Temat C2. Stosowanie funkcji w językach C++ i Python

1. Funkcje w językach C++ i Python
2. Funkcje zwracające wartość w językach C++ i Python
3. Funkcje niezwracające wartości w języku C++ i Python

Warto powtórzyć

1. W jaki sposób deklarujemy zmienne w języku C++?
2. Co w języku C++ określa typ zmiennej?
3. W jaki sposób nadajemy wartość zmiennej w wybranym języku programowania (C++ lub Python)?
4. W jaki sposób wyświetlamy wyniki na ekranie komputera w wybranym języku programowania (C++ lub Python)?
5. W jakich środowiskach programowania tworzyliśmy programy z wykorzystaniem podprogramów?

1. Funkcje w językach C++ i Python

Do magazynu sprzętu RTV dowożone są kilkakrotnie w ciągu miesiąca różne artykuły.
Jak napisać program zliczający, ile sztuk każdego artykułu dostarczono we wszystkich dostawach łącznie?

Podprogram (procedura lub funkcja) **P**

Wyodrębniona część programu, mająca unikatową nazwę i ustalony sposób wymiany danych z innymi częściami programu.

Jeśli, wykonując zadanie, musimy kilkakrotnie zsumować liczby w różnych zbiorach, możemy opracować cząstkowe rozwiązanie obliczające sumę, a potem wielokrotnie wykorzystać je w programie. Do opracowywania problemów cząstkowych, które są częściami zadania rozwiązywanego przez dany program, służą **podprogramy**.

Podprogram realizujący określone zadanie (np. sumowanie liczb) może być wielokrotnie wykorzystywany – w tym samym programie lub w innych. Stosowanie podprogramów zwiększa przejrzystość programu.

W językach C++ i Python wszystkie podprogramy nazywamy **funkcjami**.

Poznaliśmy kilka wbudowanych funkcji Pythona, m.in.: `print()`, `input()`, a w języku C++ funkcję `main()` stanowiącą główny program. W językach C++ i Python można definiować własne funkcje.

Funkcje mogą **zwracać wartość** do programu głównego lub nic nie zwracać, mogą mieć określone **parametry** lub ich nie posiadać.

Funkcje dzielimy na: zwracające wartość i niezwracające wartości.

P **Parametr**

Wartość przekazywana do funkcji.

W **Wartość zwracana**

Wartość przekazywana z funkcji do miejsca wywołania (np. programu głównego).

Nazwy funkcji tworzymy na takich samych zasadach jak nazwy zmiennych (można używać tylko liter, znaków podkreślenia i cyfr; nazwa nie może zaczynać się od cyfry; przyjęte jest niestosowanie polskich liter). Funkcjom należy nadawać nazwy opisujące ich działanie. Ważne jest, aby przyjąć jednolity sposób nazewnictwa, np. w nazwach funkcji i zmiennych używa się wyłącznie małych liter, a w nazwach wieloczłonowych stosuje się znak podkreślenia, np. *liczba_dostaw*, *suma_sztuk*, *wprowadz_dane*, *wyprowadz_dane*, *najwiekszy*.

Aby zastosować funkcję, trzeba ją **zdefiniować**, a następnie **wywołać** w programie.

2. Funkcje zwracające wartość w językach C++ i Python

Funkcję zwracającą wartość stosujemy, jeśli celem funkcji jest obliczenie i zwrócenie pewnej wartości do programu. Funkcja musi zawierać instrukcję `return` z wartością zwracaną do miejsca wywołania (np. programu głównego).

Funkcja zwracająca wartość	Nagłówek funkcji
	`opis_typu nazwa_funkcji(lista_parametrów_formalnych)`
	`{`
	`  lista_instrukcji;`
	`  return wartość;`
	`}`
	Treść funkcji
		C++

	Nagłówek funkcji
	`def nazwa_funkcji(lista_parametrów_formalnych):`
	`  lista_instrukcji`
	`  return wartość`
	Treść funkcji
		Python

***Tabela 1.** Ogólna postać definicji funkcji zwracającej wartość w językach C++ i Python*

Miejsce definiowania funkcji	Funkcja może zostać zdefiniowana przed jej pierwszym wywołaniem lub, jeżeli zostanie zadeklarowana (poprzez umieszczenie nagłówka funkcji) przed funkcją `main()`, w dowolnym miejscu programu – ale poza funkcją `main()`.	C++
	Funkcję definiujemy w dowolnym miejscu programu (zazwyczaj na początku), ale przed pierwszym użyciem.	Python
Określanie typu funkcji i parametrów	Należy wskazać typ funkcji (typ zwracanej wartości) i określić typy oraz nazwy parametrów formalnych.	C++
	Nie określamy typu funkcji ani typów parametrów formalnych funkcji – interpreter rozpozna je automatycznie (podobnie jak w przypadku zmiennych).	Python
Treść funkcji	Treść (*lista_instrukcji* i instrukcja **`return`**) musi się znaleźć w bloku **`{}`**.	C++
	Treść funkcji (*lista_instrukcji* i instrukcja **`return`**) musi być przesunięta w prawo przynajmniej o jedną spację względem nagłówka funkcji (przyjęte jest wcięcie składające się z czterech spacji).	Python

Tabela 2. *Zapis funkcji w językach C++ i Python*

Wywołanie funkcji powoduje wykonanie instrukcji będących treścią funkcji i zwrócenie wyniku jej działania. W momencie wywołania funkcji parametry formalne są zastępowane przez parametry aktualne.

Uwaga

W przypadku większej liczby parametrów oddzielamy je przecinkami.

Aby wywołać funkcję, należy podać nazwę funkcji, a w nawiasach – wartości parametrów. Wartość zwróconą przez funkcję możemy np. przypisać zmiennej w programie głównym. W przypadku definiowania i wywoływania funkcji bez parametrów nawiasy po nazwie funkcji pozostawiamy puste.

Sposób definiowania i wywołania funkcji zwracającej wartość pokażemy na przykładzie funkcji z jednym parametrem, obliczającej sumę *n* liczb.

Przykład 1. Definiowanie i wywołanie funkcji zwracającej wartość z jednym parametrem w językach C++ i Python

Zadanie: Do pewnego magazynu sprzętu RTV dowożone są kilkakrotnie w ciągu miesiąca telewizory i głośniki. Utwórz program obliczający i wyprowadzający na ekran łączne wielkości dostaw (transportów) telewizorów i głośników. Liczby transportów telewizorów (*l_dostaw_tv*) i głośników (*l_dostaw_glosnikow*) wprowadzaj z klawiatury po uruchomieniu programu.
Na początku programu zdefiniuj funkcję *suma* z jednym parametrem, obliczającą i zwracającą sumę *n* liczb wprowadzanych z klawiatury. Parametrem przekazywanym do funkcji jest liczba elementów zbioru *n*. Funkcję *suma* wywołaj dwukrotnie w programie głównym z parametrami aktualnymi *l_dostaw_tv* i *l_dostaw_glosnikow*.

Dane: liczby naturalne większe od zera *l_dostaw_tv* i *l_dostaw_glosnikow*, oznaczające odpowiednio liczbę dostaw telewizorów i głośników,
l_dostaw_tv i *l_dostaw_glosnikow* liczb naturalnych większych od zera oznaczających odpowiednio wielkości kolejnych dostaw telewizorów i głośników.

Wyniki: łączna liczba telewizorów z *l_dostaw_tv* transportów: *suma_tv*, łączna liczba głośników z *l_dostaw_glosnikow* transportów: *suma_glosnikow*.

Opis rozwiązania: Rysunek przedstawia trzy dostawy telewizorów (czyli *l_dostaw_tv* = 3), w których dostarczono kolejno: 8, 15 i 20 telewizorów, a więc *suma_tv* = 43.

W programie dwa razy obliczamy sumę, a zatem zdefiniowaną funkcję *suma* wywołamy dwukrotnie, ale z innym parametrem aktualnym (raz z parametrem *l_dostaw_tv*, a drugi raz – *l_dostaw_glosnikow*). Wynik działania funkcji (zwracaną wartość) przypiszemy zmiennej.

Wywołanie funkcji *suma* – sposób 1.

C++
```
suma_tv = suma(l_dostaw_tv);
suma_glosnikow = suma(l_dostaw_glosnikow);
```

Python
```
suma_tv = suma(l_dostaw_tv)
suma_glosnikow = suma(l_dostaw_glosnikow)
```

Wynik działania funkcji zwracającej wartość możemy również wyprowadzić na ekran bez używania dodatkowych zmiennych (tu: *suma_tv* i *suma_glosnikow*) i przypisywania im wartości funkcji.

Wywołanie funkcji *suma* – sposób 2.

C++
```
cout << "Liczba tv z " << l_dostaw_tv << " dostaw wynosi "
<< suma(l_dostaw_tv);
cout << "Liczba glosnikow z " << l_dostaw_glosnikow << "
dostaw wynosi " << suma(l_dostaw_glosnikow);
```

Python
```
print("Liczba tv z", l_dostaw_tv, "dostaw wynosi",
suma(l_dostaw_tv))
print("Liczba głośników z", l_dostaw_glosnikow, "dostaw
wynosi", suma(l_dostaw_glosnikow))
```

C++

[*] Sumy_dostaw.cpp

```cpp
#include <iostream>
using namespace std;

float suma (int n)
{
    int a, s = 0;

    for (int i = 0; i < n; i++)
    {
        cout << "Podaj liczbe sztuk w " << i+1 << " dostawie: ";
        cin >> a;
        s += a;
    }
    return s;
}

int main()
{
    int l_dostaw_tv, suma_tv;

    cout << "Ile bylo dostaw tv: ";
    cin >> l_dostaw_tv;
    suma_tv = suma(l_dostaw_tv);
    cout << "Liczba tv z " << l_dostaw_tv << " dostaw wynosi " << suma_tv;
    return 0;
}
```

parametr formalny *n*

definicja funkcji *suma*

wartość zwracana

wywołanie funkcji *suma* z parametrem aktualnym *l_dostaw_tv*

część wykonawcza programu

Python

Sumy_dostaw.py - C:\Python\Sumy_dostaw.py (3.7.2)

File Edit Format Run Options Window Help

```python
def suma(n):
    s = 0
    for i in range(n):
        print("Podaj liczbę sztuk w", i + 1, "dostawie:" )
        a = int(input())
        s = s + a
    return s

l_dostaw_tv = int(input("Ile było dostaw tv: "))
suma_tv = suma(l_dostaw_tv)
print("Liczba tv z", l_dostaw_tv, "dostaw wynosi", suma_tv)

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

parametr formalny *n*

definicja funkcji *suma*

wartość zwracana

część wykonawcza programu

wywołanie funkcji *suma* z parametrem aktualnym *l_dostaw_tv*

Ćwiczenie 1. Definiujemy funkcję zwracającą wartość z jednym parametrem i wywołujemy ją w programie głównym

1. Utwórz nowy plik źródłowy i przepisz wybrany program z przykładu 1.
2. Zapisz program w pliku pod nazwą *Sumy_dostaw*.
3. Uruchom i przetestuj program dla kilku różnych wartości zmiennych. Wyjaśnij znaczenie poszczególnych wierszy programu.
4. W programie zapisano tylko obliczenia łącznej wielkości dostaw telewizorów. Dodaj podobne instrukcje dla obliczenia łącznej wielkości dostaw głośników.
5. Zapisz plik pod tą samą nazwą.

Wskazówki:

- W edytorze, w którym tworzysz kod źródłowy programu, można kopiować instrukcje i wklejać je w inne miejsce.
- Ułatwiaj sobie pracę i kopiuj podobne fragmenty programu, w których wystarczy tylko zmienić nazwy zmiennych.

Ćwiczenie 2. Dodajemy do funkcji i programu sprawdzanie poprawności danych

1. Otwórz plik *Sumy_dostaw* zapisany w ćwiczeniu 1.
2. Zmodyfikuj funkcję i program tak, aby obliczenia były wykonywane dla poprawnych (czyli większych od zera) wartości wszystkich zmiennych: *a*, *l_dostaw_tv*, *l_dostaw_glosnikow*. Jeśli zostanie wprowadzona niepoprawna wartość, wyświetl na ekranie komunikat „Wprowadzono bledna liczbe" i zignoruj wprowadzoną wartość.
3. Zapisz program w pliku pod nazwą *Sumy_dostaw_popr*.
4. Uruchom i przetestuj program dla kilku różnych wartości zmiennych.

Ćwiczenie 3. Modyfikujemy program

1. Otwórz plik *Sumy_dostaw_popr* zapisany w ćwiczeniu 2.
2. Zmniejsz liczbę zmiennych, czyli do wyprowadzenia wyników funkcji użyj sposobu 2. z przykładu 1.
3. Do programu dodaj zliczanie dwóch innych artykułów przywożonych do magazynu: smartfonów i komputerów.
4. Zapisz program w pliku pod nazwą *Sumy_dostaw_zmod*.

Funkcje można definiować bez parametrów, ale wówczas, aby przekazać do nich dane, musimy zastosować tzw. **zmienne globalne** (widoczne w dowolnym miejscu programu) – w funkcjach i w programie głównym.

Niezależnie od zmiennych globalnych można w funkcji stosować **zmienne lokalne**. W języku Python zmienne lokalne są widoczne w całej funkcji, a w C++ od miejsca deklaracji do końca funkcji (do nawiasu klamrowego zamykającego definicję funkcji). W funkcji *suma* (przykład 1.) zmienne *i*, *a* oraz *s* są lokalne (przy czym w języku C++ zakres zmiennej *i* jest ograniczony tylko do pętli `for()`).

Przykład 2. Stosowanie funkcji zwracającej wartość bez parametrów w językach C++ i Python

Zadanie: Zdefiniuj funkcję *obwod* bez parametrów, zwracającą do programu głównego obwód kwadratu o boku *bok*. Wywołaj funkcję w programie głównym i wyprowadź wartość obwodu. Wartość zmiennej *bok* wprowadzaj z klawiatury.

Dane: liczba naturalna *bok* oznaczająca długość boku kwadratu.

Wynik: wartość obwodu kwadratu.

Opis rozwiązania: W programie zastosujemy zmienną globalną *bok*.

C++ W języku C++ zmienną *bok* musimy zadeklarować przed definicją funkcji *obwod*.

Python W języku Python instrukcja `bok = int(input("Podaj bok kwadratu: "))` określa typ zmiennej *bok*.

C++

Obwod_kwadratu.cpp

```
#include <iostream>
using namespace std;
int bok;

int obwod()
{
    return 4*bok;
}

int main()
{
    cout << "Podaj bok kwadratu: ";
    cin >> bok;
    cout << "Obwod kwadratu o boku " << bok << " wynosi " << obwod();
    return 0;
}
```

Python

Obwod_kwadratu.py - C:\Python\Obwod_kwadratu.py (3.7.2)

File Edit Format Run Options Window Help

```
def obwod():
    return 4*bok

bok = int(input("Podaj bok kwadratu: "))
print("Obwod kwadratu o boku", bok, " wynosi", obwod())

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

Ćwiczenie 4. Definiujemy funkcję zwracającą wartość bez parametrów i wywołujemy ją w programie głównym

1. W edytorze kodu źródłowego przepisz wybrany program z przykładu 2. Zapisz program w pliku pod nazwą *Obwod_kwadratu*.
2. Uruchom i przetestuj program dla kilku różnych wartości zmiennej. Wyjaśnij znaczenie poszczególnych wierszy programu.

Ćwiczenie 5. Modyfikujemy program

1. Dodaj do programu zapisanego w ćwiczeniu 4. funkcję *pole* bez parametru obliczającą pole kwadratu o boku *bok*.
2. Zapisz program w pliku pod nazwą *Obwod_i_pole_kwadratu*. Uruchom i przetestuj program dla kilku różnych wartości zmiennej.

Dla zwiększenia przejrzystości i uniwersalności programu należy unikać wykorzystywania zmiennych globalnych, a stosować funkcje z parametrami.

Ćwiczenie 6. Definiujemy funkcję zwracającą wartość z parametrem

Otwórz plik *Obwod_i_pole_kwadratu* zapisany w ćwiczeniu 5. Zmodyfikuj program, rezygnując ze zmiennej globalnej i zmieniając definicje funkcji *suma* i *obwod* na funkcje z parametrem. W programie głównym wywołaj funkcję z parametrem aktualnym *x*, wprowadzanym z klawiatury jako liczba całkowita. Zapisz program w pliku pod nazwą *Obwod_i_pole_kwadratu_z_par*.

Wskazówka: W obydwu funkcjach można użyć tej samej nazwy parametru. W języku C++ w 3. wierszu programu (przykład 2.) zrezygnuj z deklaracji `int bok;`

C++

```
4  int obwod(int bok)
5  {
6      return 4*bok;
7  }
8  int pole(int bok)
9  {
10     return bok*bok;
11 }
```

Python

```
def obwod(bok):
    return 4*bok

def pole(bok):
    return bok*bok
```

Rys. 1. *Definicje funkcji z parametrem – ćwiczenie 6.*

3. Funkcje niezwracające wartości w językach C++ i Python

Funkcje niezwracające wartości stosujemy, gdy mamy wykonać pewne czynności, np. wprowadzić dane, wyprowadzić dane lub obliczyć wiele wartości.

W języku C++ funkcja taka nie ma typu, dlatego w części nagłówkowej przed nazwą funkcji należy umieścić słowo kluczowe **`void`**. Funkcja nie zwraca wartości, dlatego instrukcja **`return`** nie musi wystąpić (a jeśli występuje, to nie może określać wartości).

W języku Python nie określamy typu parametru formalnego funkcji, ponieważ, podobnie jak w przypadku zmiennych, interpreter automatycznie rozpoznaje typ parametru. Funkcja nie zwraca wartości, dlatego instrukcja **`return`** nie musi wystąpić (a jeśli występuje, to nie może określać wartości).

Funkcja niezwracająca wartości	`void nazwa_funkcji(lista_parametrów_formalnych)`
	`{`
	`  lista_instrukcji;`
	`}`
		C++

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

Tabela 3. *Ogólna postać definicji funkcji niezwracającej wartości w językach C++ i Python*

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

Wywołanie funkcji niezwracającej wartości	`nazwa_funkcji(lista_parametrów_aktualnych);`
	lub
	`nazwa_funkcji();`
		C++

	`nazwa_funkcji(lista_parametrów_aktualnych)`
	lub
	`nazwa_funkcji()`
		Python

Tabela 4. *Ogólna postać wywołania funkcji niezwracającej wartości w językach C++ i Python*

Przykład 3. Stosowanie funkcji niezwracającej wartości bez parametrów w językach C++ i Python

Zadanie: Napisz program umożliwiający wyprowadzenie na ekran monitora napisu „Poznajemy funkcje", a pod nim dwudziestu gwiazdek (znaków *). Zdefiniuj funkcję niezwracającą wartości o nazwie *gwiazdki* bez parametrów. Zadaniem funkcji jest wyprowadzenie w jednym wierszu dwudziestu gwiazdek i przeniesienie kursora do nowego wiersza. Funkcję wywołaj w programie głównym.

C++ Gwiazdki.cpp

```
1  #include <iostream>
2  using namespace std;
3
4  void gwiazdki()
5  {
6      for(int i=0; i<20; i++)
7          cout << "*";
8      cout << endl;
9  }
10
11 int main()
12 {
13     cout << "Poznajemy funkcje" << endl;
14     gwiazdki();
15     return 0;
16 }
```

definicja funkcji

wywołanie funkcji

Python

Gwiazdki.py - C:\Python\Gwiazdki.py (3.7.2)

File Edit Format Run Options Window Help

```
def gwiazdki():
    print("*"*20)

print("Poznajemy funkcje")
gwiazdki()

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

definicja funkcji

wywołanie funkcji

Opis rozwiązania: Celem funkcji jest wyświetlenie z góry określonej liczby gwiazdek. Żadna wartość nie jest przekazywana z programu głównego, więc funkcja nie musi mieć parametrów.

C++ Jedyną zmienną w programie jest zmienna *i*, użyta w funkcji jako zmienna lokalna.

Python W programie nie używamy żadnych zmiennych.

Ćwiczenie 7. Definiujemy funkcję niezwracającą wartości bez parametrów i wywołujemy w programie głównym

1. W edytorze kodu źródłowego przepisz wybrany program z przykładu 3. Zapisz program w pliku pod nazwą *Gwiazdki*.
2. Uruchom i przetestuj program. Wyjaśnij znaczenie poszczególnych wierszy programu.

Ćwiczenie 8. Modyfikujemy program

1. Dodaj do programu zapisanego w ćwiczeniu 7. wyświetlenie w kolejnych wierszach tekstów: „w języku C++" (lub „w języku Python"), „na lekcji informatyki". Pod każdym napisem wyświetl gwiazdki. Ile razy wywołasz funkcję *gwiazdki()*?
2. Zapisz program w pliku pod nazwą *Gwiazdki_z_napisami*.

Przykład 4. Stosowanie funkcji niezwracającej wartości z parametrami w językach C++ i Python

Chcemy, aby liczba wyświetlanych gwiazdek była wprowadzana z klawiatury i przekazywana do funkcji jako parametr.
Do definicji funkcji dodamy parametr *n*, który w momencie wywołania funkcji będzie zastąpiony parametrem aktualnym *l_gwiazdek* (liczbą całkowitą większą od zera wprowadzaną z klawiatury).

C++

```
4  void gwiazdki(int n)
5  {
6      for(int i=0; i<n; i++)
7          cout << "*";
8      cout << endl;
9  }
```

Python

```
def gwiazdki(n):
    print("*"*n)
```

Wywołanie funkcji w programie głównym:

C++ `gwiazdki(l_gwiazdek);`

Python `gwiazdki(l_gwiazdek)`

Ćwiczenie 9. Definiujemy funkcję niezwracającą wartości z parametrami i wywołujemy w programie głównym

1. Otwórz plik *Gwiazdki* zapisany w ćwiczeniu 7. Zmień definicję funkcji *gwiazdki* na funkcję z parametrem *n* i wywołaj w programie głównym z parametrem aktualnym, będącym liczbą całkowitą (*l_gwiazdek*). Wartość parametru wprowadzaj z klawiatury. Zapisz program w pliku pod nazwą *Gwiazdki_par*.
2. Uruchom i przetestuj program dla różnych wartości zmiennej *l_gwiazdek*.

Ćwiczenie 10. Modyfikujemy program

1. Otwórz plik *Gwiazdki_z_napisami* zapisany w ćwiczeniu 8. Zmień definicję funkcji *gwiazdki* na funkcję z parametrem *n* i wywołaj w programie głównym z parametrem aktualnym będącym liczbą całkowitą (*l_gwiazdek*). Wartość parametru wprowadzaj z klawiatury. Zapisz program w pliku pod nazwą *Gwiazdki_z_napisami_par*.
2. Uruchom i przetestuj program dla różnych wartości zmiennej *l_gwiazdek*.

Warto zapamiętać

- Podprogramy (funkcje) stosujemy, aby opracowywać oddzielnie problemy cząstkowe i w ten sposób pisać przejrzyste programy.
- Funkcje w językach C++ i Python mogą zwracać wartości lub nie. Każdy rodzaj funkcji możemy definiować z parametrami lub bez.
- Funkcję wywołujemy w programie głównym, podając jej nazwę i parametry aktualne. W przypadku braku parametrów, po nazwie funkcji umieszczamy puste nawiasy. W przypadku funkcji zwracającej wartość, wartość zwracaną możemy np. przypisać zmiennej w programie głównym.
- Program główny komunikuje się z funkcją przez przekazywanie parametrów – parametry aktualne są wprowadzane w miejsce parametrów formalnych w momencie wywołania funkcji. Gdy wywołanie funkcji się zakończy, zwraca wartość jako swój wynik.

Pytania i polecenia

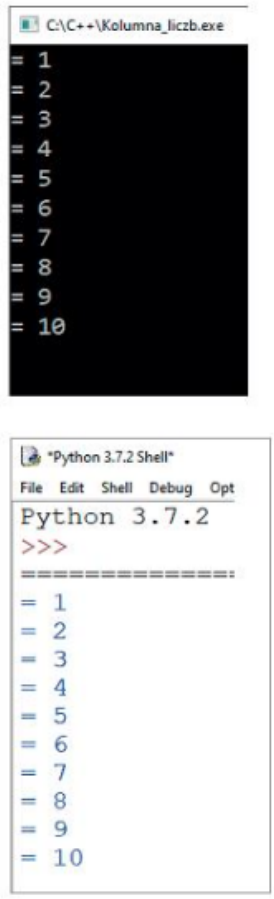
1. Dlaczego definiujemy podprogramy?
2. Czym jest parametr formalny, a czym aktualny?
3. Kiedy definiujemy funkcję niezwracającą wartości, a kiedy zwracającą wartość w języku C++ lub Python?
4. Jak wywołujemy funkcję bez parametrów, a jak z parametrami w języku C++ lub Python?

Zadania

1. Do programu *Sumy_dostaw_zmod* zapisanego w ćwiczeniu 3. dodaj jeszcze dwa inne artykuły, które są dostarczane do magazynu i wykonaj te same obliczenia co dla telewizorów i innych artykułów. Zapisz program w pliku *Sumy_dostaw_6*.
2. Napisz program umożliwiający obliczenie objętości graniastosłupa prawidłowego czworokątnego. Zdefiniuj funkcję *objetosc* z dwoma parametrami (*a*, *h*), obliczającą objętość graniastosłupa prawidłowego czworokątnego o krawędzi podstawy *a*

i wysokości *h* i zwracającą do programu głównego wynik obliczenia. Wywołaj funkcję w programie głównym z parametrami aktualnymi *bok* i *wysokosc* wprowadzanymi z klawiatury jako liczby całkowite. Wyprowadź na ekran wynik – objętość graniastosłupa prawidłowego czworokątnego o krawędzi podstawy *bok* i wysokości *wysokosc*. Zapisz program w pliku pod nazwą *Objetosc_graniastoslupa*.

3. Do programu *Objetosc_graniastoslupa* zapisanego w zadaniu 2. dodaj sprawdzanie poprawności wprowadzanych danych. Jeśli użytkownik wprowadzi niepoprawną (ujemną lub równą zero) wartość zmiennej *bok* lub *wysokosc*, wyświetl na ekranie komunikat „Wprowadzono bledna liczbe". Zapisz program w pliku pod nazwą *Objetosc_graniastoslupa_popr*.
4. Zdefiniuj funkcję *kolumna* typu `void` bez parametrów, wyświetlającą na ekranie w kolumnie liczby od 1 do 10 poprzedzone znakiem „=" i spacją – jak pokazano na rysunku 2. Funkcję wywołaj w programie głównym. Zapisz program w pliku pod nazwą *Kolumna_liczb*.
5. Zmodyfikuj program *Kolumna_liczb* zapisany w zadaniu 4., dodając do funkcji parametr *ile* określający liczbę wyświetlanych rzędów liczb. Funkcję wywołaj w programie głównym z parametrem aktualnym *ile_liczb*. Dodaj sprawdzanie poprawności wprowadzanych danych (muszą być większe od zera). Zapisz program w pliku pod nazwą *Kolumna_liczb_popr*.



Rys. 2. *Wynik działania programu – zadanie 4.*

Dla zainteresowanych

6. Wymyśl samodzielnie problem (podobnie jak w przykładzie 1.), w którym możesz wykorzystać funkcję *suma*. Sformułuj zadanie i jego specyfikację. W wybranym języku programowania napisz program realizujący to zadanie.
7. Utwórz program zliczający, ile liczb parzystych wprowadzono i wyświetlający wynik na ekranie. Zliczanie wykonuj, dopóki nie zostanie wprowadzone zero. Zdefiniuj funkcję logiczną (typu `bool`) *czy _parzysta* z parametrem całkowitym *a*, sprawdzającą, czy liczba jest parzysta. Wywołaj funkcję w programie głównym. Zapisz program w pliku pod nazwą *Zlicz_parzyste_popr*.
   Wskazówka: Funkcja logiczna *czy _parzysta* zwraca wartość logiczną PRAWDA lub FAŁSZ (w języku C++: `true` / `false`, w języku Python: `True` / `False`]. Postać funkcji może być następująca:

C++
```
4  bool czy_parzysta(int a)
5  {
6      return a%2 == 0;
7  }
```

Python
```
def czy_parzysta(a):
    return a%2 == 0
```

Rys. 3. *Definicje funkcji – zadanie 7.*

8. Napisz funkcję *nwd*(*a*, *b*) obliczającą największy wspólny dzielnik dwóch liczb całkowitych *a* i *b*, będących parametrami funkcji. Wykorzystaj funkcję w programie głównym do wyznaczenia największego wspólnego dzielnika dwóch liczb całkowitych.