2. Opakované výpočty

video prezentácie

1. opakovanie výpočtu
2. moduly math a random
3. vnorené cykly

Doteraz sme sa naučili pracovať s tromi základnými príkazmi:

- priraďovací príkaz vytvorí alebo zmení obsah nejakej premennej
- výpis nejakých hodnôt do textovej plochy pomocou `print()`
- prečítanie hodnoty zadanej klávesnicou pomocou `input()`

Z týchto troch typov príkazov sme skladali programy (skripty), ktorých príkazy sa potom vykonávali postupne jeden za druhým. Lenže pri programovaní reálnych programov budeme potrebovať, aby sa nejaké časti programov mohli vykonávať viackrát za sebou bez toho, aby sme to museli viackrát rozpísať.

Napríklad namiesto:

```
print('programujem v Pythone')
print('programujem v Pythone')
print('programujem v Pythone')
print('programujem v Pythone')
print('programujem v Pythone')
```

by sme potrebovali zapísať:

```
opakuj nasledovný príkaz 5-krát:
    print('programujem v Pythone')
```

Na toto v Pythone slúži konštrukcia **for-cyklus**

For-cyklus

Postupne ukážeme niekoľko základných typov použitia for-cyklu.

Cyklus s daným počtom opakovaní

Táto programová konštrukcia má takýto tvar:

```
for premenná in range(pocet):
    blok príkazov
```

Opakuje zadaný **počet** krát príkazy odsunutého bloku príkazov (tzv. **indentation**). Samotný riadok konštrukcie **for** obsahuje meno nejakej premennej a je ukončený znakom dvojbodka. Za tým nasleduje **blok príkazov** - sú to príkazové riadky, napríklad `print()`, ktoré sú odsunuté o 4 medzery.

Zapíšme program, ktorý 5-krát vypíše zadaný text:

```
for prem in range(5):
    print('programujem v Pythone')
```

Blok príkazov môže obsahovať nielen jeden príkaz, ale aj viac príkazov, napríklad

```
for prem in range(5):
    print('študujem na matfyze a')
    print('programujem v Pythone')
print('============')
```

Blok príkazov, ktorý sa má opakovať v danom cykle končí napríklad vtedy, keď sa objaví riadok s príkazmi na úrovni riadku s for-cyklom. Teda posledný riadok so znakmi `'============'` sa vypíše až po skončení cyklu, teda iba raz:

```
študujem na matfyze a
programujem v Pythone
študujem na matfyze a
programujem v Pythone
študujem na matfyze a
programujem v Pythone
študujem na matfyze a
programujem v Pythone
študujem na matfyze a
programujem v Pythone
============
```

Zatiaľ nevieme, na čo slúži premenná cyklu (v našom príklade `prem`). Python tejto premennej automaticky nastavuje hodnotu podľa toho, koľký krát sa už cyklus vykonal. Teda zápis:

```
for prem in range(n):
    prikazy
```

v skutočnosti znamená:

```
prem = 0
prikazy
prem = 1
prikazy
prem = 2
prikazy
...
prem = n-1
prikazy
```

Napríklad:

```
for i in range(4):
```

```
    print(i, 'riadok')
```

označuje

```
i = 0
print(i, 'riadok')
i = 1
print(i, 'riadok')
i = 2
print(i, 'riadok')
i = 3
print(i, 'riadok')
```

Teda program vypíše:

```
0 riadok
1 riadok
2 riadok
3 riadok
```

For-cyklus začne byť zaujímavý až vtedy, keď sa v tele cyklu budú robiť nejaké výpočty. Začnime jednoduchým pripočítavaním 1:

```
n = int(input('zadaj n: '))
pocet = 0
for i in range(n):
    pocet = pocet + 1
print('počet prechodov cyklu =', pocet)
```

Premennú `pocet` sme ešte pred začiatkom cyklu vynulovali. Výstup môže byť napríklad takýto:

```
zadaj n: 17
počet prechodov cyklu = 17
```

Ak budeme namiesto 1 pripočítavať hodnotu premennej cyklu:

```
n = int(input('zadaj n: '))
sucet = 0
for i in range(n):
    sucet = sucet + i
print('súčet =', sucet)
```

dostávame súčet čísel od `0` do `n-1`, teda `0 + 1 + 2 + 3 + ... + n-2 + n-1`, napríklad:

```
zadaj n: 6
súčet = 15
```

Vidíme, že `15` súčtom hodnôt `0 + 1 + 2 + 3 + 4 + 5`.

Malou zmenou spočítame druhé mocniny tejto postupnosti:

```
n = int(input('zadaj n: '))
```

```
sucet = 0
for i in range(n):
    sucet = sucet + i*i       # môžeme zapísať aj  sucet += i**2
print('súčet =', sucet)
```

Uvedomte si, že týmto algoritmom úplne zbytočne pripočítavame k premennej `sucet` aj `0` na začiatku cyklu.

Cyklus s vymenovanými hodnotami

Ukážme tento ďalší typ for-cyklu na príklade:

```
for i in 1, 2, 3, 4, 5:
    blok prikazov
```

Namiesto funkcie `range(n)`, ktorá pre nás vygenerovala postupnosť čísel od `0` do `n-1`, sme vymenovali postupnosť hodnôt v presnom poradí. Práve pre tieto hodnoty sa postupne v cykle vykoná `blok prikazov`. Vymenované hodnoty musia byť oddelené čiarkou a mali by byť aspoň dve.

Otestujme:

```
sucin = 1
for cislo in 1, 2, 3, 4, 5, 6:
    sucin = sucin * cislo
print('6 faktoriál =', sucin)
```

a naozaj dostávame:

```
6 faktoriál = 720
```

Ďalší program počíta druhé mocniny niektorých zadaných čísel:

```
for x in 5, 7, 11, 13, 23:
    x2 = x**2
    print('druhá mocnina', x, 'je', x2)
```

Po spustení dostávame:

```
druhá mocnina 5 je 25
druhá mocnina 7 je 49
druhá mocnina 11 je 121
druhá mocnina 13 je 169
druhá mocnina 23 je 529
```

Vymenované hodnoty sa môžu aj ľubovoľne opakovať, napríklad:

```
i = 1
for prem in 2, 1, 7, 2, 3:
    print(i, 'prechod cyklu s hodnotou', prem)
```

```
    i = i + 1
```

vypíše:

```
1 prechod cyklu s hodnotou 2
2 prechod cyklu s hodnotou 1
3 prechod cyklu s hodnotou 7
4 prechod cyklu s hodnotou 2
5 prechod cyklu s hodnotou 3
```

Ďalší príklad ukazuje výpočet počtu dní v roku ako súčet počtov dní v jednotlivých mesiacoch:

```
pocet = 0
for mesiac in 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31:
    pocet += mesiac
print('počet dní v bežnom roku =', pocet)
```

Zrejme, takýto typ cyklu môžeme použiť len vtedy, keď máme k dispozícii presný zoznam hodnôt a nie ľubovoľný počet, ktorý predtým zvládla funkcia `range()`.

Okrem toho, že sa hodnoty môžu opakovať, nemáme obmedzenia ani na typy vymenovaných hodnôt. Ukážme to na príklade, v ktorom spočítame ceny jednotlivých položiek nákupu a okrem tejto sumy vypočítame aj priemernú hodnotu. Vymenované hodnoty sú teraz desatinné čísla (typ `float`):

```
pocet = 0
suma = 0
for cena in 1.75, 2.20, 1.03, 4.00, 3.50, 2.90, 1.89:
    suma = suma + cena
    pocet = pocet + 1
print('nakúpil si', pocet, 'položiek')
print('za', suma, 'euro')
print('priemerná cena bola', round(suma / pocet, 2), 'euro')
```

a výsledkom je:

```
nakúpil si 7 položiek
za 17.27 euro
priemerná cena bola 2.47 euro
```

Všimnite si, že opäť sme použili rovnakú schému na sčitovanie pomocou cyklu, ako sme to robili vyššie. Niekedy sa tomuto hovorí pripočítavacia šablóna.

Pripočítavacia šablóna (accumulator pattern)

Je to programátorská pomôcka (schéma, vzor), ktorá sa často opakuje v niektorých typoch programov. V tomto prípade označuje, že ešte pred cyklom inicializujeme nejakú pripočítavaciu premennú (v našom príklade dokonca dve premenné `pocet` a `suma`) a v tele cyklu hodnotu tejto premennej zvyšujeme podľa potreby (napríklad pripočítame 1, alebo pripočítame premennú cyklu, alebo jej mocninu, alebo vynásobíme niečím, alebo vydelíme, …). Po skončené cyklu máme v tejto pomocnej pripočítavacej premennej očakávaný výsledok. Napríklad:

```
<inicializuj pripočítavaciu premennú>
for premenná_cyklu in postupnosť_hodnôt:
    <použi pripočítavaciu premennú>
<v pripočítavacej premennej sa nachádza nejaký súčet>
```

Aj ďalšie dva príklady ilustrujú to, že vymenované hodnoty pre for-cyklus môžu mať rôzne typy:

```
for slovo in 'Python', 'Bratislavu', 'Matfyz':
    print('mám rád', slovo)
```

Hodnotami sú znakové reťazce a výsledkom bude:

```
mám rád Python
mám rád Bratislavu
mám rád Matfyz
```

V nasledovnom príklade sú vymenované hodnoty najrôznejších typov, dokonca jednou z hodnôt je aj funkcia `abs`. Cyklus vypíše hodnotu premennej cyklu a potom aj jej typ:

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

a výpis:

```
3.14 <class 'float'>
116 <class 'int'>
text <class 'str'>
25.0 <class 'float'>
<built-in function abs> <class 'builtin_function_or_method'>
42 <class 'str'>
```

Vymenované hodnoty vo for-cykle by mali byť aspoň dve. Ak by sme otestovali:

```
for i in 123:
    print(i)
```

dostaneme chybovú správu `TypeError: 'int' object is not iterable`. Táto správa oznamuje, že celé číslo sa nedá prechádzať pomocou for-cyklu (nie je iterovateľné), alebo inými slovami: celé číslo sa nedá rozobrať na zložky tak, aby sme ich potom prechádzali for-cyklom. Podobnú správu dostaneme aj vtedy, keď sa pokúsime for-cyklom prechádzať jedno desatinné číslo (`TypeError: 'float' object is not iterable`). Iná situácia je ale so znakovými reťazcami.

Cyklus s prvkami znakového reťazca

Už sme videli, že znakové reťazce sa môžu nachádzať medzi vymenovanými hodnotami for-cyklu. Ale znakový reťazec v Pythone je v skutočnosti **postupnosť znakov**. Vďaka tomu for-

cyklus môže prechádzať aj prvky tejto postupnosti. Premenná cyklu potom postupne nadobúda hodnoty jednotlivých znakov, čo sú vlastne jednoznakové reťazce. Teda

```
for znak in 'python':
    print(znak)
```

je pre Python to isté ako:

```
for znak in 'p', 'y', 't', 'h', 'o', 'n':
    print(znak)
```

a zrejme sa vypíše:

```
p
y
t
h
o
n
```

Ďalší príklad ilustruje použitie **pripočítavacej šablóny** aj pre znakové reťazce. Najprv ešte pred cyklom inicializujeme dve reťaczové premenné `retazec1` a `retazec2`. Do nich budeme v tele cyklu postupne po jednom „pripočítavať“ (priraďovať) znaky zo zadaného reťazca:

```
vstup = input('zadaj: ')
pocet = 0
retazec1 = retazec2 = ''
for znak in vstup:
    retazec1 = retazec1 + znak
    retazec2 = znak + retazec2
    pocet = pocet + 1
print('počet znakov reťazca =', pocet)
print('retazec1 =', retazec1)
print('retazec2 =', retazec2)
```

Dostávame:

```
zadaj: Python
počet znakov reťazca = 6
retazec1 = Python
retazec2 = nohtyP
```

Všimnite si, že `retazec2` obsahuje prevrátené poradie znakov pôvodného reťazca. Otestujte, či takýto for-cyklus bude fungovať nielen s jednoznakovým reťazcom, ale aj s prázdnym.

Funkcia range() aj pre iné postupnosti celých čísel

Videli sme, že funkcia `range(n)` nahrádza vymenovanie celočíselných hodnôt od `0` do `n-1`. Táto funkcia je v skutočnosti trochu univerzálnejšia: dovolí nám zadať nielen koncovú hodnotu vygenerovanej postupnosti ale aj počiatočnú. V tomto prípade funkciu zavoláme s dvomi parametrami:

- prvý parameter potom označuje počiatočnú hodnotu postupnosti
- druhý parameter označuje hodnotu, pri ktorej generovanie postupnosti končí, t.j. postupnosť bude obsahovať len hodnoty menšie ako tento druhý parameter

Napríklad `range(5, 15)` označuje rastúcu postupnosť celých čísel, ktorá začína hodnotou `5` a všetky ďalšie prvky sú menšie ako `15`, teda vygenerovaná postupnosť by bola: `5, 6, 7, 8, 9, 10, 11, 12, 13, 14`. Ak teda potrebujeme postupnosť čísel od `1` do zadaného `n`, musíme zapísať:

```
n = int(input('zadaj n: '))
for cislo in range(1, n+1):
    print('hodnota v cykle', cislo)
print('koniec cyklu')
```

a výstupom je napríklad:

```
zadaj n: 7
hodnota v cykle 1
hodnota v cykle 2
hodnota v cykle 3
hodnota v cykle 4
hodnota v cykle 5
hodnota v cykle 6
hodnota v cykle 7
koniec cyklu
```

Teraz môžeme pomocou **pripočítavacej šablóny** vypočítať aj faktoriál pre ľubovoľnú zadanú hodnotu:

```
n = int(input('zadaj číslo: '))
faktorial = 1
for cislo in range(2, n+1):
    faktorial = faktorial * cislo
print(n, 'faktoriál =', faktorial)
```

Spustíme s rôznymi hodnotami:

```
zadaj číslo: 1
1 faktoriál = 1

zadaj cislo: 6
6 faktorial = 720

zadaj cislo: 20
20 faktorial = 2432902008176640000
```

Ak by sme spustili nasledovný program:

```
for i in range(100, 200):
    print(i)
```

dostali by sme 100-riadkový výpis s číslami od `100` do `199`. Teraz by sa nám ale hodilo, ak by `print()` v niektorých situáciách nekončil prechodom na nový riadok, ale ďalší `print()` by tu pokračoval. Využijeme na to nový typ parametra funkcie `print()`:

funkcia `print()`

`print(..., end='reťazec')`

Parametre

end='reťazec' – tento reťazec nahradí štandardný `'\n'` ľubovoľným iným, najčastejšie to bude jedna medzera `' '` alebo prázdny reťazec `''`

Tento parameter musí byť v zozname parametrov funkcie `print()` uvedený ako posledný za všetkými vypisovanými hodnotami. Vďaka nemu po vypísaní týchto hodnôt sa namiesto prechodu na nový riadok vypíše zadaný reťazec.

Napríklad:

```
for i in range(100, 200):
    print(i, end=' ')
```

teraz vypíše:

```
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
```

Ale s prázdnym reťazcom pre parameter `end`:

```
for i in range(100, 200):
    print(i, end='')
```

vypíše:

```
100101102103104105106107108109110111112113114115116117118119120121122123124125126127128129130131132133134135136137138139140141142143144145146147148149150151152153154155156157158159160161162163164165166167168169170171172173174175176177178179180181182183184185186187188189190191192193194195196197198199
```

Takýto zápis využijeme hlavne pri výpise väčšieho počtu hodnôt, ale aj vtedy, keď jeden riadok výpisu potrebujeme poskladať z viacerých častí v rôznych častiach programu, napríklad:

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

vypíše:

```
programujem_10...rokov
```

Funkcii `range()` môžeme zadať aj tretí parameter, pričom všetky tieto parametre musia byť celočíselné hodnoty. Zhrňme všetky tri varianty tejto funkcie:

funkcia `range()`

`range(stop)`

`range(start, stop)`
`range(start, stop, krok)`

Parametre

- **start** – prvý prvok vygenerovanej postupnosti (ak chýba, predpokladá sa `0`)
- **stop** – hodnota, na ktorej sa už generovanie ďalšej hodnoty postupnosti zastaví - táto hodnota už v postupnosti nebude
- **krok** – hodnota, o ktorú sa zvýši každý nasledovný prvok postupnosti, ak tento parameter chýba, predpokladá sa `1`

Najlepšie to ukážeme na príkladoch rôzne vygenerovaných postupností celých čísel. V tabuľke vidíme výsledky pre rôzne parametre:

`range(10)`	0, 1, 2, 3, 4, 5, 6, 7, 8, 9
`range(0, 10)`	0, 1, 2, 3, 4, 5, 6, 7, 8, 9
`range(0, 10, 1)`	0, 1, 2, 3, 4, 5, 6, 7, 8, 9
`range(3, 10)`	3, 4, 5, 6, 7, 8, 9
`range(3, 10, 2)`	3, 5, 7, 9
`range(10, 100, 10)`	10, 20, 30, 40, 50, 60, 70, 90
`range(10, 1)`	*prázdna postupnosť*
`range(1, 1)`	*prázdna postupnosť*
`range(0)`	*prázdna postupnosť*

Nasledovný príklad ilustruje použitie parametra krok vo funkcii `range()`. Potrebujeme spočítať súčet všetkých nepárnych čísel do `1000` (teda menších ako `1000`). Zrejme začneme s `1` a každé ďalšie pripočítavané číslo bude o `2` väčšie. Teda

```
sucet = 0
for cislo in range(1, 1000, 2):
    sucet = sucet + cislo
print('súčet nepárnych čísel je', sucet)
```

Špeciálnym prípadom parametrov funkcie `range()` je záporný krok, teda situácia, keď požadujeme klesajúcu postupnosť čísel. Napríklad zápis `range(15, 5, -1)` označuje, že prvým členom postupnosti bude `15`, všetky ďalšie budú o `1` menšie (parameter `krok` je `-1`) a posledný z nich nebude **menší alebo rovný** ako `5` (parameter `stop` je `5`). Otestujeme:

```
for i in range(15, 5, -1):
    print(i, end=' ')
```

a dostávame postupnosť:

```
15 14 13 12 11 10 9 8 7 6
```

čo je vlastne prevrátené poradie postupnosti `range(6, 16)`. Ak sa vám záporný krok pri volaní `range()` nie veľmi páči, Python to umožňuje zapísať aj elegantnejšie pomocou funkcie `reversed()` a funkcie `range()` takto:

```
for i in reversed(range(6, 16)):
    print(i, end=' ')
```

čím dostávame rovnakú postupnosť ako v predchádzajúcom príklade. Ešte skontrolujme:

```
for i in reversed(range(10)):
    print(i, end=' ')
```

s výsledkom:

```
9 8 7 6 5 4 3 2 1 0
```

čo je veľakrát čitateľnejšie ako použitie `range(9, -1, -1)`.

Moduly math a random

Poznáme už niektoré štandardné funkcie, ktoré sú zadefinované hneď pri štarte Pythonu:

- funkcia `type()` vráti typ zadanej hodnoty
- funkcie `int()`, `float()` a `str()` pretypujú (prekonvertujú) zadanú hodnotu na iný typ
- funkcie `print()` a `input()` sú určené na výpis textov a prečítanie textu zo vstupu
- funkcie `abs()` a `round()` počítajú absolútne hodnoty čísel a zaokrúhľujú desatinné čísla
- funkcie `range()` a `reversed()` generujú postupnosť čísel, resp. ju otáčajú
- funkcia `help()` vypíše textové informácie o pythonových funkciách a objektoch

Štandardných funkcií je oveľa viac a z mnohými z nich sa zoznámime neskôr. Teraz si ukážeme dva nové moduly (predstavme si ich ako nejaké knižnice užitočných funkcií), ktoré, hoci nie sú štandardne zabudované, my ich budeme potrebovať veľmi často. Ak potrebujeme pracovať s nejakým modulom, musíme to najprv Pythonu oznámiť špeciálnym spôsobom. Slúži na to príkaz `import`.

Modul math

Pomocou takéhoto zápisu:

```
import math
```

umožníme našim programom pracovať s matematickými funkciami z tejto knižnice. V skutočnosti týmto príkazom Python vytvorí novú premennú `math` (nové meno v pamäti mien premenných).

Knižnica v tomto module obsahuje, napríklad tieto matematické funkcie: `sin()`, `cos()`, `sqrt()`. Lenže s takýmito funkciami nemôžeme pracovať priamo: Python nepozná ich mená, pozná jediné meno a to meno modulu `math`. Keďže tieto funkcie sa nachádzajú práve v tomto module, budeme k nim pristupovať, tzv. bodkovou notáciou (**dot notation**): za meno modulu uvedieme prvok (v tomto prípade funkciu) z daného modulu. Napríklad `math.sin()` označuje

volanie funkcie **sínus** a `math.sqrt()` označuje výpočet druhej odmocniny čísla. Otestujme v interaktívnom režime:

```
>>> import math
>>> math
<module 'math' (built-in)>
>>> type(math)
<class 'module'>
>>> math.sin
<built-in function sin>
>>> math.sin()
Traceback (most recent call last):
  File "<pyshell#14>", line 1, in <module>
    math.sin()
TypeError: sin() takes exactly one argument (0 given)
```

Posledná chybová správa oznamuje, že funkciu `sin` musíme volať práve s jedným parametrom (volanie bez parametrov sa Pythonu nepáči).

Ak zadáme `dir(math)` (z anglického *directory*, teda adresár), Python nám vypíše všetky prvky, ktoré sa nachádzajú v tomto module:

```
>>> dir(math)
['__doc__', '__loader__', '__name__', '__package__', '__spec__', 'acos', 'acosh',
'asin', 'asinh', 'atan', 'atan2', 'atanh', 'ceil', 'comb', 'copysign', 'cos',
'cosh', 'degrees', 'dist', 'e', 'erf', 'erfc', 'exp', 'expm1', 'fabs', 'factorial'
,
'floor', 'fmod', 'frexp', 'fsum', 'gamma', 'gcd', 'hypot', 'inf', 'isclose',
'isfinite', 'isinf', 'isnan', 'isqrt', 'ldexp', 'lgamma', 'log', 'log10', 'log1p',
'log2', 'modf', 'nan', 'perm', 'pi', 'pow', 'prod', 'radians', 'remainder', 'sin',
'sinh', 'sqrt', 'tan', 'tanh', 'tau', 'trunc']
```

Väčšina prvkov modulu `math` nás zatiaľ nebude zaujímať, ale môžeme tam vidieť, napríklad aj funkcie `exp()`, `log()`, `tan()`, `radians()` ale aj známe konštanty `e` a `pi`. Ak chceme poznať detaily nejakého prvku modulu, môžeme použiť štandardnú funkciu `help()`, napríklad:

```
>>> help(math.log)
Help on built-in function log in module math:

log(...)
    log(x[, base])

    Return the logarithm of x to the given base.
    If the base not specified, returns the natural logarithm (base e) of x.
```

Môžeme sa dozvedieť, že funkcia `math.log()` počíta logaritmus čísla buď so základom `e` (prirodzené logaritmy) alebo s nejakým daným základom `base`.

Podobne môžeme zadať:

```
>>> help(math.sin)
Help on built-in function sin in module math:

sin(...)
    sin(x)
```

```
    Return the sine of x (measured in radians).
```

Toto označuje, že funkcia `sin()` z modulu `math` naozaj počíta sínus, ale uhol musíme zadať v radiánoch. Preto, napríklad pre výpočet `sin(45)` musíme zapísať jednu z možností:

```
>>> math.sin(45 * 3.14159 / 180)
0.7071063120935576
>>> math.sin(45 * math.pi / 180)
0.7071067811865475
>>> math.sin(math.radians(45))
0.7071067811865475
```

Druhý a tretí výpočet využívajú buď konštantu `pi` alebo konverznú funkciu `radians()`, ktorá prevádza stupne na radiány. Zrejme najčastejšie budeme používať tretí variant pomocou `radians()`.

Vytvorme tabuľku hodnôt funkcií sínus aj kosínus pre uhly od `0` do `90` stupňov krokom `5`:

```
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(uhol, sin_uhla, cos_uhla)
```

Výpis nie je veľmi pekný - obsahuje čísla vypísané zbytočne na veľa desatinných miest:

```
0 0.0 1.0
5 0.08715574274765817 0.9961946980917455
10 0.17364817766693033 0.984807753012208
15 0.25881904510252074 0.9659258262890683
...
```

Urobme z toho zarovnanú tabuľku s tromi stĺpcami. Využijeme to, že vo formátovacom reťazci môžeme určiť, na akú šírku sa má dané desatinné číslo vypisovať. V našom prípade to bude šírka `6` znakov, pričom z toho budú `3` desatinné miesta. Hodnota v `{}` zátvorkách môže za znakom `':'` obsahovať takúto šírku výpisu. Všimnite si posledný riadok s volaním `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}')
```

Prvé riadky výpisu teraz už vyzerajú takto:

```
  0  0.000  1.000
  5  0.087  0.996
 10  0.174  0.985
 15  0.259  0.966
 20  0.342  0.940
```

```
 25  0.423  0.906
...
```

Pokúsme sa nakresliť (v textovej ploche pomocou nejakých znakov) priebeh funkcie sínus. Keďže oborom hodnôt tejto funkcie je interval reálnych čísel `<-1, 1>` a my chceme tieto hodnoty natiahnuť na šírku výpisu do 80 znakov, 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')
```

Sínusovka v textovej ploche potom vyzerá takto:

```
                                    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
```

Všimnite si, že všetky tri premenné `uhol_v_radianoch`, `sin_uhla` a `stlpec` v tele cyklu tu slúžia „len“ na zvýšenie čitateľnosti kódu a mohli by sme to zapísať aj bez nich:

```
import math

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

Takýto zápis je menej prehľadný a neodporúčame ho používať najmä pre začiatočníkov.

Importovanie funkcií z knižníc môžeme zapísať aj trochu inak. V tomto druhom variante budú k dispozícii len tie funkcie, ktoré vymenujeme v tomto príkaze. Všeobecný tvar je:

```
from knižnica import funkcia1, funkcia2, ...
```

Týmto príkazom sa naimportujú len vymenované funkcie a ďalej sa budeme na ne v kóde odvolávať priamo ich menom a nebudeme písať meno knižnice a bodku. Napríklad, predchádzajúci príklad môžeme zapísať:

```
from math import sin, radians

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

Je na programátorovi, ktorý zápis použije. Niektorí uprednostňujú predchádzajúcu verziu s písaním mena knižnice aj s bodkou, lebo pri čítaní programu je zrejmejšie, odkiaľ daná funkcia prišla.

Modul random

Aj tento modul obsahuje knižnicu funkcií, ale tieto umožňujú generovanie náhodných čísel. My z tejto knižnice využijeme najmä tieto dve funkcie:

- `randint()` vyberie náhodné číslo z intervalu celých čísel
- `choice()` vyberie náhodný prvok z nejakej postupnosti, napríklad zo znakového reťazca (postupnosti znakov)

Aby sme mohli pracovať s týmito funkciami, nesmieme zabudnúť zapísať:

```
import random
```

Nasledovná ukážka ilustruje volanie funkcie `randint()`. Parametre tejto funkcie udávajú dolnú a hornú hranicu intervalu, z ktorého sa vyberie jedna náhodná hodnota. Každé volanie `random.randint(1, 6)` **náhodne** vyberie jednu z hodnôt intervalu `<1, 6>`, teda z množiny čísel `1, 2, 3, 4, 5, 6`. Môžeme si to predstaviť ako hod hracou kockou, na ktorej sú čísla od 1 do 6. Nalsedovný program vypíše postupnosť 100 náhodných hodov kocky:

```
import random

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

a po spustení dostaneme napríklad takéto 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
```

Veľmi podobne funguje aj druhá funkcia `choice()`. Táto má len jeden parameter, ktorým je nejaká postupnosť hodnôt. Zatiaľ sme sa stretli s dvomi postupnosťami hodnôt: funkciou `range(...)` a so znakovými reťazcami. Ak zapíšeme:

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

Vygeneruje sa náhodné číslo z postupnosti nepárnych čísel: `1, 3, 5, 7, 9`. V knižnici `random` existuje aj funkcia `randrange`, ktorá robí presne toto isté a zapísali by sme to `random.randrange(1, 10, 2)`.

Pre nás je v súčasnosti najzaujímavejšou postupnosťou postupnosť znakov, teda ľubovoľný znakový reťazec. Napríklad volanie:

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

vyberie **náhodnú** hodnotu z postupnosti šiestich znakov - postupnosti samohlások. Podobne by sme mohli zapísať:

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

aj toto volanie vyberie **náhodné** písmeno z postupnosti spoluhlások. Keď to teraz dáme dokopy, dostaneme generátor náhodne 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 spoluhlások a samohlások, teda dvojpísmenových slabík. Vždy, keď budeme potrebovať ďalšie náhodné slovo, musíme spustiť tento program (napríklad pomocou `F5`). Môžeme dostať, napríklad takéto náhodné slová:

```
gugaqo

lupiha

cyxebi
```

Vnorené cykly

Ak by sme potrebovali vygenerovať naraz 10 slov, použijeme znovu for-cyklus. Preto cely náš program (okrem úvodného `import`) obalíme konštrukciou `for`, t.j. všetky riadky súčasného programu posunieme 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)
```

Všimnite si, že v tele vonkajšieho for-cyklu (s premennou cyklu `j`) sa nachádzajú tri príkazy: priradenie, potom tzv. **vnorený** for-cyklus a na koniec volanie funkcie `print()`.

Na nasledovných príkladoch ukážeme niekoľko rôznych situácií, v ktorých sa využije vnorený for-cyklus.

Napíšme najprv program, ktorý vypíše čísla od `0` do `99` do 10 riadkov tak, že v prvom stĺpci sú čísla od `0` do `9`, v druhom od `10` do `19`, … v poslednom desiatom sú čísla od `90` do `99`:

```
for i in range(10):
    print(i, i+10, i+20, i+30, i+40, i+50, i+60, i+70, i+80, i+90)
```

Po spustení dostaneme:

```
0 10 20 30 40 50 60 70 80 90
1 11 21 31 41 51 61 71 81 91
2 12 22 32 42 52 62 72 82 92
3 13 23 33 43 53 63 73 83 93
4 14 24 34 44 54 64 74 84 94
5 15 25 35 45 55 65 75 85 95
6 16 26 36 46 56 66 76 86 96
7 17 27 37 47 57 67 77 87 97
8 18 28 38 48 58 68 78 88 98
9 19 29 39 49 59 69 79 89 99
```

Riešenie tohto príkladu využíva for-cyklus len na vypísanie 10 riadkov po 10 čísel, pričom obsah každého riadka sa vyrába bez cyklu jedným príkazom `print()`. Toto je ale nepoužiteľný spôsob riešenia v prípadoch, ak by tabuľka mala mať premenlivý počet stĺpcov, napríklad, keď je počet zadaný zo vstupu. Vytvorenie jedného riadka by sme teda tiež mali urobiť for-cyklom, t.j. budeme definovať for-cyklus, ktorý je vo vnútri iného cyklu, tzv. **vnorený** cyklus. Všimnite si, že celý tento cyklus musí byť odsadený o ďalšie 4 medzery:

```
for i in range(10):
    for j in range(0, 100, 10):
        print(i+j, end=' ')
    print()
```

Vnútorný for-cyklus vypisuje 10 čísel, pričom premenná cyklu `j` postupne nadobúda hodnoty `0`, `10`, `20`, … `90`. K tejto hodnote sa pripočítava číslo riadka tabuľky, teda premenná `i`. Tým dostávame rovnakú tabuľku, ako predchádzajúci program. Rovnaký výsledok vytvorí aj nasledovné riešenie:

```
for i in range(10):
    for j in range(i, 100, 10):
```

```
        print(j, end=' ')
    print()
```

V tomto programe má vnútorný cyklus tiež premennú cyklu `j` s hodnotami s krokom 10, ale v každom riadku sa začína s inou hodnotou.

Túto istú ideu využijeme, aj keď budeme vytvárať tabuľku čísel od 0 do 99, ale organizovanú inak: v prvom riadku sú čísla od `0` do `9`, v druhom od `10` do `19`, … v poslednom desiatom sú čísla od `90` do `99`:

```
for i in range(0, 100, 10):
    for j in range(i, i+10):
        print(j, end=' ')
    print()
```

Možných rôznych zápisov riešení tejto úlohy je samozrejme viac.

Ešte dve veľmi podobné úlohy:

1. Prečítať celé číslo `n` a vypísať tabuľku čísel s `n` riadkami, pričom v prvom je len `1`, v druhom sú čísla `1 2`, v treťom `1 2 3`, atď. až v poslednom sú čísla od 1 do `n`:

```
2. pocet = int(input('zadaj počet riadkov: '))
3. for riadok in range(1, pocet + 1):
4.     for cislo in range(1, riadok + 1):
5.         print(cislo, end=' ')
6.     print()
```

   Všimnite si mená oboch premenných cyklov `riadok` a `cislo`, vďaka čomu môžeme lepšie pochopiť, čo sa v ktorom cykle deje. Spustíme, napríklad:

```
zadaj počet riadkov: 7
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
1 2 3 4 5 6
1 2 3 4 5 6 7
```

2. Zadanie je podobné, len tabuľka v prvom riadku obsahuje `1`, v druhom `2 3`, v treťom `4 5 6`, atď. každý ďalší riadok obsahuje o jedno číslo viac ako predchádzajúci a tieto čísla v každom ďalšom riadku pokračujú v číslovaní. Zapíšeme jedno z možných riešení:

```
3. pocet = int(input('zadaj počet riadkov: '))
4. cislo = 1
5. for riadok in range(1, pocet + 1):
6.     for stlpec in range(1, riadok + 1):
7.         print(cislo, end=' ')
8.         cislo += 1
9.     print()
```

```
zadaj počet riadkov: 7
1
2 3
4 5 6
7 8 9 10
11 12 13 14 15
16 17 18 19 20 21
22 23 24 25 26 27 28
```

V tomto riešení využívame **pomocnú premennú** `cislo`, ktorú sme ešte pred cyklom nastavili na `1`, vo vnútornom cykle vypisujeme jej hodnotu (a nie premennú cyklu) a zakaždým ju zvyšujeme o `1`.

Cvičenia

L.I.S.T.

- riešenia **aspoň 10 úloh** odovzdaj na úlohový server https://list.fmph.uniba.sk/
- používaj len konštrukcie z prednášky (nepoužívaj žiadne podmienky ani hranaté zátvorky)
- pozri si Riešenie úloh 2. cvičenia

1. Napíš program, ktorý zo znakov hviezdička (`'*'`) vytvorí takýto trojuholník: v 1. riadku je jedna hviezdička, v 2. dve, v 3. tri, …, v `n`-tom riadku je `n` hviezdičiek. Môžeš dostať takýto výstup:

```
zadaj n: 6
*
**
***
****
*****
******
```

2. Napíš program, ktorý bude podobný predchádzajúcej úlohe: program prečíta nejaké slovo a trojuholník bude skladať z písmen tohto slova: v 1. riadku je prvé písmeno, v 2. druhé písmeno ale dvakrát, v 3. tretia písmeno trikrát, …, v poslednom je posledné písmeno veľakrát podľa počtu znakov slova. Môžeš dostať takýto výstup:

```
zadaj slovo: PYTHON
P
YY
TTT
HHHH
OOOOO
NNNNNN
```

3. Aj nasledovný program bude podobný predchádzajúcemu: Program prečíta nejaké slovo a trojuholník sa bude skladať z písmen tohto slova: v 1. riadku je prvé písmeno,

v 2. sú prvé dve, v 3. sú prvé tri, … v poslednom riadku je kompletné slovo. Môžeš dostať takýto výstup:

```
zadaj slovo: Python
P
Py
Pyt
Pyth
Pytho
Python
```

4. Napíš program, ktorý `n`-krát vypíše zadané slovo takto: v 1. riadku bez odsadenia, v 2. s `1` odsadením (4 medzery), v 3. s `2` odsadeniami (8 medzier), v 4. s `3` odsadeniami (12 medzier), pre ďalšie riadky sa to opakuje od začiatku. V programe využi zvyšok po delení, napríklad `i%4*4`. Môžeš dostať takýto výstup:

```
zadaj slovo: Python
zadaj n: 9
Python
    Python
        Python
            Python
Python
    Python
        Python
            Python
Python
```

5. Napíš program, ktorý z hviezdičiek vytvorí takúto pyramídu:
    - v prvom riadku je najprv `n-1` medzier a potom jedna hviezdička
    - v každom ďalšom riadku je o jednu medzeru menej a o dve hviezdičky viac

   Môžeš dostať takýto výstup:

```
zadaj n: 7
      *
     ***
    *****
   *******
  *********
 ***********
*************
```

6. Aj v tejto úlohe treba napísať program, ktorý vytvorí pyramídu z hviezdičiek, len z hviezdičiek bude len obvod trojuholníka, vnútro trojuholníka bude zo znakov mínus (`'-'`). Môžeš dostať takýto výstup:

```
zadaj n: 7
      *
     *-*
    *---*
   *-----*
```

```
  *-------*
 *---------*
************
```

7. Celé číslo môžeme rozobrať na jednotlivé cifry tak, že ho najprv prevedieme na reťazec a potom vo for-cykle každú cifru (ako znak) zvlášť prevedieme na číslo. Napíš program, ktorý prečíta celé číslo, rozoberie ho na cifry, tieto vypíše aj s poradovým číslom a zistí ich súčet. Takýto súčet voláme **ciferný súčet**. Po spustení dostaneš, napríklad:

```
zadaj číslo: 784
1. cifra 7
2. cifra 8
3. cifra 4
ciferný súčet je 19

zadaj číslo: 1003
1. cifra 1
2. cifra 0
3. cifra 0
4. cifra 3
ciferný súčet je 4
```

8. Napíš program, ktorý vyrobí jeden dlhý znakový reťazec, zložený z úsekov hviezdičiek oddelených medzerami: na začiatku je 1 hviezdička (znak `'*'`), za ňou medzera, potom 2 hviezdičky a medzera, 3 hviezdičky a medzera … Počet hviezdičkových úsekov je `n`. Program by mal mať takúto schému:

```
n = ...
retazec = ''
for ...:
    ...
print(retazec)
```

Po spustení môžeš dostať takýto výstup:

```
zadaj n: 10
* ** *** **** ***** ****** ******* ******** ********* **********
```

9. Budeš vytvárať dlhý znakový reťazec podobne ako v predchádzajúcej úlohe. Namiesto úsekov hviezdičiek budeš do reťazca zapisovať čísla z nejakého intervalu (v tvare `'<číslo>'`). Schéma programu by mala byť podobná predchádzajúcej úlohe. Po spustení môžeš dostať takýto výstup:

```
zadaj od: 88
zadaj do: 100
<88> <89> <90> <91> <92> <93> <94> <95> <96> <97> <98> <99> <100>
```

10. Napíš program, ktorý vypíše **naformátovanú** tabuľku mocnín celých čísel z nejakého daného intervalu. Prvý stĺpec tabuľky obsahuje číslo, v druhom je druhá mocnina tohto čísla, v treťom tretia, vo štvrtom štvrtá. Môžeš dostať takýto výstup:

```
zadaj od: 95
zadaj do: 103
 95   9025   857375   81450625
 96   9216   884736   84934656
 97   9409   912673   88529281
 98   9604   941192   92236816
 99   9801   970299   96059601
100  10000  1000000  100000000
101  10201  1030301  104060401
102  10404  1061208  108243216
103  10609  1092727  112550881
```

11. Výpočet **pi** podľa Liebnizovho vzorca je takýto súčet radu:

```
4 - 4/3 + 4/5 - 4/7 + 4/9 - 4/11 + 4/13 ...
```

Napíš program, ktorý vypočíta súčet tohto radu pre prvých `n` členov. Po spustení môžeš dostať:

```
zadaj počet: 10
pi = 3.0418396189294032

zadaj počet: 100000
pi = 3.1415826535897198
```

12. Asi poznáš pesničku `'Sedí mucha na stene, sedí a spí.'`. Napíš program, ktorý si najprv vypýta zoznam nejakých samohlások a potom pre každú z nich vypíše túto vetu tak, že v nej všetky samohlásky nahradí touto konkrétnou. Zrejme využiješ for-cyklus a formátovací reťazec `f'S{i}d{i} m{i}ch{i} ...'`. Môžeš dostať takýto výstup:

```
zadaj samohlásky: eaôiuý
Sede meche ne stene, sede e spe.
Sada macha na stana, sada a spa.
Sôdô môchô nô stônô, sôdô ô spô.
Sidi michi ni stini, sidi i spi.
Sudu muchu nu stunu, sudu u spu.
Sýdý mýchý ný stýný, sýdý ý spý.
```

13. Napíš program, ktorý vytvorí tabuľku násobenia, podobnú malej násobilke. Násobiť sa budú čísla z nejakého daného intervalu: v prvom riadku (aj stĺpci) sú násobky prvého čísla, v druhom druhého, atď. Môžeš dostať takýto výstup:

```
zadaj od: 8
zadaj do: 13
  64   72   80   88   96  104
  72   81   90   99  108  117
  80   90  100  110  120  130
```

```
 88   99  110  121  132  143
 96  108  120  132  144  156
104  117  130  143  156  169
```

Do výpisu tabuľky pridaj prvý stĺpec aj riadok navyše s číslami z daného intervalu, napríklad v takomto tvare:

```
zadaj od: 8
zadaj do: 13
     |    8    9   10   11   12   13
=====|==============================
   8 |   64   72   80   88   96  104
   9 |   72   81   90   99  108  117
  10 |   80   90  100  110  120  130
  11 |   88   99  110  121  132  143
  12 |   96  108  120  132  144  156
  13 |  104  117  130  143  156  169
```

14. Napíš program, ktorý vytvorí takúto tabuľku: pre všetky uhly (v stupňoch) z nejakého daného intervalu a kroku vypíše druhé mocniny príslušných sínusov a kosínusov a aj ich súčet. Druhé mocniny vypíše na šírku `6` a `4` desatinné miesta, súčet bez udania šírky a počtu desatinných miest. Môžeš dostať takýto výstup:

```
zadaj od: 0
zadaj do: 90
zadaj krok: 10
  0 sin**2=0.0000 cos**2=1.0000 súčet=1.0
 10 sin**2=0.0302 cos**2=0.9698 súčet=0.9999999999999999
 20 sin**2=0.1170 cos**2=0.8830 súčet=1.0
 30 sin**2=0.2500 cos**2=0.7500 súčet=1.0
 40 sin**2=0.4132 cos**2=0.5868 súčet=0.9999999999999999
 50 sin**2=0.5868 cos**2=0.4132 súčet=1.0
 60 sin**2=0.7500 cos**2=0.2500 súčet=1.0
 70 sin**2=0.8830 cos**2=0.1170 súčet=0.9999999999999999
 80 sin**2=0.9698 cos**2=0.0302 súčet=0.9999999999999999
 90 sin**2=1.0000 cos**2=0.0000 súčet=1.0
```

15. Napíš program, ktorý vygeneruje na číselnej osi dva **náhodné** body (v intervale `<0, 100>`) a vypočíta ich vzdialenosť. Toto urobí `n`-krát. Môžeš dostať takýto výstup:

```
zadaj n: 6
Prvý bod na priamke je 32, druhý bod 10. Ich vzdialenosť je 22
Prvý bod na priamke je 61, druhý bod 12. Ich vzdialenosť je 49
Prvý bod na priamke je 62, druhý bod 35. Ich vzdialenosť je 27
Prvý bod na priamke je 9, druhý bod 78. Ich vzdialenosť je 69
Prvý bod na priamke je 5, druhý bod 82. Ich vzdialenosť je 77
Prvý bod na priamke je 16, druhý bod 20. Ich vzdialenosť je 4
```

16. Vo vlaku sa vezie `100` cestujúcich. V každej stanici, v ktorej zastane, niekoľko ľudí vystúpi a niekoľko nastúpi. Napíš program, ktorý odsimuluje `n` takýchto staníc s

vystupovaním a nastupovaním cestujúcich. Predpokladáme, že v každej stanici vystúpi aj nastúpi náhodný počet cestujúcich z intervalu `<0, 9>`. Môžeš dostať takýto výstup:

```
zadaj n: 8
Vo vlaku bolo 100 ľudí, 0 nastúpilo, 7 vystúpilo. Zostalo 93.
Vo vlaku bolo 93 ľudí, 4 nastúpilo, 0 vystúpilo. Zostalo 97.
Vo vlaku bolo 97 ľudí, 9 nastúpilo, 5 vystúpilo. Zostalo 101.
Vo vlaku bolo 101 ľudí, 3 nastúpilo, 9 vystúpilo. Zostalo 95.
Vo vlaku bolo 95 ľudí, 6 nastúpilo, 8 vystúpilo. Zostalo 93.
Vo vlaku bolo 93 ľudí, 3 nastúpilo, 4 vystúpilo. Zostalo 92.
Vo vlaku bolo 92 ľudí, 8 nastúpilo, 6 vystúpilo. Zostalo 94.
Vo vlaku bolo 94 ľudí, 8 nastúpilo, 7 vystúpilo. Zostalo 95.
```

17. Dostali sme správu od mimozemšťanov, ktorá je zložená zo znakov `'O'` a `'-'`. Správa obsahuje istý počet riadkov a stĺpcov takýchto znakov. Napíš program, ktorým náhodne vygeneruješ podobnú správu. Môžeš dostať takýto výstup:

```
zadaj počet riadkov: 5
zadaj počet stĺpcov: 28
O-OOO----OO-OOO---O---OOOO-O
OOO-OOOO----OO----O-OOOOO-O-
O-OO-OO-OOO--O-OOO--O----OOO
---OO--OO-O-O--OO----OOOO--O
-O-----O--OOOO-OO-OOO-OO---O
```

18. Budeme simulovať hádzanie dvomi hracími kockami. Zakaždým vypíšeme aj ich súčet. Napíš program, ktorý to simuluje `n`-krát. Môžeš dostať takýto výstup:

```
zadaj n: 4
na 1. kocke padla 1
na 2. kocke padla 4
ich súčet je 5
======================
na 1. kocke padla 4
na 2. kocke padla 5
ich súčet je 9
======================
na 1. kocke padla 6
na 2. kocke padla 1
ich súčet je 7
======================
na 1. kocke padla 4
na 2. kocke padla 1
ich súčet je 5
======================
```

19. Podobný príklad predchádzajúcemu, lenže teraz budeme hádzať ľubovoľným počtom kociek: Napíš program, ktorý si najprv vypýta `n` (počet hádzaní) a počet kociek. Potom `n`-krát vypíše čísla na kockách a ich súčet. Môžeš dostať takýto výstup:

```
zadaj n: 3
zadaj počet kociek: 4
```

```
na 1. kocke padla 3
na 2. kocke padla 2
na 3. kocke padla 2
na 4. kocke padla 2
ich súčet je 9
======================
na 1. kocke padla 4
na 2. kocke padla 6
na 3. kocke padla 1
na 4. kocke padla 5
ich súčet je 16
======================
na 1. kocke padla 1
na 2. kocke padla 4
na 3. kocke padla 6
na 4. kocke padla 3
ich súčet je 14
======================
```

20. Pomocou tohto programu vieme zaplniť štvorcovú tabuľku `n` x `n` číslami od `1` do `n**2`:

```
n = int(input('zadaj n: '))
for i in range(n):
    for j in range(n):
        print(f'{i*n + j + 1:2}', end=' ')
    print()
```

Oprav tento program tak, aby vypísal túto tabuľku trikrát vedľa seba, napríklad takto:

```
zadaj n: 5
 1  2  3  4  5     1  2  3  4  5     1  2  3  4  5
 6  7  8  9 10     6  7  8  9 10     6  7  8  9 10
11 12 13 14 15    11 12 13 14 15    11 12 13 14 15
16 17 18 19 20    16 17 18 19 20    16 17 18 19 20
21 22 23 24 25    21 22 23 24 25    21 22 23 24 25
```

1. Týždenný projekt

L.I.S.T.

- riešenie odovzdaj na úlohový server https://list.fmph.uniba.sk/

Napíš pythonovský skript, ktorý:

1. pomocou príkazu `input('?')` najprv prečíta nejaký reťazec
2. vypíše jeho dĺžku (počet znakov reťazca) pomocou `print('dlzka =', dlzka_retazca)`
3. vypíše prevrátený reťazec pomocou `print('prevrat =', iny_retazec)`
4. vypíše šachovnicu (pomocou `print()`) zo znakov reťazca a znakov `'*'` a `' '`
    - striedajú sa v nej znaky načítaného reťazca a znaku `'*'`

- šachovnica má dvojnásobný počet riadkov (aj stĺpcov) v porovnaní s dĺžkou reťazca (zrejme dĺžka textov v riadkoch bude približne `4`-krát dĺžka vstupného reťazca)

Napríklad, ak zadáme reťazec `abc`, dostávame tento výstup:

```
?abc
dlzka = 3
prevrat = cba
a * b * c *
* a * b * c
a * b * c *
* a * b * c
a * b * c *
* a * b * c
```

Ak zadáme reťazec `x`, dostávame tento výstup:

```
?x
dlzka = 1
prevrat = x
x *
* x
```

Tvoj odovzdaný program s menom `riesenie.py` musí začínať tromi riadkami komentárov (zmeň meno a dátum odovzdania):

```
# 1. zadanie: retazec
# autor: Janko Hraško
# datum: 7.10.2021
```

V programe používaj len konštrukcie jazyka Python, ktoré sme sa učili na prvých 2 prednáškach. Nepoužívaj príkaz `import`.

Súbor `riesenie.py` odovzdávaj na úlohový server https://list.fmph.uniba.sk/ najneskôr do **23:00 7. októbra**, kde ho môžeš nechať otestovať. Odovzdať projekt a aj ho testovať môžeš ľubovoľný počet krát. Za toto tréningové zadanie môžeš získať **2 body**.