Assignment #11: FP with Collections

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

***Goals**: write programs using Effect-free programming style*

***Pre-requirements (prior knowledge)**: see the bibliography of Lecture #19 an Lecture #20*

***Rules**: you may work with colleagues, however, each student must write and submit in Moodle his or her this assignment separately. Be sure to indicate with whom you have worked. We may run tools to detect plagiarism (e.g. duplicate code submitted)*

***Deadline**: 8:00 Monday of the week after (17/12/2018)*

***Submission**: to submit, first pack your files in a folder RE11, then compress it with zip to a file with name `2018xxxxx.zip` (your_code.zip) and last (before the deadline) go to the Moodle activity (**you have only 2 attempts**)*

1. Sort by lambda

Write a Python function `sort_by_f(l)` that, given list `l`, returns the list sorted using a lambda function, defined as:

$$f(x) = \begin{cases} 5 - x & x \geq 5 \\ x & x < 5 \end{cases}$$

Save the program in the file `sort_by_f.py`.

For example:

- `sort_by_f([-10, -6, 2, 5, 90])` returns the list `[90, -10, -6, 5, 2]`
- `sort_by_f([-1, -2, 2, 15, 99])` returns the list `[99, 15, -2, -1, 2]`

2. Map-Reduce

Write a Python function `map_reduce(n1, n2)` whereby you create a list of the square of the odd numbers between `n1` and `n2`. Then use reduce to multiply if the accumulated result is smaller than 50 or add the numbers otherwise. Ensure the result of the function is an integer.

Have a look at `reduce()` from module `functools` (Higher-order functions and operations on callable objects).

Save the program in the file `map_reduce.py`.

For example:

- `map_reduce(0, 10)` returns the integer `164`
- `map_reduce(5, 100)` returns the integer `166640`

3. Odd Range

Write a generator function **`odd_range(start, stop, step)`** that yields the odd numbers between **`start`** and **`stop`** with a **`step`** increment between odd numbers.

Save the program in the file **`odd_range.py`**.

A generator function contains one or more *yield* statements, instead of a *return* statement. Therefore, when a generator function is called, it returns an iterator object, that can be used to iterate through the items using the *next()* function or by simply looping over it.

For example:

- **`odd_range(1, 10, 1)`** returns a generator that produces **`[1, 3, 5, 7, 9]`**
- **`odd_range(100, 150, 5)`** returns a generator that produces **`[101, 111, 121, 131, 141]`**
- **`odd_range(10, 0, 1)`** returns a generator that produces **`[]`**

4. Override operation

Write a function **`override(l1, l2)`** that, given two lists of tuples, performs the operation **`l1 ++ l2`** known as *override*. The *override* operation, given two lists of tuples, produces a new list with every member of **`l2`** and every member of **`l1`** that is not overridden by an element from **`l2`** (i.e., does not begin with the same atomic element). For example,

[(a,b), (c,d), (c, e)] ++ [(a,c), (b,d)] = [(a,c), (b,d), (c,d), (c,e)]

Note that (a,b) from the first list was overridden by (a,c) from the second list and all others elements were maintained.

The resulting list should be ordered by the first element of each tuple.

Save the result in the file **`override.py`**.

For example:

- **`override([('c','d'),('c','e'),('a','b'),('a', 'd')], [('a','c'),('b','d')])`** returns the list **`[('a', 'c'), ('b', 'd'), ('c', 'd'), ('c', 'e')]`**
- **`override([('a','b','c','e'),('f', 'p', 'r', 'o')], [('a','c'),('b','d')])`** returns the list **`[('a', 'c'), ('b', 'd'), ('f', 'p', 'r', 'o')]`**
- **`override([('a','b'),('c','d')], [('b','a'),('d','c')])`** returns the list **`[('a', 'b'), ('b', 'a'), ('c', 'd'), ('d', 'c')]`**

5. Map, Filter & Reduce

Write a function **`reduce_map_filter(args)`** that receives a collection of arguments called **`args`** and returns the result of processing **`args`**. This **`args`** can be either a list of integers or a tuple of type **`(op, f, args)`**, where **`op`** is an operator (**`"map"`**, **`"filter"`** or **`"reduce"`**), **`f`** is a function and **`args`** is again a list of integers or a tuple. Each function **`f`** must be applied in a chain according to the **`op`** operator.

Save the program in the file**`reduce_map_filter.py`**.

For example:

- `reduce_map_filter(("map", lambda x: 2*x, [1,2,3]))` returns the list [2, 4, 6]
- ```
  reduce_map_filter(
        ("map", lambda x: 2*x,
              ("filter", lambda x: x%2 != 0,
                    [1,2,3])))
  ```
  returns the list `[2, 6]`
- ```
  reduce_map_filter(
        ("reduce", lambda a,b: a+b,
              ("map", lambda x: 2*x,
                    ("filter", lambda x: x%2 != 0,
                          [1,2,3]))))
  ```
  returns the integer `8`

The end.

FPRO, 2018/19