```
%matplotlib inline
import matplotlib
import seaborn as sns
matplotlib.rcParams['savefig.dpi'] = 144
```

```
from static_grader import grader
```

Object-oriented exercises

Introduction

The objective of these exercises is to develop your familiarity with Python's `class` syntax and object-oriented programming. By deepening our understanding of Python objects, we will be better prepared to work with complex data structures and machine learning models. We will develop a `Point` class capable of handling some simple linear algebra operations in 2D.

Exercise 1: `point_repr`

The first step in defining most classes is to define their `__init__` and `__repr__` methods so that we can construct and represent distinct objects of that class. Our `Point` class should accept two arguments, `x` and `y`, and be represented by a string `'Point(x, y)'` with appropriate values for `x` and `y`.

When you've written a `Point` class capable of this, execute the cell with `grader.score` for this question (do not edit that cell; you only need to modify the `Point` class).

```
class Point(object):

    def __init__(self, x, y):
        self.x = x
        self.y = y

    def __repr__(self):
        return "Point({}, {})".format(self.x, self.y)
```

```
grader.score.vc__point_repr(lambda points: [str(Point(*point)) for point in points])
```

```
==================
Your score:  1.0
==================
```

Exercise 2: add_subtract

The most basic vector operations we want our Point object to handle are addition and subtraction. For two points $(x_1, y_1) + (x_2, y_2) = (x_1 + x_2, y_1 + y_2)$ and similarly for subtraction. Implement a method within Point that allows two Point objects to be added together using the + operator, and likewise for subtraction. Once this is done, execute the grader.score cell for this question (do not edit that cell; you only need to modify the Point class.)

(Remember that __add__ and __sub__ methods will allow us to use the + and - operators.)

```
class Point(object):

    def __init__(self, x, y):
        self.x = x
        self.y = y

    def __repr__(self):
        return "Point(%d, %d)" % (self.x, self.y)

    def __add__(self, point):
        return Point(self.x + point.x, self.y + point.y)

    def __sub__(self, point):
        return Point(self.x - point.x, self.y - point.y)
```

```
from functools import reduce
def add_sub_results(points):
    points = [Point(*point) for point in points]
    return [str(reduce(lambda x, y: x + y, points)),
            str(reduce(lambda x, y: x - y, points))]

grader.score.vc__add_subtract(add_sub_results)
```

```
==================
Your score:  1.0
==================
```

Exercise 3: multiplication

Within linear algebra there's many different kinds of multiplication: scalar multiplication, inner product, cross product, and matrix product. We're going to implement scalar multiplication and the inner product.

We can define scalar multiplication given a point P and a scalar a as

$$aP = a(x, y) = (ax, ay)$$

and we can define the inner product for points P, Q as

$$P \cdot Q = (x_1, y_1) \cdot (x_2, y_2) = x_1 x_2 + y_1 y_2$$

To test that you've implemented this correctly, compute $2(x, y) \cdot (x, y)$ for a `Point` object. Once this is done, execute the `grader.score` cell for this question (do not edit that cell; you only need to modify the `Point` class.)

(Remember that `__mul__` method will allow us to use the `*` operator. Also don't forget that the ordering of operands matters when implementing these operators.)

```
class Point(object):

    def __init__(self, x, y):
        self.x = x
        self.y = y

    def __repr__(self):
        return "Point(%d, %d)" % (self.x, self.y)

    def __add__(self, point):
        return Point(self.x + point.x, self.y + point.y)

    def __sub__(self, point):
        return Point(self.x - point.x, self.y - point.y)

    def __mul__(self, elem):
        if isinstance(elem, int):
            return Point(self.x * elem, self.y * elem)
        elif isinstance(elem, Point):
            return self.x * elem.x + self.y * elem.y
        else:
            raise TypeError('Expected number to be int or Point. Got %s' %
type(elem))
```

```
def mult_result(points):
    points = [Point(*point) for point in points]
    return [point*point*2 for point in points]

grader.score.vc__multiplication(mult_result)
```

```
==================
Your score:  1.0
==================
```

Exercise 4: Distance

Another quantity we might want to compute is the distance between two points. This is generally given for points $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$ as

$$D = |P_2 - P_1| = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}.$$

Implement a method called distance which finds the distance from a point to another point.

Once this is done, execute the grader.score cell for this question (do not edit that cell; you only need to modify the Point class.)

Hint

- *You can use the sqrt function from the math package.*

```
from math import sqrt

class Point(object):

    def __init__(self, x, y):
        self.x = x
        self.y = y

    def __repr__(self):
        return "Point(%d, %d)" % (self.x, self.y)

    def __add__(self, point):
        return Point(self.x + point.x, self.y + point.y)

    def __sub__(self, point):
        return Point(self.x - point.x, self.y - point.y)

    def __floordiv__(self, num):
        return Point(self.x // num, self.y // num)

    def distance(self, point):
        return sqrt((self.x - point.x) ** 2 + (self.y - point.y) ** 2)
```

```
def dist_result(points):
    points = [Point(*point) for point in points]
    return [points[0].distance(point) for point in points]

grader.score.vc__distance(dist_result)
```

```
==================
Your score:  1.0
==================
```

Exercise 5: Algorithm

Now we will use these points to solve a real world problem! We can use our Point objects to represent measurements of two different quantities (e.g. a company's stock price and volume). One thing we might want to do with a data set is to separate the points into groups of similar points. Here we will implement an iterative algorithm to do this which will be a specific case of the very general k-means clustering algorithm. The algorithm will require us to keep track of two clusters, each of which have a list of points and a center (which is another point, not necessarily one of the points we are clustering). After making an initial guess at the center of the two clusters, C_1 and C_2, the steps proceed as follows

1. Assign each point to C_1 or C_2 based on whether the point is closer to the center of C_1 or C_2.
2. Recalculate the center of C_1 and C_2 based on the contained points.

See reference for more information.

This algorithm will terminate in general when the assignments no longer change. For this question, we would like you to initialize one cluster at `(1, 0)` and the other at `(-1, 0)`.

The returned values should be the two centers of the clusters ordered by greatest `x` value. Please return these as a list of numeric tuples $[(x_1, y_1), (x_2, y_2)]$

In order to accomplish this we will create a class called cluster which has two methods besides `__init__` which you will need to write. The first method `update` will update the center of the Cluster given the points contained in the attribute `points`. Remember, you after updating the center of the cluster, you will want to reassign the points and thus remove previous assignments. The other method `add_point` will add a point to the `points` attribute.

Once this is done, execute the `grader.score` cell for this question (do not edit that cell; you only need to modify the `Cluster` class and `compute_result` function.)

```python
class Cluster(object):
    def __init__(self, x, y):
        self.center = Point(x, y)
        self.points = []

    def update(self):
        sum = Point(0, 0)
        for point in self.points:
            sum += point

        n = len(self.points)
        self.center = Point(sum.x / n, sum.y / n)
        self.points.clear()

    def add_point(self, point):
        self.points.append(point)
```

```python
def compute_result(points):
    points = [Point(*point) for point in points]
    a = Cluster(1,0)
    b = Cluster(-1,0)
    a_old = []
    for _ in range(10000): # max iterations
        for point in points:
            if point.distance(a.center) < point.distance(b.center):
                # add the right point
                a.add_point(point)
            else:
                # add the right point
                b.add_point(point)
        if a_old == a.points:
            break
        a_old = a.points[:]
        a.update()
        b.update()
    return [(a.center.x, a.center.y), (b.center.x, b.center.y)] # [(x, y)] * 2
```

```
grader.score.vc__k_means(compute_result)
```

```
==================
Your score:  1.0
==================
```