IB111 Úvod do programovaní skrze Python

Autor: Slavomír Krupa,
Text inšpirovaný: Valdemarom Švábenským


PREMENNÁ

Premenná

- ★ Miesto v pamäti, kde je uložená hodnota pod nejakým aliasom
- ★ Zmysluplné a výstižné meno
  - nie magic_variable
- ★ Obvykle malé písmená, čísla a podčiarkovníky
- ★ Nie rezervované slová
- ★ Anglicky
- ★ Priraďovací príkaz: premenná = hodnota

```
loop_count = 0
```

Premenná

★ Hodnota sa dá meniť

```
loop_count = 0
print(loop_count)
loop_count = 3
print(loop_count)
loop_count = "Text"
print(loop_count)
```

```
0
3
Text
```

Dátový typ

- ★ Množina hodnôt spolu s operáciami na týchto hodnotách
- ★ Celé číslo (int): 42, 0, -14
- ★ Desatinné číslo (float): 42.0, 0.21, -7.59876, 3e8
- ★ Logická hodnota (boolean): True, False
- ★ Reťazec (string): "The answer is 42!", '0.21'
- ★ Zoznam (list): [1, 4, 9, 16, 25]
- ★ ...
- ★ Python: implicitné typovanie (typ sa neuvádza)

Dátové typy

★ Typ sa dá zistiť...

```
type(1)
type("1")
type(1.0)
```

```
<class 'int'>
<class 'str'>
<class 'float'>
```

Dátové typy

★ ...alebo zmeniť

```
type(int("1"))
type(str(1))
type(str(1.0))
type(int(1.7))
```

```
<class 'int'>
<class 'str'>
<class 'str'>
<class 'int'>
```

???


Operátory

Príklady

```
a = 5 + 2 # 5+2
b = 5 - 2 # 5 -2
c = 5 * 2 # 5*2
d = 5.0 // 2#5.0//2
e = 5 / 2 # 5/2
f = 5 % 2 # 5%2
g = 5 ** 2 # 5**2
```

```
a = 7
b = 3
c = 10
d = 2
e = 2.5
f = 1
g = 25
```

Syntax sugar

```
sum = 0
sum = sum + 1
sum = sum + 2
sum = sum + 3
```

```
sum = 0
sum += 1
sum += 2
sum += 3
```

Analogicky pre operátory:

+, - ,* , / , % , ** , //

Porovnávanie

```
a = 4 == 4
b = 8 != 5
c = 9 < 4
d = 9 <= 4
e = 5 > 3
f = 5 >= 5
```

```
a = True
b = True
c = False
d = False
e = True
f = True
```

Logické spojky

```
a = True and True
b = True and False
c = False and False
d = True or True
e = True or False
f = False or False
g = not True
h = not False
i = not not True
```

```
a = True
b = False
c = False
d = True
e = True
f = False
g = False
h = True
i = True
```

Kombinácia

```
"10" == str(2 * 5) and 5 ** 4 >= 200
```

- Aritmetické
- Relačné
- Logické
- …. radšej použiť zátvorky.

VÝSTUP

Výpis

```
print("Text")
print(1)
print("a" + "b")
print("a", "b")
print("This","won't","be",end=" ")
print("ended with enter.")
```

Formátovanie textu

```
text = "Hello my name is {} and I come from {}.".format("Slavo", "Slovakia")
print(text)
```

```
Hello my name is Slavo and I come from Slovakia.
```


VETVENIE

Basic if

- ★ Riadiaci príkaz
- ★ Umožňuje vykonanie bloku príkazov v závislosti na splnení podmienky
- ★ Podmienka: logický výraz
  - Pozor na správne a konzistentné zanorenie
- ★ Nesplnená podmienka (False): zanorené príkazy sú preskočené
- ★ `False == 0, 0.0, "", [], (), {}, None, ...`

```
if age >= 18:
    print ("You can legally buy alcohol.")
```

If-else

- ★ Obohacuje príkaz if
- ★ Splnená podmienka: vykonajú sa príkazy vo vetve if
- ★ Nesplnená podmienka: vykonajú sa príkazy vo vetve else
- ★ Vzájomne výlučné

```
if rating > 80:
    print("Should be good movie.")
    print("Rating:",rating)
else:
    print("I wouldn't recommend it.")
# continue...
```

If-elif

```
salary = 1000
if salary < 5000:
    tax = 0.05
elif salary < 20000:
    tax = 0.15
elif salary < 40000:
    tax = 0.30
else:
    tax = 0.50
```

Diagram

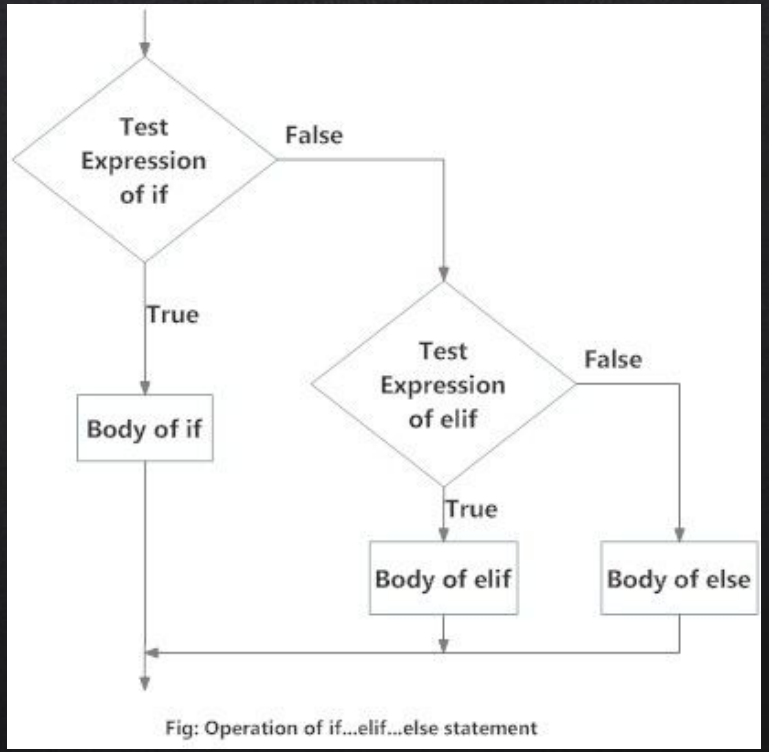


Fig: Operation of if...elif...else statement

Zdroj: http://www.programiz.com/python-programming/if-elif-else

Cyklus

Cyklus

- ★ Opakované vykonanie bloku príkazov
- ★ Cyklus for: s čítačom; známy počet opakovaní
- ★ Cyklus while: s logickou podmienkou; neznámy počet opakovaní
- ★ Cyklus do-while: v Pythone nie je

For

- ★ Opakovanie kódu určený počet krát
- ★ Premenná i postupne nadobúda celočíselné hodnoty z rozsahu [0, 3)
- ★ Odsadené príkazy sa postupne vykonajú pre každú hodnotu, ktorú i nadobudne

```
for i in range(3):
    print(i)
print("I will run just once.")
```

For

- ★ range(a, b) - rozsah [a, b) (na celých číslach)
- ★ range(b) - rozsah [0, b) (na celých číslach)
- ★ range(down,up,step) - rozsah [down,up) - s krokom step
- ★ Python často v hornom indexe má prvú hodnotu, ktorú už “nepoužijeme”

```
for i in range(1,10,2):
    print(i, end=" ")
# 1 3 5 7 9
```

While

- ★ Opakovanie kým platí podmienka
- ★ Môžu nastať nekonečné cykly
- ★ Nemusí sa vykonať
- ★ V cykle by sa mala meniť premenná by nastala neplatnosť podmienky

```
i = 0
while i < 3:
    print(i)
    i += 1
```

```
for i in range(3):
    print(i)
```

Kontrola vstupu

```
choice = input('Like the course ? (y/n)')
while choice != 'y' and choice != 'n':
    choice = input("Unexpected input. Enter again:")
```

BREAK

```
i = 0
while True:
    print(i)
    i += 1
    if i >= 3:
        break  # Jump out of 'while' block
print("All right. Off you go.")
```

Continue

```
i = 5
while i > 0:
    i -= 1
    if i == 3:
        continue  # Jump to condition
    print(i)
print("All right. Off you go.")
```


PROCEDÚRY

Procedúry

- ★ Kľúčové slovo def
- ★ Medzera
- ★ Meno funkcie
- ★ Zoznam parametrov (v zátvorkách)
  - ○ Bez parametrov = prázdne zátvorky
- ★ Dvojbodka
- ★ Blok funkcie
  - ○ Odsunutý

```
def procedure(param1, param2):
    do_something(param1)
    ...
    do_something(param2)
```

Procedúry

- Procedúra by mala mať zmysluplné a výstižné meno
  - Nie my_super_function()
- Je dobrým zvykom komentovať chovanie funkcie
  - Tzv. docstring

```
def greet(name):
    """
    Greets someone
    :param name: name of the person to be greeted
    """
    print(" Hello {} !".format(name))
```

Procedúry

- ★ Funkčné volanie: použitie mena procedúry a zadanie všetkých potrebných parametrov
- ★ Spôsobí vykonanie kódu vo procedúre
- ★ “Odovzdá” parametre a riadenie volanej procedúre

```
greet("students")
```

But why?

- ★ Možnosť písať zložitejšie/rozsiahlejšie programy
- ★ Opravy na jednom mieste
  - Pri copy paste všade kde ste skopírovali
- ★ Znovupoužiteľnosť
- ★ Ľahšie sa číta / lepšie to vyzerá
  - Ak je správne pomenované...

Default values

```
def foo(x, y=3):
    print("x =", x)
    print("y =", y)
foo(1)
foo(2,5)
```

```
x = 1
y = 3
x = 2
y = 5
```

Volanie menom

```
def foo(x, y=3):
    print("x =", x)
    print("y =", y)
foo(y=1, x=4)
foo(x=5)
print("x",end=" ")
```

```
x = 4
y = 1
x = 5
y = 3
x
```

Procedúry

```
def p(a1, a2, a3):
    if a2 >= 0.5:
        print('Wow, that is bright!')
    if a3 >= 0.5:
        print("Wow, that is light!")
p(170, 0.1, 0.6)
```

Bill saw code like this...

Procedúry

```
def color_info(hue, saturation, luminosity):
    if saturation >= 0.5:
        print('Wow, that is bright!')
    if luminosity >= 0.5:
        print("Wow, that is light!")
color_info(hue=170, saturation=0.1, luminosity=0.6)
```

… and turn it into this. Be like Bill.

Rozsahy

- V ktorom bloku je viditelná aká premenná?
- Globálne premenné viditeľné všade

```
t = Turtle()
def move():
    t.forward(100)
```

Rozsahy

Lokálne len v rámci bloku

```
if price > 10:
    tmp = price
    price *= 0.8
    print("Discount from: {} to  {} ".format(str(tmp), str(price))
print("Print original price:", str(tmp)) #error
```

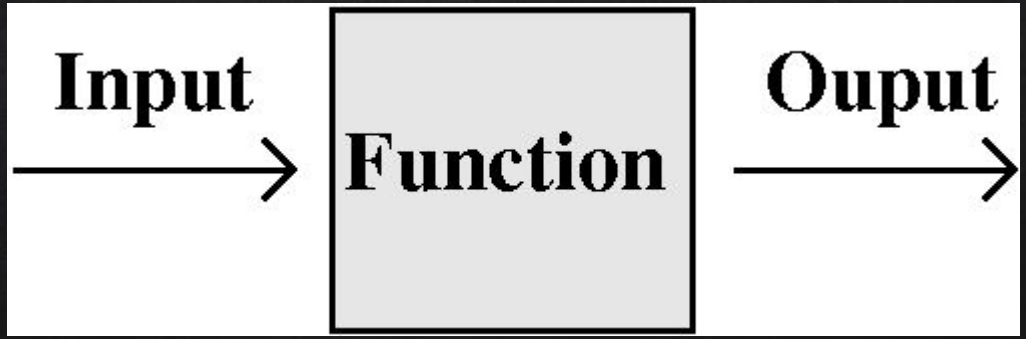
Rozsahy

```
var = "awesome"

def shadow():
    var = "boring"
    commend(var)
    if True:
        var = "interesting"
        commend(var)

shadow()
commend(var)
```


Funkcie

Funkcie

- ★ Rozdiel oproti procedúre - funkcia má návratovú hodnotu
- ★ Vykonávanie funkcie môže byť prerušené okamžite pomocou return



Zdroj: https://www.math.rutgers.edu/~greenfie/summer_135/diary.html

Volanie funkcie

```
def resolve_tax(salary):
    if salary < 5000:
        return 5
    elif salary < 20000:
        return 15
    elif salary < 40000:
        return 30
    else:
        return 50
```

If-elif

```
salary = 1000
if salary < 5000:
    tax = 0.05
elif salary < 20000:
    tax = 0.15
elif salary < 40000:
    tax = 0.30
else:
    tax = 0.50
```

Volanie funkcie

```
paid_tax = salary * resolve_tax(salary)
```

Volanie funkcie

```
def divisibility5(n):
    return n % 5 == 0

a = divisibility5(1)
b = divisibility5(20)
c = divisibility5(4)

if(divisibility5(5)):
    #...
```

```
a = False
b = True
c = False
```

Vstavané funkcie

- Napr. `print()`
- Python ich pozná vždy
  - Nie je nutné robiť import
- `min()`
- `max()`

Math

- Je nutné načítanie pomocou
  - `from math import *`
- Zaokrúhľovanie: `round()`, `ceil()`, `floor()`
- Goniometria: `sin()`, `cos()`,...
- Mocniny a logaritmy: `exp()`, `log()`, `sqrt()`, ...


Debug

Úloha 1

- Faktoriál alebo súčet prvých N čísel
  - pomocou for
    - s výpisom
    - v návratovej hodnote funkcie
  - pomocou while
    - s výpisom
    - v návratovej hodnote funkcie
- `factorial(5) = 120`
- `sum_of_numbers(100) = 5050`

Úloha 2

- Naprogramujte funkciu, ktorá vráti počet (kladných) deliteľov celého čísla `n`
- `divisor_count(18) = 6`
  - `1,2,3,6,9,18`

Úloha 3

- Naprogramujte funkciu, ktorá vráti či dané číslo je prvočíslo (skúste použiť predošlú funkciu)
- Prvočíslo má vždy dva delitele - 1 a samého seba
- Návratová hodnota bude: Boolean - True/False
  - `is_prime(7) = True`
  - `is_prime(1) = False`
  - `is_prime(4) = False`

Úloha 4

- Naprogramujte funkciu, ktorá vráti prvých n prvočísiel (opäť skúste použiť niečo čo už máte)
- Napr. `n_primes(4) = 2 3 5 7`

Úloha 5*

- Naprogramujte funkciu, ktorá vráti najväčší spoločný deliteľ
- `gcd(7,33) = 1`
- `gcd(5, 10)= 5`
- `gcd(42, 36)= 6`

Úloha 6*

- Naprogramujte funkciu, ktorá vráti najmenší spoločný násobok
- `lcm(9,3) = 9`
- `lcm(360,42) = 2520`

Úloha 7*

- Naprogramujte procedúru, ktorá vypíše rozklad na prvočísla
- `factorization(25) = “5*5”`
- `factorization(42) = “2*3*7”`

Zbierka

- https://www.fi.muni.cz/IB111/sbirka/03-jednoduche_vypocty.html
- Začať 3.4 prevodmi medzi sústavami
- Ďalej
  - Vynechané príklady
  - 3.3 aproximácie

Credits

Special thanks to all the people who made and released these awesome resources for free:

- Presentation template by SlidesCarnival
- Photographs by Unsplash