Szkriptnyelvek

– 2. zárthelyi dolgozat –

Szathmáry László

2021. december 7., 8.00

1. feladat – bináris mátrix (5 pont)

A bemeneti állományban egy bináris mátrix található. Példa (lásd `example.txt`):

```
00100
11110
10110
10111
10101
01111
00111
11100
10000
11001
00010
01010
```

Állítsuk elő az `output.txt` állományt, melynek az `i.` sorába a mátrix `i.` oszlopát írjuk be. Vagyis vesszük a mátrix első oszlopát, s ezt beírjuk a kimeneti állomány első sorába, stb.

Ha ezzel kész vagyunk, akkor az állományba tegyünk be egy elválasztó vonalat, majd írjuk be az `output.txt`-be beírt bináris számok összegét decimális formában.

A művelet végén az `output.txt` állomány tartalma így fog kinézni a fenti példa esetén:

```
011110011100
010001010101
111111110000
011101100011
000111100100
---
sum: 9512
```

Dolgozza fel az `input.txt` állományt és állítsa elő a kimeneti fájlt.

A program a futtatás során adjon egy kis visszajelzést:

```
$ ./feladatN.py
-> input.txt beolvasva
-> output.txt létrehozva
```

Figyelem #1! A programnak az `input.txt` állományt kell feldolgoznia!

Figyelem #2! Csakis a standard könyvtár használható!

2. feladat – prímszámok (4 pont)

Legyen adott a következő PHP állomány (lásd `example.php`):

```php
<?php $data = array (
  0 => 2,
  1 => 3,
  2 => 5,
  3 => 7,
); ?>
```

Az állományban a nyíl (`=>`) után a prímszámok vannak felsorolva. A nyíl előtt az indexek láthatók.[1] Dolgozzuk fel az állomány tartalmát s állítsunk elő egy JSON fájlt (`primes.json` néven) a következő szerkezettel:

```json
{
  "description": "list of prime numbers",
  "data": [2, 3, 5, 7]
}
```

Mint látható, a `data` nevű tömbben az input fájlból beolvasott prímszámok szerepelnek.

Figyelem! A JSON kimenetet **ne** manuálisan rakja össze sztringekből![2] Használja a standard könyvtár `json` modulját!

A programot úgy írja meg, hogy az a `primes.php` fájl tartalmát dolgozza fel. A program a futtatás során adjon egy kis visszajelzést:

```
$ ./feladatN.py
-> primes.php beolvasva
-> primes.json létrehozva
```

Figyelem! A programnak a `primes.php` állományt kell feldolgoznia!

[1] A primes.php az első százezer prímszámot tartalmazza.
[2] instant 0 pont

3. feladat – palindróm prímek (4 pont)

Az előző feladatban előállított `primes.json` fájlt másolja át ezen feladat mappájába. Itt most ez lesz a bemenet.

A `primes.json` az első százezer prímszámot tartalmazza.

Írjon programot, ami beolvassa a `primes.json` fájlban található prímszámokat[3], majd kiírja a palindróm prímszámok összegét. A 101 például palindróm szám, mivel visszafelé olvasva is ugyanazt a számot kapjuk.

Vegyük például az előző feladatban látható JSON fájlt:

```
{
  "description": "list of prime numbers",
  "data": [2, 3, 5, 7]
}
```

Ez csupán 4 db prímszámot tartalmaz (2, 3, 5, 7). Ezek mindegyike palindróm, így a palindróm prímek összege: 17.

A programot úgy írja meg, hogy az a `primes.json` fájl tartalmát dolgozza fel.

```
$ ./feladatN.py
28
```

28 helyett természetesen a helyes eredményt kell majd kiírni.

Figyelem! A programnak a `primes.json` állományt kell feldolgoznia!

[3] a ‘json‘ modul segítségével

4. feladat – jelszó generátor (5 pont)

Írjon egy programot, ami egy *erős jelszót* állít elő. Egy erős jelszónak a következő kritériumoknak kell megfelelnie:

- legalább 8 karakter hosszú
- tartalmaz legalább egy kisbetűt az angol ábécéből
- tartalmaz legalább egy nagybetűt az angol ábécéből
- tartalmaz legalább egy számjegyet
- tartalmaz legalább egy speciális karaktert

Speciális karakternek a következőket tekintjük:

`.,;_'*"`

Vagyis: pont, vessző, pontosvessző, aláhúzás, aposztróf, csillag, ill. idézőjel.

Hozzunk létre egy `password` nevű modult, s abban helyezzünk el egy `get_password()` nevű függvényt.

A `get_password()` formális paraméterlistáján legyen egy `length` nevű *opcionális* paraméter, melynek az alapértelmezett értéke legyen 8. A `length` segítségével a generálandó jelszó hosszát adhatjuk meg. A függvény adjon vissza egy `length` hosszúságú erős jelszót.

Használata (példa):

```
import password

p1 = password.get_password()
assert len(p1) == 8
p2 = password.get_password(length=12)
assert len(p2) == 12
```

A generálandó jelszó hosszát futásidejű paraméterként kelljen megadni. A program csak egyetlen paramétert fogadjon el. Ha a hossz kisebb mint 8, akkor hibaüzenet.

Azt is kezeljük le, ha a felhasználó nem számot ad meg!

Futási példák:

```
$ ./feladatN.py
Adj meg egy számot!

$ ./feladatN.py na
Számot adj meg!

$ ./feladatN.py 9 12
Egyetlen számot adj meg!

$ ./feladatN.py 5
A szám értéke legyen legalább 8

$ ./feladatN.py 8
V5i9wT,9

$ ./feladatN.py 12
E6NEnE_'1*6v
```

Folytatás a következő oldalon!

Mivel véletlenszerű jelszót állítunk elő, ezért a program minden futtatás során más és más kimenetet produkál.

Próbáljuk elérni azt, hogy a generált jelszóban a különböző karaktercsoportba tartozó karakterek vegyesen szerepeljenek. Vagyis **ne az legyen** például, hogy a jelszóban mindig két kisbetű szerepel, utána két nagybetű, majd két számjegy, végül két speciális karakter. Az ilyet próbáljuk elkerülni!