# *Introduction to Programming (ITAS 185)*

# *Lab 6 – Classes*

Date due: **Fri, October 27, 2023, by 14:00**

**Learning Objectives**

Upon successful completion of this lab exercise, the student will be able to:

1. Create simple Python classes
2. Create simple Python programs to run each class
3. Create constructors, attributes and methods for classes

**To be handed in:**

1. The ***username185L06*** folder should be zipped and uploaded to the ITAS Portal. ***Username*** isyour logon username (mine would be allan.mcdonald).

**To start:**

1. Create a folder called ***username*185L06** that you will use for ALL the files in the lab.
2. Open this **FOLDER** in VS Code. Make sure you open the folder and not just the files.

# Employee Class

**To Do**:

## Create a file called Employee.py and write a class named Employee that has the following attributes:

### emp\_name: The name field is a string attribute that holds the employee’s name.

### id\_number: The id\_number is an integer attribute that holds the employee’s ID number.

### department: The department is a string attribute that holds the name of the department where the employee works.

### position: the position field is a string attribute that holds employee’s job title.

## Add a method called get\_info that returns a string with the Employee information in the form: Name (id\_number)<<tab>>department<<tab>>position

## Do **NOT** create a constructor for this class.

## Once you have written the class, write a separate program called employee\_run.py that imports the Employee class and creates three Employee objects to hold the following data:

|  |  |  |  |
| --- | --- | --- | --- |
| Name | ID Number | Department | Position |
| Kay Oss | 47899 | Accounting | Vice President |
| Ben Dover | 39119 | IT | Programmer |
| Al E. Gator | 81774 | Manufacturing | Engineer |

## The program should store this data in the three objects (once again, you DO NOT have a constructor) and then display the data for each employee on the screen using the get\_info method.

# Car Class

**To Do**:

## Write a class named Car that has the following attributes:

### model\_year: The model\_year field is an int attribute that holds the car’s year of manufacturer;

### make: The make field is a String attribute that holds the make of the car, such as “Ford”, “Chevrolet”, “Honda”, etc.

### speed: The speed field is an int that holds the car’s current speed.

## In addition, the class should have the following methods:

### Constructor (\_\_init\_\_). The constructor should accept the car’s model\_year and make as arguments. These values should be assigned to the appropriate fields of the object. The constructor should also and assign 0 to the speed field (make sure to assign it to self.speed).

### String method (\_\_str\_\_). This method returns a string in the for ‘The {model\_year} {make} is going {speed} kph’; for example, ‘The 2022 Tesla is going 213 kph’

### accelerate method. The accelerate method adds 5 to the speed attribute each time it is called.

### brake method. The brake method subtracts 5 from the speed attribute each time it is called. If the speed goes less than 0, the speed is set to 0.

## Once you have written the class, write a separate program called car\_run.py to demonstrate that the class works, create an object, and then calls the accelerate method 5 times. After each call to the method, display the car information. Then, call the brake method six times. After each call to the method, display the car information.

# Mass Class

1. You are going to create a Mass class in Python which will allow the person to create objects of different units and perform operations on them. Call the file Mass.py.
2. The following dictionary defines the conversion rates relative to the kilogram

A screen shot of a computer

Description automatically generated

1. Define a class called Mass in a file called Mass.py. Copy the above dictionary from the source file info.txt (in the folder you downloaded) and put it immediately after the class declaration.
2. Define the constructor for the class (dunder init) to assign the **value** and the **unit** attributes. Default the unit to kg.
   1. For example, mass1 = Mass(10, ‘lb’) creates an object of 10 pounds, mass2 = Mass(5) an object of 5 kg, etc.
3. Define a method called convert\_to\_kg which has no parameters and returns the value of the current object in kg. To do this, return the product of the current value by the value in the above dictionary (self.value \* Mass.\_conversion[self.unit])
4. Override the print function (dunder str) to return the string consisting of the value in kg with the string kg added to it. For example, 2.4 kg. Do this by calling the function convert\_to\_kg and returning the value with kgs appended to it. The number is printed with 2 decimal places.
5. Override the representational function (dunder repr) to return a string in the form Mass(self.value, self.unit); for example, Mass(14.30, stone). Again, display to 2 decimal places.
6. Create test cases in a file called mass\_test.py to make sure the constructor, print, print(repr()), and convert\_to\_kg all work.

# To finish:

## 1. Show your completed lab to me, to be marked.

2. Zip your **entire** folder.

3. Submit the zip file to the ITAS portal.