Tartalomjegyzék

Feladatok

Elmaradt feladat

1. Zárt terület kifestése

Olvassuk be a picture.txt fájlt listák listájába (minden karakter egy elem)! Írjunk egy fill(x,y) függvényt, ami ugyanazt csinálja, mint a Paint kitölt? funkciója! Az (x,y) pontból kiindulva a . helyére # jelet tesz, amíg a # jel által jelölt falba nem ütközik! A módszer rekurzív: kifestjük az (x,y) pontot, majd a szomszédait, ha azok nem # jelek. Hívjuk meg a szomszédokra (akik nem # jelek) a függvényt rekurzívan. Ha nincs kit kiszínezni, akkor álljunk meg!

```
.....................................
...#######################...........
...#.....................#...........
...#.....................#...........
...#.....................#...........
...#.....................#...........
...#.....................#...........
...#.....................#######.....
...###.................##......#.....
```

```
...#..##............##........#.....
...#....##........##..........#.....
...#......##.....##............#.....
...#........#####..............#.....
...#........#..................#.....
...#.......##..................#.....
...#.....##....................#.....
...#...##......................#.....
...############################.....
....................................
....................................
....................................
....................................
```

Lekezelések

2. Osztás

Írjunk egy divide() nev? függvényt, aminek nincs paramétere. A függvény futásakor inputként kér két számot és kimenetként a két szám hányadosát adja vissza. Kezeljük le azokat a kivételeket ha nem számot adunk meg, hanem például stringet, illetve azt az esetet, amikor nullával osztanánk.

Függvényekr?l általában

3. lista összeg

Hívjunk fának egy objektumot ha szám, vagy ha fák listája. Tehát például 0, 1 és 2 fák, mert számok, és így [0, 1, 2] is fa, mert fák listája. Ugyanezért [0, 1, [0, 1, 2], 2] is fa, és [0, [0, 1, 2], [0, [0, 1, 2], 1, [0, 1, 2], 2], 2] is az.
Írjon egy sumtree() nev? függvényt, amely egy fában szerepl? számok összegét adja vissza. Például a fenti utolsó fára 14-et.

4. részlisták

`Írjon egy sublists() nev? függvényt, amely az argumentumaként adott`

lista összes részlistájának listáját adja vissza (tetsz?leges sorrendben). (Itt most l1 részlistája l2-nek, ha kalkulus értelemben részsorozata, azaz ha l1 minden tagja szerepel l2-ben, mégpedig ugyanabban a sorrendben.) Például:

```
sublists([])
[[]]
sublists([1])
[[1], []]
sublists([1,2])
[[1, 2], [1], [2], []]
sublists([1,2,3])
[[1, 2, 3], [1, 2], [1, 3], [1], [2, 3], [2], [3], []]
```

5. függvény meghívás

Definiáljon egy kétargumentumú apply() függvényt, ami az els? argumentumát (egy egyargumentumú függvény) alkalmazza a második argumentumára, és az eredményt adja vissza.

6. többszörös kompozíció

Definiáljon egy kétargumentumú self_compose függvényt úgy, hogy ha fun egyargumentumú függvény és n természetes szám, akkor self_compose(fun, n) fun n. hatványát (azaz n-szeres kompozícióját saját magával) adja vissza. Speciálisan, self_compose(fun, 1) magát fun-t, self_compose(fun, 2) pedig fun kompozícióját magával. Mit kellene visszaadnia self_compose(fun, 0)-nak? Ha nem tudja, tegye fel (és ellen?rizze assert-tel), hogy a második argumentum mindig pozitív.
Például:

```
(self_compose(lambda x: x+1,3))(0)
3
```

7. Átlag

Írjunk egy atlag() nev? függvény, melynek tetsz?legesen sok bemenete van és kimenetnénk a bemeneti számok átlagát adja vissza. Kezeljük le azt az esetet ha a bemenetek között van nem szám típusú objektum.

8. jó zárójelezés

Írjunk egy függvényt, aminek a bemenete egy string és eldönti, hogy jól van-e zárójelezve, vagyis, hogy minden nyitó '(' zárójel után következik-e megfelel?en egy ')' zárójel valahol a stringben.