Programowanie Python 1

(CP1S02005)

Politechnika Białostocka - Wydział Elektryczny

Cyfryzacja przemysłu, sem. II, studia stacjonarne I stopnia

Rok akademicki 2023/2024

Wykład nr 8 (24.04.2024)

dr inż. Jarosław Forenc

Plan wykładu nr 8

- Funkcje w Pythonie
  - ogólna struktura funkcji
  - przekazywania i zwracanie wartości
  - moduły
  - zalecenia
  - docstringi

Python - funkcje (struktura funkcji)

- ogólna struktura funkcji w języku Python:

```
def nazwa_funkcji(parametry):
    """Opis funkcji (opcjonalny)"""
    # Ciało funkcji
    wynik = wyrażenie
    return wynik
```

- definicja funkcji rozpoczyna się od słowa kluczowego def, po którym podana jest nazwa funkcji i jej parametry (w nawiasach zwykłych)
- na końcu definicji funkcji znajduje się dwukropek (:)
- opis funkcji (tzw. docstring) to specjalny, wieloliniowy ciąg znaków umieszczany po definicji funkcji służący do dokumentowania kodu
- ciało funkcji to blok kodu definiujący operacje wykonywane w funkcji
- instrukcja return służy do określenia wartości zwracanej przez funkcję

Python - funkcje

- wywołując funkcję podajemy nazwę funkcji, a po niej umieszczamy w nawiasach zwykłych argumenty funkcji (wartości)

```
wynik = nazwa_funkcji(argumenty)
```

- funkcja obliczająca i zwracająca sumę dwóch liczb

```
def dodaj(a, b):
    suma = a + b
    return suma

wynik = dodaj(5, 10)
print(f"Suma liczb wynosi: {wynik}")
```

```
Suma liczb wynosi: 15
```

- ciało funkcji tworzą wszystkie wcięte wiersze kodu

Python - funkcje

- wywołując funkcję podajemy nazwę funkcji, a po niej umieszczamy w nawiasach zwykłych argumenty funkcji (wartości)

```
wynik = nazwa_funkcji(argumenty)
```

- funkcja obliczająca i zwracająca sumę dwóch liczb (wersja krótsza)

```
def dodaj(a, b):
    return a + b

print(f"Suma liczb wynosi: {dodaj(5, 10)}")
```

```
Suma liczb wynosi: 15
```

Python - funkcje

- wywołanie funkcji możliwe jest po jej deklaracji
- nie można wywoływać funkcji, która nie została jeszcze zadeklarowana

```
wynik = dodaj(5, 10)
print(f"Suma liczb wynosi: {wynik}")

def dodaj(a, b):
    suma = a + b
    return suma
```

```
Traceback (most recent call last):
  File "d:\MyApp.py", line 1, in <module>
    wynik = dodaj(5, 10)
            ^^^^^
NameError: name 'dodaj' is not defined
```

Python - funkcje

- wywołanie funkcji bez argumentów lub z ich błędną liczbą także spowoduje błąd

```
def dodaj(a, b):
    suma = a + b
    return suma

wynik = dodaj()
print(f"Suma liczb wynosi: {wynik}")
```

```
Traceback (most recent call last):
  File "d:\MyApp.py", line 5, in <module>
    wynik = dodaj()
            ^^^^^^^
TypeError: dodaj() missing 2 required positional
arguments: 'a' and 'b'
```

Python - funkcje

- funkcja może nie mieć żadnych argumentów
- taka funkcja zazwyczaj wyświetla informacje na ekranie

```
def powitanie():
    print("Witaj świecie!")

powitanie()
```

```
Witaj świecie!
```

- w funkcji nie musi występować instrukcja return (ale może)

```
def powitanie():
    print("Witaj świecie!")
    return

powitanie()
```

Python - funkcje

- funkcja z argumentami też może tylko wyświetlać informacje na ekranie

```
def powitanie(imie):
    print(f"Witaj, {imie}!")
    return

powitanie("Ania")
powitanie("Paweł")
```

```
Witaj, Ania!
Witaj, Paweł!
```

Python - funkcje (przekazywanie argumentów)

- argumenty pozycyjne - argumenty w wywołaniu funkcji mają taką samą kolejność jak parametry w definicji funkcji

```
def oblicz_bmi(waga, wzrost):
    bmi = waga / (wzrost ** 2)
    return bmi

BMI = oblicz_bmi(75, 1.80)
print(f"BMI wynosi: {BMI:.2f}")
```

```
BMI wynosi: 23.15
```

Python - funkcje (przekazywanie argumentów)

- argumenty w postaci słów kluczowych - argumenty to pary nazwa-wartość przekazywane do funkcji
- nie ma znaczenia kolejność umieszczenia argumentów w wywołaniu funkcji

```
def oblicz_bmi(waga, wzrost):
    bmi = waga / (wzrost ** 2)
    return bmi

BMI = oblicz_bmi(waga=75, wzrost=1.80)
print(f"BMI wynosi: {BMI:.2f}")

BMI = oblicz_bmi(wzrost=1.80, waga=75)
print(f"BMI wynosi: {BMI:.2f}")
```

```
BMI wynosi: 23.15
BMI wynosi: 23.15
```

Python - funkcje (przekazywanie argumentów)

- argumenty w postaci słów kluczowych - argumenty to pary nazwa-wartość przekazywane do funkcji
- jeśli nazwy parametrów nie będą zgadzały się to wystąpi błąd

```
def oblicz_bmi(waga, wzrost):
    bmi = waga / (wzrost ** 2)
    return bmi

BMI = oblicz_bmi(Waga=75, wzrost=1.80)
print(f"BMI wynosi: {BMI:.2f}")
```

```
Traceback (most recent call last):
  File "d:\MyApp.py", line 4, in <module>
    BMI = oblicz_bmi(Waga=75, wzrost=1.80)
          ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
TypeError: oblicz_bmi() got an unexpected keyword
argument 'Waga'
```

Python - funkcje (parametry domyślne)

- dla każdego parametru funkcji można zdefiniować wartość domyślną

```
def kod_pocztowy(kod="15-351", miejscowosc="Białystok"):
    wynik = f"{kod} {miejscowosc}"
    return wynik

print(f"Adres 1: {kod_pocztowy("00-950","Warszawa")}")
print(f"Adres 2: {kod_pocztowy("15-433")}")
print(f"Adres 3: {kod_pocztowy()}")
```

```
Adres 1: 00-950 Warszawa
Adres 2: 15-433 Białystok
Adres 3: 15-351 Białystok
```

- jeśli podczas wywołania funkcji nie zostanie podany argument dla parametru, to zostania zastosowana wartość domyślna parametru

Python - funkcje (parametry domyślne)

- parametry domyślne powinny być umieszczane po parametrach bez wartości domyślnych

```
def kod_pocztowy(kod, miejscowosc="Białystok"):
    wynik = f"{kod} {miejscowosc}"
    return wynik

print(f"Adres 1: {kod_pocztowy("00-950","Warszawa")}")
print(f"Adres 2: {kod_pocztowy("15-433")}")
```

```
Adres 1: 00-950 Warszawa
Adres 2: 15-433 Białystok
```

Python - funkcje (argumenty opcjonalne)

- w funkcji można zastosować argumenty opcjonalne

```
def osoba(imie1, nazwisko, imie2=""):
    if imie2:
        wynik = f"{imie1} {imie2} {nazwisko}"
    else:
        wynik = f"{imie1} {nazwisko}"
    return wynik

os = osoba("Jan", "Kowalski")
print(os)
os = osoba("Anna", "Jopek", "Maria")
print(os)
```

```
Jan Kowalski
Anna Maria Jopek
```

Python - funkcje (zwracanie wartości)

- w najprostszym przypadku funkcja może zwracać prostą wartość

```
def get_text(marka, rok):
    tekst = f"Twój {marka} ma {2024-rok} lat"
    return tekst

samochod = get_text("Opel",1988)
print(samochod)
```

```
Twój Opel ma 36 lat
```

- instrukcja return pobiera wartość z wnętrza funkcji i przekazuje ją do miejsca wywołania funkcji

Python - funkcje (zwracanie wartości)

- funkcja może zwracać wartość dowolnego typu, np. słownik, listę

```
def osoba(imie, nazwisko):
    person = {"imię":imie, "nazwisko":nazwisko}
    return person

os = osoba("Jan", "Kowalski")
print(os)
```

```
{'imię': 'Jan', 'nazwisko': 'Kowalski'}
```

Python - funkcje (zwracanie wartości)

- funkcja może zwracać wartość dowolnego typu, np. słownik, listę

```
def osoba(imie, nazwisko, wiek=None):
    person = {"imię":imie, "nazwisko":nazwisko}
    if wiek:
        person["wiek"] = wiek
    return person

os = osoba("Jan", "Kowalski", wiek=23)
print(os)

os = osoba("Jan", "Kowalski")
print(os)
```

```
{'imię': 'Jan', 'nazwisko': 'Kowalski', 'wiek': 23}
{'imię': 'Jan', 'nazwisko': 'Kowalski'}
```

- wartość None będzie użyta, gdy zmiennej nie zostanie przypisana żadna konkretna wartość

Python - funkcje (przekazywanie wartości)

- do funkcji można przekazywać wartości złożonych typów, np. listy

```
def powitanie(osoby):
    for osoba in osoby:
        print(f"Witaj, {osoba}!")

osoby = {"Jan","Kasia","Piotr","Magda"}
powitanie(osoby)
```

```
Witaj, Jan!
Witaj, Piotr!
Witaj, Kasia!
Witaj, Magda!
```

Python - funkcje (przekazywanie wartości)

- lista przekazywana do funkcji może być modyfikowana
- wszystkie wprowadzone zmiany są trwałe

```
def pan_pani(osoby):
    for i, osoba in enumerate(osoby):
        osoby[i] = f"Pani {osoba}" if osoba[-1] == "a"
                                   else f"Pan {osoba}"

osoby = ["Jan","Kasia","Piotr","Magda"]
print(osoby)
pan_pani(osoby)
print(osoby)
```

```
['Jan', 'Kasia', 'Piotr', 'Magda']
['Pan Jan', 'Pani Kasia', 'Pan Piotr', 'Pani Magda']
```

Python - funkcje (przekazywanie wartości)

- do funkcji można przekazać kopię listy (stosujemy notację wycinka [:])

```
def pan_pani(osoby):
    for i, osoba in enumerate(osoby):
        osoby[i] = f"Pani {osoba}" if osoba[-1] == "a"
                                   else f"Pan {osoba}"

osoby = ["Jan","Kasia","Piotr","Magda"]
print(osoby)
pan_pani(osoby[:])
print(osoby)
```

```
['Jan', 'Kasia', 'Piotr', 'Magda']
['Jan', 'Kasia', 'Piotr', 'Magda']
```

- należy to robić tylko w uzasadnionych przypadkach, gdyż tworzenie kopii zajmuje czas i pamięć

Python - funkcje (przekazywanie wartości)

- do funkcji można przekazać dowolną liczbę argumentów

```
def powitanie(*osoby):
    for osoba in osoby:
        print(f"Witaj, {osoba}!")

powitanie("Anna")
powitanie("Marian", "Paweł")
```

```
Witaj, Anna!
Witaj, Marian!
Witaj, Paweł!
```

- gwiazdka w nazwie parametru powoduje utworzenie krotki o nazwie osoby i umieszczenie w niej otrzymanych wartości
- krotka jest tworzona także wtedy, gdy podany jest jeden argument
- dowolna liczba argumentów często nazywana jest *args

Python - funkcje (przekazywanie wartości)

- jeśli funkcja ma parametry różnego rodzaju, to parametr przyjmujący wiele wartości musi znajdować się na końcu
- Python najpierw dopasowuje argumenty pozycyjne oraz argumenty w postaci słów kluczowych, a dopiero później zbiera pozostałe argumenty

```
def powitanie(tekst, *osoby):
    for osoba in osoby:
        print(f"{tekst}, {osoba}!")

powitanie("Witaj", "Anna")
powitanie("Hello", "Marian", "Paweł")
```

```
Witaj, Anna!
Hello, Marian!
Hello, Paweł!
```

Python - funkcje (przekazywanie wartości)

- jako dowolną liczbę argumentów można podać słowa kluczowe

```
def telefon(marka, model, **dane):
    dane["marka"] = marka
    dane["model"] = model
    return dane

Samsung = telefon("Samsung", "Galaxy", ekran="AMOLED")
print(Samsung)
```

```
{'ekran': 'AMOLED', 'marka': 'Samsung', 'model': 'Galaxy'}
```

- dwie gwiazdki przed nazwą parametru powodują utworzenie pustego słownika o nazwie dane oraz umieszczenie w nim wszystkich otrzymanych par klucz-wartość
- dowolna liczba argumentów w postaci par klucz-wartość często nazywana jest *kwargs

Python - funkcje (moduły)

- napisane funkcje można umieścić w oddzielnym pliku (module), a następnie importować ten moduł do programu, w którym mają być one wywołane
- moduł jest plikiem z rozszerzeniem .py zawierającym kod, który będzie importowany

obwod.py

```
def kwadrat(a):
    wynik = 4 * a
    return wynik

def prostokat(a,b):
    wynik = 2 * (a + b)
    return wynik
```

myapp.py

```
import obwod

a = float(input("Bok kwadratu: "))
obw = obwod.kwadrat(a)
print(f"Obwód wynosi: {obw}")
```

```
Bok kwadratu: 3
Obwód wynosi: 12.0
```

Python - funkcje (moduły)

- polecenie import obwod powoduje import całego modułu czyli skopiowanie wszystkich funkcji z pliku obwod.py do bieżącego programu

myapp.py

```
import obwod

a = float(input("Bok kwadratu: "))
obw = obwod.kwadrat(a)
print(f"Obwód wynosi: {obw}")
```

- wywołując funkcję podajemy nazwę modułu, kropkę, nazwę funkcji i jej argumenty w nawiasach zwykłych

```
nazwa_modułu.nazwa_funkcji(argumenty)
```

Python - funkcje (moduły)

- można zaimportować tylko wybrane funkcje z modułu

```
from nazwa_modułu import nazwa_funkcji
```

lub

```
from nazwa_modułu import nazwa1, nazwa2, nazwa3, ...
```

- w takim przypadku przy wywołaniu funkcji nie trzeba używać notacji z kropką

myapp.py

```
from obwod import kwadrat

a = float(input("Bok kwadratu: "))
obw = kwadrat(a)
print(f"Obwód wynosi: {obw}")
```

Python - funkcje (moduły)

- podczas importowania funkcji z modułu można nadać jej nową nazwę (alias)

```
from nazwa_modułu import stara_nazwa as nowa_nazwa
```

Przykład:

myapp.py

```
from obwod import kwadrat as kw

a = float(input("Bok kwadratu: "))
obw = kw(a)
print(f"Obwód wynosi: {obw}")
```

- stosuje się to wtedy, gdy nazwa importowanej funkcji koliduje z nazwą funkcji już istniejącej w programie lub też nazwa importowana jest zbyt długa

Python - funkcje (moduły)

- słowo kluczowe as może być zastosowane także do zdefiniowania aliasu dla modułu

```
import stara_nazwa_modułu as nowa_nazwa_modułu
```

Przykład:

myapp.py

```
import obwod as o

a = float(input("Bok kwadratu: "))
obw = o.kwadrat(a)
print(f"Obwód wynosi: {obw}")
```

Python - funkcje (moduły)

- za pomocą operatora * można zaimportować wszystkie funkcje znajdujące się w module

```
from nazwa_modułu import *
```

- dzięki temu można wywołać funkcje bez nazwy modułu i kropki

myapp.py

```
from obwod import *

a = float(input("Bok kwadratu: "))
obw = kwadrat(a)
print(f"Obwód wynosi: {obw}")
```

Python - funkcje (zalecenia)

- nazwa funkcji powinna jasno określać jej przeznaczenie
- nazwa funkcji powinna składać się z małych liter i podkreślenia
- w kodzie funkcji powinien znajdować się komentarz wyjaśniający przeznaczenie funkcji, umieszczony po definicji funkcji, zgodnie z formatem docstring
- jeśli parametr ma wartość domyślną, to po obu stronach znaku równości nie umieszczamy spacji
- jeśli parametr jest zapisany w postaci klucz-wartość, to po obu stronach znaku równości nie umieszczamy spacji

Python - funkcje (docstringi)

- docstring jest ciągiem znaków służącym do dokumentowania kodu
- docstring umieszcza się za definicją funkcji, pomiędzy trzema znakami cudzysłowu

```
def dodaj(a, b):
    """
    Funkcja dodająca dwie liczby

    Parametry
    a (float): pierwsza liczba
    b (float): druga liczba

    Zwraca:
    float: suma dwóch liczb
    """
    suma = a + b
    return suma
```

docstring zawiera:

- opis funkcji
- opis parametrów (nazwa, typ, opis)
- opis zwracanej wartości
- informacje o wyjątkach (jeśli są obsługiwane)
- przykład użycia funkcji

Koniec wykładu nr 8

Dziękuję za uwagę!