Carleton University
Department of Systems and Computer Engineering
ECOR 1051 - Fundamentals of Engineering I

Lab 8 - Learning about Python Lists

Objective

- To learn some of the operators supported by Python's `list` type, and some of the built-in functions that operate on lists.
- To develop functions that have lists as arguments and/or create and return lists.

Learning outcomes: 4, 5, 7; Graduate attributes: 1.3, 5.3 (see the course outline)

Overview

This lab consists of twelve short exercises. Because of the large number of exercises involved, you are invited to do manual testing; automated testing is not required for this lab. (See Lab 6 for definitions of manual and automated testing).

Your solutions can use:

- the `[]` operator to get and set list elements (e.g., `lst[i]` and `lst[i] = a`)
- the `in` and `not in` operators (e.g., `a in lst`)
- the list concatenation operator (e.g., `lst1 + lst2`)
- the list replication operator (e.g., `lst * n` or `n * lst`)
- Python's built-in `len`, `min` and `max` functions

Your solutions **cannot** use:

- `for` or `while` loops
- list slicing (e.g., `lst[i : j]` or (`lst[i : j] = t`)
- the `del` statement (e.g., `del lst[0]`)
- Python's built-in `reversed` and `sorted` functions.
- any of the Python methods that provide list operations; e.g., `append`, `clear`, `copy`, `count`, `extend`, `index`, `insert`, `pop`, `remove`, `reverse` and `sort`
- list comprehensions

Debugging hint: Refer back to Lab 3 for instructions on using the debugger in Wing.

Getting Started

Begin by creating a new file within Wing 101. **Save it as lab8.py**

Exercise 1

Use the function design recipe to develop a function named `first_last6`. The function takes a list of integers. (Assume that the list will not be empty.) The function returns `True` if a 6 is the first element or the last element or if both the first and last element are 6. Otherwise, the function returns `False`.

Common Question: What items should my list contain? Are we supposed to put in random values?

Answer: Yes, you yourself must come up with the content of the lists to be tested. The values should not be completely randomly, but we are not imposing any conditions on their values (outside of those in the exercise itself). For example, you are welcome to use 10 or 1002 or -10, but for this exercise, it would be logical to include 6 as one of the elements of a list.

Common Question: How many docstring test examples are needed.

Answer: The number of tests, and the list contents for each test will depend on what the function does. For this exercise, the following examples are logical:

1. - a list in which the first element is 6
2. - a list in which the last element is 6
3. - a list in which both the first and last element are 6
4. - a list that has no 6's
5. - a list that has one or more 6's, but they aren't the first or last elements

The docstring test examples aren't an exhaustive set of test cases, but each one should demonstrate a different aspect of the function's behaviour. For example, the three different reasons for which the function will return True can be illustrated by 3 different docstring examples - see the first three points, above. The remaining examples illustrate cases in which the function will return False.

Exercise 2

Use the function design recipe to develop a function named `same_first_last`. The function takes a list of integers. (The list may be empty.) The function returns `True` if the list is not empty and if the first and last elements are equal. Otherwise, the function returns `False`.

Exercise 3

Use the function design recipe to develop a function named `make_pi`. The function has no arguments. It returns a list of length 3containing the first three digits of π.

Exercise 4

Use the function design recipe to develop a function named `common_end`. The function takes two lists of integers that are not empty, but which may have different lengths. The function returns `True` if they have the same first element or the same last element or if the first and last elements of both lists are the same. Otherwise, the function returns `False`.

Exercise 5

Use the function design recipe to develop a function named `sum3`. The function takes a list containing three integers. The function returns the sum of all the elements.

Exercise 6

Use the function design recipe to develop a function named `rotate_left3`. The function takes a list containing three integers. The function returns a new list containing the same elements, but they are "rotated left". For example, when the argument is `[1, 2, 3]`, the function returns `[2, 3, 1]`.

Exercise 7

Use the function design recipe to develop a function named `reverse3`. The function takes a list containing three integers. The function returns a new list containing the same elements in reverse order. For example, when the argument is `[1, 2, 3]`, the function returns `[3, 2, 1]`.

Exercise 8

Use the function design recipe to develop a function named `max_end3`. The function takes a list containing three integers. The function determines which element is larger: the first element or the last element. It returns a new list in which all the elements are initialized to that value. For example, when the argument is `[2, 9, 3]`, the function returns `[3, 3, 3]`.

Exercise 9

Use the function design recipe to develop a function named `sum2`. The function takes a list of integers. The function returns the sum of the first two list elements. It should return 0 if the list is empty, or the element if the list has one element.

Exercise 10

Use the function design recipe to develop a function named `middle_way`. The function takes two lists that each contain three integers. The function returns a new list containing their middle elements. For example, when the arguments are lists `[1, 2, 3]` and `[4, 5, 6]`, the function returns `[2, 5]`.

Exercise 11

Use the function design recipe to develop a function named `make_ends`. The function takes a list of integers. (Assume that the list will not be empty.) The function returns a new list containing the first and last elements from the original list. For example, when the argument is `[4, 5, 6, 7]`, the function returns `[4, 7]`.

Exercise 12

Use the function design recipe to develop a function named `has23`. The function takes a list containing two integers. The function returns `True` if the list contains a 2 or a 3 or both values. Otherwise, the function returns `False`.

Wrap Up

Ensure that your code meets the posted marking rubrics for the labs.

Submit file lab8.py.

You are required to keep a backup copy of (all) your work for the duration of the term.

Last edited: April 23, 2020