Python

Proměnné a operátory, řízení běhu programu

VŠCHT

2019

Python - obecné informace

Python - zásadní vlastnosti

- je to case-sensitivní jazyk
- naprosto zásadní je pro Python odsazení, obvykle se používá tabelátor
- komentář v řádku začíná znakem #, mezi komentářem a kódem by měli být dvě mezery, mezi znakem # a komentářem jedna mezera
- blokové komentáře lze dělat pomocí znaků """blok komentáře """, existují nástroje, které z blokových komentářů vytvoří dokumentaci ke kódu a lze v komentářích využívat různá klíčová slova (pro zájemce viz Sphinx)
- **styl psaní kódu v Pythonu je specifikován dokumenten PEP8** - https://www.python.org/dev/peps/pep-0008/
- kód který nedodržuje PEP8 je špatný kód, i když funkcionalita je správná

POZOR

Kódy v materiálech nejsou určeny k použití metodou copy/paste.
Pro vyzkoušení si stáhněte přiložené kódy u každého tématu!

Vytváření proměnných

Proměnné

Na rozdíl od většiny jazyků, není v Pythonu potřeba deklarovat typ proměnné. Stačí přiřadit k názvu proměnné požadovanou hodnotu. Proměnné lze i smazat, k tomu slouží příkaz *del*.

```
some_int = 7
some_float = 7.0
my_complex = 2 + 3.5j
some_string = "Hello world"
my_tuple = (10, "Some string", 6.666)
my_list = [10, "Some string", 6.666]
my_dictionary = {"name": "John",
                 "age": 50}
my_bool = True
del some_int
```

Pravidla a konvence pro pojmenování proměnných

Povinně

- proměnné musí začínat písmenem nebo podtržítkem
- proměnná nesmí začínat číslem
- proměnná může obsahovat pouze alfanumerické znaky a podtržítko
- jména jsou case-sensitivní

Konvence

- proměnné používají malá písmena
- slova jsou odděleny podtržítky
- pojmenování všech proměnných, funkcí atd. je v anglickém jazyce
- pro konstanty používáme velká písmena, slova oddělená podtržítky

Numerické proměnné - int, float

Integer

- slouží k uchování celých čísel
- na 64-bit platformě lze uložit čísla v rozmezí –9223372036854775808 až 9223372036854775807

Float

- slouží k uchování čísel s desetinným rozvojem
- lze uložit čísla v rozmezí $2.2250738585072014 \cdot 10^{-308}$ až $1.7976931348623157 \cdot 10^{308}$

Číselné typy lze mezi sebou libovolně konvertovat:

```
my_int = 7
my_float = float(my_int)
other_float = 8.7
other_int = int(other_float)
```

Komplexní čísla

Komplexní čísla v Pythonu

- můžeme pracovat s reálnou a imaginární částí komplexního čísla
- můžeme vytvořit číslo komplexně sdružené.

```
complex_number = 3 + 5.4j
print(complex_number)
print(complex_number.real)
print(complex_number.imag)
print(complex_number.conjugate())
```

Pro zájemce

Všechna komplexní čísla jsou uložena jako dvojice float čísel, reprezentující reálnou a imaginární část v kartézském souřadnicovém systému. Pro práci s komplexními čísly lze použít modul **cmath**, který obsahuje celou řadu funkcí.

Aritmetické operátory

Aritmetické operátory v Pythonu

operátor	funkcionalita	příklad
+	součet hodnot	a + b
-	rozdíl hodnot	a - b
*	násobení hodnot	a * b
/	dělení hodnot	b / a
%	zbytek po dělení	b % a
**	umocní a na b	a ** b
//	celočíselné dělení	a // b

```
a, b = 2, 3
print(a + b, a - b, a * b)  # prints 5 -1 6 1.5
print(b / a, b % a, a ** b)  # prints 1.5 1 8
print( b // a)  # prints 1
```

Relační operátory

Relační operátory operátory v Pythonu

operátor	funkcionalita	příklad
==	rovnost hodnot	a == b
!=	různé hodnoty	a != b
>	vetší než	a >b
<	menší než	a <b
>=	větší nebo rovnost	a >= b
<=	menší nebo rovnost	a <= b

```
a, b = 2, 3
print(a == b, a != b)  # prints False True
print(a > b, a < b)  # prints False True
print( a >= b,  a <= b)  # prints False True
```

Řetězce

String

- řetězce jsou v Pythonu tvořeny poli bytů reprezentujících jednotlivé znaky ve formátu Unicode a nelze je měnit
- pole jsou indexována od 0 (lze použít i záporného indexování, tj. od konce)
- chceme-li vytvořit proměnnou obsahující jeden znak, vytvoří se pole o délce 1 obsahující požadovaný znak, v Pythonu neexistuje char
- pro zjištění délky řetezce slouží funkce *len*()
- pro převedení argumentu na řetězec slouží funkce *str*()

```
my_string = "Hello World."
print(my_string[1])  # prints letter e
print(my_string[1:3])  # prints letters el
print(len(my_string))  # prints length of my_string
print(str([1, 2]))  # prints [1, 2]
```

Řetězce - další vybrané funkce

Základní funkce pro práci s řetězci

- převedení řetězce na malá písmena - *lower*
- převedení řetězce navelká písmena - *upper*
- nahrazení vybraného znaku v řetězci - *replace*
- rozdělení řetězce podle vybraného znaku - *strip*
- odstranění mezer na začátku a konci řetězce - *replace*
- počet výskytů v řetězci - *count*

```
my_string = "Hello, World."
print(my_string.lower())  # prints hello, world.
print(my_string.upper())  # prints HELLO, WORLD.
print(my_string.replace('l', 'x'))  # prints Hexxo, Wordxd.
print(my_string.split(","))  # prints ['Hello', ' World.']
other_string = "    Other string   "
print(other_string.strip())  # prints Other string
print(my_string.count('l'))  # prints 3
```

Řetězce - operátory

Operátory pro práci s řetězci

- test výskytu znaku - *in*
- spojení řetězců - $+$
- opakování znaku *n*-krát - $*$
- iterování řetězcem po jednotlivých znacích- *for character in string*

```
my_str = "abc"
print("b" in my_str)  # prints True
print(my_str + "de")  # prints abcde
print(2 * my_str)  # prints abcabc
for character in my_str:
    print(character)
# prints a
# prints b
# prints c
```

Pole, seznam, n-tice, množina

Rozdíly

- pole (array) obsahuje prvky se stejným datovým typem, jednotlivé hodnoty lze po vytvoření modifikovat, s poli lze pracovat prostřednictvím modulu **array**
- seznam (list) obsahuje hodnoty různých datových typů, jednotlivé hodnoty lze po vytvoření modifikovat
- n-tice (tuple) obsahuje hodnoty různých datových typů, po vytvoření nelze jednotlivé hodnoty modifikovat, chceme-li modifikovat vybranou hodnotu, musíme vytvořit tuple znovu
- množina (set) obsahuje jedinečné hodnoty, jednotlivé hodnoty nelze modifikovat
- tuple lze převést na seznam (*list*) a seznam lze převést na tuple (*tuple*)

Seznamy

Seznamy (list)

- seznam může obsahovat i další seznamy, případně seznamy seznamů
- seznamy jsou indexovány od 0 (lze použít i záporného indexování, tj. od konce)
- je možné měnit ucelené části seznamu
- pro zjištění délky seznamu slouží funkce *len*()
- pro smazání prvku ze seznamu slouží funkce *del*

```
my_list = ["a", 1.5, 2, [3, "b"]]
print(my_list[-1])  # prints [3, 'b']
print(my_list[0:2])  # prints ['a', 1.5]
del (my_list[1])  # delete second item
print(my_list)  # prints ['a', 2, [3, 'b']]
my_list[0:2] = ["x", "y"]  # replace 2 items
print(my_list)  # prints ['x', 'y', [3, 'b']]
print(len(my_list))  # prints 3
```

Seznamy - další vybrané funkce

Základní funkce pro práci se seznamy

- odstranění prvku ze seznamu - *list.pop(index)*
- vložení prvku na konec seznamu - *list.append(item)*
- vložení prvku na pozici v seznamu - *list.insert(index, item)*
- přidání prvků ze seznamu A do seznamu B - *A.extend(B)*
- obrácení pořadí prvků v seznamu - *list.reverse()*
- setřídění seznamu podle velikosti - *list.sort()*
- počet výskytů v seznamu - *list.count(item)*
- součet seznamu - *sum(list)*
- nalezení nejmenšího prvku řetězce - *min(list)*
- nalezení největšího prvku řetězce - *max(list)*

Seznamy - další vybrané funkce - příklady

```
my_list = [1, 2.5, 4, 1]
print(min(my_list))  # prints 1
print(max(my_list))  # prints 4
print(sum(my_list))  # prints 8.5
print(my_list.count(1))  # prints 2
my_list.sort()
print(my_list)  # prints [1, 1, 2.5, 4]
my_list.append(0)
print(my_list)  # prints [1, 1, 2.5, 4, 0]
my_list.extend([7, 8])
print(my_list)  # prints [1, 1, 2.5, 4, 0, 7, 8]
my_list.insert(0, 100)
print(my_list)  # prints [100, 1, 1, 2.5, 4, 0, 7, 8]
my_list.pop(-2)
print(my_list)  # prints [100, 1, 1, 2.5, 4, 0, 8]
my_list.remove(100)
print(my_list)  # prints [1, 1, 2.5, 4, 0, 8]
my_list.reverse()
print(my_list)  # prints [8, 0, 4, 2.5, 1, 1]
```

Seznamy - operátory

Operátory pro práci se seznamy

- test výskytu prvku - *item in list*
- spojení seznamů - $list1 + list2$
- opakování seznamu *n*-krát - $n * list$
- iterování řetězcem po jednotlivých znacích- *for item in list*
- odstranění prvku z *k*-té pozice - $del\ list[k]$
- nahrazení prvku na pozici k hodnotou x - $list[k] = x$

```
my_list = [1, 2, "a"]
my_list2 = [3]
print(1 in my_list)  # prints True
print(my_list + my_list2)  # prints [1, 2, 'a', 3]
print(3 * my_list2)  # prints [3, 3, 3]
del my_list[-1]
print(my_list)  # prints [1, 2]
my_list[0] = 100
print(my_list)  # prints [100, 2]
for item in my_list:
    print(item)
# prints 100
# prints 2
```

Množina (set)

Množiny (set)

- v množinách nezáleží na pořadí prvků
- hodnoty se nemohou opakovat a nelze je měnit
- prázdnou množinu vytvoříme příkazem $A = set()$
- počet prvků množiny lze zjistit pomocí funkce *len*
- s množinami lze provádět základní množinové operace (sjednocení (|), průnik (&), rozdíl (−))

```
my_set = {1, 2, "a"}
my_set2 = set([4, 5, 1])
print(my_set | my_set2)  # prints {1, 2, 4, 5, 'a'}
print(my_set & my_set2)  # prints {1}
print(my_set - my_set2)  # prints {2, 'a'}
print(len(my_set))  # prints 3
```

Množiny - další operátory a funkce

Některé operátory a funkce pro práci s množinami

- test existence prvku v množině - *item in set*
- přidání prvku do množiny - *set.add(item)*
- odstraní prvek z množiny bez vyvolání vyjímky - *set.discard(item)*
- odstranění prvku z množiny, při neexistenci prvku vyvolá vyjímku - *set.remove(item)*
- odstranění náhodného prvku z množiny - *set.pop()*
- odstranění všech prvků z množiny - *set.clear()*
- přidání prvků z množiny B do množiny A - *A.update(B)*
- náhodná iterace množinou - *foriteminset*
- ověření, že množina A je podmnožinou B - $A < B$

Množiny - další operátory a funkce - příklad

```
set1 = set([1, 2, 3])
set2 = set([1, 2, 3, 4])
print(1 in set2)  # prints True
print(set1 < set2)  # prints True
set1.update(set2)
print(set1)  # prints {1, 2, 3, 4}
set2.pop()
print(set2)  # prints {2, 3, 4}
set2.remove(3)
print(set2)  # prints {2, 4}
set2.add("a")
for item in set2:
    print(item)
# prints 2
# prints 4
# prints a
set2.clear()
print(set2)  # prints set()
```

Řízení toku programu - if

If

- příkaz *if* slouží k podmíněnému větvení programu, určitý blok kódu se tedy vykoná, pokud je podmínka uvedená za *if* splněná
- vyhodnocením podmínky musí vzniknout logická hodnota, tj. *True* nebo *False*
- pomocí příkazu *elif* lze větvit do více než dvou větví

```
a = 2
if a == 1:
    print("code if a == 1")
elif a == 2:
    print("code if a == 2")
else:
    print("code if a is something else than 1 or 2")
# this code prints code for a == 2
```

Řízení toku programu - for

For

- narozdíl od většiny programovacích jazyků umožňuje příkaz *for* iterovat přes libovolnou sekvenci
- pomocí for lze iterovat i přes subsekvenci zvolené sekvence
- pomocí funkce *range*() lze iterovat přes sekvenci čísel (*range*() je tzv. generátor)

```
seq1 = ["apple", 2, 3.5]
for item in seq1:
    print(item)  # prints on new lines: apple, 2, 3.5

for item in seq1[1:]:
    print(item)  # prints on new lines: 2, 3.5

for number in range(2, 4):
    print(number)  # prints on new lines: 2, 3
```

Řízení toku programu - while, break

while a break

- příkaz *while* umožňuje vykonávat blok kódu, dokud je podmínka uvedená za tímto příkazem platná
- při používání toho příkazu lze vytvořit nekonečnou smyčku, proto je potřeba dbát opatrnosti při jeho použití
- příkaz *break* umožní ukončit for nebo while smyčku

```
a, b = 0, 0
while a < 3:
    print(a)
    a += 1  # this loop prints on new lines: 0 1 2
while b < 3:
    print(b)
    b += 1
    if b == 1:
        break  # this loop prints 0 only
```