Python tutoriál

Obsah

Ahoj, pred oči sa Ti dostal Python tutoriál napísaný pri príležitosti Online Školy Programovania 2020. Jeho autormi sú Roman a Andrej (Andrej a Roman). Za korektúru a rozvrhnutie tutoriálu sme veľmi vďační Žabovi. Ak v tutoriále nájdeš nejaké chyby alebo pasáže, ktoré sme podľa teba nie zrovna dobre podali, neváhaj nás kontaktovať. Ako môžeš spoznať zle napísanú pasáž? Napríklad tak, že ani po niekoľkonásobnom prečítaní nerozumieš tomu, čo text, resp. kód znamená. Kontaktovať nás môžeš buď na Discorde pod nickom Aj0SK a sobkulir alebo na adresách andrejkorman@gmail.com a r.sobkuliak@gmail.com. Budeme Ti veľmi vďační.

1 Úvod

Predtým ako s Pythonom začneme, je dôležité zodpovedať niekoľko otázok. Prvá a najdôležitejšia je, čo je programovanie. Programovanie je vytváranie programov, ktoré na počítači vieme spustiť a majú nejakú funkciu. Príkladom programu môže byť aplikácia na četovanie, webový prehliadač ale aj niečo jednoduchšie, ako napríklad kalkulačka. Dôležité na programoch je, že majú vstup a výstup. Pre program fungujúci ako kalkulačka môže byť vstupom matematický výraz typu $5 + 3 \cdot 7$ a výstupom hodnota tohto výrazu 26. My sa v tomto tutoriále budeme venovať jednoduchým programom. V prípade kalkulačky na vašom počítači ste zrejme zvyknutí zadať vstup teda nejaký matematiký výraz, pomocou tlačidiel, teda nejakého grafického rozhrania (tlačidlá, okná a podobne). Takéto rozhranie naše programy nebudú mať, čo ich mimochodom značne skracuje. Vstup aj výstup bude teda v textovej podobe.

Keď už vieme čo je program, potrebujeme vedieť ako taký program vytvoriť. Na to slúžia, ako už isto tušíte, programovacie jazyky. Tie nám dovoľujú vytvoriť nejaký kód - postupnosť príkazov v danom jazyku, ktoré spolu tvoria program. Program od nás po spustení dostane nejaký vstup a vyprodukuje naň odpoveď (výstup), ktorú typicky nejako zobrazí, napr. vypíše.

1.1 Prečo Python?

Programovacích jazykov existuje neúrekom. Prečo sme v tomto texte zvolili práve Python? Myslíme si, že jeho príkazy sú prirodzené a zároveň je široko využívaný v praxi. Podľa dotazníku StackOverflow z roku 2019 je to druhý najpopulárnejší programovací jazyk. Python vám teda otvorí dvere do mnohých zákutí informatiky :)

1.2 Inštalácia

Predtým, než napíšete svoj prvý program, musíte si Python nainštalovať. Existuje rada veľmi pokročilých nástrojov pre programovanie v Pythone. My s nimi zatiaľ pracovať nebudeme, pretože na úvod sú zbytočne komplikované. Použijeme preto jednoduchý editor s názvom `Python IDLE`, ktorého autor vytvoril aj samotný jazyk Python. Tento editor podporuje všetky štandardné operačné systémy (Windows, MacOS, Linux) a stiahnuť si ho môžete z oficiálnej stránky Pythonu. Odporúčame nainštalovať si najnovšiu verziu, v čase písania tohto textu je to verzia 3.8.2. Pripravili sme tiež videonávod k inštalácii.

2 Prvé programy a `print`

Keď už máme Python nainštalovaný, otvoríme program Python IDLE. To je editor, v ktorom budeme písať naše programy. Zvoľte `File -> New File` a do nového okna, ktoré sa otvorilo, skopírujte program:

```
# Prvy program v Pythone
print(5)
print(5+2+7)
print(7-10)
print(2*15)
print(15/10)
print(15//10)
print(7%3)
```

Následne program uložte pomocou `File -> Save` do ľubovoľného priečinku. Odporúčame vám, aby ste si vytvorili samostatný priečinok, do ktorého budete vaše programy ukladať. Teraz program môžete spustiť pomocou `Run -> Run Module` alebo stlačením `F5`.

To čo vidíte vyššie je kód (postupnosť príkazov) tvoriaci veľmi jednoduchý program. Prvý riadok nie je nijak dôležitý ale je potrebné vysvetliť o čo ide. Občas chcete niekomu, kto po vás váš kod číta zanechať nejakú informáciu. Či už vysvetliť čo nejaká komplikovanejšia časť robí alebo upriamiť jeho pozornosť na niektoré príkazy. V kóde sa však všetko považuje za príkaz. Preto existujú tzv. komentáre - časti textu, ktoré sa nevykonávajú a slúžia pre potreby nás ľudí. Na zanechanie komentára slúži znak #. Všetko čo napíšeme zaň sa považuje

iba za komentár. Pri vykonávaní jednotlivých príkazov Python tieto komentáre ignoruje. My vám za týmto znakom budeme zanechávať nejaké odkazy pre vaše lepšie porozumenie naším príkladom. Teraz už poďme na to podstatné.

V tomto programe používame funkciu `print`. Funkcia je nejaká menšia postupnosť príkazov, ktorá sa natoľko opakuje alebo je logicky ucelená, že dostala názov. Vlastne je to taký malý program v programe. Funkcii zväčša dáme niekoľko *parametrov*, teda jej povieme na čom má vykonať operáciu na ktorú je určená. Vieme asi uhádnuť, čo táto konkrétna funkcia robí – vypisuje (po anglicky print) na výstup to, čo je vo vnútri zátvoriek. Toto je zatiaľ všetko, čo vám treba o funkciách vedieť. Funkcie sú pomerne zložitá vec, ktorej sa bližšie venuje kapitola 10.

Vyššie vidíme, že `print` je funkcia, ktorá má ako parameter to, čo chceme vypísať. Program si spustite a pozrite sa na jeho výstup. Zamyslite sa, ktoré riadky vás prekvapili. Je logické, že sa výraz 15/10 nevyhodnotil na celé ale desatinné číslo. Špeciálnu pozornosť si ale zaslúži výraz 15//10. Dve znamienka delenia po sebe v Pythone označujú často používané tzv. celočíselné delenie, teda delenie bezozvyšku. $x//y$ vieme interpretovať ako: „Najviac koľkokrát sa zmestí y do x?“ Inak povedané ide o klasické delenie s tým, že vo vo výsledku vynecháme všetko za desatinnou čiarkou. Zrejme je pre vás novým ešte výraz 7%3 kde % označuje zvyšok po celočíselnom delení. Výsledkom tejto operácie je tá časť čísla čo z neho „ostane“ po celočíselnom delení. $7\%3 = 1$ a to preto, že $7//3 = 2$ a to čo zostalo vieme vyjadriť ako $7 - 2 \cdot 3 = 1$.

V tomto programe sme si v skutočnosti iba ukázali, že `print` vie vypísať číslo. Všetky výrazy ($7-10$, $5+2+7$ atď.) obsahujúce základné matematické operácie sa totiž priamo vyhodnotia na číslo (-3, 14) a úlohou `print` je už len vypísať dané číslo. `print` vie ale toho samozrejme viac. Skopírujte si do vášho súboru s kódom nasledovný kód:

```
print("Python tutorial!")
print("Python" + "tutorial!")
print(2 + 3)
print("2" + "3")
```

Znova si kód prezrite, spustite a porovnajte svoje očakávania s výsledkami na výstupe. Na prvom riadku vidíme, že vieme vypísať aj text. Takýto text v Pythone nazývame *reťazec znakov* alebo skrátene *reťazec*. Ide o znaky ohraničené úvodzovkami, teda `"<text>"` alebo `'<text>'`. Všimnite si, že nezavísi na tom, či použijeme „obyčajné úvodzovky“, teda `"`, alebo apostrof, teda `'`. Najčastejšie keď hovoríme v Pythone o znakoch, myslíme tým anglickú abecedu, číslice, interpunkciu, zátvorky a podobne. V tomto texte vôbec nepracujeme s dĺžňami, mäkčeňmi a slovenskou abecedou.

V druhom riadku vidíme, že podobne ako čísla aj reťazce vieme sčítavať. Čo urobí znamienko plus v tomto prípade je, že zlepí reťazce za seba. Ako je možné, že výsledok 3. a 4. riadku sa líši? Dôvodom je, že v prvom prípade ide o plus aplikované na čísla. V druhom prípade sme aplikovali plus na reťazce. Python vidí, že v 4. riadku sa naľavo aj napravo od plus nachádzajú čísla s úvodzovkami okolo, čím sme dali najavo, že chceme tieto čísla reprezentovať ako reťazce a Pythonu teda neostáva nič iné, ako použiť plus pre reťazce. Keď sa pokúsime tieto dve reprezentácie skombinovať a napíšeme

```
print(2 + "3")
# Chyba!
# TypeError: unsupported operand type(s) for +: 'int' and 'str'
# Tento program po spusteni skonci chybou vyssie. Python nam v tejto
# chybe hovori, ze operator + nemozno pouzit medzi typmi 'int' a 'str'.
```

tak spôsobíme chybu. Je to z dôvodu, že Python nerozumie ako má sčítať *int*, teda *integer* (celé číslo) a *str*, teda *string* (reťazec). Toto neplatí, ak plus nahradíme znamienkom krát. Tak schválne, čo vypíše program:

```
print(2 * "Ahoj!")
```

Posledná vec, ktorá sa týka príkazu `print` je, že po tom, čo tento príkaz vypíše čo sme mu prikázali, ďalšie zavolanie funkcie `print` v rovnakom programe píše na ďalší riadok. Teraz vám niečo prezradíme. To, že text skočí na ďalší riadok je spôsobené tým, že sa v texte, ktorý vypisujete skrýva znak, ktorý nepatrí medzi tzv. tlačiteľné znaky. Teda ho nie je pri vypisovaní vidieť ale spôsobí, že text za ním je už na novom riadku. Tento znak vieme zapísať ako `\n`. Vyskúšať si to môžete tak, že do programu napíšete príkaz:

```
print("Tu bude novy riadok->\n a aj tu->\nbol novy riadok.")
# Vystup:
# Tu bude novy riadok->
#  a aj tu->
# bol novy riadok
```

Funkcia `print` tak nerobí nič iné ako to, že pridá na koniec toho, čo chceme vypísať `\n`. Tvorcovia funkcie `print` mysleli na to, že možno za tým, čo vypíšeme nechceme dať znak konca riadka a pridali možnosť ako znak nového riadku pri výpise zmeniť. Funkcia `print` mala doteraz v zátvorkách za sebou iba jeden *parameter*

a to, čo má vypísať. Parametrov môže mať funkcia viac a oddeľujeme ich čiarkou. Funkcia `print` má druhý parameter, ktorým vieme ovládať čo bude na konci vypísaného textu. Tento parameter je navyše *pomenovaný*, a asi nás neprekvapí, že sa volá `end`. Ak by sme teda namiesto znaku nového riadku chceli na konci dve `X`, spravíme to nasledovne.

```
print("Toto␣cele", end="XX")
print("chcem␣na␣jednom␣riadku.")
# Toto celeXXchcem na jednom riadku.
```

Špeciálne, ak za `print` nechceme vypísať nič, parametru `end` dáme prázdny reťazec.

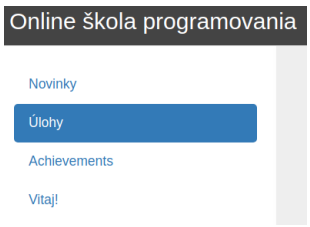
```
print("Zostavame␣", end='')
print("na␣rovnakom␣riadku.")
# Zostavame na rovnakom riadku.
```

Príklady

Na konci kapitoly nájdeš vždy sadu úloh, ktoré by si si mal predtým ako budeš pokračovať vyriešiť. Úlohy slúžia na preverenie tvojich schopností a zväčša sa stupňujú v obtiažnosti. V tejto kapitole sme si pre teba prichystali iba jednu úlohu: Ahoj

Zrejme si však nikdy ešte neodovzával program na náš testovač a tak ti celý proces v skratke vysvetlíme. Testovač je náš server, kam pošleš svoj program, ktorý rieši nejakú úlohu. Náš testovač tvoj program spustí na rôznych vstupoch a zistí, či robí naozaj to, čo sa v zadaní požaduje. Poďla toho ho ohodnotí OK, WA (wrong answer, nesprávna odpoveď), prípadne nejak inak. Všetky možné výsledky odovzdávania nájdeš v sekcii `Odpovede testovača`. Neboj sa, o nič nejde a môžeš odovzdávať ľubovoľne veľakrát. Neodporúčame ti ale zaseknúť sa a hodiny riešiť ten istý príklad. Pokiaľ ti príklad nejde a začínaš byť frustrovaný ozvi sa na Discorde v časti začiatočníci alebo napíš priamo jednému z užívateľov sobkulir alebo Aj0SK a my ti radi poradíme.

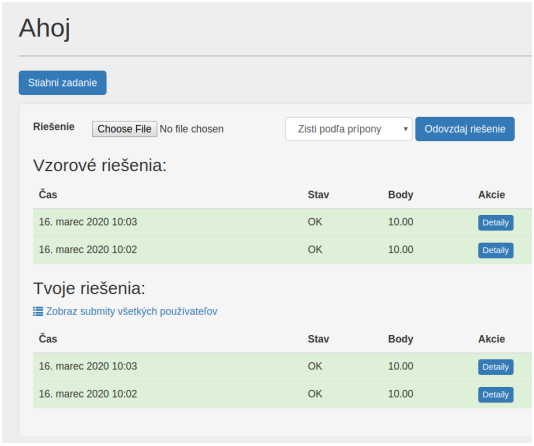
Teraz už k tomu ako odovzdať svoj program na testovač. Roman pripravil krátky videotutoriál. Testovač nájdeš na adrese testovac.ksp.sk. Prvýkrát zvoľ možnosť `Zaregistruj sa` a zaregistruj sa ako na hocijakej inej stránke. Po prihlásení sa môžeš stránkou preklikávať pomocou panelu naľavo. Keď sa rozhodneš riešiť, prejdi pomocou panelu naľavo do časti `Úlohy`.



Potom si všimni sadu príkladov s názvom `Úvod do programovania` a klikni na `Zobraziť úlohy` čím sa ti pod týmto tlačidlom vyrolujú úlohy:

Po kliknutí na príklad sa už dostaneš na stránku, na ktorú by ťa inak zaviedol link úlohy v tomto tutoriále. Odovzdať príklad môžeš tak, že ho nahráš pomocou tlačidla `Choose file`, príponu necháš na voľbe `Zisti podľa prípony` a stlačíš `Odovzdaj riešenie`. Dôležité je, že po vyriešení úlohy bude toto okno s úlohou vyzerať nejak takto:

Dole sa nachádzajú tvoje submity (pokusy o riešenia), pričom správne riešenie má pri sebe stav OK. Po tom, čo si doriešil úlohu si môžeš rozkliknúť aj `Vzorové riešenia`, ktoré sú prístupné zväčša v dvoch až troch jazykoch.



3 Premenné

Program si k svojmu chodu potrebuje často pamätať nejaké informácie. Či už je to vstup, ktorý programu užívateľ zadal alebo nejaké medzivýsledky výpočtu, ktorý program vykonáva. V Pythone na toto slúžia premenné. Premenná je krabička (miesto v pamäti počítača), kde si program ukladá dáta. Naviac je táto krabička pomenovaná. Akonáhle sa vytvorí krabička s názvom napríklad x, vieme do nej uložiť číslo pomocou `=`.

```
x = 4   # do krabicky x vlozime cislo 4
```

Neskôr potom vieme použiť hodnotu v tejto krabičke napríklad takto:

```
print(5*x)   # pri vyhodnocovani vyrazu sa x nahradi za stvorku
```

Do krabičky môžeme hodnotu vložiť a aj ju použiť opakovane.

```
a = 10
print(4*a)   # vypise: 40
a = 7
print(4*a)   # vypise: 28
```

Zatiaľ sme zatajili ako sa taká premenná vytvára. Tajomstvom je, že Python premennú vytvorí automaticky keď jej prvý krát priradíme nejakú hodnotu. Na vytvorenie premennej a priradenie hodnoty do nej teda stačí napísať:

```
a = 4   # do a priradime 4, premenna a sa pritom vytvori
b = 2 * a - 3   # do b priradim hodnotu vyrazu napravo, teda 5
c, d = 10, 12   # vieme robit aj viac priradeni naraz (c = 10 a d = 12)
a = 8 # zmenili sme hodnotu a, teraz sa uz premenna znova nevytvorila
```

Na začiatku sme hovorili, že do premennej si program ukladá dáta avšak doteraz sme iba ukázali, ako do premennej vložiť číslo. Podľa toho, aký typ dát práve premenná v sebe ukladá hovoríme aj o jej `type`. My sme sa zatiaľ v tomto tutoriále skryto stretli s tromi typmi. Prvým bolo celé číslo alebo tiež `int`. Druhým bolo desatinné číslo alebo aj `float`. Tretím reťazec alebo `string`. V nasledujúcej tabuľke prinášame aké základné typy v Pythone existujú.

Typ	Slovný popis	Príklady
int	celé čísla	1, -10, 0, 123
float	desatinné čísla	0.5, .3, 3.14
bool	pravda/nepravda	`True`/`False`
string	reťazec znakov	„abc“, ’abc’
žiadny	ani jeden z typov	`None`

Ďalším dôležitým pojmom je pojem `operátor`. Spomenuli sme si už premenné a aj to, že majú rôzne typy a vieme ich použiť v zložitejších výrazoch. Pod výrazom myslíme napr. $5x + 20 + y$. Tento výraz sa vyhodnotí a jeho výsledok sa nám vráti ako číslo. Vedeli by sme teda napísať napr. $vysledok = 5x + 20 + y$. V tomto výraze používame napr. operátor $+$ a to hneď dva krát. $+$ vezme vec napravo, naľavo, sčíta ich a vráti nám výsledok. Python má samozrejme viac operátorov. Niektoré už poznáte ako napr. $-$, $//$, $*$... Vieme už aj o tom, že rovnaký operátor môže fungovať inak keď má napravo a naľavo iné typy a teda vracia aj rôzne typy.

```
a, b = 2, 12 # do a prirad 2 a do b prirad 12
c = a + b # c bude 14, + tu vracia cele cislo
r1 = "Ano"
r2 = "Nie"
r3 = r1 + r2 # r3 bude "AnoNie", lebo + tu vracia retazec
r4 = a * r1 # r4 bude "AnoAno", lebo + tu vracia retazec
```

Aké ešte operátory existujú? Dôležitým typom v programovaní je typ `bool`. Ten môže reprezentovať iba dve hodnoty a to `True` a `False`. Dôležitosť tohto typu sa naplno prejaví v kapitole o podmienkach. Príklad použitia:

```
a = True
b = False
print(a)
print(b)
```

Operátory, ktorým sme sa doteraz venovali vracali buď celé číslo, desatinné číslo alebo reťazec. Dôležitými operátormi sú aj tie, ktoré vracajú bool a to == a ! =. Prvý funguje ako porovnanie, pričom vráti `True` vtedy keď sa vec napravo a naľavo rovnajú. Druhý je jeho presným opakom, teda vrátu `True` ak sa vec napravo a naľavo nerovnajú. Príklady použitia:

```
a, b = 5, 10
c = (a==b) # v c je False
d = (a==5) # v d je True
e = (a!=b) # v e je True
# pre citatelnost sme vyrazy napravo od = dali do zatvorky
```

Ďalšími užitočnými operátoromi, vracajúcimi `bool` sú <, >, >= a <=. Príklad použitia.

```
a = 5>3 # a bude True
b = 7<=8 # b bude True
```

4 Načítavanie vstupu a výpis výstupu

Na načítavanie zo vstupu používame funkciu `input`. Po jej použití program načíta jeden riadok zo vstupu a vráti nám ho ako *reťazec (string)*. My si musíme samozrejme tento reťazec niekam uložiť. Treba si dať pozor na to, že načítaný vstup je reťazec. Ak chceme so vstupom pracovať ako s číslom, musíme ho najskôr *skonverťovať* (premeniť) na celé číslo pomocou funkcie `int`.

```
a = input() # nacita jeden riadok zo vstupu a ulozi ho do premennej a typu string(retazec)
print(a + 10) # Chyba! Premenna a obsahuje string, 10 je int.
```

V predchádzajúcom príklade vidíme nesprávne použitie funkcie `input`. Ak si ešte spomínate na funkcie a ich parametre, môžete vidieť, že tejto funkcii sme žiadne parametre nedali(v jej zátvorkách sa nič nenachádza). To, čo robíme na prvom riadku síce je správne ale v druhom riadku sa prejaví chyba, ktorú sme nezamýšľali urobiť. Do *a* sme chceli načítať číslo a v skutočnosti sme doň načítali iba reťazec. Aby sme mohli s premennou *a* pracovať ako s číslom, musíme výstup, teda to čo nám funkcia `input` vrátila a je typu reťazec, prekonvertovať na int, teda typ celé číslo. Na to použijeme funkciu `int`, ktorá to, čo jej zadáme ako parameter premení na int a vráti nám to.

```
a = int(input()) # ak chceme vytvorit premennu a, typu int, musime retazec skonvertovat na cislo
print(a + 10) # Ok, premenna aj hodnota, ktoru pricitavame je typu int.
```

Na vypisovanie opäť použijeme funkciu `print`. Tej môžeme ako parameter dať napr. premennú/é, ktorú chceme vypísať.

```
# Trik: priradenie viacerych premennych naraz. Zapis je ekvivalentny s
# a = 10
# b = 15
a, b = 10, 15
# print s viacerymi parametrami vypise tieto parametre oddelene medzerou
# Pretoze sme nastavili end='', nevypise sa znak konca riadku
print(a, b, end='')
print(a)
# Vystup:
# 10 1510
# Vysvetlenie:
# Najskor prvy print vypise "10 15" (bez konca riadku)
# a potom druhy print prida 10 (s koncom riadku).
```

Priklady

Číslo, Zámena, Obdĺžnik 1

5 Podmienky

Často sa stane, že program potrebuje urobiť nejaké rozhodnutie. Program je vec, ktorá sa správa pri každom spustení rovnako. Nie však na každom vstupe. To znamená, že ten istý vstup vždy spôsobí rovnaké správanie programu. Kedže ale dávame programu pri rôznych spusteniach často rôzne vstupy, je nutné, aby na ne program nejako zareagoval. Často napríklad chceme aby sa určitý kód spustil iba za splnenia nejakých podmienok. Príkladom môže byť program, ktorý vypíše, či číslo na vstupe je alebo nie je 0. Vhodným prostriedkom na dosiahnutie tohto správania programu je príkaz `if` (ak). Ten zjednodušene vyzerá `if <podmienka>: <prikaz>`. To znamená, že ak platí `<podmienka>` vykonaj `<príkaz>`. `<podmienka>` je v tomto prípade akýkoľvek výraz vracajúci `bool` (teda `True` alebo `False`). **Nezabudnite**, že porovnávanie dvoch vecí sa vykonáva pomocou **dvoch** rovná sa `(==)` a nie jedného`(=)`.

```
x = int(input())
if x == 0:
    print("Je to nula!")
if x != 0:
    print("Nie je to nula!")
```

V predchádzajúcom príklade je vidieť ešte jednu dôležitú vec a to odsadenie príkazov. Hovoríme o viditeľnej medzere v 3. a 5. riadku. Odsadením hovoríme pythonu, že tento príkaz patrí k predchádzajúcej podmienke. Je to dôležité z dôvodu, keby chceme napr. pri splnení prvej podmienky vykonať viac ako jeden príkaz. V tomto prípade by stačilo len dopísať rovnako odsadený príkaz:

```
# Program zisti, ci je na vstupe 0.
x = int(input())
if x == 0:
    print("Je to nula!")
    print("Vypisem este toto!")

# Vstup1:
# 0
# Vystup1:
# Je to nula!
# Vypisem este toto!

# Vstup2:
# 3
# Vystup2:
# (prazdny)
```

Ak by sme druhý výpis neodsadili, `Vypisem este toto!` by sa vypísalo aj pre nenulový vstup. Odsadenie môžeme dosiahnúť pomocou stlačenia tabulátora.

```
# Program zisti, ci je na vstupe 0.
x = int(input())
if x == 0:
    print("Je to nula!")
print("Vypisem este toto!")

# Vstup1:
# 0
# Vystup1:
# Je to nula!
# Vypisem este toto!

# Vstup2:
# 3
# Vystup2:
# Vypisem este toto!
```

Tak ako v predchádzajúcom príklade sa občas stane, že existujú práve 2 možnosti toho čo chceme urobiť. Vtedy je nápomocný konštrukt `if <podmienka>: <prikaz> else: <prikaz>`. Teda ak platí podmienka vykonáme prvý príkaz a ak nie tak druhý.

```
x = int(input())
if x == 0:
    print("Je to nula!")
else:
    print("Nie je to nula!")
```

Poslednou možnosťou je, že máme viac ako 2 možnosti. Vtedy použijeme konštrukt `if <podmienka>: <prikaz> elif <podmienka>: <prikaz> else: <prikaz>`. Táto možnosť je vhodná práve keď máme 2 a viac možností pričom chceme aby sa vykonala **práve jedna**. Počet `elif` je ľubovolný ale pre čitateľnosť zväčša neodporúčame viac ako 3-4.

```
# ak nam nestacia 2 vetvy (if, else), mozeme pridat dalsie pomocou 'elif'
if x % 2 == 0:
  print('x je parne')
elif x % 3 == 0:
```

```
  print('x je delitelne 3')
else:
  print('x nie je delitelne ani 2 ani 3')
```

Znova si všimnite, že každý príkaz za dvojbodkou je na samostatnom riadku a je odsadený od podmienky.

Podmienka je ľubovoľný výraz, ktorý vieme vyhodnotiť ako `True` alebo `False`. Takisto však vieme skladať viac výrazov pomocou logických spojok `and` a `or`. Výraz `<podmienka1> and <podmienka2>` sa vyhodnotí ako `True` ak sú obe podmienky pravdivé. Výraz `<podmienka1> or <podmienka2>` sa vyhodnotí ako `True` ak aspoň jedna z podmienok je pravdivá. V podmienkach môžeme používať zátvorky na zvýšenie prehľadnosti poradia operácii.

```
if (x % 2 == 0) and (x % 3 == 0):
  print('x je delitene 6')

# podmienka vyssie je ekvivalentna vnorenym podmienkam
if x % 2 == 0:
  if x % 3 == 0:
    print('x je delitene 6')

if (meno == 'Jozko') or (meno == 'Maria'):
  print('meno je Jozko alebo Maria')
```

Priklad

Rovnaké, Znamienko, Najmenší, Prostredný, Priemer, Zvyšok, Nenajväčší

6 Cykly

V programe potrebujeme často nejakú časť príkazov opakovať viackrát. Napríklad ak by sme potrebovali vypísať čísla od 1 do 5, s doterajšími znalosťami by sme vytvorili nasledovný program.

```
print(1)
print(2)
print(3)
print(4)
print(5)
```

6.1 While cyklus

Ako asi už tušíte, musí to ísť aj krajšie. Použijeme preto cyklus `while` (po slovensky *pokiaľ*).

```
# Vypis cisel od 1 do 5.
i = 1
while i <= 5:
  print(i)
  i += 1
```

Príkaz `while` opakuje vykonávanie odsadených príkazov, *pokiaľ* platí podmienka uvedená za kľúčovým slovom `while`. Teraz si toto správanie popíšeme na uvedenom programe. Program najskôr priradí 1 do premennej `i`. Následne sa pri príkaze `while` otestuje podmienka `i <= 5`, ktorá platí, pretože `i` obsahuje hodnotu 1. Vykoná sa preto príkaz `print(i)` a premenná `i` sa zvýši o 1 (`i += 1`). Následne sa znovu otestuje podmienka `i <= 5`, ktorá stále platí, pretože `i` obsahuje hodnotu 2. Takto program pokračuje ďalej až do stavu, kedy sa premenná `i` dostane na hodnotu 6. Vtedy sa podmienka `i <= 5` nesplní, odsadené príkazy pod cyklom sa nevykonajú a začnú sa vykonávať až príkazy nasledujúce za `while` cyklom. V tomto prípade náš program žiadne príkazy za cyklom nemá, takže skončí.

6.2 For cyklus

Často sa ocitneme v situácii, kedy vieme presný počet opakovaní cyklu. Rovnako tomu bolo aj v prípade vypísania čísel od 1 do 5 (počet opakovaní je 5). V tomto prípade je prehľadnejšie použiť cyklus `for` (po slovensky *pre*).

```
# Vypis cisel od 1 do 5.
for i in range(1, 6):
  print(i)
```

Za kľúčovým slovom `for` nasleduje meno premennej, v tomto prípade *i*, v ktorej sa budú postupne meniť hodnoty pri jednotlivých „otočkách“ cyklu. Za menom nasleduje kľúčové slovo `in` a za ním výraz `range(1,`

`6)`. Práve `range(1, 6)` zabezpečí, že v premennej i sa postupne ocitnú hodnoty z rozsahu od 1 do 5. To môže byť mätúce, pretože druhý argument v `range(1, 6)` je 6 a nie 5. Funckii `range` totiž do druhého argumentu musíme dať vždy číslo tesne za našou zamýšlanou poslednou hodnotou i.

Teraz už rozumieme hlavičke `for`-cyklu. Samotné vykonávanie funguje podobne ako pri `while` cykle. Do premennej `i` sa najskôr priradí hodnota 1 (prvý parameter `range`), vykoná sa telo (`print(i)`). Hodnota i sa zvýši o 1, vykoná sa telo, atď. až pokiaľ i nedosiahne hodnotu 6 (druhý parameter `range`). Pre túto hodnotu sa už telo nevykoná.

6.3 Prefíkanejšie cykly

Čo ak by sme chceli vytvoriť cyklus, ktorý preskakuje každé druhé číslo (1, 3, 5, ...)? Alebo cyklus, ktorý ide odzadu (10, 9, 8, ...)? Ukážeme si teraz ako na to.

Funkcia `range` má 3 varianty, vďaka ktorým vieme vytvoriť prefíkanejšie cykly:

- `range(kon)` - vytvorí rozsah s prvkami od 0 po $kon - 1$ vrátane.
- `range(zac, kon)` - vytvorí rozsah s prvkami od zac po $kon - 1$ vrátane.
- `range(zac, kon, k)` - vytvorí rozsah s prvkami od zac po $kon - 1$ vrátane, obsahujúc každé k-te číslo.

Použitie:

```
# Vypis cisel od 0 do 7.
for i in range(8):
  print(i)

# Vypis cisel od -2 do 2.
for i in range(-2, 3):
  print(i)

# Vypis neparnych cisel do 9.
for i in range(1, 10, 2):
  print(i)

# Vypis cisel od 10 do 1 (pozadu).
# Vsimnite si, ze posledny parameter je zaporny, takze cisla
# sa budu od zaciatku po koniec zmensovat. Navyse druhy argument
# (koniec cyklu) je 0 a nie 1, pretoze range prijma cislo tesne za
# zamyslanou poslednou hodnotou.
for i in range(10, 0, -1):
  print(i)
```

Občas potrebujeme z cyklu vyskočiť skôr alebo nejakú hodnotu preskočiť. Na to slúžia príkazy `continue` a `break`. Príkaz `break` zabezpečí vyskočenie z *aktuálne* vykonávaného cyklu. Program potom pokračuje za telom cyklu, z ktorého vyskočil. Napríklad program nižšie sa zastaví už na čísle 4.

```
# Vypis cisel od 0 do 3.
for i in range(11):
  if i == 4:
    break
  print(i)
# Vystup: 0, 1, 2, 3
# Cislo 4 sa uz nestihne vypisat.
```

Príkaz `continue` zabezpečí, že vykonávanie cyklu preskočí na ďalšiu hodnotu. Zvyšok tela za príkazom `continue` sa tak už nevykoná. Napríklad v príklade nižšie sa pri nepárnych číslach preskočí výpis `Je párne!`.

```
# Vypis cisel od 0 do 6.
for i in range(7):
  print(i)
  # Zvysok po deleni je 1, cislo teda nie je parne
  if i % 2 == 1:
    continue
  print('␣Je␣parne!')
# Vystup:
# 0
# Je parne!
# 1
# 2
# Je parne!
# (atd)
```

6.4 Príklady

Cykly, Pásik, Obdĺžnik 2, Trojuholník, Pyramída, Najväčší, Fibonacci

7 List

Prestavte si, že na vstupe nedostaneme zopár čísel, ale niekoľko desiatok. Kam by sme tieto čísla uložili? Mohli by sme si vytvoriť desiatky premenných a vstup naukladať do nich. To by však nebolo veľmi prehľadné; a navyše čo ak by bol počet čísel na vstupe premenný? (Napríklad prvé číslo vstupu by určovalo koľko čísel bude nasledovať)

Pre takýto prípad musíme rozšíriť náš arzenál o `list`. List deklaruje naraz množstvo premenných uložených za sebou, ku ktorým sa dá pristupovať pomocou ich pozície.

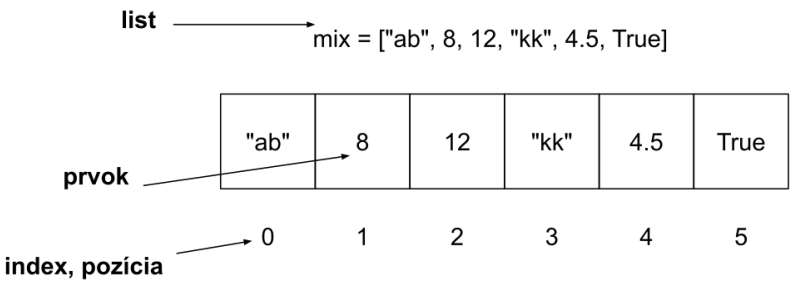
```
# List obsahujuci cisla 1 az 5
cisla = [1, 2, 3, 4, 5]

# Vypisanie prveho prvku pola "cisla".
print(cisla[0])
# Vystup: 1

# Chyba!
print(cisla[5])
# IndexError: list index out of range
# Python nam hovori, ze index 5 je mimo povoleny rozsah (index out of range)
# Pole "cisla" ma 5 prvkov indexovanych od 0, takze posledny platny index je 4, index 5 je neplatny.
```

Ako ukazuje príklad vyššie, vytvorenie listu má formát `[<hodn1>, <hodn2>, ...]`, kde hodnota môže byť ľubovoľná premenná alebo hodnota (*literál*). Listy sú číslované od 0. Ak preto vytvoríme reťazec dĺžky 5 ako v príklade vyššie, platné indexy (pozície) sú 0 až 4.

List si tiež môžete predstaviť veľa krabičiek vedľa seba, ku ktorým pristupujeme menom listu a zároveň pozíciou. Pritom krabičky na sebe nijako nezávisia a môžu obsahovať rôzne hodnoty a dokonca, ako ukazuje list z obrázku, aj rôzne typy. V liste z obrázka je na pozíciách 0, 3 `string`, na pozíciách 1, 2 `int`, atď. Na koniec



Obr. 1: Ukážka 6-prvkového listu spolu s indexami.

listov často potrebujeme pridávať hodnotu. Na to slúži *metóda*[1] `append` (po slovensky "pripojiť"). Táto metóda pridá novú hodnotu **na koniec** listu. Ak chceme pristúpiť k niektorému z prvkov listu, stačí za menom listu pozíciu obaliť do hranatých zátvoriek, teda `<list>[<pozicia>]`. Napríklad `cisla[0]`.

```
# Program dostane cislo n, pocet cisel, ktore budu nasledovat
# na dalsich riadkoch. Nasledne nacita tieto cisla a vypise ich.
n = int(input())
vstup = []
for i in range(n):
  vstup.append(int(input()))

for i in range(n):
  print(vstup[i])

# Napriklad pre vstup:
# 3
# 1
# 2
# 3
# Vyda program vystup (tu skratene do jedneho riadku):
# 1 2 3
```

Tento program je možno najkomplexnejší, s ktorým sme sa zatiaľ v tomto návode stretli. Na začiatku načíta číslo n, počet čísel, ktoré budú nasledovať na samostatných riadkoch. Následne v cykle od 0 po $n-1$ (vrátane) načíta vždy jedno číslo `int(input())` a pripojí ho do listu *vstup* pomocou `append`.

[1] Métoda je funkcia naviazaná na konkrétny *objekt*, pričom objekt v Pythone je všetko (premenné, hodnoty, funckie, ...). Zavolať metódu môžeme ako `<objekt>.<metoda>()`. Napríklad `cisla.append(8)`. Môžete si to predstaviť tak, že `append` je obyčajná funckia, ktorá ako prvý parameter dostane pole `cisla` a ako druhý parameter 8.

Následne program načítané čísla vypíše. Vo `for`-cykle od 0 po $n-1$ postupne prístupi pomocou premennej i k prvkom poľa na indexoch 0 až $n-1$ a vypíše ich.

Posledný cyklus sa dá zapísať trochu jednoduchšie. Samotný `for` cyklus má totiž hlavičku v tvare `for <premenna> in <list>:`. Premenná z cyklu teda môže nadobúdať hodnoty z ľubovoľného listu. To znamená, že funkcia `range`, ktorú sme si uviedli v predošlej kapitole nerobí nič iné ako to, že vytvorí pole.[2] Skúsme to!

```
# Funkciu list, ktora prekonvertuje range na list musime pouzit
# preto, lebo range je "lenivy" a na list sa premeni len ked naozaj musi.
print(list(range(5)))
# Vystup: [0, 1, 2, 3, 4]

print(list(range(0, 10, 2)))
# Vystup: [0, 2, 4, 6, 8]
```

Teraz už môžeme `list` *vstup* z príkladu, kde sme načítavali n čísel, vypísať trochu jednoduchšie.

```
# Do listu "vstup" priradzujeme "natvrdo" hodnoty aby bol tento kod
# spustitelny bez chybovej hlasky.
vstup = [1, 2, 3]

# Premenna "x" postupne nadobudne hodnoty 1, 2, 3.
for x in vstup:
    print(x)
# Vystup (skrateny na jeden riadok):
# 1 2 3
```

Príklady vyššie počítali s tým, že čísla dostaneme rozdelené do riadkov. Čo ak by sme n čísel dostali na jedinom riadku? Funkcia `input` by ich všetky načítala do jedného reťazca a funkcia `int` by zlyhala, pretože vie skonvertovať iba **jedno číslo z reťazca**, nie viacero.

```
# Program dostane cislo n, pocet cisel, ktore budu nasledovat
# na dalsom riadku oddelene medzerou.
n = int(input())
cisla = input()
int(input())
# Chyba!
# Naprilkad pre vstup:
# 3
# 3 4 5
# Dostaneme chybu:
#    ValueError: invalid literal for int() with base 10: '3 4 5'
# To znamena, ze funkcia int() nevie skonvertovat '3 4 5' na jedine cislo.
```

Pomôžeme si preto metódou `split`, ktorá rozseká čísla v reťazci podľa medzier a vráti pole obsahujúce jednotlivé čísla (ako reťazce!).

```
# Program dostane cislo n, pocet cisel, ktore budu nasledovat
# na dalsom riadku, oddelene medzerou.
n = int(input())
# premenna cisla bude obsahovat cely riadok vstupu ako string,
# napriklad '3 4 5'
cisla = input()

# rozsekaneStr bude obsahovat jednotlive cisla v poli ako stringy,
# pretoze metoda split rozdeli obsah premennej cisla podla medzier.
# Napriklad: ['3', '4', '5']
rozsekaneStr = cisla.split()

# Z jednotlivych stringov v poli rozsekaneStr potrebujeme dostat inty.
# Napriklad: [3, 4, 5]
rozsekaneInt = []
for x in rozsekaneStr:
  rozsekaneInt.append(int(x))

print(rozsekaneInt)

# Vstup:
# 3
# 3 4 5
# Vystup:
# rozsekaneStr: ['3', '4', '5']
# rozsekaneInt: [3, 4, 5]
```

Ukážeme si ešte jeden príklad použitia listu, tentokrát bude obsahovať reťazce. Program má za úlohu vypísať kód mesiaca zo vstupu.

```
# Program dostane cislo mesiaca (indexovaneho od 0) a vypise jeho skratku.
mesiace = ["jan", "feb", "mar", "apr", "maj", "jun",
           "jul", "aug", "sep", "okt", "nov", "dec"]
n = int(input())
print(mesiace[n])
```

[2] Realita je ešte o trošku komplikovanejšia, pretože range je *generátor*, ale priblíženie s listom je dostačujúce.

7.1 Príklady

Zväčši 2, Súčet menších, Počet najmenších

8 Vnorené cykly

Skúsme vytvoriť program, ktorý načíta číslo n a čísla od 0 po $n-1$ vypíše na 3 riadkoch za sebou. To nemôže byť ťažké, nie?

```
# Program vypise cisla od 0 po n - 1 ("n" je na vstupe) na 3 riadkoch za sebou.
n = int(input())
for i in range(n):
  print(f'␣{i}', end='')
print() # novy riadok

for i in range(n):
  print(f'␣{i}', end='')
print() # novy riadok

for i in range(n):
  print(f'␣{i}', end='')
print() # novy riadok

# Vstup:
# 3
# Vystup:
# 0 1 2
# 0 1 2
# 0 1 2
```

V tomto programe uvádzame nový konštrukt pre tvorbu reťazca `f'text {<premenna>} text {<premenna>}'`. Pomocou uvedenia reťazca so znakom `f` povolíte tzv. *interpoláciu* premenných pomocou kučeravých zátvoriek. V reťazci sa tak objaví hodnota premennej na mieste, kde bol jej názov v kučeravých zátvorkách. Napríklad `f'Toto je {meno}. Jeho obľúbené zviera je {zviera}.'`.

Príklad sme ale uvádzali aj z iného dôvodu. Asi už tušíte, že trikrát skopírovaný `for`-cyklus nevyzerá najlepšie. Ako by sme teda mohli zopakovať riadky viackrát? No predsa cyklom! V cykle, ktorý sa zopakuje 3-krát napíšeme cyklus, ktorý vypíše prvky od 0 po $n-1$.

```
# Program vypise cisla od 0 po n - 1 ("n" je na vstupe) na 3 riadkoch za sebou. Pouziva vnoreny cyklus.
n = int(input())
for i in range(3):
  for j in range(n):
    print(f'␣{j}', end='')
  print() # novy riadok

# Vstup:
# 3
# Vystup:
# 0 1 2
# 0 1 2
# 0 1 2
```

Dva cykly od seba v označení oddelíme tak, že cyklus opakujúci sa 3-krát nazveme *vonkajší* a cyklus v ňom nazveme *vnútorný*. Všimnite si, že vnútorný cyklus musí použiť inú premennú ako i, pretože inak by došlo ku kolízii mien a vonkajší cyklus by mohol byť ovplyvnený vnútorným. Vo vnútornom cykle sme preto použili názov j. Vždy, keď sa začne vykonávať telo vonkajšieho cyklu, spustí sa znovu vnútorný cyklus vypisujúci čísla od 0 po $n-1$.

8.1 Tabuľka násobilky

Predstavte si teraz, že chceme vytvoriť malú tabuľku násobilky do 4. Tabuľka násobilky je jednoduchá tabuľka, ktorá v ľubovoľnej bunke obsahuje vynásobené číslo riadku a stĺpca.

Teraz napíšeme program, ktorý vypíše telo takejto tabuľky. Ako by sme postupovali, keby sme chceli takúto tabuľku vypísať po riadkoch zľava doprava? V riadku s číslom 1 sa vystriedajú všetky hodnoty stĺpcov (1 až 4). Potom sa presunieme na ďalší riadok s číslom 2 a opäť sa vystriedame všetky hodnoty stĺpcov (1 až 4). Takto by sme pokračovali až do posledného riadku.

Všimnite si, že pre každý riadok prejdeme vždy všetky stĺpce. Môžeme si to predstaviť ako cyklus cez riadky od 1 do 4, v ktorom je navyše *vnorený* cyklus cez stĺpce od 1 do 4.

```
# Vypis tabulky nasobilky velkosti 5x5.
for i in range(1, 5):
  for j in range(1, 5):
    # Medzeru medzi cislami nevypisujeme pred prvym cislom v riadku
    if j > 1:
       print('␣', end='')
```

*	1	2	3	4
1	1	2	3	4
2	2	4	6	8
3	3	6	9	12
4	4	8	12	16

Tabuľka 1: Tabuľka malej násobilky do 4.

```
    print(i * j, end='')
  print() # novy riadok
```

Za zmienku ešte stojí spomenúť chovanie `break` a `continue` vo vnorených cykloch. Oba príkazy sa viažu na *najbližší uzatvárajúci* cyklus. To napríklad znamená, že z vnoreného cyklu sa nedá vyskočiť „úplne“ (mimo cyklov), ale iba do vonkajšieho cyklu.

```
# Program vypisuje dvojice cisel od 0 po 8 v poradi od najmensich
# dvojic az pokial prve cislo nie je delitelne 3 a zaroven druhe cislo
# nie je delitelne 2.
for i in range(8):
  for j in range(8):
    # Chyba! Z vonkajsieho cyklu nevyskocime!
    if i % 3 == 0 and j % 2 == 0:
      break
    # Vypise i a j oddelene medzerou, ekv. s f'{i} {j}'.
    print(i, j)

# Vystup:
# 0 0
# 0 1
# ...
# 0 7
# 1 0
# ...
# 3 1
# ! 3 2 sa nevypise, ale program pokracuje (chyba!)
# 4 0
# 4 1
# ...
```

V prípade, že by sme v príklade vyššie použili `continue` namiesto `break` by ďalšia vypísana dvojica bola po vynechanej (3 2) bola (3 3), pretože z vnútorného cyklu by sme nevyskočili úplne, ale iba preskočili výpis.

9 List listov

Keď už vieme vnárať cykly, asi vás neprekvapí, že vnárať sa dajú aj listy :) Napríklad tu je list listov, ktorý obsahuje informácie o domácich zvieratách jedného z vedúcich.

```
zvierata = [
  ["Bax", "pes", "ciernobiely"],
  ["Kralovna", "macka", "biela"],
  ["Fero", "kocur", "hnedobiely"]
]
```

Všimnite si, že list `zvierata` obsahuje 3 ďalšie listy. Každý z týchto 3 listov popisuje jedno zviera. List konkrétneho zvieraťa obsahuje jeho meno, typ a farbu.

Vnorené listy indexujeme rovnako ako jednorozmerné. Keď teda zaindexujeme do listu `zvierata`, dostaneme list.

```
# zvierata = ...z predzchadzajuceho prikladu...

print(zvierata[1])
# Vystup (list): ["Kralovna", "macka", "biela"]
```

Tento list môžeme opäť indexovať.

```
# zvierata = ...z predzchadzajuceho prikladu...

# Farba prveho (nulteho) zvierata
print(zvierata[0][2])
# Vystup: "ciernobiely"
```

Skúsme teraz napísať program, ktorý vypíše mená všetkých zvierat.

```
# zvierata = ...z predzchadzajuceho prikladu...

for zviera in zvierata:
  print(zviera[0])

# Vystup (skrateny na jeden riadok):
# Bax Kralovna Fero
```

Vo for cykle prejdeme cez vsetky `zvierata`. Premenná `zviera` bude preto postupne obsahovať polia `[„Bax“, „pes“, „ciernobiely“]` až `[„Fero“, „kocur“, „hnedobiely“]`. Z týchto polí potom vypisujeme 0-tú hodnotu, čo je meno.

Ako načítať *dvojrozmerné pole*[3] (tabuľku čísel)? Pomocou vnorených cyklov.

```
# Program dostane cisla m a n na jednom riadku, pocet riadkov a pocet stlpcov tabulky.
# Nasledne dostane jednotlive hodnoty tabulky.

rozmery = input().split()
# Trik: skratka za
# m = int(rozmery[0])
# n = int(rozmery[1])
m, n = int(rozmery[0]), int(rozmery[1])

tabulka = []
for i in range(m):
  riadokStr = input().split()
  riadokInt = []
  for x in riadokStr:
    riadokInt.append(int(x))
  tabulka.append(riadokInt)

print(tabulka)

# Vstup:
# 3 4
# 1 2 3 4
# 5 6 7 8
# 9 10 11 12
# Vystup:
# [
#  [1, 2, 3, 4],
#  [5, 6, 7, 8],
#  [9, 10, 11, 12]
# ]
```

9.1 Príklady

Rozdiel, Súčtová pyramída, Výmena, Otočenie

10 Funkcie

Predstavte si, že vo svojom programe potrebujete často urobiť to isté. Napríklad potrebujete na viacerých miestach zistiť, či je číslo párne a odpoveď vypísať. Samozrejme môžete použiť `if` a `print` na všetkých miestach kde danú vec potrebujete pričom by to vyzeralo nejak takto:

```
if x%2 == 0:
    print("Parne cislo.")
else:
    print("Neparne cislo.")
#
#
#
if a%2 == 0:
    print("Parne cislo.")
else:
    print("Neparne cislo.")
#
#
# a tak dalej
```

Toto síce docieli to čo chceme avšak za následok to bude mať aj dlhší, neprehľadnejší kód s viac miestami kde sa môžeme pomýliť. Nezúfajte, zachránia nás funkcie. Horný program môžeme ekvivalentne zapísať aj krajšie:

```
def vypisParitu(cislo):
    if cislo%2 == 0:
        print("Parne cislo.")
    else:
        print("Neparne cislo.")
#
```

[3] Dvojrozmerné pole môže znieť odstrašujúco, ale s tým, čo už vieme, je vysvetlenie hračka. Je to jednoducho list listov. Pole je list a dvojrozmerné sa volá preto, lebo je to v podstate tabuľka, v ktorej môžeme chodiť po riadkoch aj stĺpcoch.

```
#
vypisParitu(x)
#
#
vypisParitu(a)
#
# a tak dalej
```

Ak si predchádzajúce dva príklady porovnáte, malo by vám byť intuitívne jasné, čo funkcie robia. S funkciami sme sa už stretli kedže `print` je príklad funkcie. Čo je teda funkcia? Je to viac príkazov pospájaných do jedného celku, tvoriac akýsi superpríkaz vykonávajúci nejakú vec. Pamätajte, funkcia môže mať aj niekoľko desiatok riadkov pokiaľ toho robí viac alebo je zložitejšia. Prvá vec, ktorú máme možnosť vidieť v príklade vyššie je, že funkcia má parameter/parametre. Parametre sa uvádzajú do zátvoriek za funkciou a ide o cestu, ako funkcii odovzdať nejakú informáciu. V prípade vyššie má funkcia *vypisParitu* jeden parameter s názvom cislo. Tento parameter má táto funkcia preto, že jej nejak chceme odovzdať číslo, ktoré potrebuje k rozhodnutiu sa, čo vypísať. Keď chceme funkciu použiť, hovoríme, že ju voláme a teda v programe vidíme dve volania funkcie *vypisParitu*. Pri volaní funkcii odovzdáme naše číslo ako parameter.

Okrem toho, že si funkcie berú parametre, vedia niečo aj vracať. Funkcia vyššie nič nevracia. Ak by sme ale chceli vidieť príklad toho, ako môže funkcia niečo vracať, môže to vyzerať takto.

```
def scitaj(a, b):
    return a+b
# pouzitie:
x, y = 10, 20
vysledok1 = scitaj(x, y)
vysledok2 = scitaj(x, 77)
vysledok3 = scitaj(5, 11)
```

Vyššie vidíme veľmi jednoduchú funkciu, ktorá iba sčíta svoje parametre a vráti ich pomocou kľúčového slova `return` a potom 3 rôzne korektné použitia tejto funkcie.

Jedna z dôležitých vecí je, ako funguje premenná potom, čo sa stane parametrom. Pokiaľ napíšeme jednoduchý program.

```
def zmen(x):
    x = 30
a = 10
print(a)
zmen(a)
print(a)
```

Po jeho spustení zistíme, že sa hodnota premennej a po zmene vo funkcii `zmen` nezmenila. Je to spôsobené tým, že do funkcie nebola poslaná priamo premenná a ale len jej kópia, ktorá sa po konci vykonávania funkcie zahodí. Veci sa ale trocha skomplikujú keď spustíme nasledujúci program.

```
def zmen(x):
    x[0] = 30
a = [1, 2, 3]
print(a)
zmen(a)
print(a)
```

Čakali by sme, že sa a nezmení avšak tentokrát je a `list`. Prečo sa v tomto prípade zmenila hodnota a? Je to z dôvodu, že do funkcie bola v tomto prípade poslaná nie kópia listu ale priamo list a. Je trochu nelogické, prečo sa python správa rôzne pri rôznych typoch. Je to napríklad z dôvôdu, že listy sú zväčša pomerne veľké, zaberajú viac miesta a teda aj ich kopírovanie je pomalé a Python sa mu snaží vyhnúť. Je dôležité si na toto správanie dávať pozor a keď je to dôležité, vyskúšať ako sa Python správa pre konkrétny typ.

10.1 Príklady

Tlačítka

11 Záver

Super! Ak si to dotiahol až sem a popri tom urobil všetky úlohy tak máš solídne základy v jazyku Python. Veríme, že tvoja programátorská cesta tu nekončí. Čaká ťa totiž veľa zaujimávych príležitostí, kde svoju znalosť jazyka Python môžeš ďalej prehĺbiť. Ďakujeme, že si čítal náš tutoriál a najväčším poďakovaním bude ak nám povieš, čo v ňom môžeme zlepšiť aby sa ďalším čitateľom čítal jednoduchšie. Kontaktovať nás môžeš buď na Discorde pod nickom Aj0SK a sobkulir alebo na adresách andrejkorman@gmail.com a r.sobkuliak@gmail.com. Tiež nás kontaktuj ak nevieš kam s tvojimi znalosťami ďalej :)