COMP1021
Introduction to Computer Science

# Objects

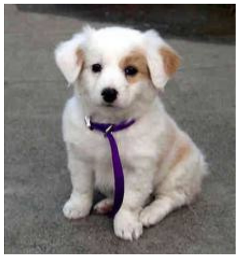
David Rossiter, Leo Tsui and Gibson Lam

## 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

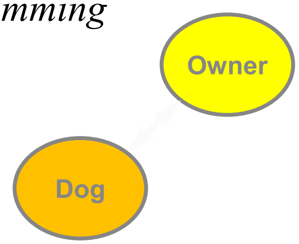
## Introduction to Objects

- 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

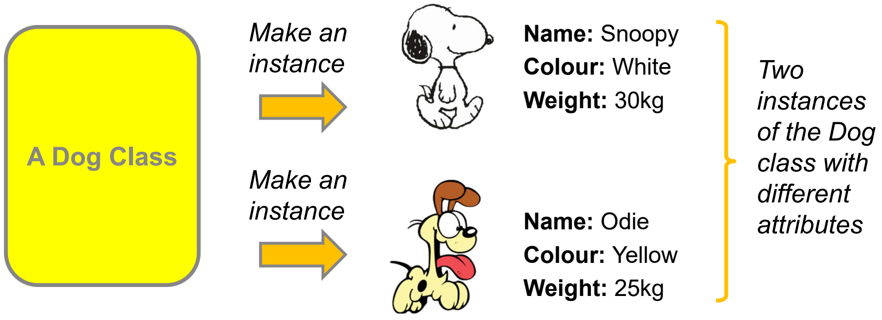


# What is a Class?

- In Computer Science we usually call the definition of an object a *Class*
- A class is only a definition of the object it represents
- When you want to create an object you need to make an *instance* of the class
- In a program you can create as many instances of the class as you want

# 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:



**Name:** Snoopy
**Colour:** White
**Weight:** 30kg

**Name:** Odie
**Colour:** Yellow
**Weight:** 25kg

*Two instances of the Dog class with different attributes*

# An Example of Using a Class 2/2

- Both Snoopy and Odie are dogs and therefore they are created using the same class, the Dog class
- They are different to each other because they have different attribute values, such as their name, colour and weight

**Name:** Snoopy
**Colour:** White
**Weight:** 30kg

**Name:** Odie
**Colour:** Yellow
**Weight:** 25kg

# Creating Python Classes

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

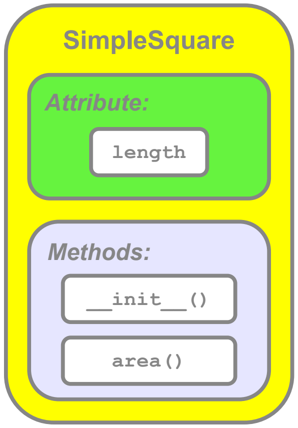
```
class Dog:
    ... Content of the class ...
```

*Content of the class is indented*

- Inside the class you can have:
  - attributes which are Python variables
  - behaviours which are Python functions
- Functions inside a class are typically called *methods* in computer programming

# 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 `length` attribute
  - an `__init__()` method, which gives the instance of the class some initial values
  - an `area()` method, which calculates the area of the square



SimpleSquare

Attribute:

length

Methods:

__init__()

area()

# The SimpleSquare Class

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

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

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

*The name of the class*

- We will explain the class in the next few slides

# The Constructor

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

- The `__init__` function is called the *constructor*
- The constructor is automatically called when one instance of the class is created
- The `self` parameter is required for every method of the class; the parameter represents the current instance of the class

# Creating the Attributes

- The attributes of a class are created and initialized in the constructor
- For example, the `SimpleSquare` class creates a `length` attribute in its constructor:

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

`self.length` *is an attribute called length from the class itself*

`length` *is the input parameter of the constructor in this example*

## The area() Method

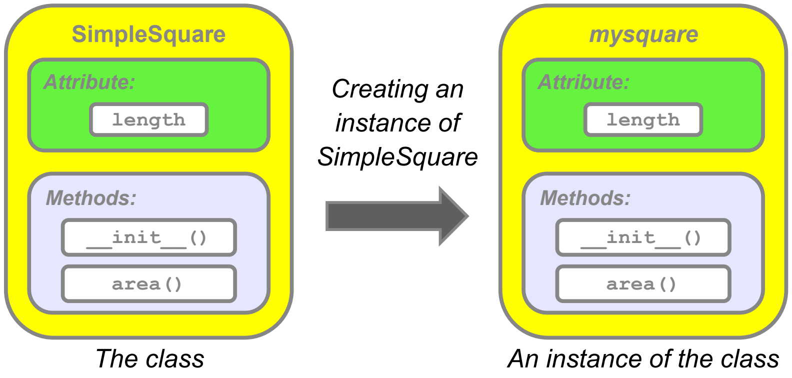
```
def area(self):
    return self.length * self.length
```

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

- The `area()` method simply returns the area of the current instance of the `SimpleSquare` class
- Remember `self.length` is the attribute of the class, which has been 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`
- So that means `mysquare` also has one attribute and two methods

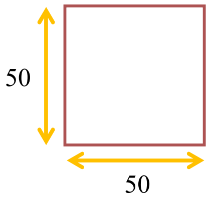


## Creating a Class Instance

- 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, which we will call `mysquare`, like this:

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

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



## Using Class Attributes and Methods

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

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

- 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

# The self Parameter

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

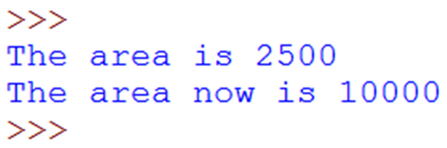
```
def area(self):
    return self.length * self.length
```

- In the example on the previous slide, we use `mysquare.area()` to run the method
- You can see that you don’t need to explicitly give a value to the `self` parameter
- The parameter is automatically given to the methods as the current instance of the class

# Example of Using the Class

- Here is another example of using the class:

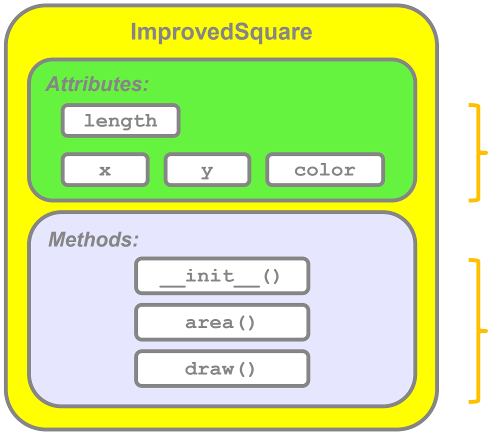
```
mysquare = SimpleSquare(50)
print("The area is", mysquare.area())

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



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

# An Improved Class

- The SimpleSquare class can’t do anything except return its area
- To add the ability to actually draw the square, we can improve the class like this:



ImprovedSquare

Attributes:

length

x y color

Methods:

`__init__()`

`area()`

`draw()`

*We have added three more attributes:* `x`, `y` *and* `color`

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

# The ImprovedSquare Class 1/2

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

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

- This method is called when the instance is created
- This method returns the area of the square

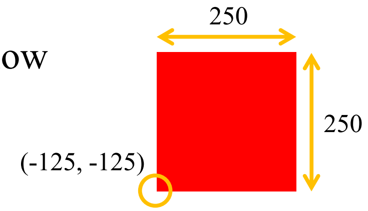
# The ImprovedSquare Class 2/2

```
def draw(self):
    turtle.up()
    turtle.goto(self.x, self.y)
    turtle.down()
    turtle.fillcolor(self.color)
    turtle.begin_fill()

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

    turtle.end_fill()
```

- This method draws the square

# Using the ImprovedSquare Class

- The Python code shown below will create this red square:

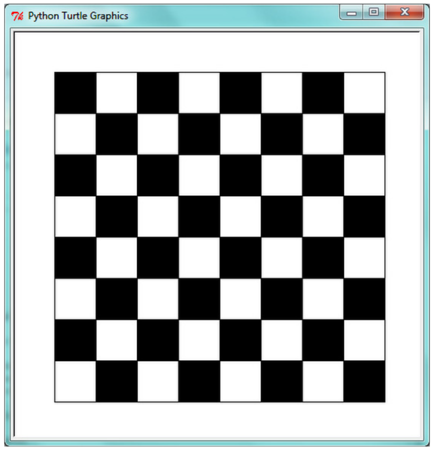


```
' Here we put the square at position (-125, -125)
' and set the size as 250 * 250, using red color

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

mysquare.draw()
```

*x* *y* *length* *color*

# Generating a Chess Board

- In the next example, we will generate a chess board using the `ImprovedSquare` class
- The chess board structure is 8 cells by 8 cells, like this:



# Using a Nested Loop

- The example first uses a nested loop to create the cells, i.e. the squares, inside 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 draw all the squares from the list

# Generating a Chess Board Code 1/3

- Here is the main part of the program:

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

side = 50
allsquares = []
```

*A list is used to store the squares to be created in the next part of the code*

# Generating a Chess Board Code 2/3

- Here is the nested loop:

```
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 is created and added to the list using the appropriate attributes*

# Generating a Chess Board Code 3/3

- Here is the code to draw all the squares:

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

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