PE2: PE of 23/11/2018

Master in Informatics and Computing Engineering
Programming Fundamentals
Instance: 2018/2019

Here you have a possible solution for each question of the Practical on computer Evaluation.

1. Find the treasure

The path to the treasure is given as a sequence of commands that are steps of length 1: *up*, *left*, *right* or *down*. Write a function **`map(pos, steps)`** that takes a coordinate **`pos`**, which is a tuple with values x and y as (x,y), and a sequence of commands in a string **`steps`**, with the steps separated by a hyphen, and computes the final position in the map.

Save the program in the file **`map.py`** inside the folder **`PE2`**.

For example:

- `map((0,0), "up-up-left-right-up-up")` returns the tuple: `(0,4)`
- `map((0,4), "up-up-left-left-up-up")` returns the tuple: `(-2,8)`

Solution:

```
def map(pos, steps):
    posx = pos[0]
    posy = pos[1]

    for s in steps.split("-"):
        if s == "up":
            posy += 1
        elif s == "down":
            posy -= 1
        elif s == "left":
            posx -= 1
        elif s == "right":
            posx += 1

    return posx, posy
```

2. The greatest number

Write a Python function **`greatest(num)`** that, given a non-negative integer **`num`**, computes the greatest number that can be made using all digits of **`num`**.

Save your program in the file **`greatest.py`** inside the folder **`PE2`**.

For example:

- **`greatest(310909)`** returns the integer: **`993100`**
- **`greatest(7187)`** returns the integer: **`8771`**
- **`greatest(99)`** returns the integer: **`99`**

Solution:

```
def greatest(number):
    """ computes the big number with given digits """
    result = 0
    astring = str(number)

    while astring != "":
        # find the largest digit
        bigest_digit = max(astring)
        # find its position and remove digit for next iteration
        pos = astring.index(bigest_digit)
        astring = astring[:pos] + astring[pos+1:]
        # update final number
        result = result * 10 + int(bigest_digit)

    return result
```

Another more "pythonic" solution:

```
def greatest(n):
    """ computes the big number with given digits """
    return int(''.join(sorted(list(str(n)), reverse=True)))
```

3. Formatting strings

Write a Python function **`exactly(s)`** that, given a string **`s`**, where each character is guaranteed to be a lowercase letter, a digit or a question mark, checks if there are exactly three question marks between all pairs of digits whose sum is exactly 10. The function must return a properly formatted string:

`The sequence <s> is OK with the pairs: <t>`

or

`The sequence <s> is NOT OK with first violation with pair: <t>`

if the string respects or not the restriction, respectively. `<s>` represents the input string and `<t>` represents a tuple with all concatenated pairs of digits that meet the condition or the first concatenated pair that does not meet the condition, respectively.

Save your program in the file **`exactly.py`** inside the folder **`PE2`**.

For example:

- `exactly("acc?7??sss?3rr1??????5???5")` returns the string:
  `The sequence acc?7??sss?3rr1??????5???5 is OK with the pairs: ('73', '55')`
- `exactly("acc?7??sss3rr1??????5")` returns the string:
  `The sequence acc?7??sss3rr1??????5 is NOT OK with first violation with pair: ('73',)`
- `exactly("aa6?9")` returns the string:
  `The sequence aa6?9 is OK with the pairs: ()`

Solution:

```
def exactly(s):
    tup = ()
    for i in range(len(s)-1):
        for j in range(i+1, len(s)):
            if s[i].isdigit() and s[j].isdigit() and int(s[i]) + int(s[j]) ==
10:
                new = s[i+1:j]
                if new.count('?') == 3:
                    tup += (s[i] + s[j],)
                else:
                    tup = (s[i] + s[j],)
                    return 'The sequence {0} is NOT OK with first violation
with pair: {1}'.format(s, tup)

    return 'The sequence {0} is OK with the pairs: {1}'.format(s, tup)
```

4. Genealogy by order

Susana needs to build a genealogy tree of her family for her school homework. She has asked her family and written everything as a list of tuples, where each tuple is **`(name, relationship)`**. The relationship is given as "sibling", “parent”, “cousin” or “grandparent".

For example:

```
l=[("maria", "parent"), ("matilde", "grandparent"),
   ("geraldes", "grandparent"), ("carlos", "sibling"),
   ("paulo", "sibling"), ("artur", "grandparent"),
   ("pedro", "parent"), ("alfredo", "cousin"), ("carla", "cousin")]
```

Write a Python function **`genealogy(l)`** to help her order the family. The order is given by relationship using the following rule: *sibling* < *parent* < *cousin* < *grandparent*. When there is a draw, use the relative's name by ascending order.

Save your program in the file **`genealogy.py`** inside the folder **`PE2`**.

For example:

- **`genealogy(l)`** (where **`l`** is the previous list) returns the list:
  `[('carlos', 'sibling'), ('paulo', 'sibling'), ('maria', 'parent'), ('pedro', 'parent'), ('alfredo', 'cousin'), ('carla', 'cousin'), ('artur', 'grandparent'), ('geraldes', 'grandparent'), ('matilde', 'grandparent')]`
- **`genealogy([("sofia", "sibling"), ("sara", "parent"), ("bernardo", "parent")])`** returns the list:
  `[('sofia', 'sibling'), ('bernardo', 'parent'), ('sara', 'parent')]`

Solution:

```
def f_order(elem):
    """ function to give the order of items """
    name, kinship = elem
    order = ("sibling", "parent", "cousin", "grandparent").index(kinship)

    return order, name

def genealogy(l):
    return sorted(l, key=f_order)
```

5. Caesar cipher with Fib

Caesar encrypted the messages he sent to his generals by **left shifting** all letters in the message by `s` ∈ ℤ places in the alphabet. For example, with a left shift of 2, C would be replaced by A, D would become B, and so on.

Write a Python function `caesar(message)` that uses a slightly more sophisticated cipher. Instead of applying the same shift to all the letters, a variable shift is used. Specifically, the shift to be applied to the *n*-th character in the string will be given by the *n*-th value in Fibonacci's sequence F_n given by the the formula:

$$F_n = \frac{(1+\sqrt{5})^n - (1-\sqrt{5})^n}{2^n\sqrt{5}}$$

You can assume that all letters will be uppercase and special characters (like spaces, commas, etc.) are not to be ciphered. For example:

Message	**H**	**E**	**L**	**L**	**O**		**W**	**O**	**R**	**L**	**D**	**!**
Fibonacci's sequence	**0**	**1**	**1**	**2**	**3**	**5**	**8**	**13**	**21**	**34**	**55**	**89**
Ciphered message	**H**	**D**	**K**	**J**	**L**		**O**	**B**	**W**	**D**	**A**	**!**

You may use the remainder operator (%) to handle shifts that circle back to the end of the alphabet, i.e. when you reach the beginning of the alphabet: 4 % 26 (= 4), -4 % 26 (= 22), -30 % 26 (= 22).

Save the program in the file `caeser.py` inside the folder `PE2`.

For example:

- `caesar("HELLO WORLD!")` returns the string: `HDKJL OBWDA!`
- `caesar("CAESAR CIPHER")` returns the string: `CZDQXM PNHETD`
- `caesar("FIBONACCI SEQUENCE")` returns the string: `FHAMKVUPN PTCVRBDT`

Solution:

```
from math import sqrt
import string

def fib(n):
    return int(((1+sqrt(5))**n-(1-sqrt(5))**n)/(2**n*sqrt(5)))

def caesar(message):
    message = message.upper()
    alpha = string.ascii_uppercase  # "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
    result = ""
    for idx, letter in enumerate(message):
        if letter in alpha:
            caesar_index = (alpha.find(letter) - fib(idx)) % len(alpha)
            result += alpha[caesar_index]
        else:
            result += letter
    return result
```

Another solution using **`ord()`** and **`char()`**:

```
from math import sqrt

def fib(n):
    return int(((1+sqrt(5))**n-(1-sqrt(5))**n)/(2**n*sqrt(5)))

def caesar(message):
    message = message.upper()
    result = ""
    for idx, letter in enumerate(message):
        if letter.isalpha():
            result += chr((ord(letter)-ord('A') - fib(idx)) % 26 + ord('A'))
        else:
            result += letter
    return result
```

The end.

FPRO, 2018/19