Funkcje (w matematyce)

W matematyce: dwa sposoby patrzenia na funkcje.

1. Jako przyporządkowanie (z odpowiednią dziedziną i przeciwdziedziną).

$$f : \mathbb{R} \to \mathbb{R}, \qquad f(x) = x^2,$$
$$g : \mathbb{R}^2 \to \{-1, 0, 1\}, \qquad g(x, y) = \operatorname{sgn}(x \cdot y).$$

Wyrażenie $f(2)$ jest tożsame z jego wartością, 4.

2. Jako obiekt matematyczny, na którym można operować.

$$T : \mathbb{R}^{\mathbb{R}} \to \mathbb{R}, \qquad T(h) = h(3),$$
$$S : \mathbb{R}^{\mathbb{R}} \times \mathbb{R}^{\mathbb{R}} \to \mathbb{R}^{\mathbb{R}}, \qquad S(f_1, f_2) = f_1 + f_2.$$

$$[(f_1 + f_2)(x) := f_1(x) + f_2(x)]$$

Funkcje (w Pythonie)

Funkcje w Pythonie (i wielu językach programowania): coś innego.

Funkcje to podprogramy, które:

- Można uruchomić („wywołać"), podając im argumenty.
- Zwracają wartość.
- Mogą wykonywać instrukcje niezwiązane z wyliczaniem tej wartości.

Funkcje są też obiektami.

Składnia definicji funkcji (pierwsza wersja):

def $\overbrace{\text{fun}}^{\textit{nazwa}}$ ($\overbrace{\text{x}_1, \text{x}_1, \ldots, \text{x}_\text{n}}^{\textit{parametry [formalne]}}$):
blok instrukcji } treść

I składnia wywołania, dla $\text{x}_\text{i} = \text{a}_\text{i}$, gdzie a_i to konkretne obiekty:

fun ($\overbrace{\text{a}_1, \text{a}_1, \ldots, \text{a}_\text{n}}^{\textit{argumenty [faktyczne]}}$)

Funkcje (w Pythonie)

Przykład (nieuproszczony).

```
def f(x):
        r = x ** 2
        return r

y = f(2)
```

def f(x): i blok poniżej: definiuje funkcję: nazwę, parametry formalne, instrukcje.
f(2) - wyrażenie (tzn. napis posiadający wartość). Wywołuje f z argumentem x=2.

return expr - instrukcja, która przerywa działanie funkcji. Wartość opcjonalnego wyrażenia expr jest wartością przyjmowaną przez wyrażenie, które wywołało funkcję.

Funkcje (w Pythonie)

Dowolna ilość return:

```
def g(x, y):
        z = x*y
        if z > 0:
                return 1
        elif z < 0:
                return -1
        return 0
```

Domyślnie, return zwraca None (tj. „return" = „return None").
Jeśli w treści funkcji skończą się instrukcje do wykonania, i nie zostanie napotkane return, wywołanie funkcji zwraca None.

```
def print_squares(n,m):
        print("%d^2 = %d" % (n, n ** 2))
        print("%d^2 = %d" % (m, m ** 2))
```

Funkcje (w Pythonie)

Pewne zalety funkcji:

- Modularyzacja kodu: funkcja to „czarna skrzynka", której możemy powierzyć wykonanie pewnych instrukcji lub wyliczenie wartości wyrażenia.
- Wielokrotne użycie tego samego kodu: jeśli program ma wiele fragmentów różniących się szczegółami, można zrobić z nich jedną funkcję, gdzie „szczegół" jest parametrem.
- Lepsza organizacja kodu: funkcje mają wyspecjalizowane zadania i konkretne znaczenie.
- Ukrycie szczegółów implementacji.

Np.

```
def is_prime(n): # True/False
        ...


...
if is_prime(a) and is_prime(b): ...
```

Zasięgi nazw

Zasięg nazw (scope of variables): „gdzie widać nazwę i do czego się ona odnosi".

Ramka: kontekst wykonywania instrukcji w Pythonie.

- Ramka globalna - zawsze obecna.
- Ramka wywołania funkcji - tworzona dla konkretnego wywołania.

Nazwy (i informacja, jakie obiekty nazywają) przynależą do ramki.

- Nazwy utworzone poza wywołaniem funkcji należą do ramki globalnej.
- Nazwy utworzone w wywołanej funkcji (w tym parametry formalne) należą do ramki tego wywołania.
- Nazwy w różnych ramkach mogą się powtarzać, lecz nazywać inne obiekty.

Przy odnoszeniu się do nazw: najpierw sprawdzana jest ramka aktualnego wywołania funkcji (jeśli istnieje), następnie ramka globalna.

Przykłady: materiały do wykładu + wizualizator.

Funkcje jako obiekty

Funkcja jako obiekt: użyta np. jako argument innej funkcji.

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

g = f # g i f nazywaja ten sam obiekt (funkcje)

def ev(fun): # chcemy: fun to funkcja jednego parametru
        return fun(3)

print(ev(g)) # 9
```

```
def call_twice(f, x):
        f(x)
        f(x)

call_twice(print, "Hello world!")
```