02 Input, konverze a práce s textem

PVA2 Programování a vývoj aplikací

Adam Fišer

Obsah

input()

```python
# Příkaz input() vyžádá zadání vstupu od uživatele
input("Zpráva pro uživatele")

# Vyžádání vstupu a uložení do proměnné username
username = input("Zadej uživatelské jméno: ")

print(username) # Vrátí zadanou hodnotu uživatele
```

Konverze datových typů

Konverze

```
int(x)    # konverze na integer
float(x)  # konverze na float
str(x)    # konverze na string


floatNumber = 14.0
print(floatNumber) # 14.0
print(int(floatNumber)) # 14
print(type(int(floatNumber))) # <class 'int'>
```

Konverze a text

```
number = 9.0
result = number / 2 #dělení
remainder = number % 2 #modulo

# skládání textového řetězce využitím
print("result = " + str(result)) # 4.5
print("remainder = " + str(remainder)) #1
```

Konverze vs. input

- Pozor na vstup od uživatele s využitím příkazu input
- Zadaná hodnota je vždy datového typu string
- U čísel je vždy nutná konverze

Rozšířené přiřazení a operace

Rozšířené přiřazení

- Rozšířené přiřazení je jediný příkaz kombinující binární operaci a příkaz přiřazení, například +=, -= atd.
- Rozšířený přiřazovací výraz jako x += 1 lze přepsat jako x = x + 1, čímž se dosáhne podobného efektu.

```
1   number = 9.0
2   print("number = " + str(number))
3
4   number -= 2
5   print("number = " + str(number)) # 7.0
6
7   number += 5
8   print("number = " + str(number)) # 12.0
```

Binární operace

- Binární operace jsou operace, které vyžadují dva operandy.
- Operandy jsou hodnoty, na kterých se provádí operace.
- Binární operace mohou být aritmetické, relační, logické nebo bitové.
- Aritmetické operace: sčítání, odčítání, násobení, dělení, modulo, mocnina
- Relační operace: rovná se, nerovná se, menší než, větší než, menší nebo rovno, větší nebo rovno
- Logické operace: a, nebo, negace
- Bitové operace: a, nebo, negace, posun
- Příklady: `+`, `-`, `*`, `/`, `%`, `**`, `==`, `!=`, `<`, `>`, `<=`, `>=`, `and`, `or`, `not`, `&`, `|`, `~`, `<<`, `>>`

```
two = 2
three = 3
isEqual = two == three
print(isEqual) #False
```

Logické binární operace

- `<` menší než
- `>` větší než
- `==` rovná se
- `>=` větší nebo rovno
- `<=` menší nebo rovno
- `!=` nerovnají se

```
1   one = 1
2   two = 2
3   three = 3
4
5   # Toto zřetězené porovnání znamená, že (jedna < dvě) a (dvě < tři)
6   # vyhodnocení porovnání se provádí současně.
7   print(one < two < three)
8
9   isGreater = three > two
10  print(isGreater)
```

Zpracování textu

Řetězení textu

```
hello = "Hello"
world = 'World'

hello_world = hello + " " + world
print(hello_world)      # Hello World
```

Multiplikace

```
1  hello = "hello"
2  tenOfHellos = hello * 10
3  print(tenOfHellos) # hellohellohellohellohellohellohellohellohellohello
```

Indexování řetězců

- Ke znaku v řetězci můžete přistupovat, pokud znáte jeho pozici.
- Například příkaz strindex zobrazí znak na pozici index v řetězci str.
- Indexování řetězců začíná vždy od 0.
- Index vyvolá chybu ValueError, pokud se x v řetězci nenachází.
- Indexy mohou být také záporná čísla, pokud potřebujete začít počítat zprava (tj. od konce řetězce).
- Všimněte si, že jelikož -0 je totéž co 0 , záporné indexy začínají od -1.

```
1  text = "This is a very long string!"
2
3
4  firstLetter = text[0] # Indexování začíná od 0
5  print(firstLetter) #T
6
7  lastLetter = text[-1] # !
8  print(lastLetter) # !
```

Podřetězce

```
str[start:end] # položky start až end-1
str[start:] # položky od začátku do konce pole
str[:end] # položky od začátku do konce-1
str[:] # kopie celého pole
```

```
monty_python = "Monty Python"
monty = monty_python[:5] #ekvivalentní zápis monty_python[0:5]
print(monty) #Monty

python = monty_python[6:]
print(python) #Python
```

Operátor `in`

```
ice_cream = "ice cream"
print("cream" in ice_cream)    # Boolean True

contains = "ice" in ice_cream
print(contains) #True
```

Délka řetězce

- Délka řetězce je vypočtena příkazem `len()`
- Příkaz `len()` vrátí počet znaků v řetězci

```
1   text = "Příliš žluťoučký kůň úpěl ďábelské ódy"
2   textLenght = len(text)
3   print(textLenght) #38
```

Escape sekvence

- Escape sekvence jsou speciální znaky, které se používají k vytvoření speciálních znaků v řetězci.
- Například znak nového řádku se zapisuje jako `\n`
- Znak tabulátoru se zapisuje jako `\t`
- Znak zpětného lomítka se zapisuje jako `\\`
- Zpětné lomítko se používá k vynechání speciálních znaků
- "It's me"
- "She said "Hello""
- Pokud chcete vypsat zpětné lomítko, musíte jej také escapovat tzn. print('\')
- Speciální symbol '\n' je používán pro nový řádek a '\t', jako tabelátor

Formátování textu

Metody formátování textu

```
1   text = "Příliš žluťoučký kůň Řehoř úpěl ďábelské ódy"
2   print(text.lower()) # všechny znaky budou malé
3
4   name = "John".upper() # JOHN
5   print(name)
```

- Metoda `format()` umožňuje vkládat proměnné do řetězce
- Proměnné jsou vkládány do řetězce pomocí složených závorek `{}`

```
1   TODO
```

Operátor %

Operátor % za řetězcem slouží ke spojení řetězce s proměnnými.

```
1  name = "John"
2  print("Hello, PyCharm! My name is %s!" % name) # %s pro text
3
4  age = 10
5  print("I'm %d years old" % age) # %d pro celá i reálná čísla
```

F-string

- Formátovaný řetězcový literál neboli f-řetězec je řetězcový literál s předponou 'f' nebo 'F'.
- F-string je způsob formátování řetězců, který umožňuje vkládat proměnné do řetězce pomocí složených závorek `{}`
- Řetězce mohou obsahovat náhradní pole, což jsou výrazy ohraničené složenými závorkami .

```
1   name = "Ginger and Fred"
2   age = "20"
3   print(f"Hello, My name is {name} and I'm {age} years old.")
```

Děkuji za pozornost

Otázky?

Repository / Prezentace