System Requirements Specification Index

For

Object Oriented Programming – Python

Version 1.0

IIHT Pvt. Ltd.
fullstack@iiht.com

Problem Statement : Object oriented programming in python

Description : This assessment contains various subparts to test object oriented programming concepts in Python

The solution contains the following folder structure.

Python_OOPS |

|--part_a.py

|--part_b.py

|--part_c.py

|--part_d.py

|--tests – contains unit test cases for the solution

Part_a.py:

In part_a.py, we will look at magic methods of python.

i) return_magic_methods() – return the list of magic methods of a class.

ii) return_no_of_magic_methods() – return no. Of magic methods of a class

iii) return_no_of_common_magic_methods() – return no of common magic methods between two classes.

Part_b.py:

In part_b.py, we will look at abstract classes. Create an abstract class Investment and declare two abstract methods lockin_period() and interest_rate(). Define two classes PF and PPF inheriting from the Investment abstract class. Implement the abstract methods and for simplicity, just return some number from all the methods.

Part_c.py:

In part_c.py, we will look at operator overloading and multiple inheritance. Let's say there is a class called FaultyCalc which does basic mathematical operations (add, sub, mul, div), but as the name suggests it’s faulty, It returns sub for add and add for sub.

Now, lets consider a scenario where you have to use this class and can't modify it. So, you define another class CorrectClac, just with addition and subtraction definitions. And define a class MainClass that inherits both FaultyCalc and CorrectCalc such that all operations returns correct results.

In case of invalid operations return -1.

Part_d.py:

In part_d.py we will make the Calculator program mode SOLID complaint. For starters separate out each operation into a class and make it complaint with Single Responsibility Principle. Now, add new operation mod (% operator) without affecting existing classes complying with Open-closed principle.

Execution Steps to Follow:

1. All actions like build, compile, running application, running test cases will be through Command Terminal.
2. To open the command terminal the test takers, need to go to Application menu (Three horizontal lines at left top) -> Terminal -> New Terminal
3. This editor Auto Saves the code
4. If you want to exit(logout) and continue the coding later anytime (using Save & Exit option on Assessment Landing Page) then you need to use CTRL+Shift+B-command compulsorily on code IDE. This will push or save the updated contents in the internal git/repository. Else the code will not be available in the next login.
5. These are time bound assessments the timer would stop if you logout and while logging in back using the same credentials the timer would resume from the same time it was stopped from the previous logout.
6. To setup environment:
   pip install requests

7. Run the application:
   python3 part_a.py
   python3 part_b.py
   python3 part_c.py
   python3 part_d.py

8. To run Test cases:
   python3 -m unittest

9. Before Final Submission also, you need to use CTRL+Shift+B-command compulsorily on code IDE. This will push or save the updated contents in the internal git/repository for code quality analysis graph.

-----x-----