Temat 9. Wykorzystanie funkcji i list do zapisywania w języku Python algorytmów porządkowania i wyszukiwania

1. **Stosowanie procedur do utworzenia kompozycji w języku Scratch**
2. **Definiowanie i stosowanie funkcji w języku Python**
3. **Stosowanie list do wprowadzania danych**
4. **Zapisywanie w języku Python wybranych algorytmów porządkowania i wyszukiwania**

Warto powtórzyć

1. Do czego służą podprogramy (procedury i funkcje)?
2. Jak możemy nadać wartość zmiennej w języku Python?
3. W jaki sposób w języku Python wyprowadzamy napisy na ekran monitora?
4. Za pomocą jakiego polecenia realizujemy sytuację warunkową w języku Python?
5. W jaki sposób w języku Python zapisujemy iterację?

1. Stosowanie procedur do utworzenia kompozycji w języku Scratch

W językach Scratch i Python definiowaliśmy własne podprogramy (procedury i funkcje) rysujące figury geometryczne (np. kwadrat, sześciokąt). Podprogramy te wykorzystywaliśmy następnie do tworzenia kompozycji graficznych. W środowisku Baltie definiowaliśmy własne procedury, które wykorzystywaliśmy do tworzenia ciekawych scen, np. budowania domów czy… robotów.

Jak już wiemy, podprogramy (procedury i funkcje) służą do opracowywania oddzielnie problemów cząstkowych, które są częściami zadania rozwiązywanego przez dany program. Aby zastosować podprogram, czyli procedurę lub funkcję, należy ją **zdefiniować**, a następnie **wywołać**.

W programie pokazanym na rysunku 1a problemem cząstkowym jest narysowanie kwadratowego okna o krawędzi dowolnej długości. Problem zapisujemy w postaci procedury *Okno* z parametrem formalnym *krawędź*, określającym długość krawędzi ramy okiennej. Następnie w programie głównym trzykrotnie wywołujemy procedurę *Okno* z parametrem aktualnym *k*.

W chwili wywołania procedury parametr formalny *krawędź* zostanie zastąpiony parametrem aktualnym *k* (wartość zmiennej *k* wprowadzamy z klawiatury).

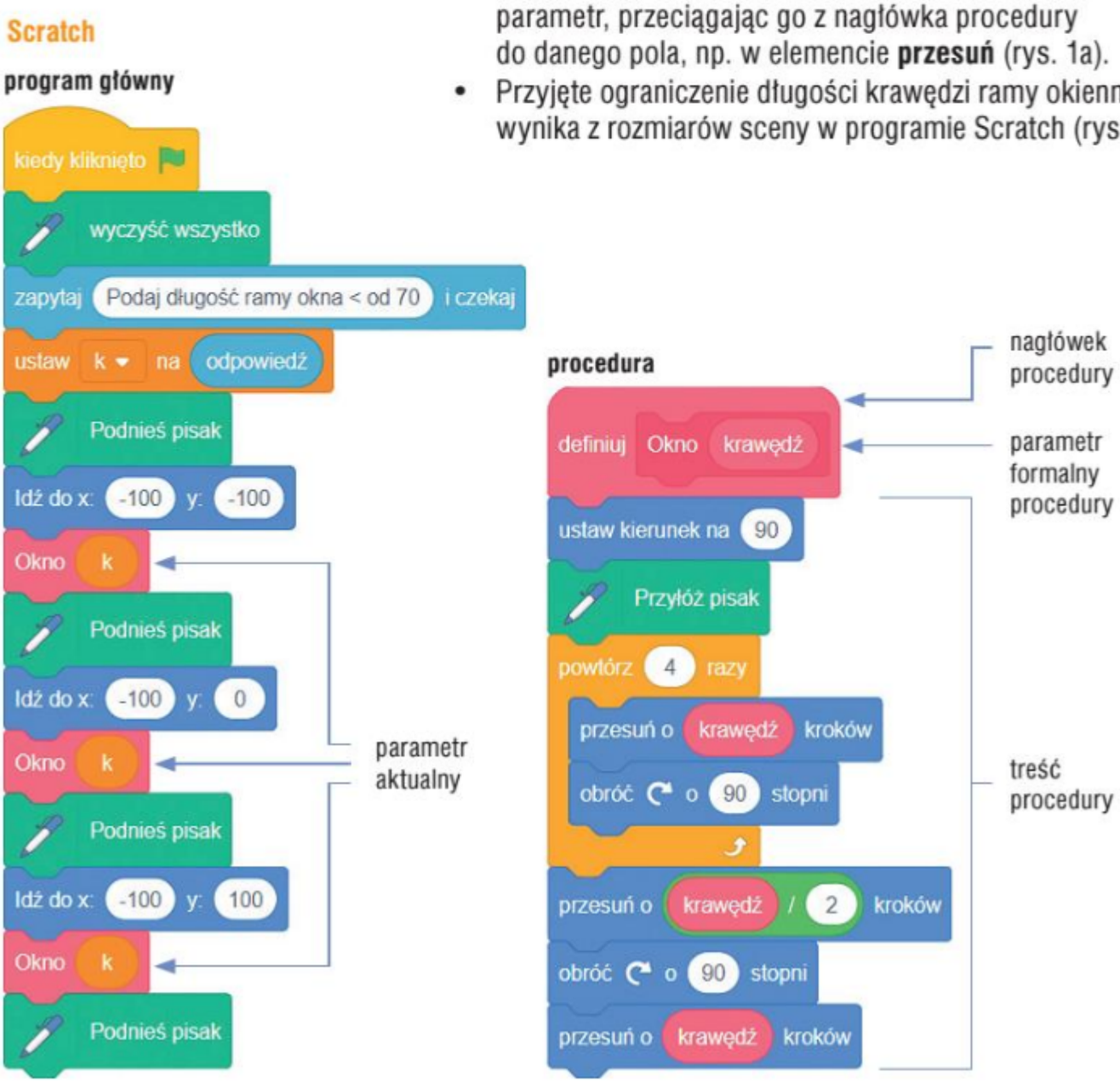
Procedurę (blok) definiujemy, wybierając element **Utwórz blok** z grupy **Moje bloki**. Parametr dodajemy w oknie **Utwórz blok**. Element reprezentujący parametr pojawi się w nagłówku procedury (rys. 1a).

Ćwiczenie 1. Definiujemy procedurę z parametrem w języku Scratch i wywołujemy ją w programie

1. Zdefiniuj procedurę *Okno* z jednym parametrem. Utwórz program rysujący trzy okna, wywołując trzykrotnie procedurę *Okno* (rys. 1a).
2. Zapisz program w pliku pod nazwą *Okna* i uruchom program dla różnych wartości parametru *k*.
3. Rozbuduj program tak, aby duszek-kot narysował kolejne trzy okna. Następnie ma narysować kontur domu, a na koniec stanąć po lewej stronie domu (rys. 1b). Zapisz program w pliku pod nazwą *Dom z oknami*. Sprawdź działanie programu dla różnych danych.

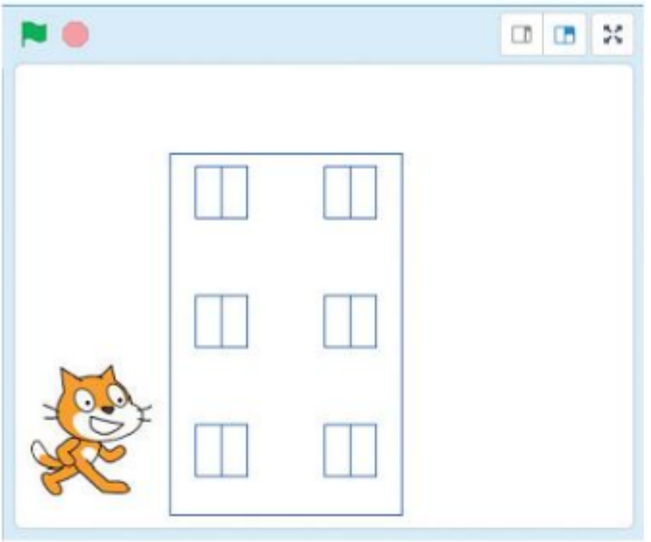
Wskazówki:

- W treści procedury używamy elementu reprezentującego parametr, przeciągając go z nagłówka procedury do danego pola, np. w elemencie **przesuń** (rys. 1a).
- Przyjęte ograniczenie długości krawędzi ramy okiennej wynika z rozmiarów sceny w programie Scratch (rys. 1a).



Rys. 1a. Zastosowanie procedury *Okno* w programie *Okna* (Scratch) – ćwiczenie 1. (pkt. 1.)

Scratch



Rys. 1b. Efekt wykonania programu *Dom z oknami* (Scratch) – ćwiczenie 1. (pkt. 3.)

2. Definiowanie i stosowanie funkcji w języku Python

W języku Python podprogramy nazywamy **funkcjami**. Poznaliśmy kilka wbudowanych funkcji Pythona, m.in.: `print()`, `input()`, `forward()`, `circle()`. W języku Python można definiować własne funkcje.

Funkcje mogą zwracać wartość do programu głównego (rys. 2a) lub nic nie zwracać (rys. 2b).

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

Python

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

Rys. 2a. Ogólna postać definicji funkcji zwracającej wartość (Python)

Python

```
def nazwa_funkcji(lista_parametrów_formalnych):
    lista_instrukcji
```

Rys. 2b. Ogólna postać definicji funkcji niezwracającej wartości (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 formalnego ani typu zwracanej wartości. Podobnie jak w przypadku zmiennych interpreter automatycznie rozpoznaje typ parametru oraz zwracanej wartości.

W nazwach funkcji można używać tylko liter, znaków podkreślenia i cyfr. Nazwa nie może zaczynać się od cyfry. Przyjęte jest niestosowanie polskich liter. Należy stosować nazwy opisujące działanie danej funkcji. Ważne jest, aby przyjąć jednolity sposób nazewnictwa.

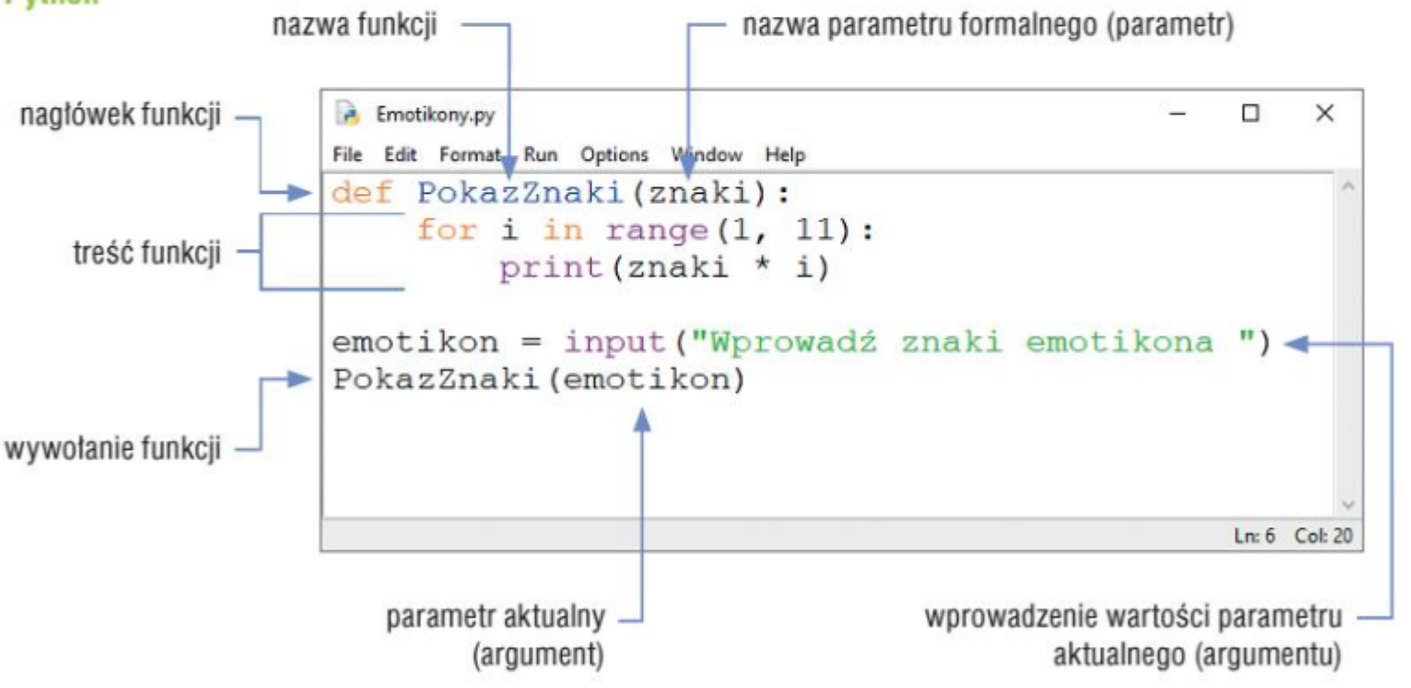
Według popularnego sposobu nazewnictwa w nazwach funkcji i zmiennych używa się wyłącznie małych liter, a w nazwach wieloczłonowych stosuje się znak podkreślenia, np. *wprowadz_dane*, *wyprowadz_dane*, *najwiekszy*.

W podręczniku przyjęliśmy jednak zapis nazw funkcji wielką literą, np. *Maksimum*, *Suma*. Jeśli nazwa jest kilkuczłonowa, następny człon również zaczynamy od wielkiej litery i nie rozdzielamy członów żadnym znakiem, np. *WprowadzDane*, *WyprowadzDane*. W nazwach będziemy stosować skróty, np. *SortWybor*. W ten sposób nazwy funkcji w tym temacie będą odpowiadały nazwom użytym w temacie 8. podręcznika.

Funkcję niezwracającą wartości (rys. 2b) stosujemy na przykład, gdy mamy wyprowadzić dane na ekran lub wyświetlić grafikę. Na rysunkach 3. i 4a. pokazano przykłady funkcji niezwracających wartości. W programie na rys. 3. wprowadzone znaki emotikona zostaną wyświetlone na ekranie 10 razy, a na rys. 4a wyświetli się kompozycja z dziesięciokątów. W treści funkcji nie umieszczamy instrukcji **`return`**.

Aby wywołać funkcję niezwracającą wartości, należy umieścić nazwę funkcji z **parametrami aktualnymi** w odpowiednim miejscu programu głównego (rys. 3.). W przypadku braku parametrów nawiasy po nazwie funkcji pozostawiamy puste.

Python



Rys. 3. Przykład definicji funkcji niezwracającej wartości z jednym parametrem (Python) – ćwiczenie 2.

Ćwiczenie 2. Definiujemy w języku Python funkcję niezwracającą wartości

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

Ćwiczenie 3. Stosujemy funkcję z parametrem do tworzenia grafiki

1. Przepisz program pokazany na rysunku 4a. Zapisz program w pliku pod nazwą *Dziesięciokąty1*.
2. Uruchom program. Objaśnij znaczenie poszczególnych wierszy programu. Które instrukcje są wykonywane w pętli? Ile razy jest wywołana funkcja *Dziesieciokat()* i z jakimi wartościami?
3. Zmodyfikuj program, aby żółw dorysował kompozycję z niebieskich dziesięciokątów tak, jak pokazano na rysunku 4b. Zapisz program w pliku pod nazwą *Dziesięciokąty2*. Uruchom program.

Python

Dziesieciokat_R.py - D:/Python/Dziesieciokat_R.py

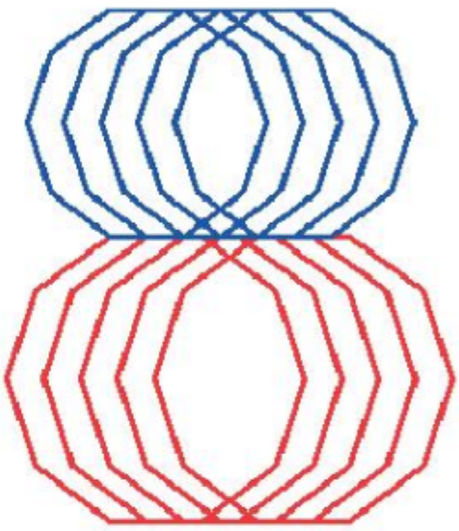
File Edit Format Run Options Window Help

```
from turtle import *

def Dziesieciokat(bok):
    for i in range(10):
        forward(bok)
        right(36)

pensize(3)
pencolor("red")
for i in range(5):
    Dziesieciokat(50)
    forward(20)
hideturtle()
```

Rys. 4a. Program w języku Python z definicją funkcji *Dziesieciokat()* z jednym parametrem – ćwiczenie 3.

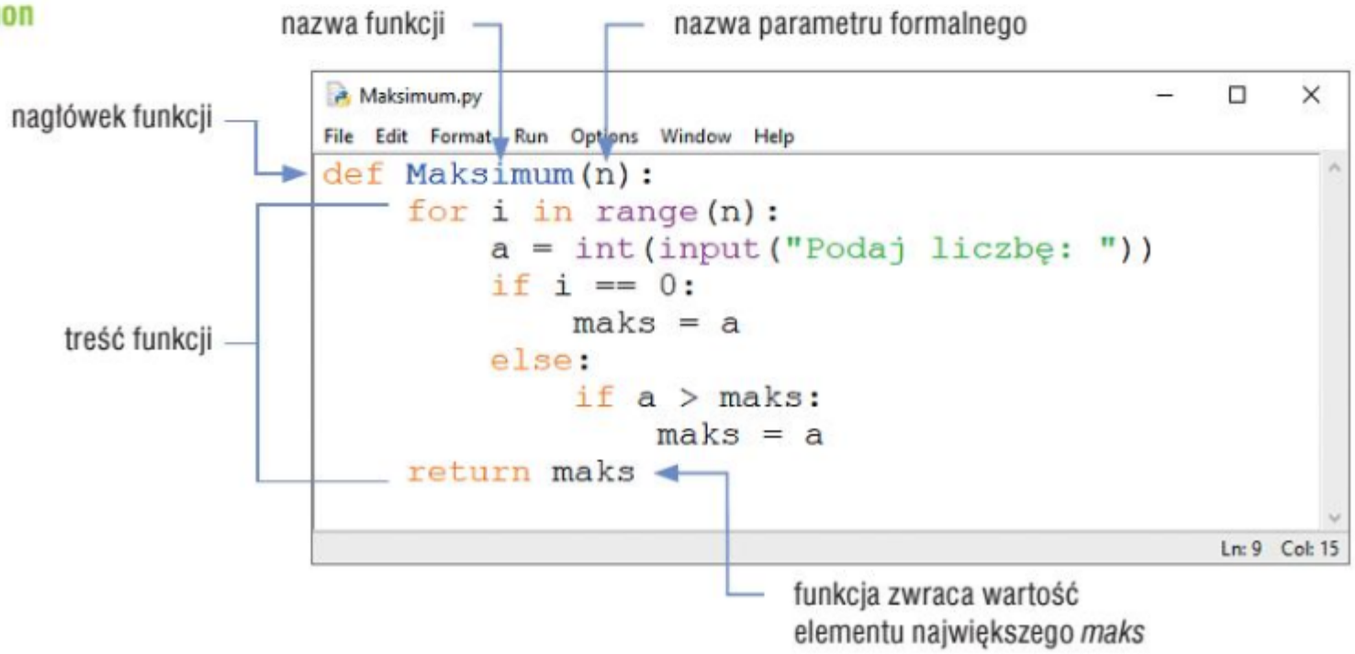


Rys. 4b. Kompozycja z dziesięciokątów o różnych długościach boków – efekt wykonania programu z ćwiczenia 3. (punkt 3.)

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 (rys. 2a).

Sposób definiowania takiej funkcji pokażemy na przykładzie funkcji z jednym parametrem wyszukującej największy element w zbiorze n liczb ($n > 0$) wprowadzonych z klawiatury (rys. 5.). Parametrem przekazywanym funkcji jest liczba elementów zbioru.

Python



Rys. 5. Przykład definicji funkcji zwracającej wartość z jednym parametrem (Python) – ćwiczenie 4.

Ćwiczenie 4. Definiujemy w języku Python funkcję zwracającą wartość

1. Zdefiniuj funkcję *Maksimum* z parametrem *n* (gdzie *n* to liczba elementów w zbiorze) wyszukującą największy element w zbiorze (rys. 5.).
2. Zapisz program w pliku pod nazwą *Maksimum_n_funkcja*.

Wartość zwróconą przez funkcję możemy przypisać zmiennej w programie głównym.

Wywołanie funkcji *Maksimum* powoduje zwrócenie wyniku jej działania. Przypiszmy go zmiennej *najwiekszy* (rys. 6.):

```
najwiekszy = Maksimum(liczba_elementow).
```

wywołanie funkcji *Maksimum* z parametrem aktualnym *liczba_elementow* – wynik przypisujemy zmiennej *najwiekszy*

wprowadzenie liczby elementów zbioru

Python

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

Rys. 6. Wywołanie funkcji *Maksimum* – ćwiczenie 5.

Ćwiczenie 5. Wywołujemy funkcję w programie głównym

1. Otwórz program *Maksimum_n_funkcja* zapisany w ćwiczeniu 4.
2. Pod definicją funkcji umieść instrukcje z rysunku 6.
3. Zapisz program w pliku pod tą samą nazwą i ponownie go uruchom. Sprawdź działanie programu dla kilku różnych danych.

3. Stosowanie list do wprowadzania danych

W funkcji *Maksimum* utworzonej w ćwiczeniu 4. wszystkie wprowadzane liczby były pamiętane w jednej zmiennej (np. *a*). Jeśli wprowadzamy z klawiatury liczby ze zbioru {5, 27, 18, 3, 4}, każda kolejno wprowadzona liczba zastępuje poprzednią (temat 8., rys. 6a). W jaki sposób zapamiętać wszystkie wprowadzone dane?

Niektóre algorytmy wymagają pamiętania wszystkich elementów zbioru w trakcie działania programu. Przykładem jest algorytm porządkowania, ponieważ w efekcie jego działania musimy otrzymać wszystkie elementy w określonym porządku.

Aby zapamiętać wszystkie elementy zbioru, stosujemy zmienne indeksowane, w których do nazwy zmiennej (np. *a*) dodaje się kolejny numer (indeks), np.: $a_0, a_1, \ldots, a_n$. Wówczas dla każdej zmiennej jest rezerwowana oddzielna komórka pamięci (temat 8., rys. 6b).

Aby utworzyć zmienne indeksowane w języku Python, możemy zdefiniować specjalną strukturę danych – **listę** (rys. 7.).

Python

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

Rys. 7. Ogólna postać definiowania listy (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 pierwszy indeks jest zawsze równy 0. Lista (w odróżnieniu od tablicy) 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.

Wartości elementów listy możemy podać od razu w definicji listy.
Na przykład:

```
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 również przypisać od razu taką samą wartość wszystkim elementom listy.

Na przykład:

```
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 też przypisać zmiennej.

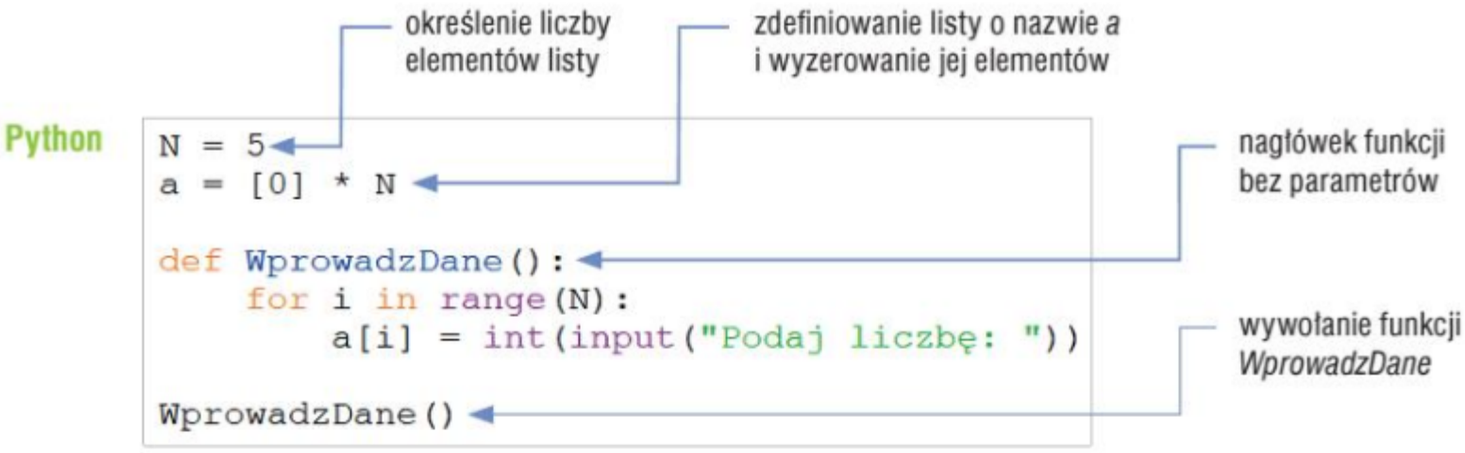
Na przykład:

```
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].

Uwaga: Liczbę przykładów definiowania list w języku Python ograniczyliśmy tylko do kilku, przydatnych do dalszych ćwiczeń.

Do wprowadzania danych do listy zastosujemy funkcję bez parametrów i niezwracającą wartości.



Rys. 8. Definicja funkcji wprowadzającej do listy dane z klawiatury – ćwiczenie 6.

Ćwiczenie 6. Wprowadzamy dane do listy

1. Napisz funkcję o nazwie *WprowadzDane* wprowadzającą dane do listy *a,* składającej się z pięciu liczb całkowitych (rys. 8.).
2. Zapisz program w pliku pod nazwą *Lista*. Uruchom program.

Wskazówka: Funkcja *WprowadzDane* zmodyfikowała wartości istniejących zmiennych (które na początku były równe zero).

Zapisany w języku Python program można również uruchomić, gdy zamkniemy okno z programem. Należy w tym celu z menu kontekstowego ikony programu (widocznej w Eksploratorze plików) wybrać polecenie:

- **Otwórz** – w otwartym oknie zobaczymy uruchomiony program,
- **Edit with IDLE** – program otworzy się w oknie powłoki Pythona (Python Shell) i będzie można go uruchomić.

Python

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

Rys. 9. Definicja funkcji wyprowadzającej dane z listy na ekran – ćwiczenie 7.

Ćwiczenie 7. Wyprowadzamy dane z listy na ekran

1. Otwórz program *Lista* zapisany w ćwiczeniu 6.
2. Zdefiniuj funkcję o nazwie *WyprowadzDane* wyprowadzającą w kolejnych wierszach dane z listy na ekran (rys. 9.).
3. Po poleceniu wywołania funkcji *WprowadzDane* dodaj wywołanie funkcji *WyprowadzDane*.
4. Zapisz program w pliku pod tą samą nazwą. Uruchom program i sprawdź jego działanie dla różnych danych.

4. Zapisywanie w języku Python wybranych algorytmów porządkowania i wyszukiwania

Zamierzamy wykorzystać możliwość zapamiętania danych na liście do zapisania w języku programowania algorytmów porządkowania (metodą przez wybieranie i metodą przez zliczanie) oraz algorytmu wyszukiwania elementu w zbiorze uporządkowanym. Jak to zrobić w języku Python?

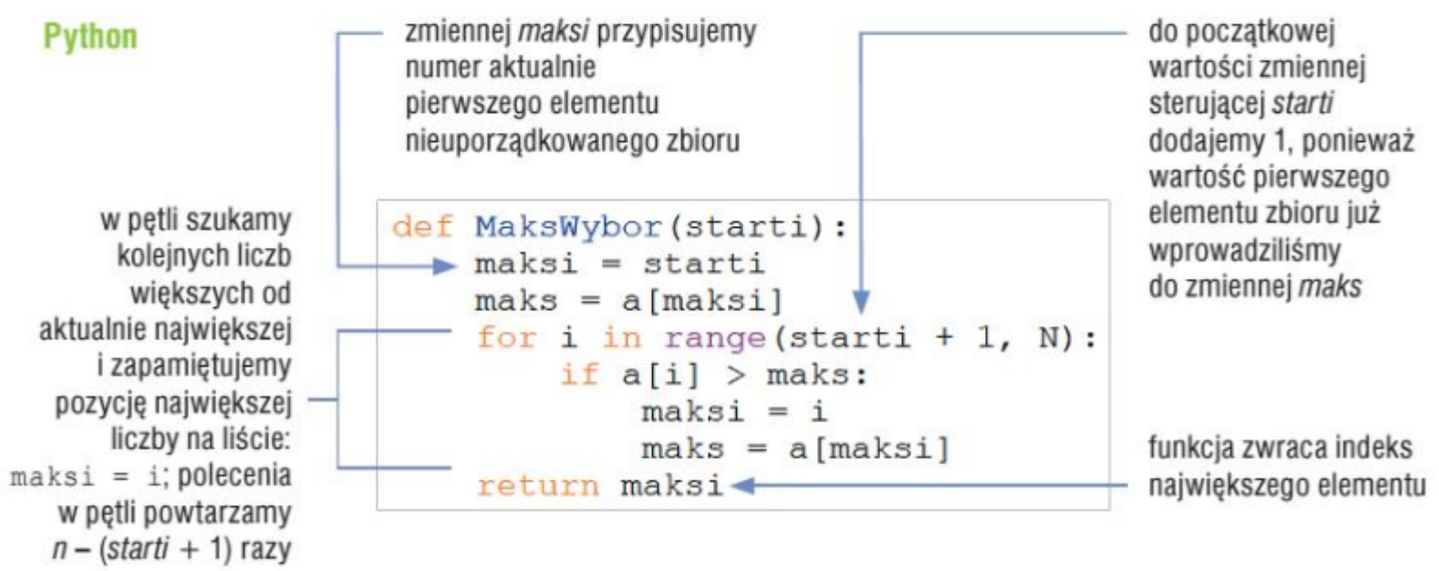
Algorytm porządkowania metodą przez wybieranie

Zapiszemy w języku Python algorytm porządkowania liczb od największej do najmniejszej metodą przez wybieranie. W programie wykorzystamy utworzone wcześniej dwie funkcje *WprowadzDane* do wprowadzenia danych do listy i *WyprowadzDane* – do wyprowadzenia danych na ekran po uporządkowaniu.

W programie wyszukujemy na liście danych *a* największą liczbę *maks* i zapamiętujemy jej pozycję (indeks) w zmiennej *maksi*. Następnie zamieniamy miejscami wyszukaną liczbę z liczbą, która jest aktualnie na początku zbioru. Do zamiany stosujemy zmienną pomocniczą *t*. Wymienione czynności powtarzamy z pominięciem liczb już uporządkowanych.

Zdefiniujemy dwie funkcje:

- *MaksWybor* (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) – rys. 10.,
- *SortWybor* (niezwracającą wartości i bez parametrów), w której wywołujemy w pętli funkcję *MaksWybor* z parametrem aktualnym *i* oraz zamieniamy miejscami znaleziony element największy *maks* z aktualnie pierwszym elementem zbioru – rys. 11. Po każdym przejściu pętli liczba elementów zbioru jest o 1 mniejsza, ponieważ nie sprawdzamy już elementów ustawionych na właściwych miejscach (uporządkowanych).



Rys. 10. Definicja funkcji wyszukującej największy element i zwracającej jego pozycję na liście – ćwiczenie 8.

Python

```
def SortWybor():
    for i in range(N-1):
        maksi = MaksWybor(i)
        t = a[i]
        a[i] = a[maksi]
        a[maksi] = t
```

wywołujemy funkcję *MaksWybor* z parametrem aktualnym *i*, a wynik przypisujemy zmiennej *maksi*

zmienną pomocniczą *t* stosujemy do zamiany miejscami znalezionego największego elementu z aktualnie pierwszym

Rys. 11. Definicja funkcji porządkującej elementy zbioru metodą przez wybieranie – ćwiczenie 8.

Python

```
WprowadzDane()
SortWybor()
print("Dane posortowane metodą przez wybieranie")
WyprowadzDane()
```

Rys. 12. Wywołanie w programie głównym funkcji wprowadzania danych, porządkowania i wyprowadzania danych na ekran – ćwiczenie 8.

W programie głównym wywołamy trzy funkcje: wprowadzania danych (*WprowadzDane*), porządkowania (*SortWybor*) i wyprowadzania danych na ekran (*WyprowadzDane*) – rys. 12.

Ćwiczenie 8. Zapisujemy w języku Python algorytm porządkowania metodą przez wybieranie

1. Przypomnij sobie działanie algorytmu porządkowania metodą przez wybieranie.
2. Otwórz program *Lista* zapisany w ćwiczeniu 7. Uzupełnij program, dodając definicję funkcji *MaksWybor* i *SortWybor* oraz wywołanie funkcji *SortWybor* w programie głównym.
3. Zapisz program w pliku pod nazwą *Wybieranie*. Uruchom program. Sprawdź działanie programu dla różnych danych (elementów listy).

Algorytm porządkowania metodą przez zliczanie

Zapiszemy w języku Python algorytm porządkowania liczb od najmniejszej do największej metodą przez zliczanie. W programie wykorzystamy utworzone wcześniej dwie funkcje: *WprowdzDane* do wprowadzenia danych do listy i *WyprowadzDane* – do wyprowadzenia danych na ekran po uporządkowaniu. Funkcję *WprowadzDane* musimy zmodyfikować, dodając sprawdzanie, czy liczba należy do określonego zakresu. Ustalimy zakres <0, *MAX_DANA*>. Zmiennej *MAX_DANA* przypiszemy stałą wartość, np. 99.

Zdefiniujemy funkcję *SortZlicz* (niezwracającą wartości i bez parametrów), w której zdefiniujemy listę *liczniki* [*MAX_DANA* +1], służącą do zliczania wystąpień elementów zbioru. Liczba liczników, zgodnie z algorytmem, musi umożliwić zliczenie wystąpień każdej liczby z danego zakresu, więc przygotowujemy 100 liczników (dla liczb od 0 do 99).

Python

Zliczanie.py

File Edit Format Run Options Window Help

```
N = 5
a = [0] * N
MAX_DANA = 99

def WprowadzDane():
    print("Wprowadzaj liczby z przedziału < 0,", MAX_DANA, ">")
    for i in range(N):
        a[i] = int(input("Podaj liczbę: "))
        if a[i] < 0:
            a[i] = 0
        if a[i] > MAX_DANA:
            a[i] = MAX_DANA
```

Ln: 14 Col: 0

Rys. 13. Definicja funkcji wprowadzającej dane z określonego zakresu do listy – ćwiczenie 9.

W pierwszej pętli **`for`** (rys. 15., wiersze 3-4) dla każdej liczby wprowadzonej przez użytkownika zwiększamy odpowiadający jej licznik o 1.

Na rysunku 14. pokazano przykładową listę *liczniki* dla liczb {0, 2, 3, 5, 5}.

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

Rys. 14. Lista *liczniki* dla liczb {0, 2, 3, 5, 5}

W pętli **`for`** (rys. 15., wiersze 7-9) 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 7-9 komputer wykonuje dla każdej możliwej wartości danych wejściowych (rys. 15., pętla **`for`** – wiersz 6.). Mamy tu zatem do czynienia z **zagnieżdżeniem pętli.** Zmienna *poz* określa aktualną pozycję na liście *a*.

Python

```
1 def SortZlicz():
2     liczniki = [0] * (MAX_DANA +1)
3     for i in range(N):
4         liczniki[a[i]] = liczniki[a[i]] + 1
5     poz = 0
6     for i in range(MAX_DANA+1):
7         for j in range(liczniki[i]):
8             a[poz] = i
9             poz = poz + 1
```

definiujemy i równocześnie zerujemy listę liczników

dla każdej wprowadzonej liczby zwiększamy odpowiadający jej licznik o 1

układamy elementy listy *a* od najmniejszego do największego

Rys. 15. Definicja funkcji porządkującej elementy zbioru metodą przez zliczanie – ćwiczenie 9.

Python

```
WprowadzDane()
WyprowadzDane()
print("Dane posortowane metodą przez zliczanie")
SortZlicz()
WyprowadzDane()
```

Rys. 16. Wywołanie w programie głównym funkcji wprowadzania danych, porządkowania metodą przez zliczanie i wyprowadzania wyniku na ekran – ćwiczenie 9.

W programie głównym wywołamy trzy funkcje: wprowadzania danych (*WprowadzDane*), porządkowania (*SortZlicz*) i wyprowadzania danych na ekran (*WyprowadzDane*) – rys. 16.

Ćwiczenie 9. Zapisujemy w języku Python algorytm porządkowania metodą przez zliczanie

1. Przypomnij sobie działanie algorytmu porządkowania metodą przez zliczanie.
2. Otwórz program *Lista* zapisany w ćwiczeniu 7. Zmodyfikuj funkcję *WprowadzDane* tak, aby dodatkowo sprawdzała, czy liczba należy do określonego zakresu (rys. 13.).
3. Uzupełnij program, dodając definicję funkcji *SortZlicz* (rys. 15.) oraz wywołanie funkcji *SortZlicz* w programie głównym (rys. 16.).
4. Zapisz program w pliku pod nazwą *Zliczanie* i ponownie go uruchom. Sprawdź działanie programu dla różnych danych.

Algorytm wyszukiwania elementu w zbiorze uporządkowanym

Program ma wyszukiwać w zbiorze uporządkowanym daną podaną przez użytkownika i podawać jej pozycję na liście. Zakładamy, że wprowadzamy dane od najmniejszej do największej (inaczej program nie będzie działał poprawnie). Bez takiego założenia należałoby dodać również funkcję sortowania danych w określonym porządku. W programie wykorzystamy utworzoną wcześniej funkcję *WprowadzDane* (rys. 8.) do wprowadzenia danych do listy.

Zdefiniujemy funkcję *ZnajdzDana* z parametrem *wartosc* (rys. 17.). W funkcji wyznaczamy pozycję elementu środkowego (zmienna *srodek*).

Jeśli element środkowy (*a*[*srodek*]) jest szukaną daną (zmienna *wartosc*), to funkcja zwraca zmienną *srodek*, czyli pozycję danego elementu.

Jeśli szukana dana jest mniejsza od danej środkowej, odrzucamy prawą część zbioru (liczby większe od elementu środkowego) – ustalamy nowy koniec (`koniec = srodek - 1`).

W przeciwnym wypadku odrzucamy lewą część zbioru (liczby mniejsze od elementu środkowego) i ustalamy nowy początek (`poczatek = srodek + 1`).

Połowienie powtarzamy dopóki początek zbioru jest mniejszy lub równy końcowi zbioru (`poczatek <= koniec`) – o ile nie znajdziemy szukanego elementu (wtedy komputer wykonana instrukcję **`return`** `srodek`).

W algorytmie liczba iteracji zależy od spełnienia (lub niespełnienia) warunku, dlatego zamiast instrukcji iteracyjnej **`for`**, w której z góry określamy liczbę powtórzeń, użyjemy instrukcji iteracyjnej **`while`**, którą omówiliśmy w temacie 6.

Python

```
def ZnajdzDana(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
```

wyznaczamy pozycję elementu środkowego; operator // oznacza dzielenie z obcięciem części ułamkowej

sprawdzamy, czy element środkowy jest szukaną liczbą

wyznaczamy nowy koniec lub początek

Rys. 17. Definicja funkcji wyszukującej daną w zbiorze uporządkowanym – ćwiczenie 10.

W funkcji *ZnajdzDana* (rys. 17.) instrukcja **return** występuje dwukrotnie. Niezależnie od miejsca, w którym występuje, zawsze kończy wykonanie funkcji. Jeśli element środkowy jest szukaną liczbą, funkcja zwraca wartość zmiennej *srodek* (**return** srodek).

Jeśli w przeszukiwanym zbiorze nie ma szukanej liczby, funkcja *ZnajdzDana* (rys. 17.) zwraca wartość -1 (**return** -1). Programiści często używają wartości niepoprawnych (w tym przypadku indeks listy nie może być liczbą ujemną) do zasygnalizowania niepowodzenia operacji lub błędu.

W programie głównym wywołamy funkcję wprowadzania danych (*WprowadzDane*), wprowadzimy z klawiatury daną do wyszukania (zmienna *wartosc*), wywołamy funkcję wyszukiwania danej (*ZnajdzDana*) i wypiszemy wynik. Uwzględnimy również sytuację, gdy podanej wartości nie ma w zbiorze (rys. 18.).

Python

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

Rys. 18. Wywołanie w programie głównym funkcji wprowadzania danych i wyszukiwania określonej danej oraz wyprowadzenie wyniku na ekran – ćwiczenie 10.

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

1. Przypomnij sobie działanie algorytmu wyszukiwania elementu w zbiorze uporządkowanym.
2. Otwórz program *Lista* zapisany w ćwiczeniu 7. Usuń funkcję *WyprowadzDane*.
3. Uzupełnij program, dodając definicję funkcji *ZnajdzDana* (rys. 17.). W programie głównym wywołaj funkcję *ZnajdzDana* i dodaj instrukcje wprowadzania szukanej danej i wyświetlania wyniku (rys. 18.).
4. Zapisz program w pliku pod nazwą *Wyszukiwanie*. Uruchom program. Sprawdź działanie programu dla różnych danych (pamiętaj o wprowadzaniu danych uporządkowanych – od najmniejszej do największej).

Warto zapamiętać

- Aby pisać przejrzyste programy, należy opracowywać oddzielnie problemy cząstkowe, stosując podprogramy (procedury i funkcje). Podprogram należy najpierw zdefiniować, a potem wywołać w programie głównym.
- W języku Python możemy zdefiniować funkcje:
  - niezwracające wartości, gdy chcemy np. wyprowadzić dane,
  - zwracające wartość, gdy celem funkcji jest obliczenie i zwrócenie pewnej wartości, np. wyniku sumowania.
- Parametr to wartość przekazywana do funkcji, a wartość zwracana to wartość przekazywana z funkcji.
- Możemy definiować funkcje z parametrami i bez parametrów. W momencie wywołania funkcji parametr formalny przyjmie wartość parametru aktualnego.
- Aby wywołać funkcję z parametrami, należy podać jej nazwę i parametry aktualne w nawiasach okrągłych. Gdy parametry nie występują, nawiasy pozostawiamy puste.
- Program główny komunikuje się z funkcją poprzez przekazywanie parametrów.
- W funkcji zwracającej wartość musi wystąpić instrukcja `return` z argumentem przedstawiającym zwracaną wartość. Wartość zwracaną przez funkcję (reprezentowaną przez nazwę funkcji i parametry umieszczone w nawiasach okrągłych) przypisujemy zmiennej.
- Aby zaprogramować algorytm, w którym w trakcie działania programu musimy pamiętać wszystkie elementy zbioru, możemy zapisywać te dane na liście.
- Do elementów listy (np. o nazwie *lista* i liczbie elementów *N*) odwołujemy się poprzez podanie nazwy listy i indeksu umieszczonego w nawiasach kwadratowych *lista*[0], *lista*[1], …, *lista*[N - 1].
- W języku Python nie określamy typu elementów listy. Interpreter określa typ elementów na podstawie rodzaju danych, jaki im przypiszemy.
- Definiowanie funkcji i stosowanie list możemy wykorzystać do zapisania w języku Python algorytmów porządkowania metodą przez wybieranie, porządkowania metodą przez zliczanie oraz wyszukiwania elementu w zbiorze uporządkowanym.

Pytania i polecenia

1. Dlaczego definiujemy podprogramy?
2. Czym jest parametr formalny, a czym aktualny w podprogramie (procedurze, funkcji)?
3. Kiedy definiujemy funkcję niezwracającą wartości, a kiedy zwracającą wartość w języku Python? Wyjaśnij na przykładzie.
4. W jaki sposób wywołujemy w języku Python funkcję bez parametrów, a w jaki – z parametrami?
5. W jakim celu definiujemy listę? Podaj przykład użycia listy.
6. Czym jest zmienna indeksowana? Wyjaśnij na przykładach sposób definiowania listy w języku Python.
7. Wyjaśnij sposób zapisania w języku Python algorytmu wyszukiwania metodą przez wybieranie. Do czego służą funkcje zdefiniowane w tym programie?
8. Dlaczego w programie realizującym algorytm porządkowania liczb od największej do najmniejszej metodą przez wybieranie musimy zapamiętywać pozycję znalezionej największej liczby w oddzielnej zmiennej?
9. Wyjaśnij sposób zapisania w języku Python algorytmu wyszukiwania metodą przez zliczanie. Do czego służą funkcje zdefiniowane w tym programie?

10. W jakim celu definiujemy listę *liczniki* w programie realizującym algorytm porządkowania metodą przez zliczanie?
11. Wyjaśnij sposób zapisania w języku Python algorytmu wyszukiwania elementu w zbiorze uporządkowanym. Do czego służą funkcje zdefiniowane w tym programie?
12. Dlaczego program realizujący algorytm wyszukiwania elementu w zbiorze uporządkowanym zaczyna się od sprawdzenia, czy środkowy element jest szukaną liczbą?

Zadania

1. W języku Scratch zdefiniuj dwie procedury: *Pięciokąt* z parametrem *bok_p* rysującą pięciokąt foremny i *Siedmiokąt* z parametrem *bok_s* rysującą siedmiokąt foremny, gdzie *bok_p* i *bok_s* określają odpowiednio długości boków pięciokąta i siedmiokąta. Korzystając z tych procedur, utwórz kompozycję z pięciokątów i siedmiokątów według własnego pomysłu. Wartości parametrów aktualnych (*p* i *s*) dla zdefiniowanych procedur podawaj z klawiatury po uruchomieniu programu. Zapisz program w pliku pod nazwą *Pięciokąty i siedmiokąty*.
2. Napisz w języku Python program umożliwiający wyprowadzenie na ekran monitora napisu „Gwiazdkowe pozdrowienia", a pod nim piętnastu gwiazdek (znaków *). Zdefiniuj funkcję *Gwiazdki* niezwracającą wartości i bez parametrów, wyprowadzającą w jednym wierszu piętnaście gwiazdek. Funkcję wywołaj w programie głównym. Zapisz program w pliku pod nazwą *Życzenia*.
3. Zmodyfikuj program *Życzenia* zapisany w zadaniu 2. tak, aby wyświetlił w kolejnym wierszu napis „Wszystkiego najlepszego", a pod nim ponownie piętnaście gwiazdek. Zapisz program w pliku pod tą samą nazwą.
4. Napisz w języku Python program umożliwiający obliczenie objętości sześcianu. Zdefiniuj funkcję *Szescian* z jednym parametrem o nazwie *b*, obliczającą sześcian liczby *b* i zwracającą do programu głównego wynik obliczenia. Wywołaj funkcję w programie głównym z parametrem aktualnym *bok*. Wartość parametru wprowadzaj z klawiatury. Zapisz program w pliku pod nazwą *Objętość sześcianu*.
5. Korzystając ze zdefiniowanej w ćwiczeniu 4. funkcji *Maksimum*, zdefiniuj funkcję *Minimum* i napisz program wyszukujący najmniejszy element w zbiorze *n*-elementowym. Zapisz program w pliku pod nazwą *Min_n_funkcja*.
6. Zdefiniuj listę składającą się z dwunastu elementów, będących liczbami dni kolejnych miesięcy (dla lutego przyjmij 28 dni). Utwórz program wypisujący w kolumnie elementy listy, czyli liczby dni kolejnych miesięcy. Zapisz program w pliku pod nazwą *Miesiące*.
   Wskazówka: Wykorzystaj przykład listy *L_DNI_W_MIESIACU* podanej w punkcie 3. tematu.
7. Korzystając z listy zdefiniowanej w zadaniu 6., utwórz program, który dla podanego z klawiatury numeru miesiąca wyświetli liczbę dni tego miesiąca. Zapisz program w pliku pod nazwą *Miesiąc*. Efekt wykonania programu powinien być taki, jak pokazano na rysunku 18.
8. Napisz w języku Python program realizujący algorytm wyszukiwania danego elementu w zbiorze nieuporządkowanym, korzystając z opisu podanego w temacie 7., w punkcie 1.3. Do wprowadzenia danych wykorzystaj funkcję *WprowadzDane()*, a do wyprowadzenia – funkcję *WyprowadzDane()*. Do wyszukania danej zastosuj funkcję *WyszukajDana()*, której wzór pokazano na rysunku 20. Wszystkie funkcje wywołaj w programie głównym. Szukaną wartość wprowadzaj z klawiatury. Zapisz program w pliku pod nazwą *Wyszukiwanie_nieuporządkowany*.