01_mixed

July 31, 2019

Before you turn this problem in, make sure everything runs as expected. First, **restart the kernel** (in the menubar, select Kernel→Restart) and then **run all cells** (in the menubar, select Cell→Run All).

Make sure you fill in any place that says `YOUR CODE HERE` or "YOUR ANSWER HERE", as well as your name and collaborators below:

```
In [ ]: NAME = "PETROVICI STEFAN"
        COLLABORATORS = "?"
```

---

1 Files

1.0.1 1. (1 point)

List all files from a directory and move all `.txt` to a newly created folder named `copy`.

```
In [4]: !touch 1.txt
        !touch 2.txt
        !touch 3.txt

In [15]: mkdir copy

In [16]: !find -type f -name "*.txt" -exec mv {} ./copy/{} \;

mv: cannot move './copy/3.txt' to './copy/./copy/3.txt': No such file or directory


In [ ]: #trebuia sa fac cu subprocess.call() defapt

In [1]: ls ./copy

1.txt  2.txt  3.txt
```

1.0.2 2. (1 point)

Read first n lines from a file. Don't actually read the whole file.

```
In [2]: num_lines = 5
```

```
In [4]: %%writefile test.txt
        a
        b
        c
        d
        e
        f
        g
```

```
Writing test.txt
```

```
In [12]: with open("test.txt", "r") as f:
             idx = 0
             for line in f:
                 print(line.strip())
                 idx += 1
                 if idx == num_lines: break
```

```
a
b
c
d
e
```

1.0.3 3. (1.5 point)

Read last n lines from a file.
`file.seek()` for moving file cursor and use a buffer to read.

```
In [23]: num_lines = 2
         fil = open("test.txt","r")
         lines = fil.readlines()[-num_lines:]
         for line in lines:
             print(line.strip())
         fil.close()
```

```
f
g
```

1.0.4 4. (1.5 point)

Print recursively all the sizes of the files the current directory and sum them up.
`os.stat()` or `or.path.getsize()` to get size of a file
Ex output:

```
F 01.ipynb                                -       6519 Bytes
F test.txt                                -        116 Bytes
D .ipynb_checkpoints
F      01-checkpoint.ipynb                -       6627 Bytes
Total: 13262
```

```
In [44]: import os, stat

         sz = 0
         for f in os.listdir("."):
             print(f.strip(),end="")
             print(" ",end="")
             sz += os.stat(f).st_size
             print(os.stat(f).st_size)

         print("total : {}".format(sz))
```

```
02_classes.ipynb 15394
01_mixed.ipynb 9082
copy 4096
test.txt 13
.ipynb_checkpoints 4096
total : 32681
```

2 Datetime

2.1 1. (1 point)

Print a simple formatted calendar from a month and year given from user input.
Print an asterisk near the 16-th and 21-th day.
The calendar should start from Thursday.
Look into `calendar` library
Ex output:

```
      December 2011
Thu Fri Sat Sun Mon Tue Wed
  1   2   3   4   5   6   7
  8   9  10  11  12  13  14
 15 *16  17  18  19  20 *21
 22  23  24  25  26  27  28
 29  30  31
```

```
In [4]: import calendar

        yyyy = 2019
        m = 7
        cdar = calendar.TextCalendar(firstweekday=3)
        c = cdar.formatmonth(yyyy, m, 7, 1)
        st = c.find("16")
        to = c.find("21")

        s = list(c)
        s[st-1] = "*"
        s[to-1] = "*"
        print("".join(s))
```

```
                     July 2019
  Thu      Fri      Sat      Sun      Mon      Tue      Wed
                                       1        2        3
    4        5        6        7        8        9       10
   11       12       13       14       15      *16       17
   18       19       20      *21       22       23       24
   25       26       27       28       29       30       31
```

3 Random

3.0.1 1. (1 point)

Generate a ten-character alphanumeric password with at least one lowercase character, at least one uppercase character, at least one digits and at least one special character.

`string` module has theese characters as lists. ex: `string.ascii_lowercase` has all ascii lowercase letters.

```
In [24]: import string
         import random

         stp = ''.join(random.sample(string.ascii_lowercase,1) + random.sample(string.ascii_uppe

         r = list(stp)
         random.shuffle(r)
         print(''.join(r))
```

```
Jbarf6ced(
```

4 Decorators

4.1 1. (1 point)

Write a decorator which wraps functions to log function arguments and the return value on each call. Provide support for both positional and named arguments (your wrapper function should take both *args and **kwargs and print them both).

```
>>> @logged
... def func(*args):
... return 3 + len(args)
>>> func(4, 4, 4)
you called func(4, 4, 4)
it returned 6
6
```

```
In [30]: def new_decorator(func):

             def wrap_func(*list, **dict):

                 print("you called {}(".format(func.__name__),end="")
                 for el in list:
                     print(el,end=",")
                 for el in dict:
                     print(el[1],end=",")
                 print(")")

                 a = func(*list, **dict)

                 print("it returned {}".format(a))

             return wrap_func

         def f(a,b):
             return a+b

         ff = new_decorator(f)
         ff(4,5)
```

```
you called f(4,5,)
it returned 9
```

4.2 2. (1 point)

Write a decorator to cache function invocation results. Store pairs arg:result in a dictionary in an attribute of the function object. The function being memoized is:

```
def fibonacci(n):
    assert n >= 0

    if n < 2:
        return n

    return fibonacci(n-1) + fibonacci(n-2)
```

ps: think in which context you should define the cache

```
In [8]: d = dict()

        def fibonacci(n):
            assert n >= 0

            if n < 2:
                return n

            return fibonacci(n-1) + fibonacci(n-2)


        def new_decorator2(func, cashe):

            def newFunc(n):
                global c
                if cashe.get(n) != None:
                    c = 0
                    return cashe.get(n)
                if c == 0 and cashe.get(n) == None:
                    c = 1
                    cashe[n] = newFunc(n)
                    return cashe[n]

                return newFunc(n)


            return newFunc

        f = new_decorator2(fibonacci, d)
        c = 0
        f(5)
        print(d)
```

```
{5: 5}
```

5 Regex

Go trough all the exercises on RegexOne

5.0.1 1. Extract the user id, domain name and suffix from the following email addresses. (1 point)

```
emails = """zuck26@facebook.com
page33@google.com
jeff42@amazon.com"""

desired_output = [('zuck26', 'facebook', 'com'),
 ('page33', 'google', 'com'),
 ('jeff42', 'amazon', 'com')]
```

hint : `re.findall`

5.0.2 2. Split the following irregular sentence into words (1 point)

```
sentence = """A, very   very; irregular_sentence"""
desired_output = "A very very irregular sentence"
```

hint: `re.split`