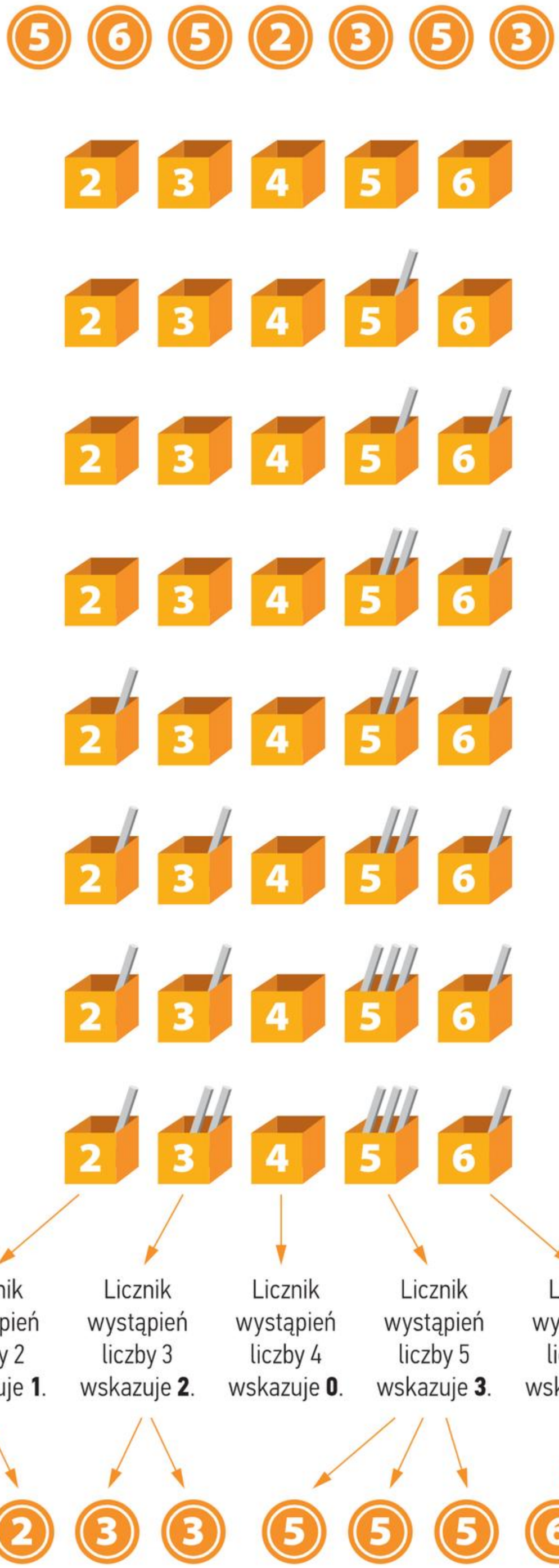
Algorytm porządkowania elementów zbioru metodą przez zliczanie (na przykładzie siedmiu liczb)

Opis algorytmu: Dany jest zbiór liczb {5, 6, 5, 2, 3, 5, 3}, który zawiera liczby całkowite z przedziału od 2 do 6 (czyli najmniejszą liczbą jest liczba 2, a największą – 6).

Do zliczania wystąpień wszystkich liczb z tego zakresu musimy przygotować pięć liczników do zliczania wystąpień liczb: 2, 3, 4, 5 i 6. Na rysunkach 4. i 5. liczniki mają formę pudełek. Na początku wszystkie pudełka liczników są puste (mają wartość 0), ale gdy zaczniemy przeglądać zbiór, wartości liczników będą się zmieniać.

Liczniki przygotowujemy dla wszystkich liczb z przedziału od 2 do 6, również dla tych, których nie ma w zbiorze.

Rys. 4. Pudełka pełniące rolę liczników



Zbiór siedmiu elementów do uporządkowania

Pięć wyzerowanych liczników (tu pustych pudełek), oznaczonych liczbami od 2 do 6.

Pierwszym elementem zbioru jest liczba 5, czyli licznik piątek zwiększamy o jeden.

Drugim elementem zbioru jest liczba 6, czyli licznik szóstek zwiększamy o jeden.

Trzecim elementem zbioru jest liczba 5, czyli licznik piątek zwiększamy o jeden.

Czwartym elementem zbioru jest liczba 2, czyli licznik dwójek zwiększamy o jeden

Piątym elementem zbioru jest liczba 3, czyli licznik trójek zwiększamy o jeden.

Szóstym elementem zbioru jest liczba 5, czyli licznik piątek zwiększamy o jeden.

Siódmym elementem zbioru jest liczba 3, czyli licznik trójek zwiększamy o jeden.

Otrzymujemy następujące wartości kolejnych liczników.

Wypisujemy daną liczbę tyle razy, ile wynosi wartość jej licznika i otrzymujemy zbiór liczb uporządkowany od najmniejszej do największej.

Rys. 5. Przykład porządkowania liczb metodą przez zliczanie

Ćwiczenie 4. Porządkujemy liczby od najmniejszej do największej metodą przez zliczanie

Przygotuj odpowiednie pomoce dydaktyczne (pudełka lub kubeczki i patyczki lub ołówki – rys. 5.), a następnie wykonaj wspólnie z koleżankami i kolegami z klasy algorytm porządkowania liczb metodą przez zliczanie dla zbioru składającego się z dziesięciu elementów: {–2, 5, 2, 0, 4, –2, 4, –1, –2, 5}. Liczby uporządkuj od najmniejszej do największej.

Wskazówki:

- Pudełek (kubeczków) ma być tyle, ile jest liczb całkowitych z zakresu ⟨*najmniejszy*, *największy*⟩, gdzie *najmniejszy* to najmniejsza liczba w zbiorze, a *największy* – największa.
- Zbiór zaczynamy przeglądać od pierwszego elementu i kolejno wrzucamy patyczki (ołówki) do kubeczków oznaczonych daną liczbą (jak pokazano na rysunku 5.).

Aby zbiór liczb był uporządkowany od największej do najmniejszej, wystarczy podczas wypisywania zbioru wartości liczników przejrzeć go w odwrotnej kolejności – z prawej strony do lewej.

Zapisujemy algorytm porządkowania liczb metodą przez zliczanie w języku C++

Zapiszemy w języku C++ algorytm porządkowania liczb metodą przez zliczanie od najmniejszej do największej. Przyjmujemy, że jest siedem liczb do uporządkowania.

W programie wykorzystamy funkcje:

- `wprowadz_dane()` do wprowadzenia danych do tablicy. Dodamy w funkcji sprawdzanie, czy liczba należy do określonego zakresu. Ustalimy zakres ⟨`0, MAX_DANA`⟩. `MAX_DANA` będzie stałą, której przypiszemy określoną wartość, np. 99. Dzięki temu będziemy mogli zadeklarować tablicę, która będzie przechowywała liczbę wystąpień każdej wartości z przedziału od 0 do 99,
- `wyprowadz_dane()` do wyświetlenia danych na ekranie po uporządkowaniu,
- `sort_zlicz()` (niezwracającą wartości i bez parametrów), w której utworzymy tablicę `liczniki[MAX_DANA + 1]`, służącą do zliczania wystąpień elementów zbioru. Liczba liczników, zgodnie z algorytmem, musi umożliwić zliczenie każdej liczby z danego zakresu, więc przygotowujemy 100 liczników (dla liczb od 0 do 99).

[*] Zliczanie.cpp

```cpp
#include <iostream>
using namespace std;

const int N = 7;
const int MAX_DANA = 99;
int a[N];

void wprowadz_dane()
{
    for(int i = 0; i < N; i++)
    {
        cout << "Podaj daną nr " << i << " (z przedziału <0," << MAX_DANA << ">): ";
        cin >> a[i];
        if(a[i] < 0)
            a[i] = 0;
        if(a[i] > MAX_DANA)
            a[i] = MAX_DANA;
    }
}

void wyprowadz_dane()
{
    for(int i = 0; i < N; i++)
        cout << "a[" << i << "] = " << a[i] << endl;
}

void sort_zlicz()
{
    int liczniki[MAX_DANA + 1] = {};
    int i, j;
    for(i = 0; i < N; i++)
        liczniki[a[i]]++;
    int poz = 0;
    for(i = 0; i <= MAX_DANA; i++)
        for(j = 0; j < liczniki[i]; j++)
            a[poz++] = i;
}

int main()
{
    wprowadz_dane();
    cout << endl;
    cout << "Wprowadzone dane:" << endl;
    wyprowadz_dane();
    cout << "Dane posortowane rosnąco przez zliczanie:" << endl;
    sort_zlicz();
    wyprowadz_dane();
    return 0;
}
```

Funkcja wprowadzająca dane do tablicy

Funkcja wyświetlająca dane na ekranie

Zerujemy tablicę `liczniki`.

Dla każdej wprowadzonej liczby zwiększamy odpowiadający jej licznik o 1.

Układamy elementy tablicy `a` od najmniejszego do największego.

Wywołujemy funkcje wprowadzania danych, porządkowania i wyświetlania danych na ekranie.

Rys. 6. Program w języku C++ realizujący algorytm porządkowania metodą przez zliczanie – ćwiczenie 5.

W pętli **`for`** (rys. 6., wiersze 31–32) dla każdej liczby wprowadzonej przez użytkownika zwiększamy odpowiadający jej licznik o 1.

Zapis `liczniki[a[i]]++` (rys. 6., wiersz 32.) odpowiada instrukcji przypisania: `liczniki[a[i]] = liczniki[a[i]] + 1` i służy do zwiększenia o 1 wartości odpowiedniego licznika.

W pętli **for** (rys. 6., wiersze 35–36) przeglądamy tablicę `liczniki` i wpisujemy do tablicy z danymi (`a`) daną wartość tyle razy, ile razy pojawiła się w zbiorze (liczbie wystąpień odpowiada element tablicy `liczniki`). Pętla z wierszy 35–36 jest wykonywana dla każdej możliwej wartości danych wejściowych (pętla **for** – wiersz 34.), mamy tu zatem do czynienia z zagnieżdżeniem pętli. Zmienna `poz` określa aktualną pozycję w tablicy `a`.

Zapis `a[poz++] = i;` (rys. 6., wiersz 36.) odpowiada sekwencji instrukcji:

```
a[poz] = i;
poz++;
```

W funkcji głównej `main()` wywołamy trzy funkcje: wprowadzania danych (`wprowadz_dane()`), porządkowania (`sort_zlicz()`) i wyświetlania danych na ekranie (`wyprowadz_dane()`).

Na rysunku 7. pokazano stan tablicy `liczniki` dla wprowadzonych liczb {1, 1, 2, 4, 5}.

liczniki[0]	liczniki[1]	liczniki[2]	liczniki[3]	liczniki[4]	liczniki[5]
0	2	1	0	1	1

Rys. 7. Wartości elementów tablicy `liczniki` dla liczb {1, 1, 2, 4, 5}

Ćwiczenie 5. Zapisujemy w języku C++ algorytm porządkowania metodą przez zliczanie

1. Program w języku C++ pokazany na rysunku 6. realizuje algorytm porządkowania metodą przez zliczanie. Przepisz ten program i zapisz go w pliku pod nazwą *Zliczanie*.
2. Skompiluj i uruchom program. Sprawdź jego działanie dla różnych danych, w tym dla danych pokazanych na rysunku 5. i podanych w ćwiczeniu 4.
3. Objaśnij działanie programu.

Zapamiętaj

- Zbiór elementów możemy uporządkować, stosując algorytmy porządkowania, np. metodą przez wybieranie lub metodą przez zliczanie.
- W porządkowaniu metodą przez wybieranie wykorzystujemy algorytm wyszukiwania elementu (najmniejszego lub największego) w zbiorze nieuporządkowanym.
- Definiowanie funkcji i stosowanie tablic możemy wykorzystać do zapisania w języku C++ algorytmów porządkowania metodą przez wybieranie oraz porządkowania metodą przez zliczanie.

OKO W OKO Z MONITOREM

Pytania i polecenia

1. Wskaż przynajmniej dwie różnice między porządkowaniem metodą przez wybieranie a metodą porządkowania przez zliczanie.
2. Dlaczego w programie realizującym algorytm porządkowania metodą przez wybieranie musimy zapamiętywać pozycję znalezionego największego elementu w oddzielnej zmiennej?
3. Dlaczego w porządkowaniu metodą przez zliczanie należy przygotować liczniki dla wszystkich liczb z danego zakresu, nawet jeśli niektóre liczniki pozostaną zerowe?
4. Po co potrzebna jest tablica `liczniki` w programie realizującym algorytm porządkowania metodą przez zliczanie?

Zadania

1. Napisz listę kroków sortowania metodą przez wybieranie dla *n* elementów.
2. Dany jest zbiór liczb: {12, 3, 8, 2, 12, 5, 8, 4, 10, 4}. Przedstaw na rysunku (podobnym do rys. 5.) uporządkowanie tego zbioru metodą przez zliczanie. Ile liczników musisz przygotować? Ile liczników pozostało pustych?
3. Zmodyfikuj program *Wybieranie* (zapisany w ćwiczeniu 3.), aby umożliwił wprowadzenie dziesięciu elementów. Zapisz program w pliku pod nazwą *Wybieranie_10*.

DLA ZAINTERESOWANYCH

1. Zmodyfikuj program *Wybieranie* (zapisany w ćwiczeniu 3.), aby porządkował wprowadzone dane od najmniejszej do największej. Zapisz program w pliku pod nazwą *Wybieranie_od_najmniejszej.*
2. Zmodyfikuj program *Zliczanie* (zapisany w ćwiczeniu 5.), aby porządkował wprowadzone dane od największej do najmniejszej. Zapisz program w pliku pod nazwą *Zliczanie_od_największej.*

Projekty

4.1. Dokumentacja szkolnej imprezy sportowej

Cel projektu

Waszym zadaniem będzie przygotowanie w zespole dokumentacji szkolnej imprezy sportowej, zawierającej samodzielnie zebrane i opracowane materiały.

Podczas pracy nad projektem wykorzystacie poznane możliwości arkusza kalkulacyjnego: utworzycie tabele, wprowadzicie dane, wykonacie na nich obliczenia oraz zaprezentujecie dane i wyniki na wykresach. Użyjecie również edytora tekstu do zaprojektowania plakatu z programem imprezy.

Zadania szczegółowe

Zadanie 1. Przygotowanie ogólnego programu szkolnej imprezy sportowej.
Zadanie 2. Przygotowanie szczegółowego harmonogramu i zasad rozgrywek.
Zadanie 3. Zaprojektowanie tabeli do zapisywania nazwisk uczniów biorących udział w poszczególnych zawodach.
Zadanie 4. Zaprojektowanie tabeli do zapisywania wyników w poszczególnych zawodach.
Zadanie 5. Przygotowanie zestawień wyników dla wszystkich klas w poszczególnych zawodach.
Zadanie 6. Wydrukowanie wszystkich dokumentów.

Wskazówki do zadań szczegółowych

1. Zastanówcie się wspólnie, jakie zawody (w jakich dyscyplinach sportowych) będzie można zorganizować w waszej szkole, np. zawody pływackie (jeśli jest dostępny basen), mecze piłki nożnej, ręcznej, siatkowej i koszykowej (jeśli są odpowiednie boiska), biegi i skoki (jeśli są odpowiednie bieżnie i skocznie), turniej w tenisa stołowego (gdy w szkole są stoły pingpongowe). Przy wyborze dyscyplin warto również uwzględnić porę roku. Zależnie od warunków wpiszcie do programu kilka dyscyplin lub tylko jedną. Ogólny program imprezy sportowej przygotujcie w edytorze tekstu, np. w formie plakatu.

Rys. 1. Przykładowy plakat wydarzenia

2. Ustalcie liczby zawodników startujących w każdej dyscyplinie, system rozgrywek (np. pucharowy, „każdy z każdym", „do dwóch przegranych"), punktację za start w danej dyscyplinie, punktację za zdobyte miejsca medalowe, nagrody za miejsca zajęte w poszczególnych zawodach i dla liderów w punktacji generalnej. Szczegółowy godzinowy harmonogram, uwzględniający zasady rozgrywek, przygotujcie w edytorze tekstu (np. w tabeli).
3. Tabele do zapisywania imion i nazwisk uczniów przygotujcie w arkuszu kalkulacyjnym. W projekcie użyjcie fikcyjnych danych uczniów. Uwzględnijcie podział na dziewczęta i chłopców. Utwórzcie formuły do obliczania procentowego udziału uczniów z poszczególnych klas (z podziałem na dziewczęta i chłopców) w danych zawodach oraz formuły obliczające wyniki i wskazujące nagrodzonych.
4. Tabele do zapisywania wyników w poszczególnych zawodach może wykonywać kilka grup – oddzielnie dla różnych dyscyplin. Ustalcie właściwy układ tabeli oraz rodzaj prowadzonych obliczeń. Na przykład w przypadku skoku w dal tabela powinna zawierać nazwiska i imiona zawodników, wyniki osiągnięte w trzech turach skoków, najlepszy wynik każdego ucznia. Na koniec można wyróżnić wiersze zawierające dane trzech najlepszych zawodników w danej dyscyplinie.
5. Po zakończeniu zawodów wpiszcie wyniki do przygotowanych tabel. Przedstawcie wyniki z poszczególnych dyscyplin w postaci czytelnych wykresów.
6. Wydrukujcie przygotowane dokumenty.

4.2. Sterowanie obiektem na ekranie

Cel projektu

Waszym zadaniem będzie zaprogramowanie gry, w której można sterować obiektem na ekranie za pomocą… ruchu ręki.

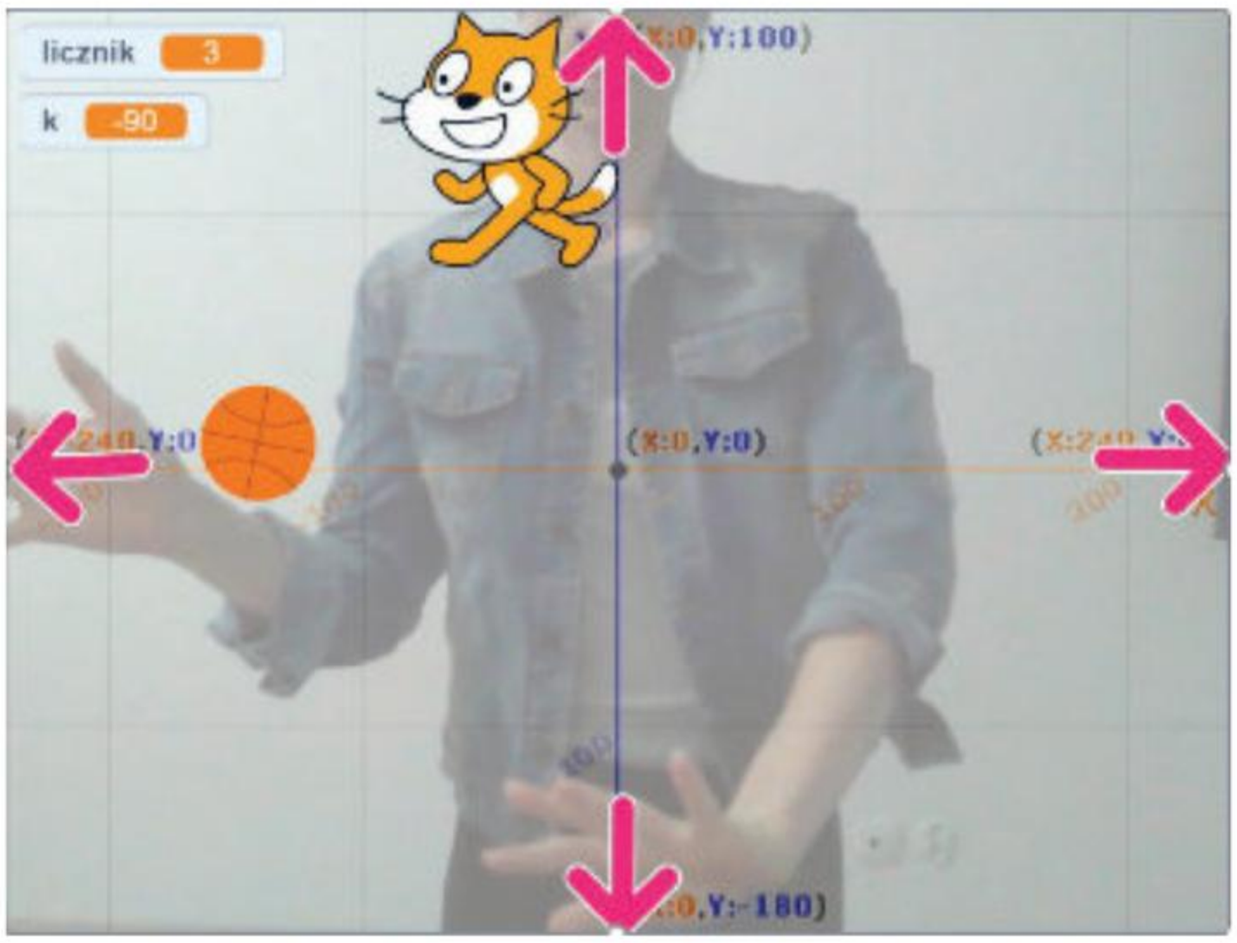
Podczas pracy nad projektem wykorzystacie dydaktyczne środowisko programowania Scratch 3. Zastosujecie poznane wcześniej możliwości języka Scratch (polecenie warunkowe, stosowanie zmiennych) i możliwości dodawania tzw. rozszerzeń. Skorzystacie z poleceń z grupy Czujniki wideo, które umożliwiają wykrywanie ruchu (np. ręki) za pomocą kamery.

Zadania szczegółowe

Zadanie 1. Przypomnienie zasad pracy w środowisku programowania Scratch.
Zadanie 2. Zapoznanie się z możliwościami poleceń z grupy **Czujniki wideo**.
Zadanie 3. Zaprojektowanie ogólnej koncepcji gry.
Zadanie 4. Opracowanie pierwszego poziomu gry.
Zadanie 5. Rozbudowanie gry.
Zadanie 6. Przeprowadzenie rozgrywek.
Zadanie 7. Wymyślenie nowej gry.

Wskazówki do zadań szczegółowych

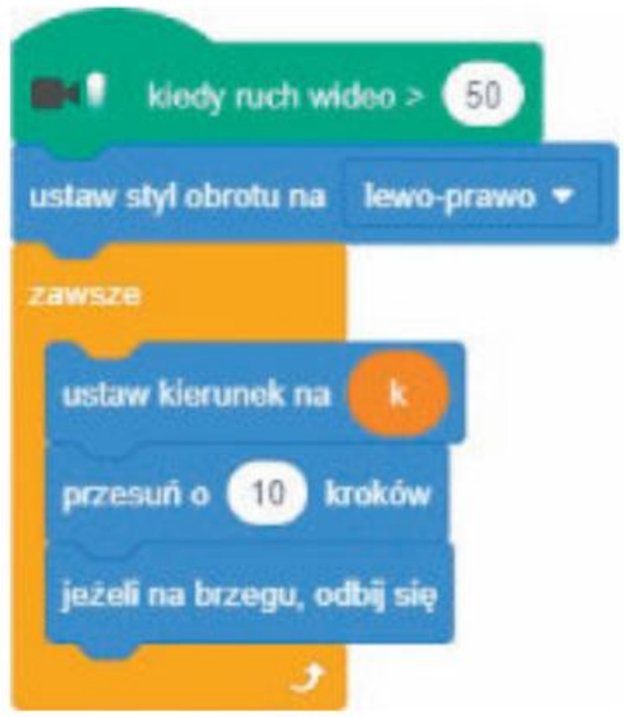
1. Przypomnijcie sobie zasady tworzenia, zapisywania i uruchamiania programu w języku Scratch, tworząc prosty program zawierający, m.in.: instrukcję warunkową i zmienne, polecenia do przesuwania duszka po scenie (o określoną liczbę kroków, losowe), zmianę kierunku ruchu duszka oraz dodawanie nowego duszka i tła.
2. W programie Scratch dodajcie grupę poleceń **Czujniki wideo**, klikając przycisk **Dodaj rozszerzenie** . Aby można było korzystać z tych poleceń, komputer powinien być wyposażony w kamerę. Utwórzcie krótki skrypt, który zaczyna się od bloku . W razie potrzeby możecie posłużyć się przykładami z Samouczków w programie Scratch. Aby duszek reagował na ruch ręki (czyli komputer wykonał skrypt utworzony dla tego duszka), należy zbliżyć rękę do ekranu tak, by ruch uchwyciła kamera. W polu tekstowym bloku możemy umieszczać wartości od 0 do 100 określające dynamikę ruchu wykrywanego przez kamerę – im większa wartość, tym ruch musi być szybszy, żeby został wykryty.
3. Uwzględnijcie możliwość sterowania obiektem na ekranie za pomocą ruchu ręki, np. zbliżenie ręki do obiektu na ekranie (duszka strzałki) spowoduje przemieszczenie się innego obiektu (np. duszka kota) w kierunku, który wyznacza strzałka



Rys. 1. Przykładowa scena do gry z wykorzystaniem poleceń **Czujniki wideo**

a.

b.



Rys. 2. Przykładowe skrypty dla duszka strzałki (rys. a) i duszka kota (rys. b)

(rys. 1.). Po scenie może poruszać się losowo np. duszek piłka. Dodajcie zliczanie punktów, np. jeśli duszek kot i duszek piłka dotkną się, gracz otrzymuje punkt. Opis działania gry zapiszcie w dokumencie tekstowym.

4. Utwórzcie program realizujący pierwszy poziom gry. Na rysunku 1. pokazano przykładową scenę do gry. Umieszczono na niej cztery duszki strzałki, które będą służyć do wykrywania ruchu w określonym punkcie ekranu. Na rysunku 2a pokazano przykładowy skrypt dla duszka strzałki. W skrypcie zastosowano zmienną *k*, która odpowiada kierunkowi ruchu podanemu w stopniach: dla ruchu w prawo 90, a w skryptach pozostałych duszków strzałek odpowiednio: –90 (ruch w lewo), 0 (ruch w górę) i 180 (ruch w dół). Zastosowanie zmiennej *k* w skrypcie dla duszka kota (rys. 2b) powoduje, że duszek będzie się poruszał w kierunku określonym przez zmienną *k*, której wartość jest ustalana na podstawie ruchu w kamerze (dla danego duszka strzałki). Dla duszka piłki utwórzcie skrypt umożliwiający losowe poruszanie się po całej scenie.
5. Rozbudujcie grę zgodnie z wcześniejszymi ustaleniami. Jeśli gra powinna mieć kilka poziomów, dodajcie je. Ustalcie dodatkowe możliwości zdobywania i utraty punktów na poszczególnych poziomach. Określcie również warunki przejścia na kolejne poziomy i warunek zakończenia gry.
6. Ustalcie zasady przeprowadzenia rozgrywek i opiszcie je w edytorze tekstu (np. w tabeli). Zagrajcie w grę. Powodzenia!
7. Wymyślcie kolejną grę, w której zastosujecie sterowanie obiektem na ekranie z wykorzystaniem kamery, ale bez użycia strzałek, np. odbijanie piłki.

Uwaga: Jeśli twój komputer nie jest wyposażony w kamerę, możesz sterować ruchami duszka na ekranie za pomocą klawiszy z klawiatury, używając odpowiednich poleceń języka Scratch.

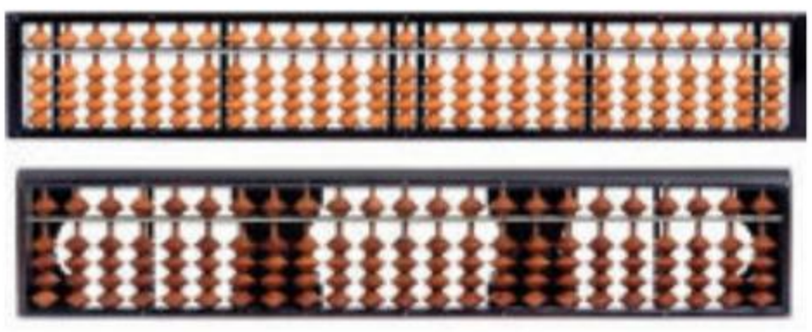
4.3. Historia i rozwój informatyki

X wiek p.n.e.

Już w X wieku przed naszą erą znano przyrządy do liczenia – abakusy, zbliżone wyglądem do liczydeł. Abakusy to gliniane tabliczki z wyżłobionymi rowkami, w których umieszczano przesuwane kamyki. Są one najstarszymi maszynami liczącymi.

XV wiek

Odmianą abakusa jest soroban, który stosowano od XV wieku w Japonii. Liczymy na nim, przesuwając koraliki w określony sposób – do lub od belki poprzecznej. Mimo powszechności komputerów, w Japonii nadal wykorzystuje się sorobany.

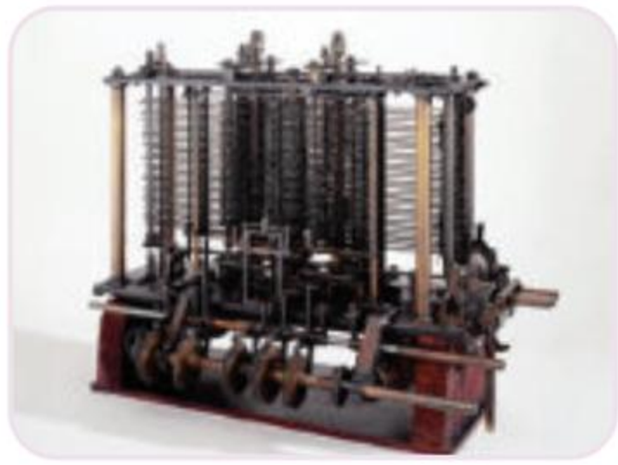


XVII wiek

Jedną z pierwszych na świecie maszyn wykonujących cztery działania (dodawanie, odejmowanie, mnożenie i dzielenie liczb całkowitych) wynalazł w 1623 roku niemiecki matematyk Wilhelm Schickard (czyt. szikard) – urządzenie wykonał z drewnianych elementów.

XVII wiek

W latach 1672–1694 powstaje mechanizm zastosowany w kalkulatorze opracowanym przez niemieckiego uczonego Wilhelma Leibniza (czyt. wilhelma lajbnica) – wykorzystywany do budowy różnych urządzeń aż do XX wieku.



1833 r.

Angielski matematyk Charles Babbage (czyt. czarls babydż) opracowuje model maszyny analitycznej napędzanej silnikiem lokomotywy parowej. Maszyna miała wykonywać dowolne ciągi instrukcji oraz operować na danych. Do wprowadzania i wyprowadzania danych służyły karty dziurkowane. Mimo że maszyna nigdy nie została wykonana w całości, jej model pomógł w opracowaniu koncepcji współczesnych komputerów.

1911 r.

Powstaje amerykańskie przedsiębiorstwo IBM, na początku znane pod nazwą CTR. Nazwę IBM przedsiębiorstwo przyjęło dopiero w 1924 roku. Początkowo firma nie zajmowała się produkcją komputerów, bo ich jeszcze nie wynaleziono. Przedsiębiorstwo wytwarzało różne produkty, m.in. systemy pomiaru czasu pracy, wagi, krajalnice do mięsa i karty perforowane. Poza wyprodukowaniem komputera osobistego IBM PC (1981) firma opracowywała różne technologie używane w informatyce, tj. nośniki pamięci (dysk twardy Winchester, dyskietkę), oprogramowanie (systemy operacyjne), a także... kursor.

1936 r.

Angielski matematyk Alan Turing opracowuje model teoretyczny maszyny do wykonywania algorytmów. Model ten składał się z nieskończenie długiej taśmy, zawierającej komórki z przetwarzanymi symbolami, głowicy odczytująco-zapisującej i układu sterowania.

1941 r.

Niemiecki inżynier Konrad Zuse (czyt. konrad cuze) konstruuje programowalną maszynę Z3, w której zastosował przekaźniki elektromagnetyczne. Z3 wykonywała operacje matematyczne (tj. dodawanie, odejmowanie, mnożenie, dzielenie i pierwiastkowanie) w systemie binarnym. Działaniem maszyny sterował program zapisany na taśmie perforowanej. Dane wprowadzano z klawiatury, a wynik odczytywano z wyświetlacza. Maszynę Z3 określa się czasem jako pierwszy komputer.

1948 r.

Powstaje tranzystor, którego masową produkcję rozpoczęto w 1954 roku. Pozwoliło to na znaczną miniaturyzację kolejnych powstających maszyn. Na przerzutnikach tranzystorowych budowano pierwsze polskie komputery z serii Odra.

Amerykański matematyk John von Neumann (czyt. dżon fon nojman) opracowuje (korzystając z modelu Charlesa Babbage'a) koncepcję logiczną i proponuje architekturę komputerów stosowaną do dziś.

Idea maszyny von Neumanna:

- programy i dane mogą mieć taką samą postać,
- programy i dane mogą być przechowywane w tej samej pamięci.

1944 r.

W Stanach Zjednoczonych powstaje komputer Harvard Mark I – największy w historii kalkulator elektromechaniczny o długości prawie 16 m i wysokości 2,5 m. Ważył ponad 5 ton i zawierał ponad 800 km przewodów z trzema milionami połączeń. Była to pierwsza maszyna, której pracą sterował wbudowany program.

1945 r.

Pierwszy komputer elektroniczny

W Stanach Zjednoczonych powstaje pierwszy w historii komputer elektroniczny o nazwie ENIAC. Dotychczas używane koła zębate, przekaźniki i silniki zastąpiono lampowymi układami elektronicznymi. Do budowy wykorzystano prawie 20 tysięcy lamp elektronowych, a do zasilania tego komputera zbudowano oddzielną elektrownię. Średni bezawaryjny czas pracy maszyny wynosił... 2 godziny, ponieważ po tym czasie któraś z lamp przepalała się i należało ją wymienić. ENIAC ważył ponad 27 ton i zajmował powierzchnię ok. 140 metrów kwadratowych. Komputer ten nie miał pamięci operacyjnej.

1954 r.

Powstaje FORTRAN – pierwszy język programowania wysokiego poziomu.

1968 r.

Powstaje firma Intel zajmująca się opracowywaniem i produkcją układów scalonych (integrujących w jedną funkcjonalną całość proste elementy logiczne: przerzutniki, rejestry i bramki logiczne) o różnych zastosowaniach. Opracowanie pierwszych układów scalonych to kolejny krok postępu technologicznego.

1974 r.

W firmie Intel szybko powstała nowsza, ośmiobitowa wersja procesora oznaczona symbolem 8008, a w roku 1974 – uniwersalny mikroprocesor 8080 – serce wielu przyszłych komputerów.

1975 r.

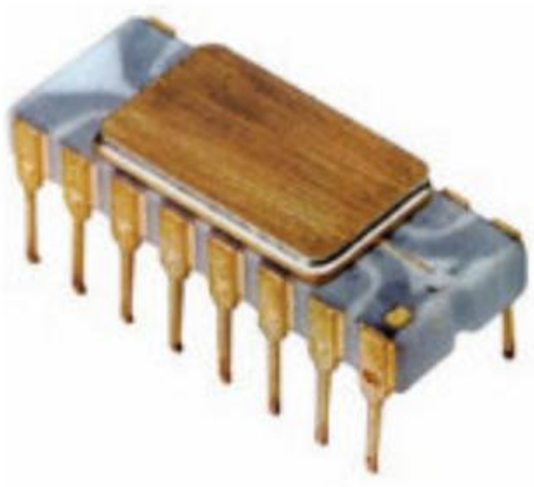
Pierwszy komputer osobisty

Firma MITS produkuje Altair 8800 – pierwszy mikrokomputer oparty na procesorze 8080. Początkowo był on dostępny w zestawach do samodzielnego montażu. Kosztował znacznie mniej niż wcześniejsze urządzenia – poniżej 500 dolarów. Komputer ten nie miał klawiatury. Programowano go za pomocą przełączników na płycie czołowej.

1971 r.

Pierwszy układ scalony

Firma Intel opracowuje pierwszy mikroprocesor Intel 4004. W 1971 roku pewna japońska firma zleciła Intelowi opracowanie i wyprodukowanie kilkunastu układów scalonych do produkowanych przez siebie kalkulatorów. Konstruktorzy zespołu, któremu zlecono projekt, zauważyli, że zamiast kilkunastu różnych układów można wykonać jeden, a wymagane funkcje realizować za pomocą krótkiego programu, umieszczonego w wewnętrznej pamięci typu ROM. Rozwiązanie to znacznie uprościło proces technologiczny i obniżyło koszty, a na dodatek przez modyfikację programu można było je przystosować do wielu różnych funkcji. Tak powstał pierwszy na świecie czterobitowy mikrokontroler 4004. Zawierał ok. 2300 tranzystorów.



1976 r.

Powstaje firma Apple. Pierwszym produktem firmy był komputer osobisty Apple I. Funkcję monitora pełnił domowy odbiornik telewizyjny.

1981 r.

Firma IBM przedstawia pierwszy komputer osobisty z serii **IBM PC**, który produkowała przez rok (potem zastąpiły go bardziej rozbudowane modele XT i AT). Komputer miał od 16 do 256 KB pamięci operacyjnej, monochromatyczny monitor oraz stację dyskietek 5,25 cala (nie miał jeszcze dysku twardego). System operacyjny DOS do tego komputera napisała firma Microsoft, której właścicielem był 25–letni Bill Gates.

1991 r.

Powstają pierwsze strony WWW.

1982–1985

Początki internetu – do powszechnego użytku wchodzi protokół TCP/IP.

1990 r.

Pojawia się MS Windows 3.0 (graficzna nakładka na DOS-a).

1981 r.

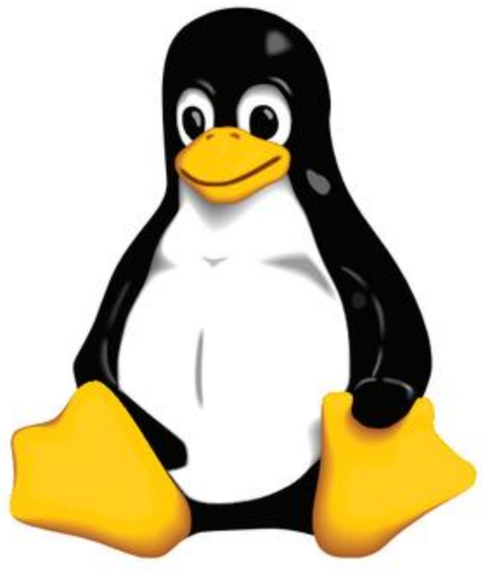
Firma SINCLAIR produkuje komputer ZX81 z 16 KB pamięci operacyjnej i zaimplementowanym interpreterem języka BASIC. Komputery ZX81 i następny model ZX Spectrum były bardzo popularne w Polsce.

1984 r.

Firma Apple zaczyna masowo sprzedawać pierwszy komputer z interfejsem graficznym – Macintosh i... odnosi komercyjny sukces. Od 2001 roku komputery Apple sprzedaje się z systemem operacyjnym macOS (początkowo Mac OS X).

1991 r.

Pojawia się pierwsza wersja systemu operacyjnego Linux. Wielozadaniowy, 32-bitowy system dzięki udostępnieniu kodu źródłowego w internecie szybko zyskał entuzjastów i tysiące programistów na całym świecie zaczęło bezinteresownie pracować nad jego rozwojem.



1992 r.

Pierwszy smartfon

Powstaje pierwszy smartfon – prototypowe urządzenie o nazwie Simon, które firma IBM wprowadza na rynek rok później. Poza funkcją telefonu komórkowego urządzenie ma kalendarz, książkę adresową, kalkulator, notatnik, pocztę elektroniczną i gry. Opcje wybiera się, dotykając ekranu rysikiem.

1999 r.

Pierwszy raz użyto terminu „Internet rzeczy". Określono nim „sieć połączonych ze sobą przedmiotów", np. urządzeń gospodarstwa domowego, artykułów oświetleniowych i grzewczych. Systemy Internetu rzeczy obejmują m.in.: inteligentne domy, inteligentne budynki, inteligentne miasta.

2007 r.

Firma Apple wprowadza na rynek smartfon iPhone, stanowiący przełom w rozwoju urządzeń mobilnych. Ma on dotykowy ekran z kolorowymi ikonami, które dotyka się palcem. Pozwala łączyć się z internetem za pomocą wi-fi, przeglądać strony internetowe, odtwarzać muzykę, oglądać filmy, robić zdjęcia. Transmisja danych oparta jest na sieci 2G o szybkości przesyłania danych 50 kb/s.

1994–2000

Następuje eksplozja usług internetowych. Pojawiają się e-sklepy, e-banki, e-szkoły, kawiarenki internetowe, komunikatory internetowe, telefonia VoiP, internet mobilny.

2004–2006

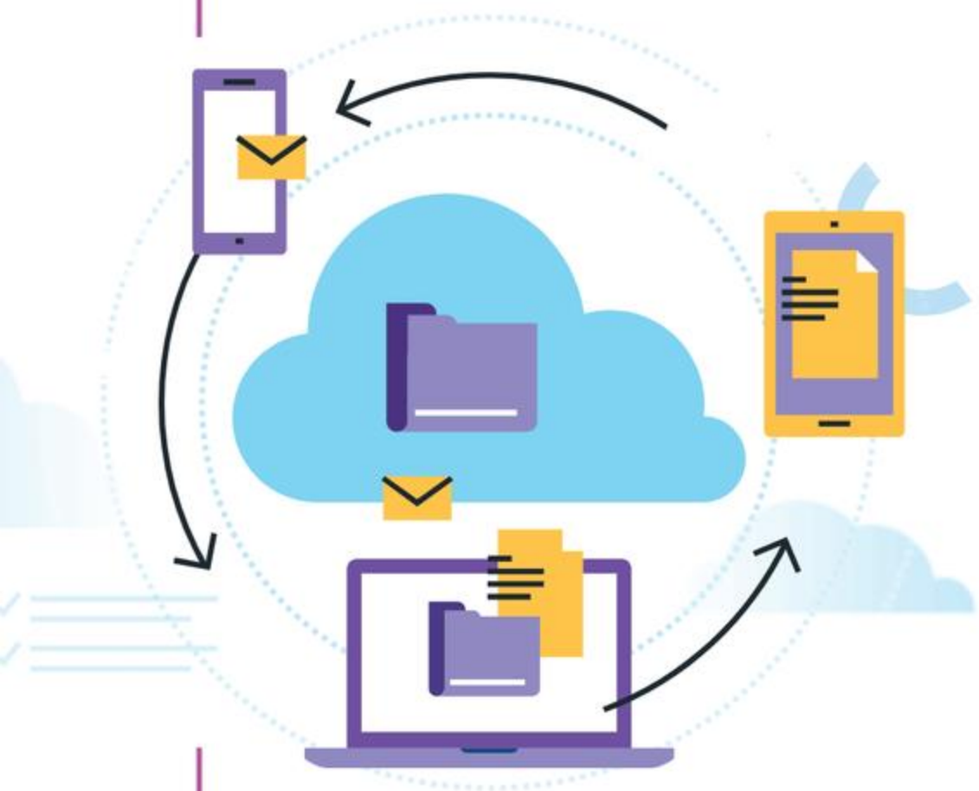
Powstają serwisy Web 2.0, w których użytkownicy współtworzą treści. Powstają: komunikator internetowy Skype (2004) oraz serwisy społecznościowe Facebook (2004) i Twitter (2006). Powstaje serwis internetowy YouTube (2005), który umożliwia bezpłatne umieszczanie, ocenianie i komentowanie filmów.

2006 r.

Adrian Bowyer (czyt. adrian bołjer) buduje pierwszy prototyp drukarki 3D dla użytkowników domowych. Samą technikę przestrzennego druku opracował Chuck Hull (czyt. czak hal) w 1984 roku XX wieku, a dwa lata później produkowano już pierwsze drukarki 3D, które od początku XXI wieku zdobywają coraz większą popularność. Technologia druku 3D znajduje również zastosowanie w medycynie, np. do tworzenia protez, implantów.

2010 r.

Możliwe stało się wykonywanie operacji na danych za pomocą przeglądarki internetowej – w chmurze obliczeniowej. Wcześniej pojawiły się dyski sieciowe, m.in.: Dropbox, Microsoft OneDrive, dysk Google.



2018 r.

W stanie Kalifornia (USA) prawo pozwala na to, by samochody mogły zabierać pasażerów na pokład. Samochodem bez udziału kierowcy steruje komputer. W samochodach tych zastosowano technologie umożliwiające omijanie przeszkód, tj.: radar, GPS, widzenie komputerowe.

2019–2020

Pojawia się technologia mobilna piątej generacji – standard sieci komórkowej 5G. Prędkość transmisji danych wzrasta do 20 Gb/s. Jest to ogromny skok w porównaniu z siecią 2G. W Polsce sieć 5G jest dostępna od 2020 r.

2010 r.

Pojawia się Instagram – fotograficzny serwis społecznościowy umożliwiający użytkownikom edycję zdjęć i filmów, stosowanie do nich filtrów cyfrowych oraz udostępnianie ich w różnych serwisach społecznościowych.

2019–2020

Pojawia się nowy trend w projektowaniu smartfonów – składany ekran, który pozwala na zastosowanie dużego ekranu w małej obudowie.

Minęło wiele lat od chwili, kiedy von Neumann przedstawił koncepcję, według której do dziś buduje się komputery. Zmieniły się technologie, ale komputery w istocie pozostały jedynie liczydłem, tyle że niezwykle rozbudowanym i bardzo szybkim, a inteligencja komputerów ciągle jest domeną literatury SF.

Cel projektu

Waszym zadaniem będzie przedstawienie (m.in. w postaci prezentacji multimedialnej) historii i rozwoju informatyki.

Podczas pracy nad projektem wykorzystacie wybrane treści podane na poprzednich stronach podręcznika. Do przygotowania projektu użyjecie poznanych we wcześniejszych klasach programów komputerowych, np. edytora tekstu, edytora grafiki, programu do tworzenia prezentacji multimedialnej.

Jeśli będziecie wykorzystywać zasoby pobrane z internetu, pamiętajcie o przestrzeganiu zasad korzystania z cudzych materiałów. Bez ograniczeń możecie korzystać z grafiki i innych utworów w tzw. domenie publicznej.

Zadania szczegółowe

Zadanie 1. Abakus i soroban a liczydło.
Zadanie 2. Maszyny liczące Schickarda i Leibniza a współczesne kalkulatory.
Zadanie 3. Pierwsze programowalne maszyny obliczeniowe.
Zadanie 4. Komputery Mark I i ENIAC.
Zadanie 5. Komputery oparte na tranzystorach półprzewodnikowych.
Zadanie 6. Początek internetu i rozwój usług internetowych.
Zadanie 7. Rozwój informatyki (sprzętu, technologii) w XXI wieku.

Wskazówki do zadań szczegółowych

1. Odszukajcie w internecie więcej informacji na temat abakusa i sorobanu oraz wykonywania obliczeń za ich pomocą. Jeśli macie dostęp do liczydła, przynieście je na lekcję i pokażcie, jak się go używa. Zastanówcie się, dlaczego wciąż używa się sorobanów, skoro są dostępne szybko liczące komputery. Porównajcie sposób wykonywania obliczeń na liczydle z wykonywaniem obliczeń na abakusie i sorobanie. Wskażcie podobieństwa i różnice pomiędzy sorobanem a komputerem. Przygotujcie prezentację multimedialną na ten temat (pięć do ośmiu slajdów).
2. Znajdźcie w internecie więcej informacji (zdjęcia, opis działania) na temat pierwszych kalkulatorów mechanicznych, w tym kalkulatorów znanych pod nazwą „kręciołki”. Te ostatnie wynaleziono w 1930 r. i używano w biurach jeszcze pod koniec lat 60. XX wieku. Wyjaśnijcie, dlaczego nazywano je „kręciołkami”. Porównajcie działanie dawnych i współczesnych kalkulatorów (możecie przynieść na lekcję przykładowy kalkulator). Przygotujcie prezentację multimedialną na ten temat (około dziesięciu slajdów).
3. Porównajcie osiągnięcia Charlesa Babbage’a i Alana Turinga w zakresie konstruowania maszyn obliczeniowych. Dodatkowo uzasadnijcie słuszność

nazywania Charlesa Babbage'a „ojcem komputerów". Dowiedzcie się więcej na temat maszyn, których nazwy zaczynają się na literę Z, a ich konstruktorem był Konrad Zuse. Wyjaśnijcie, czym są przekaźniki elektromagnetyczne. Skorzystajcie z internetu i innych źródeł, aby wyszukać dodatkowe informacje. Przygotujcie krótką prezentację multimedialną obejmującą te tematy, dobierając odpowiednią liczbę slajdów.

4. Korzystając z internetu i innych źródeł, dowiedzcie się więcej na temat komputerów Mark I i ENIAC. Porównajcie komputer Mark I z komputerem ENIAC. Uwzględnijcie w porównaniu wielkość, budowę, ogólny sposób działania. Przedstawcie wyniki porównania w tabeli. Sprawdźcie wymiary i wagę komputera, którego używacie, i dołączcie wyniki do tabeli. Następnie obliczcie, ile razy są mniejsze od wymiarów i wagi pierwszych komputerów. Pod tabelą umieśćcie wnioski z porównania komputerów (trzy zdania). We wnioskach uwzględnijcie wymiary, wagę i ogólny sposób działania. Do wykonania zadania użyjcie edytora tekstu i arkusza kalkulacyjnego.
5. Przedstawcie w postaci prezentacji multimedialnej rozwój informatyki od tranzystora półprzewodnikowego i układu scalonego do pojawienia się pierwszego komputera z serii IBM PC i pierwszego komputera z interfejsem graficznym – Macintosha. Wyjaśnijcie, na czym polega idea maszyny von Neumanna. Porównajcie osiągnięcia z lat 70. i 80., uwzględniając m.in. pojawienie się mikrokomputerów (np. ZX Spectrum, Commodore C64, Atari 800). Dodajcie na końcu prezentacji wnioski. Znajdźcie w internecie więcej informacji na ten temat.
6. Przedstawcie w wybranej postaci (np. w formie referatu, osi czasu, prezentacji multimedialnej) chronologicznie uporządkowane najważniejsze wydarzenia z rozwoju internetu. Uwzględnijcie daty powstania: sieci ARPANET (1969), poczty elektronicznej (1972), protokołu TCP/IP (1983), serwisu WWW (1991). Znajdźcie w internecie więcej informacji na ten temat. Uwzględnijcie również współczesną powszechność dostępu do usług internetowych i podajcie przykłady.
7. Znajdźcie w internecie (i innych źródłach) informacje na temat nowych pomysłów na komputer. Opiszcie w edytorze tekstu (na jednej stronie), na czym polegają te pomysły. Jeśli znajdziecie w internecie zdjęcia, dołączcie je do dokumentu. Znajdźcie w internecie definicję sztucznej inteligencji i dowiedzcie się więcej na jej temat. Przedstawcie również informacje o coraz powszechniej stosowanych robotach i sterujących nimi komputerach. Opracujcie to zagadnienie w wybranej formie (np. referatu, prezentacji multimedialnej).

4.4. Informatyka w moim przyszłym życiu

Cel projektu

Waszym zadaniem będzie przeprowadzenie dyskusji na temat wyboru przyszłego zawodu, w tym wymagającego kompetencji informatycznych.

Posługiwanie się komputerem i jego oprogramowaniem, w tym umiejętne korzystanie z internetu, jest kompetencją niezbędną w wielu zawodach. Podczas pracy nad projektem spróbujecie uzyskać odpowiedź na pytanie, co należy zrobić (m.in. jakie zdobyć wykształcenie), aby wykonywać w przyszłości wybrany zawód (np. grafika komputerowego, programisty, elektronika, architekta).

Wykorzystacie poznane możliwości edytora tekstu, arkusza kalkulacyjnego lub programu do tworzenia prezentacji multimedialnych. Będziecie również wyszukiwać informacje w internecie i zdobywać je z innych źródeł (np. przeprowadzając wywiady, czytając informatory edukacyjne).

Zadania szczegółowe

Zadanie 1. Dyskusja nad wyborem przyszłego zawodu.
Zadanie 2. Zaprojektowanie tabeli (tabel) do zapisywania informacji o zawodach.
Zadanie 3. Zebranie informacji dotyczących wybranych zawodów.
Zadanie 4. Umieszczenie informacji w tabelach.
Zadanie 5. Połączenie wszystkich dokumentów w jeden dokument.
Zadanie 6. Wydrukowanie dokumentu.

Wskazówki do zadań szczegółowych

1. Zastanówcie się wspólnie, prowadząc dyskusję (np. w formie „burzy mózgów"), kim chcielibyście być w przyszłości, jakie wykonywać zawody. Czy będą to zawody związane z informatyką, a może tylko wykorzystujące narzędzia informatyczne i wymagające kompetencji informatycznych? Jeśli nie macie jeszcze dokładnie sprecyzowanych planów, możecie posłużyć się przykładami podanymi we wprowadzeniu do projektu. Podczas dyskusji zapisujcie na bieżąco swoje pomysły. W wyniku dyskusji powinna powstać lista zawodów, nad którymi będziecie pracować w projekcie. Zapiszcie tę listę w dokumencie tekstowym.

2. Ustalcie, jakie informacje o zawodach powinniście zebrać, aby zaplanować dalsze kształcenie po szkole podstawowej. Na tej podstawie zaprojektujcie tabelę do opisywania poszczególnych zawodów. Ustalcie jej właściwy układ, formatowanie, tytuły nagłówków i kolumn, np. *Wymagane kompetencje (z podziałem na informatyczne i inne)*, *Wykształcenie*, *Wynagrodzenie* itp. Projekt tabeli utwórzcie w edytorze tekstu i zapiszcie w pliku, który przekażecie wszystkim grupom opracowującym poszczególne zawody. Plik możecie przesłać z wykorzystaniem poczty elektronicznej lub udostępnić w chmurze.
3. Ustalcie, jakie kompetencje (informatyczne i inne) są potrzebne w każdym wybranym zawodzie. Dowiedzcie się, jakie wykształcenie jest wymagane (wyższe, średnie). Jeśli konieczne są studia wyższe (np. informatyczne), ustalcie, jakie uczelnie znajdują się w waszej okolicy (np. mieście wojewódzkim) i na których z nich można studiować informatykę. Po wybraniu uczelni dowiedzcie się, jakie są wymagania, aby się na nie dostać, np. mogą być wymagane punkty z matury z matematyki, fizyki lub informatyki. Na podstawie zebranych informacji możecie wybrać szkołę ponadpodstawową (liceum, technikum). Poszukajcie w swojej okolicy szkół ponadpodstawowych i dowiedzcie się, jakie przedmioty są w nich realizowane w zakresie rozszerzonym (czy jest to np. matematyka, fizyka lub informatyka). Źródłem informacji może być internet, informatory edukacyjne, rozmowy w rodzinie czy wśród znajomych.
4. Po zakończeniu zbierania materiałów umieśćcie zdobyte informacje we wcześniej przygotowanych tabelach.
5. Połączcie wszystkie dokumenty w jeden dokument tekstowy. Dokument powinien składać się z tylu tabel (jednakowo sformatowanych), ile było opisywanych zawodów. Pliki możecie przesłać do siebie nawzajem z wykorzystaniem poczty elektronicznej lub udostępnić w chmurze.
6. Wydrukujcie przygotowany dokument.

Macie gotowy plan na przyszłość. Należy go tylko skutecznie zrealizować.

Indeks najważniejszych terminów

A

algorytm 54, 132
adres bezwzględny 25
adres komórki 7
adres mieszany 26
adres względny 12

F

formuła 9

I

instrukcja przypisania 71, 147
instrukcja warunkowa 76, 152
instrukcja iteracyjna 80, 155
interpretacja 59, 137
iteracja 78, 153

J

język programowania 58, 136

K

kompilacja 59, 137
komórka pamięci 69, 146

L

lista 85
lista kroków algorytmu 56, 134

N

NWD 95, 170

O

opis słowny algorytmu 55, 133

P

parametr formalny (parametr) 82, 157
parametr aktualny (argument) 82, 157
pasek formuły 7
pętla 78, 153
podprogram 82, 157
program komputerowy (kod źródłowy) 58, 136
program komputerowy (kod wynikowy) 59, 137

S

schemat blokowy algorytmu 56, 134
seria danych 31
specyfikacja zadania 55, 133

T

tablica 160
translacja 59, 137

Z

zmienna 69, 146

	Imię i nazwisko	Klasa	Rok szkolny	Podpis
1.	.....	.....	.....	.....
2.	.....	.....	.....	.....
3.	.....	.....	.....	.....

Lubię to!

nowa era | neon

Język C++

		Opis	Operator	Przykład	Wynik
Operatory arytmetyczne		Dodawanie	+	5 + 2	7
		Odejmowanie	-	8 - 5	3
		Mnożenie	*	3 * 4	12
		Dzielenie całkowite	/	10 / 3	3
		Dzielenie zmiennoprzecinkowe	/	10.0 / 3	3.3333
		Modulo (reszta z dzielenia)	%	12 % 5	2
		Opis	**Operator**	**Przykład**	
Operatory porównania		Równy	==	a == 5	
		Różny	!=	a != 5	
		Większy	>	a > 5	
		Większy lub równy	>=	a >= 5	
		Mniejszy	<	a < 5	
		Mniejszy lub równy	<=	a <= 5	
Operatory logiczne		Alternatywa logiczna (lub)	||	a < 5 || a > 10	
		Koniunkcja logiczna (i)	&&	a > 5 && a < 10	
		Negacja logiczna	!	! (a == 5)	

Język Python

		Opis	Operator	Przykład	Wynik
Operatory arytmetyczne		Dodawanie	+	5 + 2	7
		Odejmowanie	-	8 - 5	3
		Mnożenie	*	3 * 4	12
		Dzielenie całkowite	//	10 // 3	3
		Dzielenie zmiennoprzecinkowe	/	10.0 / 3	3.3333
		Modulo (reszta z dzielenia)	%	12 % 5	2
		Opis	**Operator**	**Przykład**	
Operatory porównania		Równy	==	a == 5	
		Różny	!=	a != 5	
		Większy	>	a > 5	
		Większy lub równy	>=	a >= 5	
		Mniejszy	<	a < 5	
		Mniejszy lub równy	<=	a <= 5	
Operatory logiczne		Alternatywa logiczna (lub)	or	a < 5 or a > 10	
		Koniunkcja logiczna (i)	and	a > 5 and a < 10	
		Negacja logiczna	not	not a == 5	

Nowa Era Sp. z o.o.

www.nowaera.pl nowaera@nowaera.pl

Centrum Kontaktu: 58 721 48 00

ISBN 978-83-267-4965-0