

# Lekce 6: Funkce a moduly

Python pro GIS - Znovupoužitelný kód

Vojtěch Barták, FŽP ČZU Praha

2025-10-20

## Table of contents

<b>1</b>	<b>Cíle lekce</b>	<b>4</b>
<b>2</b>	<b>Proč funkce?</b>	<b>4</b>
2.1	Motivace . . . . .	4
2.2	DRY princip . . . . .	5
<b>3</b>	<b>Definice funkcí</b>	<b>5</b>
3.1	Základní syntaxe . . . . .	5
3.2	První funkce . . . . .	5
<b>4</b>	<b>Parametry funkcí</b>	<b>6</b>
4.1	Funkce s jedním parametrem . . . . .	6
4.2	Funkce s více parametry . . . . .	6
4.3	Default hodnoty parametrů . . . . .	7
<b>5</b>	<b>Návratové hodnoty - return</b>	<b>7</b>
5.1	Problém s print() . . . . .	7
5.2	Řešení: return . . . . .	7
5.3	return ukončuje funkci . . . . .	8
<b>6</b>	<b>Dokumentační řetězce (docstrings)</b>	<b>9</b>
6.1	Co jsou docstrings? . . . . .	9
6.2	Proč používat docstrings? . . . . .	9
<b>7</b>	<b>Přepis úloh jako funkcí</b>	<b>10</b>
7.1	Fibonacci jako funkce . . . . .	10
7.2	Cvičení 1: Faktoriál jako funkce . . . . .	11
7.3	Cvičení 2: Test prvočíselnosti jako funkce . . . . .	11

7.4	Cvičení 3: Bubble Sort jako funkce . . . . .	11
<b>8</b>	<b>Moduly</b>	<b>12</b>
8.1	Co je modul? . . . . .	12
8.2	Vytvoření vlastního modulu . . . . .	13
8.2.1	Krok 1: Vytvořte soubor <code>math_utils.py</code> . . . . .	13
8.2.2	Krok 2: Vytvořte soubor <code>main.py</code> ve STEJNÉ složce . . . . .	14
8.3	Způsoby importu . . . . .	14
8.3.1	1. Import celého modulu . . . . .	14
8.3.2	2. Import konkrétních funkcí . . . . .	14
8.3.3	3. Import s aliasem . . . . .	14
8.3.4	4. Import všeho (NEDOPORUČENO!) . . . . .	15
8.4	Vestavěné moduly . . . . .	15
8.4.1	Modul <code>math</code> . . . . .	15
8.4.2	Modul <code>random</code> . . . . .	15
8.4.3	Modul <code>os</code> (operační systém) . . . . .	16
8.4.4	Modul <code>csv</code> (práce s CSV) . . . . .	16
8.5	Preview: <code>import arcpy</code> . . . . .	16
<b>9</b>	<b>Struktura projektu</b>	<b>16</b>
9.1	Ideální organizace . . . . .	16
<b>10</b>	<b>Praktická úloha</b>	<b>17</b>
10.1	Zadání . . . . .	17
10.1.1	1. Soubor <code>math_utils.py</code> . . . . .	17
10.1.2	2. Soubor <code>main.py</code> . . . . .	18
<b>11</b>	<b>Shrnutí</b>	<b>18</b>
11.1	Co jsme se naučili . . . . .	18
11.2	Co bude příště? . . . . .	19
<b>12</b>	<b>Domácí úkol</b>	<b>19</b>
12.1	Varianta A (základní) . . . . .	19
12.2	Varianta B (pokročilá) . . . . .	19
12.3	Varianta C (výzva) . . . . .	20
<b>13</b>	<b>Cheatsheet</b>	<b>21</b>
<b>14</b>	<b>Poznámky pro vyučujícího</b>	<b>23</b>
14.1	Běžné chyby studentů . . . . .	23
14.2	Časový plán (90 min) . . . . .	23
14.3	Klíčové momenty . . . . .	24
14.3.1	<code>print()</code> vs. <code>return</code> (30-40 min): . . . . .	24
14.3.2	Moduly (60-85 min): . . . . .	24

14.3.3 Docstrings (40-50 min): . . . . .	24
14.4 Rizika . . . . .	25
14.5 Tipy . . . . .	25
14.6 Materiály k přípravě . . . . .	25

# 1 Cíle lekce

Po absolvování této lekce budete umět:

- Definovat vlastní funkce pomocí `def`
- Používat parametry a návratové hodnoty
- Rozumět rozdílu mezi `print()` a `return`
- Psát dokumentační řetězce (docstrings)
- Organizovat kód do modulů
- Importovat a používat moduly
- Vytvořit vlastní modul `math_utils.py`

Časová dotace: 90 minut

---

## 2 Proč funkce?

### 2.1 Motivace

Vzpomeňte si na předchozí lekce:

Napsali jste kód pro: - Fibonacci číslo - Faktoriál - Hledání minima/maxima - Třídění seznamu  
- Test prvočíselnosti

**Co kdybyste tyto operace potřebovali použít vícekrát?**

```
# Fibonacci pro číslo 10
n = 10
a, b = 0, 1
for i in range(n):
    a, b = b, a + b
print(f"Fibonacci({n}) = {a}")

# Fibonacci pro číslo 20 - MUSÍME PSÁT ZNOVU!
n = 20
a, b = 0, 1
for i in range(n):
    a, b = b, a + b
print(f"Fibonacci({n}) = {a}")

# Fibonacci pro číslo 15 - ZASE ZNOVU!
# ...
```

**Problém:** Opakujeme stejný kód → neefektivní, náchylné k chybám

## 2.2 DRY princip

**DRY = Don't Repeat Yourself** (Neopakuj se)

Pokud píšete stejný kód vícekrát, měli byste ho **zabalit do funkce**.

💡 Funkce = pojmenovaný kus kódu

Funkce je **pojmenovaný blok kódu**, který: - Můžete volat kdykoli - Může přijímat vstupy (parametry) - Může vracet výstup (návrátovou hodnotu) - Dělá kód čitelnějším a znovupoužitelným

---

## 3 Definice funkcí

### 3.1 Základní syntaxe

```
def jmeno_funkce():  
    # Tělo funkce (odsazené)  
    prikaz1  
    prikaz2
```

**Klíčové slovo:** def (define)

**Závorky:** () jsou povinné

**Dvojtečka:** : na konci řádku

**Odsazení:** Tělo funkce je odsazené (4 mezery)

### 3.2 První funkce

```
def pozdrav():  
    print("Ahoj!")  
    print("Vítej v kurzu Pythonu!")  
  
# Volání funkce:  
pozdrav()
```

Výsledek:

Ahoj!

Vítej v kurzu Pythonu!

! Funkci musíte ZAVOLAT!

Definice funkce ji jen **vytvoří**, ale **neprovede** (“nezavolá”):

```
def pozdrav():  
    print("Ahoj!")  
  
# Nic se nevypíše! Funkce jen existuje.  
  
pozdrav() # TEĎ se zavolá!
```

## 4 Parametry funkcí

### 4.1 Funkce s jedním parametrem

```
def pozdrav(jmeno):  
    print(f"Ahoj, {jmeno}!")  
  
pozdrav("Jan")      # Ahoj, Jan!  
pozdrav("Marie")    # Ahoj, Marie!
```

**Parametr** = proměnná, která přijímá hodnotu při volání funkce

### 4.2 Funkce s více parametry

```
def secti(a, b):  
    vysledek = a + b  
    print(f"{a} + {b} = {vysledek}")  
  
secti(5, 3)  # 5 + 3 = 8  
secti(10, 7) # 10 + 7 = 17
```

## 4.3 Default hodnoty parametrů

Můžete nastavit **výchozí hodnotu** pro parametr:

```
def pozdrav(jmeno, jazyk="cs"):
    if jazyk == "cs":
        print(f"Ahoj, {jmeno}!")
    elif jazyk == "en":
        print(f"Hello, {jmeno}!")

pozdrav("Jan")          # Ahoj, Jan! (použije default "cs")
pozdrav("John", "en")   # Hello, John!
pozdrav("Marie", "cs")  # Ahoj, Marie!
```

---

## 5 Návrátové hodnoty - return

### 5.1 Problém s print()

```
def secti(a, b):
    vysledek = a + b
    print(vysledek)

# Můžeme vypsat výsledek:
secti(5, 3) # 8

# Ale NEMŮŽEME ho použít dál:
x = secti(5, 3) # x = None (funkce nic nevrací!)
y = x + 10      # CHYBA! None + 10 nejde
```

### 5.2 Řešení: return

```
def secti(a, b):
    vysledek = a + b
    return vysledek # Vrábí hodnotu
```

```
# Teď MŮŽEME výsledek použít:  
x = secti(5, 3)    # x = 8  
y = x + 10         # y = 18  
print(f"Výsledek: {y}") # Výsledek: 18
```

#### ⚠ print() vs. return - KRITICKÝ ROZDÍL!

- **print()** - vypíše na obrazovku, ale **nevrací** hodnotu
- **return** - **vrací** hodnotu, kterou můžete použít dál

```
def spatne(x):  
    print(x * 2)    # Vypíše, ale nevrací  
  
def spravne(x):  
    return x * 2    # Vrací hodnotu  
  
a = spatne(5)      # Vypíše 10, ale a = None  
b = spravne(5)     # b = 10 (můžeme použít dál)
```

**Pravidlo:** Funkce by měly **vracet** hodnoty pomocí **return**, ne je vypisovat pomocí **print()**!

### 5.3 return ukončuje funkci

```
def je_kladne(cislo):  
    if cislo > 0:  
        return True    # Funkce SKONČÍ zde  
    return False       # Tohle se provede jen pokud cislo <= 0  
  
print(je_kladne(5))    # True  
print(je_kladne(-3))   # False
```



## 6 Dokumentační řetězce (docstrings)

### 6.1 Co jsou docstrings?

**Docstring** = dokumentace funkce napsaná hned pod definicí v trojitých uvozovkách.

```
def fibonacci(n):  
    """Vrátí n-tý člen Fibonacciho posloupnosti.  
  
    Args:  
        n: Pořadové číslo členu (int)  
  
    Returns:  
        n-tý člen posloupnosti (int)  
    """  
    a, b = 0, 1  
    for i in range(n):  
        a, b = b, a + b  
    return a
```

### 6.2 Proč používat docstrings?

```
# Můžete si přečíst dokumentaci:  
help(fibonacci)
```

**Výstup:**

Help on function fibonacci:

```
fibonacci(n)  
    Vrátí n-tý člen Fibonacciho posloupnosti.  
  
    Args:  
        n: Pořadové číslo členu (int)  
  
    Returns:  
        n-tý člen posloupnosti (int)
```

💡 Dobrý docstring obsahuje

1. **Stručný popis** - co funkce dělá
2. **Args** - jaké přijímá parametry
3. **Returns** - co vrací
4. **Raises** (volitelně) - jaké může vyvolat chyby

Nemusíte psát složité docstringy, ale alespoň jednořádkový popis je dobrý!

---

## 7 Přepis úloh jako funkcí

### 7.1 Fibonacci jako funkce

Původní kód:

```
n = int(input("Které číslo? "))
a, b = 0, 1
for i in range(n):
    a, b = b, a + b
print(f"{n}. člen: {a}")
```

Jako funkce:

```
def fibonacci(n):
    """Vrátí n-tý člen Fibonacciho posloupnosti."""
    a, b = 0, 1
    for i in range(n):
        a, b = b, a + b
    return a

# Použití:
print(f"10. člen: {fibonacci(10)}")
print(f"20. člen: {fibonacci(20)}")

# Nebo pro seznam čísel:
for i in [5, 10, 15, 20]:
    print(f"F({i}) = {fibonacci(i)}")
```

## 7.2 Cvičení 1: Faktoriál jako funkce

Přepište váš kód pro faktoriál jako funkci:

```
def faktorial(n):
    """Vrátí faktoriál čísla n."""
    # Zde doplňte kód...

# Test:
print(faktorial(5))    # Mělo by být 120
print(faktorial(10))   # Mělo by být 3628800
```

## 7.3 Cvičení 2: Test prvočíselnosti jako funkce

```
def je_prvocislo(n):
    """Vrátí True, pokud je n prvočíslo, jinak False."""
    # Zde doplňte kód...

# Test:
print(je_prvocislo(7))    # True
print(je_prvocislo(8))    # False
print(je_prvocislo(17))   # True
```

## 7.4 Cvičení 3: Bubble Sort jako funkce

```
def bubble_sort(seznam):
    """Seřadí seznam pomocí Bubble Sort algoritmu.

    Args:
        seznam: Seznam čísel (list)

    Returns:
        Seřazený seznam (list)
    """
    # POZOR: Měli byste vytvořit kopii!
    serazeny = seznam.copy()

    # Zde doplňte třídící kód...
```

```
    return serazeny

# Test:
cisla = [5, 2, 8, 1, 9]
serazene = bubble_sort(cisla)
print(f"Původní: {cisla}")
print(f"Seřazené: {serazene}")
```

### ⚠ Pozor na úpravu seznamů!

Seznamy jsou **mutable** (měnitelné). Pokud funkce upravuje seznam, mění původní:

```
def spatne_sort(seznam):
    seznam.sort() # Mění původní seznam!
    return seznam

cisla = [5, 2, 8]
vysledek = spatne_sort(cisla)
print(cisla) # [2, 5, 8] - ZMĚNĚNO!

# Lepší:
def dobre_sort(seznam):
    kopie = seznam.copy()
    kopie.sort()
    return kopie
```

---

## 8 Moduly

### 8.1 Co je modul?

**Modul** = Python soubor (.py) obsahující funkce, které můžete použít v jiných programech.

**Proč moduly?** - **Organizace** - rozdělení programu do logických celků - **Znovupoužitelnost**  
- jednou napíšete, použijete všude - **Sdílení** - můžete sdílet s kolegy

## 8.2 Vytvoření vlastního modulu

### 8.2.1 Krok 1: Vytvořte soubor math\_utils.py

```
# math_utils.py

def fibonacci(n):
    """Vrátí n-tý člen Fibonacciho posloupnosti."""
    a, b = 0, 1
    for i in range(n):
        a, b = b, a + b
    return a

def faktorial(n):
    """Vrátí faktoriál čísla n."""
    vysledek = 1
    for i in range(1, n + 1):
        vysledek *= i
    return vysledek

def je_prvocislo(n):
    """Vrátí True, pokud je n prvočíslo."""
    if n < 2:
        return False
    for i in range(2, n):
        if n % i == 0:
            return False
    return True

def bubble_sort(seznam):
    """Seřadí seznam pomocí Bubble Sort."""
    serazeny = seznam.copy()
    for i in range(len(serazeny)):
        for j in range(len(serazeny) - 1 - i):
            if serazeny[j] > serazeny[j + 1]:
                serazeny[j], serazeny[j + 1] = serazeny[j + 1], serazeny[j]
    return serazeny
```

### 8.2.2 Krok 2: Vytvořte soubor main.py ve STEJNÉ složce

```
# main.py

import math_utils

# Teď můžeme používat funkce z math_utils:
n = int(input("Zadej číslo: "))

print(f"Fibonacci: {math_utils.fibonacci(n)}")
print(f"Faktoriál: {math_utils.faktorial(n)}")
print(f"Je prvočíslo: {math_utils.je_prvocislo(n)}")

cisla = [5, 2, 8, 1, 9]
print(f"Seřazené: {math_utils.bubble_sort(cisla)}")
```

## 8.3 Způsoby importu

### 8.3.1 1. Import celého modulu

```
import math_utils

math_utils.fibonacci(10)
math_utils.faktorial(5)
```

### 8.3.2 2. Import konkrétních funkcí

```
from math_utils import fibonacci, faktorial

fibonacci(10)    # Bez math_utils.
faktorial(5)
```

### 8.3.3 3. Import s aliasem

```
import math_utils as mu

mu.fibonacci(10)
mu.faktorial(5)
```

#### 8.3.4 4. Import všeho (NEDOPORUČENO!)

```
from math_utils import *

fibonacci(10) # Funguje, ale není jasné, odkud funkce pochází
```

##### Doporučení

- Pro **vlastní moduly**: `import math_utils` (jasné, odkud funkce pochází)
- Pro **specifické funkce**: `from math_utils import fibonacci` (kratší kód)
- **Nikdy** nepoužívejte `import *` - není jasné, co importujete!

### 8.4 Vestavěné moduly

Python má mnoho **vestavěných modulů**, které můžete rovnou použít:

#### 8.4.1 Modul `math`

```
import math

print(math.sqrt(16))      # 4.0 - odmocnina
print(math.pi)           # 3.14159... - číslo
print(math.ceil(3.2))    # 4 - zaokrouhlení nahoru
print(math.floor(3.8))   # 3 - zaokrouhlení dolů
```

#### 8.4.2 Modul `random`

```
import random

print(random.randint(1, 10))      # Náhodné číslo 1-10
print(random.choice([1, 2, 3, 4])) # Náhodný prvek ze seznamu
```

### 8.4.3 Modul os (operační systém)

```
import os

print(os.getcwd())          # Aktuální složka
print(os.listdir('.'))      # Seznam souborů
```

### 8.4.4 Modul csv (práce s CSV)

```
import csv

# Budeme používat příští lekci!
```

## 8.5 Preview: import arcpy

**i** Připravujeme se na ArcPy!

V příštích týdnech budete psát:

```
import arcpy

# Funkce z ArcPy modulu:
arcpy.Buffer_analysis(...)
arcpy.Clip_analysis(...)
arcpy.management.CreateFeatureclass(...)
```

**Vidíte?** ArcPy je jen další modul! Funguje úplně stejně jako `math_utils`.

---

## 9 Struktura projektu

### 9.1 Ideální organizace

muj\_projekt/



```
main.py          # Hlavní program
math_utils.py    # Matematické funkce

data/
  data.csv       # Data (příští lekce)
```

**main.py:**

```
import math_utils

# Hlavní program...
```

**math\_utils.py:**

```
def fibonacci(n):
    # ...

def faktorial(n):
    # ...
```

! Soubory musí být ve stejné složce!

Aby `import math_utils` fungoval, musí být `math_utils.py` ve **stejné složce** jako `main.py`, nebo v Pythonem rozpoznané cestě.

---

## 10 Praktická úloha

### 10.1 Zadání

Vytvořte kompletní projekt:

#### 10.1.1 1. Soubor `math_utils.py`

Obsahuje funkce: - `fibonacci(n)` - `faktorial(n)` - `je_prvocislo(n)` - `bubble_sort(seznam)`

Všechny funkce mají docstringy!

### 10.1.2 2. Soubor main.py

Program, který: 1. Načte číslo N od uživatele 2. Vypočítá a vypíše: - N-té Fibonacci číslo - Faktoriál N - Zda je N prvočíslo 3. Vytvoří seznam prvních 10 Fibonacci čísel 4. Seřadí tento seznam (ačkoli už je seřazený ) 5. Vypíše výsledek

**Kostra:**

```
# main.py
import math_utils

n = int(input("Zadej číslo: "))

# Základní výpočty
print(f"Fibonacci({n}) = {math_utils.fibonacci(n)}")
print(f"Faktoriál({n}) = {math_utils.faktorial(n)}")

if math_utils.je_prvocislo(n):
    print(f"{n} je prvočíslo")
else:
    print(f"{n} není prvočíslo")

# Seznam Fibonacci čísel
fibonacci_cisla = []
for i in range(10):
    fibonacci_cisla.append(math_utils.fibonacci(i))

print(f"\nPrvních 10 Fibonacci čísel: {fibonacci_cisla}")

# Seřazení (už je seřazený, ale ukážeme funkci)
serazeny = math_utils.bubble_sort(fibonacci_cisla)
print(f"Seřazený seznam: {serazeny}")
```

---

## 11 Shrnutí

### 11.1 Co jsme se naučili

Definice funkcí pomocí def

Parametry - vstupy do funkce

**return** - návratové hodnoty  
**Rozdíl print() vs. return** - klíčové pro pochopení!  
**Docstrings** - dokumentace funkcí  
**Moduly** - organizace kódu do souborů  
**import** - použití modulů  
**Vestavěné moduly** - math, random, os, csv  
**Preview ArcPy** - import arcpy funguje stejně!

## 11.2 Co bude příště?

V příští lekci:

- Čtení textových souborů - `open()`, `read()`, `readlines()`
  - Zápis do souborů - vytváření nových souborů
  - Zpracování jednoduchých dat - např. seznam měst s populací
  - Použití funkcí z `math_utils` pro zpracování dat
- 

## 12 Domácí úkol

### 12.1 Varianta A (základní)

1. Dokončete `math_utils.py` s všemi čtyřmi funkcemi
2. Vytvořte `main.py`, který používá všechny funkce
3. Přidejte docstringy ke všem funkcím

### 12.2 Varianta B (pokročilá)

1. Rozšiřte `math_utils.py` o nové funkce:
  - `selection_sort(seznam)` - druhý třídící algoritmus
  - `najdi_prvocisla(n)` - seznam všech prvočísel do n
  - `n_te_prvocislo(n)` - najde n-té prvočíslo
2. Otestujte všechny nové funkce v `main.py`

## 12.3 Varianta C (výzva)

Vytvořte modul `statistika.py` s funkcemi:

```
def prumer(seznam):
    """Vrátí průměr čísel v seznamu."""
    # ...

def median(seznam):
    """Vrátí medián seznamu (prostřední hodnota)."""
    # ...

def maximum(seznam):
    """Vrátí maximum ze seznamu."""
    # ...

def minimum(seznam):
    """Vrátí minimum ze seznamu."""
    # ...
```

**Bonus:** Můžete použít `math_utils.bubble_sort()` v `median()`!

---

## 13 Cheatsheet

```
# === DEFINICE FUNKCE ===
def jmeno_funkce(parametr1, parametr2):
    """Dokumentační řetězec."""
    # Tělo funkce
    return vysledek

# === VOLÁNÍ FUNKCE ===
vysledek = jmeno_funkce(hodnota1, hodnota2)

# === DEFAULT PARAMETRY ===
def pozdrav(jmeno, jazyk="cs"):
    # ...

pozdrav("Jan")          # Použije default
pozdrav("John", "en")   # Přepíše default

# === DOCSTRING ===
def funkce(x):
    """Stručný popis.

    Args:
        x: Popis parametru

    Returns:
        Popis návratové hodnoty
    """
    return x * 2

# === MODULY ===
# Vytvoření: uložte funkce do souboru.py

# Import celého modulu:
import math_utils
math_utils.fibonacci(10)

# Import konkrétní funkce:
from math_utils import fibonacci
fibonacci(10)
```

```
# Import s aliasem:
import math_utils as mu
mu.fibonacci(10)

# === VESTAVĚNÉ MODULY ===
import math
math.sqrt(16)
math.pi

import random
random.randint(1, 10)

import os
os.getcwd()

# === PRINT vs RETURN ===
def spatne(x):
    print(x * 2)    # Vypíše, ale nevrací!

def spravne(x):
    return x * 2    # Vrací hodnotu!

a = spatne(5)      # a = None
b = spravne(5)     # b = 10
```

---

## 14 Poznámky pro vyučujícího

### 14.1 Běžné chyby studentů

```
# 1. Zapomínají return
def secti(a, b):
    vysledek = a + b
    # Zapomněli return!

x = secti(5, 3) # x = None

# 2. Používají print místo return
def secti(a, b):
    print(a + b) # Špatně!

vysledek = secti(5, 3) + 10 # Chyba!

# 3. Volají funkci s def
def fibonacci(n) # Zapomněli dvojtečku!

# 4. Zapomínají závorky při volání
fibonacci # Toto je funkce samotná, ne výsledek!
fibonacci(10) # Správně - volání

# 5. Špatný import
import math_utils
fibonacci(10) # Chyba - musí být math_utils.fibonacci(10)

# Nebo:
from math_utils import fibonacci
math_utils.fibonacci(10) # Chyba - už není potřeba math_utils.

# 6. Soubory v jiných složkách
# main.py je v C:\Users\Jan\projekt\
# math_utils.py je v C:\Users\Jan\Desktop\
# import math_utils # NEFUNGUJE!
```

### 14.2 Časový plán (90 min)

Čas	Obsah
0-10 min	Motivace - proč funkce? DRY princip
10-30 min	Definice funkcí, parametry, return
30-40 min	print() vs. return - DŮLEŽITÉ!
40-50 min	Docstrings, přepis Fibonacci/faktoriál
50-60 min	Cvičení - přepis prvočísla a třídění
60-75 min	Moduly - vytvoření math_utils.py
75-85 min	Import, použití v main.py
85-90 min	Preview ArcPy, zadání úkolu

## 14.3 Klíčové momenty

### 14.3.1 print() vs. return (30-40 min):

- **NEJDŮLEŽITĚJŠÍ KONCEPT** v této lekci!
- Studenti často nerozumí rozdílů
- Ukázat konkrétní příklad, kde print() selhává:

```
def spatne(x):
    print(x * 2)

vysledek = spatne(5) + 10 # TypeError!
```

- Vysvětlit: “print je pro LIDI, return je pro PROGRAM”

### 14.3.2 Moduly (60-85 min):

- Prakticky vytvořit math\_utils.py společně
- Ukázat, že musí být ve stejné složce
- Otestovat import v main.py
- Zdůraznit: “ArcPy bude fungovat úplně stejně!”

### 14.3.3 Docstrings (40-50 min):

- Nemusí být dokonalé, ale alespoň jednořádkové
- Ukázat help(funkce)
- Zdůraznit, že v praxi je dokumentace důležitá



## 14.4 Rizika

1. **print() vs. return může být matoucí (může trvat 15 min)**
  - Řešení: Věnovat tomu dostatek času, ukázat konkrétní příklady
  - Nechat studenty experimentovat
2. **Import nemusí fungovat (chyby s cestami)**
  - Řešení: Zdůraznit, že soubory musí být ve stejné složce
  - Mít připravené screenshoty struktury složek
3. **Studenti budou různě rychlí s přepisem funkcí**
  - Řešení: Rychlejší pomáhají pomalejším
  - Můžete poskytnout kostru `math_utils.py`

## 14.5 Tipy

- **print() vs. return je nejčastější problém** - věnujte tomu extra čas!
- Na konci vždy zdůrazněte: “ArcPy je jen modul - budete psát `import arcpy` stejně jako `import math_utils`”
- Ukažte `help()` funkci - je užitečná pro čtení dokumentace
- Vytvořte `math_utils.py` společně na projektoru - ať vidí celý proces
- Nechte je otestovat všechny funkce v `main.py`

## 14.6 Materiály k přípravě

- ☐ Připravit kostru `math_utils.py` (pro pomalé studenty)
- ☐ Ukázková `main.py` s použitím všech funkcí
- ☐ Diagram struktury projektu (složky a soubory)
- ☐ Cheatsheet pro různé způsoby importu