Assignment #8: Lists

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

***Goals**: To write functions using lists*

***Pre-requirements (prior knowledge)**: See bibliography of Lecture #12 and Lecture #13*

***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 (26/11/2018)*

***Submission**: to submit, first pack your files in a folder RE08, 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**)*

***Warning**: The use of `numpy` (a package for scientific computing with Python) is not allowed in the submission tool FPROtest.*

1. Inner dot product

Write a Python function `inner(u, v)` which implements the inner-dot product operation between the two given vectors `u` and `v` with the same length.

Save the program in the file `inner.py`

For example:

- `inner([], [])` returns the integer: `0`
- `inner([1, 2], [2, 4])` returns the integer: `10`
- `inner([1, 2, 3, 4, 5], [2, 3, 4, 5, 6])` returns the integer: `70`

2. Matrix multiplication

Write a Python function `mult(M, N)` that computes the matrix multiplication between matrices `M` and `N`, in that order. Return `[]` if the dimensions disagree.

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

For example:

- `mult([[1, 2], [3, 4]], [[2, 0], [1, 2]])` return the list: `[[4, 4], [10, 8]]`
- `mult([[1, 2, 3], [4, 5, 6]], [[9], [8], [7]])` return the list: `[[46], [118]]`
- `mult([[7, 8, 9, 10]], [[5], [3], [2], [7]])`return the list: `[[147]]`

3. List manipulation

Consider the following set of commands:

1. **`insert i e`**: Insert integer **`e`** at position **`i`**.
2. **`print`**: Save list to result string.
3. **`remove e`**: Delete the first occurrence of integer **`e`**.
4. **`append e`**: Insert integer **`e`** at the end of the list.
5. **`sort`**: Sort the list in ascending order.
6. **`pop`**: Remove the last element from the list.
7. **`reverse`**: Reverse the list.

Write a Python function **`manipulator(l, cmds)`** that, given a list of commands from the previous list as a list of strings, applies them sequentially to a list **`l`** and returns a string which contains the result of all print commands (separated by a single space). You can assume all command arguments are valid (e.g. insertion index is valid, remove specifies an element that exists in the list at that time).

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

For example:

- `manipulator([], ["insert 0 5", "insert 1 10", "insert 0 6", "print", "remove 6", "append 9", "append 1", "sort", "print", "pop", "reverse", "print"])` returns the string: `[6, 5, 10] [1, 5, 9, 10] [9, 5, 1]`
- `manipulator([2, 4], ["print", "remove 4", "append 1", "sort", "print", "pop", "reverse", "print"])` returns the string: `[2, 4] [1, 2] [1]`
- `manipulator([], ["print"])` returns the string: `[]`

4. Find local minima

Given a list of integers **`alist`**, write a Python function **`local_minima(alist, n)`** that finds all the local minima on it, within a specified odd neighborhood **`n`**. The function should return a list with all (local minimum, index) pairs found in **`alist`**.

An element **`alist[x]`** is a local minimum if it is less than or equal to its neighbors. The number of neighbors of an element **`alist[x]`** depends on the neighborhood **`n`**. For example, if the neighborhood is 3, the element **`alist[x]`** has two neighbors: **`alist[x-1]`** and **`alist[x+1]`**. If **`n=5`**, **`alist[x]`** has four neighbors: **`alist[x-2]`**, **`alist[x-1]`**, **`alist[x+1]`** and **`alist[x+2]`**.

Note that for corner elements, you have to consider only the neighbours that lie within the list range (in one side only). If there are adjacent local minima (i.e., with the same value within the neighborhood range), you have to return just the local minima with the lowest index.

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

For example:

- `local_minima([10, 3, 3, 14, 5, 7, 4], 3)` returns the list: `[(3, 1), (5, 4), (4, 6)]`
- `local_minima([0, 3, 3, 14, 5, 7, 4], 3)` returns the list: `[(0, 0), (3, 2), (5, 4), (4, 6)]`
- `local_minima([2, 1, 1, 1, 7, 3, 1], 5)` returns the list: `[(1, 1), (1, 6)]`

5. Anagrams

Given a list of strings, write a Python function **`anagrams(alist)`** that groups anagrams together and returns them in a nested list. An *anagram* is a word or phrase formed by rearranging the letters of another word or phrase by using all the original letters exactly once.

Note that you can add and/or remove white spaces between the letters to form the anagrams (they are not relevant when evaluating if two words/phrases are anagrams of each other) and case is irrelevant.

The output list as well as its nested lists should be sorted alphabetically.

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

For example:

- `anagrams(["they see", "eat", "The eyes", "car has", "ate", "a crash", "tea"])` returns the nested list: `[['a crash', 'car has'], ['ate', 'eat', 'tea'], ['The eyes', 'they see']]`
- `anagrams(["sentence", "lives", "ten scene", "bat", "Elvis", "CE sennet"])` returns the nested list: `[['bat'], ['CE sennet', 'sentence', 'ten scene'], ['Elvis', 'lives']]`

The end.

FPRO, 2018/19