# Mod 2 Lab: OOP and TDD

Use test driven development (TDD) to implement a class for 2D cartesian points.

The majority of the grade for this assignment is for test cases, not class functionality. Focus on using TDD to develop code, not on getting functionality as quickly as possible.

## Test-Driven Development

Test-driven development, or TDD, involves three stages often referred to as Red-Green-Refactor:

- Red - write a test, then run your code to **verify that the test case fails**
- Green - modify your code until the test passes
- Refactor - extract duplicate algorithms/classes into a parent function/superclass.

### 1) Red

Create a file called `Point.py` and add the following skeleton code:

```
class Point:
    def __init__(self, x, y):
        pass
```

Then, write a test for the `init` function in your `if __name__ == '__main__'` block.

```
if __name__ == '__main__':
    p1 = Point(3, 4)
    assert p1.x == 3
    assert p1.y == 4
```

Run `Point.py`, and you should see something like the following:

```
$ python3 .\Point.py
Traceback (most recent call last):
  File "Point.py", line 7, in <module>
    assert p1.x == 3
AttributeError: 'Point' object has no attribute 'x'
$
```

**Terminal** Our test fails - **it is a good test.** This assert statement checks for functionality we do not currently have. If it passes in the future, then we know it is because of changes we have made to the code.

### 2) Green

Modify your code until it passes the test case. You should see no output when you execute the script in terminal - `assert` statements produce no output when they pass.

```
class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y
```

```
$ python3 .\Point.py
$
```

### 3) Refactor

In this case, we have nothing to refactor - we have only written one function.

That's it! You have completed one round of TDD. The next step is to pick a new piece of functionality and repeat the process.

## class `Point`

Use TDD to implement the class `Point`.

You need test cases for all functionality implemented, and you should write the tests *before* implementing the functionality.

### Magic Methods

Make sure to use methods that begin and end with 2 underscores, like `__init__`, for the functionality below.

For some of these are functions we have purposefully omitted the appropriate "magic method" name. To access them, print out the directory of a similar type (e.g. `dir(list)`) and scan for the appropriate name, or just search for "magic method to do X in python". If you are having difficulty finding the correct magic method after looking for it yourself, feel free to ask in the Discord.

- `init` (already done)
  - initializes a new `Point` object with `x` and `y` coordinates.
- Comparators
  - Support the comparators `<`, `>`, and `==`
  - `p1` is less than `p2` if its distance from the origin is less
  - `p1` is greater than `p2` if its distance from the origin is greater
  - `p1` and `p2` are equal if they are the same distance from the origin
  - Write at least two test cases for each comparator, an expected true and an expected false. For example, both of the following are required to satisfactorily test the less than method:
    * `assert p1 < p2 # expected True`
    * `assert not (p2 < p1) # expected False`
- `str`
  - returns a string representation of a `Point`

```
>>> p1 = Point(3, 4)
>>> s = str(p1) # `str` should return a value, not print
>>> print(s)
Point(3, 4)
>>>
```

### Non-magic Methods

- `dist_from_origin`
  - returns the cartesian distance of this point from the origin
  - do not use the `math` module (it's faster, but we want you to practice coding without any other modules)

### Examples

Any examples below are intended to be illustrative, not exhaustive. Your code may have bugs even if it behaves as below. Write your own tests, and think carefully about edge cases.

```
>>> from Point import *
>>> p1 = Point(3, 4) # dist_from_origin() - 5
>>> p2 = Point(3, 4) # dist_from_origin() - 5
>>> p3 = Point(4, 3) # dist_from_origin() - 5
>>> p4 = Point(0, 1) # dist_from_origin() - 1
>>> p1 > p4 # expected True
True
>>> p4 > p1 # expected False
False
>>> p1 > p3 # same magnitude, different coordinates
False
>>> p1 == p3 # expected True
True
>>> str(p1)
'Point(3, 4)'
>>> p1.dist_from_origin()
5.0
```

## External Modules

Do not use any external modules (e.g. `math`, `collections`, `unittest`) on this assignment.

## Grading

Gradescope only looks at functionality here, but the goal of this lab is to get comfortable writing tests. Show your TA your tests as you go, and they'll give you feedback on whether they're good or not.

**To earn participation credit on this lab, you need to show your TA your full suite of test cases before leaving lab.**

## Submitting

At a minimum, submit the following file:

- `Point.py`

Students must submit **individually** by the due date (typically, Sunday at 11:59 pm EST) to receive credit.