Computing Induction

Welcome to SJWMS computing.

The purpose of these lessons – one on programming theory (OOP) and the other about 2D arrays in Python – are to give you a taste of what computing A level is all about. The combination of coding and theory.

Hopefully you enjoy both and if you have any problems please email:
michael.sandman@sjwms.org.uk

For lesson 3 (python) you can download python from their website or use an online emulator.

Lesson 2:
Object oriented Programming (OOP)

Classes, objects, instantiation, attributes and methods

Learning objectives

- Explain what object oriented programming is.
- Know what the following are:
  - Class
  - Object
  - Instantiation
- Know what methods and attributes are.
- Write simple code in OOP that uses classes, objects and instantiation

Object oriented programming (OOP)

- In OOP we use objects which refer to nouns. For instance, person, animal or place.
- Objects model the real world more closely by handling data and procedures together.
- Objects are responsible for their own data
- Objects have characteristics
- Objects have behaviour

Why use object oriented programming?

- OOP is based on two principles:
  - Duplicate code is bad
  - Code will always be changed
- Can modify and maintain existing code much more easily
- Reuse of code in other programs through libraries
- Has better structure and design so is suited to big projects
- Clear modular structure with a defined interface and which is able to abstract and hide away details

Classes

- Classes are the integral component of object oriented programs.
- A class is the blueprint or template for the structure of an object.
- A class contains methods and property/attribute fields that describe the behaviours and characteristics of objects.

- To create a class:

```
class NameOfClass(object):
```

e.g.

```
class Employee(object):
```

By convention class names start with a capital letter.

Objects and instantiation

- If a class is the template of an object then an object is the realisation or instance of a class. Multiple objects can be created from a class.
- Instantiation refers the creation of objects. Remember the class is just a template, we can then create multiple objects from the template and this is what we refer to as instantiation.
- Object instantiation in Python:

```
object=NameOfClass(object)
```

e.g.

```
fred=Employee()
```

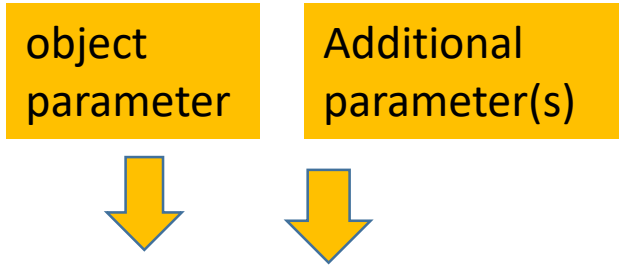
Methods

- Classes define behaviours using methods. Methods are effectively functions defined inside a class. Classes can contain multiple methods.
- In Python, all methods need to have the object parameter (by convention called self) and they can have additional parameters. A class method can be called by:

```
object=NameOfClass.nameOfMethod()
```

e.g

```
Scooby=Dog.name("Scooby")
```




```
class Greeting():
    def salutation(self,name):
        print("Hello",name)

g=Greeting()
g.salutation("Fred")
```

Bounded versus unbounded method calls

The need for an object parameter (self) becomes evident when we consider an unbounded method call where the object itself is explicitly passed as a parameter into the method call. By convention this object parameter is called self

object parameter

```
class Greeting():
      def salutation(self,name):
            print("Hello",name)

# Object Instantiation
g=Greeting()
g.salutation("Fred") # bounded
Greeting.salutation(g,"Fred") # unbounded
```

Calling a method from a method

A methods can call another method within the same class using:

```
self.method()
```

Calling a method

```
class Greeting:
    def salutation(self):
        self.formal()
        self.informal()

    def formal(self):
        print ("Good afternoon sir")

    def informal(self):
        print ("Wagwan")

g = Greeting()
g.salutation()
```

Special method: Constructor

The constructor is a special method that is called automatically whenever an object is created. In Python, it has the name __init__

```
    def __init__(self):
```

```
class Greeting():
    # Constructor
    def __init__(self):
        print("Hello World")
# Object Instantiation
g=Greeting()
```

Attributes

Attributes are characteristics of an object and can be defined and initialised in the constructor method.

```
class Employee():
      #constructor
      def __init__(self,name,dob,salary):
            self.name=name
            self.dob=dob
            self.salary=salary

# Object Instantiation
Fred=Employee("Fred","26th April 2000","£40,000")
Lucy=Employee("Lucy","5th May 2003","£23,000")
print(Fred.name,Fred.dob,Fred.salary)
print(Lucy.name,Lucy.dob,Lucy.salary)
```

Methods example 1

```
class Square():
        def __init(self, size=2)
                self.size=size

        def area(self):
                return self.size * self.size

        def setSize(self,size):
                self.size=size

        def getSize(self):
                return self.size

s=Square()
s.setSize(4)
print(s.area())
```

Task 2: Simple calculator

Create a calculator program using classes. The calculator needs to have multiply, subtract and divide methods. Modify the code below to achieve this.

```
class Calculator(object):
      #methods
      def add(self, x, y):
            return x + y
      # Instantiation
obj1=calculator()
print(obj1.add(3,4))
```

Summary

Creating a class	`class NameOfClass(object):`
Defining a Method	`def methodName(self):`
Defining a constructor method	`def __init__(self):`
Instantiation	`object=NameOfClass(object)`
Calling a method (outside class)	`object.methodName()`
Calling a method (inside class by convention)	`self.methodName()`

Plenary: define the following terms

Class	
Object	
Instantiation	
Attributes	
Methods	