# Mod 2 Lab: OOP and TDD

You will use TDD to implement a class for points in this assignment.

## Test-Driven Development

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

1. Write a test, then run your code to **verify that the test case fails**
2. Modify your code until the test passes
3. Refactor your code: extract duplicate algorithms/classes into a parent function/superclass.

The majority of the grade for this assignment is for your tests - focus on using TDD to develop code, rather than on trying to get functionality as soon as possible.

### 1) Red

Start by adding the following skeleton code to `Point.py`.

**`Point.py`**

```
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:

**Terminal**

```
$ python .\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'
$
```

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

Re-read the previous sentence until you parse it - it is subtle and important. You will be a better programmer when you learn to write good tests, and part of writing good tests is veryifying that they fail when you expect them to.

### 2) Green

Modify your code until it passes the test case.

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

```
$ python .\Point.py
$
```

### 3) Refactor

In this case, we have nothing to refactor - we have only written one function. Next, it's time to pick the next piece of funcitonality and start from the beginning.

## `Point` class

Use TDD to implement the class `Point`.

### Magic Methods

- `init`
  - initializes a new `Point` object with `x` and `y` coordinates.
- Comparators
  - Support the comparators `<`, `>`, and `==`
  - Two points are equal if they have the same `x` and `y` attributes
  - `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
  - if `p1` and `p2` are the same distance from the origin, but do not have the same `x` and `y` attributes, all three comparators should return `False`
  - Comparators should have at least three tests each
    - expected `True`
    - expected `False`
    - same magnitude, different coordinates
- `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

### 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)
>>> p2 = Point(3, 4)
>>> p3 = Point(4, 3)
>>> p4 = Point(0, 1)
>>> p1 > p4 # expected True
True
>>> p4 > p1 # expected False
False
>>> p1 > p3 # same magnitude, different coordinates
False
>>> 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.

## Submitting

At a minimum, submit the following file:

- `Point.py`

Students must submit to Mimir **individually** by the due date (typically, two days after lab at 11:59 pm EST) to receive credit.

## Grading

Most of your grade for this assignment will be for your tests. These will be manually graded after the due date.

We have hidden some of the code used for the functionality tests in Mimir because most of your grade is for writing tests.

- 25 - Functionality in `Point.py`
  - 5 - `eq`
  - 5 - `lt`
  - 5 - `gt`
  - 5 - `str`
  - 5 - `dist_from_origin`
- 75 - Tests (manually graded)
  - 15 - `eq`
  - 15 - `lt`
  - 15 - `gt`
  - 15 - `str`
  - 15 - `dist_from_origin`

## Feedback

If you have any feedback on this assignment, please leave it here.

We check this feedback regularly. It has resulted in:

- A simplified, clear **Submitting** section on all assignments
- A simplified, clear **Grading** section on all assignments
- Clearer instructions on several assignments (particularly in the recursion module)