**Assignment 2 – Objects**

**Dean Zeller 20 points**

### CS120 Due: Friday, August 31st

**Fall 2018**

***Objective***Students will refactor the requirements of Assignment 1 to include basic object oriented programming, including attributes, methods, and a separate-file tester.

***Background***: ***Objects First***

Object oriented programming has been a staple of the programming industry since the mid 1990s. The first OOP languages were Smalltalk, \_\_, and\_\_\_, but OOP was not a true standard until Bjarne Stroustrup created the C++ language based on C, by Dennis Ritchie. Because of OOP’s late arrival on the scene, it has always been though of as more complex, and too difficult for introductory students.

However, in 20\_\_, an educational movement formed, called *Object First!* The movement indicates that maybe objects should be covered very early in a programming curriculum. Before conditionals, loops, and functions. Objects help with long-term program organization, and the concepts are not difficult. This assignment is about introducing students to the very popular programming topic of Object Oriented Programming.

***Terms:***

Object – A collection of variables and functions associated together for a common purpose. An instantiated class.

Attribute – A variable belonging to an object.

Method – A set of programming commands that may be called, either by the object or an external program.

Importing – Coding access to one program to access another.

Tester – A program file specifically meant to test an object.

Instantiate – Create an executable instance of an object.

--init-- - A special method in Python that executes when an object is instantiated.

***Instructions***

1. Create a folder names assignment 2. Create the Python files *AnalysisEngine.py*  and another named *AnalysisEnginerTester.py*.
2. In *AnalysisEngine.py* enter the following code *exactly*. It will create a very basic object.

class AnalysisEngine:

def \_\_ init \_\_ (self):

print(“Initializing Attributes”)

data = 0

def enterData (self):

self.data = input(“Enter data =>”)

def printData (self):

print(“Data: “, self.data)

1. In *AnalysisEngineTester.py*, enter the following code, exactly. This will instantiate a new Analysis Engine object and test its methods.

study1 = AnalysisEngine()

study1.enterData()

study1.printData()

1. Make the appropriate changes to the *AnalysisEngine.py* and *AnalysisEngineTester.py* to accept three integers instead of one.
   1. In the *\_\_ init ­­\_\_* method, declare the attributes a, b, c, sum, and average.
   2. Modify the *enterData* method to accept three integers as input from the user, setting the values as attributes.
2. Write the method to calculate the sum of the three attributes, a, b, and c. And set the *self.sum* equal to a+b+c.
   1. Create the attribute *sum* in the *\_\_ init \_\_* method with the code *sum = 0.*
   2. Within the object, create a new method named *calcSum*, with no parameters. Within *calcSum*, calculate the sum of the three numbers, saving the result in *self.sum*. Also print a message, indicating the sum was calculated.
   3. Within the tester, call the *calcSum* method for *study1.* It should only print the message “Calculating sum”.
3. Write the method to calculate the average of the three attributes, a, b, and c.
   1. Create the attribute *average* in the *\_\_ init \_\_* method, with the code average = 0.0.
   2. Within the object, create a new method named *calcAverage* with no parameters. Within *calcAverage*, calculate the average of the three numbers saving the result in *self.average*. Also print a message, indicating the average was calculated.
   3. Within the tester, call the *calcAverage* method for *study1.* It should only print the message “Calculated Average”.
4. Modify *printData*, changing its name to *printReport.* It should print the same report done in assignment 1.
5. Make all appropriate changes to recreate the report from Assignment 1. All printing and data entry should be done within the object. If written correctly. Your tester should only be:

study1 = AnalysisEngine()

study1.enterData()

study1.calcSum()

study1.calcAverage()

study1.printData()

1. Add file documentation to both files, to ensure they are identified as yours.
2. Add method documentation for each method. Use the following template, replacing information as necessary. Parameters and return values will be added in future assignments.

###################################################################### method-name: #

# #

# parameters: none #

# return value: none #

#####################################################################

***File Documentation***

Include this section of documentation at the top of your Python program. It is documentation that contains information about the file, but no programming itself. Make changes to the appropriate fields, denoted with dashes-in-the-name; this will reflect that this is your code.

################################################################################

# title-of-assignment #

# #

# PROGRAMMER: your-name #

# CLASS: CS102 #

# ASSIGNMENT: Assignment assig-# #

# INSTRUCTOR: Dean Zeller #

# TA: Robert Carver #

# SUBMISSION DATE: date-of-submission #

# #

# DESCRIPTION: #

# Describe-the-program-implemented-in-your-own-words #

# #

# COPYRIGHT: #

# This program is (c) 2018 your-name and Dean Zeller. This is original work, #

# without use of outside sources. #

################################################################################

***Grading***

You will be graded on the following criteria:

***Evaluation Instructions (Summary)***

On Monday after the due date, the solution and all student submissions will be made available for download. Find three students in class to evaluate your work, and you are to evaluate their work. You will be given the rubrics similar to the following page to evaluate their work.

***Assignment 1 Grading Rubric (probably need to be changed) Programmer Name: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_***

***CG120 -------------SELF EVALUATION-------------***

Program Testing

\_\_\_\_\_\_ Execution

Does the program execute correctly?

-20 for inexecutable code

\_\_\_\_\_\_ Output

Does the video correctly demonstrate the execution of the program?

-20 for altered or fake output

\_\_\_\_\_\_ Test input/output

Does the program work correctly for the given test input from the assignment write-up?

-2 to -10 for errors in output

\_\_\_\_\_\_ Other input/output

Test the code for other combinations of data, does it work correctly?

-2 to -10 for insufficient testing

\_\_\_\_\_\_ Is the output formatted exactly like the test run?

-2 to -5 for incorrect formatting

-2 to -10 for elements missing

Submission Requirements

\_\_\_\_\_\_ Program

Does the submission have all required files to execute?

-10 for missing any necessary program file(s)

\_\_\_\_\_\_ Video

Is there a video demonstration included in the submission?

-10 for missing video

Does the video include the five requirements (intro, code, given test, other test, thanks)

-2 to -9 for insufficient video or missing elements

\_\_\_\_\_\_ Program Style

Does the program follow the specified style guidelines including indentation, variable names, and blank spaces?

-2 to -5 for insufficient attention to program style

Documentation Requirements

\_\_\_\_\_\_ File documentation

Is there correct documentation at the top of the file following the documentation guidelines in the writeup.

-10 for no documentation

-2 to -9 for insufficient or incorrect documentation

\_\_\_\_\_\_ Spelling/grammar

Is the documentation properly written and free of spelling and grammar errors?

-1 to -5 for spelling and grammar errors