# Lab Exercise #11

## Assignment Overview

You will work with a partner on this exercise during your lab session. Two people should work at one computer. Occasionally switch the person who is typing. Talk to each other about what you are doing and why so that both of you understand each step.

## Part A: Class Date

1. Download the files for this laboratory exercise, then run the Python shell and enter the following commands:

```
>>>  import date
>>>  help( date )
>>>  help( date.Date )
>>>  x = date.Date()
>>>  x
>>>  print( x )
```

2. Consider the Python statements shown below (from the file named "lab11a.py"). Using the output produced by the commands above, predict what the output will be, then test your predictions by executing the statements.

```
import date

A = date.Date( 1, 1, 2014 )

print( A )
print( A.to_iso() )
print( A.to_mdy() )
print( A.is_valid() )
print()

B = date.Date( 12, 31, 2014 )

print( B )
print( B.to_iso() )
print( B.to_mdy() )
print( B.is_valid() )
print()

C = date.Date()

C.from_iso( "2014-07-04" )

print( C )
print( C.to_iso() )
print( C.to_mdy() )
print( C.is_valid() )
print()
```

```
D = date.Date()

D.from_mdy( "March 15, 2015" )

print( D )
print( D.to_iso() )
print( D.to_mdy() )
print( D.is_valid() )
print()

E = date.Date()

print( E )
print( E.to_iso() )
print( E.to_mdy() )
print( E.is_valid() )
print()
```

3. Revise the program to test the behavior of the member function "__init__" when one or more of the arguments to that function are erroneous. That is, test a date such as 13/40/2017.

4. Revise the program to test the behavior of the member functions "from_iso" and "from_mdy" when the arguments to those functions are strings which contains spaces.

5. Revise the program to test the behavior of the member functions "from_iso" and "from_mdy" when the arguments to those functions are erroneous.

★ **Demonstrate your completed program to your TA. On-line students should submit the completed program (named "lab11a.py") for grading via the Mimir system. (Note that there are no automatic tests for this program—your TA will grade it by hand.)**

**Part B: Class Time**

You will develop a new data type to model the time on a 24 hour clock, where the three data members represent the hour, the minutes and the seconds. Use the `date.py` module as a guide.

It will be named "class Time", and each object of that type will have three instance variables ("__hour", "__mins" and "__secs"). The module will be named "clock.py".

1. Develop the function member (method) named "__init__" which will be used to initialize an object of type "Time" when it is created. That function will receive four parameters: a reference to the current object ("self"), the hour, the minutes and the seconds.

   a. Assume that the last three of the four parameters are integers.
   b. Use a default value of 0 for those last three parameters.
   c. Assign those last three parameters to the three instance variables.
   d. Add a "doc string" to the constructor documenting the purpose of the function.

2. Save the file containing your class as “clock.py”, then experiment with your module in the iPython shell:

```
>>>  import clock
>>>  help( clock )
>>>  help( clock.Time )
>>>  A = clock.Time()
>>>  A
>>>  print( A )
```

3. Develop the function member named “__repr__” which will be used to display the formal representation of an object of type “Time” in the Python shell.

a. Return a string with the format `"Class Time: hh:mm:ss".`
b. Be sure to place leading zeroes in the string for the hour, minutes and seconds.
c. Add a “doc string” to the constructor documenting the purpose of the function.

4. Save the file containing your class as “clock.py”, then experiment with your module in the iPython shell:

```
>>>  import clock
>>>  A = clock.Time()
>>>  A
>>>  B = clock.Time(7,12,3)
>>>  B    # should display: Class Time: 07:12:03
```

5. Develop the function member named “__str__” which will be used to display a human-readable representation of an object of type “Time” in Python programs:

a. Return a string with the format `"hh:mm:ss".`
b. Be sure to place leading zeroes in the string for the hour, minutes and seconds.
c. Add a “doc string” to the constructor documenting the purpose of the function.

6. Save the file containing your class as “clock.py”, then experiment with your module in the iPython shell:

```
>>>  import clock
>>>  A = clock.Time()
>>>  print( A )
>>>  B = clock.Time(7,12,3)
>>>  print( B )
```

7. Develop the function member named “from_str” which will be used to update an object of type “Time” after it has been created. That function will receive two parameters: a reference to the current object (“self”) and a string of the form “hh:mm:ss”.

a. Assume that the string parameter can be split into three integers.
   That is, assume that the parameter is in the format specified—no errors.
b. Assign those three integers to the three instance variables.

c. Add a “doc string” to the constructor documenting the purpose of the function.

8. Save the file containing your class as “clock.py”, then experiment with your module in the iPython shell:

```
>>>  import clock
>>>  A = clock.Time()
>>>  print( A )
>>>  A.from_str('07:12:03')
>>>  print( A )
>>>  A
```

9. We provide the Python program named “clockDemo.py” that demonstrates the use of your class “Time”. Test your class using clockDemo.py.

10. Create a new file named “lab11b.py” which contains the source code from the files named “clock.py” and “clockDemo.py”. That is, create a file which contains both the definition of class “Time” and the demonstration program. The class definition should be at the top of the file. To create that “stand alone” file you will need to remove (or comment out) the “import clock” line (you don’t need to import it because it is already in the file). Also, since you have not imported “clock” you need to change “clock.Time” to “Time”. Test that your file is working before submitting.

★ **Demonstrate your completed program to your TA. On-line students should submit the completed program (named “lab11b.py”) for grading via the Mimir system.**

Optional: If you have the time and interest, develop the function member named “add_times” which will receive two objects of type “Time” and return a new object of type “Time”. The new object will represent the sum of the two parameters.

Use modulo arithmetic to handle the addition of the seconds, minutes and hours. If the sum of the seconds exceeds 60, the minutes should be increased by 1. Similarly, if the sum of the minutes exceeds 60, the hour should be increased by 1. Note that we are not keeping track of days, so if the sum of the hours exceeds 24, you should ignore the “carry”.

Revise your program to demonstrate the use of “add_times”. Call the new program lab11c.py and let your TA know that you have done the extra work.