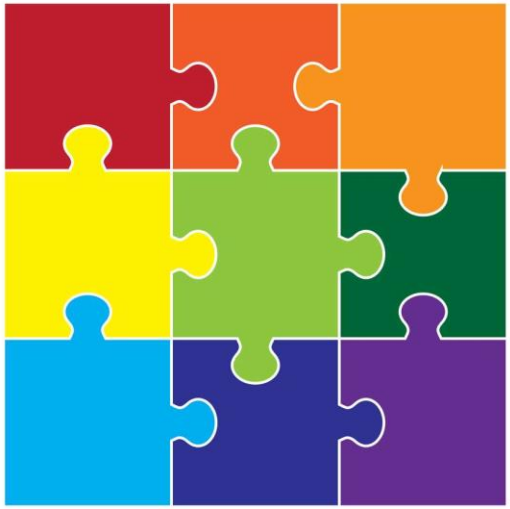

Funkcje w Pythonie

M@я3k Pų𝔡€£kØ

Programowanie w Pythonie

Spis treści

Podprogram

- Przy wielkich programach problemem staje się zarządzanie dużą ilością linijek kodu.
- Rozwiązaniem jest podział dużego kodu na części zwane podprogramami.
- Podprogram zawiera część kodu, który stanowi zwartą całość i może być używany wielokrotnie.
- Pozwala uprościć program główny i zwiększyć czytelność kodu.

Funkcje

- Funkcje w Pythonie to bloki kodu wielokrotnego użytku, definiowane słowem kluczowym **def**.
- Pozwalają na modularność, łatwiejsze zarządzanie kodem i unikanie powtórzeń.
- Dzięki nim uzyskuje się jasną i przejrzystą strukturą przy dużych możliwościach i elastyczności.
- Funkcje organizują program, przyjmują argumenty (dane wejściowe) w nawiasach, wykonują operacje i opcjonalnie zwracają wynik za pomocą return.

Przykład funkcji

```
def modul_liczby(a):
    if a < 0:
        a = -a
    else:
        a = a
    return a

a = float(input("Podaj liczbę: "))
m = modul_liczby (a)
print (m)

print(modul_liczby(a))

print(f"Modul liczby {a} wynosi {modul_liczby(a)}")
```

Przykład funkcji

```
def modul_liczby(a):      #początek funkcji modul_liczby
    if a < 0:
        a = -a
    else:
        a = a
    return a        #Zwracana wartość funkcji
#koniec funkcji
# ---------program główny----------
a = float(input("Podaj liczbę: "))
# Wywołanie funkcji i drukowanie wyniku
m = modul_liczby (a)
print (m)

print(modul_liczby(a))

print(f"Modul liczby {a} wynosi {modul_liczby(a)}")
```

Opis budowy funkcji

def	Słowo kluczowe rozpoczynające definicję funkcji
nazwa_funkcji	Opisowa nazwa (zwykle zapisywana konwencją **snake_case**), która określa, co funkcja robi (np. `oblicz_pole`)
parametr1, **parametr2**, …	Zmienne (argumenty), które funkcja przyjmuje; mogą być opcjonalne
Ciało funkcji	Wcięty kod (4 spacje), który wykonuje zadania
return	Zwraca wartość z funkcji, kończąc jej działanie

```
def modul_liczby(a):
    if a < 0:
        a = -a
    else:
        a = a
    return a
```

Nazwa funkcji

- W języku Python nazwy funkcji powinny być zgodne z konwencją **snake_case**, czyli wszystkie słowa z małych liter, oddzielone znakiem podkreślenia _.
  - Ta konwencja określona jest przez twórców tego języka i większość programistów się jej trzyma.

```
wyczysc_ekran
divide_numbers
exclude_point_from_screen
```

- Nazwa funkcji określa, co się powinno wydarzyć, gdy tę funkcję się wywoła.
  - Jest najczęściej czasownikiem (`clear`, `go_to_store`, `print`).

Zwracanie wartości

- Funkcja zwracająca wartość:

```
def modul_liczby(a):
    if a < 0:
        a = -a
    else:
        a = a
    return a          #zwracanie wartości
```

- Funkcja bezwrotna:

```
def wyswietl_liczby(a):
    for i in range (a):
        print (i)
```

- Funkcja może zarówno zwracać wartość zwrotną jak i nie robić tego. Jeśli funkcja nie zwraca żadnej wartości przy użyciu **return**, to domyślnie zwraca ona wartość **None**.

```
def fun():
    print("Have a good time!")
ret = fun()   # Have a good time!
print(ret)    # None.
```

Wywoływanie funkcji cz. 1

- Wywołanie funkcji bez parametrów

```
def wyswietl_komunikat ():
      print("Błąd dzielenia przez zero! ")

a, b, c = 0, 0, 0
if b == 0:
    wyswietl_komunikat ()
c = a/b
```

- Wywołanie funkcji z parametrami

```
def pole_prostokata (a, b):
      p = a * b  return p
a, b, p = 3, 4, 0
p = pole_prostokata (a,b)
print (p)
```

- Dzięki nim uzyskuje się jasną i przejrzystą strukturą przy dużych możliwościach i elastyczności.

Wywoływanie funkcji cz. 2

- Wywołanie funkcji z parametrami domyślnymi.

```
def pole_prostokata (a, b = 10):
        p = a * b
        return p

a, b, p = 3, 4, 0
p = pole_prostokata (a, b)       # podano b
print (p)                        #12
p = pole_prostokata (a)          #nie podano b
print (p)                        #30
```

- Wywołując funkcję można zastosować **nazywanie parametrów**. Python pozwala wtedy na napisanie parametrów nie po kolei.

```
def dzielenie (a, b):
        c = a / b
        return c

a, b, c = 20, 4, 0
c = dzielenie (a, b)                 # zwykła kolejność
print (c)                        #5
c = dzielenie(b = 10, a = 5) #nazwanie parametrów i odwrotna kolejność
print (c)                        #0.5
```

Przekazywanie zmiennych

- Przekazując zmienne typu **int**, **float**, **string**, **bool** do funkcji to przekazuje się ich kopie. Są to zmienne lokalne widoczne tylko w danej funkcji lub części programu.
- Zmiany ich wartości nie będą widoczne w głównej części programu. :

```
def kwadrat (a):
    a = a * a
    print ("a w funkcji" , a)
    return a
a = 4
print ("a startowe" , a)
kwadrat (a)
print ("a poza funkcja" , a)
a = kwadrat (a)
print ("a poza funkcja zmienione" , a)
```

Zmienne globalne

- Zmienne globalne to zmienne, które mają zasięg w całym programie.
- Tworzy się je stawiając słowo kluczowe `global` przed nazwą zmiennej:

```
def kwadrat(b):
    global a
                    # zmienna globalna widoczna w całym programie
    a = b * b
a = 100
kwadrat (a)
print (a) # zostanie wypisane 10000
```

- Zalety zmiennych globalnych:
  - Dostęp do nich możliwy z dowolnego miejsca w kodzie.
  - Umożliwiają łatwe przechowywanie konfiguracji lub wspólnych stanów dla wielu funkcji.
  - W bardzo prostych, krótkich skryptach mogą przyspieszyć pisanie kodu.
- Wady zmiennych globalnych:
  - Dowolna część programu może zmienić zmienną globalną, co utrudnia zrozumienie, kiedy i dlaczego wartość uległa zmianie.
  - Zagrożenie kolizją nazw - ryzyko przypadkowego nadpisania zmiennej.
  - Utrudnione testowanie.
  - Dostęp do zmiennych globalnych jest wolniejszy niż do lokalnych.

Ćwiczenie cz.1

1. Napisz program wczytujący liczbę n i wyliczający dla niej **silnię** n! (mnożenie liczb od 1 do n) i **sumę** liczb od 1 do n. Każdą z nich zapisz w oddzielnej funkcji.
2. Napisz program podający informacje o kole. Wczytuje wielkość promienia i podaje informacje wyliczane w oddzielnych funkcjach:
   a) Średnica, pole, obwód, pole kwadratu wpisanego w koło i pole kwadratu opisanego na kole.
3. Napisz program liczący pola figur geometrycznych. Każda z nich ma być zapisana i liczona w oddzielnej funkcji. Wywołując funkcje jako argumentu użyj odpowiednich wielkości (bok, podstawa, wysokość)
   a) Kwadrat, prostokąt, romb, trójkąt.
4. Napisz program wczytujący dwie liczby a i b, a potem realizujący na nich obliczeń. Każda z nich ma być realizowana w oddzielnej funkcji:
   a) Suma, różnica, mnożenie, dzielenie, dzielenie modulo, potęgowanie a^b

Ćwiczenie cz.2

4. Napisz program wczytujący 10 liczb i obliczający w oddzielnych funkcjach różne informacje o tych liczbach.
   a) Suma, średnia arytmetyczna, element maksymalny, element minimalny, różnicę między największym i najmniejszym elementem, liczbę liczb parzystych, liczbę liczb nieparzystych.
5. Napisz program wczytujący liczbę i obliczający w oddzielnych funkcjach wartości ciągów:
   a) Fibonacciego, Tribonacciego, Tetrabonacciego.
6. Napisz funkcję `czy_pierwsza(n)`, która sprawdzi, czy liczba jest pierwsza.
7. Napisz funkcję `czy_palindrom(tekst)`, która sprawdzi, czy napis jest palindromem
8. Napisz program wczytujący tekst i podający w oddzielnych funkcjach informacje o tym tekście:
   a) Długość tekstu, ilość samogłosek, ilość spółgłosek, odwrócenie tekstu.

Powtórzenie

- Dlaczego dzielimy programy na podprogramy?
- Jak jest realizowana praktycznie idea dzielenia programu w Pythonie?
- Jakie właściwości musi mieć nazwa funkcji?
- Jak przekazywać zmienne do funkcji?
- Czym się różnią zmienne lokalne i globalne?
- Co robi polecenie return?