COMP1021
Introduction to Computer Science

Objects

David Rossiter, Leo Tsui and Gibson Lam

Outcomes

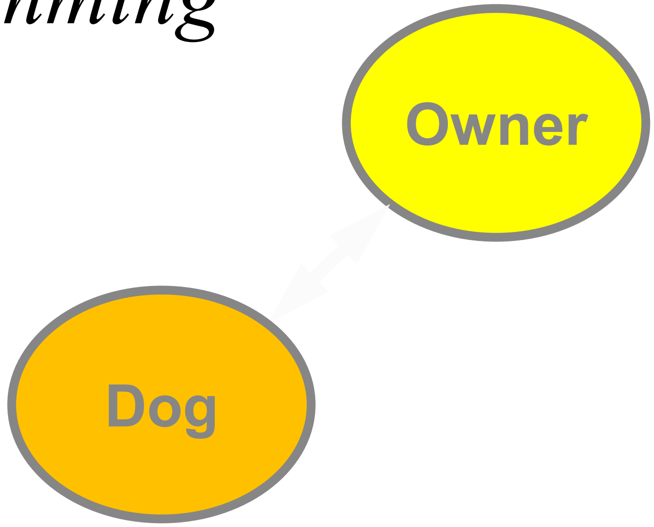
- After completing this presentation, you are expected to be able to:
    1. Explain briefly what object-oriented programming is
    2. Create and use simple Python classes

Introduction to Objects

- There are many ‘objects’ around us in the real world, e.g. a dog and a car are both objects
- We can say that each object has two kinds of characteristics: *attributes* and *behaviours*
- For example, a dog has:
  - *attributes* such as name, colour and weight
  - *behaviours* such as eating, barking and running

Object-Oriented Programming

- We are dealing with ‘objects’ every day
- It would be great if we can ask a program to ‘think’ using objects too
- This way of programming, thinking using objects, is called *object-oriented programming*
- To do that we first design the objects and then use the objects to interact with each other



What is a Class?

- In Computer Science the definition/ the design of an object is called a *class*
- A *class* is not actually an object by itself
- You need to create an *instance* of the class; the result is an object
- In a program you can create as many instances of the class as you want - in other words, you can make as many objects as you like

An Example of Using a Class 1/2

- Let’s say we have created a Dog class
- In order to make Snoopy and Odie we need to create an instance of the Dog class for each of them, like this:

A Dog Class

Make an instance of the class

Name: Snoopy

Colour: White

Weight: 30kg

Make an instance of the class

Name: Odie

Colour: Yellow

Weight: 25kg

- *Here there are two instances of the Dog class*
- *In other words, there are two objects*

An Example of Using a Class 2/2

- Both Snoopy and Odie have been created using the same class, the Dog class

Name: Snoopy

Colour: White

Weight: 30kg

- They are different because they have different attribute values, such as their name, colour and weight

Name: Odie

Colour: Yellow

Weight: 25kg

Creating Python Classes

- You create a class in Python using `class`
- For example, the `Dog` class can be created like this:

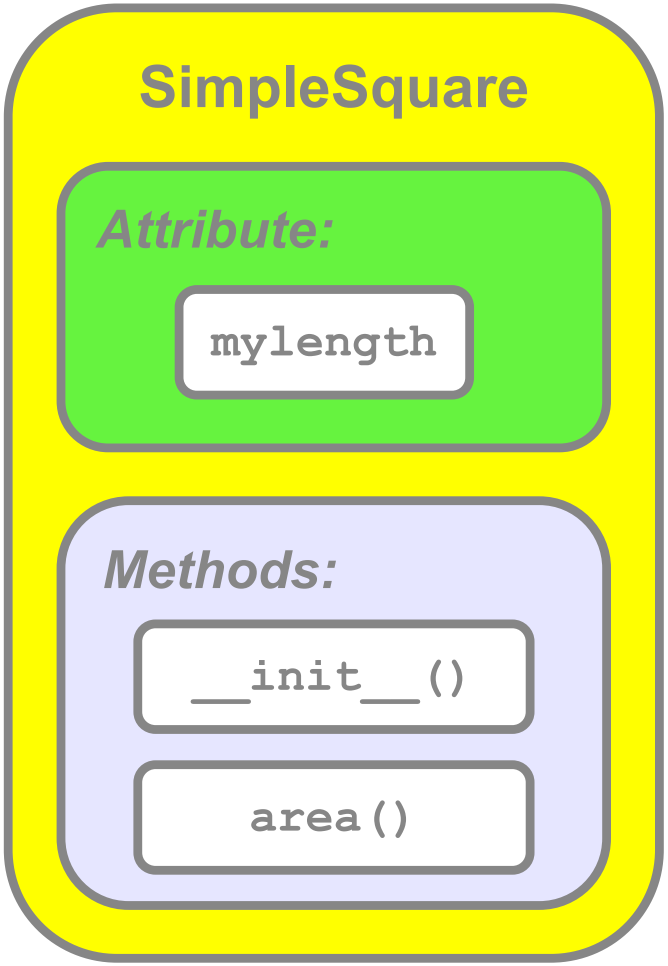
```
class Dog:
        ...  content of the class  ...
```

The Python code in a class must be indented

- Inside the class you can have:
  - **attributes** which are Python variables
  - **behaviours** which are Python functions
- In the world of computer programming functions inside a class are usually called *methods*

Creating a SimpleSquare Class

- Let’s create our own class
- In the following example we create a class which we will call *SimpleSquare*, which has:
  - a `mylength` attribute, which contains the width/height
  - an `__init__()` method, which gives the instance of the class some initial values
  - an `area()` method, which calculates the area of the square



The SimpleSquare Class

- Here is the complete code of the `SimpleSquare` class:

```
class SimpleSquare:
    def __init__(self, length):
        self.mylength = length

    def area(self):
        return self.mylength * self.mylength
```

The name of the class

There is one variable stored in this class

There are two functions in this class

The SimpleSquare Class

- Here is the complete code of the `SimpleSquare` class:

```
class SimpleSquare:
    def __init__(self, length):
        self.mylength = length

    def area(self):
        return self.mylength \
            * self.mylength
```

The first parameter of a function in a class always has to be `self`*, which means* ***itself/myself*** *(meaning the instance of the class)*

- We will explain this class in the next few slides

The Constructor

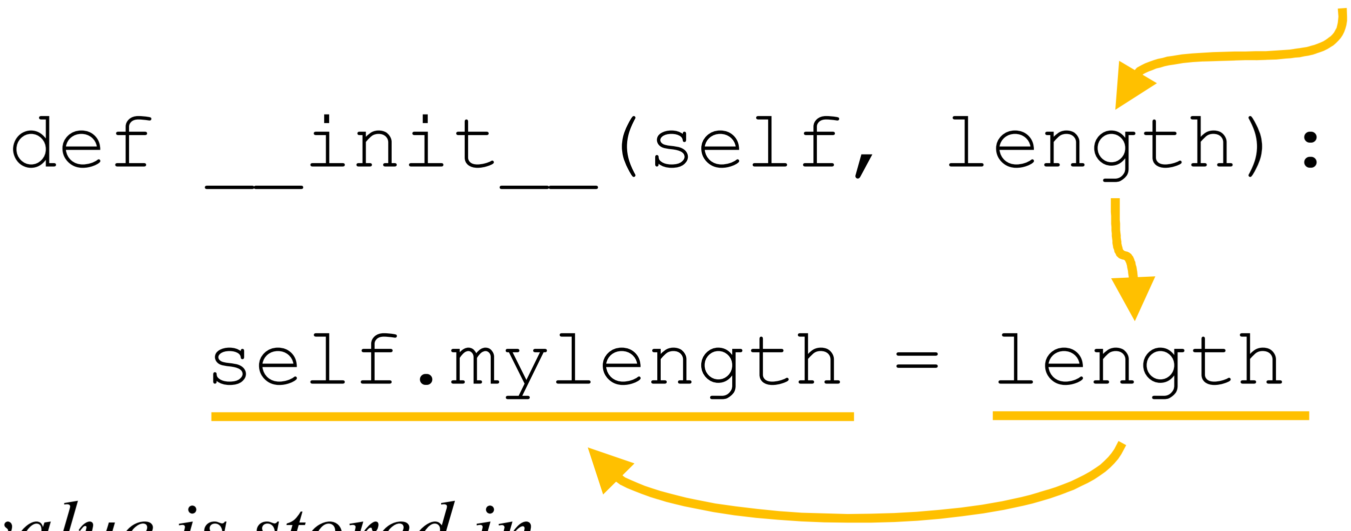
```
def __init__(self, length):
    self.mylength = length
```

- The `__init__` function is called the *constructor*
- The constructor function is automatically executed when a new instance of the class is created
- In the class, the word `self` has to be included as the first parameter of every method; it means the current instance of the class (in other words, it means the object)

Creating the Attributes

- The attributes of a class are created and initialized in the constructor function
- For example, here the `mylength` attribute is created:

A value is passed to the function



```
def __init__(self, length):
    self.mylength = length
```

The value is stored in the mylength attribute

The area() Method

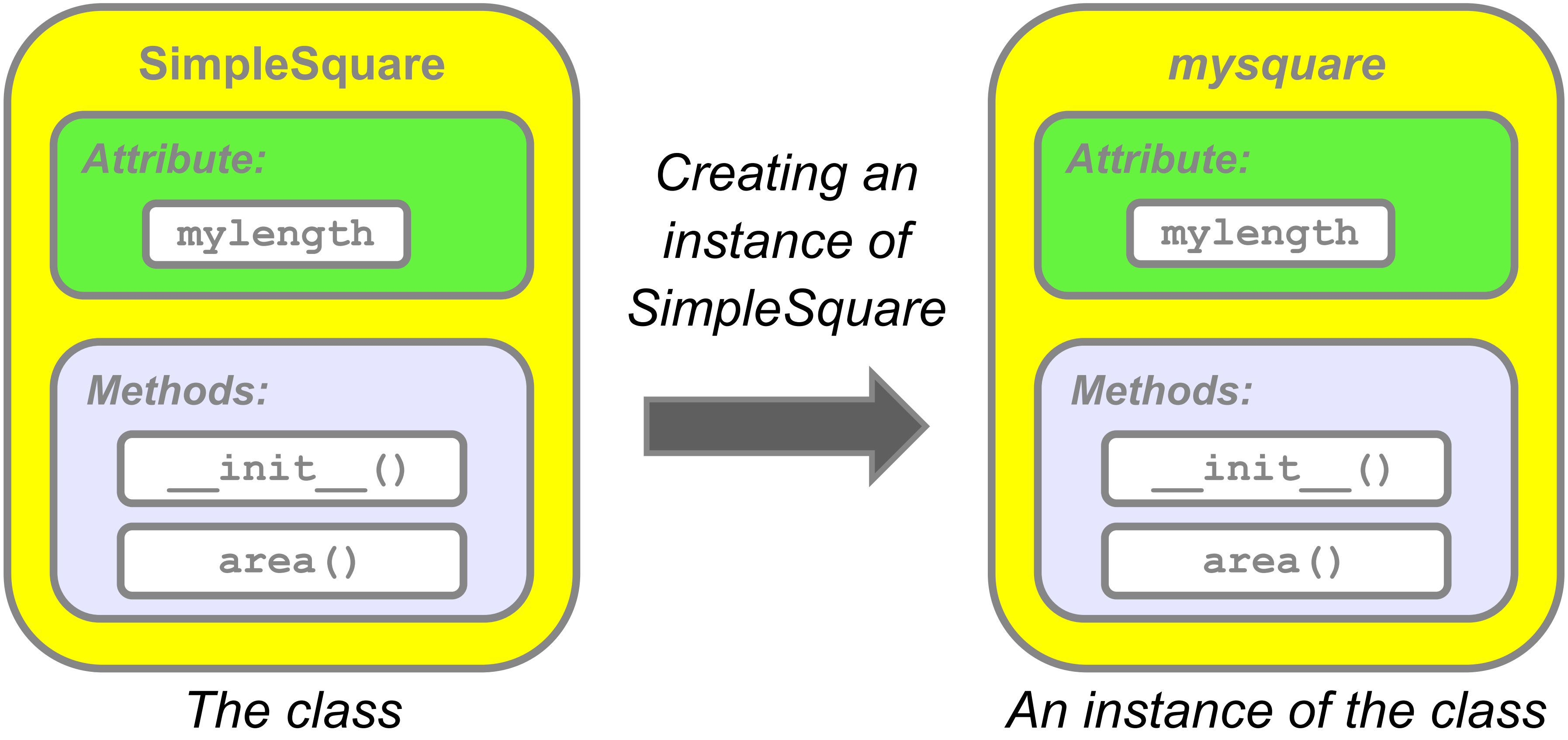
```
def area(self):
    return self.mylength * self.mylength
```

`mylength` *is one of the attributes of the class*

- The `area()` method calculates and returns the area of the instance of the `SimpleSquare` class
- Remember `self.mylength` is an attribute of the class, which was created in the constructor

A SimpleSquare Instance

- After we have created the `SimpleSquare` class we can create an instance of it, and call it `mysquare`
- This means `mysquare` also has one attribute and two methods



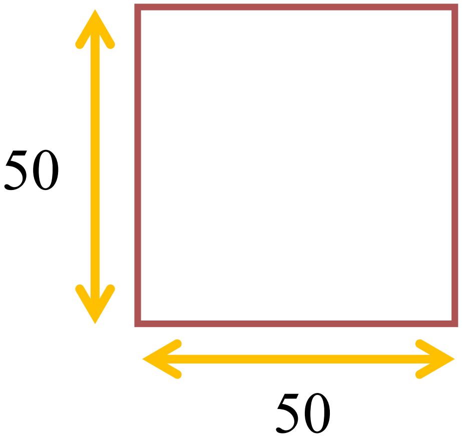
Creating a Class Instance

- So at this point we have defined a class
- Now we can use it as many times as we like
- For example, we can create a `SimpleSquare` object and call it `mysquare`, like this:

```
mysquare = SimpleSquare(50)
```

This value is the input parameter of the constructor function `__init__()`

You don't pass something for the first parameter, `self` *- that is invisibly handled by Python*



Using Class Attributes and Methods

This tells Python the code continues on the following line

- You can use the `mylength` attribute of `mysquare`, like this:

```
print("Length of the square is", \
      mysquare.mylength)
```

- Similarly you can use the `area()` method like this:

```
print("Area of the square is", \
      mysquare.area())
```

- As you can see, you put `mysquare.` in front of the attributes and methods you want to use which are inside the instance (in other words, inside the object)

The self Parameter

- Here is the definition of the `area()` method:

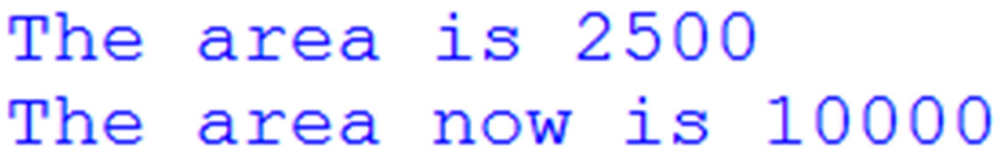
```
def area(self):
    return self.mylength * self.mylength
```

- In the example on the previous slide we use `mysquare.area()` to execute the method
- You can see that you don't explicitly pass a value for the `self` parameter
- That parameter is invisibly handled by Python

Example of Using the Class

- The result of the code is shown below

```
mysquare = SimpleSquare(50)
print("The area is", mysquare.area() )

mysquare.mylength = 100
print("The area now is", mysquare.area() )
```

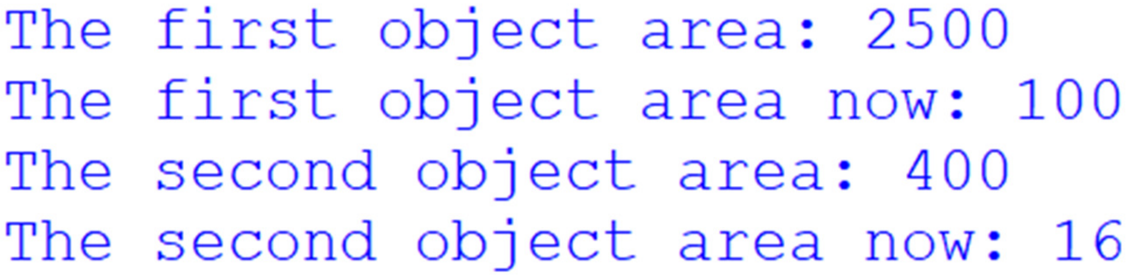


```
The area is 2500
The area now is 10000
```

Two Instances

- We could easily use two instances:

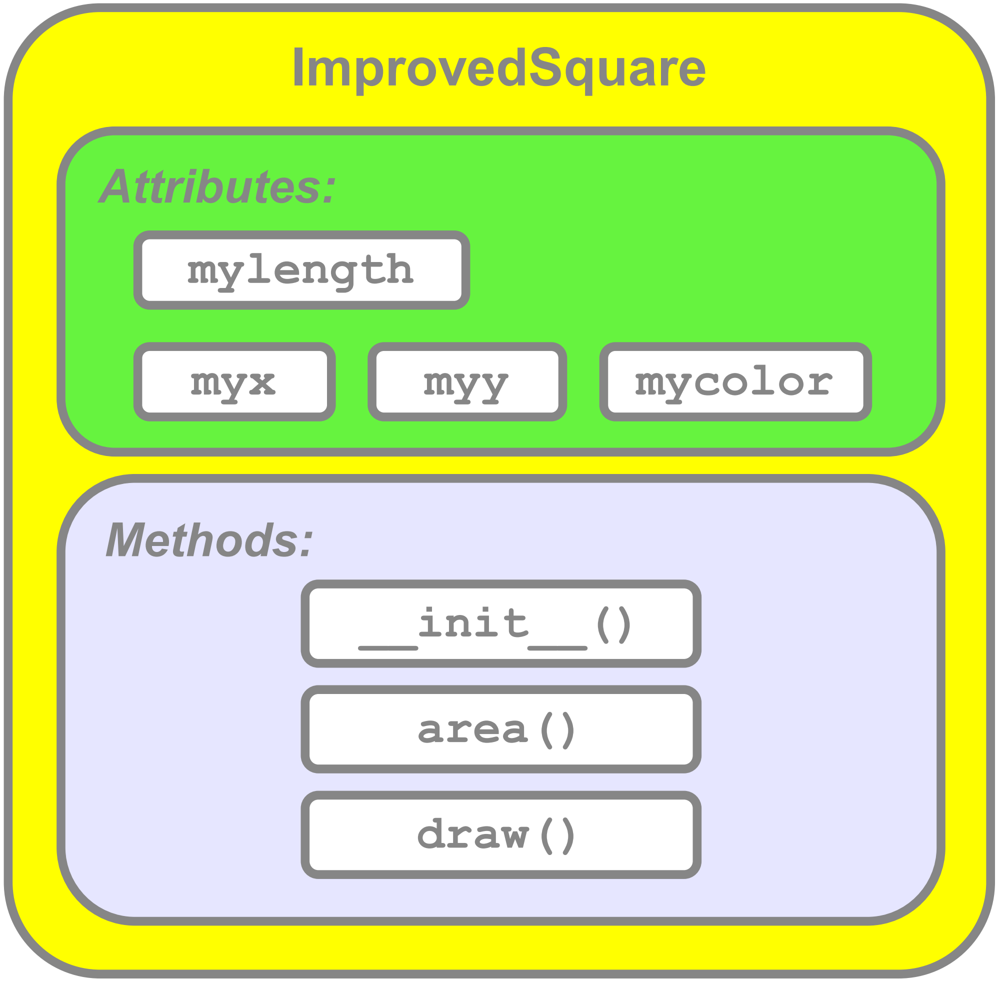
```
mysquare = SimpleSquare(50)
print("The first object area:", mysquare.area() )
mysquare.mylength = 10
print("The first object area now:", mysquare.area() )
mysquare2 = SimpleSquare(20)
print("The second object area:", mysquare2.area() )
mysquare2.mylength = 4
print("The second object area now:", mysquare2.area() )
```



```
The first object area: 2500
The first object area now: 100
The second object area: 400
The second object area now: 16
```

An Improved Class

- The SimpleSquare class can’t do anything except return its area
- It would be nice if we could see the square
- We will add the ability to draw the square to the class



ImprovedSquare

Attributes:

mylength

myx

myy

mycolor

Methods:

`__init__()`

`area()`

`draw()`

We have added three more attributes: `myx`, `myy` *and* `mycolor`

We have added one more method: `draw()`

The ImprovedSquare Class 1/2

```
class ImprovedSquare:
    def __init__(self, x, y, length, color):
        self.myx = x
        self.myy = y
        self.mylength = length
        self.mycolor = color

    def area(self):
        return self.mylength * self.mylength
```

- This method is automatically called when the instance is created

- This method returns the area of the square

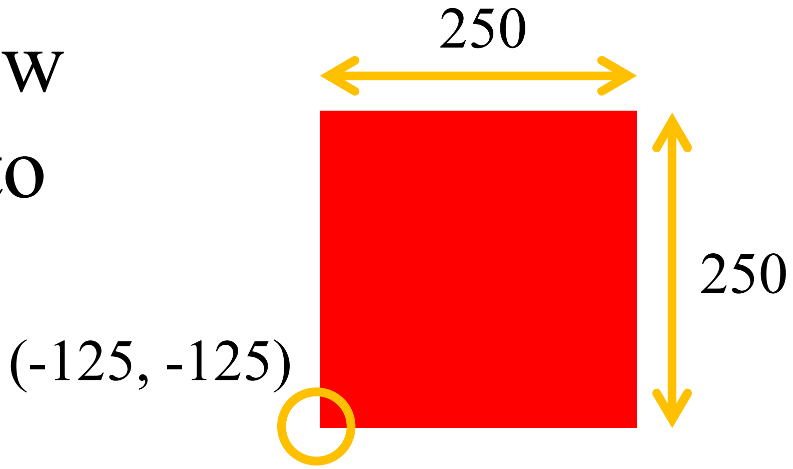
The ImprovedSquare Class 2/2

```
def draw(self):
    turtle.up()
    turtle.goto(self.myx, self.myy)
    turtle.down()
    turtle.fillcolor(self.mycolor)
    turtle.begin_fill()

    for _ in range(4):
        turtle.forward(self.mylength)
        turtle.left(90)

    turtle.end_fill()
```

- This method draws the square

Using the ImprovedSquare Class

- The Python code shown below creates an object and tells it to draw itself. This red square is then drawn:



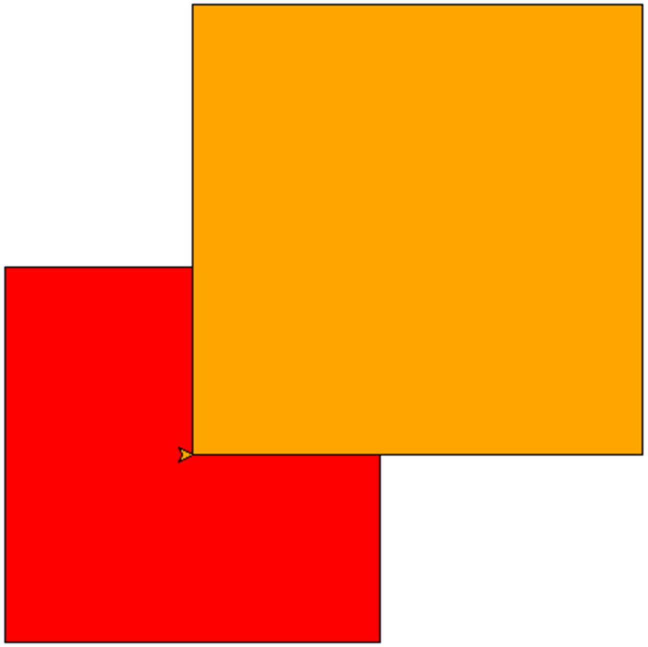
```
' Here we put the square at position (-125, -125)
' and set the size as 250 * 250, using red color

mysquare = ImprovedSquare(-125, -125, 250, "red")

mysquare.draw()
```

x *y* *length* *color*

Two Instances

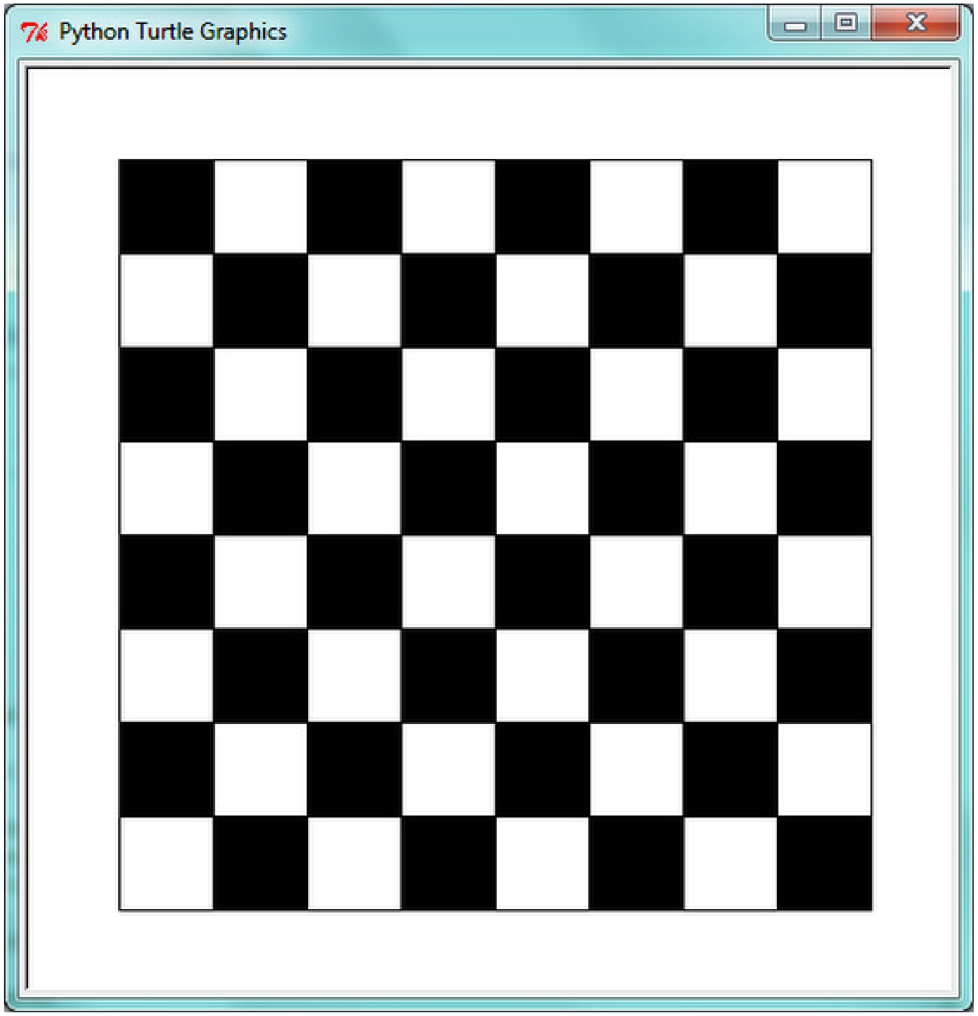
- We could easily use two instances:

```
mysquare = ImprovedSquare(-125, -125, 250, "red")

mysquare.draw()

mysquare2 = ImprovedSquare(0, 0, 300, "orange")

mysquare2.draw()
```



Generating a Chess Board

- In the next example we will generate a chess board using the `ImprovedSquare` class
- The chess board structure is 8 * 8



Using a Nested Loop

- The example uses a nested loop to create the 64 square objects which together make the chess board
  - An if statement is used to determine whether to use black or white for the square colour
  - The squares are then added to a Python list

- After creating the squares another for loop is used to tell each square in the list to draw itself

Generating a Chess Board Code 1/3

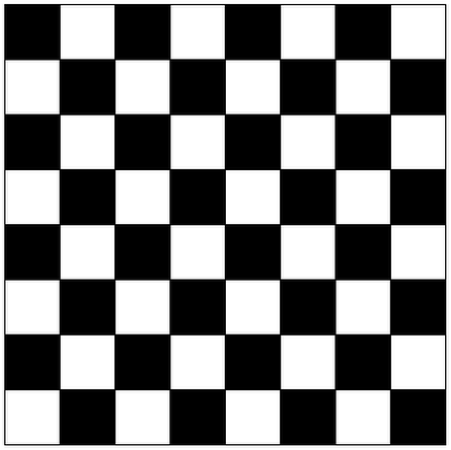
- Here is the start of the program

```
turtle.setup(500, 500)
turtle.hideturtle()
turtle.tracer(False)

side = 50 # square width/height

allsquares = []
```

A list will be used to store the square objects that will be created in the next part of the code

Generating a Chess Board Code 2/3

- Here is the main part of the program



```
for row in range(8):
    for column in range(8):
        if row % 2 == column % 2:
            thiscolor = "white"
        else:
            thiscolor = "black"

        x = row * side - 4 * side
        y = column * side - 4 * side

        square = ImprovedSquare(x, y, side, \
                                thiscolor)
        allsquares.append(square)
```

A square object is created and added to the list using the appropriate attributes

Generating a Chess Board Code 3/3

- Here is the final code
- It tells all the 64 square objects to draw themselves

```
for square in allsquares:
    square.draw()

turtle.tracer(True)
turtle.done()
```