Učebnica Pythonu pre stredné školy

- 1. Zoznámme sa s jazykom Python (01.html)
- 2. Python ako kalkulačka, premenné, priradenie (02.html)
- 3. Prvý program, výpisy, výpočty (03.html)
- 4. Grafická plocha, obdĺžniky (04.html)
- 5. Farby pri kreslení obdĺžnikov (05.html)
- 6. Použitie premenných a výrazov pri kreslení (06.html)
- 7. Vytvárame podprogramy (07.html)
- 8. Náhoda (08.html)
- 9. Text v grafickej ploche (09.html)
- 10. Fonty a farby pre texty v grafike (10.html)
- 11. Program s opakovaním (11.html)
- 12. Riešenie úloh s for-cyklom (12.html)
- 13. Kreslenie elíps a kruhov (13.html)
- 14. Kruhy a cykly (14.html)
- 15. Parameter podprogramu (15.html)
- 16. Parametre podprogramov (16.html)
- 17. Premenné typu znakové reťazce, operácie, prevody (17.html)
- 18. Premenné typu desatinné čísla (float), operácie (18.html)

1. Zoznámme sa s jazykom Python

Čo je to programovanie

Informatika je vedná disciplína … algoritmické riešenie problémov … veda, ktorá sa zaoberá algoritmami (rôznymi postupmi riešenia problémov najčastejšie pomocou počítača)

Samotné programovanie je tá časť informatiky, ktorá sa zaoberá zápisom algoritmov do takej formy/jazyka, ktorému „rozumie“ počítač. Počítač môže rozumieť rôznym programovacím jazykom, ale vždy sú to len technické zápisy, ktoré sa dosť líšia od prirodzenej reči.

Aby sme sa naučili programovať (riešiť algoritmické problémy zápisom ich riešenia v niektorom konkrétmon jazyku), mali by sme

- sa naučiť **algoritmicky myslieť** - t.j. zapisovať algoritmy len pomocou povolených konštrukcií, prípadne pomocou správne zostavených kombinácií (napr. podmienené vykonanie alebo opakované vykonanie nejakých časti algoritmu)
- spoznať **syntax jazyka** - t.j. pravidlá zápisu konkrétneho programovacieho jazyka
- sa naučiť pracovať s rôznymi **údajovými typmi** - napr. ako sa pracuje s číslami a textami, prípadne rôzne veľkými skupinami údajov, ktoré sú uložené v tzv. **dátových štruktúrach**

Programovanie teda bude pre nás označovať schopnosť zostaviť algoritmus, zapísať ho do programovacieho jazyka len pomocou povolených konštrukcií a údajových typov.

Jazyk Python

Každý počítač, bez ohľadu na to, či je to netebook, stolný alebo veľký sálový počítač, alebo tablet či mobilný telefón, prípadne počítač v modernejších autách, chladničkách, bankomatoch, herných konzolách a pod. na najnižej úrovni „rozumie“ len svojmu konkrétnemu **strojovému jazyku**. V takomto jazyku by sa nielen nám

veľmi zle programovalo, ale aj veľmi skúseným profesionálnym programátorom. Z tohto dôvodu sa vymysleli **vyššie programovacie jazyky**. Takémuto jazyku bude počítač rozumieť, len ak bude mať k dispozícii prekladač, resp. interpreter.

Počítač potom bode rozumieť našim algoritmom, ktoré sú zapísané v tomto konkrétnom jazyku. Ak počítaču vložíme viac rôznych prekladačov, pri riešení úlohy sa môžeme rozhodnúť, ktorý z nich použijeme - ale musíme to počítaču jednoznačne označiť.

Predstavme si to tak, že „materinským jazykom“ počítača je taká hatlatinka, v ktorej sa nik okrem samotných počítačov nebaví. Ak chceme do počítača zadávať naše riešenia algoritmických problémov, mal by už poznať tento náš jazyk zápisu … mal by sa naučiť aspoň jeden nejaký cudzí jazyk, v ktorom sa s ním zvládneme „rozprávať“ (teda treba mu dodať prekladač).

Táto učebnica je postavená na programovacom jazyku **Python** (čítame „pajton“), ktorého prekladač sa už musí nachádzať v našom počítači (je už nainštalovaný - inštrukcie sú v prílohe). Iné učebnice možno učia úvod do programovania v iných jazykoch, my sme tu zvolili práve Python, lebo

- jeho prekladač je voľne stiahnuteľný pre ľubovoľný typ operačného systému (MS Windows, Linux, Mac OS)
- jeho základné konštrukcie majú veľmi jednoduchý a dobre čitateľný zápis, preto sa odporúča ako jeden z najvhodnejších jazykov na zápis algoritmov najmä pre úvod do programovania
- napriek tomu, že má už skoro 30 rokov, je to moderný jazyk so všetkými dôležitými konštrukciami a štruktúrami, podporuje procedurálne, objektovo orientované aj funkcionálne programovanie, dajú sa v ňom zapisovať aplikácie napr. pre prácu s databázami, grafikou a udalosťami, internetom a webom, pre umelú inteligenciu, …
- v súčasnosti je to jeden z najpopulárnejších programovacích jazykov, ktorý sa používa na väčšine vysokých škôl vo svete, používajú ho aj najväčie softvérové firmy, ale je obľúbený medzi bežnými výskumníkmi aj z neprogramátorských oblastí

2. Python ako kalkulačka, premenné, priradenie

Interaktívny režim

Budeme využívať základné **vývojové prostredie** na prácu s Pythonom, tzv. **IDLE**. Po jeho spustení sa otvorí textové okno s takýmto výpisom:

```
Python 3.6.4 (v3.6.4:d48eceb, Dec 19 2017, 06:04:45) [MSC v.1900 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
```

V tomto textovom okne sa budeme „rozprávať" s počítačom v „pythončine", teda my sem budeme písať text v jazyku Python a prekladač to preloží počítaču do jazyka, ktorému už rozumie. Text musíme zadať úplne správne, bez chýb, za znaky `>>>` . Keď zapíšeme správny matematický výraz, dozvieme sa aj odpoveď. Napr.

```
>>> 1 + 2 + 3
6
>>> 123
123
>>> 42 - 17
25
```

Vidíme, že Python tu „porozumel" matematickému zápisu. Matematické výrazy teda môžu obsahovať čísla a operácie `+` a `-` . Môžete experimentovať aj s ďalšími matematickými operáciami: znamienko `*` (hviezdička sa používa na násobenie) a okrúhlymi zátvorkami vieme určiť poradie vyhodnocovania operácií v takomto výraze. Vyskúšajte napríklad:

```
>>> 3 + 4 * 5
23
>>> (3 + 4) * 5
35
>>> 25 - 7 - 10
8
>>> 25 - (7 - 10)
28
```

Takáto Pythonová kalkulačka dokáže aj deliť pomocou operácie `/` (lomka), napr.

```
>>> 132 / 11
12.0
>>> 1 / 2
0.5
>>> 1 + 2 / 3
1.6666666666666665
```

Vo výsledkoch výrazov, v ktorých sa nachádza delenie, vidíme čísla s desatinnou bodkou (na rozdiel od desatinnej čiarky v matematike).

Matematické zápisy musia byť zapísané úplne správne. Aj najmenšia chyba spôsobí upozornenie na chybu. Napríklad:

```
>>> 22 + 7 *
SyntaxError: invalid syntax
>>> 19 - (3 4)
SyntaxError: invalid syntax
>>> 10 / 0
Traceback (most recent call last):
  File "<pyshell#48>", line 1, in <module>
    10 / 0
ZeroDivisionError: division by zero
```

Takýmito upozorneniami sa nám Python snaží pomôcť, aby sme mohli lepšie pochopiť, aká je to chyba a mali šancu ju opraviť. Napr. `SyntaxError: invalid syntax` označuje, že aritmetický výraz nie je zapísaný správne, pravdepodobne v ňom chýba znamienko operácie, operand (napr. číslo) alebo zátvorka. Chybová správa `ZeroDivisionError: division by zero` oznamuje, že sme sa snažili deliť nulou.

Premenné

Počítač umožňuje nielen vypočítať nejaké matematické zápisy (tzv. aritmetické výrazy), ale umožňuje si niektoré vypočítané hodnoty dočasne zapamätať. Podobne, aj niektoré kalkulačky majú tzv. pamäť, do nej si vedia uložiť jednu alebo aj viac hodnôt a tie neskôr použiť v ďalších výpočtoch. Pri práci s Pythonom môžeme vytvoriť ľubovoľný počet takýchto „pamätí“ a každá si môže pamätať jednu hodnotu. V programovaní sa takýmto pamätiam hovorí **premenná**.

Premennou budeme nazývať zapamätanú hodnotu, ktorú musíme pomenovať nejakým **menom**. Vďaka tomuto menu sa ľahko dostaneme k tejto zapamätanej hodnote. Premennú si môžeme predstaviť ako škatuľku, v ktorej sa nachádza nejaká hodnota a spredu škatuľky je napísané jej meno. Podľa tohto mena vieme hocikedy škatuľku vyhľadať a zistiť, aká sa v nej nachádza hodnota.

Novú premennú vytvoríme veľmi jednoducho: použijeme na to špeciálny **príkaz**, budeme mu hovoriť **priraďovací príkaz**. Jeho zápis sa trochu podobá matematickej rovnosti. Napr. potrebujeme vytvoriť **premennú**, v ktorej budeme uchovávať nameranú výšku nejakého žiaka. Nazveme takúto premennú **menom**, napr. `vyska` a zapíšeme:

```
>>> vyska = 167
```

Týmto zápsom vznikne nová premenná (škatuľka) s menom `vyska` , ktorá má hodnotu `167` . Ak teraz zapíšeme:

```
>>> cena = 22 + 7
```

Opäť vzniká nová premenná, teraz s menom `cena` , ktorej hodnota je výraz za znakom `=` , teda `29` . Teraz ešte tretie priradenie:

```
>>> qb7rsh = 4788
```

Tento príkaz predvádza, že menami premenných môžu byť aj slová, ktoré nemajú nejaký zmysel. Ak premennú pomenujeme nič nehovoriacim menom, napríklad `qb7rsh`, vznikne väčšie riziko, že si ho horšie zapamätáme, prípadne zabudneme, čo za hodnotu sme sem uložili. Preto programátori zvyknú používať také mená, ktoré sa ľahko zapamätajú a ľahšie z nich môžeme pochopiť, aký typ hodnoty je v nej uložený.

V pamäti to teraz vyzerá nejako takto:

Priraďovacím príkazom budeme teda rozumieť taký zápis, v ktorom sa pred znakom `=` nachádza nejaké meno premennej a za znakom `=` je hodnota, ktorú treba do tejto premennej uložiť. Ak je hodnotou aritmetický výraz, tak sa najprv vyhodnotí a až potom priradí do premennej. Mohli by sme to zapísať takto:

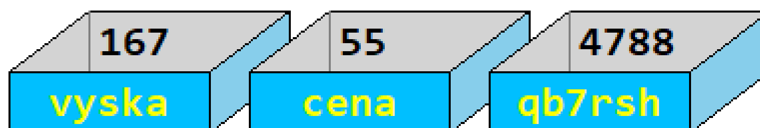
```
>>> meno_premennej = hodnota_výrazu
```

Týmto zápisom vznikne nová premenná (škatuľka) s udaným menom a s danou hodnotou.

Ak teraz zapíšeme priraďovací príkaz s premennou `cena`:

```
>>> cena = 5 * 11
```

nevytvorí sa nová premenná, predsa premenná s týmto menom už existuje, ale existujúca premenná získava novú hodnotu. Namiesto pôvodnej hodnoty `29` sa zapamätá hodnota `55`. Momentálny stav pamäti by sme mohli zakresliť takto:



Premenné už vieme vytvárať, dokonca aj im meniť zapamätanú hodnotu, ale ešte potrebujeme vedieť zistiť **hodnotu premennej**. Našťastie zisťovanie hodnoty je veľmi jednoduché: stačí zapísať jej meno a Python nám okamžite oznámi jej hodnotu. Napr.

```
>>> vyska
167
>>> cena
55
>>> vek
Traceback (most recent call last):
  File "<pyshell#0>", line 1, in <module>
    vek
NameError: name 'vek' is not defined
```

V poslednom riadku sme sa snažili zistiť hodnotu premennej `vek`. Lenže takáto premenná zatiaľ neexistuje, preto nám to Python oznámil textom **NameError: name ‚vek' is not defined** (z výpisu si budeme všímať hlavne posledný riadok), kedže premenná ‚vek' nie je definovaná.

Meno premennej môžeme použiť aj v ďalších matematických zápisoch a Python nám namiesto mena dosadí jej hodnotu. Napr.

```
>>> 190 - vyska
23
>>> 13 + qb7rsh / cena
100.05454545454545
```

Mená premenných

Neskôr budeme vytvárať programy, ktoré budú pracovať s viacerými premennými. Mená týchto premenných môžeme zvoliť skoro ľubovoľne, ale správny informatik ich pomenuje tak, aby sa v neskorších zápisoch výrazov lepšie vyznal. Mená premenných môžeme skladať z malých písmen abecedy (najlepšie z anglickej abecedy) a

číslic od `0` do `9`. Môžeme použiť aj znak podčiarkovník `_`. Platia len dve obmedzenia: meno premennej nesmie začínať číslicou a musí byť rôzne od špeciálnych Pythonových rezervovaných mien (ako napr. slová `def`, `for`, `and`, atď). Ak si zvolíme nesprávne meno premennej, Python nás na to upozorní správou:

```
>>> def = 123
SyntaxError: invalid syntax
>>> 2cena = 0
SyntaxError: invalid syntax
```

Prvý priraďovací príkaz sa snaží priradiť do premennej `def`. Lenže slovo `def` je rezervované meno pre Pythonvský príkaz, takže sa nesmie použiť ako meno premennej. Všetky rezervované slová sa kvôli tomuto učiť nemusíte, Python vás na ne upozorní. Druhý priraďovací príkaz má chybné meno premennej, lebo takéto meno nikdy nesmie začínať číslicou.

Vhodnými menami sú, napr.

```
>>> strana_stvorca = 150
>>> obvod = 4 * strana_stvorca
>>> obsah = strana_stvorca * strana_stvorca
```

Všimnite si, že vďaka dobre zvoleným menám, vieme dosť presne určiť, že tieto priraďovacie príkazy pre danú veľkosť štvorca (v premennej `strana_stvorca`) vypočítajú jeho obvod (v premennej `obvod`) a jeho obsah (v premennej `obsah`). Z matematiky sme zvyknutí niektoré mená skracovať, napr. veľkosť strany štvorca môže byť `a`, resp. obdĺžnik má strany `a a b`. Predchádzajúci príklad, ktorý počíta obvod a obsah štvorca, môžeme dostatočne čitateľne zapísať aj takto:

```
>>> a = 150
>>> obvod = 4 * a
>>> obsah = a * a
```

Úlohy

1. Zapíšte do tabuľky všetky aritmetické operácie, s ktorými ste sa zatiaľ zoznámili.
2. Miro mal pred dvomi mesiacmi 16 rokov. Spočítajte koľko je to približne dní. Zjednodušte si výpočet tak, že budete predpokladať, že rok má 365 dní a mesiac má 30 dní. Koľko je to hodín a koľko sekúnd?
3. Zapíšte súčet všetkých nepárnych čísel od 1 do 19.
4. Zistite, ktorá cifra sa vyskytuje najčastejšie vo výsledku výrazu:

   ```
   123456789 * 111111111111111111111
   ```

   Toto zisťovanie musíte zatiaľ robiť ručne. Python by to dokázal zistiť tiež, ale tento zápis by bol komplikovanejší.
5. Vypočítajte súčet takýchto čísel: jedna, jedna polovica, jedna tretina, jedna štvrtina, ..., až jedna desatina.
6. Do premennej `pocet` priraďte počet kusov, do premennej `cena1` priraďte cenu za jeden kus, do premennej `vysledna_cena` zapíšte celkovú sumu, ktorú bude treba zaplatiť za daný počet kusov. Skontrolujte zápis pre `cena1 = 15 a pocet = 7`
7. V premenných `dlzka` , `sirka a hlbka` máme priradené rozmery školského bazénu v centimetroch. Vypočítajte, koľko litrov vody treba na napustenie celého bazéna. Zistite, koľko je to kubických metrov. Počítajte napríklad s hodnotami `dlzka=2500` , `sirka=1000` , `hlbka=180` .
8. Zapíšte takýto výpočet: v premennej `x` máme priradenú nejakú hodnotu, potrebujeme vypočítať: k `x` pripočítaj `1` a výsledok vynásob `2` , opäť k výsledku pripočítaj `1` a vynásob `2` a do tretice opäť k výsledku pripočítaj `1` a vynásob `2` . Napr. pre `x` rovné `5` , by si mal dostať výsledok `54` .
9. V matematike sa počíta **faktoriál** nejakého čísla `n` ako súčin čísel od `1` do `n` . Napr. `4` faktoriál je súčin čísel `1 * 2 * 3 * 4` . Do premennej `faktorial10` priraďte hodnotu `10` faktoriál. Vypočítate ho ako súčin čísel od 1 do 10.

Úlohy pre pokročilých

10. Výpočet `2 * 2 * 2 * 2 * 2 * 2 * 2 * 2 * 2 * 2` vieme zapísať ako umocnenie 2 na 10, teda v Pythone `2 ** 10` . Vypočítajte 2 umocnené na 30 a zistite počet cifier tohto výsledku. Cifry musíme zatiaľ spočítať ručne.

11. Zistite, ako sa počíta hodnota `2**8-1`. Teda, či sa najprv vypočíta mocnina `2**8`, od ktorej sa odpočíta `1`, alebo sa najprv vypočíta rozdiel `8-1` a touto hodnotou sa potom počíta mocnina `2`. Zistite, ako je to s operáciami násobenia a umocňovania: teda, ako sa počítajú výrazy `3*2**5` a `2**5*3`.
12. Matematici vedia, že keď spočítajú niekoľko za sebou idúcich mocnín `2`, dostanú inú druhú mocninu zmenšenú o `1`. Skontrolujte, či napr. súčet čísel `2**0`, `2**1`, `2**2`, ... `2**9`, dáva hodnotu `2**10-1`.

Zhrnutie

čo sme sa naučili:

- interaktívny režim - režim, v ktorom Pythonu zadávame príkazy (za znaky `>>>`) a ten ich vyhodnotí, prípadne vypíše výsledok výpočtu
- premenná - pomenovaná zapamätaná hodnota, premenné vznikajú pomocou priraďovacieho príkazu
- priradenie - príkaz, ktorým sa nastaví nejaká hodnota do premennej
- poznáme tieto číselné operácie: `+` (sčitovanie), `-` (odčitovanie), `*` (násobenie), `/` (delenie)

dôležité zásady:

- je vhodné voliť dobré mená premenných tak, aby popisovali hodnotu, ktorú uchovávajú
- na mená premenných využívame malé písmená abecedy, číslice a znaky `_` (podčiarkovník)
- zapisujeme čitateľný kód, napr. vkladáme medzery medzi aritmetickými operáciami aj okolo znaku `=` pre priradenie

3. Prvý program, výpisy, výpočty

Zopakujme si:

- s Pythonom pracujeme v interaktívnom režime
- premenné vytvárame pomocou priraďovacích príkazov
- vieme pracovať s číslami a výrazmi, ktoré používajú základné operácie `+`, `-`, `*`, `/` a okrúhle zátvorky

V tejto téme sa naučíme ukladať naše výpočty, presnejšie postupy do textových súborov.

Výpočet ukladáme do súboru

Zatiaľ sme s Pythonom pracovali **v interaktívnom režime**: za symboly `>>>` sme zapísali buď nejaký výpočet alebo priraďovací príkaz a Python to okamžite vyhodnotil a znovu vypísal výzvu `>>>`. Tým nám dal najavo, že čaká ďalšie príkazy.

Častejšie ale budeme zostavovať príkazy, ktoré sa majú postupne vykonať, do textového súboru a až potom ich necháme vykonať. Takýmto textovým súborom budeme hovoriť **programy**. V Pythone sa im niekedy hovorí aj **skripty**. Takéto programy môžeme zostavovať skoro v ľubovoľnom textovom editore, ale pohodlnejšie to bude priamo v Pythonovskom editore, ktorý je súčasťou **IDLE**.

Textový editor na prípravu a opravovanie pythonovských programov otvoríme pomocou klávesovej skratky `<Ctrl+N>` (alebo pomocou voľby **New File** v menu **File**). Otvorí sa prázdne textové okno, do ktorého budeme zapisovať samotný program. Napr. môžeme sem zapísať:

```
# moj prvy program

strana_stvorca = 55
obvod = 4 * strana_stvorca
obsah = strana_stvorca * strana_stvorca
```

Tento program okrem troch známych príkazov priradenia obsahuje aj špeciálny riadok, ktorý začína znakom `#` (krížik). Znak `#` má pre Python špeciálnu funkciu: zvyšok riadku za týmto znakom sa bude Pythonom ignorovať (Pythonu je jedno, čo je tu zapísané). Tomuto hovoríme **komentár** a asi je to zaujímavejšia informácia pre nás čitateľov ako pre počítač. My budem vkladať komentáre do programu preto, aby sme takto buď niečo vysvetlili čitateľovi alebo pripomenuli.

V textovom editore sme teraz pripravili náš prvý niekoľkoriadkový program, ktorý slúži na to, aby ho Python vyhodnotil, teda vykonal jeho všetky príkazy. Procesu vyhodnocovania, resp. vykonávania sa hovorí **spustenie programu** (po anglicky **Run**). Lenže, aby sa dal program spustiť, musí byť uložený v textovom súbore (teraz sa nachádza len v okne textového editora). Ukladanie do súboru sa robí podobne ako v iných aplikáciach, teda pomocou klávesovej skratky `<Ctrl+S>` (alebo výberom voľby **Save** v menu **File**). Po uložení programu do súboru (nazvime ho napr. `prvy`, tento dostáva príponu `.py`) ho môžeme konečne spustiť. Slúži na to klávesová skratka `<F5>` (alebo voľba **Run Module** v menu **Run**).

Všimnite si, čo sa stalo po spustení programu:

1. Python sa prepol späť do interaktívneho okna (tzv. **Shell**)
2. Vypísal sa riadok:

   ```
   ====================== RESTART ======================
   ```

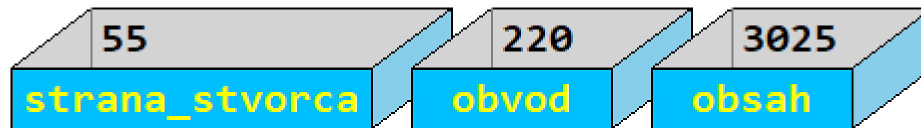
   Toto označuje, že Python sa pripravil na spustenie nášho programu tým, že zrušil všetky naše doterajšie výpočty, teda všetky naše premenné.

3. Spustil náš program: postupne vykonal riadok za riadkom (teda naše tri priradenia)
4. Po skončení vykonania všetkých príkazov Python vypíše známe tri znaky `>>>`, ktorými dáva najavo, že môžeme pokračovať v interaktívnom zadávaní príkazov.

Keďže náš program nevypisoval žiadne hodnoty - robil len priradenia nejakých hodnôt do premenných, jediný spôsob, ako sa presvedčíme, že naozaj niečo počítal, je zistenie hodnôt premenných, do ktorých sa niečo priradilo:

```
>>> strana_stvorca
55
>>> obvod
220
>>> obsah
3025
```

Naozaj teraz vidíme, že náš program s výpočtom obvodu a obsahu štvorca s danou veľkosťou strany 55 prebehol a aj správne vypočítal zadané vzorce. V pamäti to teraz vyzerá nejako takto:



Keď potrebujeme v interaktívnom režime zistiť obsah nejakej premennej, prípadne zistiť hodnotu nejakého výpočtu, tak zapisujeme:

```
>>> obvod
220
>>> obsah + 75
3100
```

A hneď sa dozvedáme výsledok, totiž **shell** po zadaní výrazu, ho automaticky vyhodnotí a vypíše jeho hodnotu. Žiaľ, takýto mechanizmus pre spúšťanie programov nefunguje. Ak by sme opravili náš program:

```
# moj prvy program

strana_stvorca = 55
obvod = 4 * strana_stvorca
obvod
obsah = strana_stvorca * strana_stvorca
obsah + 75
```

Teda po výpočte hodnôt premenných `obsah` a `obvod` by sme ich chceli vypísať podobne, ako sme to robili v interaktívnom režime. Lenže takto zapísaný program po (uložení a) spustení robí presne to isté, ako predtým: teda nič nevypisuje. Zapamätajme si túto vlastnosť vyhodnocovania pythonovských programov:

- Python postupne vyhodnocuje program riadok za riadkom
- ak je príkazom priradenie, vykoná ho, teda priradí vypočítanú hodnotu do zadanej premennej
- ak riadok programu obsahuje len **výpočet bez priradenia**, Python tento výpočet vyhodnotí (zistí hodnotu premennej `obsah`), ale keďže sme túto hodnotu nikam nepriradili, tak ju **zahodí**

Nám sa to javí tak, že riadky programu, ktoré obsahujú len výrazy bez priradenia, **nerobia nič**. My už teraz vieme, že takto sa hodnoty z programu vypisovať nedajú. Na toto nám bude slúžiť špeciálny príkaz `print`.

Výpis hodnôt

Skôr ako použijeme tento príkaz v programe `prvy.py`, budeme ho testovať v interaktívnom režime. Zapíšme:

```
>>> print(obvod)
220
>>> print(obsah + 75)
3100
```

Príkaz vypisuje presne rovnaké výsledky, ako sme dosiahli bez použitia `print`. Okrem slova **print** musíme vypisovanú hodnotu (premennej alebo výrazu) uzavrieť do okrúhlych zátvoriek.

Opravme teraz náš prvý program tak, aby po spustení vypísal obvod, obsah a teraz aj zväčšený obsah o 75:

```
# moj prvy program

strana_stvorca = 55
obvod = 4 * strana_stvorca
print(obvod)
obsah = strana_stvorca * strana_stvorca
print(obsah)
print(obsah + 75)
```

Program teraz naozaj vypíše tri očakávané čísla:

```
====================== RESTART ======================
220
3025
3100
>>>
```

Uvedomte si, že **program**, ktorý sme uložili do súboru a **spustili**, Python **vykonáva** príkaz za príkazom, pritom:

- komentáre (začínajúce znakom `#`) **ignoruje**
- aritmetické výrazy **vyhodnotí** a ak sú súčasťou priraďovacieho príkazu, tak túto hodnotu vloží, **priradí** do premennej
- výrazy (aj premenné), ktoré sa nachádzajú v príkaze `print` tiež **vyhodnotí** a hodnoty **vypíše** do textového okna Shell
- výrazy (aj premenné), ktoré nie sú súčasťou priraďovacieho príkazu a nie sú ani súčasťou príkazu `print` sa tiež **vyhodnotia**, ale tento výsledok sa **zahodí**

Vypisovanej hodnote (výraz alebo premenná), ktorú sme uzatvárali do zátvoriek, hovoríme **parameter** príkazu `print`. V skutočnosti môže mať tento príkaz viac parametrov a vtedy ich medzi zátvorky oddelíme čiarkami, napr.

```
>>> print(2, 2+1, 10/2, 2*3+1)
2 3 5.0 7
>>> print(strana_stvorca, obvod, obsah)
55 220 3025
```

Ako parameter môžeme uviesť aj **text** uzavretý medzi apostrofy (alebo úvodzovky). Hovoríme mu **znakový reťazec**. Môžeme zapísať napr.

```
>>> print('Pozdravujem vas, lesy, hory!')
Pozdravujem vas, lesy, hory!
>>> print('obvod stvorca so stranou', strana_stvorca, 'je', obvod)
obvod stvorca so stranou 55 je 220
>>> print('6 faktorial =', 1*2*3*4*5*6)
6 faktorial = 720
```

Vďaka takémuto použitiu príkazu `print` dokážeme vylepšiť náš prvý program:

```
# moj prvy program

strana_stvorca = 55
print('stvorec so stranou', strana_stvorca)
obvod = 4 * strana_stvorca
print('ma obvod', obvod)
obsah = strana_stvorca * strana_stvorca
print('ma obsah', obsah)
```

Teraz už bude spustenie nášho programu vypisovať:

```
stvorec so stranou 55
ma obvod 220
ma obsah 3025
```

Chyby v programe

Každý, kto programuje, sa stretá aj s chybami. Už ste sa zoznámili so **syntaktickými chybami**, pri ktorých nás Python upozorňuje na to, že v zápise príkazu nie je niečo v poriadku. Napr.

```
>>> 19 - (3 4)
SyntaxError: invalid syntax
>>> 2cena = 0
SyntaxError: invalid syntax
```

V prvom zápise pravdepodobne chýba nejaká operácia v zátvorkách. Možno sme chceli zapísať `19 - (3 + 4)`. V druhom zápise priraďovacieho príkazu sme zvolili chybné meno premennej: meno premennej nesmie začínať číslicou. Správou `SyntaxError` nám Python oznamuje, že ďalej sa nedá pokračovať, kým si neopravíme chybu.

Ak máme syntaktickú chybu v programe, ktorý je zapísaný v súbore, takýto program nespustíme (pomocou **Run**), kým chybu neopravíme.

Už ste videli napr. aj takúto chybovú správu:

```
>>> vek
Traceback (most recent call last):
  File "<pyshell#0>", line 1, in <module>
    vek
NameError: name 'vek' is not defined
```

Tato chyba nemusí označovať, že sme sa pomýlili v zápise. Python nám tu neoznamuje syntaktickú chybu, ale hoci takýto zápis vyhovuje jeho pravidlám zápisu (syntax jazyka), pri snahe o jeho vykonanie sa Pythonu niečo „nepáči". V tomto prípade je to použitie mena premennej, napriek tomu, že ešte neexistuje, teda, že sme do nej ešte nič nepriradili. Zrejme takúto premennú treba najprv vytvoriť (priradením).

Podobnú chybu dostaneme aj pri:

```
>>> strana_stvorca = 55
>>> obvod = 4 * strana_stvrca
Traceback (most recent call last):
  File "<pyshell#11>", line 1, in <module>
    obvod = 4 * strana_stvrca
NameError: name 'strana_stvrca' is not defined
```

kde asi Python neočakáva, že vytvoríme premennú s menom `strana_stvrca`, ale opravíme preklep v mene tejto premennej. Chyby, ktoré nie sú syntaktické, ale Python takýto príkaz nevie vykonať, nazývame **sémantické chyby**. Ak sa sémantická chyba vyskytne v programe, takýto program môžeme spustiť, ale pri vykonávaní chybného príkazu **program spadne** s chybovou správou. Zrejme môžeme chybu opraviť (ak vieme, čo sa stalo) a program spustiť znovu.

Najnáročnejšia situácia je s tretím typom chýb, sú to tzv. **logické chyby**. Program, ktorý vyzerá, že je správny, niekedy môže dať nesprávne výsledky. Problémom samozrejme nie je chybné správanie Pythonu, ale skrytá logická chyba. Takéto chyby sa ťažšie odhaľujú a niekedy aj ťažko opravujú. Často si to vyžaduje väčšie programátorské skúsenosti.

Pozrite toto riešenie programu na výpočet obvodu a obsahu štvorca:

```
# moj vylepseny prvy program

a = 55
print('stvorec so stranou', a)
obvod = 55 + 3 * a
print('ma obvod', obvod)
obsah = obvod + 51 * a
print('ma obsah', obsah)
```

Úmyselne sme tu použili nejaké magické vzorce, ale po spustení to funguje dobre:

```
stvorec so stranou 55
ma obvod 220
ma obsah 3025
```

Lenže, ak v programe zmeníme veľkosť strany štvorca napr. na 45:

```
# moj vylepseny prvy program

a = 45
print('stvorec so stranou', a)
obvod = 55 + 3 * a
print('ma obvod', obvod)
obsah = obvod + 51 * a
print('ma obsah', obsah)
```

dostaneme výsledky, ktoré sa niekomu môžu zdať správne:

```
stvorec so stranou 45
ma obvod 190
ma obsah 2485
```

Ale výpis dáva úplne zlé hodnoty (obvod mal byť 180 a obsah 2025). Takže si zapamätajte, že ak vám program dá pre nejaké hodnoty správne výsledky, nemusí to znamenať, že správne hodnoty dostaneme aj pre iné vstupné údaje.

Uvedomte si, že chybovými správami sa nám Python snaží pomôcť, aby sme našli chyby v našich programoch a mohli ich čo najlepšie opraviť. Čím budete viac programovať, čím sa budete častejšie stretať s rôznymi chybovými správami, tým rýchlejšie ich budete vedieť odhaľovať a opravovať. Treba si zapamätať, že program, ktorý prejde bez chybovej správy, nemusí znamenať, že je bezchybný. Často budú naše prvé programy obsahovať **logické chyby**, pri ktorých nám Python nepomôže žiadnou chybovou správou. Hľadať chyby v takýchto programoch a potom ich aj opravovať je veľmi náročný proces, ktorý si bude vyžadovať veľa trpezlivosti a hlavne získavanie programátorských skúseností.

Úlohy

1. Napíšte program, ktorý vypíše návrh vašej biznisovej vizitky. Výpis môže vyzerať napr.

```
+--------------------+
|             ~~~     |
|  Juraj    / o o \  |
|  JÁNOŠÍK  \  ~  /  |
|             """     |
|    IT konzultant    |
+--------------------+
```

2. Napíšte program, ktorý zo znakov `#` (krížik) vypíše zväčšený nejaký text, napr.

```
####  #   # ##### #   #  ###  #   #
#   # #   #   #   #   # #   # ##  #
####   ###    #   ##### #   # # # #
#       #     #   #   # #   # #  ##
#       #     #   #   #  ###  #   #
```

3. Napíšte program, ktorý vypočíta a vypíše 2 umocnené na 100. Výsledok vypíšte v tvare:

```
2 ** 100 = 126...
```

4. Napíšte program, ktorý pod seba vypíše hodnoty súčinov `1*1`, `11*11`, `111*111`, `1111*1111`, ..., `111111111*111111111`:

```
1
11
121
12321
...
12345678987654321
```

5. Napíšte program, ktorý vypíše hodnoty faktoriálov čísel od 1 do 10:

```
1 faktorial = 1
2 faktorial = 2
3 faktorial = 6
4 faktorial = 24
5 faktorial = 120
6 faktorial = 720
7 faktorial = 5040
8 faktorial = 40320
9 faktorial = 362880
10 faktorial = 3628800
```

Uvedomte si, že napr. 6 faktoriál vieme vypočítať ako `1*2*3*4*5*6`.

6. Napíšte program, v ktorom je na začiatku v premennej `cislo` priradená nejaká hodnota. Program vypíše najprv samotné číslo, potom o `1` väčšie číslo (nasledovníka) a na záver o `1` menšie číslo (predchodcu) v tvare, napr.:

```
Zadane cislo je 142
Nasledovnik cisla 142 je 143
Predchodca cisla 142 je 141
```

7. Škôlkari **Jurko**, **Marienka** a **Peťo** sa hrajú s guľôčkami. Vieme, že **Jurko** má `pocet1` guľôčok, **Marienka** má od neho o `pocet2` guľôčok viac a **Peťo** má o `pocet3` guľôčok viac ako **Jurko** a **Marienka** dokopy. Napíšte program, ktorý pre tri premenné `pocet1`, `pocet2` a `pocet3` vypíše, koľko gulôčok má každý zo škôlkarov. Ak predpokladáme, že `pocet1=7`, `pocet2=5`, `pocet3=3`, program vypíše:

```
Jurko ma 7 gulocok.
Marienka ma 12 gulocok.
Peto ma 22 gulocok.
```

8. Napíšte program, ktorý prepočíta sumu peňazí v českých korunách na euro. V premennej `ck` máme priradenú túto sumu a predpokladáme, že kurz je 26 korún za 1 euro, potom program vypíše, napr.

```
suma 235.5 ck
bude 9.057692307692308 euro
```

Neskôr sa naučíme takéto čísla zaokrúhľovať.

10. Napíšte program, ktorý bude robiť opačný prevod medzi menami ako v príklade (6): v premennej `euro` máme priradenú nejakú sumu v eurách a program ju vypíše v českých korunách. Napr.

```
suma 17.5 euro
bude 455.0 ck
```

11. Na účte v banke máme nejakú sumu peňazí (v premennej `ucet`). Na tomto účte každý rok pribudnú 4%. Vypíšte stav účtu po jednom, dvoch a troch rokoch. Napr.

```
na ucte je 150 euro
po prvok roku 156.0 euro
po druhom roku 162.24 euro
po tretom roku 168.7296 euro
```

Zhrnutie

čo sme sa naučili:

- s Pythonom sa dá pracovať v interaktívnom aj v programovom režime
- program sa musí nachádzať v textovom súbore a môže obsahovať komentáre, priraďovacie príkazy a príkazy `print` pre výpis
- program spúšťame pomocou povelu **Run** (klávesovou skratkou `<F5>` alebo cez menu)
- v programe môžu byť syntaktické, semantické alebo logické chyby

dôležité zásady:

- vkladajme do programu komentáre, aby sme my a aj iný čitateľ ľahšie pochopili jeho funkčnosť
- programy zapisujme čo najviac čitateľne, napr. voľbou vhodných mien premenných, medzerami medzi operáciami a tiež za čiarkami v príkaze `print`

4. Grafická plocha, obdĺžniky

Zopakujme si:

- program sa nachádza v súbore
- v každom riadku je jeden príkaz (priradenie alebo výpis) [1
- program môže obsahovať komentáre aj prázdne riadky [2
- ak riadok obsahuje nejakú hodnotu alebo výraz, ktorý sa nepriradí do premennej ani nie je parametrom v `print()` , takáto hodnota sa ignoruje [3

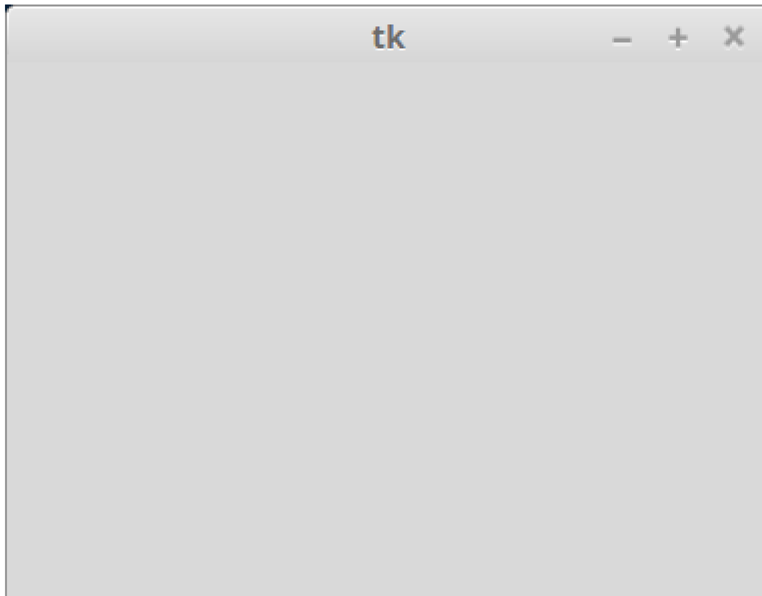
V tejto téme sa naučíme vytvárať program (textový súbor s príkazmi), ktorý bude kresliť do **grafickej plochy**. Zatiaľ sme sa naučili vypisovať texty a číselné hodnoty do textovej plochy (okno Shell) pomocou príkazu `print` . Na kreslenie do grafickej plochy budeme potrebovať viac rôznych príkazov.

Vytvorme si plátno

Skôr ako začneme kresliť, musíme použiť príkazy na **vytvorenie grafickej plochy**. Nasledovná postupnosť príkazov vytvorí grafické okno, v ktorom sa bude nachádzať plocha (tzv. **plátno**) určená na kreslenie. Najjednoduchší variant vytvorenia grafickej plochy s plátnom je [4:

```
import tkinter

platno = tkinter.Canvas()
platno.pack()
```

Spustíme a vidíme:



Všimnite si grafickú aplikáciu. Táto sa správa rovnako ako iné bežné aplikácie: samotné okno môžeme presúvať aj mu meniť veľkosť ťahaním. Vnútro grafického okna je šedé: to je plátno, na ktoré budeme kresliť. Túto aplikáciu **zatvoríme** rovnako ako iné aplikácie, kliknutím na tlačidlo `x` v pravom hornom rohu.

Ďalej sa budeme postupne zoznamovať s grafickými príkazmi, ktoré všetky majú takýto tvar:

```
platno.create_utvar(parametre)
```

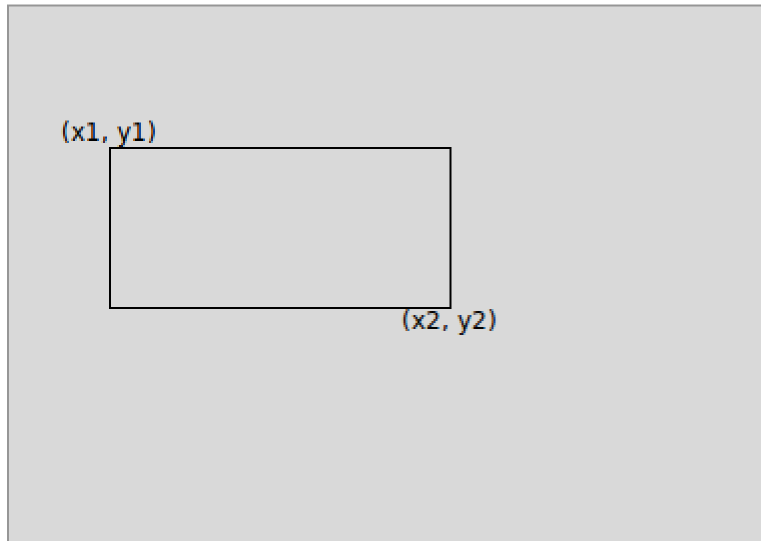
Každý takýto príkaz nakreslí do plátna nejaký útvar. Aký útvar sa nakreslí, určuje meno v príkaze `create_`. Napr. `create_rectangle` nakreslí obdĺžnik, `create_line` nakreslí úsečku a `create_text` vloží do grafickej plochy nejaký text. **Parametre** príkazu určujú detaily kresleného útvaru, napr. pozíciu, farbu, hrúbku a pod. [5

Príkaz na kreslenie obdĺžnikov

Základný tvar príkazu na nakreslenie obdĺžnikov je:

```
platno.create_rectangle(x1, y1, x2, y2)
```

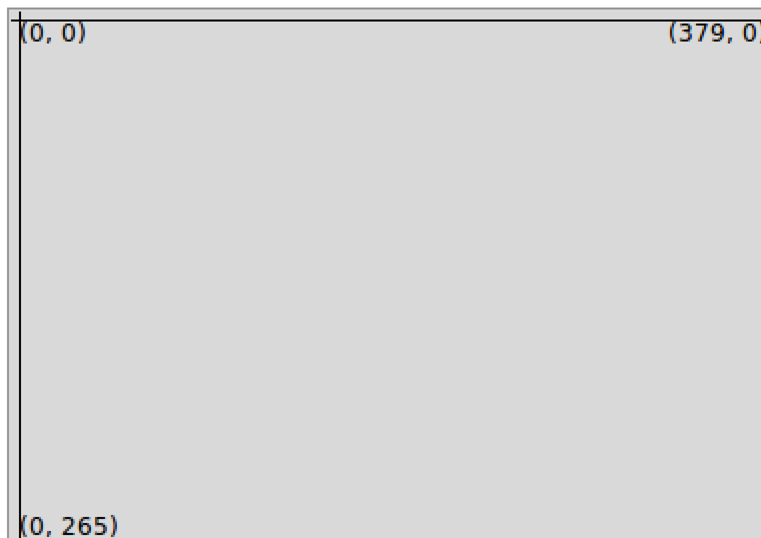
kde 4 parametre `x1`, `y1`, `x2` a `y2` označujú súradnice kresleného obdĺžnika:



Vidíte, že obdĺžnik má strany rovnobežné so stranami grafickej plochy. Prvé dva parametre `x1`, `y1` označujú súradnice ľavého horného vrcholu obdĺžnika a `x2`, `y2` pravého dolného vrcholu.

Súradnicová sústava

Musíme rozumieť, ako v Pythone funguje sústava súradníc. **x-ová os** ide zľava doprava, pričom prechádza po hornej hrane grafickej plochy. **y-ová os** ide zhora nadol a prechádza po ľavej hrane grafickej plochy. Počiatok `(0, 0)` je v ľavom hornom rohu:

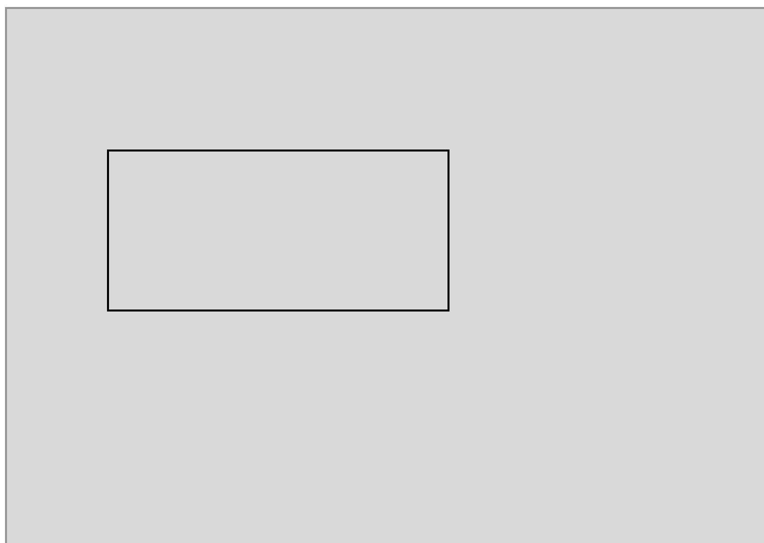


Všimnite si, že **y-ová os** je orientovaná opačne, ako sme zvyknutí z matematiky: smerom nadol sú kladné hodnoty a smerom nahor (nad počiatok `(0, 0)`) sú to záporné hodnoty.

Teraz už môžeme zapísať náš prvý grafický program, ktorý bude kresliť obdĺžnik:

```
# prvy graficky program

import tkinter

platno = tkinter.Canvas()
platno.pack()

platno.create_rectangle(50, 70, 220, 150)
```

ktorý vytvorí takúto kresbu:



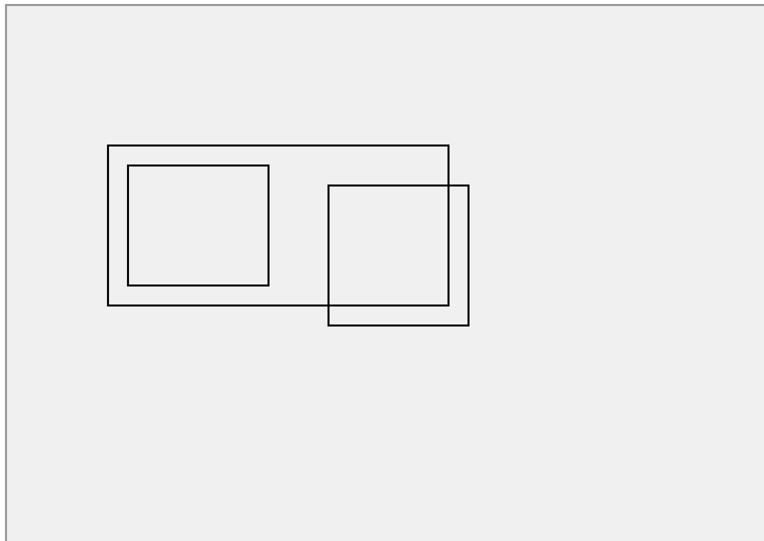
Preskúmajte, čo v tomto programe znamenajú číslené konštanty `50` , `70` , `220` , `150` : prvá dvojica čísel `50` a `70` označujú súradnice ľavého horného rohu, teda bod `(50, 70)` (kde `50` je **x**-ová súradnica tude vzdialenosť 50 jednotiek od ľavého okraja a `70` je **y**-ová súradnica, teda koľko jednotiek od horného okraja). Druhá dvojica

čísel `(220, 150)` označuje súradnice pravého dolného bodu. Uvedomte si, že **šírka** obdĺžnika sa počíta ako rozdiel (vzdialenosť) **x**-ových súradníc a **výška** ako rozdiel **y**-ových súradníc. Teda rozmery tohto obdĺžnika sú `170` (šírka) a `80` (výška).

Obdĺžnikov môžeme nakresliť aj viac a môžu sa aj rôzne prekrývať:

```
# prvy graficky program

import tkinter

platno = tkinter.Canvas()
platno.pack()

platno.create_rectangle(50, 70, 220, 150)
platno.create_rectangle(60, 80, 130, 140)
platno.create_rectangle(160, 90, 230, 160)
```

a teraz to vyzerá takto:



Prepočítajme ešte rozmery týchto troch obdĺžnikov:

- prvý obdĺžnik `platno.create_rectangle(50, 70, 220, 150)` sme už počítali vyššie a vieme, že jeho rozmery sú `170 x 80` (šírka x výška)
- druhý obdĺžnik `platno.create_rectangle(60, 80, 130, 140)` má rozmery `70 x 60` (t.j. `130-60` a `140-80`)
- tretí obdĺžnik `platno.create_rectangle(160, 90, 230, 160)` má rozmery `70 x 70`

Chyby v programe

Čím spoznáme viac programátorských konštrukcií, tým viac sa stretneme s rôznymi typmi chýb. Zatiaľ budú veľmi časté **preklepy** pri zápise zložitejších konštrukcií. Tu nás najčastejšie upozorní Python, napr. môžeme otestovať v interaktívnom režime:

```
>>> import tknter
...
ModuleNotFoundError: No module named 'tknter'
```

Označuje preklep v mene grafického modulu `tkinter`, opravíme:

```
>>> import tkinter
>>> platno = tkinter.canvas()
...
AttributeError: module 'tkinter' has no attribute 'canvas'
```

Slovo `Canvas` malo mať prvé písmeno veľké, Python je na malé a veľké písmená veľmi citlivý. Opravíme

```
>>> platno = tkinterCanvas()
...
NameError: name 'tkinterCanvas' is not defined
```

Medzi slová `tkinter` a `Canvas` sme mali vložiť znak `.` (bodka). Python to teraz pochopil ako jedno dlhé nezrozumiteľné slovo `tkinterCanvas`. Opravíme a pokračujeme

```
>>> platno = tkinter.Canvas()
>>> platno.Pack()
...
AttributeError: 'Canvas' object has no attribute 'Pack'
```

Opäť preklep v slove `Pack`, ktoré malo byť zapísané malými písmenami:

```
>>> platno.pack()
```

Horšie sa odhaľujú chyby s chýbajúcimi zátvorkami. Ak zabudneme zátvorky sa slovom `Canvas`, dostávame veľmi nezrozumiteľnú správu až v ďalšom príkaze:

```
>>> platno = tkinter.Canvas
>>> platno.pack()
...
TypeError: pack_configure() missing 1 required positional argument: 'self'
```

Ešte menej zrozumiteľné je to s chýbajúcimi zátvorkami za slovom `pack`:

```
import tkinter
platno = tkinter.Canvas()
platno.pack
platno.create_rectangle(50, 50, 150, 100)
```

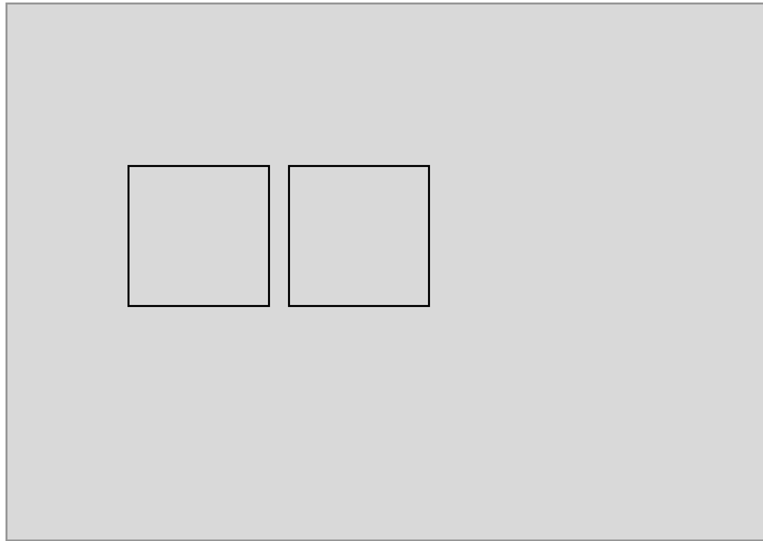
V tomto prípade Python nehlási žiadnu chybu: zápis `platno.pack` je pre počítač korektným zápisom, lenže tento zápis nezobrazí plátno grafickej plochy a ďalší príkaz `create_rectangle` kreslí do plátna, ale toto plátno sa zatiaľ nezobrazuje. Teraz by stačilo zadať napr.

```
>>> platno.pack()
```

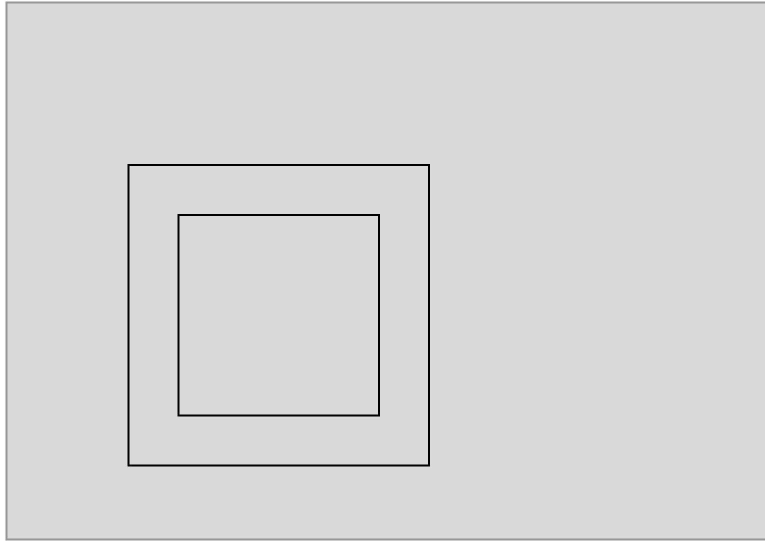
a v grafickom okne vidíme plátno aj s nakresleným obdĺžnikom.

Úlohy

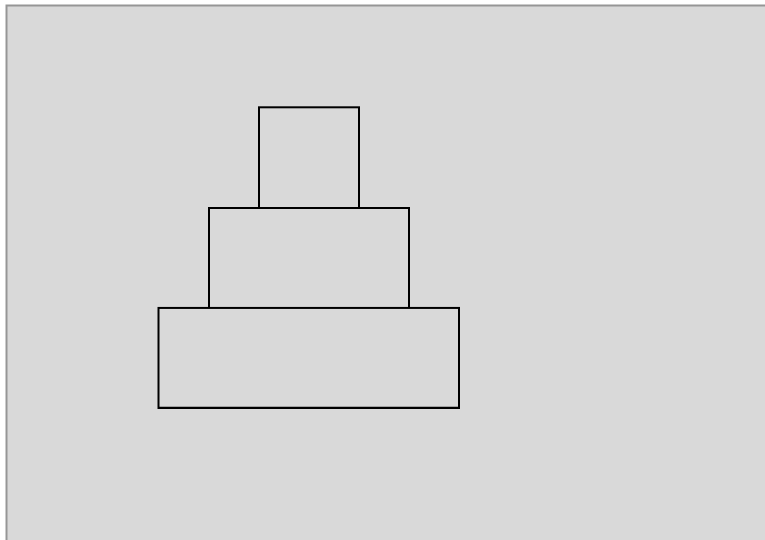
1. Nakreslite obdĺžnik, ktorého ľavý horný vrchol má súradnice `(30, 100)`, jeho šírka je `300` a výška `50`.
2. Napíšte program, ktorý nakreslí dva rovnako veľké štvorce, ktoré sú umiestnené vedľa seba takto:



3. Napíšte program, ktorý nakreslí dva veľké štvorce: jeden so stranou 100 a druhý 150, ktoré majú spoločný stred:



4. Napíšte program, ktorý z troch obdĺžnikov 150x50, 100x50 a 50x50 nakreslí takúto pyramídu:



Zhrnutie

čo sme sa naučili:

- programy v Pythone môžu pracovať s textovou plochou (výsledky sa dozvedáme pomocou `print`) alebo s grafickou (do plátna kreslíme pomocou `create_rectangle`).
- príkaz `create_rectangle` na kreslenie obdĺžnikov má prvé štyri parametre číselné (súradnice vrcholov obdĺžnika)

dôležité zásady:

- pri zadávaní najmä zložitejších zápisov príkazov si treba dávať veľký pozor na zámenu malých a veľkých písmen, prípadne treba skontrovať, či ste nezabudli `()` tzv. **prázdne okrúhle zátvorky**

---

[1] Každý príkaz musí začínať už v prvom znaku riadka, t.j. riadok nesmie začínať medzerami.

[2] Riadky s komentármi môžu začínať v ľubovoľnom stĺpci. Komentáre môžu byť aj súčasťou riadkov s nejakými príkazmi. Vtedy sa za komentár považuje celý zvyšok riadka

[3] Python takýto výraz naozaj vyhodnotí, ale keďže „nevie“ čo s tým, tak ju zahodí. Častou chybou začiatočníka sú práve takéto stratené hodnoty, napr. zahodenie výsledkov volania funkcie

[4] Na prácu s plátnom sme tu zvolili premennú s menom `platno` . V skutočnosti sme mohli použiť ľubovoľné iné meno a ďalej namiesto `platno` pracovať s týmto menom. Niekedy sa zvykne používať meno `canvas` alebo `grafika` , niekedy skrátene aj `c` alebo `g` .

[5] V tejto aj v ďalších častiach sa budete zoznamovať s väčším počtom príkazov a parametrov. Je vhodné si tieto príkazy zapisovať do nejakej rozumnej tabuľky, aby ste to mali stále na očiach. Bolo by nevhodné, keby ste sa mali toto učiť naspamäť.

5. Farby pri kreslení obdĺžnikov

Zopakujme si:

- program, ktorý kreslí do grafickej plochy, musí začínať tromi špeciálnymi riadkami:

```
import tkinter
platno = tkinter.Canvas()
platno.pack()
```

-

Zafarbíme útvary

Každý kreslený útvar môže byť zafarbený rôznymi farbami. Napr. vieme zafarbiť nielen vnútro obdĺžnika, ale aj obrysové úsečky. Pre nás budú najužitočnejšie tieto farby:

- **blue** - modrá
- **red** - červená
- **green** - zelená
- **yellow** - žltá
- **gray** - šedá
- **black** - čierna
- **white** - biela

Môžete ich vidieť:

Python používa veľkú škálu preddefinovaných (vyše 500) farieb. Tu môžete vidieť ukážku niekoľkých z nich:

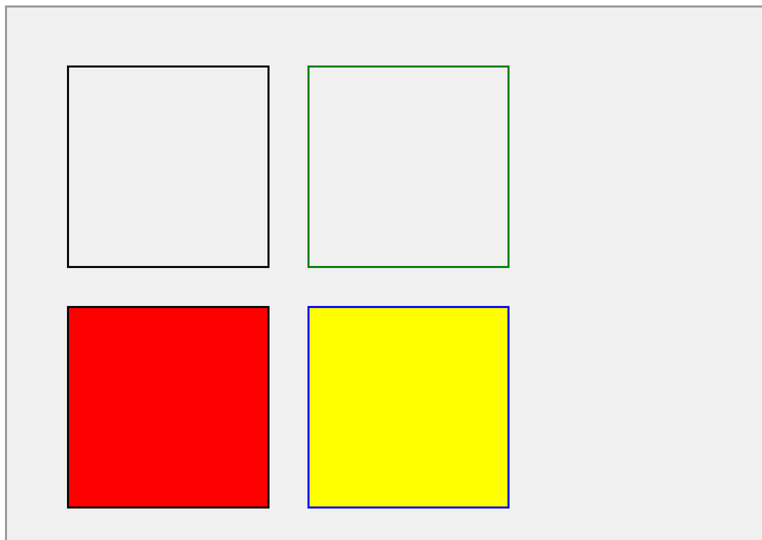
Blue	LightBlue	Cyan	SkyBlue	CornFlowerBlue	DeepSkyBlue	DodgerBlue
RoyalBlue	SlateBlue	SteelBlue	MediumBlue	Navy	Red	SandyBrown
Salmon	Coral	Tomato	Orange	DarkOrange	OrangeRed	IndianRed
Chocolate	Tan	Maroon	Sienna	Brown	SaddleBrown	Pink
Plum	Violet	Orchid	Magenta	Purple	DarkMagenta	Green
PaleGreen	YellowGreen	MediumSeaGreen	LawnGreen	LimeGreen	ForestGreen	DarkGreen
Yellow	Khaki	Gold	Gray	LightGray	Black	White

Už vieme, že Python dokáže zafarbovať útvary pomocou mien farieb. Robíme to tak, že do príkazu `create_rectangle` môžeme pridať ďalšie parametre, ktoré upresňujú napr. farebnosť útvaru. Pre obdĺžniky sú to dva parametre: tzv. `fill` , ktorý určuje výplň a `outline` , ktorý určuje farbu obrysových čiar obdĺžnika. Zatiaľ sme tieto parametre nenastavovali, preto sa obdĺžniky kreslili bez výplne (mali priesvitné vnútro) s čiernym obrysom.

Tieto dva nové parametre sa zapisujú za štyri čísla, ktoré určujú súradnice obdĺžnika. Ukážme to na príklade, v ktorom nakreslíme 4 obdĺžniky:

```
# program kresli farebne obdlzniky

import tkinter

platno = tkinter.Canvas()
platno.pack()

# bez vyplne s ciernym obrysom:
platno.create_rectangle(30, 30, 130, 130)

# cervena vypln s ciernym obrysom:
platno.create_rectangle(30, 150, 130, 250, fill='red')

# bez vyplne so zelenym obrysom:
platno.create_rectangle(150, 30, 250, 130, outline='green')

# zlta vypln s modrym obrysom:
platno.create_rectangle(150, 150, 250, 250, fill='yellow', outline='blue')
```

Dostávame tieto útvary v grafickej ploche:

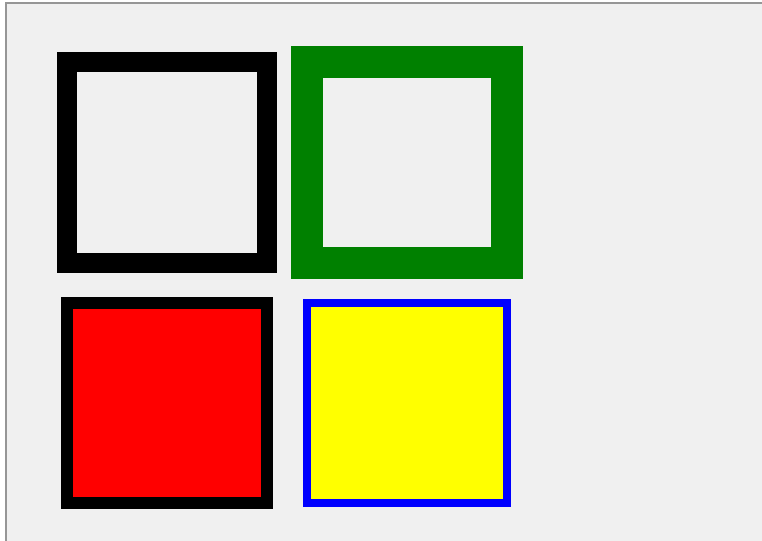


Parametre `fill` , resp. `outline` píšeme za súradnice vrcholov, dávame za ne znak **rovná sa** a meno farby v tvare **znakový reťazec**. Zrejme si meno farby nemôžeme vymyslieť podľa ľubovôle, ale musíme sa trafiť do jednej z Pythonovských farieb. [1

Asi ste si všimli, že farbu obrysu až tak veľmi vidieť nie je: keďže sú tieto obrysové čiary veľmi tenké, ťažšie sa odlíši, či je čiara čierna, zelená alebo modrá. Tu sa nám môže hodiť parameter `width` , ktorým zmeníme hrúbku obrysových čiar. Už vieme, že štandardne je to `1` , ale jednoducho môžeme zmeniť aj toto. Predchádzajúci program upravíme tak, že každému z obdĺžnikov nastavíme nejakú hrúbku:

```
# program kresli farebne obdlzniky s hrubsimi ramikmi

import tkinter

platno = tkinter.Canvas()
platno.pack()

# bez vyplne s ciernym obrysom:
platno.create_rectangle(30, 30, 130, 130, width=10)

# cervena vypln s ciernym obrysom:
platno.create_rectangle(30, 150, 130, 250, width=6, fill='red')

# bez vyplne so zelenym obrysom:
platno.create_rectangle(150, 30, 250, 130, width=16, outline='green')

# zlta vypln s modrym obrysom:
platno.create_rectangle(150, 150, 250, 250, width=4, fill='yellow', outline='blue')
```

Takto sa zmenia nakreslené útvary:

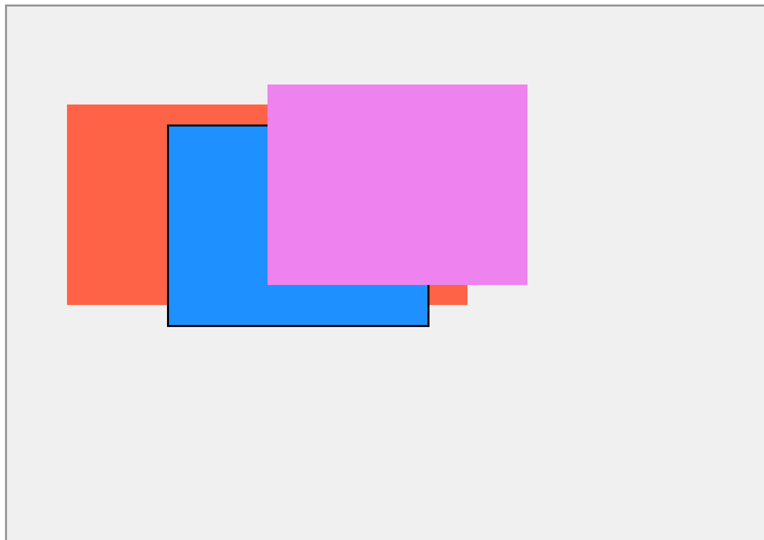


Všimnite si, že pomocou parametra `width` môžeme vytvárať rôzne hrubé rámiky. Ak by sme ale potrebovali rámik úplne zrušiť, nastavíme mu hrúbku obrysu `width=0` alebo farbu obrysu nastavíme na priesvitnú `outline=''`.

Nakreslime dva farebné obdĺžniky, ktoré sa čiastočne prekrývajú:

```
# program nakresli prekryvajuce sa obdlzniky

import tkinter

platno = tkinter.Canvas()
platno.pack()

platno.create_rectangle(30, 50, 230, 150, fill='tomato', outline='')
platno.create_rectangle(80, 60, 210, 160, fill='dodger blue')
platno.create_rectangle(130, 40, 260, 140, fill='violet', width=0)
```

Všimnite si, že neskôr nakreslený útvar prekrýva tie, ktoré už boli nakreslené skôr:

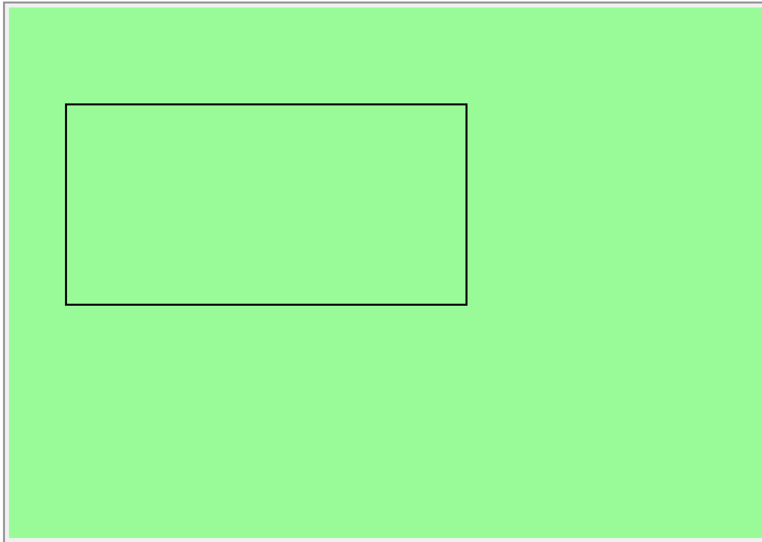


Toto je veľmi dôležitá vlastnosť, ktorú si treba uvedomiť vždy vtedy, keď sa obrázok skladá z viacerých prekrývajúcich sa útvarov.

Doteraz mala naša grafická plocha šedé pozadie. Niekedy sa nám môže ale hodiť, aby malo plátno inú farbu hneď na začiatku. Hoci sa to dá urobiť pomocou veľkého obdĺžnika, ktorému nastavíme väčšie rozmery, ako sú rozmery plátna, napr.

```
import tkinter

platno = tkinter.Canvas()
platno.pack()

platno.create_rectangle(0, 0, 400, 300, fill='paleGreen')

platno.create_rectangle(30, 50, 230, 150)
```

Dostaneme:



Podobný výsledok ale dosiahneme, keď už pri vytváraní plátna mu pomocou parametra `bg` nastavíme **farbu pozadia**:

```
import tkinter

platno = tkinter.Canvas(bg='paleGreen')
platno.pack()

platno.create_rectangle(30, 50, 230, 150)
```

Zrejme, toto riešenie má tú výhodu, že bude fungovať, aj keď sa neskôr naučíme meniť veľkosť grafickej plochy.

RGB model

Z informatiky na predchádzajúcich hodinách pravdepodobne viete, že jedným z modelov, ako sa v počítači dajú „namiešať" farby je tzv. **RGB model**. V tomto modeli sa každá farba skladá z troch zložiek: červenej (Red), zelenej (Green) a modrej (Blue). V závislosti od toho, koľko ktorej zložky namiešame, dostávame rôzne farby. Najčastejšie sa množstvá zložiek určujú číslami od `0` do `255`, kde `0` označuje, že danú zložku nepoužijeme vôbec a `255` označuje, že ju použijeme v maximálnom možnom množstve. Napr. oranžová farba (`orange`)

vzniká tak, že zoberieme maximum červenej (teda 255), pridáme dosť aj zelenej (165) a modrej nedáme vôbec. Toto budeme zapisovať takouto trojicou čísel (255, 165 0), teda presne v poradí koľko je tam červenej, koľko zelenej a koľko modrej zložky.

Tyrkysová farba `cyan` je reprezentovaná trojicou (0, 255, 255), čo označuje, že v nej úplne chýba červená zložka a zelená a modrá je na maxime. Už si len zapamätajme, že bile farba `white` má trojicu (255, 255, 255) a čierna `black` (0, 0, 0).

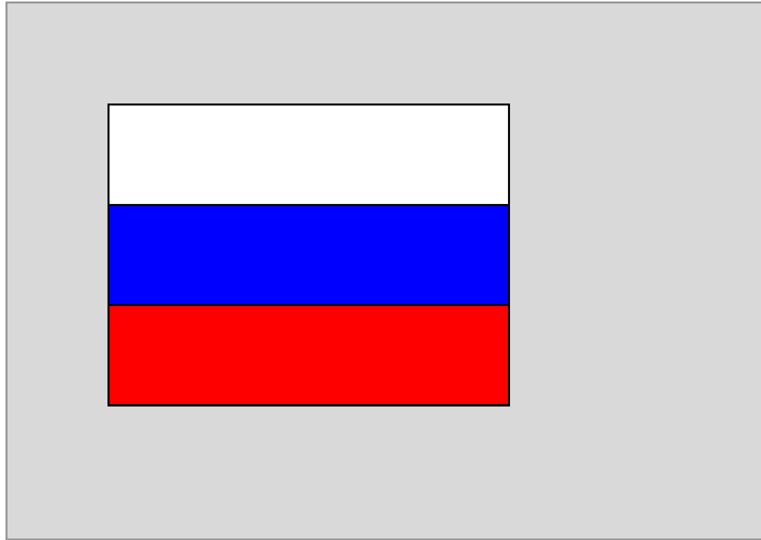
Samozrejme, že sa tieto trojice nemusíme učiť naspamäť. Na tomto mieste to pripomíname preto, že aj v Pythone môžeme určovať farby pomocou trojíc čísel RGB modelu. Hoci nie je to veľmi jednoduché, ale tí z vás, ktorí máte skúsenosti so 16-ovou sústavou, resp. s kódovaním farieb v HTML stránkach, môžete pochopiť, ako zakódujete ľubovoľnú RGB trojicu do mena farby:

```
'#rrggbb'
```

Teda namiesto mena farby použijeme šestnástkový zápis jednotlivých zložiek ako dve šestnástkové cifry pre červenú `rr`, dve cifry `gg` pre zelenú a dve cifry `bb` pre modrú zložku. Oranžovú farbu `orange` teda môžeme teda zapísať aj ako `'#ffa500'`, čo zodpovedá trojici (255, 165, 0).

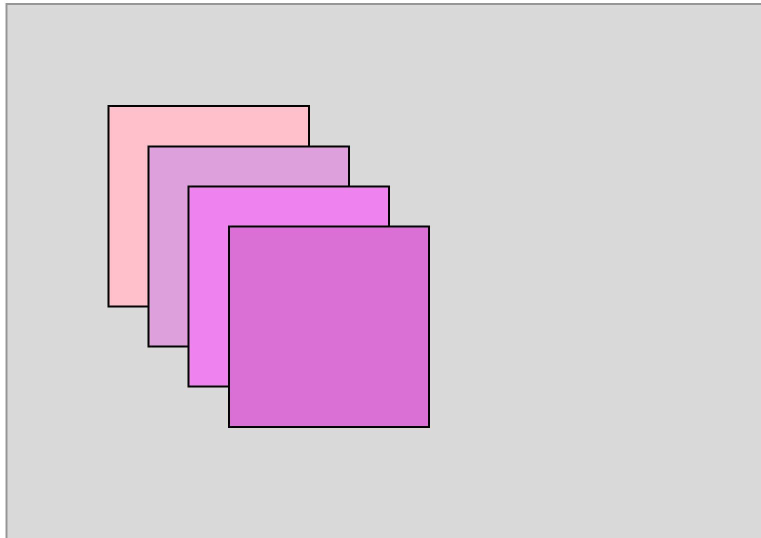
Úlohy

1. Napíšte program, ktorý nakreslí tri farebné obdĺžniky vo farbách slovenskej trikolóry:



Navrhnite to tak, aby veľkosti týchto farebných obdĺžnikov boli: 200x150, 200x100, 200x50. Môžete zvoliť nejaké tri iné rôzne farby (pre trikolóru iného štátu).

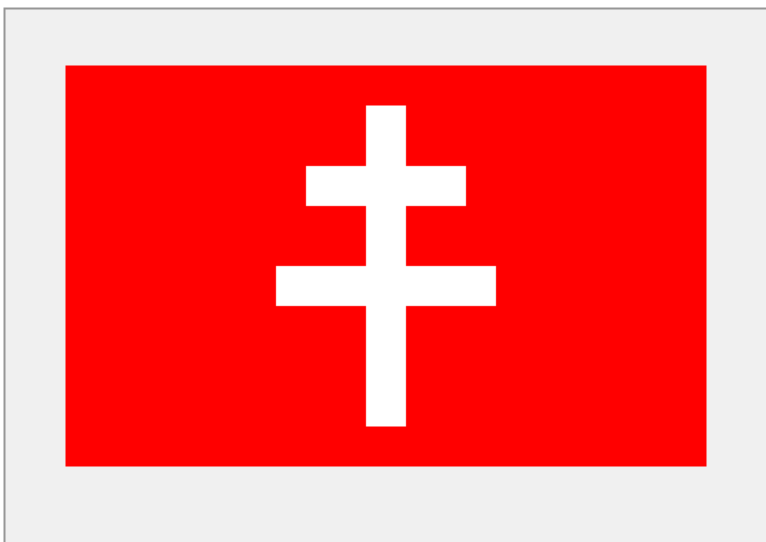
2. Nasledovný obrázok vznikol zo štyroch štvorcov. Prvý z nich má súradnice ľavého horného vrcholu (50, 50). Napíšte program, ktorý nakreslí tieto štyri prekrývajúce sa štvorce. Zvoľte si ľubovoľné štyri rôzne farby (môžu byť iné ako na tomto obrázku):



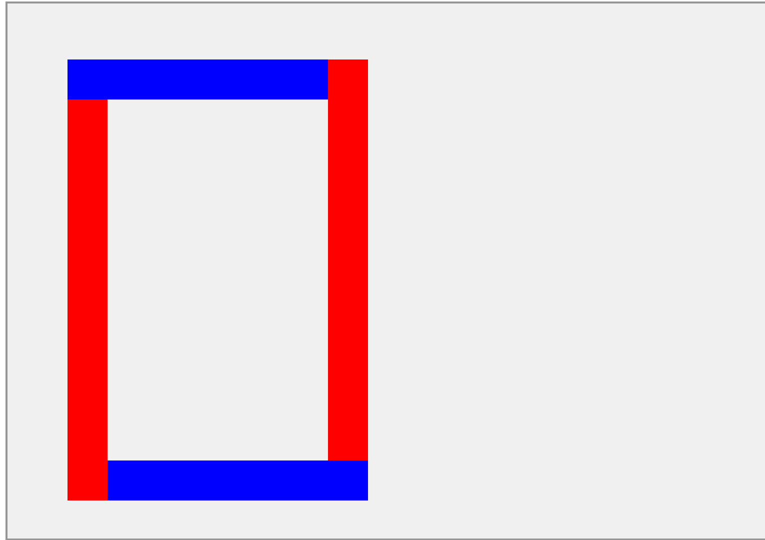
3. Týchto 5 volaní `create_rectangle` nakreslí nejaký obrázok. Prepíšte program tak, aby sa nakreslil presne taký isý obrázok, ale len s tromi volaniami `create_rectangle` :

```
platno.create_rectangle(90, 90, 150, 150, fill='yellow')
platno.create_rectangle(150, 90, 210, 150, fill='red')
platno.create_rectangle(90, 150, 150, 210, fill='green')
platno.create_rectangle(30, 90, 90, 150, fill='red')
platno.create_rectangle(90, 30, 150, 90, fill='green')
```

4. Nakreslite slovenský dvojkríž, napr.



5. Zo štyroch úzkych obdĺžnikov vytvorte takýto rámik:



Zhrnutie

čo sme sa naučili:

- Príkaz `create_rectangle` na kreslenie obdĺžnikov má prvé štyri parametre číselné (súradnice vrcholov obdĺžnika) a za tým môže mať ďalšie parametre, ktorými sa nastavujú farby a hrúbka čiar.
- Farby sa zadávajú ako znakové reťazce buď ich menami (napr. `'blue'` alebo `'white'`) alebo šestnástkovými kódmi RGB reprezentácie (napr. `'#00ffff'` pre tyrkysovú farbu `cyan`).

---

[1] Python pri menách farieb nerozlišuje malé a veľké písmená a viacslovné mená dovolí rozdeliť na viac slov. napr. `'SkyBlue'` môžeme zapísať napr. aj ako `'sky blue'` .

6. Použitie premenných a výrazov pri kreslení

Zopakujme si:

- pomocou príkazu `create_rectangle` vieme do grafickej plochy kresliť obdĺžniky
- tieto obdĺžniky môžeme rôzne zafarbiť a určovať im aj rôzne hrúbky obrysových čiar
- polohy a veľkosti obdĺžnikov určujeme štyrmi číslami, tieto sú súradnice dvoch protiľahlých vrcholov obdĺžnikov

Parametre s premennými a výrazmi

Vo všetkých doterajších programoch, ktoré kreslili pomocou príkazu `create_rectangle` , sme ako súradnice používali len číselné konštanty. Pritom parametrami môžu byť aj aritmetické výrazy. Pozrite nasledovný príkaz:

```
platno.create_rectangle(130, 40, 250, 160, fill='violet')
```

Nakreslí fialový obdĺžnik nejakých rozmerov. Ak ale toto isté zapíšeme takto:

```
platno.create_rectangle(130, 40, 130 + 120, 40 + 120, fill='violet')
```

môžeme vidieť, že obdĺžnikom je štvorec s veľkosťou strany 120, ktorého ľavý horný vrchol má súradnice (130, 40).

Možno ešte prehľadnejšie by to bolo, keď parametre zapíšeme pomocou premenných. Napr.

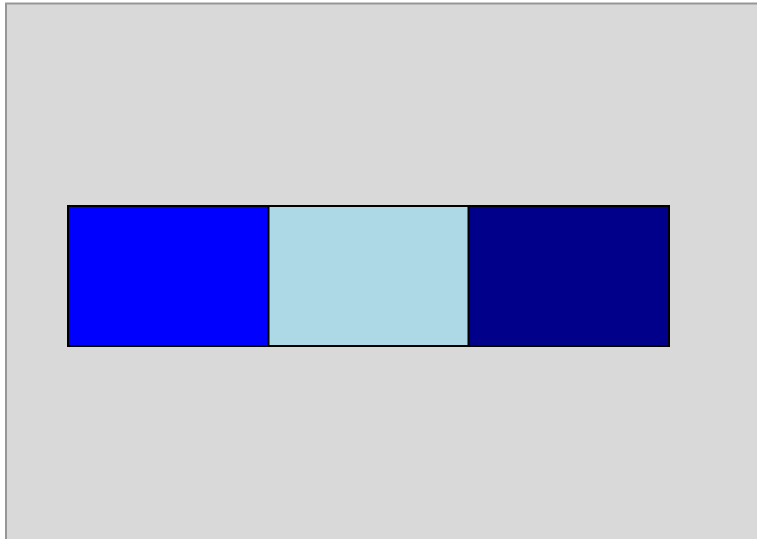
```
x = 100
y = 70
sirka = 55
vyska = 35
platno.create_rectangle(x, y, x + sirka, y + vyska)
```

Nakreslí obdĺžnik so veľkosti 55x35 s ľavým horným vrcholom (100, 70). Zrejme, ak by premenné `sirka` a `vyska` mali rovnakú hodnotu, príkaz `create_rectangle` by nakreslil štvorec.

Použitie premenných ukážeme na takomto príklade: nakreslíme vedľa seba tri farebné obdĺžniky veľkosti `a x b` tak, že ľavý horný vrchol prvého z nich je na súradniciach (`x` , `y`):

```
import tkinter

platno = tkinter.Canvas()
platno.pack()

x = 30
y = 100
a = 100
b = 70
platno.create_rectangle(x, y, x+a, y+b, fill='blue')
platno.create_rectangle(x+a, y, x+2*a, y+b, fill='light blue')
platno.create_rectangle(x+2*a, y, x+3*a, y+b, fill='dark blue')
```

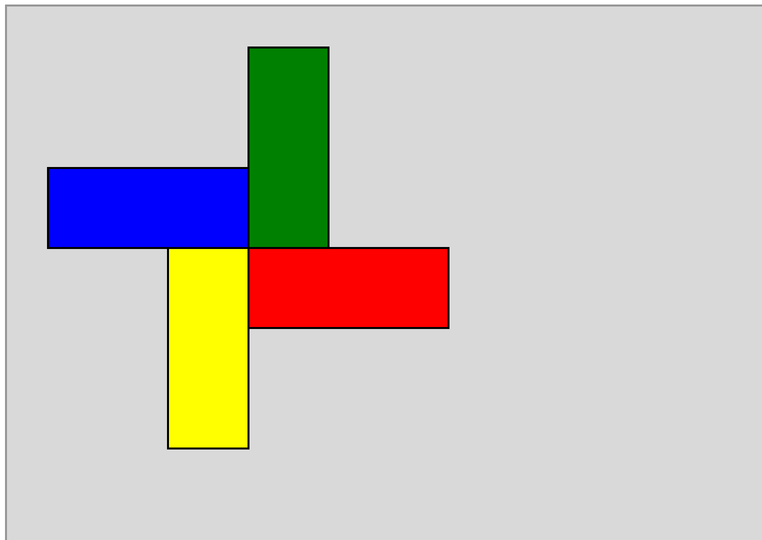
a vyzerá to potom takto:



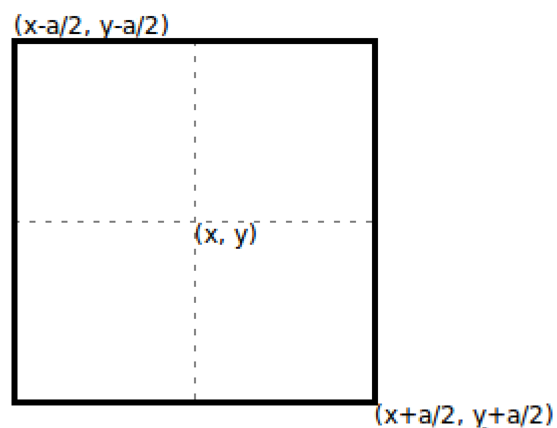
Nasledovný príklad ilustruje použitie premenných `x`, `y`, `a`, `b` na kreslenie štyroch obdĺžnikov, pričom bod (`x`, `y`) nie je ich ľavým horným vrcholom:

```
import tkinter

platno = tkinter.Canvas()
platno.pack()

x = 120
y = 120
a = 100
b = 40
platno.create_rectangle(x, y, x+a, y+b, fill='red')
platno.create_rectangle(x, y, x-a, y-b, fill='blue')
platno.create_rectangle(x, y, x-b, y+a, fill='yellow')
platno.create_rectangle(x, y, x+b, y-a, fill='green')
```

Preštudujte tento program, ktorý nakreslí takýto obrázok:



Pri kreslení štvorcov (aj obdĺžnikov) sa niekedy stretáme s pojmom stred štvorca. Napr. potrebujeme nakresliť dva štvorce, ktoré majú spoločný stred. Tu by sa nám zišlo si premyslieť, ako sa pomocou `create_rectangle` kreslí štvorec so stranou `a` , keď poznáme stred štvorca (`x` , `y`). Najprv si to nakreslime:

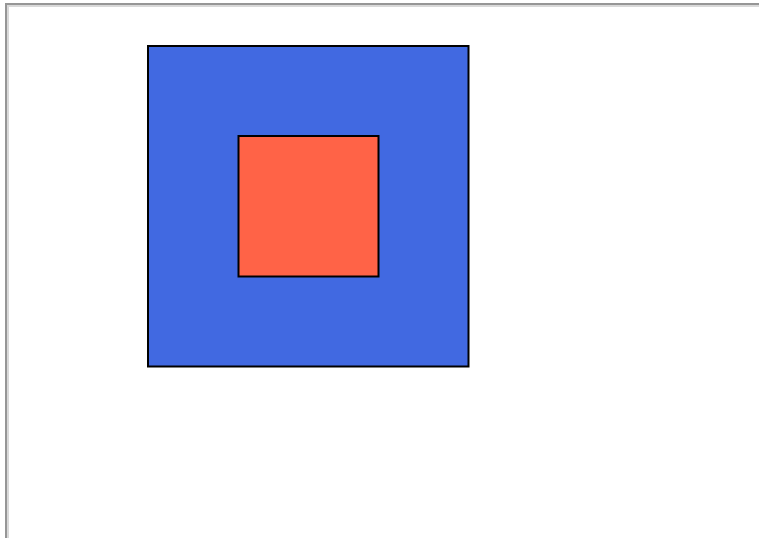


Aby sem sa od stredu štvorca dostali k ľavému hornému vrcholu, musíme `x` -ovú aj `y` -ovú súradnicu zmenšiť o polovicu strany štvorca t.j. o `a/2` . Pravý dolný vrchol štvorca zase dostaneme tak, že k obom súradniciam stredu (`x` , `y`) pripočítame polovicu strany štvorca.

Teraz zapíšeme riešenie úlohy, v ktorej nakreslíme dva farebné štvorce so spoločným stredom:

```
import tkinter

platno = tkinter.Canvas(bg='white')
platno.pack()

x = 150
y = 100
platno.create_rectangle(x-80, y-80, x+80, y+80, fill='royal blue')
platno.create_rectangle(x-35, y-35, x+35, y+35, fill='tomato')
```

Uvedomte si, že väčší štvorec má veľkosť strany `160` a menší z nich veľkosť `70` :



Chyby v programe

Pri kreslení do grafickej plochy môžu nastať situácie, ktoré pre Python nie sú chybou, ale my nikde nevidíme očakávaný výsledok. Tu je niekoľko ukážok:

```
x = 100
y = -70
platno.create_rectangle(x, y, x+50, y+50, fill='red')
```

Časť programu mala nakresliť štvorec so stranou `50`, ktorého ľavý horný vrchol má súradnice (`x` , `y`). Lenže v ploche nevidíme nič. V skutočnosti sa obdĺžnik nakreslil, ale mimo viditeľnú časť plátna: na súradniciach (100, -70), (150, -20).

Podobne je to aj pri kreslení dvoch štvorcov so spoločným stredom:

```
x = 150
y = 100
platno.create_rectangle(x-35, y-35, x+35, y+35, fill='red')
platno.create_rectangle(x-80, y-80, x+80, y+80, fill='blue')
```

Tu vidíme len modrý väčší štvorec, ktorý úplne prekryl menší červený.

Úlohy

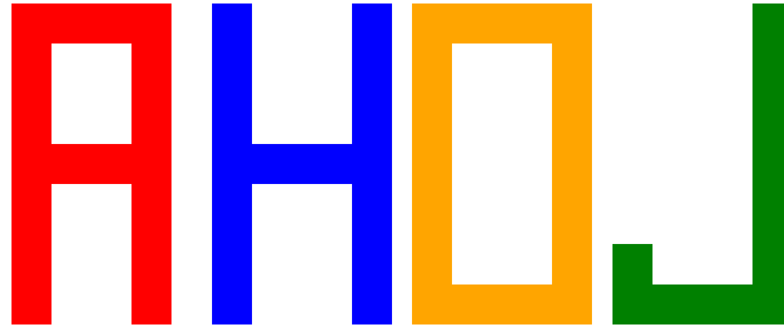
1. Bez toho, aby ste tieto príkazy spúšťali na počítači, zistite, ktoré z týchto riadkov nakreslia štvorce (predpokladajte, že `x` aj `y` je 100):

    ```
    platno.create_rectangle(0, 0, 1, 1)
    platno.create_rectangle(10, 20, 30, 40)
    platno.create_rectangle(100, 150, 150, 100)
    platno.create_rectangle(x, y-50, x+50, y)
    platno.create_rectangle(100-20, 70-30, 100+30, 70+20)
    ```

    Pre každý z týchto obdĺžnikov určte súradnice ľavého horného vrcholu a veľkosť jeho strán.

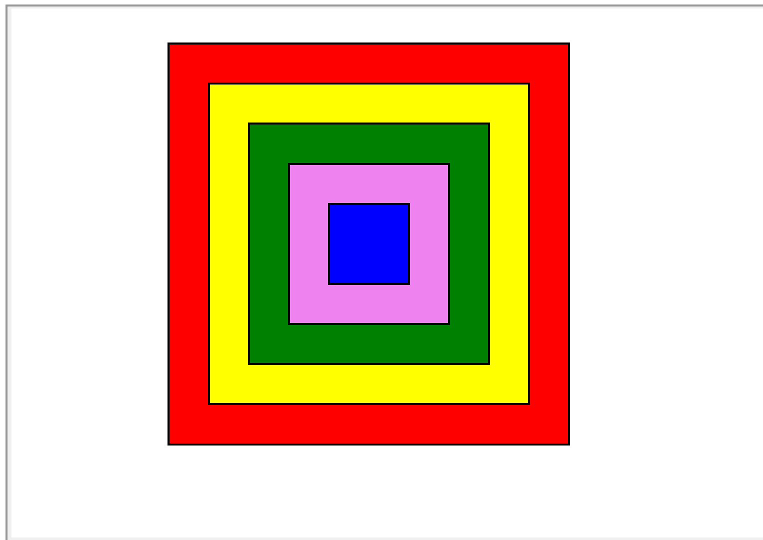
2. Napíšte program, ktorý pomocou farebných obdĺžnikov nakreslí hlavu robota (alebo mimozemšťana). Na hlave by mali byť minimálne dve rovnaké oči a jedny ústa.

3. Napíšte program, ktorý pomocou obdĺžnikov nakreslí niekoľko rôznych písmen, napr.

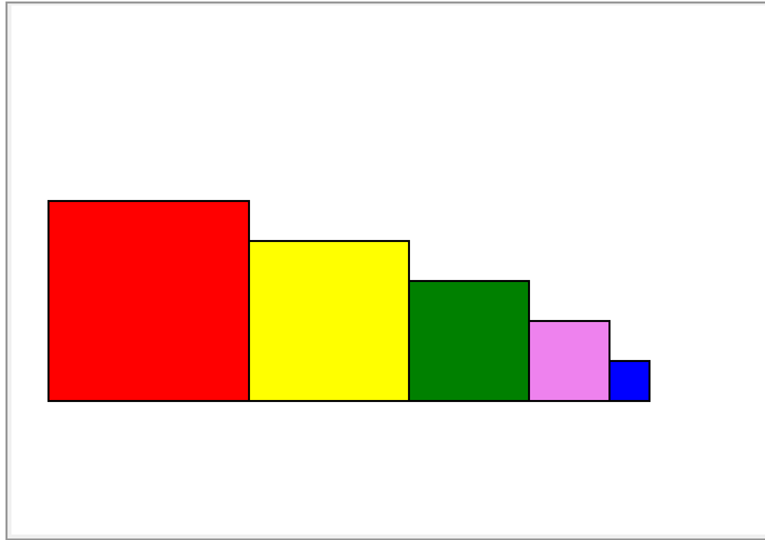


Uvedomte si, že niektoré z týchto nakreslených obdĺžnikov sú biele (teda `fill='white', width=0`), niektoré obdĺžniky môžu mať hrubý obrys (napr. `width=20`).

4. Nakreslite päť vnorených štvorcov so spoločným stredom (napr. `(180, 120)`). Zvoľte ľubovoľných päť farieb. Veľkosti strán štvorcov nech sú 200, 160, 120, 80 a 40. Napr. takto



5. Päť farebných štvorcov ležia tesne vedľa seba na jednej podložke. Veľkosti strán sú postupne 100, 80, 60, 40, 20. Napíšte program, v ktorom ľavý dolný vrchol prvého štvorca má `x=20 a y=200`. Napr. takto



Zhrnutie

čo sme sa naučili:

- Programy v Pythone môžu pracovať s textovou plochou (výsledky sa dozvedáme pomocou `print`) alebo s grafickou (do plátna kreslíme pomocou `create_rectangle`).
- Príkaz `create_rectangle` na kreslenie obdĺžnikov má prvé štyri parametre číselné (súradnice vrcholov obdĺžnika) a za tým môže mať ďalšie parametre, ktorými sa nastavujú farby a hrúbka čiar.
- Farby sa zadávajú ako znakové reťazce buď ich menami (napr. `'blue'` alebo `'white'`) alebo šestnástkovými kódmi RGB reprezentácie (napr. `'#00ffff'` pre tyrkysovú farbu `cyan`).
- Číselnými parametrami príkazu môžu byť aj zložitejšie aritmetické výrazy, ktoré obsahujú aj premenné.

dôležité zásady:

- číselné parametre v príkaze `create_rectangle` je vhodné rozpísať tak, aby bolo čitateľovi zrozumiteľnejšie o aký obdĺžnik, resp. štvorec sa jedná
- je vhodné používať pomocné premenné tak, aby sa ľahšie dalo odladiť kreslenie viacerých navzájom súvisiacich útvarov

7. Vytvárame podprogramy

V tejto časti sa naučíme vytvárať a potom aj používať nové vlastné príkazy. Pripomeňme si najprv, aké príkazy už poznáme:

- **priraďovací príkaz** vytvorí novú premennú, resp. zmení zapamätanú hodnotu premennej
- `print` vypíše do textového okna nejaké texty a číselné hodnoty (aj hodnoty premenných)
- `platno = tkinter.Canvas()` vytvorí plátno grafickej aplikácie
- `platno.pack()` zobrazí plátno v grafickej aplikácii
- `platno.create_rectangle` do plátna nakreslí obdĺžnik

Už sme použili aj `import tkinter` - príkaz, ktorý sme zatiaľ nevysvetľovali. Stačí si pamätať, že vďaka tomuto zápisu môžeme v Pythone pracovať s grafikou `tkinter`.

Zrejme príkazy `platno = tkinter.Canvas()` a `platno.pack()` použijeme len raz na začiatku programu a ďalej sa budú **volať** aj veľakrát ostatné príkazy.

Vlastné príkazy

Z existujúcich príkazov môžeme vytvárať vlastné nové príkazy. Špeciálnym spôsobom označíme **postupnosť príkazov** (tzv. blok), ktorá bude vykonávať náš nový príkaz a tejto postupnosti priradíme **meno**, ktoré sa stane novým príkazom. Budeme hovoriť, že podprogram je **pomenovaný blok príkazov**.

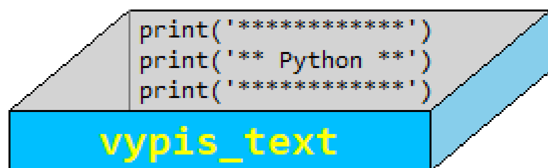
Definovanie nového príkazu (teda podprogramu) zapisujeme takto:

```
def meno_príkazu():
    príkaz
    príkaz
    ...
```

Ukážme to na príklade, v ktorom nový podprogram bude vedieť vypísať nejaký dopredu daný text:

```
def vypis_text():
    print('************')
    print('** Python **')
    print('************')
```

Týmto zápisom sme zadefinovali nový príkaz `vypis_text` , ktorý je od tohto momentu pripravený na vykonanie. Samotné definovanie nového príkazu (podprogramu) ešte nič z postupnosti príkazov nevykonáva. Budeme hovoriť, že definícia podporgramu sa skladá z **hlavičky** - to je prvý riadok definície `def vypis_text():` a z **tela** - blok príkazov, v ktorom je každý riadok **odsunutý o 4 medzery** vpravo. Túto definíciu si zjednodušene môžeme predstaviť ako vytvorenie novej premennej s menom `vypis_text` , ktorej hodnotou je telo, teda blok príkazov:



Môžeme sa o tom presvedčiť tak, že sa pokúsime zistiť hodnotu premennej `vypis_text` :

```
>>> vypis_text
<function vypis_text at 0x0306C348>
```

Python nám namiesto hodnoty tejto premennej oznámil, že meno `vypis_text` je **funkcia**. Takto pythonisti volajú nové príkazy, teda podprogramy. Ak chceme, aby sa tento náš nový príkaz vykonal, musíme ho **zavolať**. Volanie podprogramu sa robí tak, že k jeho menu prilepíme okrúhle zátvorky:

```
meno_príkazu()
```

Vyskúšame to s našim novým príkazom:

```
>>> vypis_text()
************
** Python **
************
>>>
```

Vidíme, že sa postupne vykonali všetky tri príkazy z tela podprogramu a keď už urobil všetko čo mal, vypísali sa znaky `>>>`, ktorými nám Python oznamuje, že je pripravený na ďalšie príkazy. Všimnite si chybovú správu, ktorú dostaneme, keď sa preklepneme v mene príkazu:

```
>>> vypys()
...
NameError: name 'vypys' is not defined
```

Podprogram s menom `vypys` sa nedá zavolať, lebo do neho nebolo ešte nič priradené, nexistuje takáto premenná, v pamäti nemá priradenú hodnotu. Ak teraz zapíšeme napr.

```
>>> vypys = 123
>>> vypys()
...
TypeError: 'int' object is not callable
```

Chybová správa sa zmenila: premenná je už definovaná a keďže je to celé číslo (v Pythone `int`), tak toto sa nedá zavolať (`callable`) ako podprogram.

Vráťme sa k nášmu novému príkazu `vypis_text`. Už sme ho raz zavolali a vtedy vypísal tie tri riadky textu. Lenže my ho môžeme zavolať ľubovoľný počet krát a kým nemeníme definíciu podprogramu, bude stále robiť to isté, teda vypisovať tieto tri riadky. Otestujme to programom:

```
# program definuje a zavola podprogram

def vypis_text():
    print('************')
    print('** Python **')
    print('************')

print('Vitajte!')
vypis_text()
print()
vypis_text()
print('to je koniec')
```

V tomto programe sa najprv definuje podprogram `vypis_text` , potom sa budú pomocou `print` vypisovať nejaké texty a medzi tým sa dvakrát zavolá definovaný podprogram `vypis_text` . Spustíme a dostávame:

```
Vitajte!
************
** Python **
************

************
** Python **
************
to je koniec
```

Všimnite si, že volanie príkazu `print` bez parametrov, teda `print()` , označuje vypísanie prázdneho riadka.

Je dôležité si zapamätať, že podprogram môžeme zavolať až vtedy, keď už bol definovaný predtým. Ak by sme vyskúšali opraviť náš program:

```
# program definuje a zavola podprogram

print('Vitajte!')
vypis_text()

def vypis_text():
    print('************')
    print('** Python **')
    print('************')
```

Dozvieme sa, že v 4. riadku programu sme použili meno `vypis_text()`, ktoré ešte nebolo definované:

```
Vitajte!
Traceback (most recent call last):
  File "podprogram.py", line 4, in <module>
    vypis_text()
NameError: name 'vypis_text' is not defined
>>>
```

Vlastné príkazy s grafikou

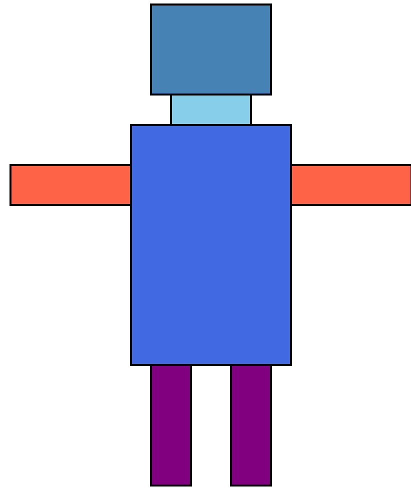
Zatiaľ poznáme jediný grafický príkaz `create_rectangle` na kreslenie obdĺžnikov. Zostavme pomocou neho štyri podprogramy, pomocou ktorých nakreslíme jednoduchého robota:

- nový príkaz `hlava` nakreslí na správnom mieste dva obdĺžniky: pre krk a hlavu
- `telo` nakreslí veľký obdĺžnik pre telo
- `ruky` nakreslí dva úzke obdĺžniky pre dve ruky, prípadne ich spojíme do jedného, aby sa obe ruky nakreslili naraz
- `nohy` nakreslí dva úzke obdĺžniky pre dve nohy

Pozrite takéto použitie podprogramov:

```
# program nakresli robota

import tkinter

platno = tkinter.Canvas(bg='white')
platno.pack()

def hlava():
    platno.create_rectangle(160, 40, 200, 100, fill='sky blue')
    platno.create_rectangle(150, 10, 210, 55, fill='steel blue')

def telo():
    platno.create_rectangle(140, 70, 220, 190, fill='royal blue')

def ruky():
    platno.create_rectangle(80, 90, 280, 110, fill='tomato')

def nohy():
    platno.create_rectangle(150, 160, 170, 250, fill='purple')
    platno.create_rectangle(190, 160, 210, 250, fill='purple')

hlava()
ruky()
nohy()
telo()
```

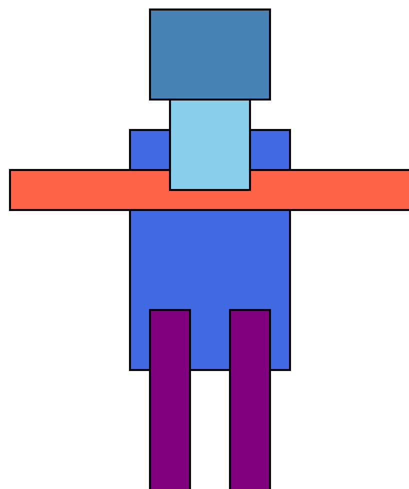
Dostaneme takúto kresbu:



Keď vymeníme záverečné poradie volaní týchto podprogramov:

```
telo()
ruky()
hlava()
nohy()
```

dostaneme trochu iný obrázok:



Úlohy

1. V 3. časti ste vytvárali program, ktorý vypísal, napr. takúto vašu vizitku:

   ```
   +--------------------+
   |             ~~~    |
   |  Juraj    / o o \  |
   |  JÁNOŠÍK  \  ~  /  |
   |             """    |
   |    IT konzultant   |
   +--------------------+
   ```

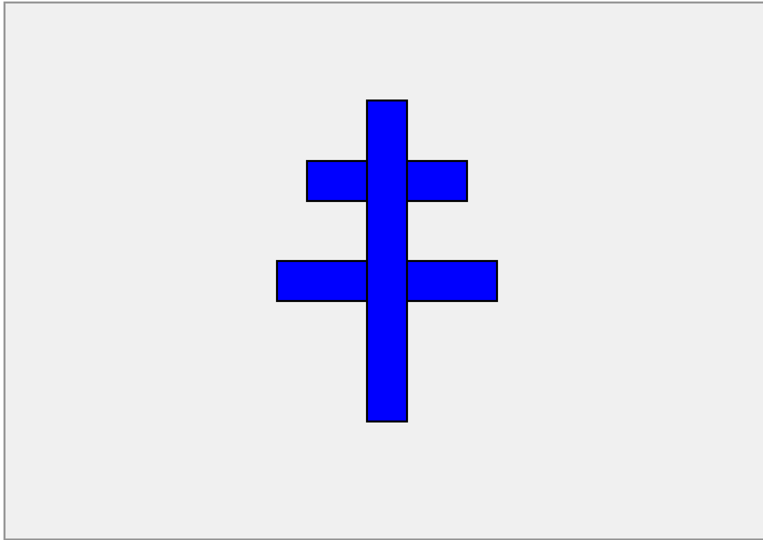
   Napíšte podprogram `vizitka` , ktorý vie vypísať takúto vizitku. Keď tento podprogram zavoláme napr. 10-krát za sebou, dostávame vypísaných 10 vizitiek.

2. Napíšte podporgram `domcek` , ktorý pomocou znakov hviezdička vypíše napr. takýto domček:

   ```
      *
     * *
    *   *
   *******
   **   **
   * * * *
   *  *  *
   * * * *
   **   **
   *******
   ```

   Volaním tohto podprogramu vypíšte pod seba 5 domčekov.

3. Napíšte podprogram `slovensky_kriz` , ktorý nakreslí dvojkríž zo Slovenského štátneho znaku pomocou troch obdĺžnikov. Napr.



Zhrnutie

čo sme sa naučili:

- podprogramy slúžia zoskupenie nejakej postupnosti príkazov, takéto podprogramy môžeme potom volať pomocou ich mena aj viackrát
- podprogramy umožňujú rozdeliť riešenie väčšej úlohy na menšie časti, pričom každú časť rieši nejaký podprogram - takéto riešenia môžu byť prehľadnejšie a lepšie čitateľné

8. Náhoda

Aby sme mohli ešte lepšie ilustrovať vytváranie nových príkazov pre kreslenie do grafickej plochy, zavedieme najprv pojem **náhodné čísla**.

Náhodné čísla

Programátori veľmi často potrebujú, aby im počítač **vygeneroval** nejaké náhodné číslo. Počítač vtedy nejako **simuluje** náhodnosť, teda „sa tvári", že si vymyslí číslo z nejakého intervalu. Keby sme potrebovali napr. interval od `1` do `6`, počítač by dokázal donekonečna vymýšľať postupnosť takýchto čísel, akokeby hádzal hracou kockou, na stranách ktorej sú čísla od 1 do 6. Generovať takúto náhodnú postupnosť je z matematického hľadiska veľmi zaujímavé, my ale budeme predpokladať, že to počítač zvláda dobre a nikdy nebudeme vedieť predpokladať, aké číslo padne nabudúce.

Náhodný generátor programátori využívajú nielen v počítačových hrách (aby sa nepriateľ správal nepredvídateľne), ale aj na simuláciu rôznych procesov (mapr. hádzanie hracou kockou) a často aj na ladenie našich programov (počítač napr. náhodne určí, kde sa nakreslí nejaký útvar).

Naučme sa teraz používať náhodný generátor v Pythone. Tak, ako sme pri grafickje ploche zapísali `import tkinter` (označuje, že chceme pracovať s grafikou), aj pri práci s náhodným generátorom musíme najprv uviesť:

```
import random
```

Týmto príkazom Python vie, že budeme pracovať s náhodnými číslami, teda dostávame k dispozícii niekoľko funkcií, ktoré dokážu generovať takéto čísla. My sa zoznámime jednou z nich `random.randint`, hoci v dokumentácii sa dajú nájsť aj ďalšie [1. Funkcia `random.randint` je taký podprogram, ktorý nič nevypisuje, nevykresľuje, ale vždy keď ho zavoláme, **vráti** nejaké náhodné číslo. Interval, z ktorého sa náhodne toto číslo vyberie, sa určuje parametrami:

```
random.randint(od, do)
```

Každé zavolanie tejto funkcie vráti náhodnú hodnotu z intervalu `od až do`. Môžeme to otestovať v interaktívnom režime:

```
>>> import random
>>> random.randint(1, 6)
3
>>> random.randint(1, 6)
1
>>> random.randint(1, 6)
6
>>> random.randint(1, 6)
1
>>> random.randint(1, 6)
1
```

Každým ďalším zavolaním tejto funkcie dostaneme nejakú náhodnú hodnotu. Samozrejme, že sa tieto hodnoty môžu opakovať rovnako, ako by sa mohli opakovať, keby sme hádzali hracou kockou (s číslami od `0` do `5`).

Napíšme teraz program, ktorý hodí dvomi kockami (dvakrát zavolá funkciu `random.randint`), potom tieto dve hodnoty vypíše aj s ich súčtom:

```
# program s nahodnymi cislami

import random

def hadz_dvomi_kockami():
    prva = random.randint(1, 6)
    druha = random.randint(1, 6)
    print('hodil si', prva, 'a', druha, 'a ich sucet je', prva + druha)

hadz_dvomi_kockami()
```

Po spustení môžeme dostať napr.

```
hodil si 1 a 4 a ich sucet je 5
```

Tento podprogram môžeme zavolať aj viackrát:

```
hadz_dvomi_kockami()
hadz_dvomi_kockami()
hadz_dvomi_kockami()
hadz_dvomi_kockami()
hadz_dvomi_kockami()
```

a po spustení môžeme vidieť napr. takýto výstup:

```
hodil si 3 a 3 a ich sucet je 6
hodil si 6 a 1 a ich sucet je 7
hodil si 1 a 2 a ich sucet je 3
hodil si 6 a 1 a ich sucet je 7
hodil si 3 a 2 a ich sucet je 5
```

Zrejme každé spustenie programu s najväčšou pravdepodobnosťou vygeneruje úplne inú päticu hodov dvomi kockami.

Grafika a náhodné čísla

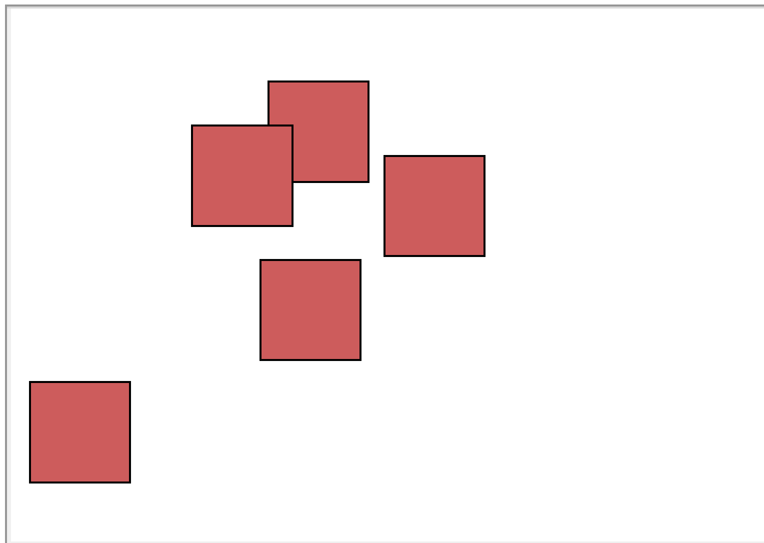
Generátor náhodných čísel teraz využijeme na vygenerovanie náhodnej pozície umiestnenia nejakého útvaru. Napíšeme podprogram `stvorec_nahodne` , ktorý nakreslí štvorec veľkosti 50x50 na náhodné pozície. Keďže predpokladáme, že grafická plocha má rozmery približne 370x260, ľavý horný vrchol náhodného štvorca nech je od (`10` , `10`) do (`310` , `200`). Teda náhodná `x` -ová súradnica je náhodné číslo od 10 do 310 a `y` -ová od 10 do 200. Použijeme funkciu `random.randint` :

```
# program nakresli pat stvorcov na nahodne pozicie

import tkinter
import random

platno = tkinter.Canvas(bg='white')
platno.pack()

def stvorec_nahodne():
    x = random.randint(10, 310)
    y = random.randint(10, 200)
    platno.create_rectangle(x, y, x+50, y+50, fill='indian red')

stvorec_nahodne()
stvorec_nahodne()
stvorec_nahodne()
stvorec_nahodne()
stvorec_nahodne()
```

Všimnite si:

- na začiatok programu sme umiestnili riadok `import tkinter` aj riadok s `import random`
- podobne ako pri hádzaní dvoch kociek, aj tu sme v podporgrame `stvorec_nahodne` vygenerovali dve náhodné hodnoty do premenných `x` a `y`, pričom každá z nich má iný interval
- aj tu sme na záver päť krát zavolali príkaz `stvorec_nahodne`, aby sme dostali 5 náhodných štvorcov

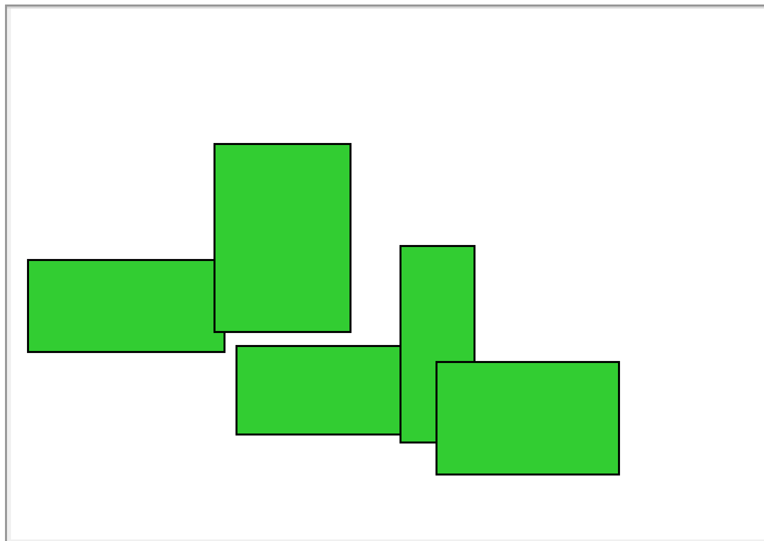
Po spustení môžeme dostať nejaký podobný obrázok:



Náhodný generátor môžeme využiť nielen na určenie náhodnej pozície, ale aj na veľkosti strán obdĺžnika. Nasledovný program je miernou modifikáciou predchádzajúceho programu, ktorý kreslil náhodné štvorce:

```
# program nakresli pat nahodne velkych obdlznikov na nahodne pozicie

import tkinter
import random

platno = tkinter.Canvas(bg='white')
platno.pack()

def nahodny_obdlznik():
    a = random.randint(10, 100)
    b = random.randint(10, 100)
    x = random.randint(10, 360-a)
    y = random.randint(10, 250-b)
    platno.create_rectangle(x, y, x+a, y+b, fill='lime green')

nahodny_obdlznik()
nahodny_obdlznik()
nahodny_obdlznik()
nahodny_obdlznik()
nahodny_obdlznik()
```

Po spustení môžeme dostať nejaký podobný obrázok:



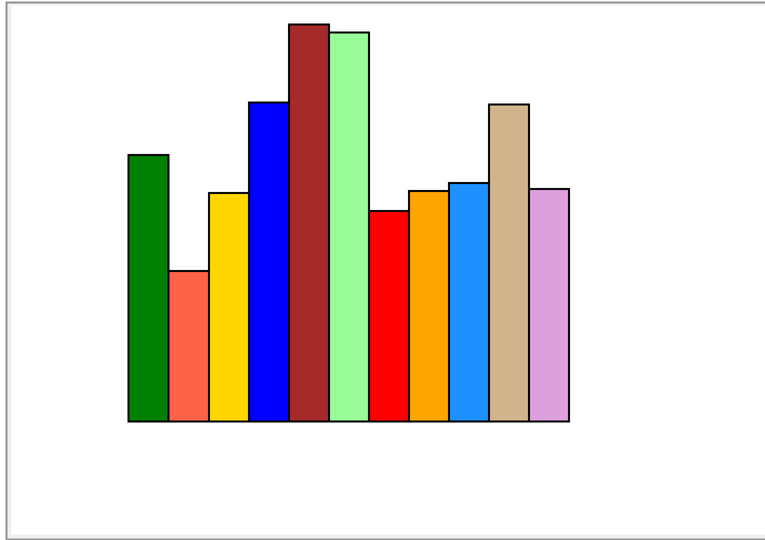
Úlohy

1. Napíšte podprogram `predpoved`, ktorý vypíše správu:

   ```
   dnes bude 15 stupnov
   ```

   v ktorej číselný údaj podprogram zvolí náhodne z intervalu <-15, 35>.

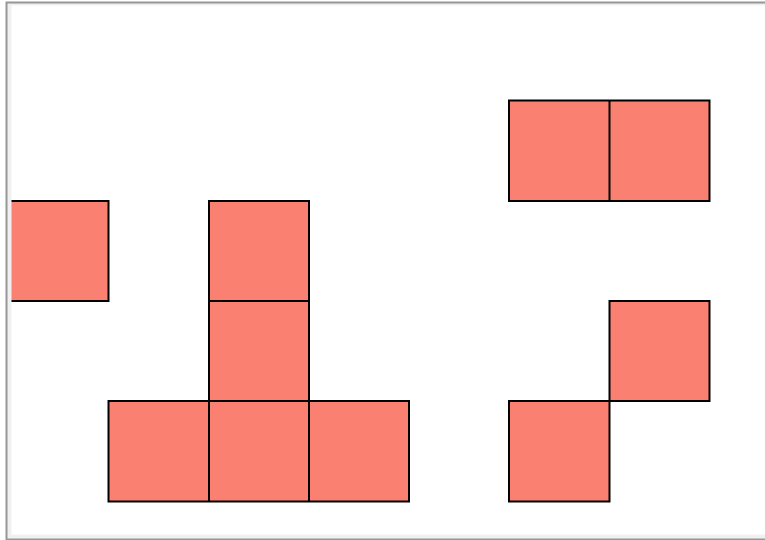
2. Napíšte program, ktorý nakreslí tesne vedľa seba 10 obdĺžnikov, ktorých šírka je 30 a výška je náhodné číslo od 10 do 200. Pre tieto obdĺžniky zvoľte rôzne farby, napr.



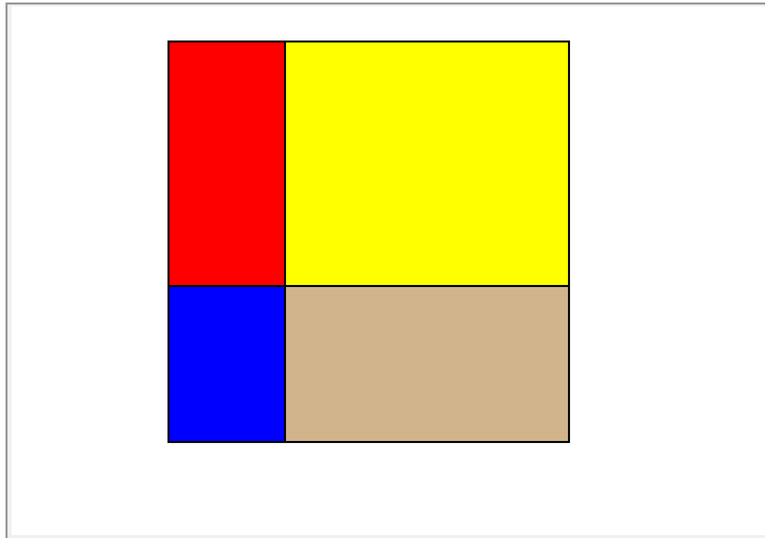
3. Napíšte podprogram, ktorý vygeneruje náhodné tri sumy za náše nákupy (náhodné číslo od 1 do 15), vypíše ich a na záver vypíše ich súčet. Napr. takto:

```
tvoj prvý nákup za 12 euro
tvoj druhý nákup za 4 euro
tvoj tretí nákup za 9 euro
spolu si zaplatil 25 euro
```

4. Podprogram `v_sieti` na náhodné políčko v sieti, v ktorej sú štvorčeky 50x50, položí štvorček. Po 10 zavolaniach to môže vyzerať napr. takto:



5. Napíšte program, ktorý najprv zvolí dve náhodné čísla `a` a `b` s hodnotami od 10 do 190 a potom rozdelí štvorec 200x200 na štyri časti (obdĺžniky) pričom ľavá horná časť má rozmery `a x b` a pravá spodná časť má rozmery `200-a x 200-b`. Každý z týchto obd=lžnikov zafarbite inou farbou, napr. takto:



Zhrnutie

čo sme sa naučili:

- náhodné čísla sú pre informatikov, ale aj matematikov, užitočným nástrojom, pomocou ktorého dokážeme simulovať rôzne náhodné javy, napr. hádzanie kockou

---

[1] Ďalšou podobne užitočnou funkciou z modulu `random` je `random.randrange`. Je na učiteľovi, ktorú z týchto dvoch použije. Na začiatok nie je vhodné uvádzať obe. Funkcia `random.randrange` môže mať jeden alebo dva parametre: `random.randrange(hranica)` alebo `random.randrange(od, hranica)`. Teraz bude generovať náhodné čísla z intervalu < `od` , `hranica-1` >.

9. Text v grafickej ploche

Zatiaľ sme sa naučili jediný grafický príkaz:

```
platno.create_rectangle(x1, y1, x2, y2, fill='farba', outline='farba', width=číslo)
```

pomocou ktorého vieme do grafickej plochy (presnejšie na `platno` grafickej plochy) nakresliť obdĺžnik. Tento je určený:

- dvomi protiľahlými vrcholmi (najčastešie pomocou ľavého horného (`x1` , `y1`) a pravého dolného (`x2` , `y2`) vrchola),
- farbou výplne (`fill`),
- farbou obrysu (`outline`),
- hrúbkou čiary obrysu (`width`).

Príkaz na vypisovanie textu

Už sme naznačili, že príkazov, ktoré kreslia nejaké útvary do grafickej plochy je oveľa viac. Všetky majú tvar:

```
platno.create_útvar(parametre)
```

kde slovo `útvar` sa nahradí špecifickým menom grafického útvaru. Okrem `rectangle` si teraz ukážeme grafický útvar `text` . Takýto útvar je grafickou reprezentáciou nejakého znakového reťazca, t.j. do grafickej plochy môžeme umiestniť na ľubovoľnú pozíciu nejaký text. Pozrime najjednoduchšiu verziu tohto príkazu:

```
platno.create_text(x, y, text='znakový reťazec')
```

Na dané súradnice (`x` , `y`) sa vypíše zadaný znakový reťazec. Pozrime tento program:

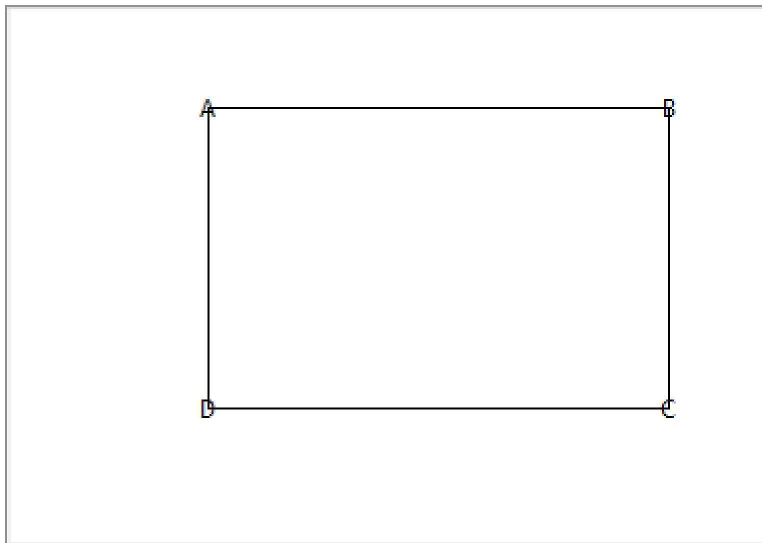
```
import tkinter

platno = tkinter.Canvas(bg='white')
platno.pack()

platno.create_text(180, 20, text='Horný okraj')
platno.create_text(180, 230, text='Dolný okraj')
platno.create_text(310, 120, text='Pravý okraj')
platno.create_text(50, 120, text='Ľavý okraj')
```

Program vytvorí štyri grafické útvary, ktoré sú textami - budeme hovoriť, že program vypíše do grafickej plochy štyri texty. Dostávame takýto obrázok:

Využime vypisovanie textu na označovanie geometrických útvarov. Nakreslíme obdĺžnik (na súradnice `x1` , `y1` , `x2` , `y2`) a potom všetky vrcholy obdĺžnika označíme písmenami `A` , `B` , `C` , `D` . Konkrétne, nech ľavý horný vrchol (`x1` , `y1`) má označenie `A` , pravý horný vrchol (`x2` , `y1`) má označenie `B` , pravý dolný vrchol (`x2` , `y2`) má označenie `C``a ľavý dolný vrchol (``x1` , `y2` ) má označenie `D` :

```
import tkinter

platno = tkinter.Canvas(bg='white')
platno.pack()

x1 = 100
y1 = 50
x2 = 330
y2 = 200
platno.create_rectangle(x1, y1, x2, y2)
platno.create_text(x1, y1, text='A')
platno.create_text(x2, y1, text='B')
platno.create_text(x2, y2, text='C')
platno.create_text(x1, y2, text='D')
```

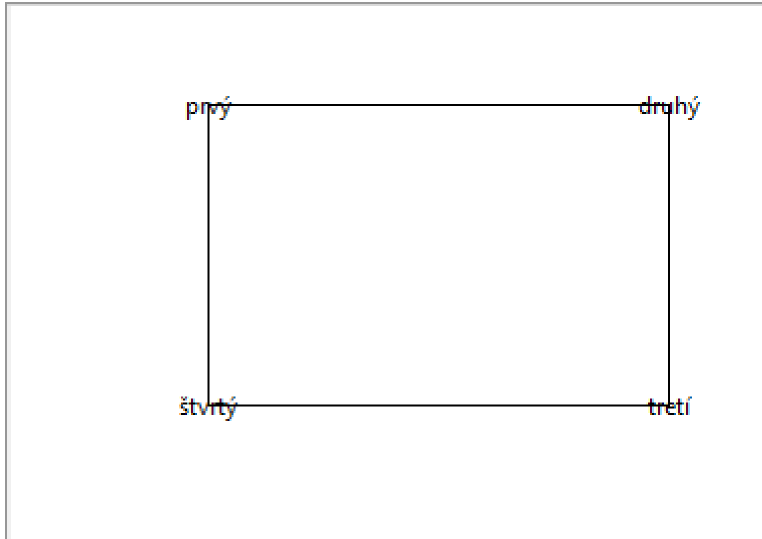
Po spustení vidíme obdĺžnik aj s menami vrcholov:



Vidíme dôležitú vlastnosť vypisovaných textov do grafickej plochy: text sa vypíše tak, že (`x` , `y`) určuje jeho **stred**. Keď namiesto písmen `ABCD` použijeme dlhšie slová:

```
platno.create_text(x1, y1, text='prvý')
platno.create_text(x2, y1, text='druhý')
platno.create_text(x2, y2, text='tretí')
platno.create_text(x1, y2, text='štvrtý')
```

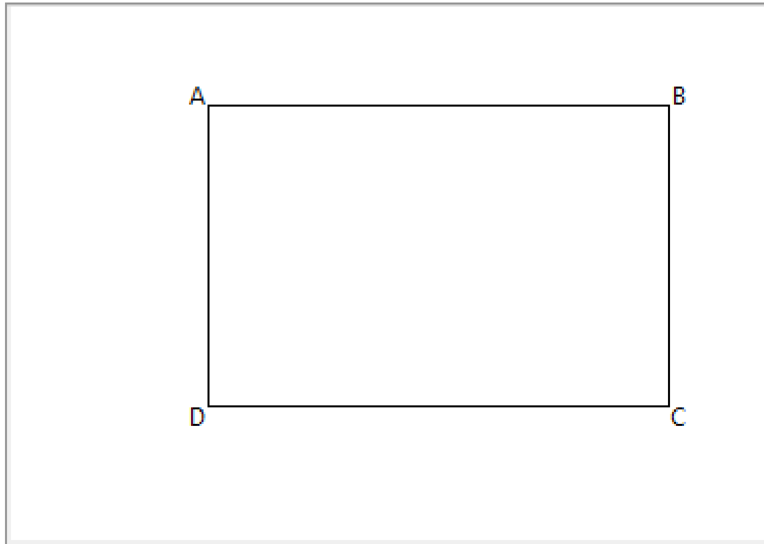
Teraz ešte názornejšie vidíme, že presný stred každého z týchto slov sa nachádza v niektorom z vrcholov obdĺžnika:



Ak by sme chceli pomenovať vrcholy obdĺžnika písmenami `ABCD` ale tak, aby tieto označenia boli čitateľné, musíme každý z týchto textov posunúť nejakým smerom, napr.

```
platno.create_text(x1-5, y1-5, text='A')
platno.create_text(x2+5, y1-5, text='B')
platno.create_text(x2+5, y2+5, text='C')
platno.create_text(x1-5, y2+5, text='D')
```

Ktorým smerom a o koľko posuniete `x` -ovú a ktorým `y` -ovú súradnicu, záleží len od vás a od veľkosti vypisovaného textu. Náš program teraz vykreslí:



Môžete vyskúšať rôzne smery posúvania.

Úlohy

1. Vypísať riadky nejakej známej básničky alebo pesničky. Napr.

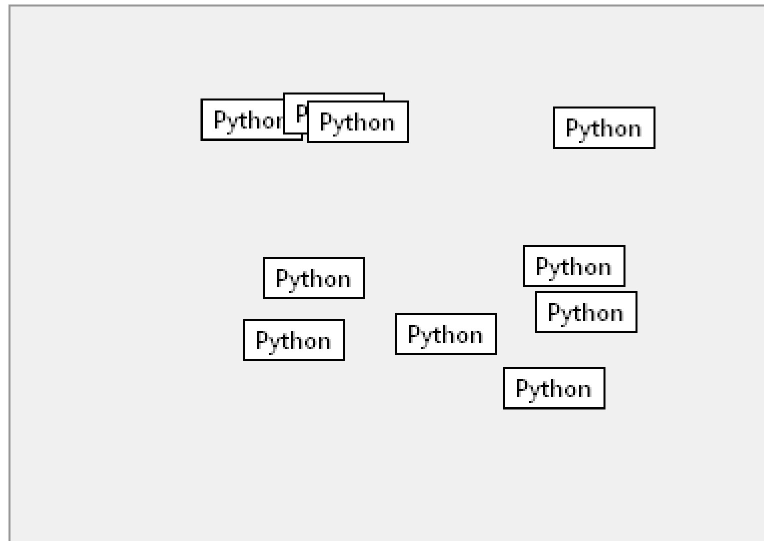
   Išiel Macek do Malacek,

   šošovicku mlácic,

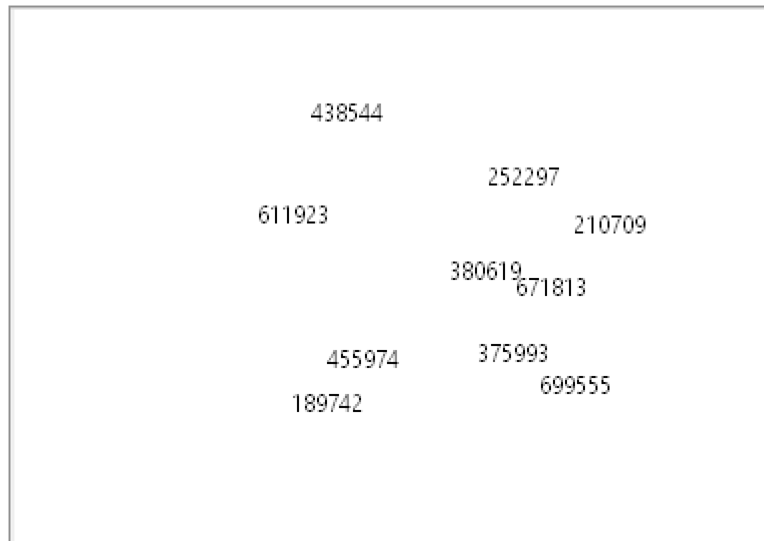
   zabudol si cepy doma,

   mosel sa on vrácic.

2. Podprogram `stitok` na náhodnú pozíciu v grafickej ploche nakreslí biely obdĺžnik veľkosti 50 x 20 a do jeho stredu napíše nejaký text, napr. Python. Po desiatich zavolaniach podprogramu môžeme dostať napr.



3. Napíšte podprogram `nahodne_cislo`, ktorý na náhodnú pozíciu v grafickej ploche vypíše náhodné šesťciferné číslo (teda číslo z intervalu od 100000 do 999999). Po niekoľkých zavolaniach podprogramu môžeme dostať napr.



Zhrnutie

čo sme sa naučili:

- do grafickej plochy môžeme okrem obdĺžnikov zapisovať aj texty, využijeme príkaz `create_text`
- pozíciu vypisovaného textu do grafickej plochy určujeme súradnicami stredu tohoto textu

10. Fonty a farby pre texty v grafike

Nastavenie fontu

Zatiaľ boli všetky naše vypísané texty do grafickej plochy veľmi maličké a je načase sa naučiť meniť veľkosť aj tvar písma. Z textových editorov už iste viete, že v textových dekumentoch môžete nastavovať, tzv. **písmo** (napr. **Arial**), jeho **rez** (napr. **tučné**, tzv. **bold**) a **veľkosť** (napr. **16**). Zvykne sa tomuto hovoriť aj **font**. Pri vypisovaní textu do grafickej plochy môžeme zadať aj parameter `font`, ktorým určíme nielen meno písma, ale aj jeho veľkosť. Uvidíme,že pritom môžeme nastaviť aj rez (napr. **bold** alebo **italic**).

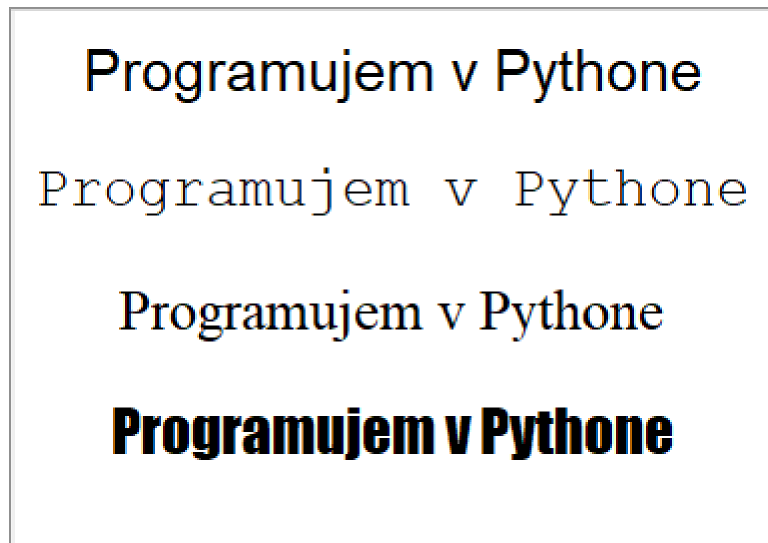
Meno písma závisí od oepračného systému (napr. MS Windows má inú sadu, ako Linux a MacOS) hoci existuje malá množina písiem, ktoré budú fungovať v rôznych operačných systémoch veľmi podobne. Vy si môžete vo svojich programoch zvoliť ľubovoľné meno písma, zatiaľ ale platí jedno dôležité obmedzenie: meno písma sa musí skladať len z jedného slova, preto nebude fungovať napr. `'Times New Roman'`. My budeme pracovať s týmito menami písiem:

- `Arial`
- `Courier`
- `Times`
- `Impact`

Pre ilustráciu vypíšeme rovnaký text `'Programujem v Pythone'` v rôznych fontoch:

```
import tkinter

platno = tkinter.Canvas(bg='white')
platno.pack()

platno.create_text(190, 30, text='Programujem v Pythone', font='Arial 22')
platno.create_text(190, 90, text='Programujem v Pythone', font='Courier 22')
platno.create_text(190, 150, text='Programujem v Pythone', font='Times 22')
platno.create_text(190, 210, text='Programujem v Pythone', font='Impact 22')
```

Vidíte rozdiely medzi týmito písmami:



Veľkosť písma budeme väčšinou voliť podľa situácie, napr. ak chceme, aby sa nám text `'Python'` zmestil celý do grafickej plochy, zvolíme:

```
import tkinter

platno = tkinter.Canvas(bg='white')
platno.pack()

platno.create_text(190, 130, text='Python', font='Arial 90')
```

a dostávame:

Farba písma

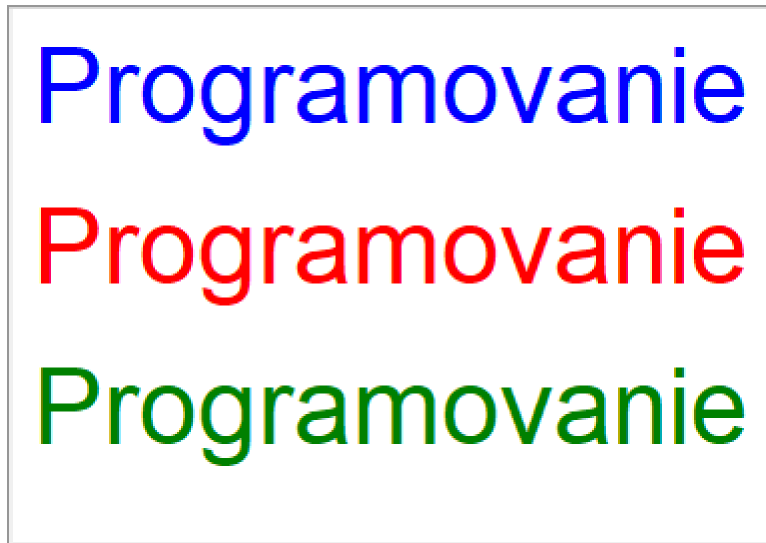
Už sme sa naučili vyfarbovať vnútro obdĺžnikov ale aj ich obrys. Vieme, že farby sa najlepšie nastavujú pomocou ich mien a pre obdĺžniky sa to robí napr. takto:

```
platno.create_rectangle(50, 50, 250, 200, fill='red')
```

Pre farbu písma je to veľmi podobné: pomocou parametra `fill` môžeme nastaviť ľubovoľnú farbu vypisovaného textu. Napr. nasledovný program vypíše pod seba rovnaký text v rôznych farbách:

```
import tkinter

platno = tkinter.Canvas(bg='white')
platno.pack()

platno.create_text(190, 40, text='Programovanie', font='Arial 40', fill='blue')
platno.create_text(190, 120, text='Programovanie', font='Arial 40', fill='red')
platno.create_text(190, 200, text='Programovanie', font='Arial 40', fill='green')
```

dostávame:



Zrejme bude veľmi závisieť od pozadia, na ktoré vypisujeme text s danou farbou. Predchádzajúci program doplníme o pozadie pre každý z textov:

```
import tkinter

platno = tkinter.Canvas(bg='white')
platno.pack()

platno.create_rectangle(10, 10, 370, 70, fill='medium blue', width=0)
platno.create_text(190, 40, text='Programovanie', font='Arial 40', fill='blue')
platno.create_rectangle(10, 90, 370, 150, fill='orange red', width=0)
platno.create_text(190, 120, text='Programovanie', font='Arial 40', fill='red')
platno.create_rectangle(10, 170, 370, 230, fill='forest green', width=0)
platno.create_text(190, 200, text='Programovanie', font='Arial 40', fill='green')
```

a keďže sme úmyselne vybrali také farby, aby boli blízke k farbe textu, dostávame:

Na tomto príklade vidíte, že niekedy úmyselne pod vypisované texty vkladáme obdĺžniky tak, aby sme zvýraznili nejaký text, alebo jeho časť.

Veľmi efektným je k danému textu vytváranie jeho tieňa. Napr. takéto tiene:

vzniknú týmto programom:

```
import tkinter

platno = tkinter.Canvas(bg='white')
platno.pack()

x = 190
y = 60
platno.create_text(x+10, y-5, text='Python', font='Arial 60', fill='light gray')
platno.create_text(x, y, text='Python', font='Arial 60', fill='blue')
x = 190
y = 180
platno.create_text(x+8, y+4, text='Python', font='Times 70', fill='light gray')
platno.create_text(x, y, text='Python', font='Times 70', fill='green')
```

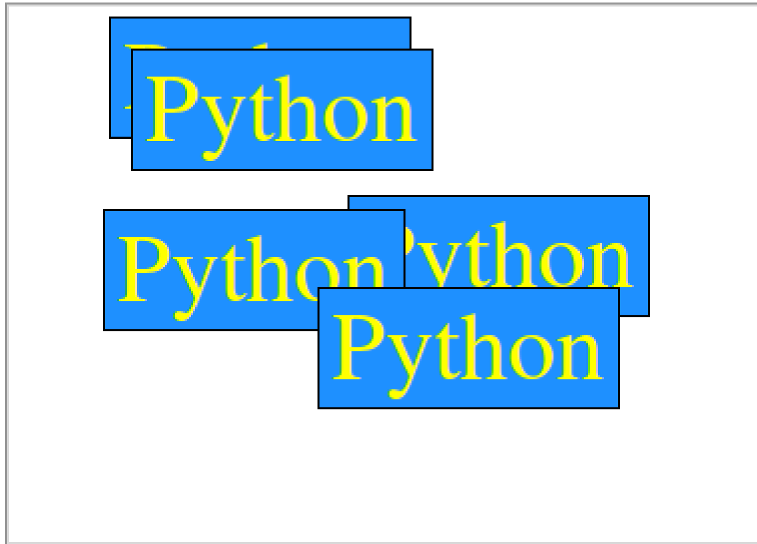
Dalo sa to aj bez premenných `x` a `y`, ale s nimi je to asi názornejšie, ako vznikli tieto tiene. Zrejme závisí na poradí v akom sa najprv nakreslí posunutý tieň a potom ten istý už zafarbený text.

Text v rámiku

Napíšeme podprogram `karticka_nahodne` , ktorý na náhodnú pozíciu napíše nejaký text, napr. `'Python'` , lenže tento text sa bude nachádzať vo farebnom obdĺžniku. Použijeme ideu podprogramu `stvorec_nahodne` :

```
import tkinter
import random

platno = tkinter.Canvas(bg='white')
platno.pack()

def karticka_nahodne():
    x = random.randint(75, 300)
    y = random.randint(30, 210)
    platno.create_rectangle(x-75, y-30, x+75, y+30, fill='dodger blue')
    platno.create_text(x, y, text='Python', fill='yellow', font='Times 30')

karticka_nahodne()
karticka_nahodne()
karticka_nahodne()
karticka_nahodne()
karticka_nahodne()
```

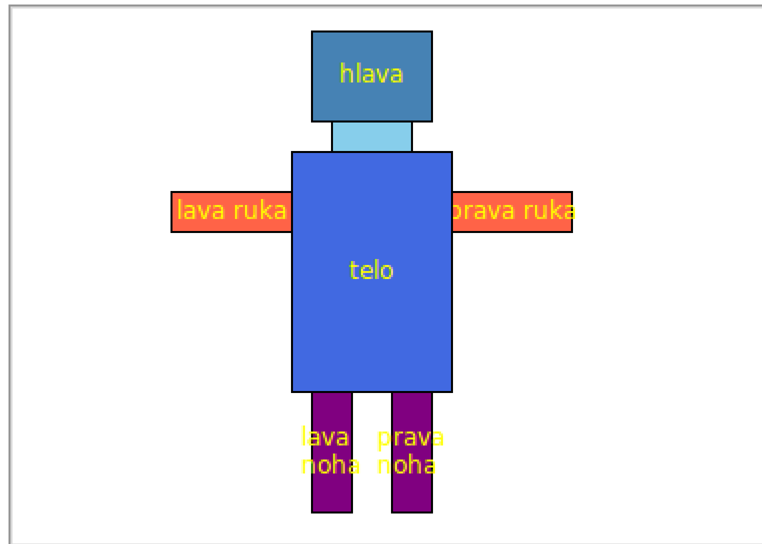
Po spustení dostaneme:



Asi si uvedomujete, že veľkosť rámika, resp. farebného obdĺžnika, bude závisieť od veľkosti textu a hlavne od veľkosti použitého písma.

Úlohy

1. Do programu z predchádzajúcej časti, ktorý kreslil robota, dopíšte do všetkých podprogramov volania `create_text` tak, aby sa v každej časti robota objavil aj text, teda `'hlava'` , `'telo'` , `'lava ruka'` , `'prava ruka'` , `'lava noha'` , `'prava noha'` , napr.



2. Napíšte program, ktorý zoberie nejaký konkrétny text (napr. báseň, alebo slová piesne) a vypíše ho do grafickej plochy tak, že riadky textu pritom rôzne zafarbíte. Napr.

3. Navrhnite si vlastnú vizitku: zvoľte si farby obdĺžnika a textu, vyberte si nejaké zaujímavé písmo. Napr.

Zhrnutie

čo sme sa naučili:

- do grafickej plochy môžeme okrem obdĺžnikov pomocou `create_text` zapisovať aj texty
- ak chceme zmeniť veľkosť a tvar písma, musíme nastaviť parameter `font`
- farbu textu nastavujeme parametrom `fill`

11. Program s opakovaním

Tento program paťkrát vypisuje tú istú dvojicu riadkov:

```
print('Programujem v Pythone')
print('=====================')
print('Programujem v Pythone')
print('=====================')
print('Programujem v Pythone')
print('=====================')
print('Programujem v Pythone')
print('=====================')
print('Programujem v Pythone')
print('=====================')
```

Pripomeňme program, ktorý päťkrát zavolá podprogram a ten kreslí štvorček na náhodnú pozíciu:

```
import tkinter
import random

platno = tkinter.Canvas(bg='white')
platno.pack()

def stvorec_nahodne():
    x = random.randint(10, 310)
    y = random.randint(10, 200)
    platno.create_rectangle(x, y, x+50, y+50, fill='indian red')

stvorec_nahodne()
stvorec_nahodne()
stvorec_nahodne()
stvorec_nahodne()
stvorec_nahodne()
```

V oboch predchádzajúcich príkladoch sa opakuje vykonávanie jedného alebo viacerých príkazov. V programovaní je veľmi bežné, že sa nejaké výpočty, výpisy alebo kreslenia útvarov **opakujú viackrát** za sebou. Ak sa to opakuje málokrát, napr. len dvakrát alebo trikrát, nie je problém, tú istú časť skopírovať viackrát za sebou. Iná situácia je, keď sa niečo opakujeveľmi veľakrát, pripadne počet opakovaní závisí od nejakej premennej. Programátori dobre vedia, že na tieto situácie sa v každom programovacom jazyku nachádza **príkaz opakovania**.

Príkaz opakovania s daným počtom opakovaní

Naučíme sa nový príkaz v Pythone, ktorý umožní opakovať nejakú časť programu (tzv. **blok príkazov**) viackrát. Tento príkaz má takáto tvar:

```
for premenná in range(počet):
    príkaz
    príkaz
    ...
```

Aj v tejto programovej konštrukcii, podobne ako pri definovaní nového príkazu, odsunutý blok príkazov určuje **čo sa bude opakovať** (opäť riadky bloku odsúvame o 4 znaky vpravo) a v hlavičke cyklu sa udáva, ktorá premenná sa pri tom použije ako tzv. **počítadlo cyklu**. V tomto počítadle, teda **premennej cyklu** Python nastavuje (počítaz), koľko krát sa už cyklus doteraz vykonal. Výraz `range(počet)` určuje **rozsah** číselných hodnôt, pre ktorý bude tento cyklus bežať.

Najlepšie to ukážeme na predchádzajúcich príkladoch, ktoré prepíšeme s použitím tzv. **for-cyklu**.

Prvá ukážka

V prvok príklade sa päťkrát za sebou opakuje táto dvojica príkazov:

```
print('Programujem v Pythone')
print('=====================')
```

Keď tieto dva príkazy vložíme ako **telo cyklu** a opakovanie (teda `range`) nastavíme na 5, dostaneme:

```
for i in range(5):
    print('Programujem v Pythone')
    print('=====================')
```

Program teraz vypíše týchto 10 riadkov:

```
Programujem v Pythone
=====================
Programujem v Pythone
=====================
Programujem v Pythone
=====================
Programujem v Pythone
=====================
Programujem v Pythone
=====================
```

Python tu opakoval celý blok príkazov (tzv. **telo cyklu**) päťkrát. Premennú cyklu sme tu nazvali menom `i`. Programátori veľmi často (keď nemajú fantáziu) nazývajú premennú cyklu písmenami ´´i´´ alebo `j` alebo `k`. Neskôr uvidíme aj iné názvy pre premenné cyklu.

Uvedomte si, že blok príkazov končí vtedy, keď nasleduje príkaz, ktorý už nie je odsunutý. Zmeňme predchádzajúci program:

```
for i in range(5):
    print('Programujem v Pythone')
print('=====================')
```

V tomto prípade telo cyklu obsahuje jediný riadok - príkaz `print('Programujem v Pythone')` a riadok s príkazom `print('=====================')` je už **mimo cyklu**. Tento program päťkrát zopakuje jediný riadok vypisu a na záver vypíše riadok so znakmi rovná sa:

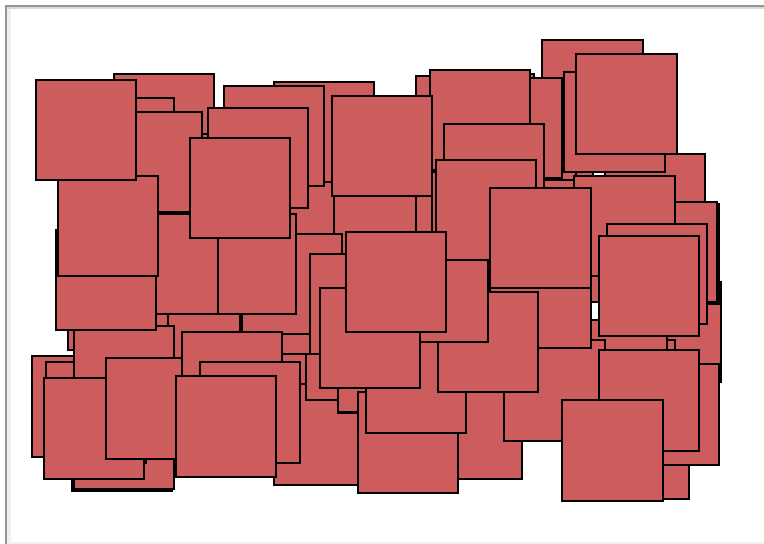
```
Programujem v Pythone
Programujem v Pythone
Programujem v Pythone
Programujem v Pythone
Programujem v Pythone
=====================
```

Druhá ukážka

V druhom programe sa najprv definoval podprogram `stvorec_nahodne` a potom sa tento podprogram zavolal päťkrát. Aj v tomto príklade päť volaní `stvorec_nahodne()` nahradíme for-cyklom:

```
import tkinter
import random

platno = tkinter.Canvas(bg='white')
platno.pack()

def stvorec_nahodne():
    x = random.randint(10, 310)
    y = random.randint(10, 200)
    platno.create_rectangle(x, y, x+50, y+50, fill='indian red')

for i in range(100):
    stvorec_nahodne()
```

Všimnite si, že sme päť zavolaní podprogramu nahradili číslo `100`. Program nakreslí 100 malých farebných štvorčekov na náhodné pozície. Dostaneme:



Keby sme nepoznali for-cyklus, bol by tento program veľmi dlhý: asi by sme 100-krát pod seba zapisovali volanie `stvorec_nahodne()`.

Premenná cyklu

Táto premenná pri každom prechode cyklu postupne nadobúda číselné hodnoty z rozsahu, ktorý je daný hodnotou **počet**.

Otestujeme to týmto krátkym programom:

```
print('zaciatok')
for i in range(4):
    print(i, 'test')
print('koniec')
```

Tento program najprv vypíše reťazec `zaciatok`, potom vo for-cykle niekoľko krát jeden `print` s premennou `i` a na záver po skončení cyklu reťazec `koniec`:

```
zaciatok
0 test
1 test
2 test
3 test
koniec
```

Na tomto príklade vidíme, že príkaz `print(i, 'test')` sa vykonal presne štyrikrát a pri každom prechode cyklu premenná (počítadlo) `i` postupne nadobudla hodnoty `0`, `1`, `2`, `3`, teda sú to postupne všetky čísla od `0` až do `počet-1`. Tento program by sme mohli prepísať bez cyklu takto:

```
print('zaciatok')
i = 0
print(i, 'test')
i = 1
print(i, 'test')
i = 2
print(i, 'test')
i = 3
print(i, 'test')
print('koniec')
```

Zrejme by sme po spustení dostali rovnaký výsledok, ako verzia s for-cyklom. Vidíme, že zápis `range(4)` označuje postupnosť hodnôt pre premennú cyklu `i`, ktorá začína `0` a končí o jedna menším číslom ako `4`.

Pozrime tento ďalší príklad:

```
cislo = 0
for i in range(5):
    cislo = cislo + i * i
print('vysledok =', cislo)
```

Vidíme, že v cykle sa nebude nič vypisovať, ale bude sa päťkrát priraďovať do tej istej premennej. Aj tejto for-cyklus vieme rozpísať na niekoľko príkazov bez použitia cyklu:

```
cislo = 0
i = 0
cislo = cislo + i * i
i = 1
cislo = cislo + i * i
i = 2
cislo = cislo + i * i
i = 3
cislo = cislo + i * i
i = 4
cislo = cislo + i * i
print('vysledok =', cislo)
```

Pozorne pozrime na jeden z priraďovacích príkazov, napr.

```
cislo = cislo + i * i
```

Ak by to bola matematická rovnica, tak to pre konkrétne `i` nedáva veľký zmysel (napr. `cislo = cislo + 1`). Lenže v tomto prípade to označuje **príkaz priradenia**: najprv sa vypočíta hodnota výrazu na pravej strane za `=`, teda `cislo + i * i` a až potom sa táto hodnota priradí do premennej `cislo`. Tým sa zmení pôvodná hodnota tejto premennej. Poďme tento program postupne **odkrokovať** (niekedy hovoríme aj **trasovať**): ku každému riadku programu pripíšeme obsahy premenných a výrazov:

Ttrasovacia tabuľka

príkaz	**`cislo`**	**`i`**	**`i*i`**	**výpis**
`cislo = 0`	0			
`i = 0`		0	0	
`cislo = cislo + i * i`	0			
`i = 1`		1	1	

príkaz	cislo	i	i*i	výpis
`cislo = cislo + i * i`	1			
`i = 2`		2	4	
`cislo = cislo + i * i`	5			
`i = 3`		3	9	
`cislo = cislo + i * i`	14			
`i = 4`		4	16	
`cislo = cislo + i * i`	30			
`print('vysledok =', cislo)`				`vysledok = 30`

Vidíme, že priraďovací príkaz `cislo = cislo + hodnota` pripočíta danú hodnotu k doterajšiemu obsahu premennej `cislo`. Budeme hovoriť, že toto priradenie zvyšuje hodnotu danej premennej (inkrementuje). Z trasovacej tabuľky si môžeme dovoliť povedať, že daný program počíta súčty druhých mocnín čísel od `0` do `4`.

Úlohy

1. Zistite, aký výsledok dostaneme pre:

```
cislo = 0
for i in range(11):
    cislo = cislo + i * i
print('vysledok =', cislo)
```

2. Program z predchádzajúcej úlohy počíta `cislo = 0*0 + 1*1 + 2*2 + 3*3 + 4*4 + ... + 10*10`. Ako dlho by vám to trvalo,keby ste to počítali ručne, prípadne s použitím obyčajnej kalkulačky?

3. Ako treba opraviť tento program, aby počítal súčty druhých mocním až do 100? Opravte program a zistite výsledok.

Zhrnutie

čo sme sa naučili:

- pomocou príkazu `for` môžeme opakovať vykonávanie jedného alebo viacerých príkazov (tzv. **blok príkazov**)
- všetky riadky bloku príkazov musia byť odsunuté o 4 medzery vpravo
- for-cyklus používa premennú cyklu, ktorá pri každom prechode postupne nadobúda hodnoty s udanej postupnosti pomocou `range`

dôležité zásady:

- meno premennej cyklu zvoľte tak, aby čo najlepšie vyjadrovalo hodnotu, ktorá sa v cykle bude automaticky meniť

12. Riešenie úloh s for-cyklom

Príkaz opakovania s vymenovanými hodnotami

Ukážeme aj iný spôsob opakovania príkazov pomocou for-cyklu.

V tomto program opakujeme vykonávanie jedného príkazu `print` s rôznymi hodnotami premennej `n`:

```
n = 11
print(n, '*', n, '=', n*n)
n = 111
print(n, '*', n, '=', n*n)
n = 1111
print(n, '*', n, '=', n*n)
n = 11111
print(n, '*', n, '=', n*n)
n = 111111
print(n, '*', n, '=', n*n)
```

Dostávame takýto výpis:

```
11 * 11 = 121
111 * 111 = 12321
1111 * 1111 = 1234321
11111 * 11111 = 123454321
111111 * 111111 = 12345654321
```

Zrejme príkaz **for-cyklu**, tak ako ho poznáme, tu teraz nevyužijeme. Doteraz sme poznali tento cyklus tak, že opakoval jedne alebo viac príkazov (blok príkazov), pričom premenná cyklu nadobúdala hodnoty z nejakého rozsahu od `0` do `počet-1`. Lenže tu by sme potrebovali, aby premenná cyklu namiesto toho nadobúdala postupne hodnoty `11`, `111`, `1111`, `11111`, `111111`.

Príkaz **for-cyklu** môže mať v Pythone ešte aj takýto tvar:

```
for premenná in postupnosť_hodnôt:
    príkaz
    príkaz
    ...
```

Aj v tomto prípade sa bude niekoľko krát opakovať vykonávanie odsunutého bloku príkazov, pričom **premenná cyklu** bude nadobúdať hodnoty z danej postupnosti.

V našom programe sa opakuje vykonávanie jediného riadku:

```
print(n, '*', n, '=', n*n)
```

pre postupnosť `11`, `111`, `1111`, `11111`, `111111`, čo môžeme elegantne zapísať:

```
for n in 11, 111, 1111, 11111, 111111:
    print(n, '*', n, '=', n*n)
```

Teda nemiesto `range(počet)` môžeme jednoducho vymenovať hodnoty, ktoré chceme, aby premenná cyklu nadobudla.

Teraz trochu pozmeňme program, ktorý počítal súčty druhých mocnín, takto:

```
cislo = 0
for i in 2, 3, 5, 7, 11, 13, 17:
    cislo = cislo + i
print('vysledok =', cislo)
```

Mohli by ste tento program prečítať ako výpočet súčtu niekoľkých čísel - náhodou sú to prvočísla do `17`. Výsledok bude:

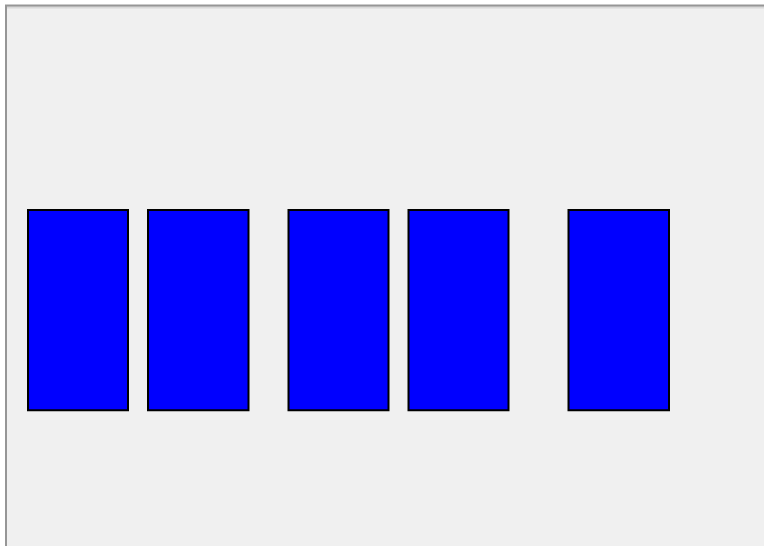
```
vysledok = 58
```

Takúto konštrukciu využijeme najčastejšie, keď budeme potrebovať vykonať nejaké príkazy (výpočty) pre nejakú konkrétnu skupinu hodnôt.

Ďalší program kreslí 5 obdĺžnikov, pričom sa im mení iba ich `x`-ová súradnica:

```
import tkinter

platno = tkinter.Canvas()
platno.pack()

x = 10
platno.create_rectangle(x, 100, x+50, 200, fill='blue')
x = 70
platno.create_rectangle(x, 100, x+50, 200, fill='blue')
x = 140
platno.create_rectangle(x, 100, x+50, 200, fill='blue')
x = 200
platno.create_rectangle(x, 100, x+50, 200, fill='blue')
x = 280
platno.create_rectangle(x, 100, x+50, 200, fill='blue')
```

Dostávame takúto kresbu:



Aj v tomto programe sa opakuje len jeden príkaz:

```
platno.create_rectangle(x, 100, x+50, 200, fill='blue')
```

pričom sa mení hodnota premennej `x`. Aj z nej urobíme **premennú cyklu** a zadefinujeme aj postupnosť hodnôt:

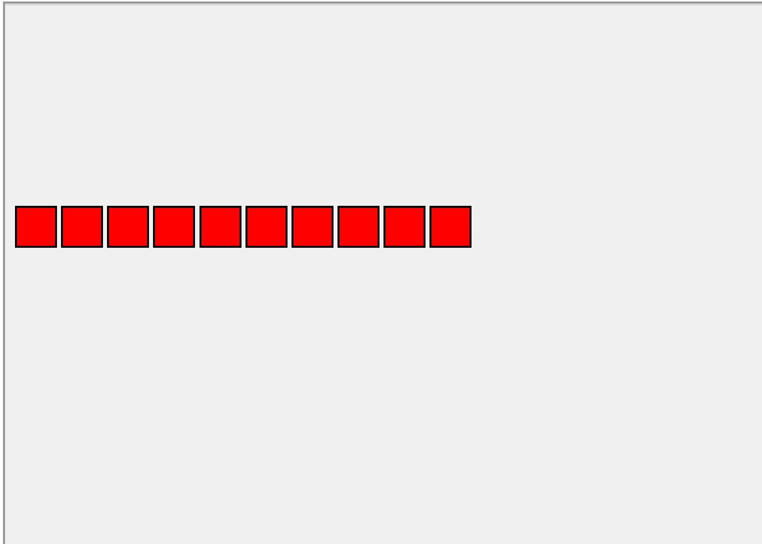
```
import tkinter

platno = tkinter.Canvas()
platno.pack()

for x in 10, 70, 140, 200, 280:
    platno.create_rectangle(x, 100, x+50, 200, fill='blue')
```

Vo všetkých týchto programoch vidíte spôsob, ako môžeme z postupnosti príkazov vyrobiť cyklus: musíme najprv odhaliť, aká časť programu sa opakuje a ako pritom využijeme premennú cyklu. Tiež sa pritom musíme rozhodnúť, či použijeme vymenovanie hodnôt, alebo zápis `range(počet)`.

Ukážme teraz riešenie úlohy, v ktorej chceme vedľa seba nakresliť 10 štvorčekov so stranou `20`. Medzi samotnými štvorčekmi ale chceme malú medzeru veľkosti `3`. Riešenie bez cyklu by mohlo vyzerať napr. takto:

```
import tkinter

platno = tkinter.Canvas()
platno.pack()

y = 100
x = 5
platno.create_rectangle(x, y, x+20, y+20, fill='red')
x = 28
platno.create_rectangle(x, y, x+20, y+20, fill='red')
x = 51
platno.create_rectangle(x, y, x+20, y+20, fill='red')
x = 74
platno.create_rectangle(x, y, x+20, y+20, fill='red')
x = 97
platno.create_rectangle(x, y, x+20, y+20, fill='red')
x = 120
platno.create_rectangle(x, y, x+20, y+20, fill='red')
x = 143
platno.create_rectangle(x, y, x+20, y+20, fill='red')
x = 166
platno.create_rectangle(x, y, x+20, y+20, fill='red')
x = 189
platno.create_rectangle(x, y, x+20, y+20, fill='red')
x = 212
platno.create_rectangle(x, y, x+20, y+20, fill='red')
```

a grafická plocha vyzerá takto:



Prerobiť to na **for-cyklus** s vymenovaním hodnôt je už teraz hračka:

```
import tkinter

platno = tkinter.Canvas()
platno.pack()

y = 100
for x in 5, 28, 51, 74, 97, 120, 143, 166, 189, 212:
    platno.create_rectangle(x, y, x+20, y+20, fill='red')
```

Ale už to nebude tak jednoduché, keby sme chceli využiť funkciu `range`, do ktorej chceme nastaviť počet kreslených štvorčekov.

Vráťme sa k pôvodnej verzii bez cyklu a pozrime, ako sa mení hodnota premennej `x`:

```
x = 5
platno.create_rectangle(x, y, x+20, y+20, fill='red')
x = 28
platno.create_rectangle(x, y, x+20, y+20, fill='red')
x = 51
platno.create_rectangle(x, y, x+20, y+20, fill='red')
x = 74
...
```

Každá ďalšia hodnota `x` je presne o `23` väčšia ako predchádzajúca (`20` je veľkosť štvorčeka a `3` je medzera medzi nimi). Teda platí, že:

```
nové_x = staré_x + 23
```

Už vieme, že to môžeme zapísať:

```
x = x + 23
```

Teda vypočítaj hodnotu pravej strany priraďovacieho príkazu `x + 23` a túto novú hodnotu priraď späť do premennej `x` . Takže časť programu bez cyklu môžeme zapásať:

```
x = 5
platno.create_rectangle(x, y, x+20, y+20, fill='red')
x = x + 23
platno.create_rectangle(x, y, x+20, y+20, fill='red')
x = x + 23
platno.create_rectangle(x, y, x+20, y+20, fill='red')
x = x + 23
...
```

Tu vidíme, že v cykle sa stále opakujú dva príkazy:

```
platno.create_rectangle(x, y, x+20, y+20, fill='red')
x = x + 23
```

a to sa robí presne `10` -krát (resp. toľko, koľko potrebujeme štvorčekov). Prepíšeme to na **for-cuklus**:

```
import tkinter

platno = tkinter.Canvas()
platno.pack()

y = 100
x = 5
for x in range(10):
    platno.create_rectangle(x, y, x+20, y+20, fill='red')
    x = x + 23
```

Toto riešenie je už teraz tak flexibilné, že zvládne nakresliť ľubovoľný počet štvorčekov, ktoré budú nakreslené v jednom rade.

Vylepšená postupnosť range

Už vieme, že zápis `range(5)` nahrádza postupnosť celých čísel `0` , `1` , `2` , `3` , `4` . Môžeme to použiť aj všeobecne s nejakou premennou napr. `n` , kde `range(n)` označuje postupnosť začínajúcu `0` a končiacu `n-1` (pre nezáporné `n` je počet prvkov tejto postupnosti presne `n`). Uvedomte si, že `range(0)` označuje postupnosť nulovej dĺžky (tzv. **prázdna postupnosť**), teda cyklus:

```
print('start')
for i in range(0):
    print(i)
print('koniec')
```

neprejde ani raz, teda program vypíše len riadok pred cyklom a po skončení cyklu:

```
start
koniec
```

Niekedy by sa nám mohlo hodiť, keby **premenná cyklu** nezačínala nulou, ale nejakou inou celočíselnou hodnotou, napr. `1` . Predstavme si, že potrebujeme vypočítať súčin čísel `1 * 2 * 3 * ... * n` (faktoriál čísla `n`). Veľmi výhodne by sa to počítalo pomocou cyklu, v ktorom budeme nejakú hodnotu `fakt` násobiť ďalším číslom `cislo` . Predpokladajme, že číslo bude postupne `1` , `2` , `3` , ..., `n` . Teda cyklus bude vyzerať nejako takto:

```
fakt = 1
for cislo in ...........:
    fakt = fakt * cislo
print(fakt)
```

Je jasne, že v hlavičke for-cyklu nemôžeme zadať `range(n)` , lebo to by znamenalo, že medzi číslami by bola aj `0` a tou určite nechceme násobiť.

Našťastie existuje druhý variant volania `range` , ktorý umožňuje zadať aj počiatočnú hodnotu postupnosti:

```
range(od, do)
```

označuje postupnosť, ktorá začína hodnotou `od` , každá ďalšia hodnota je o 1 väčšia ako predchádzajúca a táto postupnosť končí vtedy, keď by nasledovné hodnota bola väčšia alebo rovná `do` (teda maximálne `do-1`). Ukážme to na niekoľkých príkladoch:

```
range(7)              # 0, 1, 2, 3, 4, 5, 6
range(0, 7)           # 0, 1, 2, 3, 4, 5, 6
range(1, 7)           # 1, 2, 3, 4, 5, 6
range(5, 7)           # 5, 6
range(7, 7)           # prázdna postupnosť
range(9, 7)           # prázdna postupnosť
range(-2, 7)          # -2, -1, 0, 1, 2, 3, 4, 5, 6
```

Z tohoto vidíme, že keď budeme potrebovať postupnosť od `1` do `n`, musíme ju zapísať `range(1, n+1)`.

Teraz už môžeme zapísať podprogram `faktorial`, ktorý pre dané `n` vypočíta a vypíše **faktoriál** tohto čísla:

```
def faktorial(n):
    fakt = 1
    for cislo in range(1, n+1):
        fakt = fakt * cislo
    print('faktorial', n, '=', fakt)
```

Tento podprogram môžeme otestovať takýmto cyklom:

```
for n in range(11):
    faktorial(n)
```

a dostaneme tento výpis:

```
faktorial 0 = 1
faktorial 1 = 1
faktorial 2 = 2
faktorial 3 = 6
faktorial 4 = 24
faktorial 5 = 120
faktorial 6 = 720
faktorial 7 = 5040
faktorial 8 = 40320
faktorial 9 = 362880
faktorial 10 = 3628800
```

Zápis `range` má ešte jedno vylepšenie:

```
range(od, do, krok)
```

Ak má tri parametre, potom tretí parameter `krok` označuje hodnotu, o ktorú sa bude kaˇždý nasledovný člen zvyšovať. Napr. ak je krok `2`, tak prvý člen postupnosti je `od`, druhý `od+2`, tretí `od+4`, atď. až po maximálnu hodnotu, ktorá **nebude väčšia** ako horná hranica `do`. Napr.

```
range(0, 7)          # 0, 1, 2, 3, 4, 5, 6
range(0, 7, 1)       # 0, 1, 2, 3, 4, 5, 6
range(0, 7, 2)       # 0, 2, 4, 6
range(1, 7, 2)       # 1, 3, 5
range(-3, 7, 2)      # -3, -1, 1, 3, 5
```

príklady ...

Úlohy

1. Vypíšte prvých 30 mocnín čísla `2` v tvare:

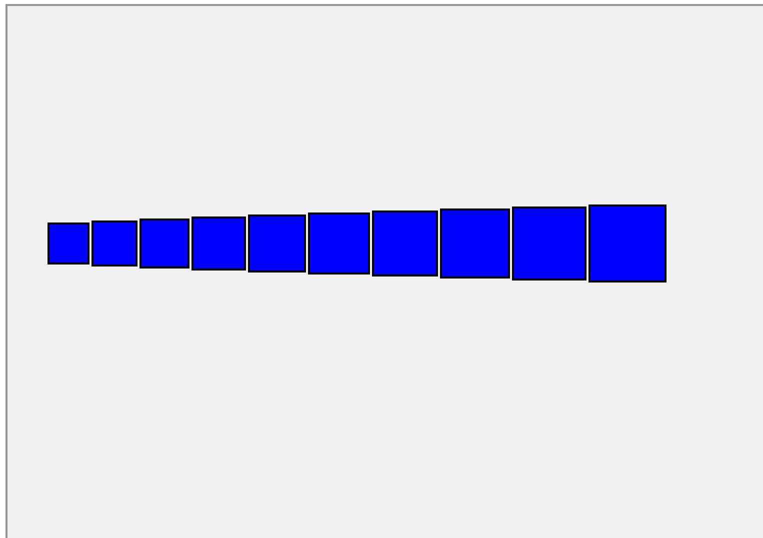
   ```
   0 1
   1 2
   2 4
   3 8
   4 16
   5 32
   6 64
   ...
   ```

   V programe nepoužívajte umocňovanie (napr. `2**6 = 64`), ale v cykle predchádzajúci výsledok vynásobíte `2`.

2. Napíšte program, ktorý postupne vypočíta mocniny `11` a vypíše ich pod seba do grafickej plochy. V programe nepoužívajte umocňovanie, ale postupne predchádzajúci výsledok násobite `11`. Dostanete takýto obrázok:

```
      11
     121
    1331
   14641
  161051
 1771561
 19487171
 214358881
 2357947691
 25937424601
 285311670611
3138428376721
```

3. Napíšte program, ktorý nakreslí desať tesne vedľa seba položených štvorcov, veľkosti ich strán sú postupne 20, 22, 24, ... Dostanete takýto obrázok:



Zhrnutie

čo sme sa naučili:

- pomocou príkazu `for` môžeme opakovať vykonávanie jedného alebo viacerých príkazov (tzv. **blok príkazov**)
- všetky riadky bloku príkazov musia byť odsunuté o 4 medzery vpravo
- for-cyklus používa premennú cyklu, ktorá pri každom prechode postupne nadobúda hodnoty buď z vymenovanej postupnosti, alebo s udanej postupnosti pomocou `range`

dôležité zásady:

- meno premennej cyklu zvoľte tak, aby čo najlepšie vyjadrovalo hodnotu, ktorá sa v cykle bude automaticky meniť
- for-cyklus s vymenovanými hodnotami používajte len vtedy, keď tieto hodnoty netvoria nejakú pravideľnú postupnosť a nie je predpoklad, že v budúcnosti bude treba zovšeobecňovať tento cyklus - inak radšej použite konštrukciu `range`

13. Kreslenie elíps a kruhov

Zoznámime sa s ďalším grafickým príkazom. Zatiaľ sme sa naučili vytvárať grafické útvary obdĺžniky a texty. Kreslenie elíps (v Pythobe im hovoríme aj ovály) a ich špeciálnych prípadov kružníc bude oproti tomu trochu náročnejšie. Môžu nám pritom veľmi dobre pomôcť skúsenosti s kreslením obdĺžnikov.

Kreslenie elipsy

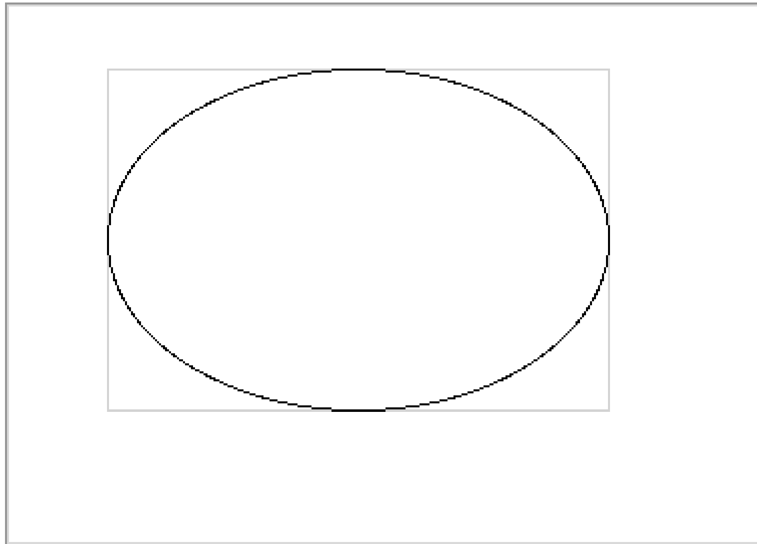
Príkaz na kreslenie elipsy má tento základný tvar:

```
platno.create_oval(x1, y1, x2, y2)
```

Tento príkaz nakreslí elipsu, ktorá sa kompletne zmestí do obdĺžnika s rovnakými parametrami. Matematici tomuto hovoria vpísaná elipsa. Aby sme to lepšie videli, napíšeme program, ktorý nakreslí elipsu aj obdĺžnik s rovnakýni parametrami:

```
import tkinter

platno = tkinter.Canvas(bg='white')
platno.pack()

platno.create_rectangle(50, 30, 300, 200, outline='light gray')
platno.create_oval(50, 30, 300, 200)
```

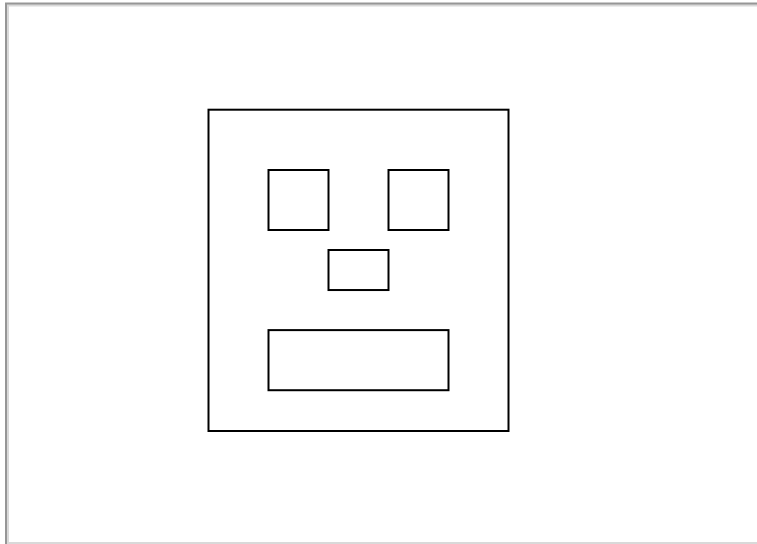
Po spustení vidíme obdĺžnik aj elipsu, ktorá má rovnaké parametre ako tento obdĺžnik:



Kým si nezvykneme na tieto parametre, môžeme si najprv očakávaný obrázok nakresliť pomocou obdĺžnikov a keď s tým budeme spokojní, prepíšeme ich na kreslenie elíps. Napr. pre tento program zatiaľ len s obdĺžnikmi:

```
import tkinter

platno = tkinter.Canvas(bg='white')
platno.pack()

platno.create_rectangle(100, 50, 250, 210)
platno.create_rectangle(130, 80, 160, 110)
platno.create_rectangle(190, 80, 220, 110)
platno.create_rectangle(160, 120, 190, 140)
platno.create_rectangle(130, 160, 220, 190)
```

dostávame:



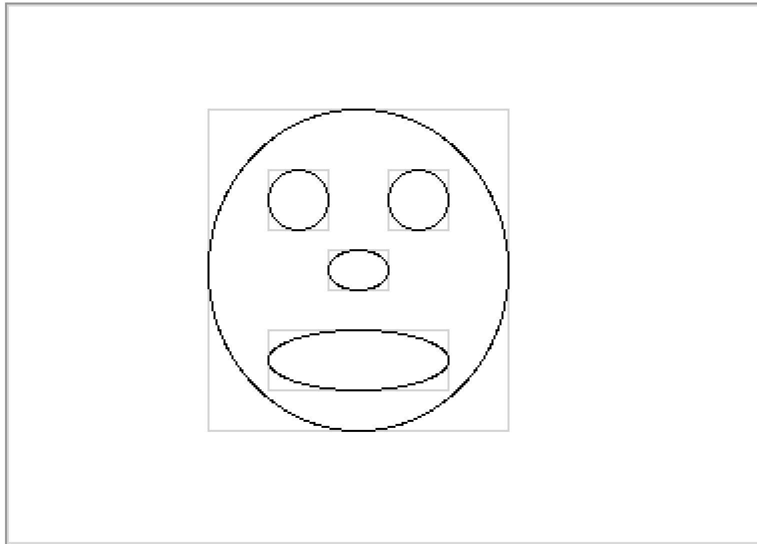
Teraz vymeníme všetky výskyty príkazu `create_rectangle` na `create_oval`:

```
import tkinter

platno = tkinter.Canvas(bg='white')
platno.pack()

platno.create_oval(100, 50, 250, 210)
platno.create_oval(130, 80, 160, 110)
platno.create_oval(190, 80, 220, 110)
platno.create_oval(160, 120, 190, 140)
platno.create_oval(130, 160, 220, 190)
```

Vidíme:

```
.. image:: image/13_03.png
```

Ak by sme pritom ponechali naznačené obdĺžniky, videli by sme naozaj súvis medzi týmito dvomi typmi útvarov v Pythone:



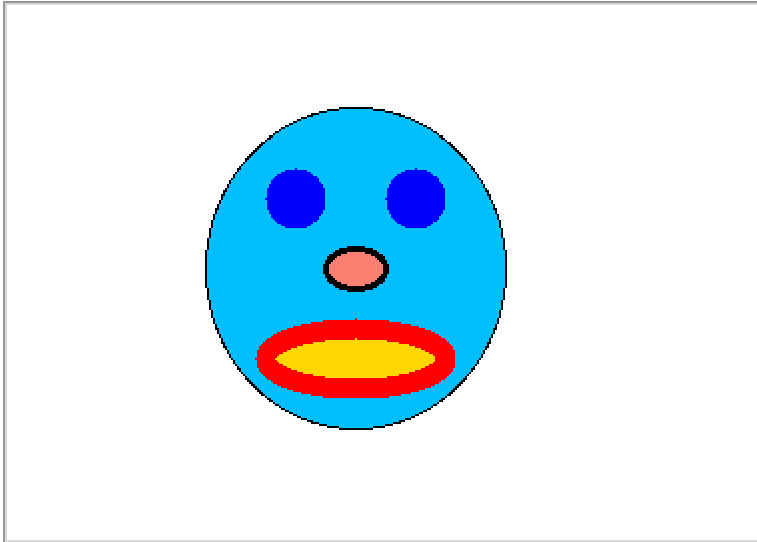
Veľmi dôležité je aj to, že všetky parametre v príkaze `create_rectangle`, ktoré sme sa naučili pri kreslení obdĺžnikov, fungujú aj pre elipsy. Teda pomocou parametrov môžeme pri kreslení elipsy nastaviť:

- pomocou `fill` farbu výplne
- pomocou `outline` farbu obrysu
- pomocou `width` hrúbku obrysovej čiary

Predchádzajúcemu obrázku zmeňme niektoré tieto atribúty:

```
import tkinter

platno = tkinter.Canvas(bg='white')
platno.pack()

platno.create_oval(100, 50, 250, 210, fill='deep sky blue')
platno.create_oval(130, 80, 160, 110, fill='blue', width=0)
platno.create_oval(190, 80, 220, 110, fill='blue', width=0)
platno.create_oval(160, 120, 190, 140, fill='salmon', width=3)
platno.create_oval(130, 160, 220, 190, fill='gold', outline='red', width=10)
```

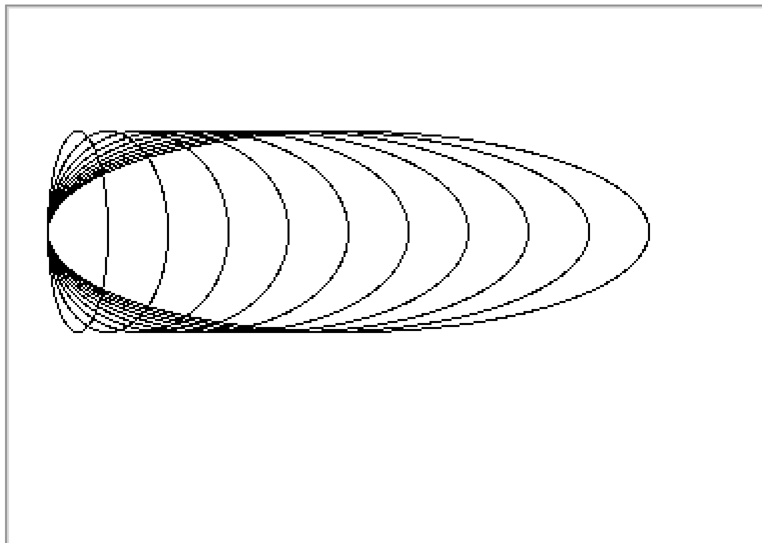
a takto sa to zafarbilo:



Elipsy poskytujú pekné námety aj pri použití vo for-cykloch. Nakreslime 10 elíps, ktoré majú spoločný ľavý horný vrchol (mysleného) obdĺžnika a `x` -ová súradnica pravého dolného vrchola sa zvyšuje (inkrement) o nejakú konštantu:

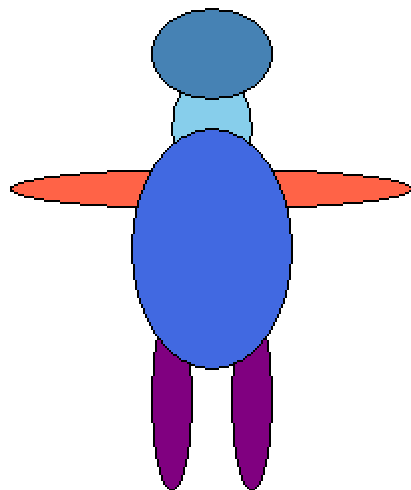
```
import tkinter

platno = tkinter.Canvas(bg='white')
platno.pack()

x = 50
for i in range(10):
    platno.create_oval(20, 60, x, 160)
    x = x + 30
```

Dostali sme takýto obrázok:

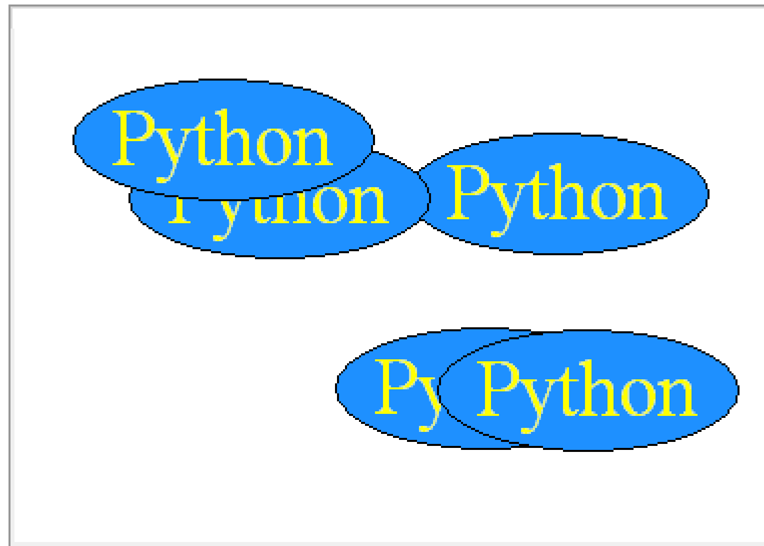


Úlohy

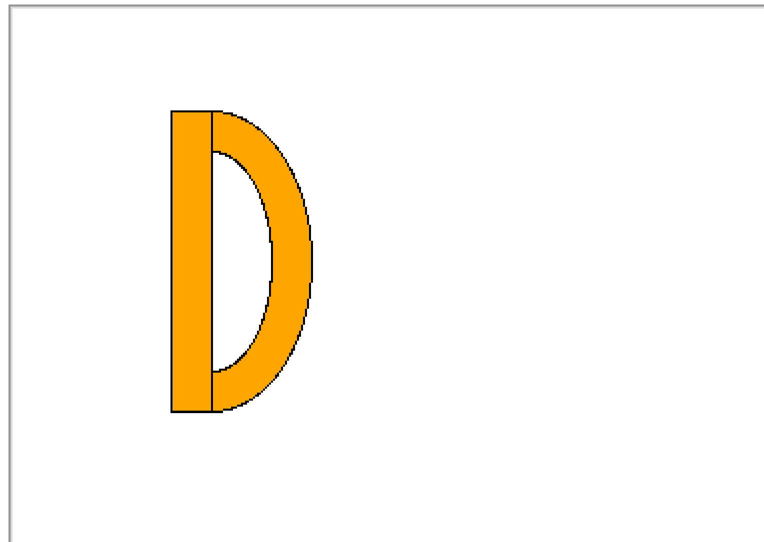
1. Experimentujte s programom, ktorý v 7. časti kreslil robota a nahraďte všetky volania `create_rectangle` na `create_oval` .



2. V 10. časti sme vytvorili podporgram `karticka_nahodne` , ktorá do modrého obdĺžnika zapísala nejaký text. Poexperimentujte s tvarom tejto kartičky a nahraďte `create_rectangle` na `create_oval`

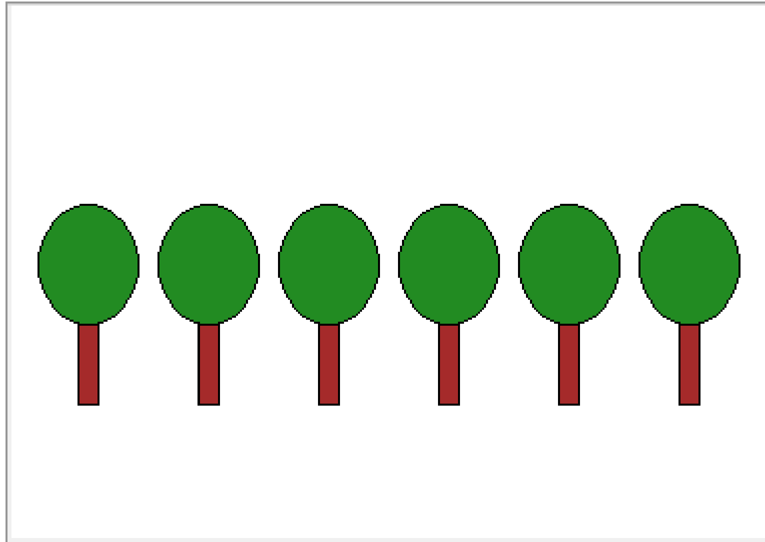


3. Pomocou elíps a obdĺžmikov nakreslite takéto písmeno **D**:



Pri kreslení tohto obrázku sa najprv nakrelila oranžová elipsa, potom o trochu menšia biela, potom biely obdĺžnik zmaže polovicu veľkej elipsy a nakoniec sa nakreslí oranžový obdĺžnik.

4. Pomocou úzkeho hnedého obdĺžnika (rozmerov 10x40) a zelenej elipsy (rozmerov 50x60) vieme nakresliť jeden strom. Napíšte cyklus, pomocou ktorého sa nakreslí celý rad stromov. Rozostupy medzi stromami nech sú `60` :



Použite for-cyklus, v ktorom `range` bude mať krok `60`.

Zhrnutie

čo sme sa naučili:

- elipsy a kružnice sa kreslia na rovnakom princípe ako sa kreslili obdĺžniky
- aj elipsy môžeme vyfarbovať, resp. im nastaviť hrúbku a farbu obrysu

14. Kruhy a cykly

Kreslenie kružnice

Zrejme kreslenie kružnice je len špeciálnym prípadom kreslenia elipsy. Ak pri kreslení elipsy „myslený“ obdĺžnik bude štvorcom (oba rozmery: šírku aj výšku má rovnaké), nakreslí sa kružnica, resp. ak bude vyplnená farbou, tak kruh.

Pripomeňme si, nový príkaz `stvorec_nahodne` zo 4. časti:

```
import tkinter
import random

platno = tkinter.Canvas(bg='white')
platno.pack()

def stvorec_nahodne():
    x = random.randint(10, 310)
    y = random.randint(10, 200)
    platno.create_rectangle(x, y, x+50, y+50, fill='indian red')
```

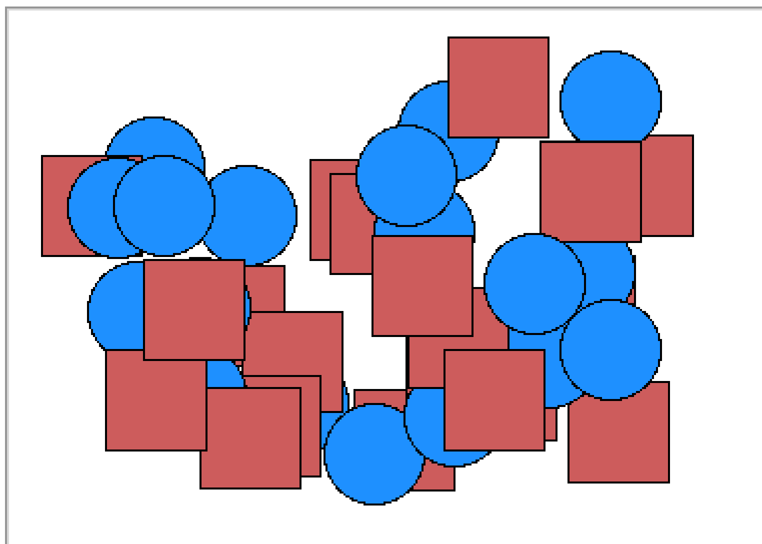
Keďže tento podprogram kreslí štvorce, výmenou `create_rectangle` za `create_oval` dostaneme kruhy. Vytvorme na to podprogram `kruh_nahodne` :

```
def kruh_nahodne():
    x = random.randint(10, 310)
    y = random.randint(10, 200)
    platno.create_oval(x, y, x+50, y+50, fill='dodger blue')
```

Aby sme otestovali oba tieto podprogramy `stvorec_nahodne` aj `kruh_nahodne` , zavoláme ich vo for-cykle tak, aby sa striedali:

```
for i in range(20):
    stvorec_nahodne()
    kruh_nahodne()
```

Dostávame takýto obrázok:



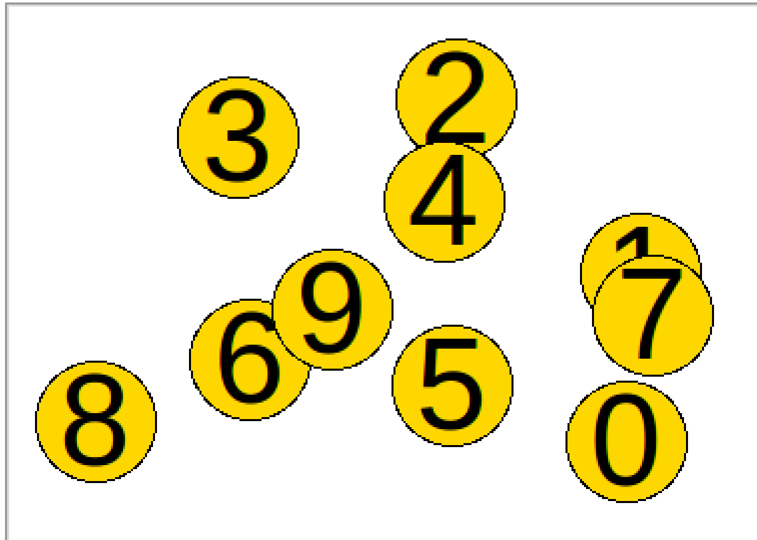
Vo väčšine prípadov, pri kreslení kružníc, by sa nám hodilo, keby sme mohli vychádzať z toho, že ich vieme nakresliť pomocou daného stredu (`x` , `y`) a polomeru `r` . Ak si uvedomíme, že je to takto jednoduché:

```
platno.create_oval(x - r, y - r, x + r, y + r)
```

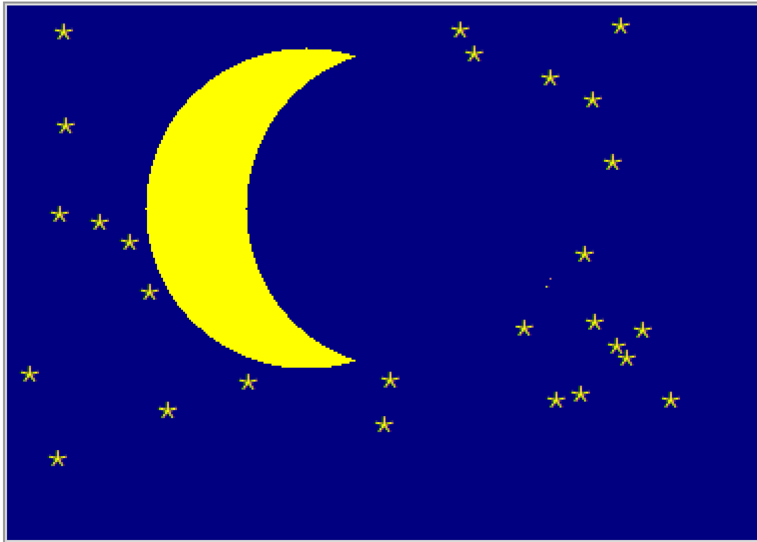
Zjednoduší sa nám pohľad na mnohé úlohy. Vyriešme takúto úlohu: na náhodné pozície nakreslíme 10 žltých kruhov a do každého z nich postupne zapíšeme číslo od `0` do `9` . Zrejme využijeme for-cyklus, v ktorej bude premenná cyklu nadobúdať hodnoty od `0` do `9` a v tele cyklu sa vygenerujú náhodné súradnice `x` a `y` a na ne sa nakreslí žltý kruh aj vypíše text:

```
import tkinter
import random

platno = tkinter.Canvas(bg='white')
platno.pack()

for cislo in range(10):
    x = random.randint(30, 330)
    y = random.randint(30, 220)
    platno.create_oval(x-30, y-30, x+30, y+30, fill='gold')
    platno.create_text(x, y, text=cislo, font='arial 40')
```

Všimnite si, ža ako parameter `text` pre vypisovanie textu v `create_text` sme nastavili premennú cyklu `cislo`. Po spustení sa zobrazí podobný takýto obrázok:



V ďalšej ukážke nakreslíme takýto obrázok:



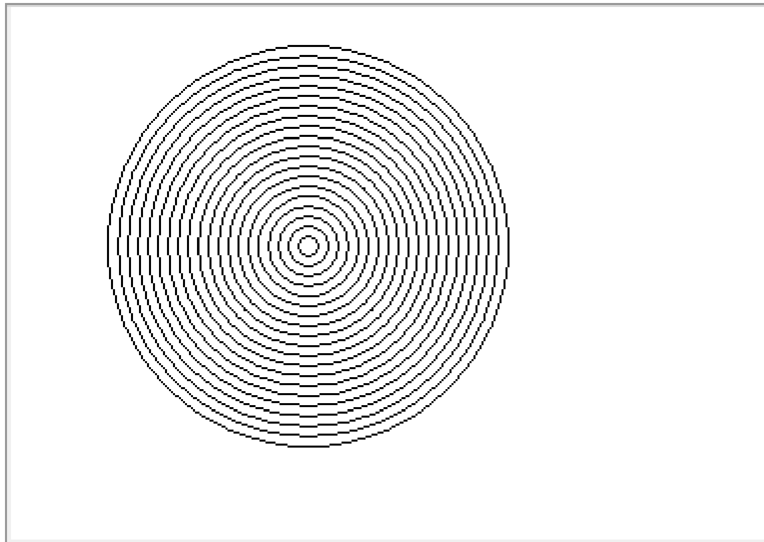
Obrázok sa skladá z mesiačika (dva prekrývajúce sa kruhy, jeden žltý a druhý tmavomodrý) a 30 žltých hviezdičiek na náhodných pozíciách. Hviezdičky vykresľujeme ako žlté znaky `'*'` :

```
import tkinter
import random

platno = tkinter.Canvas(bg='navy')
platno.pack()

for i in range(30):
    x = random.randint(10, 370)
    y = random.randint(10, 230)
    platno.create_text(x, y, text='*', fill='yellow', font='courier 15')

x = 150
y = 100
r = 80
platno.create_oval(x-r, y-r, x+r, y+r, fill='yellow', width=0)
x = x + 50
platno.create_oval(x-r, y-r, x+r, y+r, fill='navy', width=0)
```

For-cyklus o vylepšený `range` môžeme využiť napr. pri kreslení sústredných kružníc (kružnice so spoločným stredom), ktoré majú postupne polomery `5`, `10`, `15`, `20`, ..., `100`:

```
import tkinter

platno = tkinter.Canvas(bg='white')
platno.pack()

for r in range(5, 101, 5):
    platno.create_oval(150-r, 120-r, 150+r, 120+r)
```

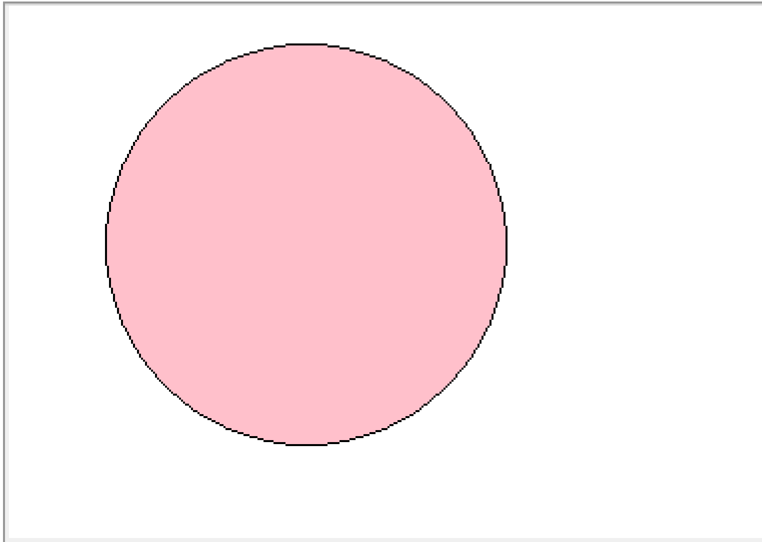
dostaneme:



Zaujímavý výsledok dostaneme, keď budeme tieto kružnice zafarbovať:

```
for r in range(5, 101, 5):
    platno.create_oval(150-r, 120-r, 150+r, 120+r, fill='pink')
```

dostaneme:



Hoci sme zafarbovali všetky kružnice, vidíme len najväčšiu z nich. Tá prekryla všetky menšie. Ak by sme chceli vidieť všetky nakreslené zafarbené kružnice, musíme ich kresliť v opačnom poradí: od vajväčšej po najmenšiu. Teda stačí zmeniť `range(...)` tak, aby namiesto postupnosti `5`, `10`, `15`, `20`, ..., `100`, sa vygenerovala `100`, `95`, ..., `15`, `10`, `5`. Dá sa to zapísať pomocou záporného kroku:

```
range(100, 0, -5)
```

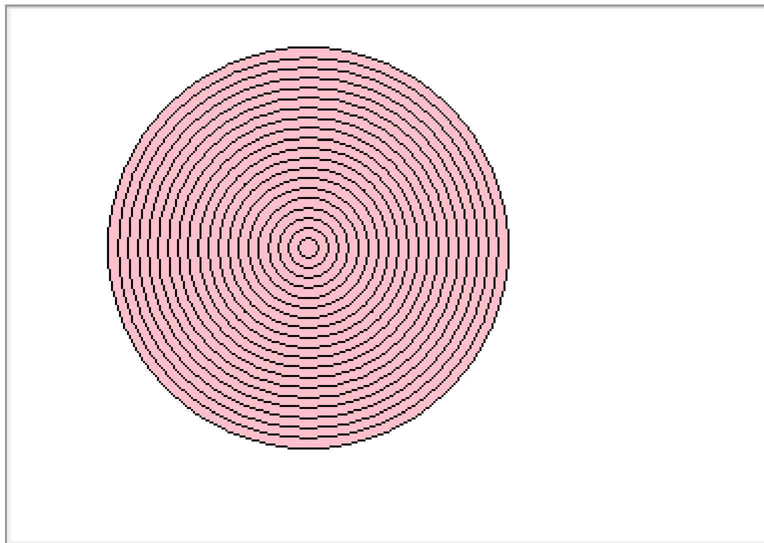
a bude to fungovať správne. My sa naučíme užitočnejší zápis, pomocou ktorého otočíme poradie hodnôt ľubovoľnej postupnosti:

```
reversed(range(5, 101, 5))
```

Vďaka tomuto nemusíme rozmýšľať, ako správne zapísať dolnú a hornú hranicu pre záporný krok. Teraz naše riešenie vyzerá takto:

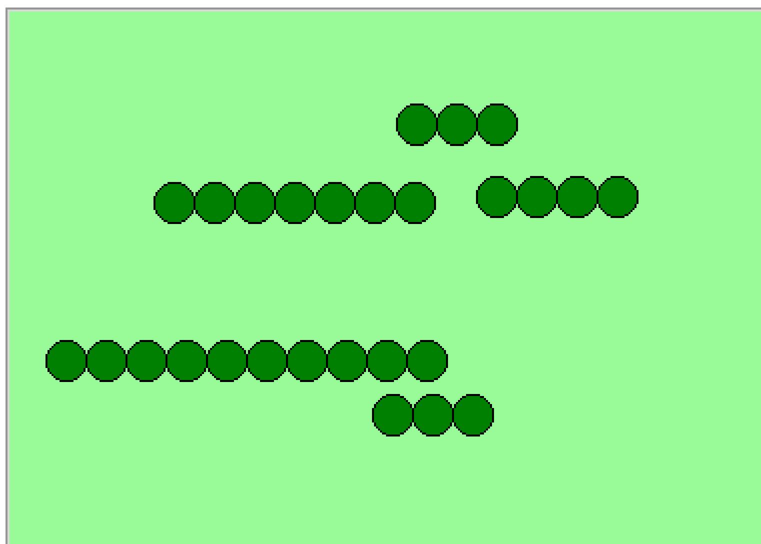
```
for r in reversed(range(5, 101, 5)):
    platno.create_oval(150-r, 120-r, 150+r, 120+r, fill='pink')
```

a vidíme:

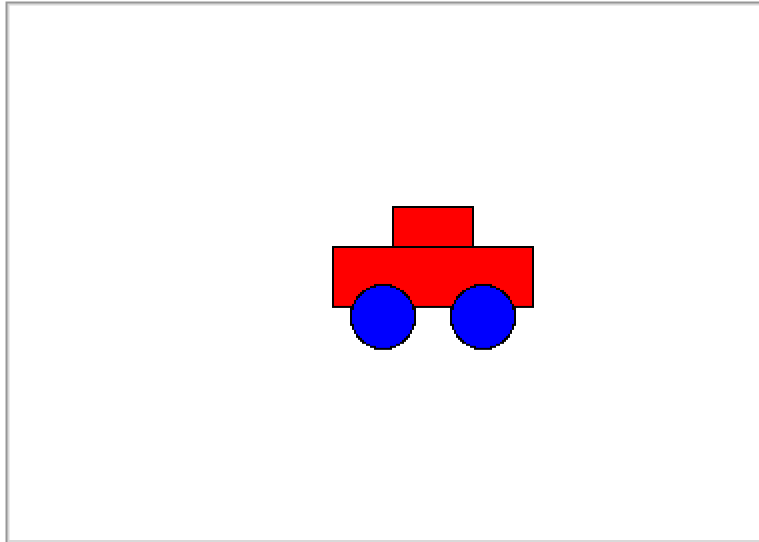


Úlohy

1. Napíšte podprogram `husenica`, ktorá na náhodnú pozíciu nakreslí húseničku, ktorá je zložená z náhodného počtu zelených krúžkov. Dĺžka húseničky je od `3` do `10`. Po niekoľkých zavaloniach tohto podprogramu, môžeme dostať, napr.

2. Napíšte podrogram `auto`, ktorá na náhodnú pozíciu nakreslí približne takéto autíčko:



3. Napíšte podprogram `kruznice10`, ktorý na náhodnú pozíciu nakreslí 10 kružníc so spoločným stredom a s polomermi postupne `5`, `10`, `15`, `20`, `25`, `30`, `35`, `40`, `45`, `50`. Použite for-cyklus s `range(10)`.

   Bude tento podprogram fungovať aj vtedy, keď im nastavíme nejakú výplň, napr. `fill='pink'`?

Zhrnutie

čo sme sa naučili:

- elipsy a kružnice sa kreslia na rovnakom princípe ako sa kreslili obdĺžniky
- aj elypsy môžeme vyfarbovať, resp. im nastaviť hrúbku a farbu obrysu
- kružnice so stredom (`x`, `y`) a plomerom `r` sa dobre kreslia pomocou `create_oval(x-r, y-r, x+r, y+r)`

15. Parameter podprogramu

Už od 7. témy **Vytvárame podprogramy** sme zostavovali veľa rôznych podprogramov, napr.

- zavolanie `stvorec_nahodne()` na náhodnú pozíciu nakreslí zafarbený štvorček veľkosti 50x50
- zavolanie `karticka_nahodne()` opäť na náhodnú pozíciu vykreslí bledomodrú kartičku s textom `Python`
- zavolanie `hlava()`, `telo()`, `ruky()`, `nohy()` nakreslia na konštantnú pozíciu časti robota

Všetky tieto naše nové príkazy, teda podprogramy, fungujú úplne nezávisle. Ak by sme im chceli hocičo zmeniť, museli by sme buď preprogramovať podprogram, alebo vyrobiť z neho kópiu a v nej urobiť opravu.

Ukážeme to na príklade. Použijeme opäť podprogram `stvorec_nahodne`:

```
import tkinter
import random

platno = tkinter.Canvas(bg='white')
platno.pack()

def stvorec_nahodne():
    x = random.randint(10, 310)
    y = random.randint(10, 200)
    platno.create_rectangle(x, y, x+50, y+50, fill='indian red')
```

Tento podprogram vždy na náhodné pozície nakreslí červený štvorec veľkosti 50x50. Ak by sme ale potrebovali nakresliť aj štvorec napr. veľkosti 10x10, alebo nebodaj zelený, museli by sme vyrobiť kópiu tohto podprogramu (s novým menom) a v nej by sme urobili opravy, napr. takto:

```
def stvorec_nahodne2():
    x = random.randint(10, 310)
    y = random.randint(10, 200)
    platno.create_rectangle(x, y, x+10, y+10, fill='pale green')
```

Prípadne by sme pre každú predpokladanú veľkosť vytvorili nový podprogram s novým menom, prípadne s novou farbou.

Ale aj na toto existuje vo všetkých programovacích jazykoch mechanizmus, pomocou ktorého vieme riešiť takéto situácie bez toho, aby sme vyrábali kópie podprogramov s malými zmenami oproti pôvodným verziám.

Podprogram s parametrom

Pomocou parametra vieme do podprogramu pri jeho zavolaní priniesť z vonku nejakú informáciu, napr. veľkosť štvorčeka, počet kružníc, farba štvorčeka, text na kartičke a pod. To znamená, že podprogram musíme na toto pripraviť špeciálnym spôsobom, aby vedel s touto informáciou spávne pracovať.

Pri definovaní podprogramu môžeme zapísať:

```
def meno_príkazu(parameter):
    príkaz
    príkaz
    ...
```

V zátvorkách hlavičky podprogramu môžeme uviesť meno parametra, s ktorým bude vedieť tento podprogram pracovať. Je vhodné tu zvoliť také meno, aby už z neho bolo jasné s akou informáciu bude podprogram pracovať. Napr. keď zapíšeme:

```
def stvorec_nahodne(velkost):
    ...
```

už z hlavičky podprogramu tušíme, že podprogram bude kresliť štvorec danej veľkosti. Ak by sme videli hlavičku podprogramu:

```
def karticka_nahodne(vypisovany_text):
    ...
```

budeme predpokladať, že text, ktorý je parametrom bude vypísaný aj na kartičke.

Takže **v tele podprogramu** (odsunutý blok príkazov) by sa mal príslušný parameter použiť. Z pohľadu podprogramu je meno parametra obyčajnou premennou a preto s ňou podprogram môže normálne pracovať, napr.

```
def stvorec_nahodne(velkost):
    x = random.randint(10, 360-velkost)
    y = random.randint(10, 250-velkost)
    platno.create_rectangle(x, y, x+velkost, y+velkost, fill='indian red')
```

Vidíte, že v tele podprogramu sa pracuje s premennou `velkost` ako s obyčajnou premennou, ktorá je zatiaľ „neznáma“. Pri volaní podprogramu teraz namiesto `stvorec_nahodne()` musíme do týchto okrúhlych zátvoriek zapísať aj **hodnotu parametra**. Táto hodnota sa pri zavolaní podprogramu priradí do parametra `velkost` a tým sa bude dať korektne vykonať telo podprogramu.

Ak by sme teraz zapísali:

```
>>> stvorec_nahodne()
...
TypeError: stvorec_nahodne() missing 1 required positional argument: 'velkost'
```

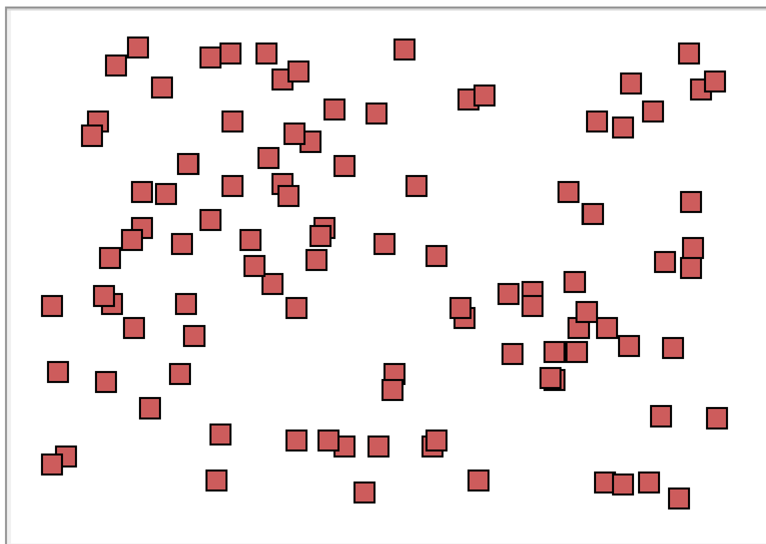
Python nám touto správou oznamuje, že podprogram `stvorec_nahodne` očakával, že zadáme hodnotu parametra `velkost` . Hodnotu parametra (tzv. **skutočný parameter**) zadávame pri volaní podprogramu v okrúhlych zátvorkách. Napr. takéto volanie:

```
>>> stvorec_nahodne(10)
```

už funguje správne: hodnota `10` sa priradí do parametra `velkost` a potom sa normálne vykoná telo podprogramu už s touto priradenou hodnotou. Podprogram teda na náhodnú pozíciu nakreslí červený štvorček 10x10. Môžeme to vidieť. aj keď to zavoláme vo for-cykle:

```
for i in range(10):
    stvorec_nahodne(100)
```

dostaneme:



Teraz ukážeme niekoľko príkladov podprogramov s parametrami.

Podprogram `mocniny` vypíše `n` riadkov, pričom v každom bude jeho poradové číslo (číslujeme od `0`) a tiež druhá a tretia mocnina tohto čísla. Zrejme parametrom procedúry bude `n` a samotný podprogram bude vo for-cykle vypisovať tri čísla: premennú cyklu a jej druhú a tretiu mocninu:

```
def mocniny(n):
    for i in range(n):
        print(i, i**2, i**3)
```

Otestujeme:

```
>>> mocniny(10)
0 0 0
1 1 1
2 4 8
3 9 27
4 16 64
5 25 125
6 36 216
7 49 343
8 64 512
9 81 729
```

Ďalší podprogram `vypis` bude mať parameter, ktorým nastavíme, aký text sa má vypísať, že ho máme radi:

```
def vypis(co):
    for i in range(5):
        print('*** mam rad', co, '***')
```

Otestujeme:

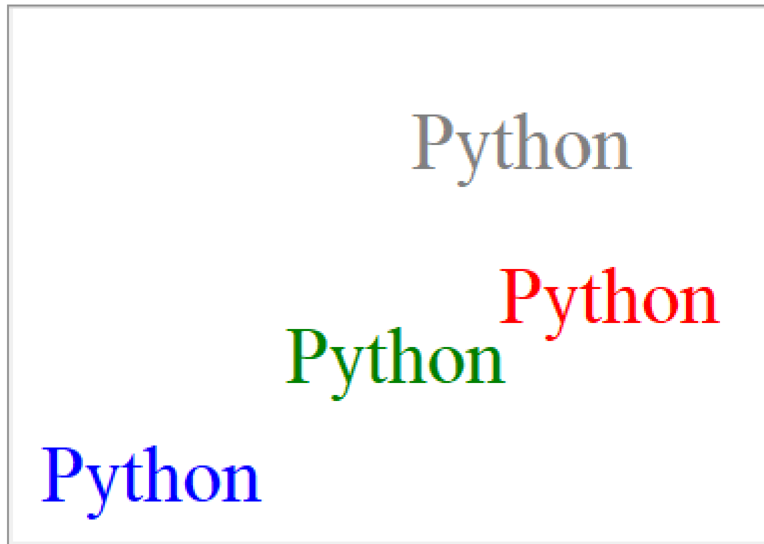
```
>>> vypis(37)
*** mam rad 37 ***
*** mam rad 37 ***
*** mam rad 37 ***
*** mam rad 37 ***
*** mam rad 37 ***
>>> vypis('matfyz')
*** mam rad matfyz ***
*** mam rad matfyz ***
*** mam rad matfyz ***
*** mam rad matfyz ***
*** mam rad matfyz ***
```

Vidíme, že hodnotou parametra nemusí byť len číslo, ale aj znakový reťazec.

Ďalší podprogram `vypis_python` na náhodnú pozíciu grafickej plochy vypíše slovo `'Python'`. Parametrom podprogramu bude farba tohto textu:

```
import tkinter
import random

platno = tkinter.Canvas(bg='white')
platno.pack()

def vypis_python(farba):
    x = random.randint(50, 350)
    y = random.randint(50, 250)
    platno.create_text(x, y, text='Python', fill=farba, font='times 30')

vypis_python('red')
vypis_python('green')
vypis_python('blue')
vypis_python('gray')
```

Tento program na náhodné pozície vypíše slovo `'Python'` rôznymi farbami:



Úlohy

1. Napíšte podprogram `sucet` s jedným parametrom `n`, ktorý spočíta súčet prvých `n` nepárnych čísel. Napr. pre `n=4` treba spočítať `1+3+5+7`. Procedúra vypíše tento súčet, napr.

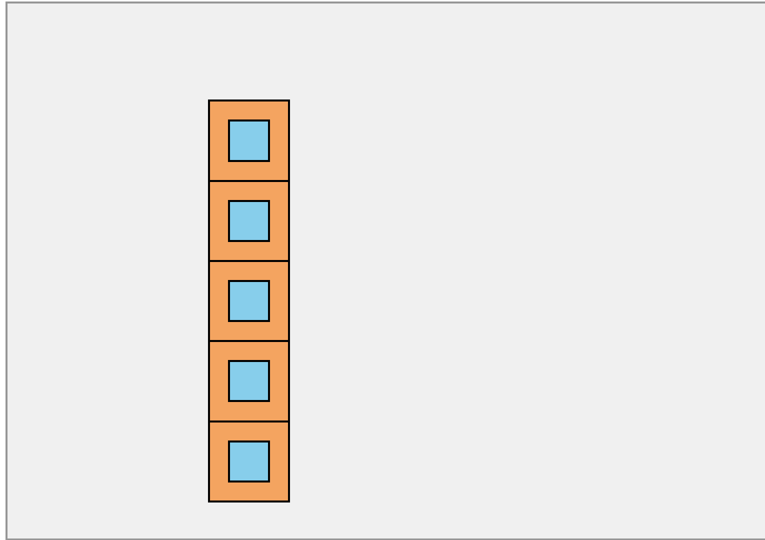
   ```
   >>> sucet(4)
   vysledok = 16
   ```

   V procedúre využite for-cyklus, v ktorom bude mať `range` krok `2`.

2. Napíšte podprogram `panelak` s jedným parametrom `pocet`, ktorý zo štvorcov veľkosti 40x40 nakreslí panelák so zadaným počtom poschodí. Každé poschodie paneláku by mohlo mať nakreslené aj jedno okno. Napr.

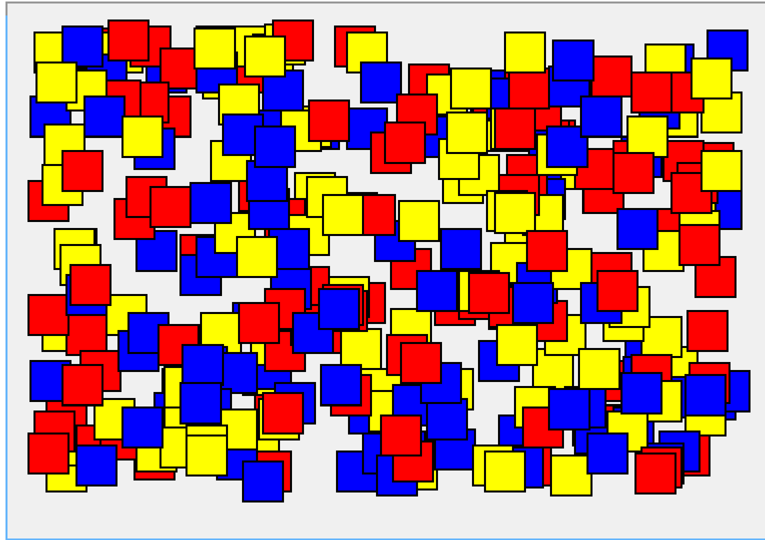
   ```
   >>> panelak(5)
   ```

   nakreslí takýto obrázok:



3. Napíšte podprogram `farebny_stvorec` s jedným parametrom `farba`, ktorý na náhodnú pozíciu nakreslí štvorec veľkosti 20x20 s danou farbou. Napr. nasledovný kód nakreslí takýto obrázok:

```
for i in range(100):
    farebny_stvorec('red')
    farebny_stvorec('blue')
    farebny_stvorec('yellow')
```



Zhrnutie

čo sme sa naučili:

- podprogramy môžu mať 0 alebo jeden parameter - definujú sa v hlavičke podprogramu `def`
- pri volaní podprogramov musíme do zátvoriek uviesť presne toľko hodnôt, koľko parametrov sa tu očakáva
- Python potom tieto hodnoty parametrov dosadí do dočasných premenných, ktoré sa po skončení vykonávania tela podprogramu zrušia

16. Parametre podprogramov

Viac parametrov

Pri vytváraní podprogramu môžeme využiť aj viac ako jeden parameter. Vtedy ich v hlavičke podprogramu navzájom oddelíme čiarkami. Zapíšeme:

```
def meno_príkazu(parameter1, parameter2, ...):
    príkaz
    príkaz
    ...
```

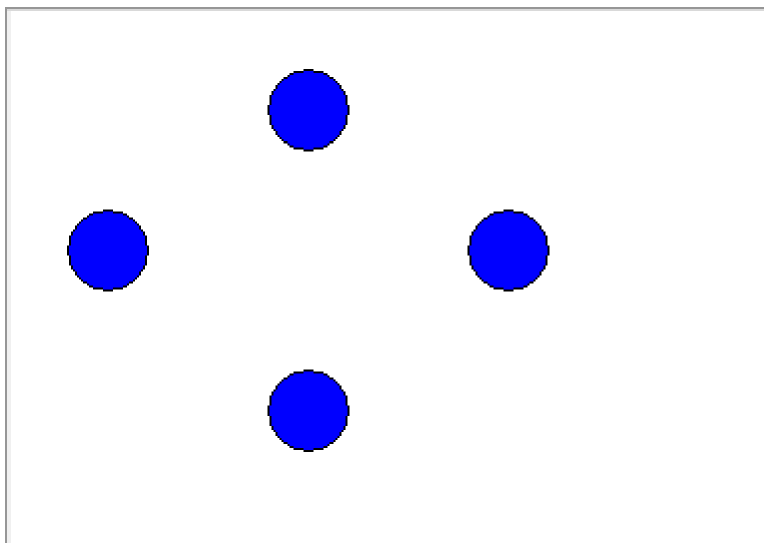
Ako to funguje, ukážeme na príklade. Napíšeme podprogram, ktorý pre dané (`x` , `y`) nakreslí modrý kruh s daným stredom a polomerom `20` :

```
import tkinter

platno = tkinter.Canvas(bg='white')
platno.pack()

def modry_kruh(x, y):
    platno.create_oval(x - 20, y - 20, x + 20, y + 20, fill='blue')
```

Podprogram `modry_kruh` má dva parametre `x` a `y` , a preto, keď ho chceme zavolať, musíme určiť hodnoty oboch parametrov. Zrejme pri volaní podprogramu sa prvá hodnota priradí do prvého parametra `x` a druhá hodnota do druhého `y` . Napr.

```
modry_kruh(150, 50)
modry_kruh(150, 200)
modry_kruh(50, 120)
modry_kruh(250, 120)
```

Nakreslí:



Nič nám nebráni v tom, aby sme tento podprogram ešte trochu vylepšili. Napíšeme podprogram `kruh`, ktorý bude mať okrem `x` a `y` ešte dva ďalšie parametre `r` a `farba`:

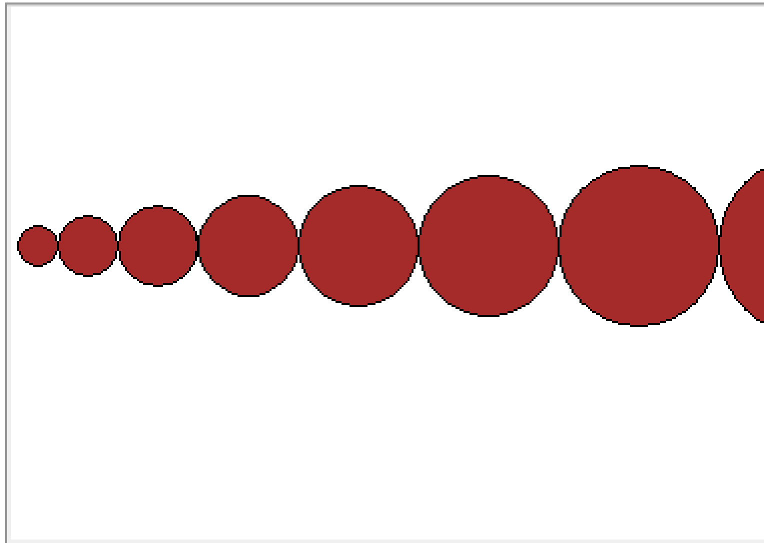
```
def kruh(x, y, r, farba):
    platno.create_oval(x - r, y - r, x + r, y + r, fill=farba)
```

Z mien parametrov sa dá pochopiť, že podprogram bude kresliť kruh so stredom (`x`, `y`), s polomerom `r` a s farbou výplne `farba`. Takáto logika parametrov je pri programovaní bežná, len si bude treba dávať trochu väčší pozor na poradie parametrov pri volaní podprogramu.

Vďaka tomuto jednoduchému podprogramu môžeme takto elegantne zapísať program, ktorý nakreslí do jedného radu niekoľko zväčšujúcich sa kruhov:

```
x = 15
r = 10
for i in range(8):
    kruh(x, 120, r, 'brown')
    x = x + r + r + 5
    r = r + 5
```

a takto to vyzerá po spustení:



Keďže v premennej `x` je `x` -ová súradnica stredu práve kresleného kruhu, preto po nakreslení kruhu najprv toto `x` zväčšíme o `r` (dostali sme sa na pravý okraj kružnice) a keďže nasledovný kruh bude mať polomer `r` zväčšený o `5` , tak aj `x` zväačšíme o `r+5` - teraz je `x` nastavené presne na stred nasledovného už väčšieho kruhu.

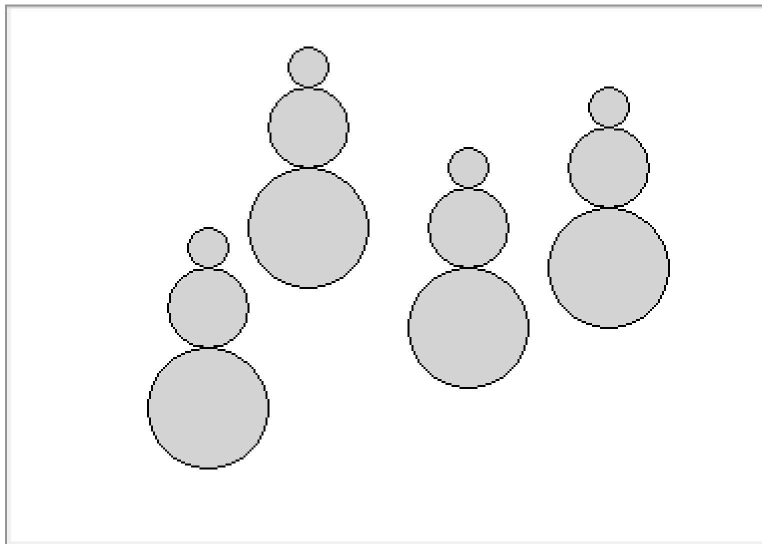
Teraz, keď už máme podprogram `kruh` , môžeme nakresliť snehuliaka, ktorý sa bude skladať z troch na sebe postavených kruhov. Napíšeme nový podprogram, ktorý bude mať iba dva parametre `x` a `y` a na ich základe nakreslí tri kruhy takto: prvý najväčší bude mať svoj stred v (`x` , `y`) a polomer `30` , druhý s polomerom `20` bude umiestnený na ňom, teda jeho `y` -ová súradnica bude menšia nielen o polomer väčšieho `30` ale aj jeho polomer teda `20` . Najmenší kruh s polomerom `10` bude mať ešte zmenšenú `y` -ovú súradnicu oproti druhému o svoj aj jeho polomer. Zapíšme tento nový podprogram:

```
def snehuliak(x, y):
    kruh(x, y, 30, 'light gray')
    kruh(x, y-30-20, 20, 'light gray')
    kruh(x, y-30-20-20-10, 10, 'light gray')
```

Farbu kruhov sme zvolili svetlošedú, aby boli lepšie viditeľné na bielom podklade. Napr. volania:

```
snehuliak(100, 200)
snehuliak(150, 110)
snehuliak(230, 160)
snehuliak(300, 130)
```

Nakreslia:



Je dobré si uvedomiť, ako vlastne fungujú vzájomné volania podprogramov:

- náš program štyrikrát zavolal podprogram `snehuliak` - zakaždým s inými parametrami
- v podprograme `snehuliak` sa trikrát zavolal podprogram `kruh` - zakaždým s niekorými inými parametrami
- v podprograme `kruh` sa raz zavolal príkaz `create_oval` , ktorý je v skutočnosti tiež podprogram, lenže tento už dávnejšie naprogramoval niekto iný a je súčasťou modulu `tkinter`

Čo všetko sa v skutočnosti udeje, keď sa na nejakom mieste zavolá nejaký podprogram?

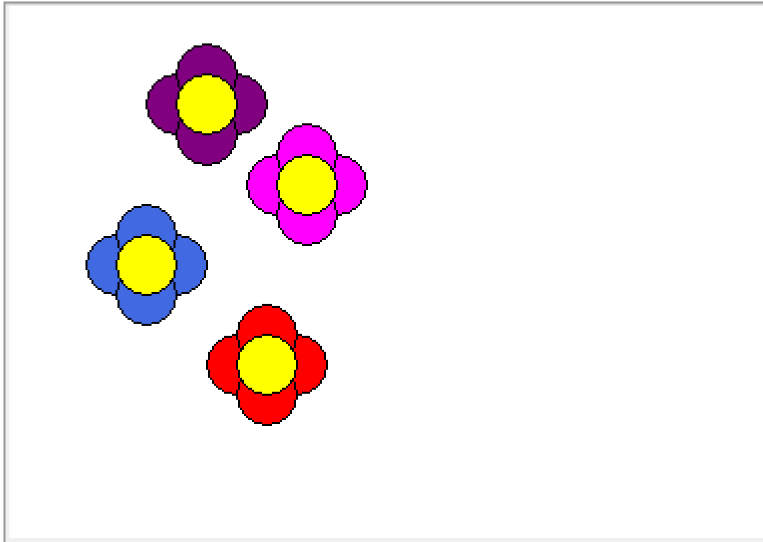
1. Python si musí zapamätať presné miesto, kam sa bude treba po vykonaní podprogramu vrátiť
    - napr. pri volaní `snehuliak(100, 200)` sa zapamätá, že potom bude nasledovať vykonávanie 2. riadku programu
2. Pre každý z parametrov v hlavičke podprogramu sa vytvoria **dočasné premenné** a priradia sa im hodnoty skutočných parametrov
    - napr. pri volaní `snehuliak(100, 200)` sa v tomto podprograme vyrobia dve **dočasné premenné** `x` a `y` a priradia sa im hodnoty `100` a `200`
3. Postupne sa vykonajú všetky príkazy v tele podprogramu
    - pre podprogram `snehuliak` sú to tri volania `kruh`
4. Keď sa sončí vykonávanie všetkých príkazov z tela podprogramu, zrušia sa všetky **dočasné premenné**, teda `x` a `y`
5. Ďalší výpočet pokračuje na zapamätanom riadku z prvého bodu postupu

Asi ste si všimli, že oba podprogramy `snehuliak` aj `kruh` majú parametre `x` a `y`. Každý z týchto podprogramov má ale svoju vlastnú verziu týchto premenných. Python bez problémov zvláda veľa verzií tých istých premenných, ktoré ale existujú len počas behu nejakého podprogramu.

Úlohy

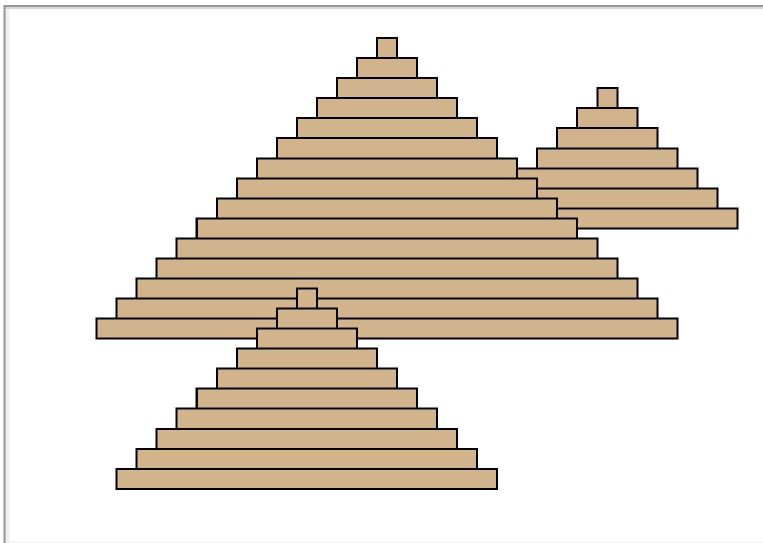
1. ...

2. Napíšte podprogram `kvet`, ktorý bude mať 3 parametre `x`, `y` a `farba`. Podprogram nakreslí takúto kvetinku: najprv štyri lupene (zafarbené kruhy, ktoré sa dotýkajú bodu (`x`, `y`)) a potom do stredu (`x`, `y`) žltý kruh. Všetky tieto kruhy majú polomer `15`. Napr.



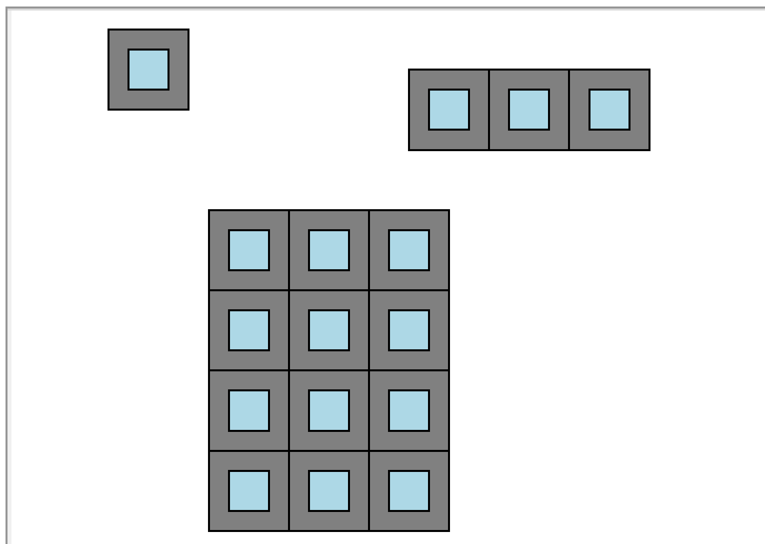
Na kreslenie kruhov použite procedúru `kruh` .

3. Napíšte podprogram `pyramida` s tromi parametrami: `vyska` , `x` , `y` . Podprogram nakreslí pyramídu, ktorá je zložená s `vyska` obdĺžnikov, pričom najmenší má rozmer 10x10, každý ďalší má šírky o 20 väčšiu, teda postupne 30x10, 50x10, … Parametre `x` a `y` určujú stred najmenšieho obdĺžnika. Napr.



Na tomto obrázku sú nakreslené tri pyramídy, ktoré boli vytvorené volaním podprogramu `pyramida` s rôznymi parametrami.

4. Napíšte tri podprogramy: `panel(x, y)`, `rad(n, x, y)` a `panelak(m, n, x, y)`. Podprogram `panel` nakreslí šedý štvorec 40x40 s modrým oknom 20x20 uprostred. (`x`, `y`) je ľavý horný vrchol tohto štvorca. Podprogram `rad` nakreslí rad panelov, t.j. `n` -krát zavolá podprogram `panel`. (`x`, `y`) je ľavý horný vrchol prvého panelu v rade. Podprogram `panelak` zavolá `m` -krát podprogram `rad`, aby sa postavil panelák zložený z `m` radov po `n` panelov. Na nasledovnom obrázku vidíte jeden `panel`, jeden `rad` aj `panelak` so 4 radmi po 3 panely v každom:



Zhrnutie

čo sme sa naučili:

- podprogramy môžu mať 0, 1, 2, alebo aj viac parametrov - definujú sa v hlavičke podprogramu `def`
- pri volaní podprogramov musíme do zátvoriek uviesť presne toľko hodnôt, koľko parametrov sa tu očakáva
- Python potom tieto hodnoty parametrov dosadí do dočasných premenných, ktoré sa po skončení vykonávania tela podprogramu zrušia

17. Premenné typu znakové reťazce, operácie, prevody

S pojmom **znakový reťazec** sa stretáme od prvých hodín s Pythonom. Najprv to bol vypisovaný text v `print` :

```
>>> print('Pozdravujem vas, lesy, hory!')
Pozdravujem vas, lesy, hory!
```

Potom to bolo meno farby pri kreslení obdĺžnikov, textov a elíps:

```
platno.create_rectangle(30, 150, 130, 250, width=6, fill='red')
```

Neskôr sa objavili znakové reťazce aj pri vypisovaní textov do grafickej plochy aj pri určovaní písma takéhoto textu:

```
platno.create_text(190, 40, text='Programovanie', font='Arial 40', fill='blue')
```

Potom sme videli znakové reťazce aj ako parametre podprogramov:

```
def vypis(co):
    print('*** mam rad', co, '***')

vypis('matfyz')

def vypis_python(farba):
    platno.create_text(150, 120, text='Python', fill=farba, font='times 30')

vypis_python('red')
```

Pozrime sa na tento dátový typ trochu podrobnejšie.

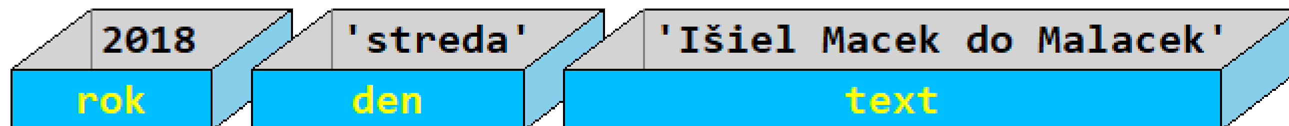
Znakové reťazce

Už vieme, že znakový reťazec je nejaký text uzavretý v apostrofoch. Niekedy môžete namiesto apostrofov vidieť úvodzovky. Python akceptuje aj úvodzovky. Väčšinou je to vecou zvyku.

Znakový reťazec môžeme priradiť do premennej rovnako, ako sme to robili s číslami. Napr.

```
rok = 2018
den = 'streda'
text = 'Išiel Macek do Malacek'
```

V pamäti sa vytvorili tri nové premenné. Mohli by sme ich zakresliť takto:



Zrejme Python si pri každej premennej pamätá aj jej **typ**. To znamená, že premenná `rok` je celé číslo (po anglicky **integer**) a premenné `den` a `text` sú znakové reťazce (po anglicky **string**). Momentálny typ nejakej hodnoty (alebo premennej) vieme zistiť pomocou špeciálnej funkcie `type`. Napr.

```
>>> type(rok)
<class 'int'>
>>> type(den)
<class 'str'>
>>> type(text)
<class 'str'>
```

V týchto výpisoch sa objavilo `'int'` a `'str'`, tieto označujú **celočíselný typ** (integer) a **typ znakový reťazec** (string). Všimnite si ešte:

```
>>> 22 / 7
3.142857142857143
>>> type(22 / 7)
<class 'float'>
```

V Pythone takéto delenie dvoch čísel vráti **desatinné číslo** a takýto typ má meno **float**.

Čítanie zo vstupu

Python má jeden veľmi dôležitý príkaz `input`, pomocou ktorého môže bežiaci program získať od používateľa nejaké hodnoty. Najčastejšie sa tento príkaz používa v takomto priraďovacom príkaze:

```
odpoved = input('nejaký text? ')
```

Tento príkaz najprv vypíše zadaný text a potom Python od nás **čaká**, že zadáme ľubovoľný text na klavesnici a stlačíme kláves `<Enter>`. Vtedy sa nami zadaný text priradí do príslušnej premennej, ktorú sme uviedli na ľavej strane priraďovacieho príkazu.

Napíšme takýto malý testovací program:

```
print('Ahoj, ja som Python.')
kto_si = input('A ty sa ako volas? ')
print('ahoj', kto_si)
```

Po spustení tohto programu prebehne približne takýto dialóg:

```
Ahoj, ja som Python.
A ty sa ako volas? Adam
ahoj Adam
```

Program najprv pomocou `print` vypíše `Ahoj, ja som Python.`. Do ďalšieho riadku príkaz `input` vypíše text `A ty sa ako volas?` a ďalej čaká, že na klávesnici napíšeme ľubovoľný text a ukončíme ho klávesom `<Enter>`. My sme tu zadali slovo `Adam` a preto sa do nasledovného riadku vypísal text `ahoj Adam`.

Ďalší program ukazuje, ako môžeme využiť čítanie zo vstupu v grafike:

```
import tkinter

farba = input('zadaj farbu podkladu: ')
vypisat = input('zadaj vypisovany text: ')

platno = tkinter.Canvas(bg=farba)
platno.pack()

platno.create_text(180, 120, text=vypisat, font='arial 50')
```

Po spustení je najprv tento dialóg:

```
zadaj farbu podkladu: light blue
zadaj vypisovany text: Bratislava
```

Potom sa do grafickej plochy nakreslí:

Bratislava

V tomto príklade nám premenné so znakovými reťazcami poslúžili na nastavovanie nejakých parametrov v grafických príkazoch.

Znakové reťazce vo for-cykloch

Pripomeňme si for-cyklus, v ktorom sme vymenovali všetky hodnoty, ktoré sa postupne priradia do premennej cyklu:

```
for co in 7, 11, 37, 42:
    print('mam rad', co)
```

```
mam rad 7
mam rad 11
mam rad 37
mam rad 42
```

Lenže znakové reťazce sa môžu nachádzať aj medzi vymenovanými hodnotami, napr.

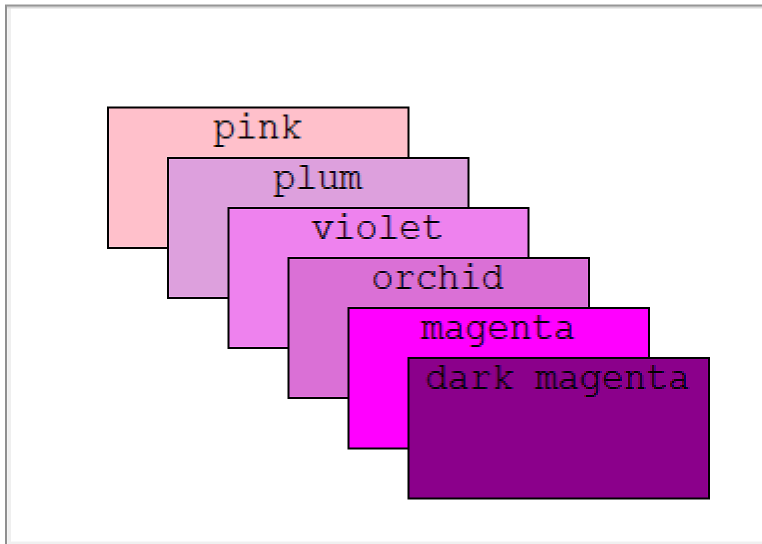
```
for co in 7, 11, 'matfyz', 'kremes':
    print('mam rad', co)
```

```
mam rad 7
mam rad 11
mam rad matfyz
mam rad kremes
```

Niekedy môžeme využiť takýto for-cyklus s reťazcami aj pri kreslení do grafickej plochy:

```
import tkinter

platno = tkinter.Canvas(bg='white')
platno.pack()

x = 50
y = 50
for farba in 'pink', 'plum', 'violet', 'orchid', 'magenta', 'dark magenta':
    platno.create_rectangle(x, y, x+150, y+70, fill=farba)
    platno.create_text(x+75, y+10, text=farba, font='courier 14')
    x = x + 30
    y = y + 25
```

Vidíme, že premenná cyklu `farba` tu postupne nadobúda jednu z vymenovaných hodnôt:



Znakové reťazce v spojitosti s for-cyklom majú ešte jednu špeciálnu vlastnosť: keď namiesto vymenovanej postupnosti hodnôt pre premennú cyklu uvedieme len jeden reťazec, tak Python to pochopí tak, že tento reťazec je vymenovanou postupnosťou znakov tohto reťazca. Možeme si to predstaviť tak, že keď zapíšeme:

```
for i in 'AHOJ':
    ...
```

tak Python to rozoberie na znaky a urobí z toho cyklus:

```
for i in 'A', 'H', 'O', 'J':
    ...
```

Môžeme to otestovať napr. takto:

```
for znak in 'Python':
    print('***', znak, '***')
```

```
*** P ***
*** y ***
*** t ***
*** h ***
*** o ***
*** n ***
```

Operácie s reťazcami

Python má niekoľko operácií, ktoré pracujú so znakovými reťazcami. Napr. pre čísla funguje operácia `+`, ktorá označuje súčet dvoch čísel. Zrejme `12 + 34` je súčet dvoch celých čísel a výsledkom je tretie číslo `46`.

Operácia zreťazenia

Jednou z najčastejšie používanýchu operácií s reťazcami je tzv. **zreťazenie**. Pomocou tejto operácie vznikne z dvoch reťazcov jeden **nový**, ktorý je zlepením dvoch pôvodných. Na túto operáciu sa používa rovnaké znamienko ako pre súčet čísel, t.j. `+`.

Môžeme vyskúšať:

```
>>> a = 'Juraj'
>>> b = 'Jánošík'
>>> a + b
'JurajJánošík'
>>> b + a
'JánošíkJuraj'
>>> a + ' ' + b
'Juraj Jánošík'
```

Zreťazenie sa môže využiť aj vo for-cykle takto:

```
ret = '*'
for i in range(30):
    ret = ret + ' *'
print(ret)
```

Keďže na začiatku mal reťazec `ret` len jeden znak a potom sme k nemu na koniec pridali dva znaky `' *'` a to sme spravili 30-krát, dostávame znakový reťazec dĺžky 61:

```
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
```

Ešte zaujímavejčší je tento cyklus:

```
ret = '+'
for i in range(6):
    ret = ret + ret
print(ret)
```

Na začiatku ešte pre cyklom mal reťazec `ret` jeden znak `'+'`. V prvom prechode cyklu sme zreťazili dva reťazce `'+'` a `'+'` a výsledok sme priradili do `ret`. Teraz je v ňom `'++'`. V ďalšom prechode sa opäť zreťazí `'++'` a `'++'` a výsledok `'++++'` sa priradí do `ret`. Každý ďalší prechod cyklom zdvojnásobí počet znakov `'+'` v premennej `ret`. Po šiestom prechode cyklu bude v premennej `ret` znakový reťazec, ktorý bude obsahovať **64** znakov `'+'`. Po spustení programu naozaj dostaneme:

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

Operácia viacnásobné zreťazenie

Videli sme, že pomocou for-cyklu a zreťazenia vieme vyrábať dosť dlhé znakové reťazce. Napr. aj zápis:

```
>>> vela = 'ahoj ' + 'ahoj ' + 'ahoj ' + 'ahoj ' + 'ahoj '
>>> vela
'ahoj ahoj ahoj ahoj ahoj '
```

Tento by sme vedeli zapísať for-cyklom, v ktorom by sa reťazec `'ahoj '` zlepil 5-krát za sebou tak, ako sa 30-krát zlepoval reťazec `' *'`.

V Pythone je ďalšia operácia, tzv. **viacnásobné zreťazenie**, pomocou ktorej vieme veľmi elegantne zlepiť ten istý reťazec zadaný počet krát. Táto operácia sa zapisuje pomocou znamienka `*`: napr. `číslo * reťazec` alebo `reťazec * číslo` označuje, že sa daný reťazec zreťazí za seba určený počet krát. Napr.

```
>>> 'ahoj ' * 5
'ahoj ahoj ahoj ahoj ahoj '
>>> 20 * '*'
'********************'
>>> '>' * 10 + ' ' * 10 + '<' * 10
'>>>>>>>>>>          <<<<<<<<<<'
>>> 3 * ('xy' * 5 + ' O ')
'xyxyxyxyxy O xyxyxyxyxy O xyxyxyxyxy O '
```

Uvedomte si, že takto sa dá vyrobiť dosť dlhý reťazec, s ktorým môže mať váš počítač už problém.

Pomocou viacnásobného zreťazenia vieme vyriešiť aj takúto úlohu: podprogram `trojuholnik` s parametrom `n` nakreslí zo znakov `'*'` takýto trojuholník (napr. pre `n=8`):

```
       *
      ***
     *****
    *******
   *********
  ***********
 *************
***************
```

Samotný program je takto jednoduchý:

```
def trojuholnik(n):
    for i in range(n):
        print(' ' * (n-i-1) + '*' * (2*i+1))
```

Zisťovanie dĺžky reťazca

Už vieme vyrábať znakové reťazce najrôznejších dĺžok. Python má štandardnú funkciu `len`, pomocou ktorej vieme zistiť dĺžku znakového reťazca. Napr.

```
>>> a = 'Python'
>>> len(a)
6
>>> b = a * 10
>>> b
'PythonPythonPythonPythonPythonPythonPythonPythonPythonPython'
>>> len(b)
60
```

Ešte sme sa zatiaľ asi nestretli s tzv. **prázdnym reťazcom**, to je reťazec, ktorý neobsahuje žiadne znaky. Zapisujeme ho `''`. Môžeme zistiť aj jeho dĺžku:

```
>>> c = ''    # prázdny reťazec
>>> len(c)
0
>>> d = c * 10
>>> len(d)
0
```

Pripomeňme si for-cyklus, v ktorom sme veľakrát zdvojnásobovali dĺžku reťazca:

```
>>> x = 'a'
>>> for i in range(10):
        x = x + x
>>> len(x)
1024
```

Vidíme, že takto získame reťazec dĺžky `1024` (čo je `2` umocnené na `10`, teda `2**10`). Vieme vyrobiť aj takto veľký reťazec:

```
>>> y = 'a' * 100000000
>>> len(y)
100000000
```

čo je 100 miliónov, teda tento reťazec zaberá v pamäti až 100 MB (megabajtov). Nepokúšajte sa ale takýto reťazec vypisovať napr. pomocou `print` - Python s tým bude mať veľké problémy.

Čísla a reťazce

Už poznáme hodnoty troch typov: celé čísla (`int`), desatinné čísla (`float`) a znakové reťazce (`str`)- Tiež vieme, že operácia plus `+` vie sčítať dve čísla (napr. `2+3=5`) alebo zreťaziť dva reťazce (napr. `'a'+'b'='ab'`). Ale, ako je to s operáciou `+`, keď budeme sčitovať reťazec a číslo? Vyskúšajme:

```
>>> 'a' + 7
...
TypeError: must be str, not int
>>> '7' + 7
...
TypeError: must be str, not int
```

Podobne to dopadne, ak by sme sčitovali číslo s reťazcom. Python pri každej operácii robí kontrolu typov oboch operandov a hlasi chybu, ak sa táto operácia vykonať nedá. Podľa prvého operandu (čo je znakový reťazec) Python pochopil, že sme plánovali robiť zreťazovanie dvoch reťazcov a preto nám aj oznámil túto chybu.

Pozrite si tento program:

```
prve = input('zadaj prve cislo: ')
druhe = input('zadaj druhe cislo: ')
print('sucet zadanych cisel', prve, 'a', druhe, 'je', prve + druhe)
```

Dostávame napr.

```
zadaj prve cislo: 12
zadaj druhe cislo: 34
sucet zadanych cisel 12 a 34 je 1234
```

My už vieme, že príkaz `input` nám vždy priradí **znakový reťazec** a preto aj „súčet“ dvoch znakových reťazcov v skutočnosti zrealizoval zreťazenie dvoch reťazcov. Na tomto mieste by sme potrebovali **prerobiť znakový reťazec na celé číslo**. V Pythone máme na toto špeciálnu funkciu `int`, ktorá sa zo znakového reťazca vyrobí celé číslo. Nie náhodou má rovnaké meno ako meno typu `int` pre celé čísla - tejto funkcii hovoríme **prevod** reťazca na číslo. Skôr ako to použijeme v predchádzajúcom programe so súčtom, vyskúšajme túto funkciu v interaktívnom režime:

```
>>> int('1234')
1234
>>> int('7') + 7
14
>>> int('7a')
...
ValueError: invalid literal for int() with base 10: '7a'
```

Teraz môžeme opraviť náš program so súčtom dvoch prečítaných čísel:

```
prve = int(input('zadaj prve cislo: '))
druhe = int(input('zadaj druhe cislo: '))
print('sucet zadanych cisel', prve, 'a', druhe, 'je', prve + druhe)
```

Dostávame napr.

```
zadaj prve cislo: 12
zadaj druhe cislo: 34
sucet zadanych cisel 12 a 34 je 46
```

Všimnite si zápis `int(input('zadaj prve cislo: '))`. Takto sa zvykne zapisovať príkaz na čítanie vstupu z klávesnice, ak predpokladáme, že týmto vstupom je celé číslo. Týmto zápisom sa teda prečítaný reťazec automaticky **prevedie** na celé číslo.

Niekedy môžeme riešiť opačnú úlohu: potrebujeme zreťaziť znakový reťazec a celé číslo do jedného reťazca. Napr.

```
>>> cislo = 2 ** 10
>>> cislo
1024
>>> retazec = 'x' + cislo
...
TypeError: must be str, not int
```

Tu sme predpokladali, že sa nám vyrobí znakový reťazec `'x1024'`. Už vieme, že takto to nebude fungovať. Musíme najprv z celého čísla `1024` vyrobiť znakový reťazec `'1024'` a až potom tento reťazec spojíme s reťazcom `'x'`. Opäť potrebujeme funkciu, ktorá **prerobí celé číslo na znakový reťazec** (teda opačnú k funkcii `int`). Takou funkciu je `str` a ak je jej parametrom celé číslo, hovoríme tomu **prevod reťazca na celé číslo**. Opäť sa meno tejto funkcie zhoduje s menom typu pre znakové reťazce `str`.

Teraz to už zapíšeme správne:

```
>>> retazec = 'x' + str(cislo)
>>> retazec
'x1024'
```

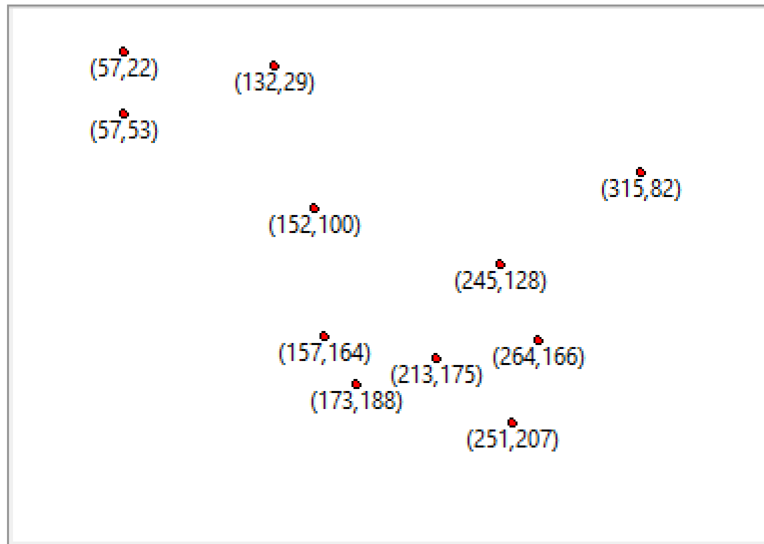
Mohli sme to zapísať aj takto priamo:

```
>>> retazec = 'x' + str(2 ** 10)
```

Napíšeme podprogram `oznac_bod`, v ktorom využijeme funkciu `str` na prevod čísla na reťazec. Tento podprogram s dvomi parametrami `x` a `y` na tejto pozícii nakreslí bodku a pod ňou napíše súradnice tohro bodu v tvare, napr. `'(57,22)'`:

```
import tkinter
import random

platno = tkinter.Canvas(bg='white')
platno.pack()

def oznac_bod(x, y):
    platno.create_oval(x-2, y-2, x+2, y+2, fill='red')
    retazec = '(' + str(x) + ',' + str(y) + ')'
    platno.create_text(x, y+8, text=retazec)

oznac_bod(57, 22)

for i in range(10):
    oznac_bod(random.randint(10, 350), random.randint(10, 220))
```

Pre otestovanie tohto podprogramu sme okrem bodu `(57, 22)` zvolili ešte nejakých 10 náhodných bodov. Dostávame takýto obrázok:



Úlohy

1. Napíšte program, v ktorom sú na začiatku priradenia do troch premenných `a`, `b` a `c` (alebo sú prečítané pomocou `input`). Napr.

   ```
   a = 'mam'
   b = 'rad'
   c = 'Python'
   ```

   Program potom vypíše všetkých 6 rôznych usporiadaní (permutácií) týchto troch hodnôt. Napr.

   ```
   mam rad Python
   Python rad mam
   rad Python mam
   ...
   ```

   Program by mal správne fungovať pre ľubovoľné hodnoty týchto troch premenných.

2. Napíšte program, ktorý najprv pomocou `input` prečíta nejaký reťazec a potom ho vypíše po znakoch tak, že každý ďalší znak je vypísaný o kúsok vpravo a kúsok dole od predchádzajúceho. Napr.

   ```
   zadaj retazec: informatika
   ```

   Dostávame:

3. Napíšte podprogram `do_stlpca` s jedným parametrom `cislo`, ktorý zadané celé číslo vypíše do grafickej plochy tak, že cifry tohto čísla budú zoradené v jednom stĺpci pod sebou. Napr. volanie `do_stlpca(3**11)` vytvorí tento obrázok v grafickej ploche:

18. Premenné typu desatinné čísla (float), operácie

Desatinné čísla

Vo väčšine našich doterajších úlohách už od 1. časti učebnice sme pracovali s celými číslami, prípadne so znakovými reťazcami. Lenže pre mnohé úlohy, ktoré potrebujeme riešiť na počítači, nám celočíselná aritmetika nestačí. Zišli by sa nám aj desatinné čísla (v informatike sa tomu hovorí aj **reálna aritmetika**). Napr. celé čísla sme delili pomocou aritmetickej operácie `//`, napr.

```
>>> 22 // 7
3
```

ale ak sme omylom namiesto dvoch lomiek zapísali len jednu, dostali sme:

```
>>> 22 / 3
7.333333333333333
```

Táto operácia `/` totiž vždy vráti desatinné číslo aj v prípade, že výsledok je delenie bez zvyšku, napr.

```
>>> 21 / 3
7.0
```

Desatinné čísla v Pythone sú teda také čísla, ktoré **obsahujú desatinnú bodku** (nie čiarku) alebo sú v tzv. **vedeckom zápise** (niekedy semilogaritmický tvar). Ukážme to na príklade:

```
>>> 3.14000
3.14
>>> 314e-2
3.14
>>> 0.0314e2
3.14
```

Všetky tri zápisy reprezentujú to isté desatinné číslo `3.14` . Druhý a tretí zápis obsahujú písmeno `e` , ktoré označuje, že za ním sa nachádza celočíselný exponent. Výsledné desatinné číslo získame tak, že zoberieme časť pred písmenom `e` (tzv. mantisa) a vynásobíme ho mocninou čísla `10` exponentom, ktorý je za písmenom `e` . Preto napr.

```
314e-2    označuje   314 * 10 ** -2

0.0314e2  označuje   0.0314 * 10 ** 2
```

Tento vedecký zápis čísel budeme používať veľmi zriedkavo. Častejšie to budú desatinné čísla bez písmena `e` .

Operácie s desatinnými číslami

Všetky operácie, ktoré fungovali s celými číslami, budú fungovať aj pre desatinné čísla. Platí tu jedno pravidlo, keď je aspoň jeden z operandov desatinné číslo, tak aj výsledok je desatinné číslo (zrejme okrem `/` , pri ktorom je výsledok vždy desatinné číslo). Preveríme to v interaktívnom režime:

```
>>> 123.0 + 456
579.0
>>> 1234 * 5678
7006652
>>> 1234. * 5678
7006652.0
>>> 1000 // 7
142
>>> 1000.0 // 7
142.0
>>> 1000 / 7
142.85714285714286
```

Desatinné čísla ale nemajú takú presnosť ako celé. Počítač si pri desatinných číslach pamätá len istý rozsah (približne 16 až 17 cifier), preto môžu byť niektoré výpočty dosť nepresné. Napr.

```
>>> 10000000000000000000000000 + 1
10000000000000000000000001
>>> 10000000000000000000000000 + 1.
1e+25
>>> 2 ** 100
1267650600228229401496703205376
>>> 2.0 ** 100
1.2676506002282294e+30
>>> 2 ** 100
1267650600228229401496703205376
>>> 2.0 ** 100
1.2676506002282294e+30
>>> 0.1 + 0.1 + 0.1
0.30000000000000004
```

Preto si treba vždy pri práci s desatinnými číslami uvedomiť, že tieto výpočty môžu byť naozaj nepresné.

Zaujímavé sú aj mocniny s desatinnými číslami. Napr. umocniť nejaké číslo na `1/2` (jednu polovicu) v skutočnosti označuje druhú odmocninu čísla. Napr.

```
>>> 2 ** (1/2)
1.4142135623730951
>>> 2 ** 1/2
1.0
>>> 25 ** .5
5.0
>>> 27 ** (1/3)
3.0
```

Na tomto príklade vidíte, že ak je exponentom pri umocňovaní nejaký výraz, musí sa uzavrieť do zátvoriek. Napr. `2 ** 1/2` označuje, že najprv sa vykonalo umocnenie `2 ** 1` a potom sa tento výsledok vydelil `2`. Posledný príklad `27 ** (1/3)` ukazuje tretiu odmocninu čísla `27`.

Ukážme teraz výpočet tohto súčtu:

```
1 + 1/2 + 1/3 + 1/4 + ... + 1/1000
```

Použijeme na to for-cyklus, v ktorom premenná cyklu bude nadobúdať hodnoty od `1` do `1000` a k nejakému súčtu budeme pripočitávať prevrátenú hodnotu premennej cyklu (`1 / cislo`). Zapíšme:

```
sucet = 0
for cislo in reversed(range(1, 1001)):
    sucet = sucet + 1 / cislo
print('vysledok =', sucet)
```

Dostávame výsledok:

```
vysledok = 7.485470860550343
```

Úlohy

1. Napíšte program, ktorý prečíta dve čísla a vypíše ich priemer. Napr.

   ```
   zadaj prve cislo: 15
   zadaj druhe cislo: 3.8
   priemer tychto dvoch cisel je 9.4
   ```

2. Napíšte podprogram `sucet` . ktorý pre daný parameter `n` vypočíta súčet prevrátených mocnín `2` , t.j.

   ```
   1/1 + 1/2 + 1/4 + 1/8 + 1/16 + 1/32 + ... 1/2**n
   ```

3.

Paralené priraďovanie

Na záver tejto časti ukážeme ďalšie možnosti priraďovacieho príkazu. Doteraz sme používali priradenie v tomto tvare:

```
premenná = hodnota
```

pri ktorom sa najprv vyhodnotila hodnota na pravej strane za znakom `=` (mohola to byť konštanta, premenná alebo aj zložitejší výraz) a táto hodnota sa priradila do danej premennej na ľavej strane priradenia. Videli sme, že takéto priradenie funguje napr. aj v tvare:

```
a = a + 1                       # zvýši premennú a o 1
sucet = sucet + cislo           # zvýši premennú sucet o hodnotu premennej cislo
veta = veta + 'abc'             # pridá na koniec reťazca veta reťazec 'abc'
xy = 2 * xy + '.'               # zdvojí obsah reťazca v premennej xy a pridá k tomu '.'
```

Vo všetkých týchto prípadoch sa na pravej strane vyskytla tá istá premenná, do ktorej sa potom priradí výsledok.

Lenže priradenie môžeme vylepšiť ešte takto:

```
premenná1, premenná2 = hodnota1, hodnota2
```

Toto označuje, že najprv sa vyhodnotia obe hodnoty za znakom `=` a potom sa obe tieto hodnoty priradia do zadaných premenných. Tychto premenných a potom aj výrazov môže byť aj viac, jediná podmienka je, že ich bude na oboch stranách rovnaký počet. Pozrime:

```
x, y = 100, 150
sirka, vyska = 10, 8
a, b, c = 1, 4, 2
meno, priezvisko = 'Janko', 'Hrasko'
```

Vo všetkých prípadoch sme **paralelne** priradili do viacerých premenných rôzne hodnoty. Stačí si uvedomiť, že v týchto prípadoch, napr.

```
x, y = 100, 150
```

označuje:

```
x = 100
y = 150
```

Výmena dvoch alebo viacerých hodnôt:

```
a, b = b, a
a, b, c = b, c, a
```