Python Funkcje

Python Funkcje

Agenda

1. **Funkcja w Pythonie**
2. Definiowanie funkcji
3. Parametry/argumenty funkcji
4. Rekurencja
5. Przestrzeń nazw
6. Funkcja anonimowe lambda
7. Funkcje wbudowane
8. Iteratory range/enumerate

Python Funkcje

Agenda

1. Funkcja w Pythonie
2. **Definiowanie funkcji**
3. Parametry/argumenty funkcji
4. Rekurencja
5. Przestrzeń nazw
6. Funkcja anonimowe lambda
7. Funkcje wbudowane
8. Iteratory range/enumerate

Python Funkcje

Agenda

1. Funkcja w Pythonie
2. Definiowanie funkcji
3. **Parametry/argumenty funkcji**
4. Rekurencja
5. Przestrzeń nazw
6. Funkcja anonimowe lambda
7. Funkcje wbudowane
8. Iteratory range/enumerate

infoShare ACADEMY

Python Funkcje

Agenda

1. Funkcja w Pythonie
2. Definiowanie funkcji
3. Parametry/argumenty funkcji
4. **Rekurencja**
5. Przestrzeń nazw
6. Funkcja anonimowe lambda
7. Funkcje wbudowane
8. Iteratory range/enumerate

infoShare ACADEMY

Python Funkcje

Agenda

1. Funkcja w Pythonie
2. Definiowanie funkcji
3. Parametry/argumenty funkcji
4. Rekurencja
5. **Przestrzeń nazw**
6. Funkcja anonimowe lambda
7. Funkcje wbudowane
8. Iteratory range/enumerate

Python Funkcje

Agenda

1. Funkcja w Pythonie
2. Definiowanie funkcji
3. Parametry/argumenty funkcji
4. Rekurencja
5. Przestrzeń nazw
6. **Funkcja anonimowe lambda**
7. Funkcje wbudowane
8. Iteratory range/enumerate

Python Funkcje

Agenda

1. Funkcja w Pythonie
2. Definiowanie funkcji
3. Parametry/argumenty funkcji
4. Rekurencja
5. Przestrzeń nazw
6. Funkcja anonimowe lambda
7. **Funkcje wbudowane**
8. Iteratory range/enumerate

Python Funkcje

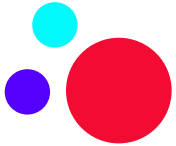
Agenda

1. Funkcja w Pythonie
2. Definiowanie funkcji
3. Parametry/argumenty funkcji
4. Rekurencja
5. Przestrzeń nazw
6. Funkcja anonimowe lambda
7. Funkcje wbudowane
8. **Iteratory range/enumerate**

Python Funkcje

Po co funkcje?

```
print("Wstań z łóżka")
print("Umyj zęby")
print("Zjedz śniadanie")
print("Pracuj")
print("Zjedz obiad")
print("Graj w gry")
print("Zjedz kolacje")
print("Idź spać")
print("Wstań z łóżka")
print("Umyj zęby")
print("Zjedz śniadanie")
print("Pracuj")
print("Zjedz obiad")
print("Graj w gry")
print("Zjedz kolacje")
print("Idź spać")
.
.
```



Python Funkcje

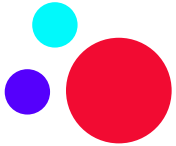
Funkcja (programowanie)

Funkcja to zamknięty zbiór instrukcji wykonujących konkretne zadanie. Funkcje zwykle pobierają dane potrzebne do wykonania operacji (argumenty) i zwracają wynik.

Funkcje są wykonywane tylko w przypadku ich wywołania. Sama definicja funkcji nie powoduje ich wykonania.

Funkcje zamykają często wykorzystywaną funkcjonalność w odpowiednio nazwanych konstrukcjach programistycznych, przez co możliwe jest wielokrotne wykorzystanie już napisanego i sprawdzonego kodu.

Funkcje mają własną przestrzeń nazw, dzięki czemu pozwalają zachować porządek w kodzie.



Python Funkcje

Definiowanie funkcji

W języku Python funkcje definiuje się przy użyciu następującej składni:

def <nazwa_funkcji>(<argumenty_funkcji>):
    <instrukcje>
    return <instrukcje> # opcjonalnie

Na przykład:

```
def F1(x,y):
    z = x**2 + y**2
    return z
```

```
def F2(x):
    x.append(3.1415)

#nic nie zwraca
```

Python Funkcje

Składnia funkcji - def

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

Ciało funkcji,
Instrukcje do wykonania.

Opcjonalnie, jeśli funkcja ma zwracać wynik.

Odpowiednie wcięcie.

Python Funkcje

Wywołanie funkcji

```
def przywitaj():
    print("Dzień dobry")
```

Python Funkcje

Wywołanie funkcji

```
def przywitaj(imie):
    print(f"Dzień dobry, {imie}!")
```

```
przywitaj("Ania")
```

Python Funkcje

Jak działa funkcja?

```
def pierwsza_funkcja():
    print("c")  # 3
    print("d")  # 4

def druga_funkcja():
    print("g")  # 7
    print("h")  # 8

print(„a") # 1
print("b") # 2
pierwsza_funkcja()
print("e") # 5
print("f") # 6
druga_funkcja()
print("i") # 9
```

wynik: **a b c d e f g h i**

Python Funkcje

Nazywanie funkcji

POPRAWNE

```
def oblicz_sume(a, b):
def wczytaj_dane():
def inna_funkcja():
```

NIEPOPRAWNE

```
def 123abc():
def for():
def funkcja-z-nazwa():
```

- `def 123abc():` ← zaczyna się od cyfry
- `def for():` ← używa słowa kluczowego 'for'
- `def funkcja-z-nazwa():` ← używa znaku '-'

Python Funkcje

Parametry funkcji

Parametry funkcji to wyszczególnione zmienne w jej definicji.

```
def przywitaj():
    print("Dzień dobry")


def F(a,b):
    return a**2 + b**2
```

parametry funkcji

W poniższym kodzie a i b są parametrami funkcji F.

Python Funkcje

Argumenty funkcji

```
def przywitaj(imie):
    print(f"Dzień dobry, {imie}!")
```

przywitaj("**Monika**") → # Dzień dobry, Monika!

przywitaj("**Wszystkim**") → # Dzień dobry, Wszystkim!

przywitaj(**2024**) → # Dzień dobry, 2024!

Python Funkcje

Argumenty funkcji

```
def przywitaj(jak, imie):
    print(jak + ", " + imie + "!")
```

```
przywitaj("Cześć!", "Wojtek")
```

```
Cześć, Wojtek!
```

Python Funkcje

Argumenty funkcji

```
def przywitaj(jak, imie):
    print(jak + ", " + imie + "!")
```

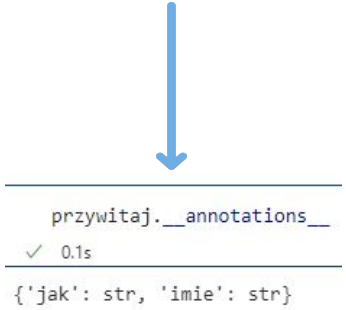
```
przywitaj("Wojtek")
```

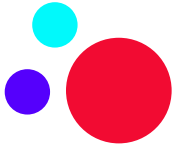
```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
c:\Users\Lenovo\Documents\projekt_ds\notebooks\notes.ipynb Cell 37 line <cell line: 1>()
----> 1 przywitaj("Wojtek")

TypeError: przywitaj() missing 1 required positional argument: 'imie'
```

Python Funkcje

Argumenty funkcji – typy danych

```
def przywitaj(jak: str, imie: str):
    print(jak + ", " + imie + "!")
```





Python Funkcje

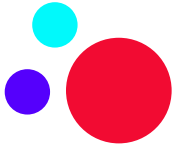
Ciało funkcji

```
def przywitaj(jak: str, imie: str):
    print(jak + ", " + imie + "!")
```

ciało funkcji

```
def pusta_funkcja():
    pass
```

funkcja nie wykonuje żadnych operacji



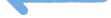
Python Funkcje

Wynik funkcji

```
def przywitaj(jak: str, imie: str):
    print(jak + ", " + imie + "!")


wynik = przywitaj("Cześć!", "Wojtek")
```

 None

```
def podwoj(x):
    return x * 2


wynik = podwoj(5)
```

 wynik = 10

Python Funkcje

Wynik funkcji

```
def liczba(x):
    if x > 0:
        return "Dodatnia"
    elif x == 0:
        return "Zero"
    else:
        return "Ujemna"
    print("Ten tekst się nie wyświetli")

wynik = liczba(5)
```

ten blok kodu nie zostanie wykonany

wywołanie funkcji

Python Funkcje

Definiowanie funkcji - podsumowanie

```
def sprawdz_parzystosc(x):
    if x % 2 == 0:
        return "Parzysta"
    else:
        return "Nieparzysta"


def silnia(n):
    if n == 0 or n == 1:
        return 1
    else:
        return n * silnia(n - 1)
```

Zadanie 7.1

Definiowanie funkcji (instrukcja)

1. Napisz funkcję, która wypisuje sumę dwóch liczb przekazywanych jako argumenty.
2. Napisz funkcję, która zamienia liczbę dni na liczbę milisekund.
3. Napisz funkcję, która zwraca największą z trzech wartości.
4. Napisz funkcję, która przyjmuje napis i zwraca pierwszą literę z alfabetu, która nie występuje w żadnym słowie.
5. Napisz funkcję, która przyjmuje dwie daty i zwraca liczbę dni między nimi.

Python Funkcje

Rekurencja

Rekurencja (rekursja) to sposób definiowania funkcji przy użyciu jej samej. Przykładem może być definicja silni:

$$0! = 1$$
$$N! = (n-1)! \cdot n$$

```
def silnia(n):
    if n == 0 or n == 1:
        return 1
    else:
        return n * silnia(n - 1)
```

rekurencja

Python Funkcje

Rekurencja vs iteracja

Rekurencja:

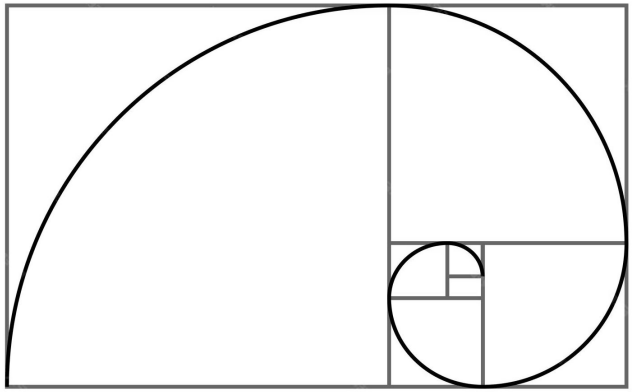
```
def silnia_rekurencyjnie(n):
    if n == 0 or n == 1:
        return 1
    else:
        return n * silnia_rekurencyjnie(n - 1)
```

Iteracja:

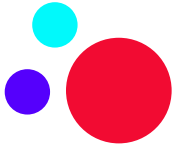
```
def silnia_iteracyjnie(n):
    wynik = 1
    for i in range(1, n + 1):
        wynik *= i
    return wynik
```

Python Funkcje

Rekurencja - zastosowanie



Ciąg Fibonacciego



Python Funkcje

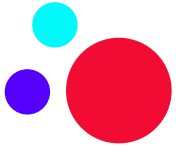
Rekurencja - podsumowanie


```
def Fibo(n):
    if n < 3:
        return 1
    return Fibo(n-1) + Fibo(n-2)

n = int(input("Podaj nr liczby z ciągu Fibonacciego: "))
```

Zadanie 7.2

Rekurencja (instrukcja)

1. Napisz funkcję rekurencyjną, która obliczy sumę liczb naturalnych od 1 do zadanej liczby całkowitej n.



Python Funkcje

Przestrzeń nazw

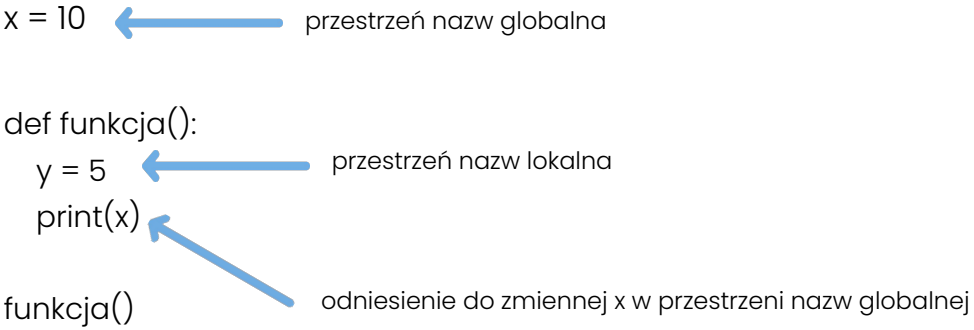
Przestrzeń nazw to pewna kolekcja nazw zmiennych, w której istnieją unikatowe identyfikatory (nazwy zmiennych).

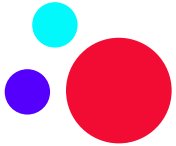
Kolizja nazw to błąd polegający na nadaniu tej samej nazwy dwu lub większej liczbie zmiennych. W języku Python powoduje to zwyczajnie nadpisanie wartości zmiennej. Tego typu błąd prowadzi zwykle do trudnych do wyśledzenia błędów.

Python Funkcje

Przestrzeń nazw

Nazwy parametrów i zmiennych wewnątrz funkcji mogą się pokrywać z nazwami w innych funkcjach lub z nazwami zmiennych globalnych. Nie prowadzi to do kolizji nazw ponieważ istnieją one w różnych przestrzeniach nazw.





Python Funkcje

Rodzaje przestrzeni nazw

Python posiada kilka przestrzeni nazw:

- **built – in** - Wbudowana przestrzeń nazw.
- **global** - Globalna. Zawiera wszystkie nazwy utworzone na poziomie wykonania głównego programu.
- **local** - Lokalna. Przestrzeń nazw istniejąca w trakcie wykonania danej funkcji.
- **enclosing** - Otaczająca. Oznacza przestrzeń nazw, w której zawiera się rozważana przestrzeń nazw.

Python Funkcje

Przestrzeń nazw charakterystyka – built-in

```
dir(len)
✓  0.0s
['__call__',
'__class__',
'__delattr__',
'__dir__',
'__doc__',
'__eq__',
'__format__',
'__ge__',
'__getattribute__',
'__gt__',
'__hash__',
'__init__',
'__init__',
'__subclass__',
'__le__',
'__lt__',
'__module__',
'__name__',
'__ne__',
'__new__',
'__qualname__',
'__reduce__',
'__reduce_
'__ex__',
'__repr__',
'__self__',
'__setattr__',
'__sizeof__',
'__str__',
'__subclasshook__',
'__text_signature__']
```

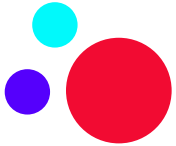
Built-in - Wbudowana przestrzeń nazw (widoczna jako moduł). Znajdują się tu wszystkie wbudowane obiekty języka Python np. None, max, min, print.

Polecenie dir (builtins) pozwala wypisać wszystkie zawarte w niej obiekty.

Zadanie 7.3

built in (instrukcja)

1. Napisz funkcję, która przyjmuje listę liczb całkowitych i zwraca średnią arytmetyczną tych liczb (za pomocą wbudowanych funkcji).



Python Funkcje

Przestrzeń nazw – global

global - Przestrzeń nazw głównego programu. W tej przestrzeni nazw istnieją np. zmienne i definicje funkcji utworzone w interpreterze czy pliku (głównego programu).

```
>>> globalna_zmienna = 10
```

zmienna globalna

```
>>> def funkcja():
        print(globalna_zmienna)
```

korzystanie z globalnej zmiennej wewnątrz funkcji

```
>>> funkcja()
```

```
>>> print(globalna_zmienna)
```

dostęp do globalnej zmiennej poza funkcją

Zadanie 7.4

global (instrukcja)

1. Napisz program, który będzie symulować prosty kalkulator zmiennoprzecinkowy. Użyj zmiennej globalnej do przechowywania bieżącego wyniku. Program powinien umożliwiać użytkownikowi wykonanie operacji dodawania, odejmowania, mnożenia i dzielenia na bieżącym wyniku.

Python Funkcje

Przestrzeń nazw – local

local - Przestrzeń nazw zmiennych wewnątrz funkcji. Zmienne zdefiniowane w tej przestrzeni nazw nie są widoczne na zewnątrz.

```
def funkcja():
    lokalna_zmienna = 8

    lokalna_zmienna += 2
    print(lokalna_zmienna)
```

wartość początkowa zmiennej lokalnej

zmodyfikowana wartość zmiennej lokalnej

Zadanie 7.5

local (instrukcja)

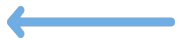
1. Napisz funkcję, która przyjmuje listę liczb całkowitych i zwraca sumę kwadratów tylko liczb parzystych z tej listy. Użyj zmiennej lokalnej w funkcji do przechowywania wyniku.

Python Funkcje

Przestrzeń nazw – enclosing

enclosing - Przestrzeń nazw nadrzędna do rozważanej. Np. definiując funkcję w interpreterze, funkcja ta ma własną przestrzeń nazw (local), natomiast nadrzędną (enclosing) przestrzenią nazw będzie przestrzeń global. Dla przestrzeni global nadrzędną będzie przestrzeń nazw wbudowana (built-in).

```
def zewnetrzna_funkcja(x):
    def wewnetrzna_funkcja(y):
        return x + y
    return wewnetrzna_funkcja
zamkniecie= zewnetrzna_funkcja(10)
result = zamkniecie(5)
```



wewnetrzna_funkcja jest zamknięciem

Zadanie 7.6

enclosing (instrukcja)

1. Stwórz funkcję zewnętrzną multiplier, która przyjmuje jeden argument factor. W funkcji tej, zdefiniuj funkcję wewnętrzną multiply, która będzie mnożyć argument przez factor. Następnie zwróć funkcję multiply jako wynik funkcji multiplier.

Python Funkcje

Zasięg i czas życia zmiennych

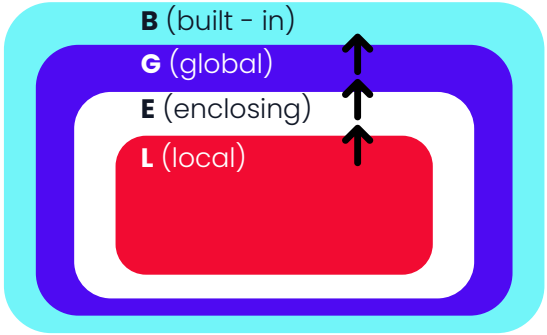
Każda zmienna ma swój zasięg i czas życia.

Zasięg zmiennej oznacza przestrzenie nazw, dla których jest ona dostępna.

Czas życia określa jak długo (w trakcie jakich operacji) istnieje ona w pamięci.

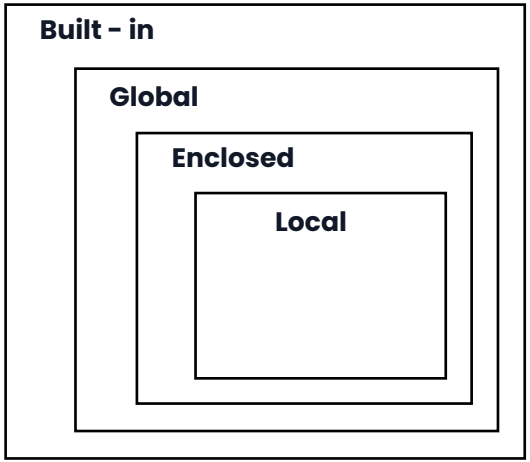
Python Funkcje

Reguła LEGB



Python Funkcje

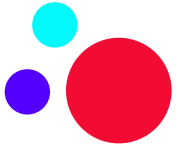
Przestrzeń nazw - podsumowanie



Zadanie 7.7

LEGB (instrukcja)

1. Stwórz funkcję o nazwie obliczenia, która przyjmuje dwa argumenty: a i b. Wewnątrz funkcji zdefiniuj dwie zmienne lokalne: suma oraz iloczyn, które będą przechowywały odpowiednio sumę i iloczyn argumentów a i b. Następnie, utwórz drugą funkcję o nazwie wyniki. Wewnątrz tej funkcji użyj funkcji print, aby wyświetlić lokalne zmienne suma i iloczyn z funkcji obliczenia.



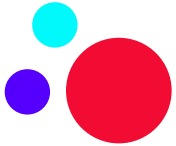
Python Funkcje

Rodzaje argumentów

W języku Python argumenty funkcji mogą być zdefiniowane jako:

- (tylko) **pozycyjne** (positional),
- **nazwane** (keyword),
- z **wartością domyślną** (default value).

Funkcje języka Python mogą przyjmować dowolną i zmienną liczbę argumentów.



Python Funkcje

Argumenty pozycyjne

Każda zmienna ma swój zasięg i czas życia.

Zasięg zmiennej oznacza przestrzenie nazw, dla których jest ona dostępna.

Czas życia określa jak długo (w trakcie jakich operacji) istnieje ona w pamięci.

```
def dodaj(a, b):
    return a + b
wynik = dodaj(3, 5)
```

kolejność wywołana ma znaczenie

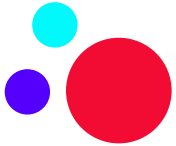
a=3, b=5

Python Funkcje

Argumenty nazwane

```
def odejmij(x, y):
    return x - y
wynik = odejmij(y=5, x=3)
```

zdefiniowanie wartości dla każdego z parametrów



Python Funkcje

Argumenty z wartościami domyślnymi

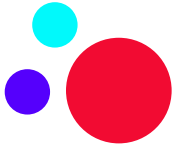
```
def przywitaj(imie="Anonim"):
    return f"Witaj, {imie}!"
wynik = przywitaj()
```

użyta zostanie wartość domyślna

Zadanie 7.8

Parametry (instrukcja)

1. Napisz funkcję, która przyjmuje trzy argumenty: liczba, napis i flaga. Funkcja ma zamieniać liczbę na string, a następnie dołączać do niego podany napis. Jeśli flaga jest ustawiona na True, to zamień liczbę i napis na wielkie litery. Domyślnie flaga ma być ustawiona na wartość True.



Python Funkcje

Parametr *args

Parametr *args oznacza listę argumentów pozycyjnych. Użycie w definicji funkcji tego parametru umożliwia przekazanie do funkcji dowolnej liczby argumentów pozycyjnych. Odwołanie się do wartości kolejnych argumentów następuje poprzez operator indeksowania[ ] lub w pętli, np.:

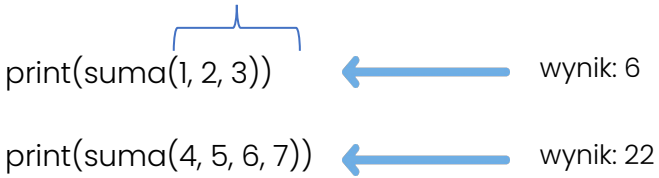
```
>>> def f_arg_test(*args):
>>>     for a in args:
>>>         print(a)


>>> f_arg_test("X",3,[])
x
3
[]
```

Python Funkcje

Parametr *args

```
def suma(*args):
    wynik = 0
    for arg in args:
        wynik += arg
    return wynik
```

zmienna liczba argumentów



Zadanie 7.9

***args (instrukcja)**

1. Napisz funkcję, która przyjmuje dowolną liczbę argumentów i oblicza ich średnią arytmetyczną. Skorzystaj z parametru *args.

Python Funkcje

Argumenty **kwargs

Parametr **kwargs oznacza słownik argumentów przekazywanych przez nazwę. Użycie w definicji funkcji tego parametru umożliwia przekazanie do funkcji dowolnej liczby argumentów poprzez nazwę. Wewnątrz funkcji kwargs jest słownikiem.

```
>>> def f_kwargs_test(**kwargs):
>>>         if "x" in kwargs:
>>>                 print(kwargs[„x"])
```

```
>>> f_kwargs_test(x=5)
5
```

Python Funkcje

Argumenty **kwargs

```
def print_info(**kwargs):
    for key, value in kwargs.items():
        print(f"{key}: {value}")
```

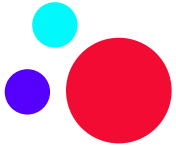
```
print_info(name="Ania", age=30, city="Gdańsk")
```

zmienne i wartości, które traktowane są jako słownik

Zadanie 7.10

****kwargs (instrukcja)**

1. Napisz funkcję, która przyjmuje dowolną liczbę argumentów kluczowych opisujących informacje o osobie. Argumenty kluczowe mogą zawierać takie dane jak imię, nazwisko, wiek, adres, numer telefonu, itp. Funkcja powinna wypisać te informacje w czytelny sposób.



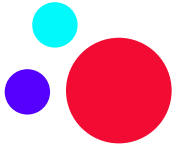
Python Funkcje

Rozpakowanie funkcji *seq

W przypadku argumentów pozycyjnych możliwe jest użycie sekwencji. Kolejne wartości w takim obiekcie zostaną odpowiednio przyporządkowane parametrom funkcji.

```
>>> def f(x,a,b,c):
>>> return a*x**b+c
```

```
>>> T = (0.25,3.0,2.0,-1.0)
>>> f(*T)
```



Python Funkcje

Rozpakowanie argumentów *seq

```
def print_characters(a, b, c):
    print(f"a: {a}, b: {b}, c: {c}")


text = "ABC"
print_characters(*text)
```

rozpakowanie tekstu

Zadanie 7.11

***seq (instrukcja)**

1. Zdefiniuj funkcję calculate_polynomial, która przyjmuje dowolną liczbę argumentów (współczynniki wielomianu) oraz ostatni argument x.

   A. Wielomian ma postać:
   a_n * x^n + a_(n-1) * x^(n-1) + ... + a_1 * x + a_0.
   B. Oblicz wartość wielomianu dla zadanej wartości x.
   C. Przetestuj funkcję dla różnych wielomianów

Python Funkcje

Rozpakowanie argumentów **dict

W przypadku argumentów przekazywanych przez nazwę możliwe jest użycie słownika. Klucze słownika zostaną zinterpretowane jako parametry, a wartości jako wartości parametrów.

```
>>> def f(x,a,b,c):
>>> return a*x**b+c
```

```
>>> D={'c':-1.0,'a':3.0,'x':0.25,'b':2.0}
>>> f(**D)
```

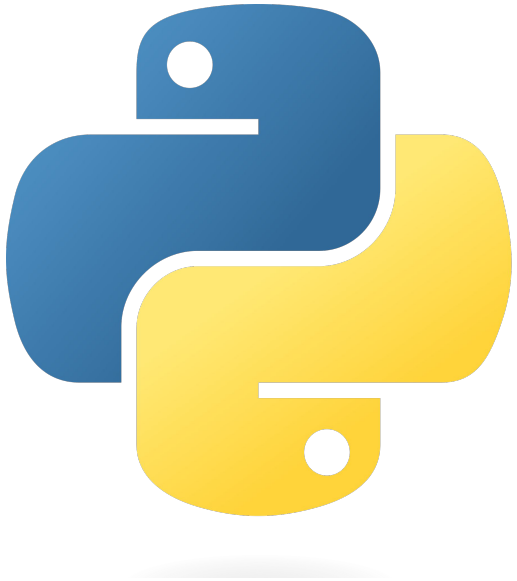
Zadanie 7.12

****dict (instrukcja)**

1. Napisz funkcję, która przyjmuje argumenty cena_podstawowa, stawka_podatku, rabat. Funkcja ma obliczyć ostateczną cenę, uwzględniając podatek i rabat.
   Użyj rozpakowywania argumentów **dict.

Python Funkcje-

Parametry specjalne

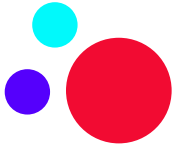


infoShareAcademy.com

Zadanie 7.13

Podsumowanie (instrukcja)

1. Napisz funkcję o nazwie przetwarzaj_dane, która przyjmuje różne rodzaje danych w postaci argumentów. Funkcja powinna obsługiwać trzy rodzaje danych: liczby całkowite, listy oraz słowniki. Jeśli jako argument przekazano liczbę całkowitą, funkcja powinna zwrócić jej kwadrat. Jeśli przekazano listę, funkcja powinna zwrócić sumę wszystkich elementów. Jeśli przekazano słownik, funkcja powinna zwrócić sumę wszystkich wartości w słowniku. W przypadku innych typów danych, funkcja powinna zwrócić komunikat "Nieobsługiwany typ danych".



Python Funkcje

Dokumentacja funkcji docstrings

Docstrings są prostym sposobem dokumentowania funkcji (i nie tylko). Tworzy się je używając potrójnego cudzysłowu pod nagłówkiem funkcji.

```
def f_doc_test(x):
"""Funkcja zwracająca
wartość x do kwadratu"""
    return x**2
```

Python Funkcje

Dokumentacja funkcji docstrings

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

    :param a: Pierwsza liczba do dodania.
    :type a: int or float
    :param b: Druga liczba do dodania.
    :type b: int or float
    :return: Suma dwóch liczb.
    :rtype: int or float
    """
    return a + b
```

Python Funkcje

type hints

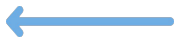
type hints są to dodatkowymi informacjami umieszczanymi w definicjach funkcji. Type hints informują o tym jakich typów danych funkcja oczekuje i jaki typ danych funkcja zwraca.

```
def f1(x:float,a:float, b:float) -> float:
    return x + a + b
```

Python Funkcje

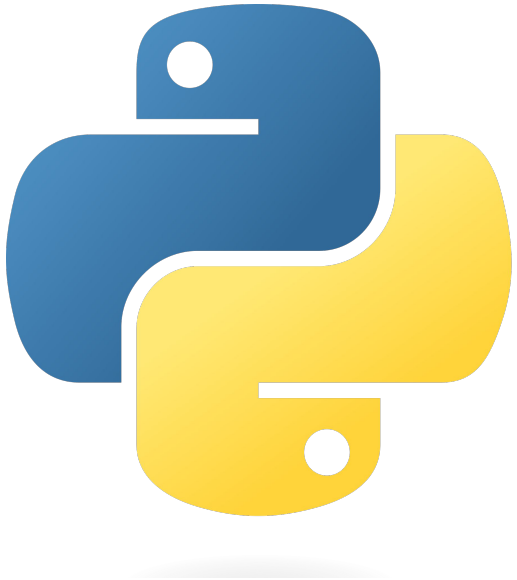
Adnotacje funkcji

```
def f1(x:float,a:float, b:float) -> float:
        return x + a + b

f1.__annotations__

{'x': float, 'a': float, 'b': float, 'return': float}
```



Python Funkcje

Dokumentacja kodu



infoShareAcademy.com

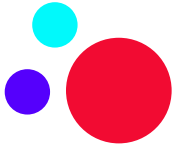
Zadanie 7.14

docstring (instrukcja)

1. Stwórz funkcję, która przyjmuje listę liczb zmiennoprzecinkowych i zwraca słownik zawierający różne statystyki. Dodaj type hints oraz skonstruuj docstring dla tej funkcji.

   Statystyki do uwzględnienia:

   A. Średnia arytmetyczna (średnia).
   B. Odchylenie standardowe.
   C. Mediana.
   D. Minimalna wartość.
   E. Maksymalna wartość.



Python Funkcje

Funkcje anonimowe lambda

Funkcje anonimowe lambda służą do tworzenia funkcji bez ich formalnej definicji. Taka funkcjonalność okazuje się przydatna m.in. jeśli zachodzi potrzeba przekazania funkcji jako argumentu.

f = lambda <argumenty>: <instrukcje>

Python Funkcje

Funkcje anonimowe lambda

Kod poniżej tworzy listę zawierającą listy z liczbami.

```
>>> LL = [[i+j for j in range (10)] for i in range (3) ]
>>> LL
[[0, 1, 2, 3, 4, 5, 6, 7, 8, 9],
 [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
 [2, 3, 4, 5, 6, 7, 8, 9, 10 11]]
```

Następnie funkcji lambda użyto do obliczenia sum kwadratów elementów każdej z list. Funkcję anonimową przekazano jako argument do funkcji **map().**

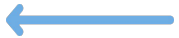
```
>>> list(map(lambda L: sum([e**2 for e in L]),LL))
[285, 385, 505]
```

Python Funkcje

Funkcje lambda

1 Podwajanie liczby:

```
double = lambda x: x * 2
print(double(5))
```



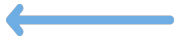
wynik: 10

Python Funkcje

Funkcje lambda

2 Sprawdzanie parzystości:

```
is_even = lambda x: x % 2 == 0
print(is_even(10))
```

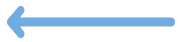
 wynik: True

Python Funkcje

Funkcje lambda

3 Sortowanie listy liczb od największej do najmniejszej:

```
numbers = [1, 8, 4, 6, 2]
sorted_numbers = sorted(numbers, key=lambda x: -x)
print(sorted_numbers)
```



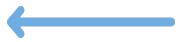
wynik: [8, 6, 4, 2, 1]

Python Funkcje

Funkcje lambda

4 Filtrowanie listy:

```
numbers = [3, 7, 2, 8, 5]
filtered_numbers = list(filter(lambda x: x > 5, numbers))
print(filtered_numbers)
```



wynik: [7, 8]

Python Funkcje

Funkcje lambda

5 Utworzenie funkcji do generowania równań kwadratowych:

```
quadratic_equation = lambda a, b, c, x: a * x**2 + b * x + c
print(quadratic_equation(1, -2, 1, 3))
```

wynik: 10

Python Funkcje

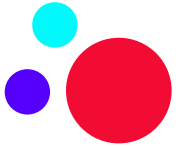
Funkcje lambda

Zadanie 7.15

Funkcja lambda (instrukcja)

1. Dla listy słów napisz funkcję lambda, która sprawdzi, czy każde słowo w liście zaczyna się od wielkiej litery. Następnie użyj funkcji filter, aby zastosować to wyrażenie lambda do listy slowa i uzyskać tylko słowa, które zaczynają się od wielkiej litery.

slowa = ["Python", "programming", "Language", "code", "Lambda"]



Python Funkcje

Funkcje wbudowane

print()
input()
range()
type()
sum()
max()
min()

funkcje wbudowane –
bez potrzeby importowania modułu

Python Funkcje

Funkcje wbudowane - przykłady

- **startswith()**

```
text = "Hello, World!"
result = text.startswith("Hello")
print(result)
```


wynik: True

- **endswith()**

```
text = "Hello, World!"
result = text.endswith("World!")
print(result)
```


wynik: True

Python Funkcje

Funkcje wbudowane - przykłady

- **strip()**

```
text = "   Hello, World!   "
result = text.strip()
```

 Hello, World!

- **lstrip()**

```
text = "   Hello, World!   "
result = text.lstrip()
```

- **rstrip()**

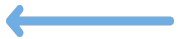
```
text = "   Hello, World!   "
result = text.rstrip()
```

Python Funkcje

Funkcje wbudowane - przykłady

- **index()**

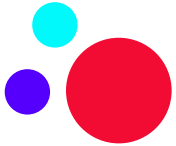
```
text = "Hello, World!"
index = text.index("o")
print(index)
```

 4

- **count()**

```
text = "Python programming is fun and Python is powerful."
count = text.count("Python")
print(count)
```

 2



Python Funkcje

Funkcje wbudowane - podsumowanie

print()
input()
range()
type()
sum()
max()
min()

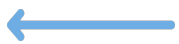
Python Funkcje

Przekazywanie funkcji jako argumentów

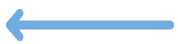
```
def podwoj(x):
    return x * 2
```

funkcja jako argument

```
def podwoj(x):
    return x * 2

def zastosuj(funkcja, argument):
    wynik = funkcja(argument)
    return wynik

podwojone = zastosuj(podwoj, 5)
potrojone = zastosuj(potroj, 7)
```



10



21

Zadanie 7.16

Funkcja jako argument (instrukcja)

1. Napisz funkcję process_list, która przyjmuje dwa argumenty: listę liczb całkowitych i funkcję operacyjną. Funkcja process_list powinna iteracyjnie przetwarzać każdy element listy, używając przekazanej funkcji operacyjnej. Wynikiem powinna być lista z wynikami przetwarzania. Funkcja operacyjna ma być funkcją przyjmującą jeden argument, który jest podniesiony do potęgi 2.

Python Funkcje

Zwracanie funkcji z funkcji

```
def power_function(power):
    """
    Funkcja zwraca funkcję, która podnosi podany argument do potęgi określonej przez 'power'.
    """
    def inner_function(x):
        """
        Funkcja wewnętrzna podnosi podany argument do potęgi 'power'.
        """
        return x ** power
    return inner_function
```

funkcja wewnętrzna

```
square_function = power_function(2)
```
← **zwraca funkcję** podnoszącą do kwadratu

```
cube_function = power_function(3)
```
← **zwraca funkcję** podnoszącą do sześcianu

```
result_square = square_function(4)
```
← podniesienie 4 do kwadratu (16)

```
result_cube = cube_function(3)
```
← podniesienie 3 do sześcianu (27)

Zadanie 7.17

Zwracanie funkcji z funkcji (instrukcja)

1. Napisz funkcję create_calculator, która przyjmuje jeden argument: operację matematyczną (dodawanie, odejmowanie, mnożenie, dzielenie) i zwraca nową funkcję, która wykonuje daną operację na dwóch argumentach. Utwórz funkcję calculate, która używa zwróconej funkcji do obliczenia wyniku dla konkretnych liczb.

Python Funkcje

Zwracanie więcej niż jednego parametru

```
def calculate_statistics(numbers):
    """
    Funkcja przyjmuje listę liczb i zwraca kilka statystyk.
    """
    total = sum(numbers)
    average = total / len(numbers)
    maximum = max(numbers)
    minimum = min(numbers)

    return total, average, maximum, minimum
```

zwracanie więcej niż jednego argumentu

```
numbers_list = [23, 45, 12, 67, 89, 34, 56]

total_sum, avg, max_value, min_value = calculate_statistics(numbers_list)

print("Suma:", total_sum)
print("Średnia:", avg)
print("Maksimum:", max_value)
print("Minimum:", min_value)
```

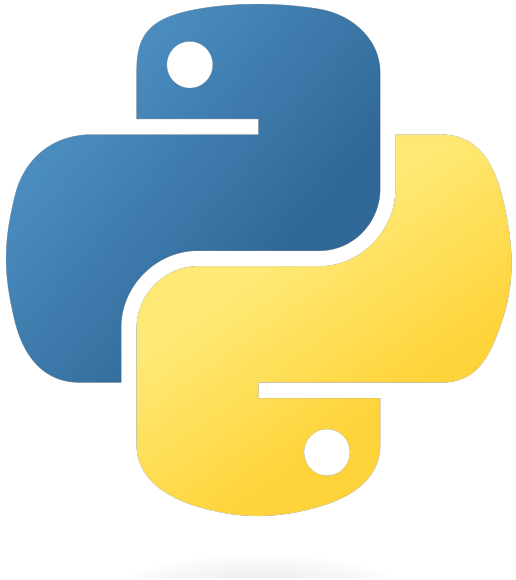
Zadanie 7.18

Zwracanie więcej niż jednego argumentu (instrukcja)

1. Napisz funkcję o nazwie analyze_text, która przyjmuje tekst jako argument i zwraca kilka informacji na temat tego tekstu. Funkcja powinna zwrócić liczbę słów, liczbę unikalnych słów oraz najczęściej występujące słowo w tekście.

Python Funkcje

Podsumowanie



infoShareAcademy.com