Python

Self refers to the object that we are conducting a function/method on

**Class**

Template from which we create an object

A template for creating objects

A class is like an *object constructor*, or a “blueprint” for creating objects

They provide a means of bundling data and functionality *together*

* Creating a new class creates a new type of object, allowing new instances of that type to be made

A class creates a user-defined data structure, which holds its own data member and member functions, which can be accessed and use by creating an instance of that class

Classes allow us to create a *template* and then objects are used to fill this template

In this template we define what attributes we want it to have – a variable, and then functions and methods for what we want these attributes to do

1. Specify what the properties are
2. Specify what the operations are
3. Specify what the name of the class is

A person standing in front of a white board

Description automatically generated

**Object**

An object is a packet of data in the form of state and behaviour (attributes and methods)

An object is a collection of data (usually variables) and methods (functions)

It is a data field that has unique *attributes* and *behaviours*

Objects have **state** and **behaviour:**

* **State 🡪** type, the attributes that the object has
* **Behaviour** 🡪 the functions of the object, the things that it can do

**Instantiation**

The creation of an instance of an object

**Initialisation**

Setting the initial values (data) to variables which can then be used in later objects

Assigning data to references

Relates to setting particular data (e.g., for an instance of a class)

\_\_init\_\_ initialises the data to the class

Self means current class

Scratch is a file that you can create that utilises the virtual environment but is not *part* of the project, so if you want to test something then you can do it as a scratch

Convention is to name directories and python files as lower case, but variables and classes, etc should use normal grammar rules

Official documentation usually incorporates triple quotes

Method – functionality within a class, function – functionality outside of a class

**Import**

The import keyword is used to make code in one module available to another

* This allows you to structure code effectively, as you can reuse code while making your products more maintainable

The import statement combines two operations: it searches for the named module, and then binds the results of that search to a name in the local scope

You can use the from keyword to only import a part of the module

* If you do this then you should not refer to the module that you imported when calling the object, just simply the name of the object and the attribute you want to call

This can also come from a **library** which is a collection of related modules:

* It contains bundles of code that can be used repeatedly in different programs
* These files contain functions that can be called upon to use

The benefit of using industry recognised modules is that they are well tested

**Random**

Random is a module that can be imported to generate random numbers, such as between a range

It implements pseudo-random number generators for various distributions

A module is a piece of software that delivers some sort of functionality, usually a file

**Procedural Programming**

A programming paradigm built around the idea that programmes are sequences of instructions to be executed

They focus heavily on splitting up programs into named sets of instructions called procedures, analogues to functions

It divides the program into small programs and refers to them as functions

**Problems with PP:**

* Data separated from functions
* Lack of modularity
* High coupling
* Re-use is difficult
* Does not model the real world

**Object Orientation Programming**

Object oriented programming (OOP) refers to languages that use objects, it aims to implement real-world entities like inheritance, abstraction, polymorphism, and encapsulation

The main aim of OOP is to bind together the data and the functions that operate on them so no other part of the code can access this data *except that function*

The benefits of OOP

* Data and functions are combined – good modularity where everything is contained
* Better modularity
* Low coupling – the degree of interdependence between modules
* Re-use is easier
* Closely modelling the real world

**Good coding practices of OOP:**

Generic practices:

1. **Meaningful Names** 🡪 should use meaningful names
2. **Fewer Arguments** 🡪 the number of arguments should be as *minimal* as possible
3. **Avoid global and non-deterministic behaviour** 🡪 we should ensure that the global behaviour of the variable and objects are *minimised*
4. **Avoid static methods** 🡪
5. **Avoid using constructors 🡪**
6. **Reducing conditional statements** 🡪

The four principles of object-oriented programming must be met to be considered one

The four principles:

**Encapsulation** 🡪 A way of hiding the *complexity* of something and exposing only the parts you wish

* **Giving access but hiding the data**
  + Grouping data and methods within a class to provide a *protective barrier* over internal implementation details
  + Provide a simple, consistent interface to use the object
  + Each class should have well defined *responsibility* – all objects within the class should be relevant to its purpose
  + In python, this can be done by adding underscores (\_) before an attribute to try and add a layer of encapsulation

**Inheritance** 🡪 used to create a new class that has all the properties and behaviours of another class, with the potential to add or override them

* + Involves having a *base class* and an *inherited class*, in which the base class takes attributes from the inherited class
  + You can create several levels of base classes and inherited classes 🡪 **inheritance hierarchy**

**Abstraction** 🡪 this involves hiding the implementation details of an object

* + It describes bundling data and methods that work on that data within one unit
  + Characteristics that relate to a specific class
  + Users only care about *using* the function, rather than the implementation of it
  + **Hiding implementation details (abstracted away) from the user**

**Polymorphism** 🡪 this allows methods to act differently based on the object type they are acting on

* We can have a number of different classes which are capable or dealing with the same kind of message/operation being called, but the individual classes know how to perform that action for their particular case

**SSH Keys**

**SSH** 🡪 Secure Shell

These create codes, one that is encrypted and another that decrypts it, like a lock and key

* One lives on your local machine (decrypter) whilst the other resides on GitHub (encrypted)