**Python Object Oriented Programming**

Element related to Object Oriented Programming

Objects have different attributes. The attributes are link specifically to the object

Objects have different methods which are the actions the objects can do. 1 action can only be done by a specific object it can’t be done by multiple objects.

Steps of object oriented analysis and design:

Analysis: define the objects

Design: define how the different object interacts with each other and create diagram of sequences

Programming: create the class diagram.

Class is way to create objects that we call instances.

An instance is a type/model of the object. For example we can create a class for a chair and then multiple instances for the different chairs (office chairs, dining chairs etc..) and their different attributes (colors, weight, material, etc…)

Class is a called a constructor. It allows to create instances based on the same model.

Create a class:

Class XXX:

Create an instance

Class XXX:

Instance1 = XXX()

Class do not function on their own they need at least 1 instance to be created in order to be used hence the Instance1 = XXX()

Create a method in a class:

To create a method in a class we will use a function:

Class XXX:

Def function\_name(self, par1):

Return string + par1

Instance1 = XXX()

To call the method with the instance:

Instance1.function\_name(arg1)

Create the attributes with a constructor

Class XXX:

Def \_\_init\_\_(self):

self.attribute\_name = z

Instance1 = XXX()

We can call the attribute with the instance.

Instance1.attribute\_name will return the value z.

More commonly the attribute would be set as a parameter of the class.

Class XXX:

Def \_\_init\_\_(self,par1):

Self.attribute\_name = par1

Instance1 = XXX(z)

Instance1.par1 will return the value z.

Import attributes from dictionaries

Class XXX

Def \_\_init\_\_(self, \*\*par):

For key, value par.items():

Setattr(self,key,value)

Key and value here are placeholders. They are normally attributes names and attributes values.

Then we can call any value by using

Instance1 = XXX(\*\*par)

Instance1.key will return the value.

We use “\*\*” before a parameter if the parameter we are using is an element with keys and value it can be a dictionary or a different element.

In order to import a dictionary from a json file.

We must ”import json” module first.

For key in json.load(open(“name\_of\_file”)) => this will import all the keys values from the json file as a dictionary

Instance\_name = XXX(\*\*par) => allocate attribute to the instances.

We can call all the values of key the same way we did before. The difference is that all the value from the json file will be shown (1 to infinity depending on the data base).

If we use a mix of method and dictionary to create an instance:

Class XXX

Def \_\_init\_\_(self, par1,par2,\*\*dictionnary):

self.par1 = x1

self.par2 = x2

For key, value par.items():

Setattr(self,key,value)

Property

We can create a property in python by using @property before a method to turn it into a property.

This way we can access the method as if it is an attribute and not a method.

Class attributes:

If we want to create attribute for the whole class:

We use CAPITAL letters to define the attribute. It should be define right after the class.

ATTRIBUTE = xxx

To use class attributes in an instance we call it with self.ATTRIBUTE.

We can also call it with class\_name.CLASS\_ATTRIBUTE

@classmethod:

Create method for the class. If an instance is used in a method, we have to change the scope of the method so it applies to the entire class not just the instance.

We replace “self” with “cls” for more clarity.

Static Method

@staticmethod

Static method: doesn’t use instances or class to run. They don’t take the class or the instance as an argument. It can be executed independently.

Encapsulation:

Public Method = accessible outside class

Private Method = can only be executed within class

Protected Method = can be accessible outside class but not easily.

Protected method will be written \_method\_name

Private method will be written \_\_method\_name

It is possible to access private method using the syntax: instance.\_className\_methodName

Inheritance in class

A child class will inherit all methods and instances from the parent.

Children method/attributes will have priority, so we can overwrite the information or set up different class attributes. Python will check the children class first then it will check the parent class.

Syntax will be:

Class XXX(parent\_class):

Def\_\_init\_\_(self, parent\_par1, parent\_par2, children\_par1):

Super().\_\_init\_\_(parent\_par1, parent\_par2) => this will take the attributes from the parent class.

Self.children\_par1 = children\_par1.

If nothing is define within the () of super.init then all the attribute not redefine will be picked from the parent class

raise NotImplementedError => this can be used in an empty parent method. The console will return an error if the method isn’t created in the child class.

Isintance(class1,class2) => check if class1 is an instance of class2

issubclass(class1,class2) => check if class1 is a subclass of class2

List comprehension:

Regular method code:

def average\_agreeableness(self):

if not self.inhabitants:

return 0

agreeableness = []

for inhabitant in self.inhabitants:

agreeableness.append(inhabitant.agreableness)

return sum(agreeableness) / self.population

Use of comprension list on method:

def average\_agreeableness(self):

if not self.inhabitants:

return 0

return sum([inhabitant.agreeableness for inhabitant in self.inhabitants]) / self.population

The loop is directly run inside the list we are creating and we don’t need to initialise the list.

Create a graph:

Graphs are usually built in 2 parts.

It would start with a parent class which would be the basic layout.

Then we will have child class for specific attributes.

Parent are called abstract class and the children are concrete classes.