

# C2184 Úvod do programování v Pythonu (2021)

## Povinné domácí úkoly

V úkolech v této sadě je potřeba (narozdíl od předchozích sad) vždy zadefinovat nějakou funkci. Není třeba načítat vstup pomocí funkce `input`, testy budou volat Vaši funkci tak, jak uvádí **vzorové volání**.

Pokud je uvedené **vzorové volání** a **vzorový výsledek**, chce se od Vás, aby návratovou hodnotou vzorového volání byl vzorový výsledek.

Pokud je uvedené **vzorové volání** a **vzorový výstup**, nemá funkce vracet žádnou návratovou hodnotou, ale má vypisovat vzorový výstup.

### DÚ 6.1: Vzdálenost bodů

Mějme dva body v rovině:  $A = [a_x, a_y]$ ,  $B = [b_x, b_y]$ .

Vzdálenost mezi body  $A$ ,  $B$  umíme spočítat podle vzorce:

$$|AB| = \sqrt{(a_x - b_x)^2 + (a_y - b_y)^2}$$

#### Úkol:

Napište funkci `distance`, která bere jako parametry souřadnice dvou bodů v rovině (tj. dvě dvojice reálných čísel) a vrací vzdálenost těchto bodů.

<b>Vzorové volání 1:</b>	<b>Vzorové volání 2:</b>
<code>distance((1, 1), (5, 4))</code>	<code>distance((-1.5, 3.2), (5.8, 12.6))</code>
<b>Vzorový výsledek 1:</b>	<b>Vzorový výsledek 2:</b>
5.0	11.901680553602503

```
[ ]: from typing import Tuple

def distance(pointA: Tuple[float, float], pointB: Tuple[float,
    ↪ float]) -> float:
    ...

# print(distance((1, 1), (5, 4)))
```

```
# print(distance((-1.5, 3.2), (5.8, 12.6)))
```

## DÚ 6.2: Fibonacciho posloupnost

Fibonacciho posloupnost je nekonečná posloupnost přirozených čísel, která začíná čísly 0, 1, a pak každé další číslo je součtem dvou předchozích:

- 0, 1, 1, 2, 3, 5, 8, 13, 21, ...

Zobecněná Fibonacciho posloupnost se může začínat libovolnými dvěma čísly a pak každé další je součtem dvou předchozích, například:

- 2, 2, 4, 6, 10, 16, 26, 42, ..
- 1, 10, 11, 21, 32, 53, 85, ...

### Úkol:

Napište funkci, která bere jako parametr číslo  $n$  a vrací seznam prvních  $n$  prvků Fibonacciho posloupnosti.

Funkce může být volána také s nepovinným parametrem `start` (dvojice čísel), který určuje první dva prvky zobecněné Fibonacciho posloupnosti.

<b>Vzorové volání 1:</b>	<b>Vzorové volání 2:</b>
<code>fibonacci(8)</code>	<code>fibonacci(6, start=(2, 2))</code>
<b>Vzorový výsledek 1:</b>	<b>Vzorový výsledek 2:</b>
<code>[0, 1, 1, 2, 3, 5, 8, 13]</code>	<code>[2, 2, 4, 6, 10, 16]</code>

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

def fibonacci(n: int, start: Tuple[int, int] = (0, 1)) -> List[int]:
    ...

# print(fibonacci(8))
# print(fibonacci(6, start=(2, 2)))
```

## DÚ 6.3: Sněhulák

Prasátko Peppa si chce vymodelovat z plastelíny sněhuláka a potřebuje vědet, kolik válečků plastelíny si musí koupit.

### Úkol:

Napište funkci `clay_pieces_for_snowman`, která bere tři argumenty:

- `snowman_diameters` je seznam průměrů jednotlivých koulí sněhuláka (mohou být 3 nebo jiný počet),

- `piece_diameter` je průměr válečku plastelíny,
- `piece_length` je délka válečku plastelíny.

Funkce vrátí počet válečků potřebný ke stavbě sněhuláka. Aby funkce nebyla příliš složitá, zadejte si pomocné funkce (např. na výpočet objemu koule, objemu celého sněhuláka, objemu válečku...), které pak můžete volat v hlavní funkci.

Nápověda: funkce `math.ceil()` zaokrouhluje nahoru, `math.floor()` zaokrouhluje dolů. Při zápisu vzorců si dejte pozor na rozdíl mezi průměrem a poloměrem.

---

**Vzorové volání 1:**

`clay_pieces_for_snowman([5, 4, 3], 2.2, 6.0)`

**Vzorový výsledek 1:**

5

---

**Vzorové volání 2:**

`clay_pieces_for_snowman([3.9, 2], 2, 6)`

**Vzorový výsledek 2:**

2

---

Pro kontrolu: ve vzorovém volání 1 má vyjít objem koulí zhruba 65.4, 33.5 a 14.1, objem válečku 22.8. Celý sněhulák tedy vyjde na 4.96 válečku, tj. musí si koupit 5.

```
[ ]: from typing import List
import math

def clay_pieces_for_snowman(snowman_diameters: List[float],
    ↳ piece_diameter: float, piece_length: float) -> int:
    ...

# print(clay_pieces_for_snowman([5, 4, 3], 2.2, 6.0))
# print(clay_pieces_for_snowman([3.9, 2], 2, 6))
```

## DÚ 6.4: EmívuIm uktápzop

### Úkol:

Napište funkci `backwards`, která vezme jako argument větu a vrátí tuto větu s každým slovem pozpátku. Velikost písmen a interpunkci nemusíte řešit (ve větě budou pouze malá písmena a mezery).

---

**Vzorové volání 1:**

`backwards('toto je tajná zpráva')`

**Vzorový výsledek 1:**

'otot ej ánjat avárpz'

**Vzorové volání 2:**

`backwards('oktásarp appep ícarv redú')`

**Vzorový výsledek 2:**

'prasátko peppa vrací úder'

---

```
[ ]: def backwards(sentence: str) -> str:
    ...

# print(backwards('toto je tajná zpráva'))
```

```
# print(backwards('oktásarp appep ícarv redú'))
```

## DÚ 6.5: Výpis slovníku

### Úkol:

Napište funkci `print_dict`, která bere slovník (parametr `dictionary`) a ten hezky vypíše na výstup v tomto formátu (viz vzorový výstup):

- řádek s nadpisem (parametr `title`) a počtem klíčů ve slovníku
- řádky s jednotlivými dvojicemi klíč:hodnota, seřazené podle klíčů, odsazené o 4 mezery

Dále UPRAVTE HLAVIČKU funkce tak, aby ji bylo možné volat i bez parametru `title` (v takovém případě se místo nadpisu vypíše pouze `dict`).

Funkce nemá nic vracet, pouze vypisovat (tj. návratová hodnota je `None`). Klíče a hodnoty mohou být libovolného typu, při jejich výpisu stačí použít defaultní formát.

### Vzorové volání 1:

```
print_dict({'bob': 10, 'cyril': 6, 'alice': 9}, title='Skóre v  
Kloboučku Hop')
```

### Vzorový výstup 1:

```
Skóre v Kloboučku Hop [3]:  
  alice: 9  
  bob: 10  
  cyril: 6
```

### Vzorové volání 2:

```
print_dict({1: 1, 3: 9, 5: 25, 2: 4, 4: 16})
```

### Vzorový výstup 2:

```
dict [5]:  
  1: 1  
  2: 4  
  3: 9  
  4: 16  
  5: 25
```

```
[ ]: def print_dict(dictionary: dict, title: str) -> None:  
    ...  
  
# print_dict({'bob': 10, 'cyril': 6, 'alice': 9}, title='Skóre v  
↪Kloboučku Hop')  
# print_dict({1: 1, 3: 9, 5: 25, 2: 4, 4: 16})
```

## DÚ 6.6: Modus

*Modus* (anglicky *mode*) je hodnota, která se v daném statistickém souboru vyskytuje nejčastěji. Obecně může mít soubor i víc modů, pokud je v něm víc různých hodnot se stejným počtem výskytů.

### Úkol:

Napište definici funkce `mode`, která bere seznam a vrátí jeho modus. Pokud bude seznam prázdný, vraťte `None`. Pokud bude víc modů, vraťte jeden z nich – je jedno který.

#### Vzorové volání 1:

```
mode([1, 2, 3, 3, 4, 4, 1, 2, 5, 3, 2, 1, 2])
```

#### Vzorový výsledek 1:

```
2
```

#### Vzorové volání 2:

```
mode(['jablko', 'pomeranč', 'hruška', 'pomeranč', 'jablko', 'jablko',  
      'hruška'])
```

#### Vzorový výsledek 2:

```
'jablko'
```

```
[ ]: from typing import List  
  
def mode(elements: List[object]) -> object:  
    ...  
  
# print(mode([1, 2, 3, 3, 4, 4, 1, 2, 5, 3, 2, 1, 2]))  
# print(mode(['jablko', 'pomeranč', 'hruška', 'pomeranč', 'jablko',  
→ 'jablko', 'hruška']))
```