For-cyklus

Je výpočet (cyklus) s daným počtem opakování a má takovýto tvar:

for *proměnná* ***in*** *range(počet):*
    blok příkazů

První řádek obsahuje úvodní slovo ***for***, za kterým je zástupná (***i***), či námi uvedená proměnná, a opakuje blok příkazů uvedený pod ním v zadaném počtu uvedeného za slovem ***in***.

První řádek končí dvojtečkou a řádky příkazů pod ním jsou odskočeny o 4 mezery.

Blok příkazů může obsahovat nejen jeden příkaz, ale i více příkazů, které se vždy provedou v zadaném pořadí.

Po ukončení opakování bloku příkazů program přejde na další řádek, kde je další příkaz uveden již bez počátečních 4 mezer.

Python po každém provedení cyklu zvýší hodnotu proměnné.

Cyklus se jmenovanými hodnotami

Namísto funkce ***range(n)***, která nám generuje počet cyklů uvedených v závorce, posloupností od 0 do n-1, můžeme zde uvést své vlastní hodnoty oddělené čárkou. S tím, že ale musí být minimálně dvě.

Vyjmenované hodnoty mohou být v libovolném pořadí a mohou se i opakovat:

Takový typ cyklu můžeme použít jen tehdy, když máme k dispozici přesný seznam hodnot a ne libovolný počet, který zvládla funkce *range().*

Přičítací šablona (accumulator pattern)

Je to programátorská pomůcka (schéma, vzor), která se často opakuje v některých typech programů. V tomto případě označuje, že ještě před cyklem inicializujeme nějakou přičítací proměnnou a v těle cyklu hodnotu této proměnné zvyšujeme podle potřeby (například přičteme 1, nebo přičteme proměnnou cyklu, nebo její mocninu, nebo vynásobíme něčím, nebo vydělíme, …). Po skončeném cyklu máme v této pomocné přičítací proměnné očekávaný výsledek. Například:

V následujícím příkladu jsou vyjmenovány hodnoty nejrůznějších typů, dokonce jednou z hodnot je i funkce abs. Cyklus vypíše hodnotu proměnné cyklu a poté i její typ:

for hodnota in 3.14, abs(7-123), 'text', 100/4, abs, '42':

Cyklus s prvky znakového řetězce

Už jsme viděli, že znakové řetězce se mohou nacházet mezi jmenovanými hodnotami for-cyklu. Ale znakový řetězec v Pythonu je ve skutečnosti **posloupnost znaků**. Díky tomu for-cyklus může procházet také prvky této posloupnosti. Proměnná cyklu pak postupně nabývá hodnot jednotlivých znaků, což jsou vlastně jednoznakové řetězce. Tedy

Parametr *end=‘‘* funkce *print()*

Pokud nenastavíme jinak, standardně je funkce *print()* nastavena tak, že po každém cyklu sama dodává na konec znak pro nový řádek *(\n*). V případě, že chceme, aby se vše vypsalo v jednom řádku, použijeme k tomu parametr funkce *print()*, který se zapisuje do závorky jako poslední za všemi vypisovanými hodnotami a značí se ***end='řetězec',*** kdy nejčastěji namísto řetězce bude jedna prázdná mezera ' ', nebo prázdný řetězec ''

Například:

```
for i in range(100, 200):
   print(i, end=' ')
>>>
100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119
120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139
140 141 142 143 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159
160 161 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179
180 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199
```

Takový zápis využijeme hlavně při výpisu většího počtu hodnot, ale i tehdy, když jeden řádek výpisu potřebujeme poskládat z více částí v různých částech programu, například:

```
print('programujem', end='_')
print(10, end='...')
print('rokov')
>>>
programujem_10...rokov
```

Funkce range() i pro jiné posloupnosti celých čísel

Funkce range(n), která nahrazuje jmenování celočíselných hodnot (od 0 do n-1) nám umožňuje i zadat počáteční a koncovou hodnotu generované posloupnosti

Například range(5, 15)označuje rostoucí posloupnost celých čísel, která začíná hodnotou 5 a všechny další prvky jsou menší než 15, tedy vygenerovaná posloupnost by byla: 5, 6, 7, 8, 9, 10, 11, 12, 13, 14.

Funkci range()můžeme zadat i třetí parametr, určující krok:

rozsah (*start* , *stop* , *krok*)
Parametry

- **start** – první prvek vygenerované posloupnosti (pokud chybí, předpokládá se 0)
- **stop** – hodnota, na které se již generování další hodnoty posloupnosti zastaví - tato hodnota již v posloupnosti nebude
- **krok** – hodnota, o kterou se zvýší každý následující prvek posloupnosti, pokud tento parametr chybí, předpokládá se 1

Speciálním případem parametrů funkce *range()* je záporný krok, kdy požadujeme klesající posloupnost čísel. Například zápis *range(15, 5, -1)* označuje, že prvním členem posloupnosti bude 15, všechny další budou o 1 menší a poslední z nich nebude **menší nebo roven** než 5. Toto samé se dá ale i zapsat pomocí funkce *reversed(),* například:

Moduly math a random

Známe již některé standardní funkce, které jsou definovány hned při startu Pythonu:

- funkce ***type()*** vrátí typ zadané hodnoty
- funkce ***int()***, ***float()***,***str()*** přetypují (překonvertují) zadanou hodnotu na jiný typ
- funkce ***print()***, ***input()*** jsou určeny pro výpis textů a přečtení textu ze vstupu
- funkce ***abs()***, ***round()*** počítají absolutní hodnoty čísel a zaokrouhlují desetinná čísla
- funkce ***range()***, ***reversed()*** generují posloupnost čísel, resp. ji otáčejí
- funkce ***help()*** vypíše textové informace o pythonových funkcích a objektech

Standardních funkcí je mnohem více a z mnoha z nich se seznámíme později. Nyní si ukážeme dva nové moduly (představme si je jako nějaké knihovny užitečných funkcí), které, přestože nejsou standardně zabudovány, my je budeme potřebovat velmi často. Pokud potřebujeme pracovat s nějakým modulem, musíme to nejprve Pythonu oznámit speciálním způsobem. Slouží k tomu příkaz ***import***.

Modul math

Pomocí takového zápisu:

import math

umožníme našim programům pracovat s matematickými funkcemi z této knihovny. Ve skutečnosti tímto příkazem Python vytvoří novou proměnnou math(nové jméno v paměti jmen proměnných).

Knihovna v tomto modulu obsahuje, například tyto matematické funkce: ***sin(), cos(), sqrt().*** Jenže s takovými funkcemi nemůžeme pracovat přímo: Python nezná jejich jména, zná jediné jméno a to jméno modulu math. Jelikož tyto funkce se nacházejí právě v tomto modulu, budeme k nim přistupovat. puntíkovou notací (dot notation): za jméno modulu uvedeme prvek (v tomto případě funkci) z daného modulu.

Například ***math.sin()*** označuje volání funkce sinus a ***math.sqrt()*** označuje výpočet druhé odmocniny čísla.

Většina prvků modulu ***math*** nás zatím nebude zajímat, ale pokud chceme znát detaily nějakého prvku modulu, můžeme použít standardní funkci ***help().***

Formátování výpisů do tabulek

Pokud výpis tabulky není moc hezký (např. obsahuje čísla vypsaná zbytečně na spoustu desetinných míst), využijeme toho, že ve formátovacím řetězci můžeme určit, na jakou šířku se má dané desetinné číslo vypisovat. V našem případě to bude šířka 6 znaků, přičemž z toho budou 3 desetinná místa. Hodnota v {}závorkách může za znakem ':'obsahovat takovou šířku výpisu. Všimněte si poslední řádek s voláním **print()**:

```
import math

for uhol in range(0, 91, 5):
    uhol_v_radianoch = math.radians(uhol)
    sin_uhla = math.sin(uhol_v_radianoch)
    cos_uhla = math.cos(uhol_v_radianoch)
    print(f'{uhol:3} {sin_uhla:6.3f} {cos_uhla:6.3f}')
```

Pokusme se nakreslit (v textové ploše pomocí nějakých znaků) průběh funkce sinus. Jelikož oborem hodnot této funkce je interval reálných čísel <-1, 1>a my chceme tyto hodnoty natáhnout na šířku výpisu do 80 znaků, zapíšeme:

```
import math

for uhol in range(0, 361, 10):
    uhol_v_radianoch = math.radians(uhol)
    sin_uhla = math.sin(uhol_v_radianoch)
    stlpec = int(sin_uhla * 35 + 40)
    print(' ' * stlpec + 'SIN')
```

```
>>>                 SIN
                       SIN
                          SIN
                             SIN
                                SIN
                                   SIN
                                      SIN
                                         SIN
                                          SIN
                                           SIN
                                            SIN
                                           SIN
                                          SIN
                                         SIN
                                      SIN
                                   SIN
                                SIN
                             SIN
                          SIN
                       SIN
                    SIN
                 SIN
              SIN
           SIN
        SIN
     SIN
  SIN
 SIN
SIN
SIN
SIN
 SIN
  SIN
    SIN
       SIN
          SIN
             SIN
                SIN
                   SIN
```

Všimněte si, že všechny tři proměnné uhol_v_radianoch, *sin_uhla* a *stlpec* v těle cyklu zde slouží „jen“ ke zvýšení čitelnosti kódu a mohli bychom to zapsat i bez nich:

```
import math

for uhol in range(0, 361, 10):
    print(' ' * int(math.sin(math.radians(uhol)) * 35 + 40) + 'SIN')
```

Importování funkcí z knihoven můžeme zapsat i trochu jinak. V této druhé variantě budou k dispozici jen ty funkce, které jmenujeme v tomto příkazu. Obecný tvar je:

from knižnice import funkcia1, funkcia2, ...

Tímto příkazem se naimportují jen vyjmenované funkce a dále se budeme na ně v kódu odvolávat přímo jejich jménem a nebudeme psát jméno knihovny a tečku. Například, předchozí příklad můžeme zapsat:

Je na programátorovi, který zápis použije. Někteří upřednostňují předchozí verzi s psaním jména knihovny i s tečkou, neboť při čtení programu je zřejmější, odkud daná funkce přišla.

Modul random

I tento modul obsahuje knihovnu funkcí, ale tyto umožňují generování náhodných čísel. My z této knihovny využijeme zejména tyto dvě funkce:

- ***randint()*** vybere náhodné číslo z intervalu celých čísel
- ***choice()*** vybere náhodný prvek z nějaké posloupnosti, například ze znakového řetězce (postupnosti znaků)

Abychom mohli pracovat s těmito funkcemi, nesmíme zapomenout zapsat:

import random

Následující ukázka ilustruje volání funkce randint(). Parametry této funkce udávají dolní a horní hranici intervalu, ze kterého se vybere jedna náhodná hodnota. Každé volání random.randint(1, 6) **náhodně** vybere jednu z hodnot intervalu <1, 6>, tedy z množiny čísel 1, 2, 3, 4, 5, 6. Můžeme si to představit jako hod hrací kostkou, na které jsou čísla od 1 do 6. Následovný program vypíše posloupnost 100 náhodných hodů kostky:

```
import random

for i in range(100):
    nahodne = random.randint(1, 6)
    print(nahodne, end=' ')
```

a po spuštění dostaneme například takové výsledky:

```
4 1 3 5 1 1 6 6 1 6 5 2 2 4 4 6 1 2 5 1 5 5 5 4 3 2 5 3 2 6 1 2 2 2 4 3 5 3 4 1
3 4 4 4 5 4 3 6 6 1 3 3 4 3 5 5 4 6 3 2 2 4 3 2 6 1 5 5 3 6 5 6 6 5 4 5 5 6 3 6
6 5 6 3 2 1 5 4 5 2 4 1 2 5 1 1 2 2 5 4
```

Velmi podobně funguje i druhá funkce **choice()**. Tato má jen jeden parametr, kterým je nějaká posloupnost hodnot. Zatím jsme se setkali se dvěma posloupnostmi hodnot: funkcí range(...)a se znakovými řetězci. Zapíšeme-li:

```
random.choice(range(1, 10, 2))
```

Vygeneruje se náhodné číslo z posloupnosti lichých čísel: 1, 3, 5, 7, 9. V knihovně random existuje také funkce ***randrange***, která dělá přesně to samé a zapsali bychom to *random.randrange(1, 10, 2).*

Generátor náhodných slov

Pro generátor náhodných slov je nejzajímavější posloupností posloupnost znaků, tedy libovolný znakový řetězec. Například volání:

```
random.choice('aeiouy')
```

vybere **náhodnou** hodnotu z posloupnosti šesti znaků - posloupnosti samohlásek. Podobně bychom mohli zapsat:

```
random.choice('bcdfghjklmnpqrstvwxz')
```

i toto volání vybere **náhodné** písmeno z posloupnosti souhlásek. Když to teď dáme dohromady, dostaneme generátor náhodně vygenerovaných slov:

```
import random

slovo = ''
for i in range(3):
   spoluhlaska = random.choice('bcdfghjklmnpqrstvwxz')
   samohlaska = random.choice('aeiouy')
   slovo = slovo + spoluhlaska + samohlaska
print(slovo)
```

Program vygeneruje 3 **náhodné** dvojice souhlásek a samohlásek, tedy dvoupísmenných slabik. Vždy, když budeme potřebovat další náhodné slovo, musíme spustit tento program (například pomocí F5).

Vnořené cykly

Pokud bychom potřebovali vygenerovat najednou 10 slov, použijeme znovu for-cyklus. Proto celý náš program (kromě úvodního import) obalíme konstrukcí for, tj. všechny řádky současného programu posuneme o 4 znaky vpravo:

```
import random

for j in range(10):
   slovo = ''
   for i in range(3):
      spoluhlaska = random.choice('bcdfghjklmnpqrstvwxz')
      samohlaska = random.choice('aeiouy')
      slovo = slovo + spoluhlaska + samohlaska
   print(slovo)
```

Všimněte si, že v těle vnějšího for-cyklu (s proměnnou cyklu j) se nacházejí tři příkazy: přiřazení, poté tzv. **vnořený** for-cyklus a na konec volání funkce print().