

**Python Object Oriented Programming**

Assignment 3

# Python Object Oriented Programming

### Software Development

|  |  |
| --- | --- |
| image001.png | Python Object Oriented Programming |

### Pledge of Honour

You are required to include the following as documentation at the beginning of every assessment:

Student name: first\_name Last\_name

Student ID: 123456789

Pledge of Honour: I pledge by honour that this program is solely my own work.

### Plagiarism

Any course work presented for assessment must be your own work. Copying or paraphrasing someone else’s work be it published, unpublished or off the internet, without clearly acknowledging it constitutes plagiarism and is considered to be academic misconduct. You are required to sign an assignment declaration stating it is your own work. You may receive a zero for part or all of the assessment submitted in first instance. Repeated incidents of plagiarism or cheating could result in you being removed from the course or the programme.

### Marking Criteria of Assignment 3

This is how the questions are marked in Assignment 3:

1. To achieve the full marks on each question, your program must be complete and work correctly. This means:
   1. The program must do what the question says.
   2. The program must make the same output as the Example Output.
2. Your program solutions need to have these documentation comments at the beginning:
   1. Student name and ID
   2. Pledge of Honour declaration
3. Your program should use the Programming Best Practices at the end of this document.

### Assignment 3

**Learning outcomes:** The course work questions cover Learning Outcomes 1 – 3

**Aims:** The course work questions aim to practice these topics:

* Basic input and output operations
* Coding standards
* Debugging and testing by using features of Integrated Development Environment (IDE)
* The use of methods (both static and instance) and parameters in solving problems
* Design and implement classes and objects
* public and private class fields
* constructors, public, and private methods
* Modular programming
* Problem solving using classes and objects
* Derived classes and protected variables and methods: Examples and problem solving
* Polymorphism and its advantages in software development: Examples and problem solving
* Using object oriented programming to develop applications
* Object oriented programming and code reuse
* List of objects
* Object persistence with file I/O

**Weighting:** 15%

**Mark allocation on questions:**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Question No. | 1 | 2 | 3 | 4 | 5 | 6 | Total |
| Marks | 2.5 | 2.5 | 2.5 | 2.5 | 2.5 | 2.5 | 15 |

**Due date:** see timetable on Moodle

**Marking process:**

Upload your solutions (as a zip file) to Moodle.

Question 1

|  |
| --- |
| Polymorphic Lists using Inheritance |
| * Learning outcomes: 1&3 * Relevant topics: input, output, iteration, list, class, objects, encapsulation, inheritance * Suggested time to complete: Week 10 * Workbook topic: *Encapsulation* |

* Make a class called **Worker**. A worker has a **name** and an **age**.
* Make the **\_\_init\_\_** method
* Make an object variable called **name** and an object variable called **age**
* Make a method called **work** that prints “I work”
* Make a class called **Teacher**. It should inherit worker
* Make the **\_\_init\_\_** method, call the **\_\_init\_\_** method from the Worker class and make an object variable called **number\_of\_students**
* Make a method called **work()** that prints “I teach … students” using **number\_of\_students**
* Make a class called **Programmer**. It should inherit worker
* Make the **\_\_init\_\_** method, call the **\_\_init\_\_** method from the Worker class and make an object variable called **language**
* Make a method called **work()** that prints “I write … programs” using **language**
* Below is the code that makes the **Worker** objects. It uses a **list** to make the objects. This list is passed to a method called **print\_workers()** that you need to write the code for:

worker\_list = [Worker(**"Sam"**, 23), Teacher(**"Ted"**, 33, 27),  
 Programmer(**"Pete"**, 25, **"Python"**)]

**def** print\_workers(workers):

<You need to write the code here>

<Print the worker name>

<Print the worker age>

<Call the work() method>

print\_workers(worker\_list)

It should make this output:

Sam

23

I work

Ted

33

I teach 27 students

Pete

25

I write Python programs

Question 2

|  |
| --- |
| Polymorphic Lists using an Abstract Class |
| * Learning outcomes: 1&3 * Relevant topics: input, output, iteration, list, class, objects, inheritance, abstract * Suggested time to complete: Week 12 * Workbook topic: *Lists of Objects* |

* Make an **abstract** class called Vehicle
* Make the \_\_init\_\_ method with one object variable called name
* Make an **abstract** method called go(). Use pass.
* Make an **abstract** method called speed(). Use pass.
* Make a class called Car. It **inherits** the Vehicle class.
* Make the \_\_init\_\_ method with 1 object variable called land\_speed
* You will need to call the \_\_init\_\_ method of the Vehicle class
* Make a method called go() that prints “Drive on land at … km/h” using land\_speed
* Make a method called speed() that **returns** land\_speed
* Make a class called Boat. It **inherits** the Vehicle class.
* Make the \_\_init\_\_ method with 1 object variable called water\_speed
* You will need to call the \_\_init\_\_ method of the Vehicle class
* Make a method called go() that prints “Drive on water at … km/h” using water\_speed
* Make a method called speed() that **returns** water\_speed
* Make a list of objects using the data below:

|  |  |  |  |
| --- | --- | --- | --- |
| Class | Name | Land speed | Water speed |
| Car | Ferrari | 220 |  |
| Car | Dodge | 198 |  |
| Boat | Speedboat |  | 80 |
| Boat | Rowboat |  | 4 |

* Use a for loop for the list of objects:
  + Print the name
  + Call the go() method
  + Calculate the total speed
  + Find the maximum speed
* Print the average speed and the maximum speed

It should make this output:

Ferrari

Drive on land at 220 km/h

Dodge

Drive on land at 198 km/h

Speedboat

Drive on water 80 km/h

Rowboat

Drive on water 4 km/h

Average speed: 125.5 km/h

Maximum speed: 220 km/h

Question 3

|  |
| --- |
| Recursion of numbers |
| * Learning outcomes: 1&3 * Relevant topics: input, output, recursion, methods * Suggested time to complete: Week 11 * Workbook topic: Recursion |

Write a **recursive** function that calculates the product of all digits in an integer. For example, if the number is 235, then the function should calculate 2 \* 3 \* 5 = 30

First, get the number from the user. Then give this number to the **recursive** function, which will return the answer. Print out the answer.

Help:

**num % 10** will give you the last digit of **num**

For example,

**25 % 10 = 5**

**523 % 10 = 3**

Help:

**int(num / 10)** will give you all of the number except for the last digit of **num**

For example,

**int(25 / 10) = 2**

**int(523 / 10) = 52**

So you need to use the recursive function to do **int(num / 10)** to remove the last digit and give it to the recursive function.

Example output:

Enter a number: 235

The product of all of the digits of 235 is 30

Enter a number: 36

The product of all of the digits of 36 is 18

Question 4

|  |
| --- |
| Recursion of a list of objects |
| * Learning outcomes: 1&3 * Relevant topics: input, output, recursion, methods, class, objects * Suggested time to complete: Week 11 * Workbook topic: Recursion |

Make a class called Book. It should have an \_\_init\_\_ method with 2 parameters title and pages. Set the object variables title and pages.

Then finish the code that uses a recursive function called add\_pages() to calculate the total number of pages in an list of books.

Here is some code to help you:

<Make class **Book** here>

**def** add\_pages(books):  
 <Make the code for this recursive function. It will calculate and return the total number of pages for all of the objects in **books**>  
  
  
books = []  
books.append(Book(**"Learning Python"**, 673))  
books.append(Book(**"English for Experts"**, 504))  
books.append(Book(**"OOP in Python"**, 146))  
  
total\_pages = add\_pages(books)  
print(**"Total number of pages: {0}"**.format(total\_pages))

Example output:

Total number of pages: 1323

Question 5

|  |
| --- |
| Q5 – File I/O |
| * Learning outcomes: 1&3 * Relevant topics: input, output, list, iteration, File input & output * Suggested time to complete: Week 12 * Workbook topic: File I/O |

* Get a string from the user with the message “Enter a list of numbers separated by commas:” using input()
* Open an empty text file called **some\_numbers.txt**
* **Write** the string into the text file
* Open the text file
* Use line\_of\_numbers = f.readline() to put the text into a variable.
* Use split to put line\_of\_numbers into a list.
* Use a for loop on the list to print each number, and find the sum and maximum number.
* Example output:

Enter a list of numbers separated by commas: *1,2,3,4,5*

1

2

3

4

5

Sum: 15

Maximum: 5

Question 6

|  |
| --- |
| Q6 – Objects and File I/O |
| * Learning outcomes: 1&3 * Relevant topics: input, output, class, objects, methods, File input & output * Suggested time to complete: Week 12 * Workbook topic: File I/O |

Make a text file called “data.txt”

Here is the data:

L0111,80.5

L0222,70.8

L0333,95.4

L0444,67.9

L0555,56.5

Save the text file somewhere. The data has the student ID and exam score for 5 students, separated by commas.

Make a class Student which has:

* An \_\_init\_\_ method with 2 object variables: id and score

Make a class ScoreApp:

* This code has already been written for you:
  + A list of student objects called students
  + The \_\_init\_\_ method which needs the name of the *data* file. It calls the method read\_student\_data()
* **You** need to write the other code:
  + Finish the code in read\_student\_data() You need to open the *data* file, and for each line of data, make a *Student* object with the data in the line, and put the *Student* object into the students list. You will need to use the split() method.
  + Finish the code in print\_all(). This will use a loop to print each id and score of each object.
  + Finish the code in get\_max\_score(). This will use a loop to find the biggest score and return the biggest score.
  + Finish the code in get\_average\_score(). This will use a loop to find the total score and then calculate and return the average score.
  + Finish the code in count\_pass\_score(). This will use a loop to count how many students passed (how many scores are >= 50). It will return the count.

**Outside the classes:**

* You need to write the other code:
  + Finish the code in get\_total\_score(student\_list). This is a RECURSIVE method. It will use recursion to calculate the total score for all students. It will return the total score at the end of the method.

**class** Student:  
  
 <Write the code. You need an \_\_init\_\_ method>  
  
**class** Score\_App:  
  
 students = []  
  
 **def** \_\_init\_\_(self, file\_name):  
 self.read\_student\_data(file\_name)  
  
 **def** read\_student\_data(self, file\_name):  
  
 <Write the code to read the file and put the values into the student objects and then into the students[] list. Use page 168 of the workbook for help>  
  
  
 **def** print\_all(self):  
  
 <Print the details of all students>  
  
 **def** get\_max\_score(self):  
  
 <Calculate and return the biggest score>  
   
 **def** get\_average\_score(self):  
  
 <Calculate and return the average score>

**def** count\_pass\_score(self):

<Calculate and return the number of students that passed>  
  
  
*# Outside all of the classes*  
**def** get\_total\_score(student\_list):  
  
 <Write this method to calculate the total score of all students. Make this method Recursive>  
  
  
app = Score\_App(**"data.txt"**) *# Use the correct location, for example C:\\Users\\XXXX\\Desktop\\data.txt*  
  
print(**"-"**\*30)  
app.print\_all()  
  
print(**"-"**\*30)  
print(**"Max score: "** + str(app.get\_max\_score()))  
print(**"Average score: "** + str(app.get\_average\_score()))  
print(**"Number of pass scores: "** + str(app.count\_pass\_score()))  
print(**"Total score: "** + str(get\_total\_score(app.students)))

Example output:

------------------------------

L0111: 80.5

L0222: 70.8

L0333: 95.4

L0444: 67.9

L0555: 56.5

------------------------------

Max score: 95.4

Average score: 74.22

Number of pass scores: 5

Total score: 371.1

### Help

Programming Best Practices

Code readability is one of the first things we learn as programmers. A program is only written once but will be looked at many times by you or other people later. It is important to make your code readable and understandable. Here are some best practices when writing readable code.

1. **Indentation** is the whitespace characters you put at the beginning of a line of code. Indentation is important in Python. Here is an example:

Good indentation:

x = 5  
  
**if** x == 5:  
 print(**"x is equal to 5"**)

NOT good indentation **(error)**:

x = 5  
  
**if** x == 5:  
print(**"x is equal to 5"**)

1. **Code spacing**. This is another making code able to be read easily. Here are some examples of code spacing.

name = **"John Smith"** *# Good*name=**"John Smith"** *# NOT good*print(**"Hello world"**) *# Good*print (**"Hello world"**) *# NOT good*a = b + c *# Good*a=b+c; *# NOT good*a = b - c *# Good*a = b-c *# NOT good*a = b \* 2 + c / 3 *# Good*a = b\*2 + c/3 *# NOT good*

1. **Comments**. Try not to write comments that do not need to be written. Remember, you do not have to write a comment for every line of code. Here are 2 examples, where the comments in the first is good, and the number of comments in the second example is too large.

**def** GoodComments():  
  
 *# Calculate and print area of a square* l = int(input(**"Enter length of side: "**))  
  
 **if** l <= 0:  
 print(**"Please enter a side greater than 0"**)  
 **else**:  
 print(**"Area is {0}"**.format(l \* l))  
  
  
  
  
**def** TooManyComments():  
  
 *# Get length of side of the square from user* l = int(input(**"Enter length of side: "**))  
  
 *# If the length is less than or equal to 0* **if** l <= 0:  
 *# Show error message* print(**"Please enter a side greater than 0"**)  
 **else**: *# If the length is greater than 0  
 # Calculate and print the area of the square* print(**"Area is {0}"**.format(l \* l))

1. **Naming variables and methods**. The names of variables and methods are important programming. The names help you and others understand what the code does. For example, the variable name max\_score is a much better name than xyz. The second aspect of naming scheme is that you should have word boundaries in the name.

You should also keep the same style of naming. There are two options, **camelCase** and **underscores**. The **underscores** option is recommended for the Python language. For example:

my\_dog, max\_temperature