Akademia Pythona

IV Funkcje

KN Pythona - Kurs Pythona

KN Pythona wita na kursie Pythona.

Plan:

- Podstawy funkcji
- Zakresy
- Argumenty

Podstawy funkcji

Funkcja jest narzędziem grupującym zbiór instrukcji w taki sposób, by mogły być one wykonane w programie więcej niż jeden raz. Funkcje obliczają wartość wyniku i pozwalają nam określić parametry służące za dane wejściowe.

Instrukcje oraz wyrażenia powiązane z funkcjami

Instrukcja	Przykłady
Wywołania	myfunc(‘a’, ‘b’, ‘c’)
def, return	def add(a, b):return a + b
global	def changer(): global x; x = ‘new’
nonlocal	def changer(): nonlocal x; x = ‘new’
yield	def squares(x): for i in x: yield i ** 2
lambda	funcs = [lambda x: x**2, lambda x: x**3

Po co używa się funkcji?

Po co używa się funkcji:

- Maksymalizacja ponownego wykorzystania kodu i minimalizacja jego powtarzalności
- Proceduralne podzielenie na części

Tworzenie funkcji

def to kod wykonywalny.

Tworzenie funkcji

Instrukcja **def** tworzy obiekt i przypisuje go do nazwy.

Tworzenie funkcji

Wyrażenie **lambda** tworzy obiekt i zwraca go jako wynik.

Tworzenie funkcji

Instrukcja **return** przesyła wynikowy obiekt z powrotem do wywołującego.

Tworzenie funkcji

Instrukcja **yield** odsyła wynikowy obiekt z powrotem do wywołującego, jednak zapamiętuje, gdzie zakończyła działanie.

Tworzenie funkcji

Instrukcja **global** deklaruje zmienne, które mają być przypisane na poziomie modułu.

Tworzenie funkcji

Instrukcja **nonlocal** deklaruje zmienne z funkcji zawierającej, które mają być przypisane.

Tworzenie funkcji

Argumenty przekazywane są przez przypisanie (referencję obiektu).

Tworzenie funkcji

Argumenty, zwracane wartości i zmienne nie są deklarowane.

Instrukcje def

Ogólny format

```
def <nazwa>(arg1, arg2, ..., argN):
    <instrukcje>
```

Instrukcje def

```
def <nazwa>(arg1, arg2, ..., argN):
    ...
    return <wartość>
```

Instrukcja def uruchamiana jest w trakcie wykonywania

```
if test:
    def func():
        <instrukcje1>
else:
    def func():
        <instrukcje2>
func() # ?
```

Funkcje są obiektami

```
othername = func # Przypisanie obiektu funkcji
othername()

func.attr = value # Dołączenie atrybutów
```

Przykład

```
def times(x, y):
    return x * y

times(2, 4) # 8
times(3.14, 4) # 12.56
times('xd', 3) # 'xdxdxd'
```

Przykład

```
def intersect(seq1, seq2):
    res = []
    for x in seq1:
        if x in seq2:
            res.append(x)
    return res
```

Przykład

```
intersect('abcd', 'cdef') # ['c', 'd']
intersect([1, 2, 3], (1, 4)] # [1]
```

Zakresy

Zakresy Pythona - miejsca, w których zmienna jest przypisywana i wyszukiwana.

Zakresy

Zakresy:

- Nazwy zdefiniowane wewnątrz instrukcji **def** mogą być widoczne jedynie dla kodu wewnątrz tej funkcji.
- Nazwy zdefiniowane wewnątrz instrukcji **def** nie wchodzą w konflikt ze zmiennymi spoza tej instrukcji, nawet jeżeli tak samo się nazywają.

Zakresy

Zakresy:

- Jeżeli zmienna zostanie przypisana wewnątrz instrukcji **def**, staje się zmienną lokalną dla tej funkcji.
- Jeżeli zmienna przypisana jest wewnątrz instrukcji **def** zawierającej inną funkcję, staje się ona zmienną nielokalną dla tej funkcji.
- Jeżeli zmienna przypisana jest poza wszystkimi instrukcjami **def**, staje się ona zmienną globalną dla całego pliku.

Zakresy

```
x = 99

def func():
    x = 87

print(x) # 99
```

Zakresy

Reguły dotyczące zakresów:

- Moduł zawierający funkcję jest zakresem globalnym.
- Zakres globalny rozciąga się jedynie na jeden plik.
- Każde wywołanie funkcji tworzy nowy zakres lokalny.
- Przypisane nazwy są lokalne, o ile nie zostaną zadeklarowane jako globalne lub nielokalne.
- Wszystkie pozostałe nazwy są lokalne dla zakresu zawierającego, globalne lub wbudowane.

Zakresy

Reguła LEGB:

- Local
- Enclosing
- Global
- Built-in

Zakresy

Rozwiązywanie konfliktów w zakresie nazw - reguła LEGB:

- Referencje do nazw przeszukują cztery zakresy: lokalny, zawierający, globalny i wbudowany.
- Przypisania nazw domyślnie tworzą lub modyfikują nazwy lokalne.
- Deklaracje global i nonlocal odwzorowują przypisane nazwy na zakres modułu zawierającego oraz funkcji zawierającej.

Zakresy

```
# Zakres globalny
x = 99

def func(y):
    # Zakres lokalny
    z = x + y
    return z

func(1) # 100
```

Zakresy

```
import builtins
dir(builtins)
```

Zakresy

```
x = 88

def func():
    global x
    x = 99

func()
print(x) # 99
```

Zakresy

```
x = 99
def f1():
    x = 88
    def f2():
        print(x)
    f2()

f1() # 88
```

Funkcje fabryczne

```
def maker(n):
    def action(x):
        return x ** n
    return action

f = maker(2)
f(2) # 4
f(3) # 9
f = maker(3)
f(2) # 9
```

Zakresy

```
def make_actions():
    acts = []
    for i in range(5):
        acts.append(lambda x, i=i: i ** x)
    return acts

a = make_actions()
a[0](3) # 0
a[2](2) # 4
```

Zakresy

```
def tester(start):
    state = start
    def nested(label):
        nonlocal state
        print(label, state)
        state += 1
    return nested

f = tester(0)
f('a') # a 0
f('b') # b 1
```

Zakresy

Zmienne nielokalne muszą istnieć w zakresie zawierającym. Zmienne globalne nie muszą istnieć przy ich deklarowaniu.

Argumenty

Podstawy przekazywania argumentów:

- Argumenty przekazywane są automatyczne przypisanie obiektów do nazw zmiennych lokalnych.
- Przypisania do nazw argumentów wewnątrz funkcji nie wpływa na wywołującego.
- Modyfikacja zmiennego obiektu argumentu w funkcji może mieć wpływ na wywołującego.

Argumenty

```
def f(a):
    a = 99

b = 88
f(b)
print(b) # 88
```

Specjalne tryby dopasowania argumentów

Tryby dopasowania:

- Pozycyjne: dopasowanie od lewej do prawej strony
- Słowa kluczowe: dopasowanie po nazwie argumentu
- Wartości domyślne: określenie wartości dla argumentów, które nie zostały przekazane
- Nieznana liczba argumentów (zbieranie): zebranie dowolnej liczby argumentów zgodnie z pozycją lub słowem kluczowym
- Argumenty mogące być tylko słowami kluczowymi: argumenty, które muszą być przekazywane przez nazwę

Składnia dopasowania

Składnia	Interpretacja
func(wartosc)	Normalny argument - dopasowanie po pozycji
func(nazwa=wartosc)	Słowo kluczowe - dopasowani po nazwie
func(*sekwencja)	Przekazanie sekwencji jako argumenty pozycyjne
func(**słownik)	Przekazanie par klucz-wartość

Składnia dopasowania

Składnia	Interpretacja
def func(nazwa)	Normalny argument
def func(nazwa=wartosc)	Domyślna wartość argumentu
def func(*nazwa)	Dopasowanie sekwencji
def func(**nazwa)	Dopasowanie słownika
def func(*args, nazwa)	Argument wymuszony jako słowo kluczowe
def func(*, nazwa)	Jedynie wymuszone argumenty kluczowe

Składnia i dopasowania

```
def func(arg0, /, arg1, *args, **kwargs, kwonly):
    pass
```