# C2184 Úvod do programování v Pythonu

## 6. Funkce

### Funkce

- Objekt, který lze volat (pomocí závorek za jménem funkce)
- *Function, callable*
- Funkce při volání
  1. Něco vezme (**argumenty**)
  2. Něco udělá
  3. Něco vrátí (**návratovou hodnotu**)
- Příklad: funkce `abs`
  1. Vezme 1 argument: číslo $x$
  2. Spočítá absolutní hodnotu $|x|$
  3. Vrátí návratovou hodnotu: $|x|$

[1]:
```
y = abs(-5)
```

[2]:
```
y
```

[2]:
```
5
```

- Příklad: funkce `print`
  1. Vezme libovolný počet argumentů: libovolných objektů
  2. Převede všechny argumenty na řetězce a vypíše je na výstup
  3. Vrátí návratovou hodnotu: `None`

[3]:
```
y = print('ahoj', 5, True)
```

```
ahoj 5 True
```

[4]:
```
y
```

- Příklad: funkce `input`

1. Vezme 0 argumentů
2. Počká na vstup od uživatele
3. Vrátí návratovou hodnotu: řetězec zadaný uživatelem

[5]:
```
y = input()
```

[6]:
```
y
```

[6]:
```
'hello'
```

**Argumenty**

- **Poziční** (*positional arguments, args*)
- **Pojmenované** (*keyword arguments, kwargs*)

Příklad:

[7]:
```
print(1, 2, 'A', sep='-', end=';\n')
```

```
1-2-A;
```

- 3 poziční argumenty: `1`, `2`, `'A'`
- 2 pojmenované argumenty: `'-'`, `';\n'`
- Pojmenované argumenty lze přehazovat

[8]:
```
print(1, 2, 'A', sep='-', end=';\n')
```

```
1-2-A;
```

[9]:
```
print(1, 2, 'A', end=';\n', sep='-')
```

```
1-2-A;
```

- Ale vždy se uvádějí nejdřív poziční, pak pojmenované

[10]:
```
print(1, 2, sep='-', end=';\n', 'A')
```

```
  Input In [10]
    print(1, 2, sep='-', end=';\n', 'A')
                                        ^
SyntaxError: positional argument follows keyword argument
```

### Metody

- Funkce, které jsou součástí objektu
- Voláme je pomocí tečky
- Objekt, ke kterému patří (`self`), je jakoby argumentem

[11]:
```
'ukazatel'.count('a')
```

[11]:
```
2
```

## Můžeme si vytvořit vlastní funkce

- **Proč?**
  - Nejakou operaci provádíme často a nechceme psát vždy to stejné (*DRY – Don't Repeat Yourself*)
    –> vytvoříme na to funkci
  - Máme dlouhý program a chceme ho zpřehlednit (*SoC – Separation of Concerns*)
    –> rozdělíme ho na několik snadno pochopitelných funkcí
- **Jak?**
  - Pomocí klíčového slova `def`

[12]:
```
import math

r1 = 1.0
V1 = 4/3 * math.pi * r1**3
print(f'Koule o poloměru {r1:.2f} má objem {V1:.2f}.')

r2 = 5.0
V2 = 4/3 * math.pi * r2**3
print(f'Koule o poloměru {r2:.2f} má objem {V2:.2f}.')

r3 = 10.0
V3 = 4/3 * math.pi * r3**3
print(f'Koule o poloměru {r3:.2f} má objem {V3:.2f}.')
```

```
Koule o poloměru 1.00 má objem 4.19.
Koule o poloměru 5.00 má objem 523.60.
Koule o poloměru 10.00 má objem 4188.79.
```

[13]:
```
def print_sphere_volume(r):
    V = 4/3 * math.pi * r**3
    print(f'Koule o poloměru {r:.2f} má objem {V:.2f}.')

print_sphere_volume(1.0)
print_sphere_volume(5.0)
print_sphere_volume(10.0)
```

```
Koule o poloměru 1.00 má objem 4.19.
Koule o poloměru 5.00 má objem 523.60.
Koule o poloměru 10.00 má objem 4188.79.
```

### Definice (vytvoření) funkce

- Pomocí klíčového slova `def`

  ```
  def identifier(parameters...):

      body...
  ```

- Funkce má svůj název (*indentifier*), parametry (*parameters*) a tělo (*body*)
- Funkce musí být nejdřív definována, až pak ji můžeme zavolat
- Tělo funkce se nevykoná, dokud funkci nezavoláme

[14]:
```
def print_sphere_volume(r):
    V = 4/3 * math.pi * r**3
    print(f'Koule o poloměru {r:.2f} má objem {V:.2f}.')
```

[15]:
```
print_sphere_volume
```

[15]:
```
<function __main__.print_sphere_volume(r)>
```

[16]:
```
print_sphere_volume(1.0)
```

```
Koule o poloměru 1.00 má objem 4.19.
```

### Volání funkce

1. Hodnoty *argumentů* se dosadí do *parametrů* v definici funkce
2. Provede se tělo funkce
3. Vrátí se hodnota uvedena za klíčovým slovem `return`

[17]:
```
def square_area(side):
    print('Počítám obsah čtverce...')
    area = side**2
    return area
```

[18]:
```
S = square_area(5)
```

```
Počítám obsah čtverce…
```

[19]:
```
S
```

[19]:
```
25
```

### Návratová hodnota funkce (*return value*)

- Hodnota, která je výsledkem volání funkce
- Pomocí klíčového slova `return` v těle funkce
- Jakmile se provede `return`, funkce skončí a zbývající část těla se ignoruje (podobné `break`)!

[20]:
```
def square_area(side):
    print('Počítám obsah čtverce...')
    area = side**2
    return area
    print('**********')
```

[21]:
```
S = square_area(5)
```

```
Počítám obsah čtverce…
```

[22]:
```
S
```

[22]:
```
25
```

### Defaultní návratová hodnota

- Provede-li se celé tělo funkce bez nalezení `return`, funkce vrátí `None`
- Pouhé `return` taky vrátí `None`

[23]:
```
def greet(name):
    print(f'Hello {name}!')

result = greet('Bob')
print(result)
```

```
Hello Bob!
None
```

[24]:
```
def greet(name):
    print(f'Hello {name}!')
    return

result = greet('Alice')
print(result)
```

```
Hello Alice!
None
```

### Parametry a argumenty funkce

- Při volání funkce se argumenty dosazují do parametrů funkce
  - Poziční argumenty po pořadí
  - Pojmenované argumenty podle názvu

[25]:
```
def cylinder_volume(radius, height):
    volume = math.pi * radius**2 * height
    return volume
```

- Poziční argumenty:

[26]:
```
cylinder_volume(1, 5)
```

[26]:
```
15.707963267948966
```

- Pojmenované argumenty:

[27]:
```
cylinder_volume(radius=1, height=5)
```

[27]:
```
15.707963267948966
```

[28]:
```
cylinder_volume(height=5, radius=1)
```

[28]:
```
15.707963267948966
```

- Počet argumentů musí sedět

[29]:
```
cylinder_volume(1)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call
  ↪last)
/home/adam/School/Praca/Python/2022/Python/cviko_06/06_Funkce.ipynb␣
  ↪Cell 51 in <cell line: 1>()
----> <a href='vscode-notebook-cell:/home/adam/School/Praca/Python/
  ↪2022/Python/cviko_06/06_Funkce.ipynb#Y101sZmlsZQ%3D%3D?line=0'>1</
  ↪a> cylinder_volume(1)

TypeError: cylinder_volume() missing 1 required positional argument:␣
  ↪'height'
```

[30]:
```
cylinder_volume(1, 5, 8)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call
  ↪last)
```

```
/home/adam/School/Praca/Python/2022/Python/cviko_06/06_Funkce.ipynb␣
  ↪Cell 52 in <cell line: 1>()
----> <a href='vscode-notebook-cell:/home/adam/School/Praca/Python/
  ↪2022/Python/cviko_06/06_Funkce.ipynb#Y102sZmlsZQ%3D%3D?line=0'>1</
  ↪a> cylinder_volume(1, 5, 8)

TypeError: cylinder_volume() takes 2 positional arguments but 3 were␣
  ↪given
```

### Defaultní hodnoty parametrů

- Můžeme nastavit v definici funkce pomocí =
- Parametry s defaultní hodnotou musí být na konci výčtu parametrů

[31]:
```
def greet(name, repeat=1):
    for i in range(repeat):
        print(f'Hello {name}!')
```

[32]:
```
greet('Bob')
```

```
Hello Bob!
```

[33]:
```
greet('Bob', repeat=3)
```

```
Hello Bob!
Hello Bob!
Hello Bob!
```

### Globální a lokální proměnné

- Globální proměnné (*globals*) – zadefinované mimo funkce
- Lokální proměnné (*locals*) – zadefinované v těle funkce
- Lokální proměnné a parametry existují pouze v rámci konkrétního volání funkce, z vnějšku jsou nedostupné

[34]:
```
def square_area(side):
    area = side**2
    return area

square_area(5)
```

[34]: 25

[35]:
```
area
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call
  ↪last)
/home/adam/School/Praca/Python/2022/Python/cviko_06/06_Funkce.ipynb␣
  ↪Cell 60 in <cell line: 1>()
----> <a href='vscode-notebook-cell:/home/adam/School/Praca/Python/
  ↪2022/Python/cviko_06/06_Funkce.ipynb#Y113sZmlsZQ%3D%3D?line=0'>1</
  ↪a> area

NameError: name 'area' is not defined
```

[36]:
```
side
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call
  ↪last)
/home/adam/School/Praca/Python/2022/Python/cviko_06/06_Funkce.ipynb␣
  ↪Cell 61 in <cell line: 1>()
----> <a href='vscode-notebook-cell:/home/adam/School/Praca/Python/
  ↪2022/Python/cviko_06/06_Funkce.ipynb#Y114sZmlsZQ%3D%3D?line=0'>1</
  ↪a> side

NameError: name 'side' is not defined
```

- Globální proměnné jsou viditelné zevnitř funkce, ale nelze do nich zapisovat

[37]:
```
the_name = 'Bob'

def print_the_name():
    print('The name is:', the_name)

print_the_name()
```

```
The name is: Bob
```

[38]:
```
the_name = 'Bob'

def change_the_name(new_name):
    the_name = new_name  # toto je lokální proměnná, která zakrývá␣
  ↪globální proměnnou the_name

change_the_name('Alice')
print(the_name)
```

```
Bob
```

[39]:
```
the_name = 'Bob'

def print_and_change_the_name(new_name):
    print('The name is:', the_name)  # globální proměnná the_name?
    the_name = new_name  # ale kdepak, the_name je lokální proměnná
    # skončí chybou, protože lokální proměnnou nelze vypsat před její␣
  ↪nastavením!

print_and_change_the_name('Alice')
print(the_name)
```

```
---------------------------------------------------------------------------
UnboundLocalError                         Traceback (most recent call
  ↪last)
/home/adam/School/Praca/Python/2022/Python/cviko_06/06_Funkce.ipynb␣
  ↪Cell 65 in <cell line: 8>()
      <a href='vscode-notebook-cell:/home/adam/School/Praca/Python/
  ↪2022/Python/cviko_06/06_Funkce.ipynb#Y121sZmlsZQ%3D%3D?line=4'>5</
  ↪a>     the_name = new_name  # ale kdepak, the_name je lokální␣
  ↪proměnná
      <a href='vscode-notebook-cell:/home/adam/School/Praca/Python/
  ↪2022/Python/cviko_06/06_Funkce.ipynb#Y121sZmlsZQ%3D%3D?line=5'>6</
  ↪a>     # skončí chybou, protože lokální proměnnou nelze vypsat přeď␣
  ↪její nastavením!
----> <a href='vscode-notebook-cell:/home/adam/School/Praca/Python/
  ↪2022/Python/cviko_06/06_Funkce.ipynb#Y121sZmlsZQ%3D%3D?line=7'>8</
  ↪a> print_and_change_the_name('Alice')
      <a href='vscode-notebook-cell:/home/adam/School/Praca/Python/
  ↪2022/Python/cviko_06/06_Funkce.ipynb#Y121sZmlsZQ%3D%3D?line=8'>9</
  ↪a> print(the_name)

/home/adam/School/Praca/Python/2022/Python/cviko_06/06_Funkce.ipynb␣
  ↪Cell 65 in print_and_change_the_name(new_name)
      <a href='vscode-notebook-cell:/home/adam/School/Praca/Python/
  ↪2022/Python/cviko_06/06_Funkce.ipynb#Y121sZmlsZQ%3D%3D?line=2'>3</
  ↪a> def print_and_change_the_name(new_name):
----> <a href='vscode-notebook-cell:/home/adam/School/Praca/Python/
  ↪2022/Python/cviko_06/06_Funkce.ipynb#Y121sZmlsZQ%3D%3D?line=3'>4</
  ↪a>     print('The name is:', the_name)  # globální proměnná␣
  ↪the_name?
      <a href='vscode-notebook-cell:/home/adam/School/Praca/Python/
  ↪2022/Python/cviko_06/06_Funkce.ipynb#Y121sZmlsZQ%3D%3D?line=4'>5</
  ↪a>     the_name = new_name

UnboundLocalError: local variable 'the_name' referenced before␣
  ↪assignment
```

- Klíčové slovo `global` umožňuje zápis do globální proměnné

[40]:
```
the_name = 'Bob'

def change_the_name(new_name):
    global the_name
    the_name = new_name  # toto je globální proměnná the_name

change_the_name('Alice')
print(the_name)
```

```
Alice
```

- Použití `global` se nedoporučuje!
  - Více funkcí může měnit proměnné zadefinované na různých místech
  - Špatná čitelnost kódu
  - Náchylnost na chyby

[41]:
```
the_name = 'Bob'

def print_the_name():
    print('The name is:', the_name)

the_name = 'Alice'
print_the_name()
```

```
The name is: Alice
```

[42]:
```
def calculate_rectangle_area(a, b):
    global rectangle_area
    rectangle_area = a*b

def calculate_triangle_area(a, b):
    global triangle_area
    calculate_rectangle_area(a, b)
    triangle_area = rectangle_area / 2

calculate_rectangle_area(2, 3)
calculate_triangle_area(10, 20)
print('Rectangle area:', rectangle_area)
print('Triangle area:', triangle_area)

# FUJ!
```

```
Rectangle area: 200
Triangle area: 100.0
```

- Místo globálních proměnných používat:
  - parametry (když chci dostat data do funkce)

  - návratovou hodnotu (když chci dostat data z funkce)
- Použití globálních konstant ve funkci je OK

[43]:
```
GREETING = 'Hello'

def greet(name):
    print(f'{GREETING}, {name}!')

greet('Cyril')
```

```
Hello, Cyril!
```

### Dokumentace

- Aby bylo jasné, co funkce dělá, je zvykem doplnit *docstring* na začátek funkce.
- Nepovinné, ale užitečné, zejména u větších projektů a při spolupráci více lidí.

[44]:
```
def cylinder_volume(radius, height):
    '''Return the volume of a cylinder with specified radius and␣
  ↪height.'''
    volume = math.pi * radius**2 * height
    return volume
```

### Typové anotace

- Můžeme označit typy parametrů a návratové hodnoty.
- Nepovinné, ale užitečné, zejména u větších projektů a při spolupráci více lidí.
- VSCode používá docstrings i typové anotace při napovídání

[45]:
```
def cylinder_volume(radius: float, height: float) -> float:
    '''Return the volume of a cylinder with specified radius and␣
  ↪height.'''
    volume = math.pi * radius**2 * height
    return volume
```

[46]:
```
def greet(name: str, repeat: int = 1) -> None:
    '''Print `repeat` greetings to a person called `name`.'''
    for i in range(repeat):
        print(f'Hello {name}!')
```

- Interpret nekontroluje typy – funkce poběží i když argumenty budou jiných typů.
- Kontrolu typů lze provést pomocí modulu `mypy` (probereme příště).

[47]:
```
greet('Alice')
```

```
Hello Alice!
```

[48]:
```
greet([1, 2, 3])
```

```
Hello [1, 2, 3]!
```

### Generické typy

- `list`, `set`, `tuple`, `dict` – nespecifikují konkrétní typ prvků

### Parametrizované generické typy

- `list[A]` – seznam prvků typu `A`
- `set[A]` – množina prvků typu `A`
- `tuple[A, B, C]` – n-tice s prvním prvkem typu `A`, druhým typu `B`, třetím typu `C`
- `dict[A, B]` – slovník s klíči typu `A` a hodnotami typu `B`

[49]:
```
def min_avg_max(numbers: list) -> tuple:
    '''Return the minimum, average, and maximum of `numbers`.'''
    minimum = min(numbers)
    average = sum(numbers) / len(numbers)
    maximum = max(numbers)
    return (minimum, average, maximum)
```

[50]:
```
def min_avg_max(numbers: list[int]) -> tuple[int, float, int]:
    '''Return the minimum, average, and maximum of the numbers.'''
    minimum = min(numbers)
    average = sum(numbers) / len(numbers)
    maximum = max(numbers)
    return (minimum, average, maximum)

min_avg_max([1, 8, 5, 3])
```

[50]:
```
(1, 4.25, 8)
```

[51]:
```
def word_indices(words: list[str]) -> dict[str, int]:
    '''Return dictionary with index of each word from `words`.'''
    result = {}
    for i, word in enumerate(words):
        result[word] = i
    return result

word_indices('they have been contaminated by pollution'.split())
```

[51]:
```
{'they': 0, 'have': 1, 'been': 2, 'contaminated': 3, 'by': 4,␣
↪'pollution': 5}
```

- Parametrizované generické typy jsou novinka v Pythonu 3.9, v starších verzích musíme použít:
  - `from __future__ import annotations` (jako první import)
  - Nebo generické typy s velkým písmenem (`List`, `Dict`...) z modulu `typing`

[52]:
```
from __future__ import annotations

def min_avg_max(numbers: list[int]) -> tuple[int, float, int]:
    ...
```

[53]:
```
from typing import List, Tuple

def min_avg_max(numbers: List[int]) -> Tuple[int, float, int]:
    ...
```

### Sjednocení typů

- Vyjádřuje, že povolujeme hodnoty více různých typů
- `int | str | bool` – celé číslo nebo řetězec nebo logická hodnota
- `float | None` – reálné číslo nebo `None`

[54]:
```
def foo(a: int | str | bool, b: float | None) -> list[str] | None:
    ...
```

- Sjednocení typů je novinka v Pythonu 3.10, v starších verzích musíme použít:
  - `from __future__ import annotations` (jako první import)
  - Nebo generické typy s velkým písmenem (`Union`, `Optional`...) z modulu `typing`:

    ```
    Union[A, B, C]  =  A | B | C

    Optional[A]  =  Union[A, None]  =  A | None
    ```

[55]:
```
from __future__ import annotations

def foo(a: int | str | bool, b: float | None) -> list[str] | None:
    ...
```

[56]:
```
from typing import Union, Optional, List

def foo(a: Union[int, str, bool], b: Optional[float]) ->␣
↪Optional[List[str]]:
    ...
```

- Další možnosti typových anotací

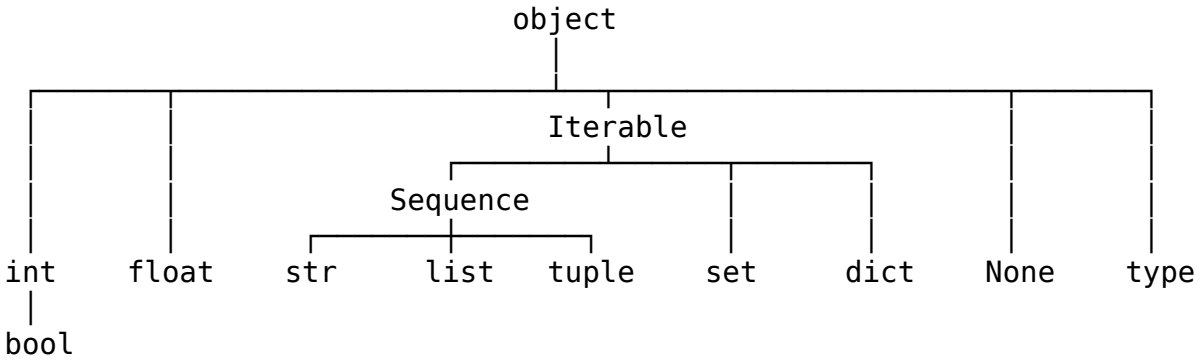
- Z modulu `typing` (https://docs.python.org/3/library/typing.html):
  - `Literal` – přesný výčet povolených hodnot
  - `Any` – sedí na libovolný typ, libovolný typ sedí na `Any`
  - ...
- Z modulu `collections.abc` (https://docs.python.org/3/library/collections.abc.html):
  - `Iterable[A]` – iterovatelný objekt s prvky typu `A`
  - `Sequence[A]` – iterovatelný objekt s prvky typu `A`, indexovatelný přes číselné indexy
  - ...

[57]:
```
from typing import Literal
from collections.abc import Iterable

def print_students(students: Iterable[str], sort_by: Literal['name',␣
  ↪'surname', 'uco']) -> None:
    ...
    # students může být např. typu list[str] nebo set[str]
    # sort_by může být pouze 'name' nebo 'surname' nebo 'uco'
```

**Hierarchie typů**

- Typ `A` může být podtypem jiného typu `B`
- Každá hodnota podtypu `A` je pak zároveň hodnotou nadtypu `B`
- Na vrcholu hierarchie je typ `object`, proto:
  - Všechny typy jsou podtypem typu `object`
  - Hodnota libovolného typu je zároveň hodnotou typu `object`
- Velmi zjednodušená hierarchie v Pythonu:




- `issubclass(A, B)` – zjišťuje, jestli typ `A` je podtypem typu `B`

- `isinstance(x, B)` – zjišťuje, jestli hodnota `x` je typu `B` (`issubclass(type(x), B)`)

[58]:
```
issubclass(int, object)
```

[58]: True

[59]:
```
issubclass(int, str)
```

[59]: False

[60]:
```
isinstance(5, int)
```

[60]: True

[61]:
```
isinstance(5, str)
```

[61]: False

[62]:
```
isinstance(5, object)
```

[62]: True

[63]:
```
def last(elements: list[object]) -> object:
    '''Return the last element of a list.'''
    return elements[-1]
```

- `int` sice není podtypem `float`, ale do parametru typu `float` můžeme vždy dosadit i `int`

[64]:
```
def square_area(a: float, b: float) -> float:
    return a * b

print(square_area(2, 3))
```

```
6
```

### Rozbalování argumentů (*unpacking*)

- Poziční argumenty můžeme rozbalit pomocí * (z iterovatelného objektu)
- Pojmenované argumenty můžeme rozbalit pomocí ** (ze slovníku)

[65]:
```
numbers = [3, 2, 1]
formatting = {'sep': ', ', 'end': '.'}
```

[66]:
```
print(numbers)
```

```
[3, 2, 1]
```

[67]:
```
print(*numbers)  # Ekvivalentní print(3, 2, 1)
```

```
3 2 1
```

[68]:
```
print(*numbers, **formatting)  # Ekvivalentní print(3, 2, 1, sep=',␣
↪', end='.')
```

```
3, 2, 1.
```

**Nenasytné parametry**

- Pokud použijete * před názvem posledního (předposledního) parametru funkce, tento parametr bude obsahovat všechny nadbytečné poziční argumenty
- Pokud použijete ** před názvem posledního parametru, tento parametr bude obsahovat všechny nadbytečné pojmenované argumenty

[69]:
```
def foo(a, b, *args, **kwargs):
    print('a:', a)
    print('b:', b)
    print('args:', args)
    print('kwargs:', kwargs)
```

[70]:
```
foo(1, 2, 3, 4, 5, 6, x=100, y=200)
```

```
a: 1
b: 2
args: (3, 4, 5, 6)
kwargs: {'x': 100, 'y': 200}
```