Politechnika Wrocławska

Programowanie Procduralne
Wykład #2(W#09) — Fizyka

Janusz Andrzejewski

Wstęp

- Parametry a argumenty funkcji
- Przekazywanie przez wartość a przez zmienną
- Argumenty mutowalne a niemutowalne
  - Pułapki
  - Uwagi
- Zmienne lokalne a nielokalne
- Funkcje `locals()` oraz `globals()`
- Komentarze dokumentacyjne

Plan czyli czego się będziemy uczyli?

- W#1 – Podsumowanie
- – Typy mutowalne oraz niemutowalne a przekazywanie argumentów do funkcji.
- – Zasięg zmiennych — zmienne lokalne, ~~nielokalne~~ oraz globalne.
- – Argumenty funkcji: pozycyjne, nazwane, domyślne, rozpakowywanie, słownikowe
- – Funkcje: „first citizen object”.
- – Funkcje: funkcje wyższego rzędu.
- – Funkcje anonimowe.
- – Dopełnienia funkcji.
- – Zastosowania funkcji.
- Laboratoria: od teorii do praktyki czyli rozwiązywanie problemów związanych z omówionymi zagadnieniami na wykładzie.

Argumenty a parametry funkcji

- Parametry funkcji – to **zmienne** które są zdefiniowane lub użyte wewnątrz nawiasów nagłówka funkcji przy definiowaniu tejże funkcji
- Argumenty funkcji – to **wartości** które są użyte wewnątrz nawiasów przy wywołaniu funkcji. Wartości te mogą być przekazane jako:
  - Zmienne
  - Wyrażenia
  - Stałe nazwane

Argumenty a parametry funkcji

```
1  def dane(imie, nazwisko): #<-- nagłówek funkcji
2              #imie oraz nazwisko - parametry funkcji
3      print("Witam Cię ", imie, nazwisko)
4
5  Imie="Janusz"
6  dane(Imie, "Andrzejewski")
7      # Imie oraz "Andrzejwski" - argumenty funkcji
```

```
Witam Cię  Janusz Andrzejewski
```

Funkcja (przypomnienie):

- Odrębny i nazwany fragment kodu
- Może mieć dowolną liczbę parametrów
- Każdy parametr może być dowolnego typu
- Może zwracać dowolną liczbę wyników dowolnych typów

Parametry funkcji

- Funkcja zawsze dowolnie może zmieniać (wartość czy typ) swoich parametrów

```
1  def funk(x):
2      x=3 # par -> int
3      print("1 par: ", x)
4      x=[1, 2] #par -> lista
5      print("2 par: ", x)
6
7  x="Janusz"
8  funk(x)    #arg -> zmienna
9  funk(24+3.0/2.0) #arg -> wyrażenie
```

```
1 par:  3
2 par:  [1, 2]
1 par:  3
2 par:  [1, 2]
```

Przekazywanie parametrów do funkcji

- Python przy przesyłaniu parametrów do funkcji stosuje tzw. **przekazanie-przez-obiekt** albo **przekazenie-przez-referencje-do-obiektu**
- Parametr funkcji jest referencją do obiektu, referencja ta przekazana jest przez **wartość**

Przez wartość a przez zmienną:

- Przekazanie przez wartość:
  - Do funkcji przekazywana jest kopia
  - Wszystkie operacje w funkcji robione są na kopii
  - Po wyjściu z funkcji wszelkie zmiany są utracone
- Przekazywanie przez zmienną
  - Do funkcji przekazany jest oryginał – poprzez referencje lub wskaźnik na oryginał
  - Wszystkie operacje wykonywane są na oryginale
  - Po wyjściu z funkcji zmiany będą widoczne

Wniosek (Python):

- Jeśli w funkcji, zmieniając zawartość obiektu jednocześnie nie zmienimy referencji tego obiektu – wówczas zmiana dokonana w obiekcie będzie widoczna po wyjściu z funkcji
- Po wyjściu z funkcji, dla parametrów typu:
  - Obiekt niemutowalny – funkcja nie może zmienić zwartości obiektu – ale … :)
  - Obiekt mutowalny – funkcja może zmieniać zawartość obiektu pod warunkiem że nie zmienia referencji do obiektu

Argument — obiekt niemutowalny(1)

```
1  def funk(x):
2      x=3
3      print(" !!W fnk, par. po zmianie="
4
5  x=0
6  print("Przed wywołaniem x =",x)
7  funk(x)
8  print("    Po wywołaniu x =",x)
9  x=(1, 2)
10 print("Przed wywołaniem x =",x)
11 funk(x)
12 print("    Po wywołaniu x =",x)
```

```
Przed wywołaniem x = 0
 !!W fnk, par. po zmianie= 3
    Po wywołaniu x = 0
Przed wywołaniem x = (1, 2)
 !!W fnk, par. po zmianie= 3
    Po wywołaniu x = (1, 2)
```

```
1  def funk(x):
2      x*=2
3      print(" !! W fnk, par. po zmianie=",x)
4
5  x=0
6  print("Przed wywołaniem x =",x)
7  funk(x)
8  print("    Po wywołaniu x =",x)
9  x=(1, 2)
10 print("Przed wywołaniem x =",x)
11 funk(x)
12 print("    Po wywołaniu x =",x)
```

```
Przed wywołaniem x = 0
 !! W fnk, par. po zmianie= 0
    Po wywołaniu x = 0
Przed wywołaniem x = (1, 2)
 !! W fnk, par. po zmianie= (1, 2, 1, 2)
    Po wywołaniu x = (1, 2)
```

Argument — obiekt mutowalny(1)

```
1  def funkL1(x):
2      x[0]=-2
3      x[-1]="Jan"
4
5  x=[1, 2, 3]
6  print("Przed wywołaniem x =",x)
7  funkL1(x)
8  print("    Po wywołaniu x =",x)
```

```
Przed wywołaniem x = [1, 2, 3]
    Po wywołaniu x = [-2, 2, 'Jan']
```

```
1  def funkL2(x):
2      x.append("Jan")
3
4  x=[1, 2, 3]
5  print("Przed wywołaniem x =",x)
6  funkL2(x)
7  print("    Po wywołaniu x =",x)
```

```
Przed wywołaniem x = [1, 2, 3]
    Po wywołaniu x = [1, 2, 3, 'Jan']
```

Jeśli NIE ZMIENIAMY referencji do obiektu wewnątrz funkcji, to jeśli zmienimy zawartość obiektu (wewnątrz funkjci) po wyjściu z funkcji wszystkie zmiany zostaną ZACHOWANE.

Argument — obiekt mutowalny(2a)

```
1  def funkL3a(x):
2      x*=2 #ref do obiektu x
3           #sie NIEzmienia
4
5  x=[1, 2, 3]
6  print("Przed wywołaniem x =",x)
7  funkL3a(x)
8  print("    Po wywołaniu x =",x)
```

```
Przed wywołaniem x = [1, 2, 3]
    Po wywołaniu x = [1, 2, 3, 1, 2, 3]
```

```
1  def funkL3b(x):
2      x=x+x #ref do obiektu x
3            #sie ZMIENIA
4
5  x=[1, 2, 3]
6  print("Przed wywołaniem x =",x)
7  funkL3b(x)
8  print("    Po wywołaniu x =",x)
```

```
Przed wywołaniem x = [1, 2, 3]
    Po wywołaniu x = [1, 2, 3]
```

Jeśli referencji do obiektu wewnątrz funkcji się ZMIENI, to jeśli zmienimy zawartość obiektu (wewnątrz funkcji) po wyjściu z funkcji wszystkie zmiany zostaną UTRACONE.

Argument — obiekt mutowalny(2b) — dokładniej

```
1  def funkL3b(x):
2      print("id par przed: ",id(x))
3      x=x+x #ref do obiektu x
4             #sie ZMIENIA
5      print("   id par po: ",id(x))
6
7  x=[1, 2, 3]
8  print("id arg przed ", id(x))
9  print("Przed wywołaniem x =",x)
10 funkL3b(x)
11 print("    Po wywołaniu x =",x)
12 print("   id arg po: ", id(x))
```

```
id arg przed  139994147487808
Przed wywołaniem x = [1, 2, 3]
id par przed:  139994147487808
   id par po:  139994147491264
    Po wywołaniu x = [1, 2, 3]
   id arg po:  139994147487808
```

Argument — obiekt mutowalny(3)

```
1  def funkL4(x):
2      x=[2, 3] #nowy lokalny obiekt
3               #inna ref
4      x.append("3")
5
6  x=[1, 2, 3]
7  print("Przed wywołaniem x =",x)
8  funkL4(x)
9  print("    Po wywołaniu x =",x)
```

```
Przed wywołaniem x = [1, 2, 3]
    Po wywołaniu x = [1, 2, 3]
```

Argument — obiekt niemutowalny(2)

```
1  def funkK(x):
2      x[1].append("Coś")
3
4  x=(1, [1, 2], "Nic") #arg niemutowalny
5  print("Przed wywołaniem x =",x)
6  funkK(x)
7  print("    Po wywołaniu x =",x)
```

```
Przed wywołaniem x = (1, [1, 2], 'Nic')
    Po wywołaniu x = (1, [1, 2, 'Coś'], 'Nic')
```

Jeśli parametrem jest obiekt niemutowalny który zawiera obiekt mutowalny, to jest możliwość zachowania zmiany po wyjściu z funkcji

Argument — obiekt niemutowalny(3a)

```
1  def funkK1(x):
2      x[1]*=2
3
4  x=(1, [1, 2], "Nic") #arg niemutowalny
5  print("Przed wywołaniem x =",x)
6  funkK1(x)
7  print("    Po wywołaniu x =",x)
```

```
Przed wywołaniem x = (1, [1, 2], 'Nic')
---------------------------------------------------------------
TypeError                                 Traceback (most r
Cell In[67], line 6
      4 x=(1, [1, 2], "Nic") #arg niemutowalny
      5 print("Przed wywołaniem x =",x)
----> 6 funkK1(x)
      7 print("    Po wywołaniu x =",x)

Cell In[67], line 2, in funkK1(x)
      1 def funkK1(x):
----> 2     x[1]*=2

TypeError: 'tuple' object does not support item assignment
```

Argument — obiekt niemutowalny(3b)

```
1  def funkK2(x):
2      K=x[1]
3      K*=2
4
5  x=(1, [1, 2], "Nic") #arg niemutowalny
6  print("Przed wywołaniem x =",x)
7  funkK2(x)
8  print("    Po wywołaniu x =",x)
```

```
Przed wywołaniem x = (1, [1, 2], 'Nic')
    Po wywołaniu x = (1, [1, 2, 1, 2], 'Nic')
```

Pamiętaj:

- Znak podstawienia (=) — nie kopiuje obiektu, kopiuje tylko referencje, oznacza to ze mamy ten sam obiekt który widnieje pod różnymi nazwami
- Sposób który jest tutaj zaprezentowany powinniśmy stosować ostrożnie

Argument - bezpieczeństwo

```python
def funkK3(x):
    K=x[1][:]
    K*=2
    print(" w fun:",K)

x=(1, [1, 2], "Nic") #krotka
print("Przed wywołaniem x =",x)
funkK3(x)
print("    Po wywołaniu x =",x)
```

```
Przed wywołaniem x = (1, [1, 2], 'Nic')
 w fun: [1, 2, 1, 2]
    Po wywołaniu x = (1, [1, 2], 'Nic')
```

```python
import copy
def funkK4(x):
    K=copy.deepcopy(x[1])
    K*=2
    print(" w fun:",K)

x=(1, [1, 2], "Nic") #krotka
print("Przed wywołaniem x =",x)
funkK4(x)
print("    Po wywołaniu x =",x)
```

```
Przed wywołaniem x = (1, [1, 2], 'Nic')
 w fun: [1, 2, 1, 2]
    Po wywołaniu x = (1, [1, 2], 'Nic')
```

Pułapki

```python
def funkZM(x, w):
    """.Informacje DO oraz Z funkcji
        tylko poprzez parametr x"""
    for i in range(len(x)):
        x[i]+=w

def funkWA(x, w):
    """Informacje DO funkcji tylko
        poprzez parametr x,
        informacja Z tylko poprzez
        wartosc funkcji - zmienna x"""
    for i in range(len(x)):
        x[i]=x[i]+w
    return x

```

```python
x=[1, 2, 3]
y=x[:] #kopia x

#jak możemy dodac 2 do każdego el. listy
#1 sposob
funkZM(x, 2)
2#sposób
z=funkWA(y, 2)
print("1 sp: ",x)
print("2 sp: ",z)
```

```
1 sp:  [3, 4, 5]
2 sp:  [3, 4, 5]
```

```python
#Ale czy z i y to różne obiekty
print(id(y), id(z))
print("Czy y i z te same obiekty?",
      id(y)==id(z))
z[0]=-2 #zmieniamy z, a co z y?
print(y[0], z[0])
```

```
140323269152448 140323269152448
Czy y i z te same obiekty? True
-2 -2
```

Kiedy przekazywać przez wartość?

```
1  def funkZM(x, w):
2      """.Informacje DO oraz Z funkcji
3          tylko poprzez parametr x"""
4      for i in range(len(x)):
5          x[i]+=w
6
7  def funkWA2(x, w):
8      """Prawdziwa inna wartość"""
9      y=[]
10     for i in range(len(x)):
11         y.append(x[i]+w)
12     return y
13
14 #Bardzo duza lista :)
15 DuzaLiczba=10 #:)
16 x=[x for x in range(DuzaLiczba)]
17 y=[x for x in range(DuzaLiczba)]
18 #jak można zwiekszysz wartość elementów
19 #listy o 3??
```

```
1  #1 sposób
2  funkZM(x, 3)
3  print("Lista x",x)
```

```
Lista x [3, 4, 5, 6, 7, 8, 9, 10, 11, 12]
```

```
1  #2 sposob
2  z=funkWA2(y, 3)
3  print("Czy listy x i z sa takie same?",
4        x == z)
```

```
Czy listy x i z sa takie same? True
```

```
1  #3 spsoob
2  y=funkWA2(y, 3)
3  print("Czy listy x i y sa takie same?",
4        x == y)
```

```
Czy listy x i y sa takie same? True
```

Który z tych programów działa? W którym miejscu jest błąd?

```
1  zmM=1
2  def funk1(x):
3      print("zmM:",zmM)
4      print("  x:",x)
5
6
7
8  funk1(2)
```

```
zmM=1
def funk2(x):
    zmM=zmM+x
    print("zmM:",zmM)
    print("  x:",x)


funk2(2)
```

```
zmM=1
def funk3(x):
    print("zmM:",zmM)
    zmM=zmM+x
    print("zmM:",zmM)
    print("  x:",x)

funk3(2)
```

```
zmM=1
def funk4(x):
    x=zmM+x
    print("zmM:",zmM)
    print("  x:",x)


funk4(2)
```

```
      2 def funk2(x):
----> 3     zmM=zmM+x
      4     print("zmM:",zmM)
      5     print("  x:",x)

UnboundLocalError: cannot access local variable 'zmM' where it is not associated with a value
```

```
      2 def funk3(x):
----> 3     print("zmM:",zmM)
      4     zmM=zmM+x
      5     print("zmM:",zmM)

UnboundLocalError: cannot access local variable 'zmM' where it is not associated with a value
```

Lokalne i globalne zmienne (1)

- Przestrzeń nazw – sekcja kodu, w których każda nazwa jest unikatowa i niezwiązana z takimi samymi nazwami znajdującymi się w innych przestrzeniach.
- W zależności od tego gdzie występuje w kodzie zmienna o tej samej nazwie może ona oznaczać co innego
- Globalna przestrzeń nazw – główny segment programu
- Zmienne globalne – są to zmienne zdefiniowane w głównym segmencie programu
- Lokalna przestrzeń nazw – każda funkcji ma swoją lokalną przestrzeń
- Zmienne lokalne – zmienne zdefiniowane w funkcji
- Jest możliwość „przenikania” zmiennej globalnej do lokalnej przestrzeni nazw

Lokalne i globalne zmienne (2)

```
1  x=22 #zmienna globalna
2
3  def funk(a, b):
4      x=0 #x - zmienna lokalna
5      y=x #y - zmienna lokalna
6      for i in range(a, b):
7          y+=i
8      return y
9
10 y=funk(2, 10)
11 print("y = ",y)
12 print("x = ",x)
```

```
y =  44
x =  22
```

```
1  x=22 #zmienna globalna
2
3  def funk(a, b):
4      y=0 #y - zmienna lokalna
5      y=x #domyślnie, zmienna
6          #globalna x jest widoczna w
7          #lokalnej przestrzeni nazw, można
8          #jej uzywać tylko w wyrażeniach
9      for i in range(a, b):
10         y+=i
11     return y
12
13 y=funk(2, 10)
14 print("y = ",y)
15 print("x = ",x)
```

```
y =  66
x =  22
```

Pojawienie się zmiennej lokalnej automatycznie przesłania dostęp do zmiennej globalnej o takiej samej nazwie, chyba że ...

Lokalne i globalne zmienne (3)

```
1  x=22 #zmienna globalna
2
3  def funk(a, b):
4      global x #pełen dostęp do
5         #zmiennej globalnej x
6         #w lokal. przest. nazw
7      x=2*x #zmiana wartosci
8      y=x #y - zmienna lokalna
9      for i in range(a, b):
10         y+=i
11     return y
12
13 y=funk(2, 10)
14 print("y = ",y)
15 print("x = ",x)
```

```
y =  88
x =  44
```

Instrukcja `global` powoduje to, że zmienna globalna jest widoczna w lokalnej przestrzeni nazw.

`locals()` oraz `globals()` – (1)

- Do zawartości przestrzeni nazw można odwoływać się za pomocą dwóch funkcji:
- `locals()` – zwracającej słownik z zawartością lokalnej przestrzeni
- `globals()` – zwracającej słownik z zawartością globalnej przestrzeni

locals() oraz globals() – (2)

```python
x=22 #zmienna globalna

def funk(a, b):
    print("Domyślny zbiór zmiennych lokalnych ", locals())
    global x
    x=2*x #zmiana wartosci
    y=x #y - zmienna lokalna
    print("Aktualny znior zmiennycj lokalnych ",locals())
    for i in range(a, b):
        y+=i
    return y

y=funk(2, 10)
print("y = ",y)
print("x = ",x)
print(globals()) #funkcja wyświetlająca duzo informacji
```

```
Domyślny zbiór zmiennych lokalnych  {'a': 2, 'b': 10}
Aktualny znior zmiennycj lokalnych  {'a': 2, 'b': 10, 'y': 44}
y =  88
x =  44
{'__name__': '__main__', '__doc__': 'Automatically created modul
None, '__spec__': None, '__builtin__': <module 'builtins' (built
```

locals() oraz globals() – (3)

```
1 help(globals)
```

```
Help on built-in function globals in module builtins:

globals()
    Return the dictionary containing the current scope's global variables.

    NOTE: Updates to this dictionary *will* affect name lookups in the current
    global scope and vice-versa.
```

```
1 help(locals)
```

```
Help on built-in function locals in module builtins:

locals()
    Return a dictionary containing the current scope's local variables.

    NOTE: Whether or not updates to this dictionary will affect name lookups in
    the local scope and vice-versa is *implementation dependent* and not
    covered by any backwards compatibility guarantees.
```

locals() oraz globals() – (4)

```
1  sl=globals()
2  x=2
3  print("zm x:",x)
4  sl['y']=3 #inny sposób na: y=3
5      #to działa tylko dla globalnej
6      #przestrzeni nazw
7  print("zm y:",y)
```

```
zm x: 2
zm y: 3
```

```
1  def funk(a, b):
2      sl=locals()
3      sl['x']=0 #próba def zm lok
4      print("loc x:",x)
5      x=a+b
6      return x
7
8  y=funk(2, 3)
9  print("y:",y)
```

```
Cell In[7], line 4, in funk(a, b)
      2 sl=locals()
      3 sl['x']=0 #próba def zm lok
----> 4 print("loc x:",x)
      5 x=a+b
      6 return x

UnboundLocalError: cannot access local variable 'x' where it is not associated with a value
```

Uwagi:

- W czasie tego semestru **NIE WOLNO** posługiwać się zmiennymi globalnymi (ani na laboratorium ani podczas sprawdzianów)
- Funkcje `globals()` oraz `locals()` — proszę traktować jako ciekawostkę

Komentarze dokumentacyjne (1)

- Ciąg znaków umieszczony na początku funkcji zaraz po jej nagłówku
- Może być bardzo rozbudowany
- Jest dostępny od razu poprzez system pomocy – help(nazwa_funkcji)
- Może być wykorzystywany do generowania dokumentacji poprzez inne zewnętrzne programy

Komentarze dokumentacyjne (2)

```
1  def funkJ(a, b):
2      """Funkcja obliczjaąca sumę liczb od a do b
3
4      Funkcja oblicza sumę liczba od a - liczba całkowita
5      do b - dowolna liczba.Type wyniku jest zgodny z typem
6      argumentu b przekazanego do funkcji.
7      """
8      if type(b) == int:
9          s=0
10     else:
11         s=0.0
12     for i in range(a, int(b)):
13         s+=i
14     return s
15
16 print("   INT: ",funkJ(2, 5))
17 print("FLOCAT: ",funkJ(2, 5.3))
```

```
1 help(funkJ)
```

```
Help on function funkJ in module __main__:

funkJ(a, b)
    Funkcja obliczjaąca sumę liczb od a do b

    Funkcja oblicza sumę liczba od a - liczba całkowita
    do b - dowolna liczba.Type wyniku jest zgodny z typem
    argumentu b przekazanego do funkcji.
```

```
1 print("Funkcja tez jest obiektem ",funkJ.__doc__)
```

```
Funkcja tez jest obiektem  Funkcja obliczjaąca sumę liczb od a do b

    Funkcja oblicza sumę liczba od a - liczba całkowita
    do b - dowolna liczba.Type wyniku jest zgodny z typem
    argumentu b przekazanego do funkcji.
```

Dziękuję za uwagę