

### Problem #1 (5 points): Learn creating a library of reusable modules and defining classes

1. Close all your open projects by right click on a project > **Close Project**.
2. Create a Python project named *FirstName-LastName-HW3* (e.g., *Jane-Doe-HW3*) with a *src* folder. (How? See the HW1 instructions).
3. Add a new package named *lib* to the *src* folder. Note that a Python package must contain an *\_\_init\_\_.py* module.
4. Add a module named *geometry* to the package *lib*.
  - a. Note 1: When you add a module that will contain one or more class definitions, you might want to pick the *Module: Class* option from the list of module templates. But it is not recommended the template follows the Python 2 style.
  - b. Note 2: In Python a module name should consist of only lowercase letters and underscores. However, class names in a module should begin with a capital letter. You may find that many built-in classes do not follow this convention. These are holdovers from earlier stages of Python when there was a much bigger difference between user-defined classes and built-in classes.
  - c. Define a class named *Triangle* in the *geometry* module as follows:
    - i. Define an *instance constructor* that initializes three instance variables, say, *side1*, *side2*, and *side3*.
    - ii. Define a method named *isTriangular*.
      1. It *returns* `True` if the values of the instance variables that are taken as arguments could be the sides of a triangle (i.e., none of them is greater than or equal to the sum of the other two), and `False` otherwise. Note that `True` and `False` are reserved keywords of Python. They are not string values but are Boolean ones.
      2. Your implementation of this method must include *nested* `if` and `return` statements *only*.
  - d. Test the class by adding a ***tester snippet*** to the module.

```
def main():  
    # statements to create an instance,  
    # call the method, and print the output  
  
if __name__ == '__main__':  
    main()
```

- e. The output is shown below. Note that you need two instance of the class *Triangle* to print this output. Note: When possible, use variable references to construct output.

```
The values of 5, 4, 3 could be the sides of a triangle.  
The values of 2, 1, 3 could not be the sides of a triangle.
```

5. Add a module named *algebra* to the package *lib*.
  - a. Define a class named *Max* in the *algebra* module as follows:
    - i. Do not define a constructor.

- ii. Define a method named `max3` that takes three `int` or `float` arguments and returns the largest one.
- iii. Define a method named `max5` that takes five `int` or `float` arguments and returns the largest one. This method must utilize the method `max3`.
- b. Test the class by adding a **tester snippet** to the module.
- c. The output is shown below. Note that you need just one instance of the class `Max` to print this output. Note: When possible, use variable references to construct output.

```
The max of 9, 6, 7 is 9
The max of 5, 7, 3, 8, 1 is 8
The max of 9.5, 6.5, 7.5 is 9.5
The max of 5.5, 7.5, 3.5, 8.5, 1.5 is 8.5
```

6. Add a new package named `hw3p1` to the `src` folder. Note that a Python package must contain an `__init__.py` module.
  - a. Add a module named `myapp` to the package `hw3p1`.
    - i. This module is going to reuse the modules in the `lib` package. Such a module is known as a *launcher* or *main* module. When you add a launcher module like `myapp`, you might want to pick the *Module: Main* option from the list of module templates to auto-add the `if __name__ == '__main__':` that should be included in every main module. Note that you need to code a complete tester snippet by adding a `main` method to this `myapp` module. Recall that the name of a main method can be any valid name.
  - b. Import the `Triangle` and `Max` classes from the `lib` package created above and call the methods of the classes to print the *combined output* shown below. Your output must match exactly, including a blank line between the output from each class.

```
The values of 5, 4, 3 could be the sides of a triangle.
The values of 2, 1, 3 could not be the sides of a triangle.

The max of 9, 6, 7 is 9
The max of 5, 7, 3, 8, 1 is 8
The max of 9.5, 6.5, 7.5 is 9.5
The max of 5.5, 7.5, 3.5, 8.5, 1.5 is 8.5
```

**Problem #2 (5 points): Learn the built-in `random` module, `random.randrange()`, `input()`, `int()`, `while` loop, `if...elif...else`, and `try: except ValueError:`. Also learn importing a module with the keyword `import` only.**

1. Add a new module named `gamemd` to the package `lib`.
2. Write code *right in the module (no functions or classes)* so that, when it is run by a launcher or main module, it prints output similar to the one shown below. Hints: Declare a variable `ceiling` and use it to generate a random integer, such as `secret = random.randrange(1, ceiling+1)`.

```

I am thinking of a secret number between 1 and 100
What is your guess? -1
Please enter a positive integer
What is your guess? e
Please enter an integer
What is your guess? 3
Too low
What is your guess? 5
Too low
What is your guess? 7
Too low
What is your guess? 23
Too high
What is your guess? 21
Too high
What is your guess? 19
You win!

```

3. Add a package named `hw3p2` to the `src` folder and add a launcher module named `myapp2`.
  - a. Import the `gamemd` to the `myapp2` module from the `lib` package by using only the keyword `import` (no `from`).
  - b. Run the imported module.

**Problem #3 (5 points):** Learn how to wrap global code into a class by converting the module of the **Problem2** to a class. Also learn importing a module with the keywords `import` and `from`.

1. Add a new module named `gamecs` to the package `lib`.
2. Define a class named `Game` in the module with *an instance constructor and a method*.
  - a. The instance constructor `__init__` is used to instantiate the `Game` class with a value for the `ceiling` variable.
  - b. The method named `play` lets the user play the game. Don't forget to add a ***tester snippet*** to test this class.

```

I am thinking of a secret number between 1 and 100
What is your guess? w
Please enter an integer
What is your guess? -1
Please enter a positive integer
What is your guess? 2
Too low
What is your guess? 3
Too low
What is your guess? 19
Too low
What is your guess? 30
Too low
What is your guess? 57
Too high
What is your guess? 39
Too low
What is your guess? 45
Too low
What is your guess? 47
Too high
What is your guess? 46
You win!

```

3. Add a package named `hw3p3` to the `src` folder and add a launcher module named `myapp3`.
  - a. Import the `gamecs` module to the `myapp3` module from the `lib` package by using only the keywords `import` and `from`.
  - b. Add a **tester snippet** to `myapp3` and write code to run the `gamecs` module. Try with 30, 50, and 100 for the `ceiling`.

**Problem #4 (5 points): Learn creating a class, for and while loop, logical operator, and string concatenation**

1. Consider the code below. You are required to implement similar functionalities using a class. Please try this code first before you implement a class.

```
import random

# declare variables with explicit types, here int variables.
# The None keyword is indicate a null value.
# A null value means not yet assigned or created or unknown.
# None is not the same as 0, False, or an empty string.
stake = int(10)      # same as stake = 10
goal = int(20)       # same as goal = 20
trials = int(1000)   # same as trials = 1000

# run experiments 'trials' times that start with stake
# and terminate on cash == 0 or meeting the goal.
bets = 0
wins = 0
# range(n) function creates a sequence of numbers from 0 to n-1
for t in range(trials):
    # run one experiment.
    cash = stake
    while cash > 0 and cash < goal:
        # simulate one bet.
        bets += 1
        # return a random integer between 0 and 1 (2 is excluded)
        if random.randrange(0, 2) == 0:
            cash += 1 # prefix increment; same as cash = cash + 1
        else:
            cash -= 1 # prefix decrement; same as cash = cash - 1
    if cash == goal:
        wins += 1 # prefix increment; same as wins = wins + 1

print('Number of Loops: ' + str(t))
print('Win rate: ' + str(100 * wins//trials) + '%')
print('Avg number of bets: ' + str(bets//trials))
```

2. Add a new module named `highroller` to the package `lib` and define a class named `HighRoller` as follows. You are supposed to write code based on the code above to replicate the output given below. Note that the wager, number of loops, win rate, and average number of bets will vary depending on the user input. The example output below shows the results of three runs.
3. Don't forget to add a **tester snippet** to test this class.

```
Enter stake: 1000
Enter goal: 2000
Enter trials: 100
Enter 1, 2, or 5: 1
Gone all in. wager: 1000
```

```
Number of loops: 99
Win rate: 48%
Avg number of bets: 1
```

```
Enter stake: 1000
Enter goal: 2000
Enter trials: 100
Enter 1, 2, or 5: 2
wager: 500
```

```
Number of loops: 99
Win rate: 54%
Avg number of bets: 3
```

```
Enter stake: 1000
Enter goal: 2000
Enter trials: 100
Enter 1, 2, or 5: 5
wager: 200
```

```
Number of loops: 99
Win rate: 59%
Avg number of bets: 26
```

```
Enter stake: 1000
Enter goal: 2000
Enter trials: 100
Enter 1, 2, or 5: 6
Invalid
Enter 1, 2, 3, 4, or 5: # continues until a valid choice
```

4. Add a package named `hw3p4` to the `src` folder and add a launcher module named `myapp4`.
  - a. Import the `highroller` module to the `myapp4` module from the `lib` package in your preferred way.
  - b. Add a *tester snippet* to `myapp4` and write code to run the `highroller` module.

## How to turn in your homework?

1. Start **Eclipse**
2. Open your project to export
3. Right click on the project name and select **Export....** (See the image below)
4. Expand **General** and choose **Archive File** and then click **Next**