

**Python Object Oriented Programming**

Assignment 2

# 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: ***ZhaoYinTing***

Student ID: ***201706011100048***

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 Copying or paraphrasing someone else’s work be it published, unpublished or off the internet, without clearly acknowledging it constitutes plagiarism and is considered to be academic misconduct.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 2

This is how the questions are marked in Assignment 2:

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 2

**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
* Using GUI elements to perform calculations and output data
* 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

**Weighting:** 20%

**Mark allocation on questions:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Question No. | 1 | 2 | 3 | 4 | Total |
| Marks | 25 | 25 | 25 | 25 | 100 |

**Due date:** see timetable on Moodle

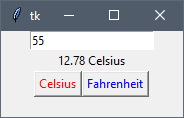
**Marking process:**

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

Question 1

|  |
| --- |
| GUI |
| * Learning outcomes: 1&3 * Relevant topics: input, output, selection, functions, GUI, global variables * Suggested time to complete: Week 10 * Workbook topic: *GUI* |

Make this form to convert temperatures between Celsius and Fahrenheit:

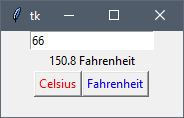


Entry

Label

Button

Button



Entry

Label

Button

Button

The user types in a temperature and clicks the *Celsius* button to change the temperature to *degrees* *Celsius,* or clicks the *Fahrenheit* button to change the temperature to *degrees Fahrenheit.*

1. Import *tkinter*
2. Make the *root* window object
3. Make a **Top Frame** and a **Bottom frame**.
4. Make an **Entry** text box
5. Make a **Label** that says ""
6. Put the **Entry** and **Label** in the **Top Frame**.
7. Make a **Function** called *calculate\_celsius()*
   1. Get the text inside the **Entry**, make it a float and put it in a variable
   2. Calculate the temperature in *degrees Fahrenheit*
   3. Put the new temperature in the label using this code:

myLabel.config(text=str(new\_temp))

1. Make a **Function** called *calculate\_fahrenheit()*
   1. Get the text inside the **Entry**, make it a float and put it in a variable
   2. Calculate the temperature in *degrees Celsius*
   3. Put the new temperature in the label using this code:

myLabel.config(text=str(new\_temp))

1. Make a **Button**
   1. Make the button text say “Celsius”
   2. Make the colour red
   3. Make the command use the *calculate\_celsius()* **Function**
2. Make a **Button**
   1. Make the button text say “Fahrenheit”
   2. Make the colour green
   3. Make the command use the *calculate\_fahrenheit()* **Function**
3. Put the **Buttons** in the **Bottom Frame**

Run the *mainloop* function of *root*

Question 2

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

Make a class called **Car**. A car has a **name** and a quantity (**number**).

* Make the **\_\_init\_\_** method
* Make an object variable called **name** and an object variable called **number**
* Make a method called **details** that prints “Car: … Number: … “
* Below is the code that makes the **Car** objects. It uses a **list** to make the objects. This list is passed to a method called **print\_cars()** that you need to write the code for:

cars = [Car(**"BMW"**, 4), Car(**"Volvo"**, 2), Car(**"VW"**, 0), Car(**"Saab"**, 8), Car(**"Kia"**, 3)]  
  
**def** print\_cars(cars):

<You need to write the code here>

<Make a variable called total>

<Loop through the cars list and call the details() method on each object>

<Add the number of cars to the total variable>

<Print the total number of cars>

print\_cars(cars)

It should make this output:

Car: BMW Number: 4

Car: Volvo Number: 2

Car: VW Number: 0

Car: Saab Number: 8

Car: Kia Number: 3

Total number of cars: 17

Question 3

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

* Make a class called Shape
* Make the \_\_init\_\_ method with one object variable called colour
* Make a method called get\_shape\_type that returns “General shape”
* Make a class called Rectangle. It **inherits** the Shape class.
* Make the \_\_init\_\_ method with 2 object variables: one is called width and one is called height
* You will need to call the \_\_init\_\_ method of the Shape class
* Make a method called get\_shape\_type() that returns “Rectangle”
* Make a method called get\_perimeter() that returns the perimeter of the rectangle.
* Below is the code that makes the **Rectangle** objects. It uses 2 **lists** to make the objects. This list is passed to a method called **print\_Rectangles()** that you need to write the code for:

rectangle\_list1 = [Rectangle(**"white"**, 4, 3),  
 Rectangle(**"red"**, 9, 5),  
 Rectangle(**"purple"**, 3, 6),  
 Rectangle(**"orange"**, 15, 10),  
 Rectangle(**"black"**, 4, 14)]  
  
rectangle\_list2 = [Rectangle(**"pink"**, 3, 4),  
 Rectangle(**"red"**, 10, 2),  
 Rectangle(**"white"**, 8, 5),  
 Rectangle(**"blue"**, 14, 4),  
 Rectangle(**"grey"**, 10, 15)]

<You need to write the code that finds how many objects have the same colour. You will need to use nested FOR loops>

<Inside the same loops, you need to write the code that finds how many objects have the same perimeter>

<Print out how many objects have the same colour>

<Print out how many objects have the same perimeter>

It should make this output:

There are 2 rectangle objects with the same colours

There are 3 rectangle objects with the same perimeters

Question 4

|  |
| --- |
| Abstract classes |
| * Learning outcomes: 1&3 * Relevant topics: output, iteration, list, class, objects, encapsulation, inheritance, abstract * Suggested time to complete: Week 11 * Workbook topic: *Polymorphism and* *Abstract* |

* Make an **abstract** class called Reading
* Make the \_\_init\_\_ method with one object variable called number\_of\_pages
* Make an **abstract** method called details() that has pass
* Make a class called Book. It **inherits** the Reading class.
* Make the \_\_init\_\_ method with 1 object variable called author
* You will need to call the \_\_init\_\_ method of the Reading class
* Make a method called details() that prints “The book was written by … and has … pages”
* Make a class called Newspaper. It **inherits** the Reading class.
* Make the \_\_init\_\_ method with 1 object variable called issue\_date
* You will need to call the \_\_init\_\_ method of the Reading class
* Make a method called details() that prints “The newspaper is for … and has … pages”
* Below is the code that makes the **Reading** objects. It uses a **list** to make the objects. This list is passed to a method called **print\_reading()** that you need to write the code for:

readings = [Book(545,**"Dan Brown"**), Book(675, **"Stephen King"**), Newspaper(54, **"2017/11/01"**), Newspaper(34, **"2017/10/23"**)]  
  
 <You need to write the code here>

<Make a variable called total>

<Loop through the readings list and call the details() method on each object>

<Add the number of pages to the total variable>

<Print the total number of pages>  
  
print\_reading(readings)

It should make this output:

The book was written by Dan Brown and has 545 words

The book was written by Stephen King and has 675 words

The newspaper is for 2017/11/01 and has 54 words

The newspaper is for 2017/10/23 and has 34 words

Total number of pages: 1308

**(words need to change to pages)**

**Question 5 ?**

### 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