Izpit iz predmeta Programiranje 1, 28. avgust 2013. **Čas pisanja: 90 minut.**

Rešitve shranite **v eno samo datoteko s končnico .py** in jo oddajte prek Učilnice. Imena funkcij naj bodo takšna, kot jih predpisuje naloga. Rešitev preverite s testnimi primeri. Za rešitev lahko dobite določeno število točk, tudi če ne prestane testov. Funkcija, ki prestane vse teste, še ni nujno pravilna. Tudi za zelo okorno napisane rešitve bomo odbili nekaj točk.

Dovoljena je vsa literatura na poljubnih medijih in ves material, ki je objavljen na Učilnici. Študenti s predolgimi vratovi in podobnimi hibami bodo morali (najmanj!) zapustiti izpit, katerega opravljanje se bo štelo kot neuspešno.

1. Spremembe smeri

Napišite funkcijo `sprememb_smeri(s)`, ki kot argument prejme seznam z vrednostmi delnice ob koncu dnevnega trgovanja, kot rezultat pa vrne število sprememb gibanja. Če delnica nekaj dni narašča, nato pada, nato narašča, je smer spremenila dvakrat (iz naraščanja v padanje in iz padanje v naraščanje).

Primeri

```
>>> sprememb_smeri([1, 2, 3, 0, -1])
1  # iz naraščanja v padanje
>>> sprememb_smeri([-1, -2, 0, 1, -10, -5, 8])
3  # pada, narašča, pada, narašča
>>> sprememb_smeri([1, 2, 3, 4])
0  # samo narasca - ni sprememb
```

2. Nedostopna polja

V neki igri je mogoče z vsakega polja priti na določena druga polja. Možne poteze opišemo s slovarjem, ki kot ključe vsebuje imena polj, kot vrednosti pa množice polj, na katere je mogoče priti s posameznega polja. Tako

```
poti = {"a": {"c"},
        "b": {"a", "c", "f"},
        "c": {"d", "e"},
        "g": {"a"}}
```

pomeni, da s polja `a` pridemo na `c`; s polja `b` na `a`, `c` in `f`; s polja `c` na `d` in `e`; s polja `g` na `a`.

Poljem, na katera ne moremo priti z nobenega polja, bomo rekli nedostopna. V gornji igri sta to polji `b` in `g`.

Primeri

```
>>> nedostopna({"a": {"b"}})
{"a"}
>>> nedostopna({"a": {"b", "c"}, "b": {"c"}})
{"a"}
>>> nedostopna({"a": {"a"}})
set()
```

Napišite funkcijo `nedostopna(s)`, ki prejme slovar možnih potez in vrne množico nedostopnih polj.

Namig: naloga spraševanje po poljih, ki se pojavijo med ključi, med vrednostmi pa ne.

3. Je Fibonaci?

Zaporedje je Fibonacijevo, če je vsak člen enak vsoti prejšnjih dveh.

Seznam vsebuje Fibonacijevo zaporedje, če ima manj kot tri elemente ALI pa je zadnji element vsota predzadnjih dveh in je tudi seznam brez zadnjega elementa Fibonacijev.

Primeri

```
>>> je_fibo([2, 5, 7, 12, 19])
True
>>> je_fibo([1, 1, 2, 3, 12, 15, 27])
False
```

Napišite ***rekurzivno*** funkcijo `je_fibo(s)`, ki preveri, ali podani seznam vsebuje Fibonacijevo zaporedje.

Namig: stori, kar piše v drugem odstavku. Če si v dvomih, ali je tvoja rešitev zastavljena pravilno, vprašaj.

4. Obljudeni stolpci

Napišite funkcijo `po_stolpcih(s)`, ki prejme seznam šahovskih polj, na katerih stojijo figure in vrne seznam z osmimi števili, ki povedo, koliko figur stoji v posameznem stolpcu.

Primeri

```
>>> po_stolpcih(["C3", "H8", "B2", "C3"]
[0, 1, 2, 0, 0, 0, 0, 1]
>>> naj_stolpec(["C3", "H8", "B2", "C3"]
"C"
```

Poleg tega napišite funkcijo `naj_stolpec(s)`, ki pove, kateri je najbolj zasedeni stolpec. Če je več najbolj zasedenih stolpcev več, naj vrne prvega po abecedi.

Pomoč: funkcija `ord(c)` vrne kodo znaka (`ord("A")=65`), funkcija `chr(i)` pa vrne znak z določeno kodo (`chr(65)="A"`).

5. Ulomki

Podan je razred, ki predstavlja ulomke (glej datoteko s testnimi primeri). Ta ima konstruktor, ki mu podamo števec in imenovalec, metodo, ki okrajša ulomek ter dve metodi, ki sta povezani z izpisom in te njuno delovanje ne zanima.

Dodajte mu metodi `pristej(self, drugi_ulomek)` in `pomnozi(self, drugi_ulomek)`, ki ulomku prišteje drugi ulomek oz. ga z njim pomnoži.

Ulomki morajo biti vedno shranjeni v okrajšani obliki.

Namig: uporabljaj metodo `okrajsaj`.

Primeri

```
>>> a = Ulomek(3, 8)
>>> a
3/8
>>> b = Ulomek(2, 16)
>>> b
1/8
>>> a.pristej(b)
>>> a
1/2
>>> a.pomnozi(b)
>>> a
1/16
>>> c = Ulomek(2, 1)
>>> c
2/1
>>> a.pomnozi(c)
>>> a
1/8
```