

# Lekce 4: Podmínky a cykly

Python pro GIS - Rozhodování a opakování

Vojtěch Barták, FŽP ČZU Praha

2025-11-24

## Table of contents

<b>1</b>	<b>Cíle lekce</b>	<b>4</b>
<b>2</b>	<b>Logické hodnoty a porovnávání</b>	<b>4</b>
2.1	Co už znáte . . . . .	4
2.2	Porovnávací operátory . . . . .	4
2.2.1	Přehled operátorů: . . . . .	5
2.3	Logické operátory . . . . .	5
2.3.1	and (a zároveň) . . . . .	5
2.3.2	or (nebo) . . . . .	5
2.3.3	not (negace) . . . . .	6
2.3.4	Kombinace operátorů . . . . .	6
2.4	Praktické příklady . . . . .	6
<b>3</b>	<b>2. Podmínky - if, elif, else</b>	<b>6</b>
3.1	Základní if . . . . .	7
3.2	if-else . . . . .	7
3.3	if-elif-else . . . . .	7
3.4	Vnořené podmínky . . . . .	8
3.5	Cvičení 1: Kategorizace čísla . . . . .	9
<b>4</b>	<b>3. Cyklus for</b>	<b>9</b>
4.1	Procházení seznamu . . . . .	9
4.2	Funkce range() . . . . .	9
4.2.1	range(n) - od 0 do n-1 . . . . .	10
4.2.2	range(start, stop) - od start do stop-1 . . . . .	10
4.2.3	range(start, stop, step) - s krokem . . . . .	10
4.2.4	Další příklady . . . . .	11

4.3	Akumulace v cyklu . . . . .	11
4.3.1	Součet čísel . . . . .	11
4.3.2	Součin čísel (faktoriál) . . . . .	11
4.4	Cvičení 2: Druhé mocniny . . . . .	12
<b>5</b>	<b>4. Cyklus while</b>	<b>12</b>
5.1	Základní syntaxe . . . . .	12
5.2	Kdy použít for vs. while? . . . . .	13
5.2.1	for - když víme předem, kolikrát opakovat . . . . .	13
5.2.2	while - když opakujeme, dokud platí podmínka . . . . .	13
5.3	Příklad: Hádání čísla . . . . .	14
5.4	break a continue . . . . .	14
5.4.1	break - okamžité ukončení cyklu . . . . .	14
5.4.2	continue - přeskočení zbytku iterace . . . . .	14
<b>6</b>	<b>5. Praktická úloha: Fibonacciho posloupnost</b>	<b>15</b>
6.1	Co je Fibonacciho posloupnost? . . . . .	15
6.2	Implementace krok za krokem . . . . .	15
6.2.1	Krok 1: Načtení vstupu . . . . .	15
6.2.2	Krok 2: Inicializace . . . . .	15
6.2.3	Krok 3: Cyklus . . . . .	15
6.2.4	Krok 4: Výpis výsledku . . . . .	16
6.3	Celý program . . . . .	16
6.4	Test programu . . . . .	16
<b>7</b>	<b>6. Samostatná úloha: Faktoriál</b>	<b>17</b>
7.1	Zadání . . . . .	17
7.2	Úkol . . . . .	17
7.3	Řešení . . . . .	17
<b>8</b>	<b>7. Shrnutí</b>	<b>18</b>
8.1	Co jsme se naučili . . . . .	18
8.2	Co bude příště? . . . . .	18
<b>9</b>	<b>8. Domácí úkol</b>	<b>18</b>
9.1	Varianta A (základní) . . . . .	18
9.2	Varianta B (pokročilá) . . . . .	19
9.3	Bonusový úkol (nepovinný) . . . . .	19
<b>10</b>	<b>9. Cheatsheet</b>	<b>20</b>
<b>11</b>	<b>Poznámky pro vyučujícího</b>	<b>22</b>
11.1	Běžné chyby studentů . . . . .	22
11.2	Časový plán (90 min) . . . . .	22

11.3	Klíčové momenty . . . . .	23
11.3.1	Fibonacci (65-80 min): . . . . .	23
11.3.2	Faktoriál (80-90 min): . . . . .	23
11.4	Rizika . . . . .	23
11.5	Tipy . . . . .	24

# 1 Cíle lekce

Po absolvování této lekce budete umět:

- Používat porovnávací operátory (`==`, `<`, `>`, ...)
- Používat logické operátory (`and`, `or`, `not`)
- Psát podmíněné příkazy (`if`, `elif`, `else`)
- Používat cyklus `for` s funkcí `range()`
- Používat cyklus `while`
- Implementovat algoritmus pro výpočet Fibonacciho posloupnosti a faktoriálu

Časová dotace: 90 minut

---

## 2 Logické hodnoty a porovnávání

### 2.1 Co už znáte

Z minulé lekce už znáte datový typ `bool` (boolean) s hodnotami `True` a `False`:

```
je_student = True
je_zamestnanec = False
```

Dnes se naučíme, jak tyto hodnoty **vytvářet porovnáváním** a jak je **používat pro rozhodování**.

### 2.2 Porovnávací operátory


Porovnáním dvou hodnot získáme logickou hodnotu (`True` nebo `False`):

```
vek = 18

vek >= 18 # True (je větší nebo rovno 18?)
vek == 21 # False (je přesně 21?)
vek < 15  # False (je menší než 15?)
```

### 2.2.1 Přehled operátorů:

Operátor	Význam	Příklad	Výsledek
==	rovná se	5 == 5	True
!=	nerovná se	5 != 3	True
<	menší než	3 < 5	True
>	větší než	5 > 3	True
<=	menší nebo rovno	5 <= 5	True
>=	větší nebo rovno	6 >= 5	True

 Pozor na == vs. =

- = je **přiřazení** hodnoty: vek = 18
- == je **porovnání**: vek == 18

Toto je častá chyba začátečníků!

## 2.3 Logické operátory

Pomocí logických operátorů můžeme kombinovat více podmínek:

### 2.3.1 and (a zároveň)

Obě podmínky musí být pravdivé:

```
vek = 25  
  
vek >= 18 and vek < 65 # True - je dospělý a není důchodce
```

### 2.3.2 or (nebo)

Alespoň jedna podmínka musí být pravdivá:

```
teplota = -5  
  
teplota < 0 or teplota > 35 # True - extrémní počasí
```

### 2.3.3 not (negace)

Obrací pravdivostní hodnotu:

```
je_prazdny = False  
  
not je_prazdny # True - není prázdný
```

### 2.3.4 Kombinace operátorů

```
vek = 20  
ma_ridicak = True  
  
# Může řídit auto?  
vek >= 18 and ma_ridicak # True  
  
# Je dítě nebo senior?  
vek < 15 or vek >= 65 # False
```

## 2.4 Praktické příklady

```
# Kontrola rozsahu  
cislo = 50  
cislo >= 0 and cislo <= 100 # Je číslo mezi 0 a 100?  
  
# Kontrola sudosti  
cislo % 2 == 0 # Je číslo sudé?  
  
# Kontrola dělitelnosti  
cislo % 3 == 0 # Je číslo dělitelné třemi?
```

---

## 3 2. Podmínky - if, elif, else

Podmínky umožňují programu **rozhodovat se** na základě splnění nějaké podmínky.

### 3.1 Základní if

Pokud je podmínka pravdivá, provede se odsazený kód:

```
vek = int(input("Váš věk: "))

if vek >= 18:
    print("Můžete řídit auto")
```

! Odsazování je POVINNÉ!

Python používá **odsazení (4 mezery = 1 tabulátor)** k označení bloků kódu. Špatné odsazení způsobí chybu!

```
if vek >= 18:
print("Chyba!") # CHYBA - není odsazené!
```

### 3.2 if-else

Co když chceme něco provést v případě, že podmínka **není** splněna?

```
vek = int(input("Váš věk: "))

if vek >= 18:
    print("Dospělý")
else:
    print("Dítě")
```

### 3.3 if-elif-else

Pro více možností použijeme **elif** (else if):

```
vek = int(input("Váš věk: "))

if vek < 15:
    print("Vstup zdarma")
elif vek < 65:
    print("Plné vstupné: 150 Kč")
else:
    print("Seniorské vstupné: 80 Kč")
```

**Jak to funguje:** 1. Zkontroluje první podmínku (`vek < 15`) 2. Pokud není splněna, zkontroluje další (`vek < 65`) 3. Pokud žádná není splněna, provede `else`

💡 Můžete mít více `elif`

```
if znamka == 1:
    print("Výborně")
elif znamka == 2:
    print("Chvalitebně")
elif znamka == 3:
    print("Dobře")
elif znamka == 4:
    print("Dostatečně")
else:
    print("Nedostatečně")
```

### 3.4 Vnořené podmínky

Podmínky můžete vnořovat do sebe:

```
vek = int(input("Věk: "))
ma_ridicak = input("Máte řidičák? (ano/ne): ") == "ano"

if vek >= 18:
    if ma_ridicak:
        print("Můžete řídit")
    else:
        print("Musíte udělat řidičák")
else:
    print("Jste příliš mladí")
```

Ale elegantněji pomocí `and`:

```
if vek >= 18 and ma_ridicak:
    print("Můžete řídit")
elif vek >= 18:
    print("Musíte udělat řidičák")
else:
    print("Jste příliš mladí")
```



### 3.5 Cvičení 1: Kategorizace čísla

Napište program, který: 1. Načte číslo od uživatele 2. Rozhodne, zda je: - Kladné, záporné nebo nula - Sudé nebo liché (pokud není nula)

**Nápověda:**

```
cislo = int(input("Zadejte číslo: "))  
  
# Zde přidejte podmínky...
```

---

## 4 3. Cyklus for

Cykly umožňují **opakovat kód** vícekrát bez nutnosti ho psát znovu.

### 4.1 Procházení seznamu

Nejjednodušší použití - projít všechny prvky seznamu:

```
mesta = ["Praha", "Brno", "Ostrava"]  
  
for mesto in mesta:  
    print(f"Město: {mesto}")
```

**Výsledek:**

```
Město: Praha  
Město: Brno  
Město: Ostrava
```

**Jak to funguje:** - Proměnná `mesto` postupně nabývá hodnot "Praha", "Brno", "Ostrava" - Pro každou hodnotu se provede odsazený kód

### 4.2 Funkce range()

Pro opakování N-krát použijeme funkci `range()`:

#### 4.2.1 range(n) - od 0 do n-1

```
for i in range(5):  
    print(i)
```

Výsledek:

0  
1  
2  
3  
4

⚠ range(5) končí na 4, ne na 5!

Python počítá od 0, takže range(5) znamená: 0, 1, 2, 3, 4

#### 4.2.2 range(start, stop) - od start do stop-1

```
for i in range(2, 6):  
    print(i)
```

Výsledek:

2  
3  
4  
5

#### 4.2.3 range(start, stop, step) - s krokem

```
for i in range(0, 10, 2):  
    print(i)
```

Výsledek:

0  
2  
4  
6  
8

#### 4.2.4 Další příklady

```
# Zpětně (od 10 do 1)
for i in range(10, 0, -1):
    print(i)

# Od 1 do 10 (často potřebujeme)
for i in range(1, 11):
    print(i)
```

### 4.3 Akumulace v cyklu

Často potřebujeme v cyklu **sbírat výsledky** (akumulovat):

#### 4.3.1 Součet čísel

```
soucet = 0 # Inicializace

for i in range(1, 11):
    soucet = soucet + i # Přičtení k součtu

print(f"Součet čísel 1-10: {soucet}") # 55
```

Kratší zápis: `soucet += i` je totéž jako `soucet = soucet + i`

#### 4.3.2 Součin čísel (faktoriál)

```
soucin = 1 # DŮLEŽITÉ: inicializovat na 1, ne 0!

for i in range(1, 6):
    soucin = soucin * i # Vynásobení

print(f"5! = {soucin}") # 120
```

## 4.4 Cvičení 2: Druhé mocniny

Napište program, který vytvoří seznam druhých mocnin čísel od 1 do 10.

**Nápověda:**

```
mocniny = [] # Prázdný seznam

for i in range(1, 11):
    # Přidejte druhou mocninu do seznamu...
```

---

## 5 4. Cyklus while

Cyklus while opakuje kód, **dokud platí podmínka**.

### 5.1 Základní syntaxe

```
pocitadlo = 0

while pocitadlo < 5:
    print(pocitadlo)
    pocitadlo = pocitadlo + 1
```

**Výsledek:**

0  
1  
2  
3  
4

 Pozor na nekonečný cyklus!

Pokud podmínka nikdy nepřestane platit, cyklus poběží donekonečna:

```
pocitadlo = 0
while pocitadlo < 5:
    print(pocitadlo)
    # CHYBA - zapomněli jsme zvýšit pocitadlo!
```

Program musíte přerušit: **Ctrl+C**

## 5.2 Kdy použít for vs. while?

### 5.2.1 for - když víme předem, kolikrát opakovat

```
# Chci 10x opakovat něco
for i in range(10):
    print("Opakování")

# Chci projít seznam
for prvek in seznam:
    print(prvek)
```

### 5.2.2 while - když opakujeme, dokud platí podmínka

```
# Opakuj, dokud uživatel nezadá správně
heslo = ""
while heslo != "tajne":
    heslo = input("Zadejte heslo: ")

print("Správně!")
```

### 5.3 Příklad: Hádání čísla

```
tajne_cislo = 42
tip = 0

while tip != tajne_cislo:
    tip = int(input("Hádej číslo: "))

    if tip < tajne_cislo:
        print("Větší!")
    elif tip > tajne_cislo:
        print("Menší!")

print("Správně!")
```

### 5.4 break a continue

#### 5.4.1 break - okamžité ukončení cyklu

```
for i in range(100):
    if i == 5:
        break # Ukončí cyklus
    print(i)

# Vypíše: 0, 1, 2, 3, 4
```

#### 5.4.2 continue - přeskočení zbytku iterace

```
for i in range(10):
    if i % 2 == 0:
        continue # Přeskoč sudá čísla
    print(i)

# Vypíše: 1, 3, 5, 7, 9
```

## 6 5. Praktická úloha: Fibonacciho posloupnost

### 6.1 Co je Fibonacciho posloupnost?

Fibonacciho posloupnost je sekvence čísel, kde každé číslo je **součtem dvou předchozích**:

$$F(0) = 0$$

$$F(1) = 1$$

$$F(n) = F(n-1) + F(n-2)$$

Posloupnost: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, ...

**Příklady:** -  $F(2) = F(1) + F(0) = 1 + 0 = 1$  -  $F(3) = F(2) + F(1) = 1 + 1 = 2$  -  $F(4) = F(3) + F(2) = 2 + 1 = 3$  -  $F(5) = F(4) + F(3) = 3 + 2 = 5$

### 6.2 Implementace krok za krokem

#### 6.2.1 Krok 1: Načtení vstupu

```
n = int(input("Který člen Fibonacciho posloupnosti? "))
```

#### 6.2.2 Krok 2: Inicializace

Potřebujeme dvě proměnné pro dva předchozí členy:

```
a = 0 # F(0)
```

```
b = 1 # F(1)
```

#### 6.2.3 Krok 3: Cyklus

V každém kroku posuneme hodnoty:

```
for i in range(n):  
    a, b = b, a + b
```

**Co se děje:** -  $a, b = b, a + b$  je **současné přiřazení** - Nejprve se vypočítá pravá strana: nové\_a = b, nové\_b = a + b - Pak se přiřadí: a = nové\_a, b = nové\_b

**Příklad pro n=5:**

Krok 0:  $a=0, b=1 \rightarrow a=1, b=0+1=1$   
Krok 1:  $a=1, b=1 \rightarrow a=1, b=1+1=2$   
Krok 2:  $a=1, b=2 \rightarrow a=2, b=1+2=3$   
Krok 3:  $a=2, b=3 \rightarrow a=3, b=2+3=5$   
Krok 4:  $a=3, b=5 \rightarrow a=5, b=3+5=8$

#### 6.2.4 Krok 4: Výpis výsledku

```
print(f"{n}. člen Fibonacciho posloupnosti: {a}")
```

### 6.3 Celý program

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

### 6.4 Test programu

Který člen Fibonacciho posloupnosti? 10  
10. člen: 55

#### Proč to funguje?

Kouzlo je v `a, b = b, a + b`. Python nejprve vyhodnotí celou pravou stranu, pak přiřadí:

```
# Špatně (nefunguje):  
a = b  
b = a + b # Tady už je a změněné!  
  
# Správně (funguje):  
a, b = b, a + b # Python přiřadí obě hodnoty najednou
```



---

## 7 6. Samostatná úloha: Faktoriál

### 7.1 Zadání

**Faktoriál** čísla  $n$  (označuje se  $n!$ ) je součin všech přirozených čísel od 1 do  $n$ :

$$n! = n \times (n-1) \times (n-2) \times \dots \times 2 \times 1$$

Příklady:

$$0! = 1 \quad (\text{definice})$$

$$1! = 1$$

$$2! = 2 \times 1 = 2$$

$$3! = 3 \times 2 \times 1 = 6$$

$$4! = 4 \times 3 \times 2 \times 1 = 24$$

$$5! = 5 \times 4 \times 3 \times 2 \times 1 = 120$$

### 7.2 Úkol

Napište program, který: 1. Načte číslo  $n$  od uživatele 2. Vypočítá faktoriál tohoto čísla 3. Vypíše výsledek

**Nápověda:** - Použijte cyklus `for` s `range()` - Začněte s `vysledek = 1` (DŮLEŽITÉ - ne 0!) - V každém kroku násobte - Rozmyslete si správný rozsah pro `range()`

**Kostra programu:**

```
n = int(input("Zadejte číslo: "))

vysledek = 1 # Inicializace

# Zde přidejte cyklus...

print(f"{n}! = {vysledek}")
```

### 7.3 Řešení

---

## 8 7. Shrnutí

### 8.1 Co jsme se naučili

**Porovnávací operátory:** ==, !=, <, >, <=, >=

**Logické operátory:** and, or, not

**Podmínky:** if, elif, else

**For cyklus:** procházení seznamů, funkce range()

**While cyklus:** opakování dokud platí podmínka

**break a continue:** ovládání cyklů

**Fibonacci:** iterativní algoritmus s akumulací

**Faktoriál:** součin čísel pomocí cyklu

### 8.2 Co bude příště?

V příští lekci:

- **Rozšíření práce se seznamy** (slicing, metody)
  - **Prvočísla** - složitější podmínky v cyklech
  - **Třídění** - vnořené cykly a algoritmické myšlení
  - **Propojení s Model Builder** - vnořené modely vs. vnořené cykly
- 

## 9 8. Domácí úkol

### 9.1 Varianta A (základní)

1. **Dokončete faktoriál** (pokud jste ho nestihli)
2. **Tabulka faktoriálů:** Vypište faktoriály čísel 1-10 ve formátu:

1! = 1

2! = 2

3! = 6

...

10! = 3628800

## 9.2 Varianta B (pokročilá)

1. **Fibonacci pro seznam:** Vypočítejte Fibonacci čísla pro seznam [5, 10, 15, 20]
2. **Největší Fibonacci pod 1000:** Najděte největší Fibonacci číslo menší než 1000

Nápověda pro B2:

```
a, b = 0, 1

while b < 1000:
    # Co dál?
```

## 9.3 Bonusový úkol (nepovinný)

**Palindrom:** Napište program, který zjistí, zda je číslo palindrom (čte se stejně zepředu i zezadu).

Příklady: 121, 1331, 12321

**Nápověda:** Převeďte číslo na string a porovnejte ho s obráceným stringem.

---

## 10 9. Cheatsheet

```
# === POROVNÁVACÍ OPERÁTORY ===
==      # rovná se
!=      # nerovná se
<       # menší než
>       # větší než
<=      # menší nebo rovno
>=      # větší nebo rovno

# === LOGICKÉ OPERÁTORY ===
and     # a zároveň (obě podmínky musí platit)
or      # nebo (alespoň jedna musí platit)
not     # negace (obrací True/False)

# === PODMÍNKY ===
if podminka:
    prikaz
elif jina_podminka:
    jiny_prikaz
else:
    alternativni_prikaz

# === FOR CYKLUS ===
for prvek in seznam:
    prikaz

for i in range(10):          # 0-9
    prikaz

for i in range(1, 11):       # 1-10
    prikaz

for i in range(0, 10, 2):     # 0,2,4,6,8
    prikaz

# === WHILE CYKLUS ===
while podminka:
    prikaz
    # nezapomenout změnit podmínku!
```

```
# === OVLÁDÁNÍ CYKLŮ ===
break      # Ukončí cyklus
continue   # Přeskočí zbytek iterace

# === AKUMULACE ===
soucet = 0
for i in range(1, 11):
    soucet += i # soucet = soucet + i

soucin = 1
for i in range(1, 6):
    soucin *= i # soucin = soucin * i

# === SOUČASNÉ PŘÍŘAZENÍ ===
a, b = b, a + b # Fibonacci výměna
```

---

## 11 Poznámky pro vyučujícího

### 11.1 Běžné chyby studentů

```
# 1. Záměna = a ==
if vek = 18:      # CHYBA - přiřazení místo porovnání
if vek == 18:    # SPRÁVNĚ

# 2. Chybějící odsazení
if vek >= 18:
print("OK")      # CHYBA - není odsazené
    print("OK")  # SPRÁVNĚ

# 3. Zapomenuté inicializace
for i in range(10):
    soucet += i  # CHYBA - soucet neexistuje

soucet = 0       # SPRÁVNĚ - nejprve inicializovat
for i in range(10):
    soucet += i

# 4. Špatný range
range(10)        # 0-9 (ne 1-10!)
range(1, 11)     # 1-10 (správně pro součet 1-10)

# 5. Nekonečný while
while True:      # CHYBA - nikdy neskončí
    print("...")

# 6. Fibonacci výměna
a = b
b = a + b        # CHYBA - a už je změněné!

a, b = b, a + b  # SPRÁVNĚ - současné přiřazení
```

### 11.2 Časový plán (90 min)

Čas	Obsah
0-15 min	Logické hodnoty, porovnávání

Čas	Obsah
15-35 min	Podmínky (if/elif/else) + cvičení
35-55 min	For cykly, range() + cvičení
55-65 min	While cykly, break/continue
65-80 min	Fibonacci - společně krok za krokem
80-90 min	Faktoriál - samostatně (nebo začít doma)

## 11.3 Klíčové momenty

### 11.3.1 Fibonacci (65-80 min):

- **DŮLEŽITÉ:** Procházet KROK ZA KROKEM
- Na tabuli/projektoru ukázat hodnoty **a** a **b** v každé iteraci
- Vysvětlit **a**, **b = b**, **a + b** pečlivě
- Nechat studenty říct, co se stane v dalším kroku

### 11.3.2 Faktoriál (80-90 min):

- Pokud nestihnou, je to OK - je to domácí úkol
- Důležitější je, aby rozuměli Fibonacci
- Pokud stihnou, ukázat řešení a porovnat for vs. while

## 11.4 Rizika

### 1. Fibonacci může trvat déle (15 min → 20 min)

- Řešení: Zkrátit while cykly (10 min → 5 min)
- Mít připravený promítaný kód

### 2. Studenti budou různě rychlí

- Řešení: Rychlejší pomáhají pomalejším
- Bonusové úkoly pro rychlé

### 3. Faktoriál nestihnou

- Řešení: To je v pořádku, dokončí doma
- Hlavní cíl = pochopit Fibonacci

## 11.5 Tipy

- **Fibonacci výměna je nejdůležitější koncept** - věnovat tomu čas!
- Ukázat na tabuli: “Co bude v **a** a **b** po každém kroku?”
- Nechat studenty předpovídat další krok
- Faktoriál je jednodušší - můžou zvládnout sami