1. **zyBooks Labs**

Please follow the link on Canvas to complete the following zyBooks labs:

* + 9.11 *LAB: Car value (classes)*
  + 9.12 *LAB: Nutritional information (classes/constructors)*
  + 9.13 *LAB: Artwork label (classes/constructors)*
  + 9.14 *LAB: Triangle area comparison (classes)*
  + 9.15 *LAB: Winning team (classes)*
  + 9.16 *LAB\*: Warm up: Online shopping cart (Part 1)*

This portion of the lab will be worth one-half of your Lab 09 grade.

1. **Creating a Clock Class**

In this lab component you will write a small, but complete Python 3 program called **Lab9A.py** that creates our own data type to represent time as a clock. Each Clock instance will represent the present time in hours, minutes, and seconds, which means that each instance will have three data attributes: hours, minutes, and seconds. This can be accomplished as follows:

* 1. Let’s first create a version of the Clock class that essentially does nothing using the pass keyword, since Python does not allow an empty function/method or class.

class Clock:

pass

Anything defined inside the class must be indented. Let’s create an instance of this Clock class outside of the class to see what it already does:

my\_clock = Clock() # calls the constructor

print('My clock is: ', my\_clock)

print('Type is:', type(my\_clock))

print(dir(my\_clock))

Run the above code to see what is print out. Even though we did not create a constructor (yet!), a constructor was created for us that essentially made a Clock instance. You will recognize that the dir command showed a lot of the double underlined elements, such as '\_\_class\_\_' that Python automatically provides for us.

* 1. Now, instead of pass, let’s build our own constructor that adds the three attributes to each instance, initializing them after Python creates the default instance as follows:

def \_\_init\_\_(self, hours = 0, minutes = 0, seconds = 0):

self.hours = hours

self.minutes = minutes

self.seconds = seconds

Python uses the \_\_init\_\_() as the constructor. All methods, including the constructor, must have self as the first parameter, which allows the class to refer to itself. Our constructor also has three *default* parameters so that our default instance will have all attributes set to 0.

The self.hours = hours statement means that the default instance, now associated with the variable self, will have created at attribute called hours within that instance. The value of that attribute will be set to the value of the parameter hours. Note that it should be clear that, for the variable hours, which is the parameter and which is the attribute. The attribute shows up as part of an instance and is scoped with self, while the parameter has no such instance.

If you check out the dir command again, you will notice that the class now contains the \_\_init\_\_ constructor as well as the attributes hours, minutes, and seconds listed in the results.

In addition to our my\_clock instance, let’s also create a new Clock instance with some non-default values for the attributes to see what is printed:

new\_clock = Clock(10, 15, 30)

print(new\_clock.hours, new\_clock.minutes, new\_clock.seconds)

You should now notice that the Clock class provides a little more useful information!

* 1. We can modify our constructor to deal with clock arithmetic so that no hour will be greater than 23 and no minute or second will be greater than 59. When a sum is larger than the limit, a carry is added (although no carry is needed for hours since there is no day represented). Make the following changes to the constructor:

self.seconds = seconds % 60

total\_minutes = minutes + (seconds // 60)

self.minutes = total\_minutes % 60

self.hours = (hours + (total\_minutes // 60)) % 24

Notice that local variable total\_minutes was included without prepending self, which means that total\_minutes will be discarded when \_\_init\_\_() method goes out of scope, instead of persisting with the object.

We can add another Clock instance to check out what happens when adding an extra second to the seconds data member when instantiating a Clock object.

tst\_clock = Clock(23, 59, 60)

print(tst\_clock.hours, tst\_clock.minutes, tst\_clock.seconds)

You should notice that tst\_clock prints all 0’s. Make sure you understand why!

* 1. So far, printing our Clock objects has been fairly simple and plain. Let’s add a member function called print\_clock() that will print a better "formatted" Clock object. Inside the Clock class, add the following code:

def print\_clock(self):

print('{:02d}:{:02d}:{:02d}'.format(self.hours, self.minutes, self.seconds)

The print\_clock() method does not take any arguments, but it does have one parameter, the instance that we are printing (i.e., self). We grab the three attributes from self and print them. Replace your print() statements for new\_clock and tst\_clock with the object function call to print\_clock(), passing no arguments.

* 1. Finally, let’s add a new method called add\_clocks() that allows us to add two Clock objects, generating a new Clock object. This new method should not modify the existing Clock instances, only return a new Clock object. This means that we should call the constructor inside the add\_clocks() method. When we make a new Clock instance, we do not have to enforce restrictions on the attribute values. We can let the constructor do that for us. Just like the print\_clock() method, we add the following method definition to the Clock class:

def add\_clocks(self, clock2):

seconds = self.seconds + clock2.seconds

minutes = self.minutes + clock2.minutes

hours = self.hours + clock2.hours

return Clock(hours, minutes, seconds)

Notice here that we build local variables hours, minutes, and seconds using the instance variables from self and clock2 and then use them to create a new Clock instance that is returned (i.e., the new Clock object).

We can use test our new method out by adding the following to our program:

c1 = Clock(10, 59, 59)

c2 = Clock (1, 1, 1)

c3 = c1.add\_clocks(c2)

c1.print\_clock()

c2.print\_clock()

c3.print\_clock()

Notice how Clock objects c1 and c2 remain unchanged. Our c3 Clock object was created using the attributes from c1 and c2. Notice that c1 is self and c2 is acting as clock2 in the add\_clocks() method call.

For example, the output should look like this (input shown in **bold**):

$ **python3 Lab9A.py**

My clock is: <\_\_main\_\_.Clock object at 0xb78bf62c>

Type is: <class '\_\_main\_\_.Clock'>

['\_\_class\_\_', '\_\_delattr\_\_', '\_\_dict\_\_', '\_\_dir\_\_', '\_\_doc\_\_', '\_\_eq\_\_', '\_\_format\_\_', '\_\_ge\_\_', '\_\_getattribute\_\_', '\_\_gt\_\_', '\_\_hash\_\_', '\_\_init\_\_', '\_\_init\_subclass\_\_', '\_\_le\_\_', '\_\_lt\_\_', '\_\_module\_\_', '\_\_ne\_\_', '\_\_new\_\_', '\_\_reduce\_\_', '\_\_reduce\_ex\_\_', '\_\_repr\_\_', '\_\_setattr\_\_', '\_\_sizeof\_\_', '\_\_str\_\_', '\_\_subclasshook\_\_', '\_\_weakref\_\_', 'add\_clocks', 'hours', 'minutes', 'print\_clock', 'seconds']

10:15:30

00:00:00

10:59:59

01:01:01

12:01:00

*Note that you will submit this file to Canvas.*

Now that you have completed this lab, it’s time to turn in your results. Once you've moved the files to your windows machine (using **WinSCP**), you may use the browser to submit them to Canvas for the **Lab 09** dropbox.

You should submit the following files:

* **Lab9A.py**
* **(Note that the zyBooks labs are submitted separately through Canvas.)**

You may want to ask your TA to check your results before submission.

Now that you've finished the lab, use any additional time to practice writing simple programs out of the textbook, lectures, or even ones you come up with on your own to gain some more experience.