PE02: PE of 07/11/2019 (solutions)

Master in Informatics and Computing Engineering
Programming Fundamentals
Instance: 2019/2020

*An example of solutions for the 5 questions in this **Practical on computer evaluation**.*

1. Iterate lists by reverse order

Write a Python function `iterate_rev(names, numbers, emails)` that given three tuples of equal size `names` (strings), phone `numbers` (integers) and `emails` (strings), returns a string with "name - number - email" in each line, in reverse order.

For example:

- if `names = ("ana", "bernando", "carlos")`, `numbers = (938421028, 916381961, 939090082)`, `emails=("ana.serra@gmail.com", "b1999@hotmail.com", "up201945321@fe.up.pt")`, then the result is `"carlos - 939090082 - up201945321@fe.up.pt\nbernardo - 916381961 - b1999@hotmail.com\nana - 938421028 - ana.serra@gmail.com"`

Solution:

```
def iterate_rev(names, numbers, emails):
    result = ''
    for i in range(len(names)):
        result = "{0} - {1} - {2}\n{3}".format(
                names[i], str(numbers[i]), emails[i], result)
    return result
```

2. Group combinations

In the classroom, the teacher wants to find the possible groups of **three** students. The students in the class are given by the tuple `students`.

For example, if `students = ("Ana", "Bernardo", "Carla", "Daniel")`, then the following groups are possible:

1. `('Ana', 'Bernardo', 'Carla')`
2. `('Ana', 'Bernardo', 'Daniel')`
3. `('Ana', 'Carla', 'Daniel')`
4. `('Bernardo', 'Carla', 'Daniel')`

Write a Python function `groups(students)` that returns a tuple containing tuples with the possible combinations of groups of three students. If no group is possible, return an empty tuple.

Solution:

```
def groups(tup):
    rtup = ()
    for i in range(0, len(tup)):
        for j in range(i+1, len(tup)):
            for k in range(j+1, len(tup)):
                rtup += ((tup[i], tup[j], tup[k]),)
    return rtup
```

3. Order digits

Write a Python function **`order_digits(n)`** that, given an integer number **`n`** greater or equal to zero, returns a tuple with the digits of the given number in decreasing order.

For example:

- if **`n = 9013322`** then the result is the tuple **`(9, 3, 3, 2, 2, 1, 0)`**

Solution:

```
def order_digits(n):
    if n == 0:
        return (0,)
    result = ()
    for i in range(9, -1, -1):
        aux = n
        while aux > 0:
            if aux % 10 == i:
                result += (i, )
            aux //= 10
    return result

# another solution using sorted()
def order_digits2(n):
    if n == 0:
        return (0,)
    a_list = ()
    while n > 0:
        a_list += (n % 10,)
        n //= 10
    return tuple(sorted(a_list, reverse=True))
```

4. Scores

Consider the *leaderboard*, given as a nested tuple of N tuples in the format *(name, (score_1, score_2, …))*, with the *names* that tried one assessment and the *score* of each attempt.

Write a Python function `sort_leaders(records)` to sort the leaderboard records according to the following criteria:

1. Sort based on the maximum score (descending order);
2. Then sort based on minimum number of tries (ascending order);
3. Then sort based on student name (ascending order);

where *name* is a string and each *score* is an integer between 0 and 100.

For example:

- if `records = (('João', (80, 90, 100)), ('Ana', (90, 100)), ('José', (100,)), ('Ana', (50, 90)), ('Rui', (50, 90)))` then the result is the tuple `(('José', (100,)), ('Ana', (90, 100)), ('João', (80, 90, 100)), ('Ana', (50, 90)), ('Rui', (50, 90)))`

Solution:

```
def sort_rule(item):
    return (100 - max(item[1]), len(item[1]), item[0])

def sort_leaders(records):
    return tuple(sorted(records, key=sort_rule, reverse=False))
```

5. Number of patterns in a word

Write a Python function **`subpatterns(astring)`** that receives a string **`astring`** and computes the number of its substrings that follow a certain pattern.

The *pattern* is matched if the number of times that the substring *grows* is the same as the number of times the substring *shrinks*. A substring is *growing* at a particular character, if the character is bigger than the previous character, and is *shrinking* if it is smaller than the previous one. For example, **`"ceda"`** grows once (**`'c'`** < **`'e'`**) and shrinks twice (**`'e'`** > **`'d'`** and **`'d'`** > **`'a'`**). In this case, there's one substring pattern match (**`"ce"`** followed by **`"ed"`**).

The function returns a string with the same format as in the following examples.

For example:

- If **`astring = "aghljcb"`** then the result is the string **`"The string 'aghljcb' contains 3 substring patterns."`**

Solution:

```
def is_pattern(astring):
    net = 0
    for i in range(len(astring)-1):
        if astring[i+1] > astring[i]:
            net += 1
        if astring[i+1] < astring[i]:
            net -= 1
    return net == 0

def subpatterns(astring):
    counter = 0
    for step in range(2, len(astring) + 1):
        for char_index in range(0, len(astring) - step + 1):
            sub_str = astring[char_index:char_index+step]
            if is_pattern(sub_str):
                counter += 1
    return "The string '{0}' contains {1} substring patterns.".format(
            astring, counter)
```

The end.

FPRO, 2019/20