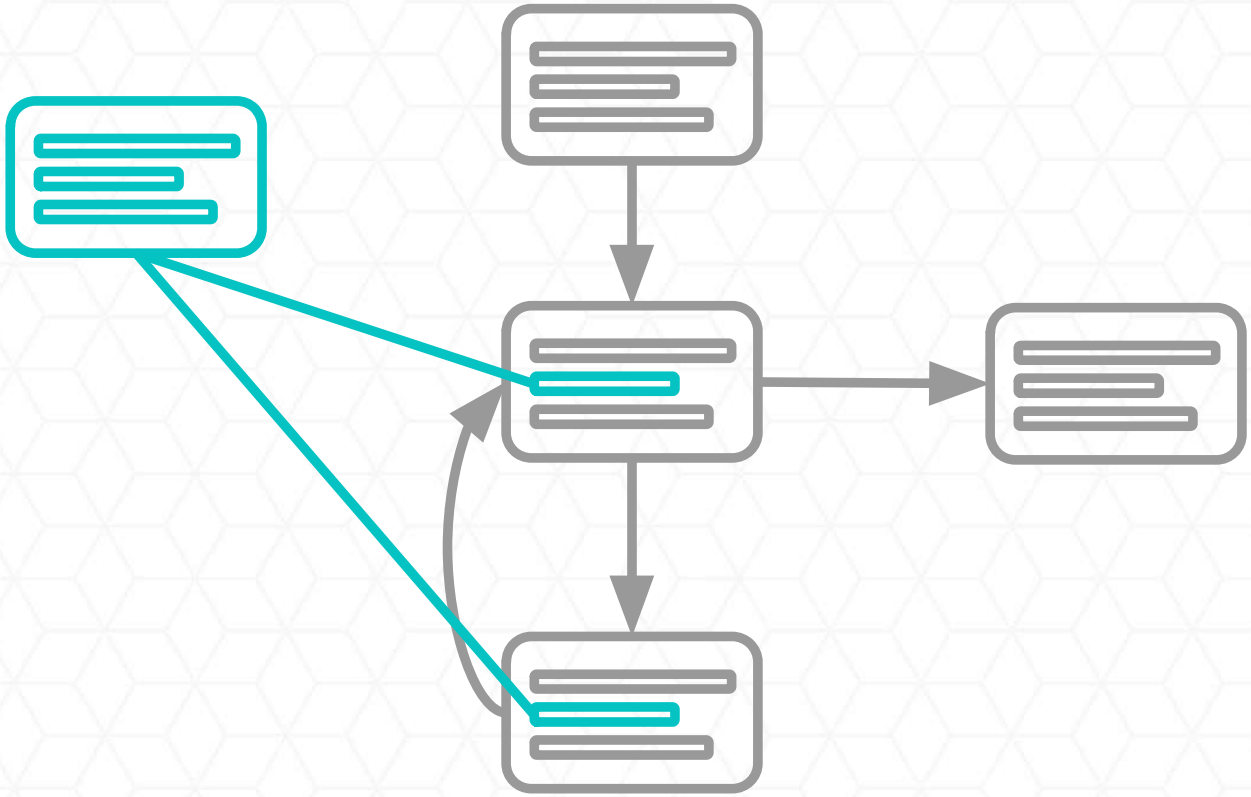
Funkcje

Funkcje



Funkcje

Możemy myśleć o funkcjach jak o niezależnych blokach kodu: przyjmują argumenty, wykonują kod, zwracają rezultat.

Funkcje - definicja

Definiujemy funkcję wyrażeniem **def**.
Wywołujemy wykorzystując nazwę i nawiasy okrągłe.

```
def funk():
    print("Hello")

funk() → results in "Hello" in console
```

Funkcje - zwracanie rezultatu

Rezultat może być zwrócony wyrażeniem **return**. W funkcji może być wiele miejsc w których mamy **return**.

```
def funk():
    return 42

funk() → 42
```

Funkcje - przekazywanie argumentów

Pierwszym typem argumentów są argumenty pozycyjne (również nazywane wymaganymi). Muszą zostać podane przy wywołaniu funkcji.

```
def funk(x):
    return x + 10

funk(5) → 15
```

Funkcje - przekazywanie argumentów

Drugi typ to argumenty opcjonalne (z wartością domyślną, przekazywane przez słowa kluczowe, keyword arguments, kwargs).

```
def funk(x=10):
    return x + 10

funk() → 20

funk(5) → 15
```

Przestrzeń/zakres nazw w funkcjach

Funkcje tworzą swoją lokalną przestrzeń nazw. Oznacza to, że zmienne utworzone wewnątrz funkcji nie będą dostępne na zewnątrz. Ponadto, zmiana wartości zmiennej z zewnętrznego zakresu wymaga dodatkowych operacji.

Przestrzeń/zakres nazw w funkcjach

Lokalne zmienne są widoczne tylko w funkcji.

```
def funk():
    value = 10

funk()
```

x = value + 5 → NameError: name 'value' is not defined

Przestrzeń/zakres nazw w funkcjach

Zmienne z zakresu globalnego są gotowe do odczytu, ale nie chcemy być od nich zależni.

```
def funk():
    value = outer_value + 10 → assign 15 to local variable

outer_value = 5
funk()
```

Przestrzeń/zakres nazw w funkcjach

Zmienna lokalna może mieć taką samą nazwę jak zmienna z zakresu globalnego. Będzie ona wtedy przesłaniać zmienną z zewnętrznego zakresu - nie ma pomiędzy nimi powiązania (shadowing).

```
def funk():
    value = 10

value = 15

funk()          → creates local variable

value + 15      → 30
```

Przestrzeń/zakres nazw w funkcjach

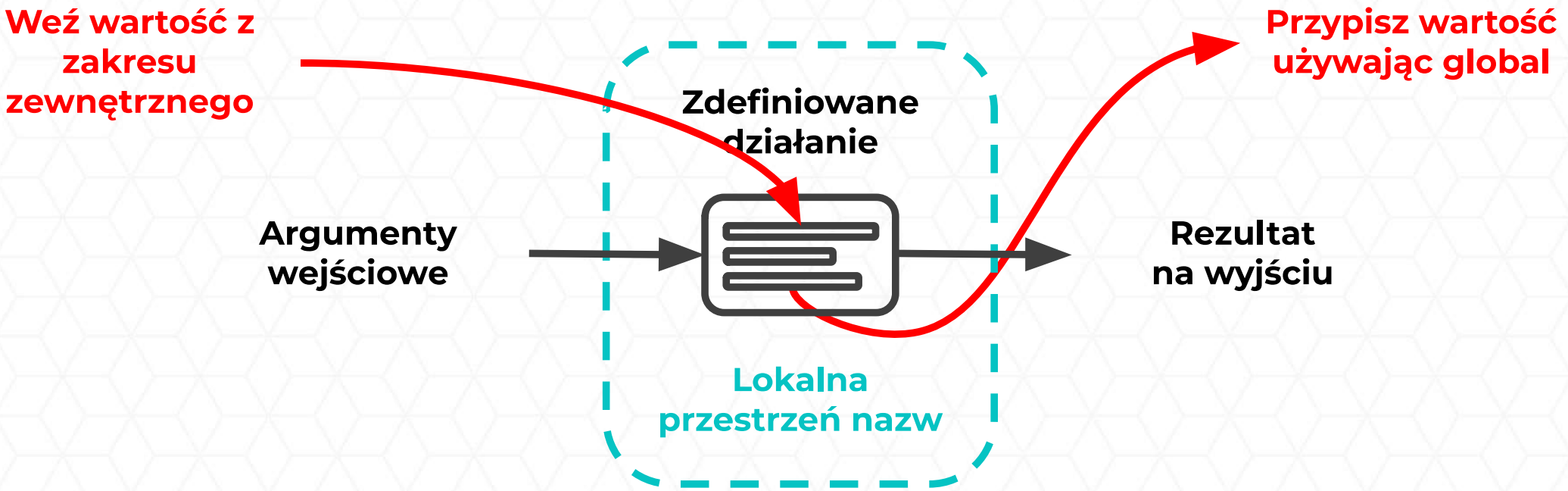
Żeby zmienić wartość zmiennej z zewnętrznego zakresu wewnątrz funkcji, potrzebujemy wyrażenia **global**. Najczęściej jednak nie chcemy tego robić.

```
def funk():
    global value
    value = 10

value = 15
funk()          → will use global variable
value + 15      → 25
```

Przestrzeń/zakres nazw w funkcjach

Korzystanie z wartości z zewnętrznego zakresu i zmiana ich wartości wyrażeniem global tworzy niejawne zależności i oddala nas od idei niezależnych bloków kodu.



Lambdy

Lambdy

Lambdy to “funkcje bez nazwy”. Jest to sposób na jednolinijkowe stworzenie obiektu funkcji, który możemy od razu przekazać.

```
funk = lambda x: x + 1
funk(3) → 4

(lambda x: x + 1)(3) → 4
```

Generatory

Funkcje generujące

Funkcje generujące (generator functions) to jeden ze sposobów na stworzenie generatora (i jednocześnie iteratora), tzn. obiektu, po którym możemy iterować i który będzie tworzył i oddawał element po elemencie (aż do ich wyczerpania).

```
def funk(x):
    for num in range(10):
        yield num + x
```

Wyrażenie generujące

Najbardziej minimalną formą generatora są wyrażenia generujące.

Wyrażenia generujące (obiekty generatora) to element języka umożliwiający stosowanie np. list comprehension.

```
my_list = [x + 1 for x in range(5)]

my_generator = (x + 1 for x in range(5))
```