Dátové typy

- Do premennej môžeme uložiť hocijaký dátový typ
- základné dátové typy v Pythone:
  - number (int celé čísla, float desatinné čísla)
  - String (reťazec)
  - Bool (hodnota True alebo False, alebo aj 1 alebo 0)
  - list
  - tuple
  - dictionary

Dátové štruktúry (pre efektívne ukladanie dát)

- Ďalšie dátové typy: complex, bytes, bytearray, ...
- *type(variable)* vráti typ premennej v tele type()
- V príklade je možné vidieť rôznu definíciu a výpis premenných

```
num1 = 5
num2 = 8
print(num1 * num2)

my_string = "This is string"
print(my_string)
print(my_string[0])
print(my_string[0:7])
print(my_string[2:])
print(my_string[-1])

var1, var2, var3 = 100, "string", -10
print(var1, var2, var3)
```

Dátové typy - numerické

- Numerické dátové typy sa týkajú čísel
- V Pythone existujú 3 typy čísel:
  - Int (celé čísla) -10, 1, 5, 80
  - Float (desatinné čísla) -25.45, 2.15, 21.5, 36.78
  - Komplexné čísla 1j (nebudeme používať)

```
11 + 11
21 - 21
5 * 5
7 / 9
# celočíselné delenie
5 // 5
25 % 5
# absolútna hodnota
abs(-3)
# mocninový zápis
2**4

# V operáciách môžeme použiť aj zátvorky
(21 + 8) / (4 + (12 % 4.0))

# V operáciách môžeme kombinovať int a float
# vráti ty float
11 + 6.75
```

Dátové typy - bool

Dátový typ bool má len 2 hodnoty **True** alebo **False**

```
# Dátový typ bool ma len dve hodnoty True a False
my_true_value = True
my_false_value = False

# Výstupom porovnávacích oeprácií je dátový typ (True/False)
5 < 9
510 > 650
21 == 21
21 <= 30
8 == 8.0
911 != 12

# Na zistenie rovnosti vždy používa dvojité '='
x = 11
11 == 11.0

# porovnávať vieme aj premenné
num1 = 21
num2 = 500
num1 >= num2
```

```
# Bool vieme aj negovať pomocou funkcie not
my_var =True
not(my_var)

# Porovnávacie operácie vieme reťazit logickými operátormi and, or
21 < 22 and 21 > 20
21 < 22 or 21 > 20
not(21 < 22 and 22 > 24)
```

Dátové typy - string

- **String** – reťazec je text uložený v premennej
- string môže byť deklarovaný jednoduchými ‘my_string‘ alebo dvojitými úvodzovkami “my_string“

```
# String môžeme definovat pomocou dvojitých, jednoduchých úvodzoviek
my_variable = "This is my string 1"
my_variable2 =' This is my string 2'

# String vieme vypísať pomocou funkcie print
print(my_variable)
print(21)
print(21.5)
print(True)
number = 81
print(number)
```

```
# String vieme definovať aj na viac riadkov
my_variable ="""
This is
multiline
string.
Cross lines.
"""
print(my_variable)

# \n je symbol na vloženie nového riadku
my_variable="This is string with \nnew line"
print(my_variable)

# Použitím r pred definíciou stringu, nevloží nový riadok
my_path=r"C:\documents\notes"
print(my_path)
```

Dátové typy - string

- V Pythone neexistuje dátový typ char (znak/charakter) tak, ako je to v iných programovacích jazykoch alebo SQL databáze
- V Pythone je 1 znak reprezentovaný ako reťazec (string) dĺžky jeden
- K jednotlivým prvkom stringu sa pristupuje cez **index**
- Prvý znak v reťazci má index 0, potom1, 2, ..., n
- Použitím záporného indexu pristupujme k znakom odzadu, napr. [-1] vráti posledný znak stringu
- Zápisom [-4:] vrátime posledné 4 znaky zo stringu
- Pozor na rozsah stringu, nevyjsť z rozsahu stringu !!!
- Ak chceme do reťazca vložiť nepovolený znak (dvojité/jednoduché úvodzovky, nový riadok, tabulátor), používame escape znak.
- Escape znak je spätná lomka \, za ktorou nasleduje znak, ktorý chceme vložiť (\“, \‘, \n)
- **String pozná rôzne metódy**:
  - Metóda count () vracia počet výskytu hodnoty v reťazci
  - Metóda find () vyhľadá prvý výskyt hľadanej hodnoty
  - Metóda isnumeric () vracia True, ak sú všetky znaky čísla (0-9), inak vráti False
  - Metóda lower () vráti reťazec, v ktorom sú všetky znaky malé
  - Metóda split () rozdelí reťazec na zoznam (list)
  - Metóda strip () odstráni všetky medzery na začiatku a na konci reťazca

```
my_string = "This is my string 1"
print(my_string)
print(my_string[0])
print(my_string[2:])
print(my_string[-1])
print(my_string[-4:])

# Funkcia len vráti dĺžku stringu
print(len(my_string))

# Zreťazenie (sčítanie) stringov
name = "Tomas"
space = " "
surname = "Tatar"
person_name = name + space + surname
print(person_name)

# Bez použitia medzery
print(name, surname)
```

Type casting

- Pomocou type casting vieme špecifikovať typ danej premennej
  - a = int(23), b = int(“56“) - **oba sú type int- celé čísla**
  - c = float(10) – **type float – desatinné číslo**
  - s = str(100), p = str(“100“) – **oba sú type string – reťazec**

- Z príkladu:
  - premenné **a** a **b** sú typ int
  - Premenná **c** je typ float
  - premenné **s** a **p** sú typ string

```
a = int(23)
b = int("56")
c = float(10)
print(type(a), type(b), type(c))

s = str(100)
p = str("100")
print(type(s), type(p))
```

Operátory

- **Základné aritmetické operátory:** +, -, /, //, *, **, %

- **Operátori nerovnosti:** ==, !=, >=, <=, >, <

- **Operátori priradenia:**
  sum = 10
  sum = sum + 5 alebo **sum += 5**
  sum = sum – 5 alebo **sum -= 5** ,... (*=, **=,/=, //=, %=, ...)

- **Logické operátori:**
  and vráti True ak celá podmienka je splnená: x < 5 and x < 10
  or – vráti true ak aspoň 1 časť podmienky je True: x < 5 or x < 4
  not – opačný výsledok, vráti False ak výsledok podmienky je True:
  not(x < 5 and x < 10)

Podmienky

- Python podporuje obvyklé logické podmienky z matematiky:
  - Rovné: a == b
  - Nie je rovné: a != b
  - Menšie ako: a < b
  - Menšie alebo rovné ako: a <= b
  - Väčšie ako: a > b
  - Väčšie alebo rovné ako: a >= b

Za logickou časťou podmienky sa píše dvojbodka

Všetko, čo sa má vykonať v tele podmienky musí byť odsadené tabulátorom

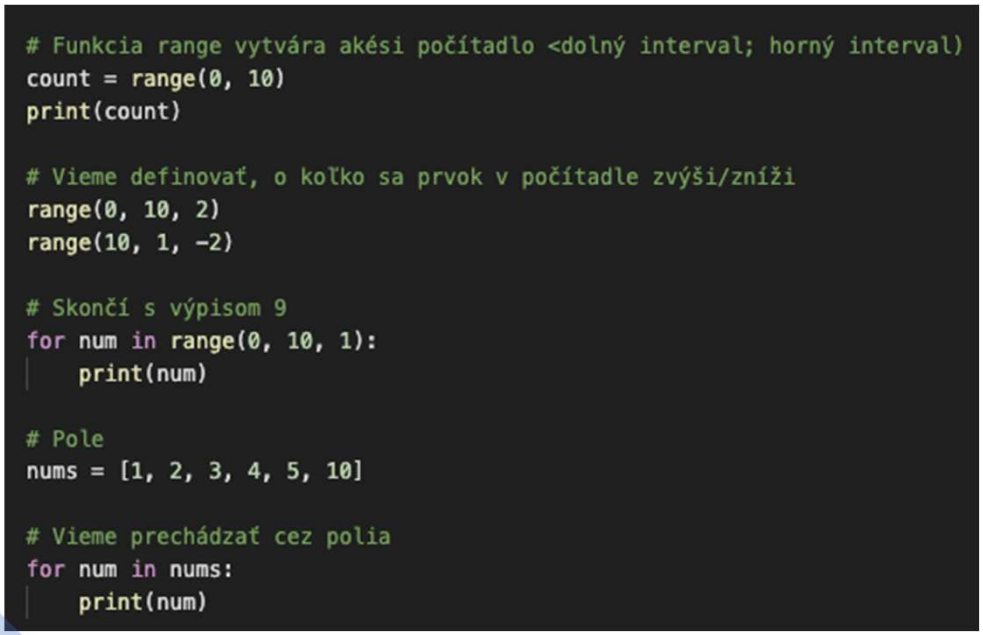
Podmienku s jedným príkazom možno napísať do jedného riadku, *if 15 > 10: print("15 is greater than 1")*

```
a = 100
b = 345

if b > a:
  print("b is greater than a")
elif a == b:
  print("a and b are equal")
else:
  print("a is greater than b")

if a > b: print("a is greater than b")

a = 250
b = 200
# Ternárny operátor
print("A") if a < b else print("B")

print("A") if a > b else print("=") if a == b else print("B")


num = 20

if num % 2 == 0 and num % 3 == 0:
    print("That is it")
```

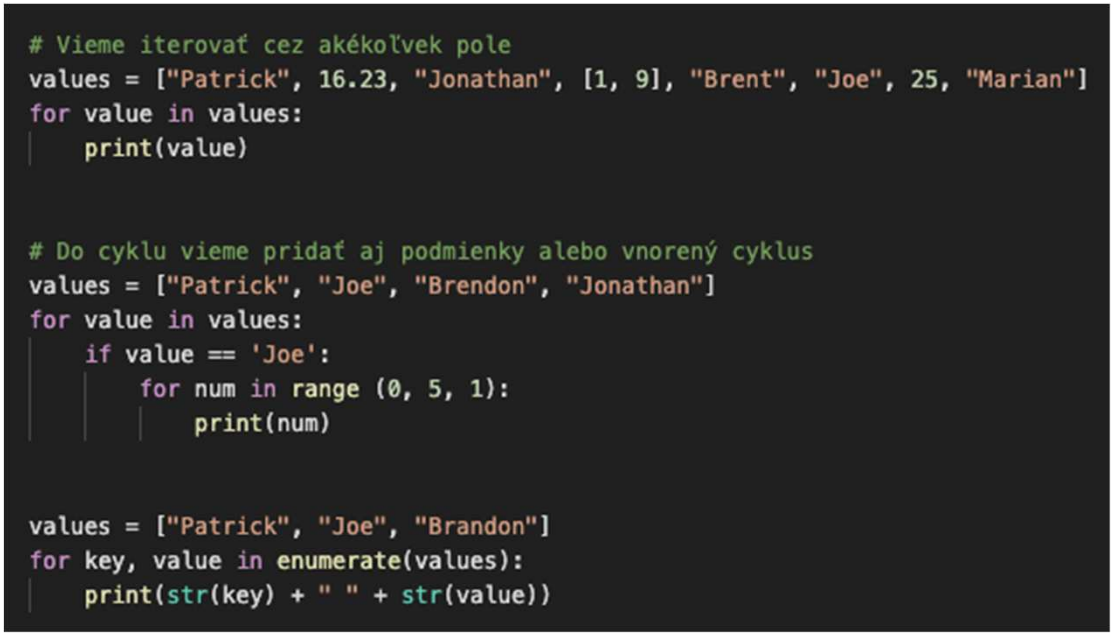
Cykly – for cyklus

- V počítačovej terminológii je cyklus programová štruktúra, ktorá opakuje sekvenciu inštrukcií kódu kým podmienka cyklu je pravdivá
- Podmienka môže byť stále pravdivá, alebo hodnota pre ukončenie cyklu nedosiahnuteľná, vtedy hovoríme o **nekončenom cykle**
- **For cyklus** – definuje začiatok a koniec cyklu, zároveň inkrement pre každú iteráciu
- For cyklus sa používa na iteráciu cez sekvencie (list, tuple, dictionary, set, string, rozsah čísel)
- Funguje viac ako iterátor cez jednotlivé položky sekvencie

Cykly – for cyklus



```python
# Funkcia range vytvára akési počítadlo <dolný interval; horný interval)
count = range(0, 10)
print(count)

# Vieme definovať, o koľko sa prvok v počítadle zvýši/zníži
range(0, 10, 2)
range(10, 1, -2)

# Skončí s výpisom 9
for num in range(0, 10, 1):
    print(num)

# Pole
nums = [1, 2, 3, 4, 5, 10]

# Vieme prechádzať cez polia
for num in nums:
    print(num)
```



```python
# Vieme iterovať cez akékoľvek pole
values = ["Patrick", 16.23, "Jonathan", [1, 9], "Brent", "Joe", 25, "Marian"]
for value in values:
    print(value)


# Do cyklu vieme pridať aj podmienky alebo vnorený cyklus
values = ["Patrick", "Joe", "Brendon", "Jonathan"]
for value in values:
    if value == 'Joe':
        for num in range (0, 5, 1):
            print(num)


values = ["Patrick", "Joe", "Brandon"]
for key, value in enumerate(values):
    print(str(key) + " " + str(value))
```

- Funkciou **enumerate()** iterujeme cez pole a zároveň získavame automatické počítadlo (0, 1, 2, 3, ...)
- v príklade, premenná **key** nadobúda hodnoty 0, 1, 2

Cykly – while cyklus

- While cyklus obsahuje podmienku a funguje podobne ako for cyklus
- opakuje sa slučka, pokiaľ podmienka je pravdivá
- Nekonečný while cyklus napíšeme while True:
- While cyklus je nutné ukončiť, aby nešiel do nekonečna, alebo neukončiť ;)

```
count = 0
while count < 10:
    print(count)
    count += 1

# Nekonečný cyklus dosiahneme pomocou while True
count = 0
while True:
    print(count)
```

Break vs. Continue v cykle

- Kľúčovým slovom **break** v tele cyklu ukončíme cyklus a vyjdeme z tela cyklu von
- Kľúčovým slovom **continue** v tele cyklu ukončíme aktuálnu iteráciu cyklu a pokračujeme s novou iteráciou, krok cyklu sa inkrementuje, cyklus pokračuje ďalej

```python
# Ukončiť cyklus vieme pomocou slova break
count = 0
while True:
    print(count)
    if count >= 10:
        break
    count += 1

# break pre ukončenie vo for cykle
for i in [1,2,3,4]:
    print(i)
    if i == 2:
        break

# continue na ukončenie aktuálnej iterácie, nie celého cyklu
count = 0
while True:
    if count == 6:
        count += 1
        continue
    if count >= 25:
        break
    print(count)
    count += 1

# continue vo for cykle
for value in [1,2,3,4,5,6,7,8,9,10]:
    if value == 8:
        continue
    print(value)
```

Funkcie

- V počítačovom odvetví a v programovaní, **funkcia** je blok inštrukcií, ktorá vykoná špecifickú úlohu

- Pre Python platí:
  - Funkcia je definovaná kľúčovým slovom **def**
  - Funkcii môžeme odovzdať dáta cez tzv. parametre funkcie
  - Funkcia vráti špecifickú hodnotu alebo objekt ako výsledok funkcie (kľúčové slovo **return**), alebo void
  - Kľúčové slovo **yield** vráti sekvenciu hodnôt alebo objektov z funkcie
  - používame ***args**, ak nie je známy počet parametrov funkcie (typ tuple)
  - Vieme definovať kľúčové argumenty, **key -> value**
  - Použitie ****kwargs** ak nie je známy počet kľúčových argumentov funkcie key->value (dictionary)
  - Môžeme definovať defaultné parametre funkcie, umiestnené za povinnými parametrami
  - Telo funkcie odsadené tabulátorom (nepoužívajú sa zátvorky)
  - Telo funkcie nesmie byť prázdne, ak je, používame kľúčové slovo **pass**
  - Názov funkcie pozostáva z malých písmen anglickej abecedy a podtržníku _
  - Názov funkcie by mal obsahovať sloveso

Funkcie poskytujú lepšiu modularitu pre aplikáciu a vysoký stupeň znovapoužiteľnosti kódu

Funkcie


```python
# definícia funkcie s povinnými argumentami name a age, nepovinným player_number
def show_name(name, age, player_number = 99):
    #lokálna premenná
    club = "Chicago Blackhawks"
    print(f"Player name is: {name} with age: {age} and player number: {player_number} in the club: {club}")

show_name("Jonathan Toews", 32)


# definícia funkcie s neznánym počtom argumentov
def show_player_details(*details):
    goals = 2 * details[3]
    print(f"Player name is {details[0]} with age: {details[1]} in the club: {details[2]}")
    print(f"Goals: {goals}")

show_player_details("Brent Seabrook", 31, "Chicago Blackhawks", 29)


def show_players(*players):
    for index in range(len(players)):
        print(players[index])

show_players("Jonathan Toews", "Brent Seabrook", "Corey Crawford", "Erik Gustafsson")


# definícia funkcie s náhodným počtom kľúčvých argumentov key->value
def show_profile(**profile):
    print(f"Player name: {profile['name']} with age: {profile['age']} in th club: {profile['club']}")

show_profile(name = "Erik Gustafsson", age = 28, club = "Chicago Blackhawks")
```

Funkcie

```python
# definícia funkcie s návratovou hodnotou, vráti súčet zoznamu
points = list(range(10))
print(points)

def sum_points(points):
    total = 0
    for point in points:
        total += point
    return total

sum_points(points)
sum(points) # Python funkcia


# definícia funkcie s viacerými návratovými hodnotami
# vráti Tuple
def score(point):
    if point % 2 == 0:
        return 1, True
    else:
        return 0, False

p1, p2 = score(20), score(21)
print(p1)
print(p2)
```

Funkcie

```
# definícia funkcie s vrátenou sekvenciou hodnôt
# kľúčové slovo return vráti chybu
def score():
    for point in range(0, 10, 1):
        if point % 2 == 0:
            yield point

for point in score():
    print(point)


# definícia funkcie s prázdnym telom
def show_club():
    pass
```

Funkcie

- Známe Python funkcie:
  - print() – výpis na konzolu
  - int(), str() – transformácia dátového typu na int, str
  - range() – generovanie dátovej štruktúry obsahujúca prvky, nech 1, 2, 3, ..., 10
  - pow() – mocninová funkcia

Rekurzia

- Funkcia volajúca samu seba sa v tele funkcie nazýva rekurzívnou funkciou, alebo **rekurzia**
- Rekurzívne riešenia môžu byť veľmi efektívne a matematicky elegantné
- Vývojár by mal byť pri rekurzii opatrný, pretože môže byť celkom ľahké skĺznuť k napísaniu funkcie, ktorá sa nikdy nekončí, alebo funkcie, ktorá využíva nadmerné množstvo pamäte alebo kapacitu procesora

```python
# vráti súčet prvých celých čísel
def sum_recursion(number):

    if number == 0:
        return 0

    return number + sum_recursion(number - 1)

print(sum_recursion(15))

# vráti faktoriál
def factorial(number):

    if number == 0:
        return 1

    return number * factorial(number - 1)

print(factorial(15))
```