Úvod

KIV/ADT – 1. přednáška

Miloslav Konopík, Jakub Sido

15. února 2024

Obsah

Organizace

Organizace přednášek a cvičení

- Přednášky:
  - Nepovinné, ale doporučené.
  - Teoretické znalosti ke zkoušce.
- Cvičení.
  - Získání a ověření praktických dovedností.
  - Na průběžných testech body nutné k získání zápočtu.

Komunikace

Discord:

- https://discord.gg/ESb9DRrE7E

Email:

- **Garant / přednášky:** konopik@kiv.zcu.cz
- **Cvičení:** sidoj@kiv.zcu.cz

Požadavky na absolvování předmětu

https://courseware.zcu.cz/portal/studium/courseware/kiv/adt/podminky-absolvovani.html

Náplň přednášek

- Co již umíte:
  - Základy algoritmizace (proměnné, podmínky, podprogramy, cykly, pole)
  - Jazyk Python (reprezentace dat, OOP, práce se soubory)
  - Algoritmy řazení.
- Co se naučíte.
  - Abstraktní datové struktury.
  - Algoritmická složitost.
  - Algoritmizace (prohledávání stavového prostoru, dynamické programování).
  - Pokročilejší datové struktury.
  - Grafové algoritmy.

Základy Pythonu – opakování a malé rozšíření.

Proč Python

- Důvody pro:
  - Světově nejpoužívanější jazyk dle TIOBE indexu.
  - Největší aktivita na GitHubu.
  - Zaměřuje se na vyřešení problému (nalezení algoritmu) bez ohledu na vazbu na vnitřní implementaci pomocných rutin a knihoven.
  - Velké množství výkonných knihoven (neguje pomalost samotného jazyka) – např. volba č. 1 pro strojové učení.
- Důvody proti:
  - Není programátorsky "čistý".
  - Slabější typová podpora.
  - Komplikovaná paralelizace programů (GIL).

Kde se Python naučit

- W3 schools: Tutoriál.
- W3 schools: Interaktivní cvičení.
- Google lekce.
- Udecative interaktivní kurz.
- A mnoho dalších...

Python kód

```python
def factorial(n: int) -> int:
    """Spočítej faktoriál zadaného čísla."""

    result: int = 1
    for i in range(1, n+1):
        result *= i
    return result

# Získání vstupu od uživatele
n: int = int(input("Zadej číslo pro výpočet jeho faktoriálu: "))

# Volání funkce faktorial a tisk výsledku
result: int = factorial(n)
print("The factorial of", n, "is", result)
```

- Definice funce a její použití.
- Stanovení typů parametrů a proměnných.
- Načtení vstupu od uživatele.
- Konverze datového typu.

Kvíz 1

Vstupní bod programu I

```
1  def factorial(n: int) -> int:
2      result: int = 1
3      for i in range(1, n+1):
4          result *= i
5      return result

6  def main() -> None:
7      # Získání vstupu od uživatele
8      n: int = int(input("Zadejte číslo pro výpočet jeho faktoriálu: "))

9      # Volání funkce factorial a výpis výsledku
10     result: int = factorial(n)
11     print("Faktoriál čísla", n, "je", result)

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

Vstupní bod programu II – parametr příkazové řádky

```python
import sys

def main() -> None:
    # Získání vstupu ze systémových argumentů
    if len(sys.argv) != 2:
        print("Nesprávný počet argumentů!")
        print("Použití: python factorial.py N")
        sys.exit(1)
    n = int(sys.argv[1])

    # Volání funkce factorial a výpis výsledku
    result: int = factorial(n)
    print("Faktoriál čísla", n, "je", result)

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

- Nutno importovat balík `sys`.
- Seznamu `argv` obsahuje parametry příkazové řádky.
- Pozor, první parametr na indexu 0 obsahuje jméno volaného souboru.
- Použití příkazu ukončení programu `sys.exit([arg])` uprostřed kódu není dobrou praxí.
- Volitelný parametr `arg` umožňuje předání návratové hodnoty do systému.

Výjimky v Pythonu

Co jsou to výjimky:

- Výjimky jsou anomálie, které se vyskytnou během provádění programu.
- Je to způsob signalizace, že něco selhalo a program nemůže pokračovat běžným chodem.
- V Pythonu jsou výjimky objekty, které jsou vyvolány při vzniku anomálie.

Příklady častých výjimek:

- `SyntaxError` : vyvolána, když dojde k problému se syntaxí programu.
- `NameError`: vyvolána, když není nalezen název proměnné nebo funkce.
- `TypeError`: vyvolána, když se provádí operace na nesprávném typu objektu.
- `ValueError`: vyvolána, když je volána funkce s argumentem správného typu, ale s nevhodnou hodnotou.

Vestavěné výjimky: Python dokumentace.

Výjimky – motivační příklad

```
Zadejte číslo pro výpočet jeho faktoriálu: b
Traceback (most recent call last):
  File "ADT/intro.py", line 25, in <module>
    main()
  File "ADT/intro.py", line 18, in main
    n: int = int(input("Zadejte číslo pro výpočet jeho faktoriálu: "))
ValueError: invalid literal for int() with base 10: 'b'
```

Mechanismus výjimek:

- V případě nevalidního zadání skončí program chybou.
- `ValueError` je výjimka.
- Výjimku můžeme zachytit a obsloužit.

Ošetřování výjimek

- Když je výjimka vyvolána, Python hledá obslužný kód, který výjimku zpracuje.
- Pokud nebyl nalezen žádný obslužný kód, program skončí s chybovou zprávou.
- Výjimku můžeme sami vyvolat klíčovým slovem `raise`.
- V Pythonu můžeme výjimky zpracovat blokem `try`/`except`/`finally`.
- Blok `try` obsahuje kód, který může vyvolat výjimku.
- Blok `except` obsahuje kód, který se provede, pokud je výjimka vyvolána.
- Blok `finally` obsahuje kód, který se provede vždy, bez ohledu na to, zda byla výjimka vyvolána nebo ne.
  - Tento blok se používá pro zajištění, že se některé kroky vždy provedou, bez ohledu na to, co se stalo dříve.
  - Např. můžeme zajistit zavření souboru.

Ošetřování výjimek – příklad

```
1  def main() -> None:
2      """Hlavní funkce aplikace."""
3      # Získání vstupu od uživatele
4      while True:
5          try:
6              n = int(input("Zadejte číslo pro výpočet jeho faktoriálu: "))
7              break
8          except ValueError:
9              print("Neplatný vstup, zadejte celé číslo.")

10     # Volání funkce factorial a výpis výsledku
11     try:
12         result: int = factorial(n)
13     except ValueError as error:
14         print(error)
15         sys.exit(1)
16     print("Faktoriál čísla", n, "je", result)
```

Vyvolání výjimky – příklad

```
1  def factorial(n: int) -> int:
2      """Spočítej faktoriál zadaného čísla.
3
3      :param n: Číslo, pro které se má počítat faktoriál.
4      :return: Faktoriál vstupního čísla.
5      :raise ValueError: Pokud je `n` záporným číslem.
6      """
7
7      if n < 0:
8          raise ValueError("Faktoriál není definován pro záporná čísla.")
9      result = 1
10     for i in range(1, n+1):
11         result *= i
12     return result
```

Vazba na volající prostředí – návratová hodnota.

- Návratovou hodnotu z příkazu `sys.exit(status=None)` lze využít ve volajícím prostředí pro kontrolu výsledku programu.
- Návratovou hodnotu lze využít i v případě automatického testování.
- Příklad výpisu textového výstupu v závislosti na návratové hodnotě:

```
1 python factorial.py
2
3 if [ $? -eq 0 ]; then
4   echo "OK"
5 else
6   echo "Chyba"
7 fi
```

Vazba na volající prostředí – přesměrování standardního vstupu a výstupu

Využití přesměrování standardního vstupu a výstupu:

- Získání vstupu ze souboru místo klávesnice.
- Uložení výstupu do souboru místo na obrazovku.
- Předání standardního výstupu programu na standardní vstup jiného programu.

V Bash lze použít operátory pro přesměrování:

- > pro přepsání výstupu do souboru.
- >> pro přidání výstupu do souboru.
- < pro přesměrování vstupu ze souboru.
- | pro předání výstupu programu na vstup jiného programu (tzv. pipe).

```
1 echo 5 | python3 factorial.py       # ... Faktoriál čísla 5 je 120
2 python3 factorial.py < input.txt    # vstup.txt: 5
3 python3 factorial.py > vystup.txt   # vystup.txt: ... Faktoriál čísla 5 je 120
```

Dokumentace v Pythonu

- Dokumentace je zásadním prvkem vývoje softwaru.
- Kvalitní dokumentace usnadňuje psaní, údržbu a sdílení kódu.
- V Pythonu existují různé typy dokumentace, včetně:
  - Oficiální dokumentace Pythonu.
  - Vestavěná dokumentace – `help()`.
  - Docstringy v Python kódu – snímek č. 22.
  - Třetí stranou poskytovaná dokumentace (Stack Overflow, PyPI (Python Package Index), Read the Docs, tutoriály.

Dokumentace I

```python
def factorial(n: int) -> int:
    """Spočítej faktoriál zadaného čísla.

    :param n: Číslo, pro které se má počítat faktoriál.
    :return: Faktoriál vstupního čísla.
    """

    result: int = 1
    for i in range(1, n+1):
        result *= i
    return result
```

Dokumentace II

intro module

`intro.`**`factorial`**`(`*`n: int`*`) → int`

Spočítej faktoriál zadaného čísla.

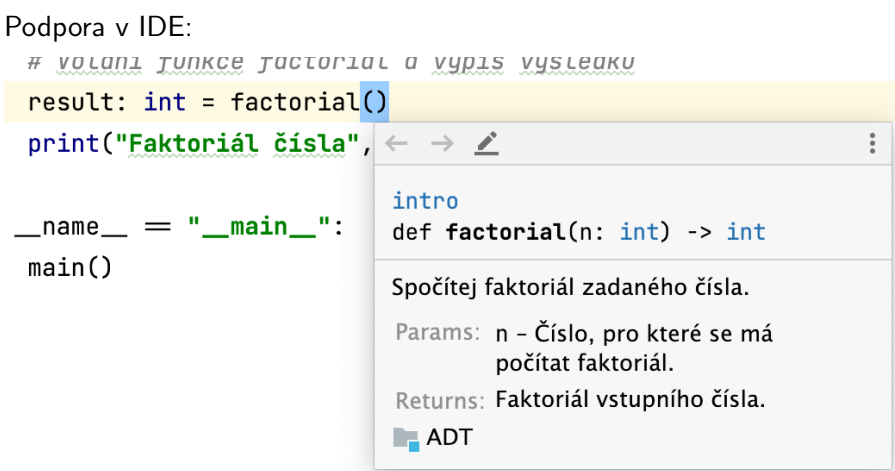
Parameters:	**n** – Číslo, pro které se má počítat faktoriál.
Returns:	Faktoriál vstupního čísla.

`intro.`**`main`**`() → None`

Hlavní funkce aplikace.

Dokumentace III

Podpora v IDE:



Datové typy.

Datové typy

Co jsou to datové typy?

- Datové typy se používají určení typu dat pro lidi i počítače.
- Datový typ se skládá z
  - definice povolených hodnot (rozsahu hodnot)
  - a povolených operací, které lze s daty tohoto typu provádět.
- Příklady datových typů jsou přirozená čísla, řetězce a seznamy.
- Datové typy existují i mimo programovací jazyky, ale programovací jazyky mají způsoby jejich reprezentace.
- Abstraktní verze běžných datových typů jsou nezávislé na programovacím jazyce, zatímco jejich konkretní realizace jsou dány jazykem, v našem případě jazykem Python.
- Pro rozlišení datových typů v programování je důležité přesně porozumět jejich definicím.

Datové typy – příklad

Například můžeme říci, že počet získaných medailí u sportovce, je přirozené číslo, což nám určuje, že hodnoty jako 2 a 16 jsou očekávané, zatímco počet -2 nebo text "David" je nesmyslný. Vědět, že počet medailí je přirozené číslo, nám také říká, jaké operace bychom mohli provést (např. "přičti 1 k počtu"), a vylučuje jiné operace (např. "seřaď tyto počty podle abecedy").

Dynamické typování v Pythonu

Definice: V Pythonu je typ proměnné určen za běhu, nikoli předem. To umožňuje proměnným měnit svůj datový typ během provádění programu.

Klíčové body:

- Proměnná má svůj typ, který definuje podporované operace s proměnnou.
- Není nutná explicitní deklarace datových typů.
- Proměnné mohou nabývat různých typů v různých částech programů.
- Zvyšuje flexibilitu, ale vyžaduje pečlivost, aby se předešlo chybám souvisejícím s typy.

Dynamické typování v Pythonu – Příklad

Příklad:

```
1  x = 5          # x je celé číslo
2  x = "Hello"    # Nyní je x řetězec
```

Dynamické typování v Pythonu – Příklad

Příklad:

```
1  x = 5          # x je celé číslo
2  x = "Hello"    # Nyní je x řetězec

3  x = 5          # x je celé číslo
4  x / 2          # 2.5

5  x = "Hello"    # Nyní je x řetězec
6  x / 2          # Chyba
```

Typové anotace v Pythonu

Typové anotace jsou spojeny s napovídáním typů (Type Hinting).
Zlepšení čitelnosti a kontroly kódu.

Definice:

Typové anotace umožňují programátorům explicitně specifikovat očekávaný datový typ proměnných, funkcí a návratových hodnot. I když Python zůstává dynamicky typovaným jazykem, typové anotace zlepšují čitelnost kódu a usnadňují statickou analýzu.

Typové anotace:

- Podporují lepší spolupráci v týmu a usnadňují revizi kódu.
- Pomáhají nástrojům jako je *mypy* při statické analýze kódu pro odhalení chyb před spuštěním.
- Nejsou vynuceny za běhu, slouží primárně pro volitelnou kontrolu kódu, nápovědu a dokumentaci.

Typové anotace v Pythonu – Příklad

Přiřazení hodnoty chybného typu při typové anotaci:

```
1  x: int = 5
2  x = "Hello"
```

Kontrola nástrojem *mypy*:

```
$ mypy mypy_example.py
mypy_example.py:2: error: Incompatible types in assignment
(expression has type "str", variable has type "int")[assignment]
Found 1 error in 1 file (checked 1 source file)
```

Kvíz II

Základní datové typy v Pythonu.

Objekty

objekty (`object`): Třída, od které jsou odvozeny všechny další třídy včetně základních vestavěných datových typů.

- V Pythonu jsou všechny hodnoty instance tříd.
- Třídy určují typ a definují vlastnosti a metody objektů.
- Každá instance má vlastní identitu, typ a hodnotu.
- Identita instance se nemění po celou dobu její existence.

Základní vestavěné metody tříd:

- `__init__`: volá se při vytváření nové instance objektu. Slouží k inicializaci atributů objektu.
- `__str__`: vrací řetězcovou reprezentaci objektu, kterou lze použít k výpisu objektu na výstup.
- `__eq__`: slouží k porovnání dvou objektů na rovnost.
- `__hash__`: vrací hash hodnotu objektu (více v tématu Hashovací tabulky).
- ...

Objekty – příklad I

```
1  class Rectangle:
2      sides: int = 4

3      def __init__(self, length: float, width: float):
4          self.length = length
5          self.width = width

6      def __str__(self) -> str:
7          return f"Obdelnik o délce {self.length} a šířce {self.width}"

8      def volume(self) -> float:
9          return self.length * self.width
```

Objekty – příklad II

```
1  class Rectangle:
2      sides: int = 4

3      def __init__(self, length: float, width: float):
4          self.length = length
5          self.width = width

6      ...

7  r = Rectangle(3, 4)
8  print("Obsah:", r.volume())              # Obsah: 12

9  # přístup k proměnné třídy
10 print("Počet stran:", Rectangle.sides)  # Počet stran: 4
```

Celá čísla

Celá čísla (`int`): Představují celá čísla, kladná i záporná, bez desetinné části. Jsou používána v široké škále operací, od počítání a indexování po matematické výpočty. V Pythonu 3 mají celá čísla neomezenou přesnost, což znamená, že mohou být tak velká, jak dovoluje paměť systému. Celá čísla lze definovat pomocí konstruktoru `int()` nebo přiřazením hodnoty bez uvozovek nebo desetinné čárky do proměnné.

```
1  x: int = 10
2  y: int = 3

3  z1: int = x + y      # z1 = 13
4  z2: int = x - y      # z2 = 7
5  z3: int = x * y      # z3 = 30
6  z4: int = x // y     # z4 = 3
7  z5: int = int("3")   # z5 = 3
```

Desetinná čísla

Desetinná čísla (`float`): Představují desetinná čísla s plovoucí desetinnou čárkou, včetně vědecké notace s písmenem (`e`). Jsou používána v matematických výpočtech, které zahrnují zlomky nebo desetinná čísla, jako je dělení nebo trigonometrie. Desetinná čísla v Pythonu jsou reprezentována jako dvojnásobně přesné 64bitové hodnoty, což znamená, že mají omezenou přesnost kvůli způsobu, jakým jsou uložena v paměti. Desetinná čísla lze definovat pomocí konstruktoru `float()` nebo přiřazením hodnoty s desetinnou čárkou do proměnné.

```
1  x: float = 3e2            # 3 · 10^2 = 300
2  y: float = 90

3  c: int = a // b           # c = 3
4  d: float = a / b          # d = 3.3333333333333335
5  e: float = float("3.14") # z = 3.14
```

Desetinná čísla – přetečení a podtečení

V jazyce Python mají čísla s pohyblivou řádovou čárkou omezený rozsah a přesnost. V důsledku toho může při provádění aritmetických operací s čísly s plovoucí desetinnou čárkou docházet k **přetečení** a **podtečení**.

- Přetečení – číslo je příliš velké a je aproximováno jako nekonečno (`inf`).
- Podtečení – číslo je příliš malé a je aproximováno jako nula.
- Důvod: nedostatečná paměť pro reprezentaci exponentu (viz KIV/PPA).

```
1  x = 1e308
2  y = 1e308
3  z = x * y     # Výsledek: inf (přetečení)
```

```
4  x = 1e-307
5  y = 1e-308
6  z = x * y     # Výsledek: 0.0 (podtečení)
```

Logické hodnoty

Logické hodnoty (`bool`): Představují hodnoty pravdy (`True`) a (`False`) a jsou používána v logických operacích a porovnání. Často se používají v řídících strukturách, jako jsou podmínky (`if`) a smyčky, k určení, zda je podmínka pravdivá nebo nepravdivá. Booleovské hodnoty lze definovat explicitně pomocí klíčových slov (`True`) a (`False`), nebo mohou být výsledkem porovnávací operace.

```
1  x: bool = True
2  y: bool = 5 < 3            # False
3  z1: bool = bool("False") # True
4  z2: bool = bool("")        # False
5  z3: bool = bool(0)         # False
```

Komplexní čísla

Komplexní čísla (`complex`): Představují čísla s reálnou a imaginární částí, přičemž imaginární část je reprezentována příponou (`j`) nebo (`J`). Jsou používána v matematických výpočtech, jako jsou Fourierovy transformace nebo řešení diferenciálních rovnic. Komplexní čísla lze definovat pomocí konstruktoru `complex()` nebo přiřazením hodnoty k proměnné s příponou (`j`) nebo (`J`).

```
1 x: complex = 2 + 3j
2 y: complex = (1 + 2j) * (3 + 4j)     # (-5+10j)
```

Řetězce

Řetězce (`str`): Představují posloupnost znaků, jako jsou písmena, čísla a symboly. Jsou používány v textových operacích, jako je manipulace s řetězci nebo formátování řetězců. Řetězce lze definovat pomocí jednoduchých nebo dvojitých uvozovek nebo trojitých uvozovek pro víceřádkové řetězce.

```
1  a: str = "Lenka"
2  b: str = "Láskorádová"
3  c: str = a + " " + b       # (str) 'Lenka Láskorádová'
4  d: str = str(5)            # (str) '5'

5  print(len("hello"))        # 5
6  print(c.upper())           # LENKA LÁSKORÁDOVÁ
7  print(c.lower())           # lenka láskorádová
```

Parsování a formátování čísel

- Parsování čísel, tedy převod řetězců do číslené reprezentace není vhodné provádět použitím konstruktorů. Robustnější řešení spočívá ve využití balíků, např. balíku `locale`.

```
1 import locale

2 a = u'545,545.2222'
3 locale.setlocale(locale.LC_ALL, 'en_US.UTF-8')
4 locale.atof(a)      # (float) 545545.2222
```

- Balík `locale` lze také použít pro korektní formátování čísel.

```
5 locale.format_string("%0.2f", f, grouping=True)  # '545,545.22'
6 locale.setlocale(locale.LC_ALL, 'cs_CZ.UTF-8')
7 locale.format_string("%0.2f", f, grouping=True)  # '545 545,22'
```

- Formátovat můžeme také formátovacími řetězci (dokumentace jazyka).

```
8 f'{f:0.8n}'                                      # '545 545,22'
```

NoneType

NoneType (`NoneType`): vestavěný typ, který reprezentuje nepřítomnost hodnoty.

- Typ má jedinou hodnotu `None`.
- `None` označuje, že proměnná nebo argument funkce nemá žádnou hodnotu nebo že návratová hodnota funkce nemá výsledek.
- `None` se také používá jako zástupná hodnota, kterou lze později v programu nahradit významovou hodnotou.
- `None` se chová jako False v logických výrazech, ale nemá žádnou `bool` hodnotu.

```
1  def moje_funkce(x):
2      if x < 0:
3          return None
4      else:
5          return x**2
6  result = moje_funkce(-2)
7  if result:
8      print(result)
9  else:
10     print("Chyba: vstup musí být
       ↪ nezáporný!")
```

Funkce jako datový typ

Funkce (`function`): lze definovat pomocí klíčového slova `def`, za kterým následuje název funkce, vstupní parametry (pokud existují) a tělo funkce. Funkce může mít návratovou hodnotu, která se zadává pomocí klíčového slova `return`. Pokud funkce nemá návratový příkaz, vrací hodnotu `None`.

- Funkce lze přiřazovat proměnným, předávat jako argumenty jiným funkcím, vracet jako hodnoty z funkcí a ukládat do kompozitních datových struktur.
- Objekt funkce získáme zapsáním identifikátoru (jména) funkce bez závorek.

```
1  from typing import Callable
2  def aplikuj(func: Callable[[int, int], int], x: int, y: int) -> int:
3      return func(x, y)

4  def secti(x: int, y: int) -> int:
5      return x + y

6  vysledek: int = aplikuj(secti, 2, 3)    # vysledek = 5
```

Lambda funkce

Lambda představuje jazyce Python kratší anonymní funkci.

- Lambda funkce mohou přijímat libovolný počet argumentů, ale mohou obsahovat pouze jeden výraz.
- Syntaxe funkce lambda je `lambda argumenty: výraz`
- Lambda funkce se často používají jako alternativa k definici funkce pro jednoduché operace.

```
1 from typing import Callable

2 def aplikuj(func: Callable[[int, int], int], x: int, y: int) -> int:
3     return func(x, y)

4 secti: Callable[[int, int], int] = lambda x, y: x + y

5 vysledek: int = aplikuj(secti, 2, 3)   # vysledek = 5
```

Lambda funkce

Lambda představuje jazyce Python kratší anonymní funkci.

- Lambda funkce mohou přijímat libovolný počet argumentů, ale mohou obsahovat pouze jeden výraz.
- Syntaxe funkce lambda je `lambda argumenty: výraz`
- Lambda funkce se často používají jako alternativa k definici funkce pro jednoduché operace.

```
1  # from typing import Callable

2  def aplikuj(func, x: int, y: int) -> int:
3      return func(x, y)

4  secti = lambda x, y: x + y

5  vysledek: int = aplikuj(secti, 2, 3)    # vysledek = 5
```

Aktualizovatelné a nezměnitelné datové typy

Aktualizovatelné a nezměnitelné datové typy

Datové typy:

- Aktualizovatelné – lze měnit jejich hodnotu.
- Neměnitelné (immutable) – nelze měnit jejich hodnotu.

Základní neměnitelné DT:

- Celá čísla (`int`)
- Desetinná čísla (`float`)
- Komplexní čísla (`complex`)
- Logické hodnoty (`bool`)
- Řetězce (`str`)
- N-tice (`tuple`)

Základní měnitelné DT:

- Uživatelské (`třídy`)
- Seznam (`list`) (dále v přednáškách)
- Slovník (`dict`) (dále v přednáškách)
- Množina (`set`) (dále v přednáškách)

Změna instance třídy VS přepsání hodnoty proměnné

```
1  class Rectangle:
2      def __init__(self, length: float, width: float):
3          self.length = length
4          self.width = width
5      ...
6  r1 = Rectangle(2, 3)
7  r2 = r1

8  r2.length = 6
9  print(r1)                    # Obdelnik o délce 6 a šířce 3

10 value1 = 6
11 value2 = value1
12 value2 = 7
13 print(value1)                # 6
```

Kolekce

Seznam

Seznam (`list`): uspořádaná, měnitelná kolekce objektů libovolného datového typu.

- Prvky jsou uzavřeny v hranatých závorkách `[ ]` a odděleny čárkami.
- Seznamy mohou obsahovat prvky různých datových typů, včetně jiných seznamů.
- Indexování začíná od 0.
- Seznamy lze spojovat a řezat pomocí operátorů `+` a `:`.
- Do seznamů lze přidávat (`append()`) nebo odebírat prvky (`pop()` nebo `remove()`).

```
1 fruits: List[str] = ["jablko", "banán", "třešeň"]
2 fruits.append("pomeranč")              # ["jablko", "banán", "třešeň",
  ↪ "pomeranč"]
3 fruits.remove("třešeň")                # ["banán", "třešeň", "pomeranč"]
4 fruits.pop()                           # ["jablko", "banán"]
5 fruits[1] = "hruška"                   # ["jablko", "hruška"]
6 fruits = fruits + ["banán", "třešeň"] # ["jablko", "hruška", "banán",
  ↪ "třešeň"]
```

Seznam – Iterace

Iterace – procházení prvků kolekcí.

```
1  fruits: List[str] = ["jablko", "banán", "třešeň"]

2  for fruit in fruits:
3      print(fruit)

4  for i in range(len(fruits)):
5      print(fruits[i])
```

Výstup:

```
jablko
banán
třešeň
```

Množina

Množina (`set`): neuspořádaná kolekce jedinečných a neměnných prvků. Definuje se pomocí vestavěné funkce `set()` nebo uzavřením posloupnosti hodnot oddělených čárkou do složených závorek `{}`. Množiny lze použít pro operace, jako je sjednocení, průnik a rozdíl, a podporují také testování příslušnosti a iteraci.

- Množiny jsou měnitelné.
- Duplicitní prvky jsou při vytváření množiny automaticky odstraněny.

```
1  fruits: Set[str]  = {"jablko", "banán", "jahoda", "hruška", "banán"}

2  "mandarinka" in fruits                    # False
3  fruits.remove("jahoda")                   # {"jablko", "banán", "hruška"}
4  f = fruits.union({"jablko", "třešeň"})    # {"jablko", "banán", "hruška",
   ↪  "třešeň"}
5  f = fruits.intersection({"banán", "kiwi", "hruška"}) # {"banán",
   ↪  "hruška"}
```

Množina – Iterace

```
1  fruits: Set[str]  = {"jablko", "banán", "jahoda", "hruška", "banán"}

2  for fruit in fruits:
3      print(fruit)
```

Výstup:

```
jablko
banán
jahoda
hruška
```

Pozor: `fruits[i]` nelze použít. Množina je neuspořádaná.

Slovník

Slovník (`dict`): neuspořádaná kolekce dvojic klíč-hodnota. Slovníky jsou proměnlivé a lze je upravovat přidáváním, aktualizací nebo odstraňováním dvojic klíč-hodnota.

- Klíče musí být jedinečné a neměnné (řetězce, čísla, zatímco hodnoty mohou být libovolného datového typu.
- Slovníky se definují pomocí složených závorek {} a dvojteček :.
- Klíče a hodnoty se oddělují čárkami ,.

```
1 fruit_counts: Dict[str, int] = {"jablko": 5, "banán": 2, "jahoda": 10,
  ↪ "pomeranč": 6}
2 fruit_counts["jablko"]          # 5
3 fruit_counts["hruška"] = 3      # {'jablko': 5,  "banán": 2, 'jahoda': 15,
  ↪ 'pomeranč': 6, 'hruška': 3}
4 fruit_counts["jahoda"] = 15     # {'jablko': 5,  "banán": 2, 'jahoda': 15,
  ↪ 'pomeranč': 6, 'hruška': 3}
5 del fruit_counts["banán"]       # {'jablko': 5, 'jahoda': 15, 'pomeranč': 6,
  ↪ 'hruška': 3}
6 "broskev" in fruit_counts       # False
```

Slovník – Iterace

```
1 fruit_counts: Dict[str, int] = {"jablko": 5, "banán": 2, "jahoda": 10,
  ↪ "pomeranč": 6}

2 for fruit in fruit_counts:
3     print(f"Počet {fruit}: {fruit_counts[fruit]}")

4 for fruit, count in fruit_counts.items():
5     print(f"Počet {fruit}: {count}")
```

Výstup:

```
Počet jablko: 5
Počet banán: 2
Počet jahoda: 10
Počet pomeranč: 6
```