CSI2100-01 Spring 2020 Lab 13

Contents

1 General Information

1.1 Coding Style Guide

We have introduced coding style guidelines with Lab 12. For convenience, those guidelines are repeated here.

The aim of a coding style is to enhance the readability and clarity of our source code. You find the guide in file `StyleGuide.pdf` provided with the **Lab 12** specification.

- Please read the style guide and have a look at the provided examples. The style guide comes with a link to our web site where you will find additional examples.
- For every lab problem, Hyeongjae Python will run a style checker to test whether your code conforms to the style guide. You are kindly asked to fix style issues in your code befor you submit on YSCEC.

1.2 Rules for Comments and Docstrings

We employ the following rules for comments and docstrings:

1. Every function and method (including code provided to you) must contain a docstring that describes its purpose.

   In case you wrap your main program into a function (which is not required), this function needs a docstring, too.

2. Your main program and every function and method (including code provided to you) must be commented. As previously, you should comment on the important/complicated parts in your code.

   In case you wrap your main program into a function (which is not required), this function needs to be commented, too.

   As an exception, you are not required to provide comments with functions and methods that contain less than 3 non-comment LOC.

   Example:

   ```
   1  def shortFunction():
   2      """
   3      This function is just a placeholder.
   4      """
   5      pass
   ```

   The above function consist of only 2 non-comment LOC (line 1 and line 5) and therefore does not require comments.

3. Every class (including code provided to you) must include a docstring that describes the purpose of the class.

4. Every file that you submit must contain a docstring that contains your name, student ID and problem name.

   Example:

   ```
   """
   Name: Hwango Lee
   Student ID: 202067584
   Lab problem: lab12_p1.py
   """

   ...
   ```

1.3 Administration and Notation

- **Because of our coronavirus precautions, you are provided with an excerpt of the relevant sections of the textbook together with this lab assignment.**

- In some places and to facilitate reading, we use "␣" to depict a space (blank) character, and "¤" for a "\n" (newline) character.

Programming Problems

Problem 1:

Excercise 10 **combined with** Exercise 11 from page 454 of the textbook.

1. Unlike stated with Exercise 10, you are asked to write the code for the entire `Range` class, not just the `__str__` special method.
2. In the textbook, the author proposes to use two *private* data attributes with the `Range` class, namely `__start` and `__end`. Note that you do not have to provide getter and setter methods for these data attributes.
3. Please add the special method `__lt__` to your `Range` class, as described with Exercise 11.
4. You should submit only the code of your `Range` class. Do not submit any other code with your class (no test code, etc.)

Below you find an example how we might test your `Range` class:

```
from lab13_p1 import Range

r1 = Range(10, 16)
r2 = Range(-10, 0)
print(r1)  # Output: 10...16
print(r2)  # Output: -10...0
print(r1 < r2)  # Output: False
print(r2 < r1)  # Output: True
```

Note in the above example that the `__str__` special method does not include the quote characters (`''`) in the output. This is different to the example in the textbook.

You can assume that the constructor of your range class will always be provided with two integer values. However, you must check in the constructor that the first argument (the lower bound of the range) is less-equal ($\leq$) than the second argument (the upper bound of the range). If this condition is violated your constructor must raise the exception `IndexError`. For example:

```
r3 = Range(10, 9)  # Must raise IndexError exception!
```

Problem 2:

Exercise P3 from page 455 of the textbook. Please use the following stub for your `computeAvg` function.

```
def computeAvg(self):
    """"
    Computes the average of a list of numeric types.
    Raises the ValueError exception if a list element is neither
    an instance of an 'int' nor a 'float' class.
    """
```

This problem is very simple, if you embrace the inheritance mechanism from object-oriented programming. Because you must make your `AvgList` class a **subclass** of Python's `list` class, it will inherit everything from the `list` class, including the `__init__` method.

The only method you need to implement is `computeAvg` itself. Think about the `self` parameter, what it represents, and what methods you can call on it to accomplish this task.

In your `computeAvg` method, how can you ensure that a list element is of type **int** or **float**? The `isinstance(object, classinfo)` built-in function allows you to check whether `object` is an instance of `classinfo`, for example:

```
>>> isinstance(22, int)
True
>>> isinstance(22, float)
False
>>> isinstance('', float)
False
```

Please refer to Python's online documentation here for additional information about the `isinstance` function.

Below you find some test code for your class.

```
f = AvgList()  # List constructor to create empty list.
f.append(22)
f.append(2.2)
# Not a numeric type, will raise ValueError exception in computeAvg():
# f.append('2.2')
print(f.computeAvg())
```

Problem 3:

Please download the archive doboggi_man.zip from YSCEC. It contains a simplified version of the pacman game. In our simplified version, doboggi_man must find and eat three servings of doboggi. Please drag-and-drop the entire "`doboggi_man`" folder into your PyCharm project root-folder. Please run file game.py within PyCharm and observe how this game works. (You can control doboggi_man with the cursor-keys on your computer's keyboard.)

In a nutshell:

- Doboggi_man must avoid the two red ghosts and eat all food.
- Doboggi_man has only a limited amount of steps to complete the mission.
- The game ends when all food has been eaten, or when doboggi_man is hit by a ghost, or when doboggi_man runs out of steps.
- When the game terminates, a diagnostic message is printed in the PyCharm shell.

The game consists of the following modules:

1. **characters.py**: contains a class hierarchy of game characters (ghost, doboggi_man, doboggi).
2. **game.py**: the game-logic of the program. We use the already studied Python turtle graphics. The game is driven by a Python timer event which repeatedly calls function `periodicTimer()`. Inside this function, the characters are moved around and collisions are checked.
3. **globals.py**: contains global definitions that are used in all parts of the program.
4. **lab13_p2.py**: contains class `auto_doboggi_man` for you to complete (see the below instructions).

The most important aspect for you to understand is that each game character has a Python turtle data attribute (named ttl). To move around, the ghost and doboggi_man classes both provide a `move()` method. Inside this method, the character's turtle is moved (how a turtle is moved we have already studied in previous lectures). For each character, the `move()` method is called repeatedly from inside of the `periodicTimer()` function.

Your task:

- In module `game.py`, comment-out line 196 and un-comment line 197. That way you're switching to the `auto_doboggi_man` class which you're going to implement.
- In module `lab13_p2.py`, implement method `move()` of the `auto_doboggi_man` class. Every time the `move()` method is called by the timer function, your program must advance doboggi_man one step.
- Note: you're not allowed to move doboggi_man for more than 10 pixels in the x- and y-directions within a single step (this is enforced in the timer function). If you move doboggi_man for a too far distance in a single step, the program terminates.
- Doboggi_man is not allowed to leave the screen. If your doboggi_man walks beyond the GUI window, the program will terminate (checked with automated grading only).
- You only have 300 steps (module globals.py) to accomplish your mission.
- Please read the provided classes to understand how you can find out about the positions of the game-characters on the screen (your doboggi_man must avoid ghosts and find all doboggi).
- The mission is accomplished if doboggi_man emptied all three doboggi plates. (Otherwise, your program will receive 0 points.)
- **Hint:** you are recommended to divide doboggi_man's mission in 3 steps:
  1. Collect the doboggi in the lower half of the screen (below the 2 ghosts).
  2. Walk doboggi_man to the upper half of the screen (past the 2 ghosts).
  3. Collect the two doboggi plates in the upper half of the screen.

  Your `move()` method must know in which step of your mission you are. This can be accomplished by adding a data attribute to your `auto_doboggi_man` class. (Similar to the `phase` variable that controls the appearance of the ghosts and of doboggi_man in the `updateShape()` methods.)

Any additional information your doboggi_man needs to keep track of you should add as data attributes, too. You are free to add any data attribute and method to the `auto_doboggi_man` class.

Please note:
This program is considerably larger than previous programs we worked on. It is not necessary for you to understand each and every line of code in detail (although you're recommended to). The program contains only Python features we have already discussed during the lectures. It is sufficient to focus on the `move()` method to navigate your doboggi_man.
Reading and understanding code is a good way to improve our programming skills.

Problem 4:

Implement a recursive function to compute the n th Fibonacci number. The Fibonacci numbers are numbers in the following integer sequence:

$$0, 1, 1, 2, 3, 5, 8, 13, 21, \ldots$$

This sequence is recursively defined as:

$$F_0 = 0, F_1 = 1 \tag{1}$$
$$F_n = F_{n-1} + F_{n-2} \tag{2}$$

See `http://en.wikipedia.org/wiki/Fibonacci_number` for background information on the Fibonacci numbers. Use the following stub for your implementation:

```
def fib(n):
    """Computes the n-th Fibonacci number."""
```

You should submit only the code of your `fib()` function. Do not submit any other code with your function (test code, etc.)

Your `fib()` function is **not allowed** to print anything. Instead, it must return the n th Fibonacci number.

Failing to comply to the above rules will result in 0 points. Please have a look below how we're going to test your Fibonacci function:

```
import lab13_p4

print(lab13_p4.fib(8))  # Should output 21
# Further testcases follow...
```

Problem 5:

If you call function `fib` with higher values of n (for example, 20, 25, 30, 35, 40...), you will find that our recursive function takes longer and longer to execute.

Why is it getting so slow? To answer this question, let us imagine how `fib(40)` is going to be computed. According to our definition of the Fibonacci sequence in Problem 4, to compute `fib(40)` we must first compute `fib(39)` (a lot of work), and add the result to `fib(38)`. But wait: to compute `fib(39)`, we already computed `fib(38)` as a recursive call. Computing it a second time is redundant. This repeated calculation is happening all over the place, which is unnecessarily slowing down the computation.

Let us quickly approximate the number of additions required to compute each Fibonacci number recursively, and compare it to the Fibonacci sequence:

n	number of additions	nth Fibonacci number
0	0	0
1	0	1
2	1	1
3	2	2
4	4	3
5	7	5
6	12	8
7	20	13
8	33	21
9	54	34

We see that the number of additions are larger than the Fibonacci numbers. The function `fib(n)` can be computed in *closed form* by

$$F_n = \left\lfloor \frac{1}{\sqrt{5}} \left(\frac{1+\sqrt{5}}{2} \right)^n \right\rceil, \tag{3}$$

where $\lfloor x \rceil$ denotes the integer closest to x. Because the above equation has an exponential term in n, and because the number of additions are larger than F_n, it follows that the number of additions grow (at least) exponentially in n. This is the reason why our computers run out of steam quickly for larger values of n.

The *memoization* technique avoids redundant computations by memorizing all previously computed values. We apply memoization to the Fibonacci sequence, by programming a function `fib_memo(n)`:

1. Set up an empty dictionary as a global variable in your main program.
2. Inside your `fib_memo(n)` function, check whether the Fibonacci number for n is already included in the dictionary.
   (a) If included, return it.
   (b) If the Fibonacci number for n is not yet included in the dictionary, compute it through recursive calls to `fib_memo`, enter it into the dictionary, and return the computed value.

Please use the following stub for your implementation:

```
def fib_memo(n):
    """Computes the n-th Fibonacci number using memoization."""
```

You should submit only the code of your `fib_memo()` function and your global variable referring to the dictionary. Do not submit any other code with your function (main program, test code, etc.)

Problem 6:

Same as Problem 4, except that your function should compute the total number of function calls required to compute the nth Fibonacci number. Please use the following stub for your implementation:

```
def fibcalls(n):
    """
    Computes the number of function calls required
    to compute the n-th Fibonacci number.
    """
```

Examples:

`fibcalls(0)` = 1 (base case)
`fibcalls(1)` = 1 (base case)
`fibcalls(2)` = 1 + `fibcalls(1)` + `fibcalls(0)` = 1 + 1 + 1 = 3
`fibcalls(3)` = 1 + `fibcalls(2)` + `fibcalls(1)` = 1 + 3 + 1 = 5
`fibcalls(4)` = 1 + `fibcalls(3)` + `fibcalls(2)` = 1 + 5 + 3 = 9
and so on ...

You are not allowed to solve this problem with a global variable. Change the solution from the previous problem so that each call to the `fibcalls()` function **returns** the number of function calls that where required to compute the result.

We will test this function similarly as described with the previous problem.

Problem 7:

Modify the recursive directory traversal function from the lecture slides such that all files of extension ".txt" are searched for the occurrence of a given string s. A list of those files (including the path) should be returned by your function.

Use the following stub for your implementation:

```
def searchDir(directory, s):
    """
    Recursively searches 'directory' for .txt files
    that contain string s.
    """
```

For example, if your function is called as

```
searchDir('C:/Users/you', 'Yonsei')
```

then it should return a list of the files in folder `C:/Users/you` (and its subfolders) which contain the word "`Yonsei`". If there are three files `foo.txt`, `subfolder/bar.txt` and `car.txt` that contain the searched word, your list should be:

```
['C:/Users/you/foo.txt',
 'C:/Users/you/subfolder/bar.txt',
 'C:/Users/you/car.txt']
```

Note:

1. With this problem, the string s may occur as a substring, too. In the above example, if a file contains the line "`myYonseiUniversity\n`", the file should be included in the output.
2. The order of files in the returned list is up to you.
3. Opening or reading a file might throw an `OSError` exception. Please catch this exception. Please exclude the file that caused the exception from any further processing, do not even close it.
4. You are not allowed to use global variables with this example!
5. You are allowed to use the `os` module with this example.
6. You should submit only the code of your function and any import declarations that it requires. Do not submit any other code (test code, etc.)

2 Marking Criteria and Plagiarism

Marking Criteria

- Programming style: code must adhere to our style guide (Section 1.1) and our documentation guidelines (Section 1.2), otherwise points will be deducted.
- Score is only given to programs that compile and produce the correct output with Python version 3.
- Points are deducted for programs that are named wrongly. See the list of deliverables for the required file names.
- Points are deducted for programs that produce warnings.
- Please pay particular attention to the requested output format of your programs. Deviating from the requested output format results in points deductions.
- Use of modules:
  - You are allowed to use the `sys` module with all programing problems.
  - Please always use an exit code of 0 with the `exit()` function of the `sys` module: `sys.exit(0)`.
  - You are allowed/asked to use modules stated with individual programming problems and contained in provided code (if any).
  - No other modules are allowed.

Plagiarism (Cheating)

- This is an individual assignment. All submissions are checked for plagiarism.
- Once detected, measures will be taken for **all** students involved in the plagiarism incident (including the "source" of the plagiarized code).

Deliverables and Due Date

In the below table, column "Style Check" refers to the coding style guidlines from Section 1.1.

Column "Correctness" refers to the correctness check that was already provided with previous labs.

Please note that no check for the documentation guidelines from Section 1.2 is provided. Please make sure to document your code as requested.

		Hyeongjae Python	
Problem	File name	Style Check	Correctness
1	lab13_p1.py	✓	✓
2	lab13_p2.py	✓	✓
3	lab13_p3.py	✓	—
4	lab13_p4.py	✓	✓
5	lab13_p5.py	✓	—
6	lab13_p6.py	✓	—
7	lab13_p7.py	✓	—

- Please submit all files in a ZIP-file named lab13_<student_id>.zip
- Please submit your archive on YSCEC by **Monday, June 29**, 2020, **23:00**. (Because this is the final weekly lab, additional time is provided to you. The due date is—as usual—strict.)