**Exercises: Advanced topics in Python**

**(List comprehensions and lambdas)**

**Exercise 1**

Use a list comprehension to create a list, `cubes\_by\_four. The comprehension should consist of the cubes of the numbers 1 through 10 only if the cube is evenly divisible by four. Finally, print that list to the console. Note that in this case, the cubed number should be evenly divisible by 4, not the original number.

**Exercise 2**

Create a variable, backwards\_by\_tens, and set it equal to the result of going backwards through to\_one\_hundred by tens. Go ahead and print backwards\_by\_tens to the console.

**Exercise 3**

Create a list, to\_21, that's just the numbers from 1 to 21, inclusive. Create a second list, odds, that contains only the odd numbers in the to\_21 list (1, 3, 5, and so on). Use list slicing for this one instead of a list comprehension. Finally, create a third list, middle\_third, that's equal to the middle third of `to\_21, from 8 to 14, inclusive.

**Exercise 4**

Create a list, squares, that consists of the squares of the numbers 1 to 10. A list comprehension could be useful here. Use filter() and a lambda expression to print out only the squares that are between 30 and 70 (inclusive).

**Exercise 5**

The string

garbled = "!XeXgXaXsXsXeXmX XtXeXrXcXeXsX XeXhXtX XmXaX XI"`

is garbled in two ways:

* First, our message is backwards;
* Second, the letter we want is every other letter.

Use lambda and filter to extract the message and save it to a variable called message. Use list slicing to extract the message and save it to a variable called message.

**Exercise : File I/O**

**Take any text file which you already had with you**

1. Write a Python program to read an entire text file.
2. Write a Python program to read first n lines of a file.
3. Write a Python program to append text to a file and display the text.
4. Write a Python program to read a file line by line and store it into a list.

**Exercises:Classes**

**Exercise 1**

Follow the steps:

* Create a class, Triangle. Its \_\_init\_\_() method should take self, angle1, angle2, and angle3 as arguments. Make sure to set these appropriately in the body of the \_\_init\_\_()method.
* Create a variable named number\_of\_sides and set it equal to 3.
* Create a method named check\_angles. The sum of a triangle's three angles is It should return True if the sum of self.angle1, self.angle2, and self.angle3 is equal 180, and False otherwise.
* Create a variable named my\_triangle and set it equal to a new instance of your Triangle class. Pass it three angles that sum to 180 (e.g. 90, 30, 60).
* Print out my\_triangle.number\_of\_sides and print out my\_triangle.check\_angles().

**Exercise 2**

Define a class called Songs, it will show the lyrics of a song. Its \_\_init\_\_() method should have two arguments:self anf lyrics.lyrics is a list. Inside your class create a method called sing\_me\_a\_song that prints each element of lyrics on his own line. Define a varible:

happy\_bday = Song(["May god bless you, ",

"Have a sunshine on you,",

"Happy Birthday to you !"])

Call the sing\_me\_song mehod on this variable.

**Exercise 3**

Define a class called Lunch. Its \_\_init\_\_() method should have two arguments:self anf menu.Where menu is a string. Add a method called menu\_price.It will involve a ifstatement:

* if "menu 1" print "Your choice:", menu, "Price 12.00", if "menu 2" print "Your choice:", menu, "Price 13.40", else print "Error in menu".

To check if it works define: Paul=Lunch("menu 1") and call Paul.menu\_price().

**Exercise 4**

Define a Point3D class that inherits from object Inside the Point3D class, define an \_\_init\_\_() function that accepts self, x, y, and z, and assigns these numbers to the member variables self.x,self.y,self.z. Define a \_\_repr\_\_() method that returns "(%d, %d, %d)" % (self.x, self.y, self.z). This tells Python to represent this object in the following format: (x, y, z). Outside the class definition, create a variable named my\_point containing a new instance of Point3D with x=1, y=2, and z=3. Finally, print my\_point.

## Exercise : Car

We are going to follow the exercise instructions step by step and give solutions to each step, so you can follow it easily. Also, we will review the concepts shown in classes to fix them.

#### Ready? The computing starts here:

* Defining a class is much like defining a function, but we use the **class** keyword instead. We also use the word object in parentheses because we want our classes to inherit the object class. This means that our class has all the properties of an object, which is the simplest, most basic class.
  + Define a new class named "Car". For now, since we have to put something inside the class, use the pass keyword.
  + We can use classes to create new objects, which we say are **instances** of those classes.
  + Below your Car class, create a new object named my\_car that is an instance of Car.

**Solution 1**

>>>

>>>

* Classes can have member variables that store information about each class object. We call them **member variables** since they are information that belongs to the class object.
  + Inside your Car class, replace the pass statement with a new member variable named `condition and give it an initial value of the string "new".
  + At the end of your code, use a print statement to display the condition of my\_car.

**Solution2**

>>>

>>>

* There is a special function named \_\_init\_\_() that gets called whenever we create a new instance of a class.The first argument passed to**init**() must always be the keywordself - this is how the object keeps track of itself internally - but we can pass additional variables after that.In order to assign a variable to the class (creating a member variable), we use **dot notation**.
  + Define the \_\_init\_\_() function of the Car class to take four inputs: self, model, color, and mpg. Assign the last three inputs to member variables of the same name by using the self keyword.
  + Then, modify the object my\_car to provide the following inputs at initialization:
  + model = "DeLorean"
  + color = "silver"
  + mpg = 88

**Solution 3**

>>>

>>>

* Calling class member variables works the same whether those values are created within the class (like our car's condition) or values are passed into the new object at initialization. We use dot notation to access the member variables of classes since those variables belong to the object.
  + Now that you've created my\_car print its member variables:First print the model of my\_car. Then print out the color of my\_car.Finally, print out the mpg of `my\_car.

**Solution 4**

>>>

>>>

* Besides member variables, classes can also have their own **methods** (functions inside the class).
  + Inside the Car class, add a method named display\_car() to Car that will reference the Car's member variables to return the string, "This is a [color] [model] with [mpg] MPG." You can use the str() function to turn your mpg into a string when creating the display string.
  + Replace the individual print statements with a single print command that displays the result of calling my\_car.display\_car().

**Solution 5**

>>>

>>>

* We can modify variables that belong to a class the same way that we initialize those member variables.
  + Inside the Car class, add a method drive\_car()that sets self.condition to the string "used".
  + Remove the call to my\_car.display\_car() and instead print only the condition of your car.
  + Then drive your car by calling the drive\_car() method.
  + Finally, print the condition of your car again to see how its value changes.

**Solution 6**

>>>

>>>

* One of the benefits of classes is that we can create more complicated classes that inherit variables or methods from their **parent classes**. This saves us time and helps us build more complicated objects, since these **child classes** can also include additional variables or methods.
  + Create a class ElectricCar that inherits from Car. Give your new class an \_\_init\_\_() method of that includes a "battery\_type" member variable in addition to the model, color and mpg.
  + Then, create an electric car named "my\_car" with a "molten salt" battery\_type. Supply values of your choice for the other three inputs (model, color and mpg).

**Solution 7**

>>>

>>>

* Since our ElectricCar is a more specialized type of Car, we can give the ElectricCar its own drive\_car() method that has different functionality than the original Car class's.
  + Inside ElectricCar add a new method drive\_car() that changes the car's condition to the string "like new".
  + Then, outside of ElectricCar, print the condition of my\_car.
  + Next, drive my\_car by calling the drive\_car() function.
  + Finally, print the condition of my\_car again

**Solution 8**

>>>

>>>

## Exercise: Robot Simulator

Introduction

Write a robot simulator.

A robot factory's test facility needs a program to verify robot movements.

The robots have three possible movements:

* turn right
* turn left
* advance

Robots are placed on a hypothetical infinite grid, facing a particular direction (north, east, south, or west) at a set of {x,y} coordinates, e.g., {3,8}, with coordinates increasing to the north and east.

The robot then receives a number of instructions, at which point the testing facility verifies the robot's new position, and in which direction it is pointing.

* The letter-string "RAALAL" means:
  + Turn right
  + Advance twice
  + Turn left
  + Advance once
  + Turn left yet again
* Say a robot starts at {7, 3} facing north. Then running this stream of instructions should leave it at {9, 4} facing west.

Exception messages

Sometimes it is necessary to raise an exception. When you do this, you should include a meaningful error message to indicate what the source of the error is. This makes your code more readable and helps significantly with debugging. Not every exercise will require you to raise an exception, but for those that do, the tests will only pass if you include a message.

To raise a message with an exception, just write it as an argument to the exception type. For example, instead of raise Exception, you should write:

raise Exception("Meaningful message indicating the source of the error")