At the start of each of these problems, the name of a Python file is given in **blue**: **foo.py**. Create a new PyCharm project, then

create and save each requested Python program's source code in a file with the same name. Also add a comment at the top of each program giving your name.

Each problem except for **[H5-4]** should define a **main()** function containing the specified code, with a final statement in your file that calls **main()**.

The Help Videos posted for this assignment were created for an earlier version of this assignment but should still be useful. Some of the details below will be different than those in the earlier assignment.

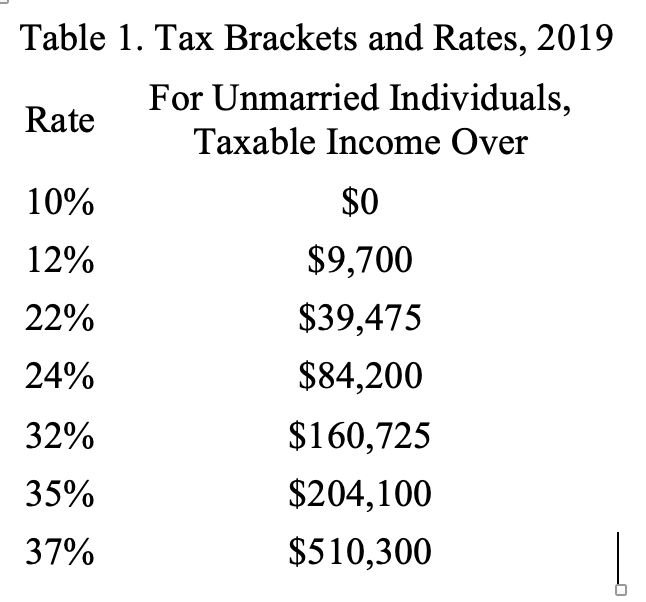
Do each of the following problems. When finished, upload each **.py** file with the specified name to the Canvas **H5 Assignment** link. Do so in a single submission.

**[H5-1]** (**boolexpr.py**) Write a Boolean (**bool**) function **is\_in\_semester(month,day)** that returns **True** when **month/day** falls within the dates of our current GPS Fall Semester 2019, starting on September 4 and ending on December 13. If **month/day** falls outside this range (which includes both starting and ending dates), return **False**.

Then write a **main()** function that reads integers **month** and **day** from the user, calls **is\_in\_semester(month,day)**, then prints out **'month/day is in Fall Semester'** or **'month/day is NOT in Fall Semester'**. Example for **month==12**, **day==1** => print **12/1 is in Fall Semester**. Finally, call **main()** as the last statement in your **.py** file.

Note that your function should only return **True** or **False**, with **main()**calling **is\_in\_semester(...)**, then printing out the text which depends upon its **bool** returned value. No "chatterbox functions"! See the **HTT6 Glossary**...

**[H5-2]** (**speeding.py**) The speeding ticket fine policy in Beldenville is $50 plus $5 for each mph over the limit, plus a penalty of $200 for any speed over 90 mph. Assume that the limit is less than 90 mph.

Write a program that reads a **float** speed limit and another **float** as the measured speed, then either prints a message indicating the speed was legal or prints the amount of the fine if the speed was illegal.

**[H5-3]** (**taxes.py**) Write a Python program that reads the taxable income **income** as a **float** value from the user, then calculate and print out the tax for an unmarried individual with that taxable income. Use the table to the right...

You should also check that the entered **income** is **>= 0.0**; otherwise print "**Bad input**" and quit.

**[H5-4]** (**test\_leap.py**) The year **y** is a leap year if and only if **y** is evenly divisible by **4,** but not by **100** - or if **y** is evenly divisible by **400**. Thus, **2016** is a leap year (evenly divisible by **4** but not by **100**. **1900** is NOT a leap year, since it's evenly divisible by **100** but NOT by **400**. And **2000** IS a leap year, since it's evenly divisible by **400**.

Define a **bool** function **is\_leap(year)** which returns **True** if **year** is a leap year, and **False** otherwise. Then write 10 different **pytest** test functions, each of which tests a different year. Try to pick a variety of both leap and non-leap years. Put both your function and your tests in the **same** file. Also be sure there's no console input or output within your code, or **pytest** will complain.

Remember that each such **pytest** test function must (a) have a name that starts with **test\_**, (b) have no arguments, and (c) provide a test of the form **assert is\_leap(YYYY)==True** if **YYYY** is a leap year, or **assert is\_leap(YYYY)==False** if not.

No **main()** function here: just the function definition and the 10 **pytest** test methods.

**[H5-5]** (**julian.py**) The Julian date of a year is the ordinal number of any date within the year. For example, January 1 is always Julian date 1, February 1 is Julian date 32, and December 31 is Julian date 365 in non-leap years and 366 in leap years.

Here's a way of calculating the Julian date **daynum**. Use **int** arithmetic in these calculations:

(1) **daynum = 31\*(month−1) + day**

(2) If the month is after February, subtract **(4\*month + 23) // 10** from **daynum**

(3) If it’s a leap year and after February 29, add 1 to **daynum**

Using this algorithm, write a function **julian(day,month,year)** which returns the **int** Julian date of its arguments. Define a **main()** function which reads the date from the user as three separate **int** values; you don't have to verify that it's valid. Then call your function passing the three input arguments and print the **int** result.

**Hint**: use your **is\_leap(year)** function from **[H5-4]** in your test for (3) above. Also be careful about implementing the above calculations: do them in the order given (do (1) first, then (2), then (3)).