**ITM891**

**Lab03: Modules & Classes**

I have uploaded a that contains the starter code for this part of the exercise. You'll find the zip file in the lab link for today's class. I would like you to extract the contents of this file and put its contents as labs/03 in your GitHub repository directory. (So put the 03 folder in the labs folder.)

* Lab03,04 - A Game of Decisions.ipynb
  + This is a Python Notebook that contains a description of the lab and lets you test your solution graphically within the notebook.
* pointy\_game.py
  + A Python module that implements the gameboard and gameplay of the PointyGame.
* strategy.py
  + A Python module that implements different strategies to play the game. We will use these later.
* creature.py
  + A Python module that you will complete to implement movement of the creature on the gameboard.
  + We will go over this solution as part of our lab next class.
* screenshot.png
  + Just a png file to show the game board in the documentation. There are ways to inline images without external files, but it'd create a large block of data in the middle of the Markdown documentation. For the purposes of demonstration, I decided not to use that solution and instead use an external file.

There are two types of Python files here: \*.py module files and \*.ipynb notebook files. \*.py files are Python modules. They contain executable Python code and are what we use to import functionality into our notebooks. We separate our functionality into different modules to isolate features and share them with others. \*.ipynb files are Python (Jupyter) Notebook files. They contain the cells and markdown that we will use to play our game and eventually collect data from it.

**Our goal for this lab is for you to understand the concept of a class and finish writing the methods of the Creature class that are not finished. These methods are:**

* is\_alive
* kill
* rotate\_left
* rotate\_right
* move\_forward

In order to implement these functions, you will have to know how to manipulate the data attributes of an instance of the Creature class and return, when needed, the appropriate values. To test your solution, I have included a *unit test* file called test\_creature.py.

**Setting Up Your Environment**

We need packages installed in your CSE801 environment you created last week.

1. Open Anaconda Navigator
2. Navigate to the Environments Tab
3. Activate your CSE801 (or whatever you called it) environment by clicking on it, turning it green.
4. Switch the package view to "Not Installed" from the menu next to the "Channels" button.
5. Search for and install the following packages:

* matplotlib
* numpy

Matplotlib is the primary package that is used to make graphs. There are many wrappers around it to make it's large feature set more usable to create better looking graphs. We'll use the package seaborn later to do this. Numpy is the Python numerical package. We'll discuss it more next week.

I recommend using Visual Studio code for this exercise, though you may use Jupyter Notebook. If you choose to use Visual Studio code:

1. Click the blocks icon on the left of VSCode to access its extension manager
2. Search for Markdown All In One extension and Install it

Markdown All In One will give us the ability to embed HTML in our Python notebooks that we open in VSCode. (By default HTML elements aren't rendered. This package adds this functionality.)

**Extending the Creature Class**

Our goal today is to modify the creature.py file to fill in the missing methods mentioned above. In order to do this, you will have to know how to not only write the code to do this but also test the code in your Python notebook or via Unit Tests.

Read through the documentation in the notebook and examine the \*.py module files. They will explain to you how to use the classes for this game to play it. This is why I like VSCode: one application that lets us quickly switch between files.

The documentation for each method is included in each python class. They should help you to understand how this game works.

Your goal is to replace the Creature methods in the file creature.py that are listed above with their correct counterparts.

The data attributes (and types) of Creature are:

* **init\_score (int):** the initial score of the creature
* **world\_size tuple(int,int):** the size of the world as a tuple (rows, columns)
* **start\_location tuple(int,int):** where in the grid the creature starts (row, column)
* **init\_facing (string):** where the creature faces
* **score (int):** the current score of the creature
* **current\_location tuple(int,int):** where the creature currently is on the grid
* **facing (string):** the direction (N, E, S, W) the creature is currently facing

You can see these in the \_\_init\_\_ method, which is called when a new instance of Creature is created. Using the attributes above and the documentation, I would like you to finish the methods listed at the start of this document.

**Testing Your Solution**

Testing is key to writing good code. You have to make sure that each step is correct in any data analysis project before you incorporate it into a larger solution. Garbage-In-Garbage-Out is quite real in analytics.

There are two ways to test your solution: by running the game play code in the notebook provided and examining the results and verifying what you wrote is correct by running the *unit tests* provided. We won't go into unit tests much in this course, but they are used extensively in software engineering to validate the correctness of code. In short, they are small snippets of code that we can use to test whether our methods are implemented correctly. If they are, the tests pass without throwing an Exception or filing an assertion. If they are incorrect, the tests fail.

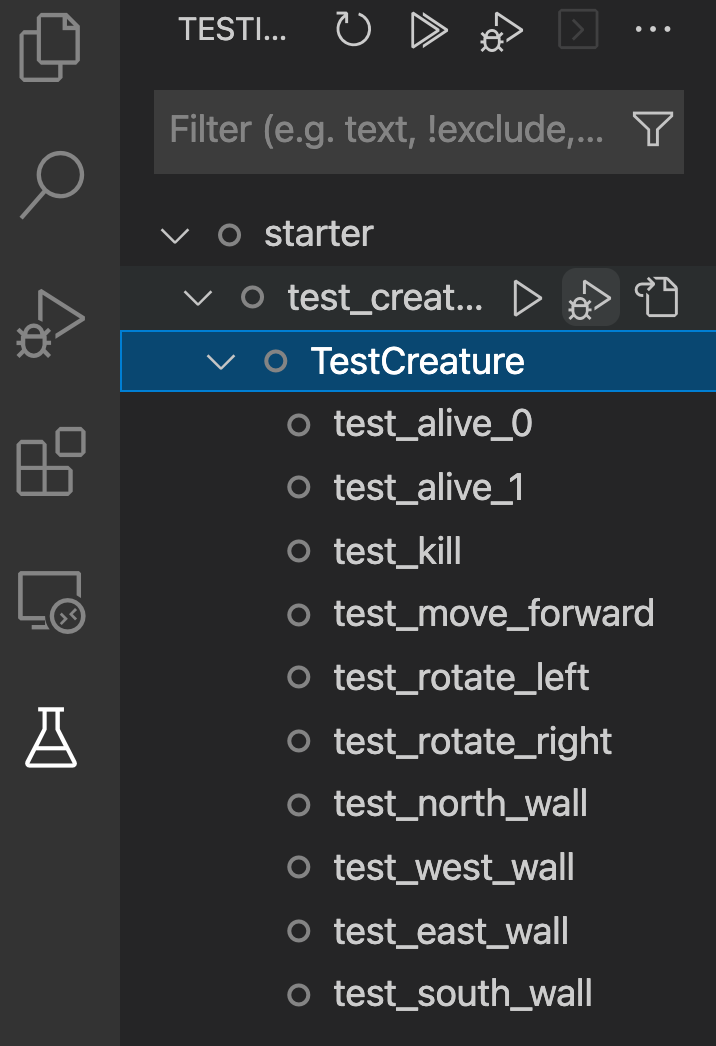
**Testing Your Code in the Notebook**

Using the examples provided, you can play the game to see if you get your rotate, movement, kill, is\_alive, and reset methods work by calling the play() method repeatedly and examining what the board looks like. You can add new cells below, call the methods you created, and display the board to see what the results of these method calls produce.

**Testing Your Code with Unit Tests in Visual Studio Code**

If you are using Visual Studio Code or PyCharm you can run the unit tests to validate your solution.

Here are the directions for Visual Studio Code. Make sure you have the labs/03 folder open.

1. Click on the beaker button on the left of the application.
2. Click on "Configure Python Tests"
3. Use the following settings when prompted:
   1. "Unittest" as the framework
   2. ". Root directory" as the directory to use
   3. "test\_\*.py" as the files to use
4. If you were successful, you will see a listing similar to this graphic. If you expand each arrow by clicking on it, you will get a listing of tests. These tests are in the file "test\_creature.py". You can examine them if you'd like to see what the code is doing.  
   
5. To run the tests, click the double arrows at the top of the listing. If the tests pass, you will see green checks. If they fail, you will see a red X or dot.
6. To debug any test, right click on it, and select "Debug Test". A console window will appear (if it doesn't, go to the menu View > Debug Console) to see what the error was. An annoying red box may appear in your code window with the same information. Feel free to X out of it. Read through the console's output to figure out why the test is failing.
7. You have completed this exercise when all tests pass.