Programování 1 (NPRG030)

Cvičení 4: Funkce

Shrnutí přednášky: Funkce

použití tzv. type hints pro lepší čitelnost a udržitelnost kódu

```python
def secti(levy_scitanec: float, pravy_scitanec: float) -> float:
    """
    Sčítá dvě reálná čísla a vrací výsledek.

    Args:
        levy_scitanec (float): První číslo.
        pravy_scitanec (float): Druhé číslo.

    Returns:
        float: Výsledek sčítání.
    """
    vysledek = levy_scitanec + pravy_scitanec
    return vysledek

def main() -> None:
    cislo1 = float(input("Zadejte první číslo: "))
    cislo2 = float(input("Zadejte druhé číslo: "))
    vysledek = secti(cislo1, cislo2)
    print(f"Výsledek sčítání {cislo1}+{cislo2} je {vysledek}")

if __name__ == "__main__":
    main()
```

Docstring dokumentující chování funkce

ukončení funkce a návrat výsledku

funkce nemusí mít vstupní parametry ani "nemusí vracet nic"

podmíněný výraz zajišťuje, že funkce main() bude spuštěna, pokud je soubor vykonáván jako hlavní program (a nikoliv importován do jiného modulu)

volání funkce

Příklad 4.3: Průměrná/minimální/maximální hodnota

- Napište *funkci* v jazyce Python, která vypočítá *aritmetický průměr* ze zadaných hodnot
  - Vstupem funkce bude seznam hodnot
  - Výstupem funkce bude vypočítaný aritmetický průměr

- **Cvičení:**
  - Zkuste vhodně upravit kód tak, abyste kromě aritmetického průměru určili a vrátili také *minimální* a *maximální hodnotu*
    - Zvolte vhodný *přístup*, který umožní snadnou *udržitelnost a čitelnost kódu*

Cvičení 4.4: Řetězec bez samohlásek

- Napište *funkci* v jazyce Python, která přijme řetězec obsahující text a vrátí nový řetězec, ve kterém budou *odstraněny samohlásky*
  - Pro jednoduchost uvažujte pouze znaky z anglické abecedy a pouze malá písmena

- **Těžší varianta**
  - Zkuste navrhnout efektivnější variantu, která nebude muset kopírovat řetězce kvůli jejich spojování

Cvičení 4.5: Caesarova šifra

- Napište *funkce* v jazyce Python, které umožní *zašifrovat* nebo *dešifrovat* text pomocí *Caesarovy šifry*
  - Každá z funkcí by měla mít pouze *jednu odpovědnost*, tj. šifrovat nebo dešifrovat
  - Kód navrhněte tak, abyste *minimalizovali opakování kódu* kvůli lepší udržovatelnosti

- Nápověda: Stačí vhodně upravit program z minulého cvičení

Cvičení 4.6: Řešení kvadratické rovnice

- Napište *funkci* v jazyce Python, která *nalezne reálné řešení kvadratické rovnice*, pokud existuje
  - Vstupem funkce budou koeficienty kvadratické rovnice $ax^2 + bx + c = 0$
  - Výstupem funkce bude seznam reálných řešení

- Nápověda:
  - Výpočet diskriminantu: $D = b^2 - 4 \bullet a \bullet c$
  - Rovnice má dva kořeny pro $D > 0$, konkrétně kořeny $x_{1,2} = \frac{-b \pm \sqrt{D}}{2a}$
  - Rovnice má jediný kořen pro $D = 0$, konkrétně kořen $x = \frac{-b}{2a}$
  - Rovnice nemá kořeny pro $D < 0$

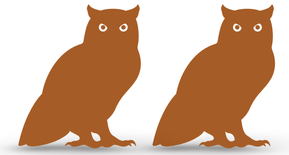
Cvičení 4.7: Refaktorování a zobecnění kódu

- Soubor `refaktoring.py` obsahuje kód funkce, která provádí několik operací

- Vaším úkolem je restrukturalizovat (angl. *refactoring*) tento kód tak, aby byl *udržovatelný*, *čitelný* a *lépe organizovaný*
  - Vhodně pojmenujte proměnné
  - Rozdělte program na funkce, (nejen) abyste dodrželi princip jedné odpovědnosti
  - Doplňte tzv. *type hints*
  - Doplňte vhodný komentář nebo *Docstring*

- Dále *zobecněte kód* tak, aby pracoval s libovolnou velikostí vstupního argumentu
  - Opravte případné bugy v kódu
    - Můžete použít metodu *early exit*

Cvičení 4.8: Násobení matic



- Napište *funkci* v jazyce Python, která implementuje *násobení* dvou *matic*

- Před násobením dvou matic zkontrolujte, zda lze matice spolu násobit
  - Tj. počet sloupců první matice je roven počtu řádků druhé matice

- Pokud *matice* spolu *nelze násobit*, vraťte vhodnou chybovou zprávu (*early exit*)

Jak psát kód: Don't Repeat Yourself (DRY)

- Vyhněte se duplicitnímu kódu
- K zapouzdření opakovaně použité logiky *používejte funkce*, *třídy* a *moduly*

```
1 # Bez použití DRY
2 print("Hello, Alice!")
3 print("Hello, Bob!")
4 print("Hello, Emily!")
```

```
1 # Lepší implementace
2 def greet(name):
3   print(f"Hello {name}!")
4
5 greet("Alice")
6 greet("Bob")
7 greet("Emily")
```

Jak psát kód: You Aren't Gonna Need It (YAGNI)

- Zavádějte *pouze to, co aktuálně potřebujete*, nikoliv to, o čem si myslíte, že byste mohli potřebovat v budoucnu
  - Ale zároveň myslete na rozšiřitelnost kódu

```
1 # Vyhněte se nadměrnému inženýrství
2 def add_and_multiply(left_op, right_op):
3   return left_op + right_op, left_op * right_op
```

```
1 # YAGNI přístup
2 def add(left_op, right_op):
3   return left_op + right_op
```

Jak psát kód: Zásada jediné odpovědnosti

- Funkce nebo třída by měla mít jedinou odpovědnost (angl. *single responsibility*)
  - To usnadňuje údržbu a pochopení kódu.

```
1 # Více odpovědností
2 def secist_a_ulozit(hodnoty):
3   soucet = sum(hodnoty)
4   ulozit_soucet_do_databaze(soucet)
```

```
1 # Jediná odpovědnost
2 def secist(hodnoty):
3   return sum(hodnoty)
4
5 def ulozit(soucet):
6   ulozit_soucet_do_databaze(soucet)
```

Jak psát kód: Méně operací na jednom řádku

- Vyhněte se vměstnání více operací do jednoho řádku
  - Řádky kódu musí být stručné, ale ne příliš složité

```
# Příliš mnoho v jednom řádku
vysledek = zpracovat(secist(levy_scitanec, pravy_scitanec), data)
```

```
# Rozdělení na více řádků
mezivysledek = secist(levy_scitanec, pravy_scitanec)
vysledek = zpracovat(mezivysledek, data)
```

Jak psát kód: Plochá struktura kódu

- Přepracujte hluboce vnořené funkce pro lepší strukturu kódu

```
# Hluboké zanoření
def secist(hodnoty):
  def zprumerovat(hodnoty):
    def vazene_zprumerovat(hodnoty):
      pass
```

```
# Plochá struktura kódu
def secist(hodnoty:
  pass

def zprumerovat(hodnoty):
  pass

def vazene_zprumerovat(hodnoty):
  pass
```

Jak psát kód: Omezení délky řádku

- Pro lepší čitelnost udržujte řádky kódu přiměřeně krátké

```
# Příliš dlouhý řádek
result = secist_hodnoty(dlouhy_argument, dalsi_dlouhy_argument, jeste_dalsi_dlouhy_argument)
```

```
# Rozdělte na několik řádků
result = secist_hodnoty(
  dlouhy_argument,
  dalsi_dlouhy_argument,
  jeste_dalsi_dlouhy_argument
)
```

Jak psát kód: Nápověda k typu

- Používejte typové nápovědy (angl. *type hints*) k dokumentaci očekávaných typů parametrů a návratových hodnot funkcí

```
# Můžeme sečíst např. řetězec a list? A co bude výsledkem?
def secist(levy_scitanec, pravy_scitanec):
  return levy_scitanec + pravy_scitanec
```

```
# Jednoznačné použití
def secist(levy_scitanec: int, pravy_scitanec: int) -> int:
  return levy_scitanec + pravy_scitanec
```

Jak psát kód: Dokumentace funkcí

- Použijte *Docstrings* pro dokumentaci funkcí
  - Dokumentujte funkce, abyste poskytli jasné vysvětlení jejich *účelu*, *parametrů* a *návratových hodnot*

```
 1 def vypocitat_obsah(polomer: float) -> float:
 2   """
 3   Vypočítá obsah kruhu.
 4
 5   Parametry:
 6     polomer (float): Polomer kruhu.
 7
 8   Návratová hodnota:
 9     float: Obsah kruhu.
10   """
11   return pi * polomer ** 2
```

Jak psát kód: Early exit

- Co nejdříve *kontrolujte podmínky*, které by *mohly vést k nežádoucímu* nebo chybnému *výsledku*
- Pokud jsou tyto podmínky splněny, *ukončete aktuální blok kódu* dříve, čímž se vyhnete zbytečnému zpracování zbytku bloku kódu

```
1 # Nejprve ošetříme nežádoucí chování
2 def vydelit(delenec: int, delitel: int) -> int:
3   if delitel == 0:
4     print("Error: Dělení nulou")
5     return None
6   else:
7     return delenec // delitel
```

Jak psát kód: Popisné názvy

- Používejte *výstižné názvy*, které vyjadřují účel proměnných a funkcí

```python
# Není výstižné
def func(x: int) -> int:
  return x ** 2
```

```python
# Výstižné pojmenování
def umocnit_hodnotu(hodnota: int) -> int:
  return hodnota ** 2
```

Jak psát kód: (Ne)použití globálních proměnných

- Minimalizujte používání globálních proměnných, abyste předešli nechtěným vedlejším účinkům

```
# Vyhněte se
suma = 0

 def pricist_k_sume(hodnota):
   global suma
   suma += hodnota
```

```
# Preferujte tuto variantu:
def secist(levy_scitanec: int, pravy_scitanec: int) -> int:
  return levy_scitanec + pravy_scitanec

suma = secist(suma, hodnota)
```

Jak psát kód: Důslednost v pojmenování

- Udržujte *konzistentní konvence pojmenování* a odsazování napříč celým zdrojovým kódem

- Příklad: Opravení gramatické chyby pouze na některých místech budí dojem různého významu (funkce nebo proměnné)
  - Lepší je gramaticky špatně pojmenovaná funkce než mít nekonzistentní názvy

```
1 # Nekonzistentní
2 def calculate_taxAmount(income):
3   pass
```

```
1 # Konzistentní
2 def calculate_tax_amount(income):
3   pass
```

Reference

Funkce

- https://www.w3schools.com/python/python_functions.asp

Scope

- https://www.w3schools.com/python/python_scope.asp