1. Zoznamy

(1) Napíšte funkciu, ktorá dostane ako parameter dva zoznamy a vráti ich kartézsky súčin.
(2) Napíšte funkciu, ktorá dostane dva zoznamy rovnakej dĺžky a vráti zoznam dvojíc, kde prvý prvok dvojice je z prvého zoznamu a druhý z druhého:

```
>>> myzip(["a","b"],[1,2])
[('a', 1), ('b', 2)]
>>>
```

(3) Napíšte funkciu, ktorá dostane ako parameter zoznam a vráti zoznam, v ktorom zlúči rovnaké po sebe idúce prvky.
(4) Napíšte funkciu, ktorá dostane ako parametre dva utriedené zoznamy a vráti zoznam, ktorý tieto dva zoznamy zlúči do utriedeného zoznamu. Využite pritom fakt, že sa jedná o utriedené zoznamy pre zrýchlenie výpočtu.
(5) Napíšte funkciu, ktorá dostane ako parametre dva zoznamy čísel f a d a vráti zoznam tých čísel z f ktoré sú deliteľné
   (a) všetkými číslami z d,
   (b) niektorým číslom z d,
   (c) práve dvomi číslami z d.

2. Slovníky

(1) Napíšte funkciu, `dict_join(`d_1`,`d_2`)` ktorá dostane ako parameter dva slovníky d_1, d_2 a vráti slovník d taký, že
   (a) d obsahuje práve tie kľúče, ktoré sú súčasne v d_1 aj d_2,
   (b) $d[k] = (d_1[k], d_2[k])$ pre k `in` d – t.j. hodnota $d[k]$ je n-tica hodnôt.
(2) Rozšírte predošlé cvičenie tak, aby fungovalo pre ľubovoľný počet parametrov. *Viď časť 4.7 Python tutorial, alebo prednáška.*
(3) Napíšte funkciu, ktorá ako parameter dostane zoznam reťazcov a vráti histogram reprezentovaný ako slovník.

```
>>> hist(["a","b","aa","c","d","e"])
{'a': 1, 'aa': 1, 'c': 1, 'b': 1, 'e': 1, 'd': 1}
>>> hist(list("kobyla␣ma␣maly␣bock"))
{'a': 3, '␣': 3, 'c': 1, 'b': 2, 'k': 2, 'm': 2, 'l': 2, 'o': 2, 'y': 2}
```

3. Relácie

V tejto je relácia reprezentovaná zoznamom dvojíc nejakých objektov, napríklad

```
[(1,2),(2,-1),(0,0)]
[("a","b"),("c","a")]
```

sú relácie, prvá je na množine $\{1, -1, 2, 0\}$, druhá na množine troch reťazcov.

Predpokladáme, že každá dvojica sa v zozname reprezentujúcom reláciu vyskytuje najviac raz.

Ak chcete mať jednoduché kódy, použite idióm (`R` je relácia)

```
for a,b in R:
      ...spracovanie a,b...
```

(1) Napíšte funkciu `reflexivna(R)`, ktorá vráti `True` práve vtedy, keď je `R` reflexívna relácia, inak vráti `False`. [1]

```
>>> R=[(1,1),(2,2),(1,2)]
>>> print reflexivna(R)
True
>>> R=[(1,2),(1,1)]
>>> print reflexivna(R)
False
```

(2) Napíšte funkciu `symetricka(R)`, ktorá vráti `True` práve vtedy, keď je `R` symetrická relácia, inak vráti `False`.
(3) Napíšte funkciu `antisymetricka(R)`, ktorá vráti `True` práve vtedy, keď je `R` antisymetrická relácia, inak vráti `False`.
(4) Napíšte funkciu `transitivna(R)`, ktorá vráti `True` práve vtedy, keď je `R` tranzitívna relácia, inak vráti `False`.
(5) Napíšte funkciu `to_dict(R)`, ktorá vráti slovník d charakterizovaný týmito vlastnosťami.
   - Kľúče v d sú práve tie prvky, ktoré sa vyskytujú v relácii `R` v prvej alebo druhej súradnici.
   - $d[x]$ je zoznam všetkých tých y, pre ktoré platí, že (x, y) je v `R`.

   ```
   >>> R=[(1,2),(1,3),(2,3)]
   >>> print to_dict(R)
   {1: [2, 3], 2: [3], 3: []}
   ```

(6) Napíšte funkciu `to_rel(d)`, ktorá je inverzná k funkcii `to_dict` z predošlého cvičenia.

4. Kombinatorika

(1) Napíšte funkciu `permute(`l`)`, ktorá dostane zoznam l ako parameter a vráti zoznam všetkých permutácií zoznamu l.

Funkcia bude zrejme rekurzívna. Ja som to spravil takto: ak l je prázdny, vráti zoznam obsahujúci prázdny zoznam. Ak l je neprázdny, odoberie z neho prvý prvok a zavolá `permute` *pre zoznam bez prvého prvku. Potom treba tento zoznam prejsť a vrátiť nový zoznam. Dávajte si pozor na to, aby ste na patričnom mieste vytvárali kópiu permutácie $n-1$ prvkov cez operátor* [:]. *Vsúvanie na pozíciu* `poz` *do zoznamu* `a` *sa robí takto:*

```
a[poz:poz]=[prvok]
```

(2) Napíšte funkciu `sublist(l,k)` ktorá má 2 parametre: zoznam l a číslo $len(l) \geq k \geq 0$. Vráti všetky zoznamy dĺžky k, ktoré vzniknú z l vynechaním niektorých prvkov.
(3) Napíšte funkciu, ktorá dostane ako parameter prirodzené čísla x, y a vráti ako výsledok vsetky cesty z bodu $(0, 0)$ do bodu (x, y) po štvorcovej sieti, pričom povolené pohyby sú iba „krok vpravo" a „krok hore".

[1] Tu predpokladáme, že doména relácie je množina všetkých prvkov, ktoré sa v nej vyskytujú.

5. Reťazce

(1) Napíšte funkciu s parametrom n, ktorá vráti zoznam tých čísel medzi 0 a n, ktoré sú deliteľné 7 a zároveň sú to symetrické slová, tzv. *palindrómy.*

```
>>> pal7(1000)
[0, 7, 77, 161, 252, 343, 434, 525, 595, 616, 686, 707, 777, 868, 959]
```

(2) Napíšte funkciu, ktorá dostane ako parameter reťazec a vráti všetky slová zo súboru `/usr/share/dict/words`, ktoré sú jeho prešmyčkou, anagramom. *Použite histogramy - viď patričné cvičenie o slovníkoch. Ak budete permutovať slová, budete čakať veľmi dlho.*

```
>>> anagrams("streets")
['setters', 'streets', 'tersest', 'testers']
>>> anagrams("integers")
['gentries', 'integers', 'steering']
```

(3) To isté ale s viacslovnými prešmyčkami. Vo výsledkoch sú povolené iba slová, ktoré majú viac ako 2 písmená.

```
>>> anagrams("i␣love␣you")
['voile␣you', 'you␣voile', 'you␣olive', 'olive␣you']
```

(4) Napíšte funkciu, ktoré ako parametre dostane dve anglické slová w_{start} a w_{end} rovnakej dĺžky a vráti zoznam anglických slov taký, že prvé slovo je w_{start}, posledné je w_{end} a každé dve susedné slová sú rovnaké až na jedno písmeno. Pre dvojicu slov `white` a `black` je riešenie takéto:

```
['white', 'whine', 'shine', 'shire', 'shirk', 'shark', 'shack', 'slack', 'black']
```

alebo iné riešenie:

```
['white', 'whine', 'thine', 'think', 'thick', 'chick', 'click', 'clack', 'black']
```

Pomôcky:

- *V každom prípade sa jedná o problém z teórie grafov.*
- *Vrcholy grafu sú slová danej dĺžky, hranou sú spojené tie, ktoré sa líšia o jedno písmeno.*
- *Graf je výhodné reprezentovať ako slovník, ktorého kľúče sú vrcholy a ktorého hodnoty sú množiny vrcholov, ktoré s daným kľúčom susedia.*
- *Problematické je zisťovanie, ktoré dve slová spolu susedia. Ja som si pomohol tak, že som množinu vrcholov obohatil o pomocné sprostredkujúce slová obsahujúce znak `'_'`; potom cesta z `'white'` do `'whine'` vyzerá takto:*

  ```
  ['white','whi_e','whine']
  ```

- *Ide to implementovať aj veľmi rýchlo, pomocou Dijkstrovho algoritmu. Rovnocenný je aj prístup pomocou prehľadávania do šírky, s použitím fronty.*

(5) Dotiahnite predošlé cvičenie tak, aby sa určil počet komponent súvislosti toho slovného grafu pre slová s daným počtom písmen. Pre štvorpísmenové slová mi vychádza presne 200 komponent súvislosti.

6. Počítanie atď.

(1) Vypočítajte súčet všetkých cifier v 1000!.
(2) Vypočítajte súčet všetkých cifier v 2^{1000}.
(3) Nájdite takú trojicu a, b, c kladných prirodzených čísel, pre ktorú platí $a + b + c = 1000$, $a^2 + b^2 = c^2$.
(4) Napíšte funkciu, ktorá vráti rozklad čísla na prvočísla ako zoznam.

```
>>> rozklad(2009)
[7, 7, 41]
```

(5) Nájdite prvých 10 čísel vo Fibonacciho postupnosti, ktoré sú prvočíslami.
(6) Goldbachova hypotéza hovorí, že každé párne prirodzené číslo väčšie ako 2 je súčtom dvoch prvočísel. Napíšte funkciu, ktorá „overí" Goldbachovu hypotézu pre svoj parameter.

7. Symbolické výpočty

Nasledujúce cvičenia reprezentujú racionálne [2] funkcie prefixovo takto (príklad)

$$\frac{x + 2y}{3 + \frac{2.5}{x}}$$

je reprezentovaná ako

`['/',['+','x',['*',2,'y']],['+',3,['/',2.5,'x']]]`.

Tomuto budeme hovoriť prefixová reprezentácia. Uvedomte si, že v tejto reprezentácii je konštantná funkcia všade rovná 5.2 reprezentovaná ako číslo `5.2` a funkcia všade rovná x reprezentovaná ako reťazec `'x'`.

Pre riešenie potrebujete použiť modul `types`. Viď dokumentácia.

Zrejme spracovanie takýchto prefixových reprezentácií bude rekurzívne.

(1) Napíšte funkciu `myevald(f,d)`, ktorá dostane ako parameter funkciu `f` v prefixovej reprezentácii a slovník `d`, ktorý zobrazuje všetky premenné použité v `f` na čísla. Ako návratovú hodnotu vráti hodnotu funkcie `f`.

```
>>> myevald(['+','x','y'],{'x':10,'y':20})
30
```

(2) Napíšte funkciu `myeval` tak, aby fungovala cez pomenované parametre: `myeval(['+','x','y'],x=2,y=5)` vráti 7. Využite predošlé cvičenie.
(3) Napíšte funkciu `compose(f,d)`, ktorá dostane ako parameter funkciu `f` v prefixovej reprezentácii a slovník `d`, ktorý zobrazuje premenné použité v `f` na funkcie v prefixovej reprezentácii. Ako návratovú hodnotu vráti zloženú funkciu v prefixovej reprezentácii. Dajte si pozor, aby ste skladanie robili správne: t.j. dosadzovaním „naraz" a nie „postupne":

```
compose(['+','x','y'],{'x':['+','x','y'],'y':'z'})
```

má vrátiť

```
['+',['+','x','y'],'z']
```

a nie

```
['+',['+','x','z'],'z'].
```

Keď to všetko zvládnete, pozrite sa na `http://code.google.com/p/sympy/` .

[2] Nemusíte implementovať umocňovanie na konštantu, stačí `['+','-','*','/']`.

8. Objekty a triedy

(1) Teraz ideme zabaliť prefixovo reprezentované funkcie tak, aby fungovali „objektovo".

Definujte triedu `class SymbFunc(object)` v module `symb.py` tak, aby fungovalo toto:

```
>>> import symb
>>> f=symb.SymbFunc(['+','x','y'])
>>> f.eval(x=10,y=20)
30
>>> f.evald({'x':10,'y':20})
30
```

(2) Premenujte teraz metódu `SymbFunc.eval` na `SymbFunc.__call__`. Dôsledok je, že to začne fungovať takto:

```
>>> import symb
>>> f=symb.SymbFunc(['+','x','y'])
>>> f(x=10,y=20)
30
```

Viď Python Reference Manual, časť 3.4.4.

(3) Pridajte metódu `__eq__(self,other)`, ktorá vráti `True` ak sú funkcie rovné. Pri určovaní rovnosti funkcií berte do úvahy komutativitu operácií + a *, t.j.

```
>>> f=symb.SymbFunc(['*',['+','x','y'],'z'])
>>> g=symb.SymbFunc(['*','z',['+','y','x']])
>>> f==g
True
```

Ignorujte všetko ostatné okrem komutativity.

(4) Skúste to teraz prerobiť tak, aby fungovalo aj volanie `f(10,20)`. Ktorá hodnota zodpovedá ktorej premennej je dané ich abecedným poradím.

Vhodné je vytvoriť abecedne utriedený zoznam premenných už v `SymbFunc.__init__`.