

# Lekce 5: Tvorba jednoduchého algoritmu

Python pro GIS - Algoritmické myšlení

Vojtěch Barták, FŽP ČZU Praha

2025-11-19

## Table of contents

<b>1 Cíle lekce</b>	<b>3</b>
<b>2 Rekapitulace a rozšíření seznamů</b>	<b>3</b>
2.1 Rychlá rekapitulace z minula . . . . .	3
2.2 Slicing (řezy) - nová technika . . . . .	3
2.2.1 Základní syntaxe: <code>seznam[start:stop]</code> . . . . .	3
2.2.2 Slicing s krokem: <code>seznam[start:stop:step]</code> . . . . .	4
2.2.3 Praktické příklady . . . . .	4
2.3 Užitečné metody seznamů . . . . .	5
2.3.1 <code>extend()</code> - připojení více prvků . . . . .	5
2.3.2 <code>insert()</code> - vložení na konkrétní pozici . . . . .	5
2.3.3 <code>remove()</code> - odstranění prvního výskytu hodnoty . . . . .	5
2.3.4 <code>index()</code> - vrací index prvního výskytu hodnoty . . . . .	5
2.3.5 <code>pop()</code> - odstranění a vrácení prvku . . . . .	5
2.3.6 <code>reverse()</code> - otočení seznamu . . . . .	6
2.3.7 <code>sort()</code> - seřazení . . . . .	6
2.4 Cvičení 1: Práce se seznamem . . . . .	6
<b>3 Hledání minima a maxima</b>	<b>7</b>
3.1 Motivace . . . . .	7
3.2 Algoritmus krok za krokem . . . . .	7
3.3 Implementace - společně . . . . .	7
3.4 Maximum - obdobně . . . . .	8
3.5 Hledání indexu minima . . . . .	8
<b>4 Třídění</b>	<b>9</b>
4.1 Motivace . . . . .	9

4.2	Bubble Sort (bublinové třídění) . . . . .	9
4.2.1	Princip . . . . .	9
4.2.2	Vizualizace na příkladu [5, 2, 8, 1, 9] . . . . .	10
4.2.3	Implementace - společně . . . . .	10
4.2.4	Alternativní implementace (volitelně) . . . . .	11
4.2.5	Proč <code>len(seznam) - 1 - i?</code> . . . . .	12
4.3	Selection Sort (třídění výběrem) . . . . .	13
4.3.1	Princip . . . . .	13
4.3.2	Vizualizace krok za krokem . . . . .	13
4.3.3	Implementace - společně . . . . .	14
<b>5</b>	<b>Vnořené cykly a propojení s Model Builderem</b>	<b>15</b>
5.1	Co jsou vnořené cykly? . . . . .	15
5.2	Praktický příklad: Tabulka násobení . . . . .	16
5.3	Propojení s Model Builderem . . . . .	16
5.3.1	Příklad: GIS úloha . . . . .	16
<b>6</b>	<b>Shrnutí</b>	<b>17</b>
6.1	Co jsme se naučili . . . . .	17
6.2	Co bude příště? . . . . .	17
<b>7</b>	<b>Domácí úkol</b>	<b>17</b>
7.1	Varianta A (základní) . . . . .	17
7.2	Varianta B (pokročilá) . . . . .	18
7.3	Varianta C (výzva) . . . . .	18
<b>8</b>	<b>Cheatsheet</b>	<b>19</b>
<b>9</b>	<b>Poznámky pro vyučujícího</b>	<b>21</b>
9.1	Běžné chyby studentů . . . . .	21
9.2	Časový plán (90 min) . . . . .	21
9.3	Klíčové momenty . . . . .	21
9.3.1	Seznamy (5-20 min): . . . . .	21
9.3.2	Minimum/Maximum (20-45 min): . . . . .	22
9.3.3	Třídění (45-85 min): . . . . .	22
9.4	Rizika a řešení . . . . .	22
9.5	Tipy . . . . .	22

## 1 Cíle lekce

Po absolvování této lekce budete umět:

- Pracovat s řezy seznamů (slicing)
- Používat pokročilé metody seznamů
- Implementovat algoritmus pro hledání minima a maxima
- Navrhnout a implementovat třídící algoritmus (Bubble Sort, Selection Sort)
- Pracovat s vnořenými cykly
- Propojit vnořené cykly s vnořenými modely z Model Builderu

**Časová dotace:** 90 minut

---

## 2 Rekapitulace a rozšíření seznamů

### 2.1 Rychlá rekapitulace z minula

```
# Co už umíme ze seznamů:  
cisla = [1, 2, 3, 4, 5]  
  
cisla[0]           # 1 - první prvek (index 0!)  
cisla[-1]          # 5 - poslední prvek  
len(cisla)         # 5 - délka seznamu  
cisla.append(6)    # Přidání prvku na konec  
cisla[3] = 2       # změna čtvrtého prvku
```

### 2.2 Slicing (řezy) - nová technika

Slicing umožňuje vybrat část seznamu:

#### 2.2.1 Základní syntaxe: seznam[start:stop]

```
cisla = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

cisla[2:5]      # [2, 3, 4] - od indexu 2 do 5 (bez 5!)
cisla[:3]       # [0, 1, 2] - od začátku do indexu 3
cisla[7:]       # [7, 8, 9] - od indexu 7 do konce
cisla[-3:]      # [7, 8, 9] - poslední 3 prvky
```

💡 Zapamatujte si

- start je **včetně** (from)
- stop je **bez** (to, but not including)
- cisla[2:5] znamená: “od indexu 2 až po (ale bez) index 5”

### 2.2.2 Slicing s krokem: seznam[start:stop:step]

```
cisla = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

cisla[::-2]     # [0, 2, 4, 6, 8] - každý druhý
cisla[1::2]     # [1, 3, 5, 7, 9] - každý druhý, ale od indexu 1
cisla[::-3]     # [0, 3, 6, 9] - každý třetí
cisla[::-1]     # [9, 8, 7, ..., 0] - OTOČENÍ seznamu!
```

### 2.2.3 Praktické příklady

```
text = "Python"
text[::-1]      # "nohtyP" - otočený string

# První polovina seznamu
polovina = len(cisla) // 2
prvni_cast = cisla[:polovina]

# Druhá polovina
druha_cast = cisla[polovina:]
```

## 2.3 Užitečné metody seznamů

### 2.3.1 `extend()` - připojení více prvků

```
a = [1, 2, 3]
a.extend([4, 5, 6])      # [1, 2, 3, 4, 5, 6]

# Alternativně:
a = a + [4, 5, 6]      # Totéž, ale vytvoří nový seznam
```

### 2.3.2 `insert()` - vložení na konkrétní pozici

```
cisla = [1, 2, 4, 5]
cisla.insert(2, 3)      # [1, 2, 3, 4, 5] - vložit 3 na index 2
```

### 2.3.3 `remove()` - odstranění prvního výskytu hodnoty

```
cisla = [1, 2, 3, 2, 4]
cisla.remove(2)        # [1, 3, 2, 4] - odstraní první výskyt 2
```

### 2.3.4 `index()` - vrací index prvního výskytu hodnoty

```
cisla = [1, 2, 3, 2, 4]
cisla.index(2)         # vrací 1
```

### 2.3.5 `pop()` - odstranění a vrácení prvku

```
cisla = [1, 2, 3, 4, 5]
posledni = cisla.pop()    # posledni = 5, cisla = [1,2,3,4]
druhy = cisla.pop(1)      # druhý = 2, cisla = [1,3,4]
```

### 2.3.6 reverse() - otočení seznamu

```
cisla = [1, 2, 3, 4, 5]
cisla.reverse()          # [5, 4, 3, 2, 1] - mění původní seznam!

# Alternativně (vytvoří nový):
otoceny = cisla[::-1]
```

### 2.3.7 sort() - seřazení

```
cisla = [5, 2, 8, 1, 9]
cisla.sort()            # [1, 2, 5, 8, 9] - vzestupně

cisla.sort(reverse=True) # [9, 8, 5, 2, 1] - sestupně

# Alternativně (vytvoří nový):
serazeny = sorted(cisla)
```

## 2.4 Cvičení 1: Práce se seznamem

```
cisla = [10, 25, 3, 47, 8, 19, 33, 5]

# 1. Vypište první tři čísla
# 2. Vypište poslední dvě čísla
# 3. Vypište každé druhé číslo
# 4. Otočte seznam a vypište ho
# 5. Seřaďte seznam vzestupně
```

---

### 3 Hledání minima a maxima

💡 Proč začínáme s minimem/maximem?

Tento algoritmus se **přímo použije** při třídění! Je to perfektní příprava na Selection Sort.

#### 3.1 Motivace

**Úkol:** Máte seznam čísel [5, 2, 8, 1, 9, 3]. Najděte nejmenší číslo.

**Otázka pro studenty:** Jak byste to udělali ručně?

Většina řekne: “*Projdu všechna čísla a pamatuji si, jaké bylo dosud nejmenší.*”

#### 3.2 Algoritmus krok za krokem

Seznam: [5, 2, 8, 1, 9, 3]

1. Začnu, `minimum = 5` (první číslo)
2. Další je `2 → 2 < 5?` Ano → nové `minimum = 2`
3. Další je `8 → 8 < 2?` Ne
4. Další je `1 → 1 < 2?` Ano → nové `minimum = 1`
5. Další je `9 → 9 < 1?` Ne
6. Další je `3 → 3 < 1?` Ne
7. Konec → `minimum = 1`

#### 3.3 Implementace - společně

```
cisla = [5, 2, 8, 1, 9, 3]

# Začnu s prvním číslem jako minimem
minimum = cisla[0]

# Projdu zbytek seznamu
for cislo in cisla:
    if cislo < minimum:
        minimum = cislo
```

```
print(f"Nejmenší číslo: {minimum}")
# Nejmenší číslo: 1
```

**i** Proč `minimum = cisla[0]`?

Mohli bychom začít s velkým číslem (např. `minimum = 999999`), ale to není elegantní. Lepší je začít s prvním číslem v seznamu - víme, že `minimum` nemůže být větší než první prvek!

### 3.4 Maximum - obdobně

```
cisla = [5, 2, 8, 1, 9, 3]
maximum = cisla[0]

for cislo in cisla:
    if cislo > maximum:
        maximum = cislo

print(f"Největší číslo: {maximum}")
# Největší číslo: 9
```

### 3.5 Hledání indexu minima

**Důležité pro třídění!** Často potřebujeme nejen hodnotu minima, ale i jeho **pozici** v seznamu.

```
cisla = [5, 2, 8, 1, 9, 3]

# Začnu s indexem 0
min_index = 0

# Projdu všechny indexy
for i in range(len(cisla)):
    if cisla[i] < cisla[min_index]:
        min_index = i

print(f"Index nejmenšího čísla: {min_index}")
print(f"Hodnota: {cisla[min_index]}")
```

```
# Index nejmenšího čísla: 3  
# Hodnota: 1
```

! Proč potřebujeme index?

Při třídění budeme chtít **prohodit** pozice prvků. K tomu potřebujeme vědět, **kde** se minimum nachází!

## 4 Třídění

### 4.1 Motivace

**Úkol:** Seřaďte seznam [5, 2, 8, 1, 9] vzestupně.

**Otzáka:** Jak byste to udělali na papíře?

**i** Brainstorming (5 minut)

Diskutujte se spolužáky: - Jak byste řadili čísla ručně? - Jaké kroky byste opakovali? - Můžete to popsat slovně?

Většina lidí přijde na jeden z těchto přístupů: 1. "Najdu nejmenší, dám ho stranou, opakuji" → Selection Sort 2. "Porovnávám sousedy, když je špatné pořadí, prohodím, opakuji DOKUD prohazuju" → Bubble Sort

### 4.2 Bubble Sort (bublinové třídění)

#### 4.2.1 Princip

Procházíme seznam a **porovnáváme dvojice sousedních prvků**. Pokud jsou ve špatném pořadí, prohodíme je. **Opakujeme DOKUD jsme něco prohazovali.**

#### 4.2.2 Vizualizace na příkladu [5, 2, 8, 1, 9]

První průchod:

```
[5, 2, 8, 1, 9] → 5 > 2? Ano, prohodí  
[2, 5, 8, 1, 9] → 5 > 8? Ne  
[2, 5, 8, 1, 9] → 8 > 1? Ano, prohodí  
[2, 5, 1, 8, 9] → 8 > 9? Ne  
[2, 5, 1, 8, 9] ← Prohodili jsme → pokračujeme!
```

Druhý průchod:

```
[2, 5, 1, 8, 9] → 2 > 5? Ne  
[2, 5, 1, 8, 9] → 5 > 1? Ano, prohodí  
[2, 1, 5, 8, 9] → 5 > 8? Ne  
[2, 1, 5, 8, 9] → 8 > 9? Ne  
[2, 1, 5, 8, 9] ← Prohodili jsme → pokračujeme!
```

Třetí průchod:

```
[2, 1, 5, 8, 9] → 2 > 1? Ano, prohodí  
[1, 2, 5, 8, 9] → 2 > 5? Ne  
[1, 2, 5, 8, 9] → 5 > 8? Ne  
[1, 2, 5, 8, 9] → 8 > 9? Ne  
[1, 2, 5, 8, 9] ← Prohodili jsme → pokračujeme!
```

Čtvrtý průchod:

```
[1, 2, 5, 8, 9] → 1 > 2? Ne  
[1, 2, 5, 8, 9] → 2 > 5? Ne  
[1, 2, 5, 8, 9] → 5 > 8? Ne  
[1, 2, 5, 8, 9] → 8 > 9? Ne  
[1, 2, 5, 8, 9] ← Nic se neprohodilo → HOTODO!
```

#### 4.2.3 Implementace - společně

```

seznam = [5, 2, 8, 1, 9]
print(f"Původní seznam: {seznam}")

# Klíčová myšlenka: "Opakuj DOKUD prohazuješ"
serazeno = False

while not serazeno:
    # Předpokládám, že seznam JE seřazený
    serazeno = True

    # Projdu všechny sousední dvojice
    for i in range(len(seznam) - 1):
        # Porovnám sousedy
        if seznam[i] > seznam[i + 1]:
            # Špatné pořadí → prohození
            seznam[i], seznam[i + 1] = seznam[i + 1], seznam[i]
            # Něco jsem prohodil → NENÍ seřazeno!
            serazeno = False

print(f"Seřazený seznam: {seznam}")
# [1, 2, 5, 8, 9]

```

 Jak to funguje?

1. **serazeno = False**: Na začátku předpokládáme, že seznam není seřazený
2. **while not serazeno**: Opakuj, DOKUD není seřazeno
3. **serazeno = True**: Na začátku každého průchodu předpokládám “je seřazeno”
4. **Projdu všechny dvojice**: **for i in range(len(seznam) - 1)**
5. **Pokud najdu špatné pořadí**: Prohodím a nastavím **serazeno = False**
6. **Pokud projdu celý seznam** a nic neprohodím → **serazeno** zůstane **True** → cyklus skončí!

#### 4.2.4 Alternativní implementace (volitelně)

Předchozí implementace je intuitivní, protože odráží způsob, jak lidé **přirozeně myslí**. Ve skutečnosti není úplně efektivní, protože po prvním průchodu máme jistotu, že poslední prvek je již na správném místě. Alternativní implementace pomocí dvou vnořených ‘for’ cyklů toto zohledňuje:

```

seznam = [5, 2, 8, 1, 9]

# Vnější cyklus - kolik průchodů potřebujeme?
for i in range(len(seznam)):

    # Vnitřní cyklus - porovnání sousedů
    for j in range(len(seznam) - 1 - i):

        # Porovnání a prohození
        if seznam[j] > seznam[j + 1]:
            seznam[j], seznam[j + 1] = seznam[j + 1], seznam[j]

print(seznam) # [1, 2, 5, 8, 9]

```

### ! Vnořené cykly!

Všimněte si **dvou cyklů jeden v druhém**: - **Vnější** (`for i`) říká: "Kolikrát opakuji celý průchod?" - **Vnitřní** (`for j`) říká: "Jak procházím dvojice?"  
Toto je **vnořený cyklus** - stejný princip jako vnořené modely v Model Builderu!

### 4.2.5 Proč `len(seznam) - 1 - i`?

```

# První průchod (i=0): porovnáváme všechny dvojice
range(len(seznam) - 1) # 0,1,2,3

# Druhý průchod (i=1): poslední je už správně, vynecháme
range(len(seznam) - 1 - 1) # 0,1,2

# A tak dále...

```

### i Která verze je efektivnější?

#### Závisí na situaci!

**While verze** je mnohem rychlejší, když: - Seznam je už seřazený nebo skoro seřazený - Může skončit brzy (best case: jen 1 průchod)

**For verze** je rychlejší, když: - Seznam je úplně zamíchaný nebo opačně seřazený - V nejhorším případě dělá méně porovnání celkem

## 4.3 Selection Sort (třídění výběrem)

💡 Teď použijeme, co jsme se naučili!

Pamatujete si hledání indexu minima? Přesně to teď použijeme!

### 4.3.1 Princip

**Myšlenka:** 1. Najdi **nejmenší prvek** v celém seznamu → dej ho na první místo 2. Najdi **nejmenší prvek** ve zbytku seznamu → dej ho na druhé místo 3. Opakuj...

**Využíváme:** Algoritmus pro hledání indexu minima, který jsme právě napsali!

### 4.3.2 Vizualizace krok za krokem

Původní seznam: [5, 2, 8, 1, 9, 3]

Krok 1: Najdi minimum v [5, 2, 8, 1, 9, 3]  
→ minimum je 1 (index 3)  
→ prohod' 5 1  
[1, 2, 8, 5, 9, 3]  
^ vyřešeno

Krok 2: Najdi minimum v [2, 8, 5, 9, 3]  
→ minimum je 2 (index 1)  
→ už je na správném místě  
[1, 2, 8, 5, 9, 3]  
^ vyřešeno

Krok 3: Najdi minimum v [8, 5, 9, 3]  
→ minimum je 3 (index 5)  
→ prohod' 8 3  
[1, 2, 3, 5, 9, 8]  
^ vyřešeno

Krok 4: Najdi minimum v [5, 9, 8]  
→ minimum je 5 (index 3)  
→ už je správně  
[1, 2, 3, 5, 9, 8]  
^ vyřešeno

Krok 5: Najdi minimum v [9, 8]  
→ minimum je 8 (index 5)  
→ prohod' 9 8  
[1, 2, 3, 5, 8, 9]  
^ vyřešeno

Hotovo!

#### 4.3.3 Implementace - společně

```
seznam = [5, 2, 8, 1, 9, 3]

# Pro každou pozici 0, 1, 2, ...
for i in range(len(seznam)):
    # Najdi index minima ve zbytku seznamu
    # (od pozice i do konce)
    # ↓↓↓ TOHLE JSME DĚLALI PŘED CHVÍLÍ! ↓↓↓
    min_index = i

    for j in range(i + 1, len(seznam)):
        if seznam[j] < seznam[min_index]:
            min_index = j
    # ↑↑↑ ALGORITMUS PRO HLEDÁNÍ INDEXU MINIMA ↑↑↑

    # Prohod' aktuální prvek s nalezeným minimem
    seznam[i], seznam[min_index] = seznam[min_index], seznam[i]

print(seznam) # [1, 2, 3, 5, 8, 9]
```

! Všimněte si!

**Vnitřní cyklus** je přesně algoritmus pro hledání indexu minima, který jsme dělali před chvílí!

Selection Sort = **opakování použití** algoritmu pro hledání minima!

💡 Pro praxi

V reálných programech používáme vestavěnou funkci `sort()` nebo `sorted()`, která je **mnohem rychlejší**:

```

seznam = [5, 2, 8, 1, 9]

# Způsob 1: sort() - upraví původní seznam
seznam.sort()
print(seznam) # [1, 2, 5, 8, 9]

# Způsob 2: sorted() - vrátí nový seznam
puvodni = [5, 2, 8, 1, 9]
serazeny = sorted(puvodni)
print(puvodni) # [5, 2, 8, 1, 9] - nezměněn
print(serazeny) # [1, 2, 5, 8, 9]

```

**Proč jsme pak psali vlastní algoritmus?** - Rozvoj algoritmického myšlení - Pochopení, jak počítače fungují uvnitř - Příprava na složitější problémy (v GIS budete řešit vlastní algoritmy!) - Pochopení **vnořených cyklů**

---

## 5 Vnořené cykly a propojení s Model Builderem

### 5.1 Co jsou vnořené cykly?

Vnořený cyklus = cyklus uvnitř jiného cyklu.

```

for i in range(3):
    for j in range(2):
        print(f"i={i}, j={j}")

```

Výstup:

```

i=0, j=0
i=0, j=1
i=1, j=0
i=1, j=1
i=2, j=0
i=2, j=1

```

**Jak to funguje:** - Pro každou hodnotu i (vnější cyklus) - Projdeme všechny hodnoty j (vnitřní cyklus)

## 5.2 Praktický příklad: Tabulka násobení

```
# Tabulka násobení 1-5
for i in range(1, 6):
    for j in range(1, 6):
        print(f"{i} × {j} = {i * j}")
    print("---") # Oddělovač
```

## 5.3 Propojení s Model Builderem

**i** Vzpomínáte na vnořené modely?

V Model Builderu jste používali **vnořené iterátory**: - Vnější iterator: Pro každý **okres** - Vnitřní iterator: Pro každý **rok**

- → Zpracovat data pro každý okres v každém roce

**V Pythonu je to stejné, jen místo Model Builderu píšete kód!**

### 5.3.1 Příklad: GIS úloha

```
okresy = ["Praha", "Brno", "Ostrava"]
roky = [2020, 2021, 2022]

for okres in okresy:
    for rok in roky:
        print(f"Zpracovávám: {okres}, {rok}")
        # Zde by byl ArcPy kód pro zpracování dat
```

**Výstup:**

```
Zpracovávám: Praha, 2020
Zpracovávám: Praha, 2021
Zpracovávám: Praha, 2022
Zpracovávám: Brno, 2020
Zpracovávám: Brno, 2021
Zpracovávám: Brno, 2022
Zpracovávám: Ostrava, 2020
Zpracovávám: Ostrava, 2021
Zpracovávám: Ostrava, 2022
```

**Vypadá vám to povědomě?** Přesně toto jste dělali v Model Builderu graficky. V Pythonu to napíšete přímo!

---

## 6 Shrnutí

### 6.1 Co jsme se naučili

**Slicing** - řezy seznamů (`seznam[start:stop:step]`)

**Metody seznamů** - `extend()`, `insert()`, `remove()`, `pop()`, `reverse()`, `sort()`

**Hledání minima/maxima** - algoritmus pro hledání nejmenší/největší hodnoty

**Index minima** - klíčové pro třídění!

**Bubble Sort** - třídění porovnáváním sousedů (while verze)

**Selection Sort** - třídění výběrem minima

**Vnořené cykly** - cyklus v cyklu

**Propojení s Model Builderem** - vnořené cykly = vnořené iterátory

### 6.2 Co bude příště?

V příští lekci:

- **Funkce** - jak psát znovupoužitelný kód
  - **Moduly** - jak organizovat program
  - Přepíšeme Fibonacci, faktoriál a třídění **jako funkce**
  - Vytvoříme vlastní modul `math_utils.py`
- 

## 7 Domácí úkol

### 7.1 Varianta A (základní)

#### 1. Procvičení minima/maxima:

- Napište funkci, která vrátí minimum a maximum najednou
- Najděte druhé nejmenší číslo v seznamu

#### 2. Prvočísla

- Napište program, který zjistí, zda je číslo prvočíslo
- **Nápověda:**

```
n = int(input("Zadej číslo: "))
je_prvocislo = True

# Zkusit dělit všemi číslly od 2 do n-1
for i in range(2, n):
    if n % i == 0: # Je dělitelné?
        je_prvocislo = False
        break

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

## 7.2 Varianta B (pokročilá)

1. Všechna prvočísla do 100 - najděte všechna prvočísla v rozsahu
2. Implementujte oba třídící algoritmy a porovnejte, který je čitelnější
3. K-té nejmenší číslo - najděte třetí nejmenší číslo v seznamu

## 7.3 Varianta C (výzva)

1. **Eratosthenovo síto** - velmi rychlý algoritmus pro hledání prvočísel:

```
n = 100
je_prvocislo = [True] * (n + 1)
je_prvocislo[0] = je_prvocislo[1] = False

# Implementujte Eratosthenovo síto...
```

2. **Vlastní třídící algoritmus** - vymyslete úplně jiný způsob třídění

## 8 Cheatsheet

```
# === SLICING ===
seznam[start:stop]      # Od start do stop (bez stop)
seznam[:n]                # První n prvků
seznam[n:]                # Od n-tého do konce
seznam[-n:]                # Posledních n prvků
seznam[::-step]            # Každý step-tý prvek
seznam[::-1]                # Otočení seznamu

# === METODY SEZNAMŮ ===
seznam.extend([a, b])      # Přidání více prvků
seznam.insert(i, x)        # Vložení x na index i
seznam.remove(x)           # Odstranění první hodnoty x
seznam.pop()                # Odstranění a vrácení posledního
seznam.pop(i)                # Odstranění a vrácení i-tého
seznam.reverse()            # Otočení seznamu (in-place)
seznam.sort()                # Seřazení (in-place)
sorted(seznam)                # Seřazení (nový seznam)

# === MINIMUM/MAXIMUM ===
# Hodnota minima
minimum = cisla[0]
for cislo in cisla:
    if cislo < minimum:
        minimum = cislo

# Index minima
min_index = 0
for i in range(len(cisla)):
    if cisla[i] < cisla[min_index]:
        min_index = i

# === BUBBLE SORT (while verze) ===
serazeno = False
while not serazeno:
    serazeno = True
    for i in range(len(seznam) - 1):
        if seznam[i] > seznam[i + 1]:
            seznam[i], seznam[i + 1] = seznam[i + 1], seznam[i]
            serazeno = False
```

```
# === SELECTION SORT ===
for i in range(len(seznam)):
    min_index = i
    for j in range(i + 1, len(seznam)):
        if seznam[j] < seznam[min_index]:
            min_index = j
    seznam[i], seznam[min_index] = seznam[min_index], seznam[i]

# === VNOŘENÉ CYKLY ===
for i in range(n):
    for j in range(m):
        # Pro každé i projdi všechna j
        print(f"{i}, {j}")
```

---

## 9 Poznámky pro vyučujícího

### 9.1 Běžné chyby studentů

```
# 1. Slicing - zapomínají, že stop je bez
seznam[2:5] # [2, 3, 4] - NE [2, 3, 4, 5]!

# 2. Bubble sort - zapomínají range(len - 1)
for i in range(len(seznam)): # CHYBA - index out of range!
    if seznam[i] > seznam[i + 1]: # i+1 přesáhne délku

# Správně:
for i in range(len(seznam) - 1):

# 3. Selection sort - prohození před nalezením minima
for i in range(len(seznam)):
    min_index = i #
    seznam[i], seznam[min_index] = ... # CHYBA - min_index ještě není správný!
    # Najít minimum NEJDŘÍV!

# 4. Minimum - zapomínají inicializovat
minimum = cisla[0] # Začít s prvním prvkem!
```

### 9.2 Časový plán (90 min)

Čas	Obsah
0-5 min	Rekapitulace minulé lekce
5-20 min	Seznamy - slicing a metody
20-45 min	<b>Minimum/Maximum</b> - hodnota + index
45-85 min	<b>Třídění</b> - Bubble Sort (20 min) + Selection Sort (20 min) + vnořené cykly (5 min)
85-90 min	Domácí úkol

### 9.3 Klíčové momenty

#### 9.3.1 Seznamy (5-20 min):

- Slicing ukázat na konkrétních příkladech

- `[::-1]` je nejužitečnější trik
- Metody rychle proletět, nejsou kritické

### 9.3.2 Minimum/Maximum (20-45 min):

- **Zdůrazněte:** "Tohle použijeme za chvíli v třídění!"
- **Index minima** je klíčový - věnujte tomu čas
- Nechte studenty implementovat minimum sami
- Ukažte debugovací verzi

### 9.3.3 Třídění (45-85 min):

- **KRITICKÉ:** Začít brainstormingem (5 min)
- Studenti často sami přijdou na algoritmy!
- **Bubble Sort s while** - zdůrazněte přirozenou logiku
- **Selection Sort** - "Vidíte? To jsme dělali před chvílí!"
- Vizualizovat na tabuli/projektoru
- Zdůraznit vnořené cykly

## 9.4 Rizika a řešení

### 1. Třídění může být těžké

- Řešení: Mít připravený promítaný kód
- Selection Sort je volitelný (můžete přeskočit)
- Důležitější je dobrý Bubble Sort

### 2. Studenti budou různě rychlí

- Řešení: Rychlejší pomáhají pomalejším
- Bonusové úkoly pro rychlé

### 3. Vnořené cykly jsou abstraktní

- Řešení: Vizualizovat tabulkou násobení
- Propojit s Model Builderem iterátory!

## 9.5 Tipy

- **Brainstorming třídění je klíčový!** Dejte studentům čas přemýšlet
- Na tabuli vizualizujte krok za krokem
- Připravte si seznam čísel na kartičkách - fyzické třídění!

- **Propojení s Model Builderem** - tento moment je důležitý, zdůrazněte ho!
- **Minimum → Selection Sort** - zdůrazněte propojení
- Ukažte, jak elegantní je `sort()` po tom, co implementovali vlastní algoritmus