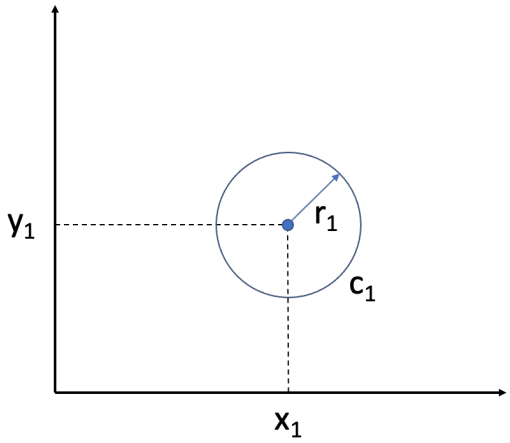
1: Point Class (with Tutor)

Create a new C# project for defining and testing the class *Point* described during the lecture. The project should include the usual file *Program.cs*, created as part of the project template, with the definition of the *Program* class and the *Main* program entry point. The *Point* class should be defined in a separate file *Point.cs*, which needs to be added to the project. The *Main* will contain the instructions to create two points (p1 = (5, 1) and p2 = (7, 2)) and to print their coordinates on the screen.

2: Circle class (with Tutor)

Think about the possible design of a class to represent circles, as it was briefly discussed in the lecture.



For instance:

```
Point p1 = new Point(6, 4);
double r1 = 1.0;

Circle c1 = new Circle(p1, r1)
```

A *Circle* object should include a `Display()` method that prints the coordinates of its centre and the value of the radius. Remember that there exists a relationship between *Circle* and *Point* objects. As such, the above object *c1* can "send a message" to the object *p1* to request to display the coordinates of that point—this can be done without *c1* having to access *p1*'s attributes directly (hint: the `Display()` method of the *Circle* can invoke the `Display()` method of the *Point* object representing its centre).

Moreover, the *Circle* class should include two additional methods for calculating and printing a circle's *circumference* and *area*.

The *Circle* class should be added to the project of exercise 1, and at least two *Circle* objects should be instantiated in the *Main entry point* created (in the *Program* class). Read the coordinates of the centre and the radius of each circle as input from the keyboard. Show the features of the created *Circle* objects, i.e., their *circumference* and *area,* via invoking the above-mentioned methods.

3: Distance between two Points (independent work)

Modify the definition of the *Point* class (of exercise 1) to include the following method:

```
public void DistanceFrom(Point p2)
```

The method should print on the screen the distance between the point on which it is invoked and the *Point* object passed as the argument.
The formula to calculate the distance between two points (x_1, y_1) and (x_2, y_2) in a two-dimensional plane is:

$$\sqrt{[(x_2 - x_1)^2 + (y_2 - y_1)^2]}$$

To implement the formula, use the methods `Math.Sqrt` and `Math.Pow`. To access an attribute of a class, use the same *dot* notation we have been using for the methods. Use the following code as a starting point and replace the green text with the proper attributes of the objects.

```
public void DistanceFrom(Point p2)
{
    double distance = Math.Sqrt(Math.Pow(replace: x2 – x1, 2) +
                      Math.Pow(replace: y2 – y1, 2));

    Console.WriteLine("Distance is: " + distance);
}
```

Are you able to access the x and y coordinates of the p2 object using *p2.x* and *p2.y* from the body of the `DistanceFrom` method? Why?
Test the `DistanceFrom(Point p2)` method from the Main entry point of the *Program* class as in the previous exercises.

4: Segment Class (independent work)

Try to design a class that models segment objects, e.g.:

```
Point p1 = new Point(2, 3);
Point p2 = new Point(3, 4);

Segment s1 = new Segment(p1, p2)
```

The class should include a method `Length()` that prints the length of a *Segment* object. As in the previous exercise, use the dot notation to access an attribute of a *Point* object. Can you still access the x and y coordinates of the p2 object using *p2.x* and *p2.y*? To solve the problem, consider the `DistanceFrom` method defined in the *Point* class in the previous exercise. Can it be used to calculate the length of a segment?