IB113 Úvod do programování a algoritmizace
Přednáška 5

Seznamy a ntice v Pythonu

Nikola Beneš

16. říjen 2017

Co bude dnes?

Proměnné v Pythonu

- co vlastně znamená pojem *proměnná* v Pythonu
- (rozdíl oproti některým jiným jazykům)

Datové typy seznamy, ntice

- seznam (`list`)
- ntice (`tuple`)
- základní operace

Proměnné

Proměnné v různých programovacích jazycích

- pojmenované místo v paměti
- odkaz na místo v paměti (*Python*)
- kombinace obou přístupů

Přiřazení v různých jazycích

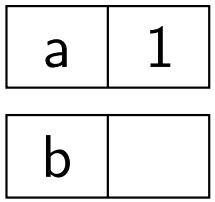
- proměnné ve stylu C (Pascal): změna obsahu paměti
- proměnné ve stylu Pythonu: změna (přesměrování) odkazu na jiné místo v paměti

Poznámka: Zatím jsme rozdíl nepozorovali, protože zatím jsme používali pouze neměnné (*immutable*) typy (čísla, řetězce).

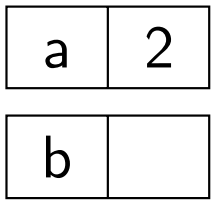
Proměnné podrobněji

Ilustrace přiřazení

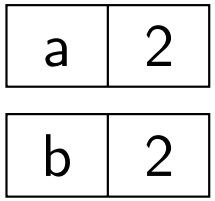
```
int a, b;
a = 1;
```




```
a = 2;
```



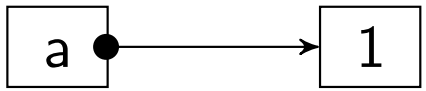

```
b = a;
```



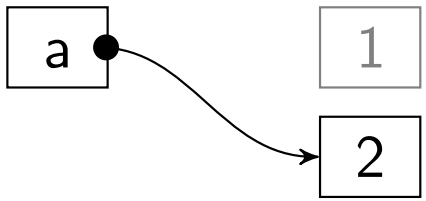
Jazyk C

Proměnné jako hodnoty

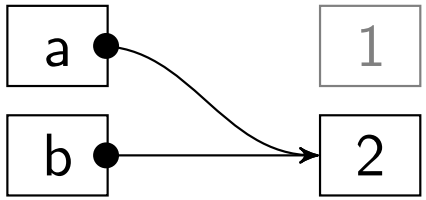
```
a = 1;
```




```
a = 2;
```




```
b = a;
```



Jazyk Python

Proměnné jako odkazy

Vizualizace běhu programu

Python Tutor

- `http://www.pythontutor.com`
- hodí se pro příklady se seznamy apod.
- zkuste, je-li nejasný některý z příkladů ve slidech

- pro zajímavost:
  - Python Tutor standardně neukazuje odkazy na neměnná data (čísla, řetězce)
  - dá se zapnout (možnost *render all objects on the heap*)

Seznamy

Motivace pro seznamy

- chceme zpracovávat větší množství položek
- nechceme psát opakovaně stejný kód (DRY)
- nemusíme předem znát počet položek

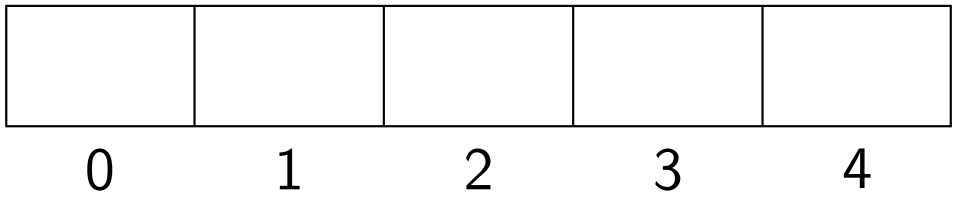
Příklady

- simulace preferencí politických stran (minule)
- frekvence písmen v textu
- seřazení studentů podle počtu bodů
- reprezentace herního plánu (piškvorky, šachy)
- ...

Příklad: Frekvence písmen v textu (nevhodné řešení)

```
def frequency_analysis(text):
    text = text.upper()
    freqA = 0
    freqB = 0
    freqC = 0
    for letter in text:
        if letter == "A":
            freqA += 1
        elif letter == "B":
            freqB += 1
        elif letter == "C":
            freqC += 1
    print("A:", freqA)
    print("B:", freqB)
    print("C:", freqC)
```

Seznamy



Seznamy v Pythonu (typ `list`)

- libovolný počet položek v pořadí za sebou
- indexováno od nuly (jako řetězce)
- běžně dostupné v jiných jazycích
  - pole (*array*) – pevná délka, všechny položky stejného typu
  - dynamická pole, seznamy, ...
- seznamy v Pythonu – obecnější než pole
  - umí měnit velikost
  - smí obsahovat položky různých typů
  - (ale práce se seznamy je pomalejší)
- pole v Pythonu (knihovna NumPy; nad rámec předmětu)

Seznamy v Pythonu

- zadány výčtem prvků

```
s = []                          # prázdný seznam
s = [3, 1, 4, 1, 5]             # položky stejného typu
s = ["ABC", 3.14, -7]           # položky různých typů
s = [[1, 2], [3, 4]]            # seznamy v seznamu
s = ["pes", "kočka", 0.01, ["velbloud", -13], []]
```

- zadány pomocí tzv. *list comprehension* (intenzionální zápis seznamu)

```
s = [2 * x for x in range(10)]
s = [x ** 2 for x in range(1, 10) if x % 2 == 0]
s = [3 * x for x in [5, 17, 23, 40]]
s = [[a, b, c] for a in range(1, 10) for b in range(1, 10)
     for c in range(1, 10) if a ** 2 + b ** 2 == c ** 2]
```

Operace se seznamy

```
len(s)          # délka seznamu
s.append(17)    # přidání prvku na konec seznamu
s.pop()         # odebrání prvku z konce seznamu
s + t           # zřetězení seznamů
s * 3           # opakování

s[0]            # přístup k prvnímu prvku
s[2] = "ABC"    # seznamy můžeme modifikovat!
s[-1]           # indexování od konce
s[1:4]          # kopie části seznamu
s[1:5:2]        # podobně jako u řetězců
s[:]            # kopie celého seznamu (k čemu to je?)

list(x)         # přetypování na seznam (uvidíme později)
```

Procházení seznamu

- mám seznam `s` a chci vypsat všechny jeho prvky, jak?
- ne úplně vhodné řešení:

```
for i in range(len(s)):
    print(s[i])
```

- lepší řešení:

```
for element in s:
    print(element)
```

- `range(...)` je *něco jako seznam*
  - (kvůli efektivitě to není seznam, ale tzv. generátor)
  - seznam můžeme vyrobit přetypováním `list(range(...))`

Seznamy a proměnné

```
s = [1, 2, 3]
t = s
s.append(4)
print(t)
```

- co bude výstupem? `[1, 2, 3, 4]`
  - proč?
  - protože proměnné v Pythonu jsou *odkazy*
  - `http://www.pythontutor.com`
- už víme, k čemu je dobré `s[:]`?
  - pokud potřebujeme vytvořit novou kopii seznamu

Seznamy a volání funkcí

```
def fun(x):
    x = 17

y = 10
fun(y)
print(y)
```

- co bude výstupem? 17
  - vypadá to, že funkce v Pythonu nemění hodnotu parametru (čísla jsou v Pythonu neměnná)

Seznamy a volání funkcí

```
def fun(s):
    s.append(17)

t = [1, 2]
fun(t)
print(t)
```

- co bude výstupem? `[1, 2, 17]`
  - vypadá to, že funkce v Pythonu mění hodnotu parametru (seznamy jsou v Pythonu měnitelné)

Seznamy a volání funkcí

```
def fun(s):
    s.append(17)
    s = [4, 5]
    s.append(19)

t = [1, 2]
fun(t)
print(t)
```

- co bude výstupem? `[1, 2, 17]`
  - proč?
  - přiřazení je změna odkazu
  - po provedení `s = [4, 5]` už `s` neukazuje na původní seznam
  - `http://www.pythontutor.com`

(podrobněji v některé z příštích přednášek)

Ntice (*tuples*)

- neměnná varianta seznamů
  - podobně jako řetězce
- fungují jako seznamy, ale nedají se měnit
- zápis: kulaté závorky místo hranatých
  - v jistých situacích se kulaté závorky smí vynechat

```
s = (1, "A", 3)
print(s[0])       # OK
s[0] = "B"        # chyba!
```

- k čemu je to dobré?
  - bezpečnější programování
  - nemůžeme omylem změnit
  - může být efektivnější (rychlejší, méně paměťově náročné)
  - typické použití: souřadnice

Typická použití ntic

- rozbalení ntice (funguje i se seznamy)

```
data = ("Rick Sanchez", "C317", "Earth")
name, dimension, planet = data
```

- prohození hodnot proměnných (*swap*)

```
a, b = b, a  # totéž, co (a, b) = (b, a)
```

- vracení více hodnot z funkce

```
def minmax(a, b):
    return min(a, b), max(a, b)

x, y = minmax(17, 10)

d, m = divmod(17, 5)  # standardní funkce v Pythonu
```

Výpočet průměru

```
def average1(nums):
    total = 0
    for i in range(len(nums)):
        total += nums[i]
    return total / len(nums)

def average2(nums):
    total = 0
    for num in nums:
        total += num
    return total / len(nums)

def average3(nums):
    return sum(nums) / len(nums)
```

- pointa? není nutné vynalézat kolo

Dělitelé čísla

```
def divisors(num):
    result = []
    for divisor in range(1, num + 1):
        if num % divisor == 0:
            result.append(divisor)
    return result
```

- dalo by se nějak vylepšit?
  - počítat jen do `num // 2` (a přidat `num` na konec)
  - využít toho, že `divisor` a `num // divisor` jsou oba dělitelé
  - menší z těchto dvou dělitelů je $\leq$ odmocnině z `num`
    (výsledný program bude na příští přednášce)

Příklad: Simulace předvolebního průzkumu

```
def survey(size, preferences):
    parties = len(preferences)
    count = [0] * parties

    for i in range(size):
        r = random.randint(1, 100)
        threshold = 0
        for p in range(parties):
            threshold += preferences[p]
            if r <= threshold:
                count[p] += 1
                break

    for p in range(parties):
        print("Party {}: {:.2f}".format(
            p + 1, 100.0 * count[p] / size))
```

Příklad: Frekvenční analýza

```
def frequency_analysis(text):
    text = text.upper()
    freq = [0 for i in range(26)]
    for letter in text:
        if "A" <= letter <= "Z":
            freq[ord(letter) - ord("A")] += 1

    for i in range(26):
        if freq[i] != 0:
            print(chr(i + ord("A")), freq[i])
```

Ještě lepší řešení by používalo tzv. slovník (uvidíme na některé z dalších přednášek).

Příklad: Převod do morseovky

```
morse = (".-", "-...", "-.-.", "-..")  # atd.

def to_morse(text):
    result = ""
    for letter in text:
        if "A" <= letter <= "Z":
            c = ord(letter) - ord("A")
            result += morse[c] + "|"
    return result
```

Vztah řetězců a seznamů

- **split** – vytvoření seznamu z řetězce

```
vowels = "a, e, i, o, u"
vowel_list = vowels.split(", ")

message = ".-|....|---|.---"
letters = message.split("|")
```

- **join** – vytvoření řetězce ze seznamu

```
data = ["abc", "kolo", "pes"]
result = "; ".join(data)
```

Vztah řetězců a seznamů

- načítání vstupu

```
x, y = input().split()
x = int(x)
y = int(y)
```

- pro zajímavost (mimo záběr předmětu):

```
x, y = map(int, input().split())
```

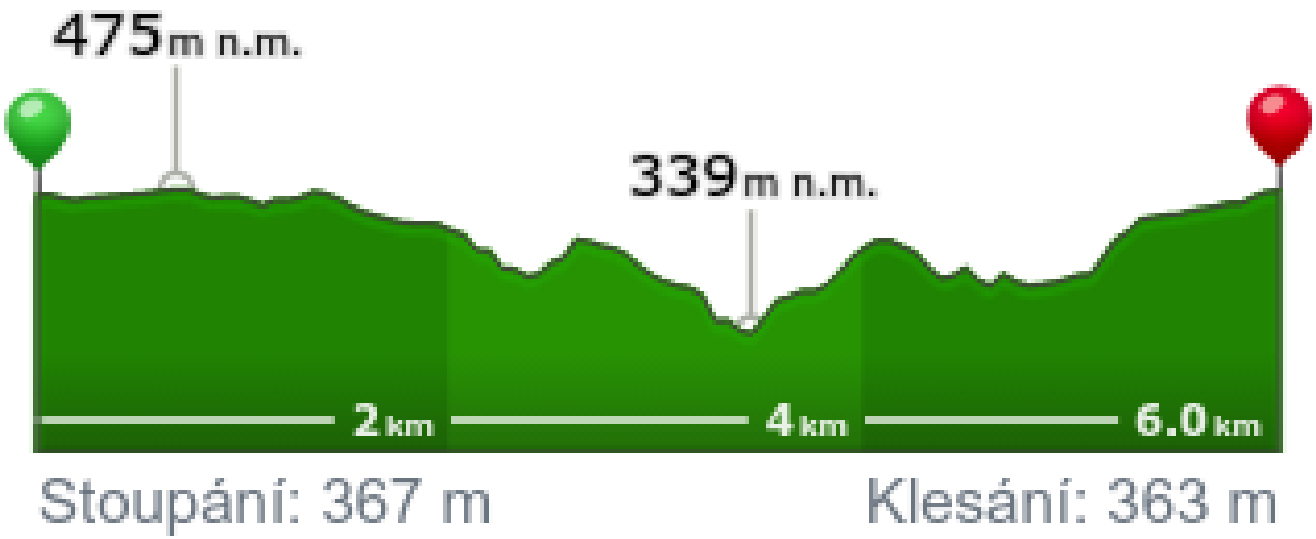
Příklad: Prvočísla – Erastothenovo síto

```
def sieve(count):
    result = []
    is_prime = [True] * count
    for p in range(2, count):
        if is_prime[p]:
            result.append(p)
            for mult in range(2 * p, count, p):
                is_prime[mult] = False
    return result

sum_primes = sum(sieve(1000000))

print("Součet všech prvočísel menších než milion je {}.".\
      format(sum_primes))
```

Příklad: Výškový profil



Příklad: Výškový profil

```
def height_profile(heights):
    for level in range(max(heights), 0, -1):
        for height in heights:
            if height >= level:
                print("# ", end="")
            else:
                print("  ", end="")
        print()
```

Příklad: Výškový profil

```
def elevation(heights):
    ascent, descent = 0, 0
    for i in range(len(heights) - 1):
        diff = heights[i + 1] - heights[i]
        if diff > 0:
            ascent += diff
        else:
            descent += -diff
    print("Total ascent:", ascent)
    print("Total descent:", descent)
```

Závěr

Co jsme se dnes dozvěděli?

- proměnné v Pythonu jsou ve skutečnosti *odkazy*
  - nemá to vliv na neměnné typy (čísla, řetězce, ntice)
  - má to vliv na měnitelné typy (seznamy, ...)
- seznamy v Pythonu a k čemu se dají použít
  - libovolný počet položek
  - různých typů
  - dají se modifikovat, rozšiřovat
- ntice v Pythonu
  - neměnná varianta seznamů
  - použití: vracení více hodnot z funkce

Co bude příště?

- jednoduché algoritmy „s myšlenkou“
- vyhledávání, řazení