Object-Oriented Programming (OOP) in Python

Object-Oriented Programming(OOP)

- OOP is a programming paradigm that allows us to structure programs in a way that **`properties (or attributes)`** and **`behaviours`** of real things are combined into separated objects. In other words, real objects are modelled as program objects.

- OOP follows a **`procedural`** approach: it provides a set of steps in the form of functions and blocks of code that are executed sequentially to complete the task.

```
Example: An
object can
represent a
person with a
name, age,
address, height,
etc. the
behaviours of
that person could
be walking,
talking and
running.
```

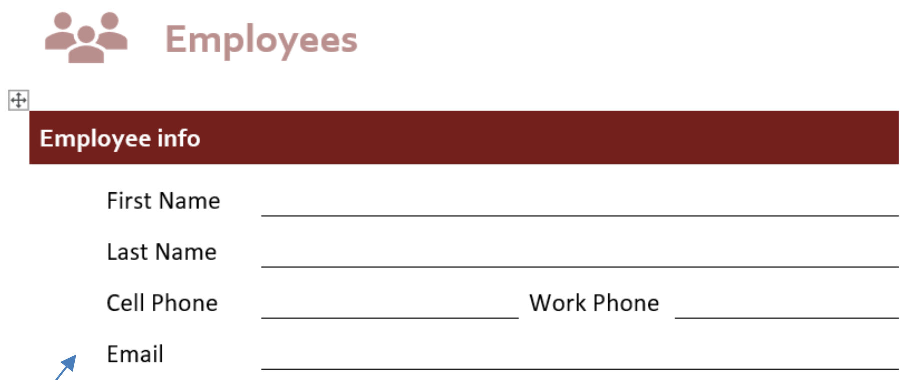
Python Classes and objects (Instances)

- A **class** is a program template (or blueprint) that allows us to create objects.

The keyword **class** is used to create a class.

For example we could create a Employees() class that will indicate that the name and other details that are required.

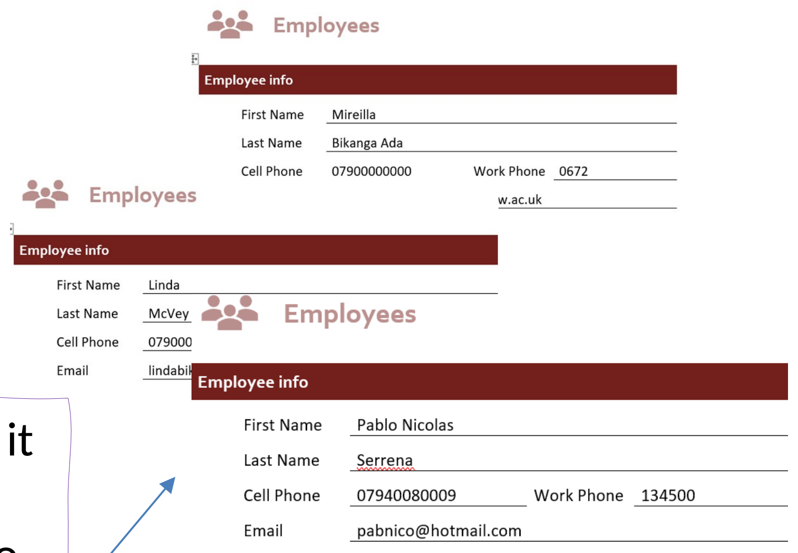
A class does not tell us what the name or other details are. It is an abstract concept. A class helps organise information. We can think of a class as an empty form.



Employees

Employee info

First Name			
Last Name			
Cell Phone		Work Phone	
Email			



Employees

Employee info

First Name	Mireilla		
Last Name	Bikanga Ada		
Cell Phone	07900000000	Work Phone	0672
	w.ac.uk		

Employees

Employee info

First Name	Linda
Last Name	McVey
Cell Phone	079000
Email	lindabik

Employees

Employee info

First Name	Pablo Nicolas		
Last Name	Serrena		
Cell Phone	07940080009	Work Phone	134500
Email	pabnico@hotmail.com		

An **instance** is a copy of a class with actual values, it is an object that belong to a particular class . Many copies can be created. But we need the class (form) to know what information is required. Creating an object of a class is called `instantiation`.

Instance Attribute & Class Attribute

Class create objects. All objects contains characteristics called **`attributes`** (properties).

The **`init ()`** method initialises (creates) the attributes of an object by giving it a default value (state).

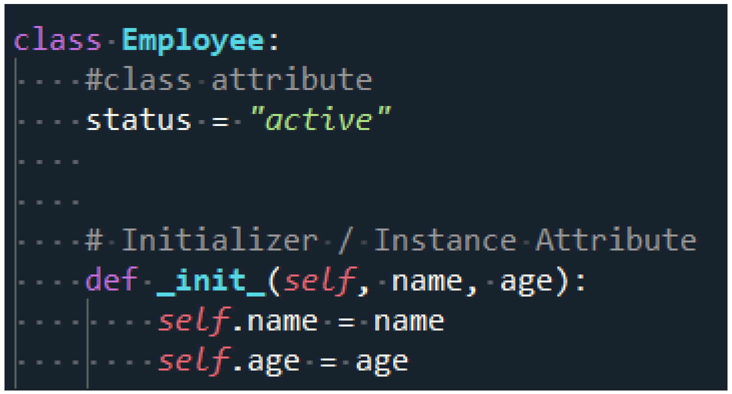
The init () method should have at least one argument and a **`self`** variable. The self variable refers to the object itself (e.g Employees)

`Constructor` is a method that is called by default whenever you create an object from a class.

Instance attributes are specific to each object.

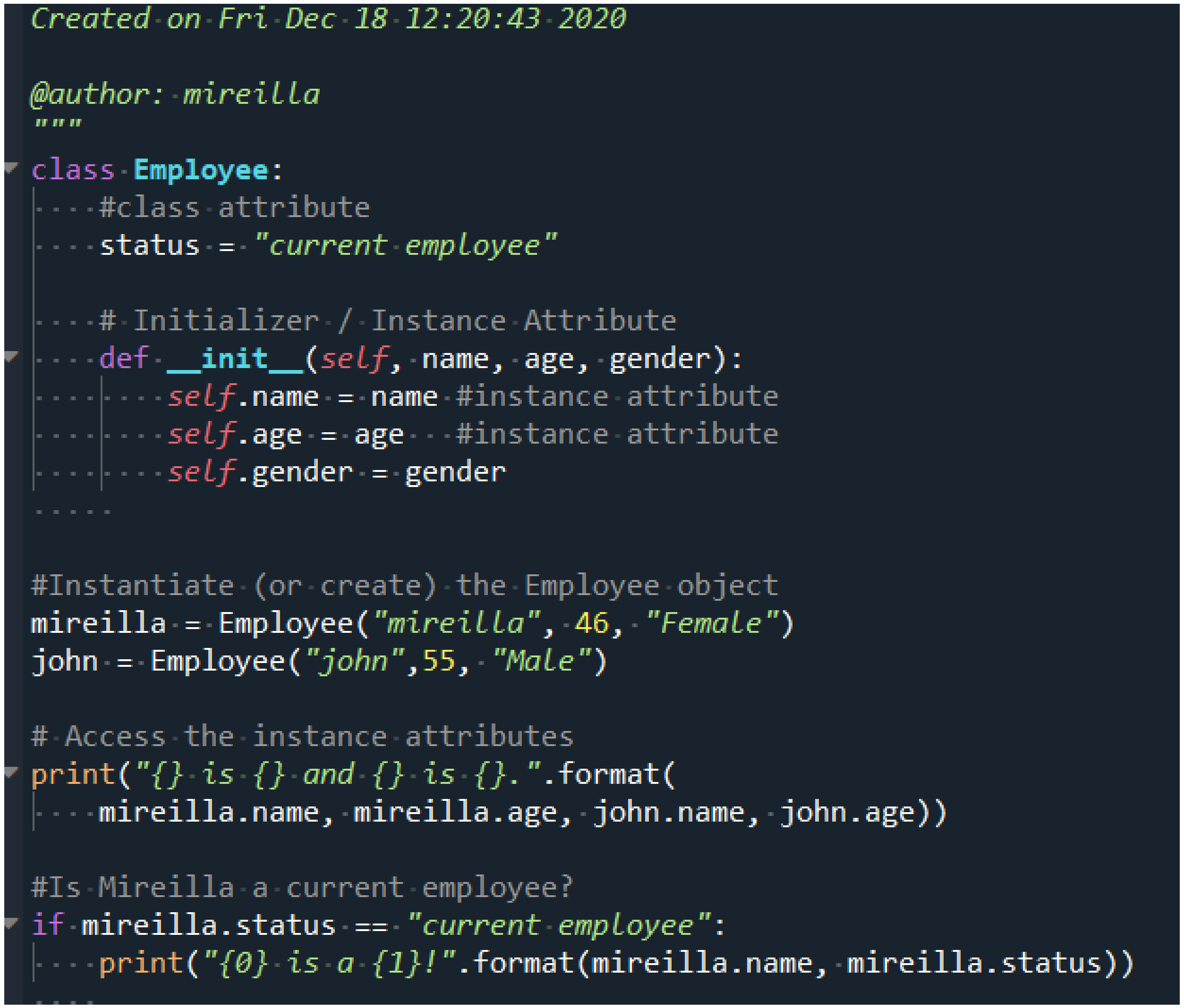
Class attributes are the same for ***ALL*** *instances.*

Although each employee has a unique name and age, each employee will be active (currently employed)

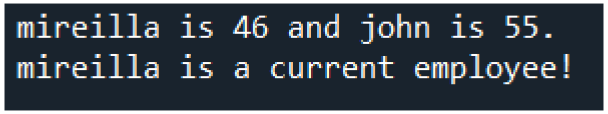


```
class Employee:
    #class attribute
    status = "active"
    
    
    # Initializer / Instance Attribute
    def _init_(self, name, age):
        self.name = name
        self.age = age
```

Class Employee

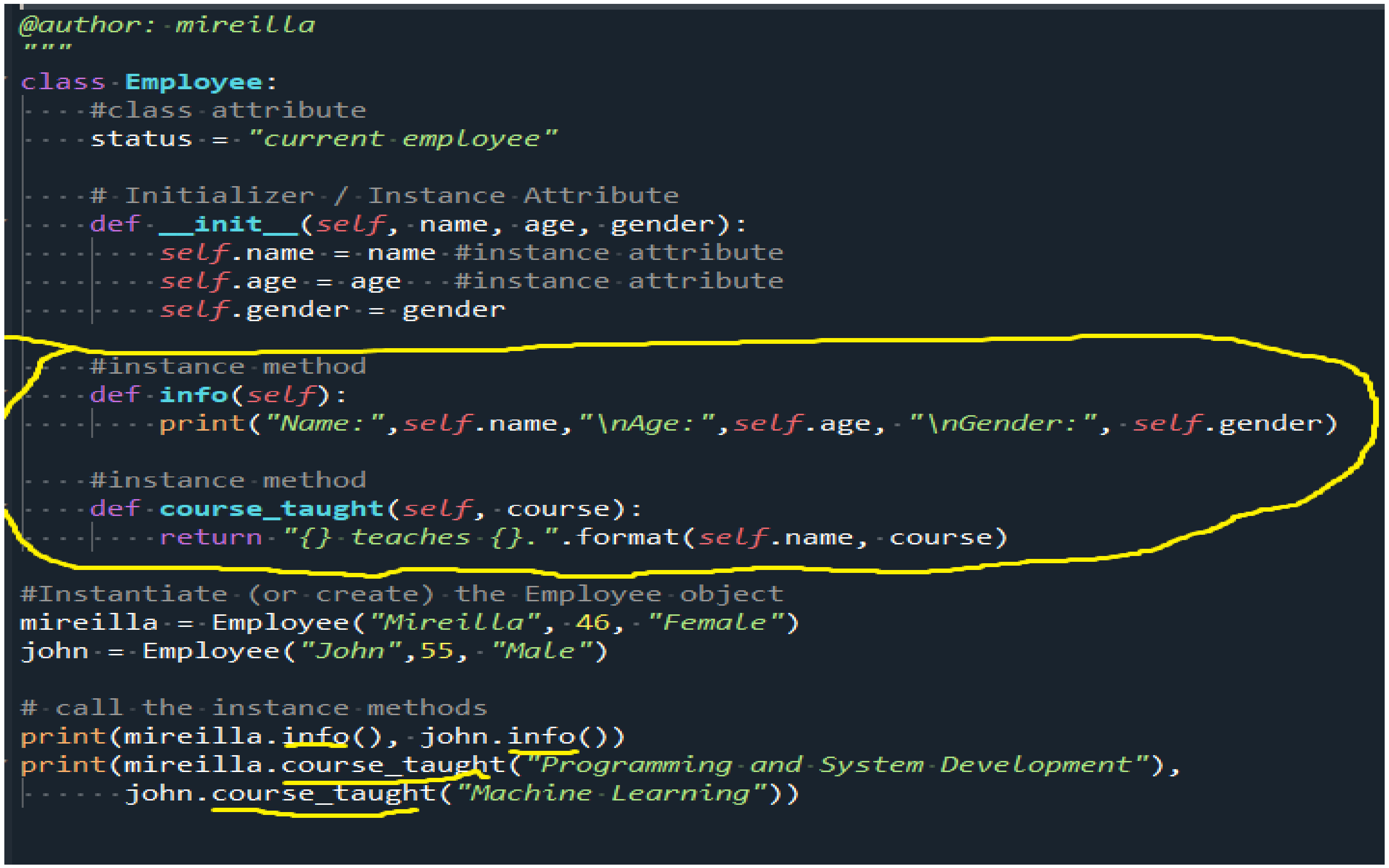


```
Created on Fri Dec 18 12:20:43 2020

@author: mireilla
"""

class Employee:
    #class attribute
    status = "current employee"

    # Initializer / Instance Attribute
    def __init__(self, name, age, gender):
        self.name  = name #instance attribute
        self.age = age   #instance attribute
        self.gender = gender
     

#Instantiate (or create) the Employee object
mireilla = Employee("mireilla", 46, "Female")
john = Employee("john",55, "Male")

# Access the instance attributes
print("{} is {} and {} is {}.".format(
    mireilla.name, mireilla.age, john.name, john.age))

#Is Mireilla a current employee?
if mireilla.status == "current employee":
    print("{0} is a {1}!".format(mireilla.name, mireilla.status))
```

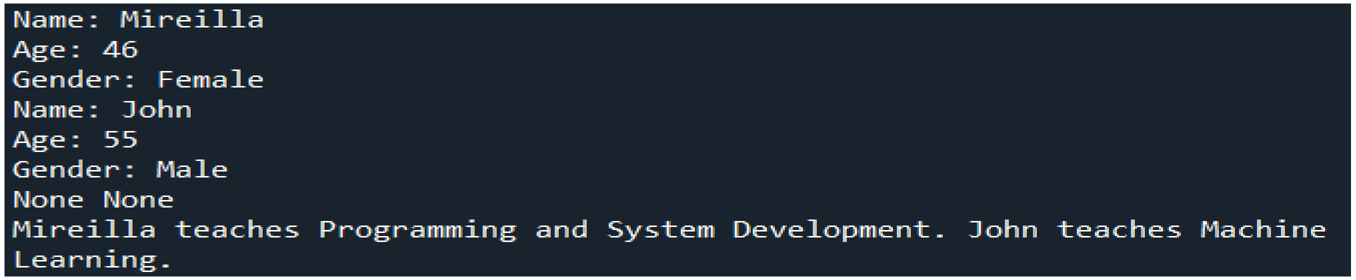


```
mireilla is 46 and john is 55.
mireilla is a current employee!
```

Instance Methods



```
@author: mireilla
"""
class Employee:
    #class attribute
    status = "current employee"

    # Initializer / Instance Attribute
    def __init__(self, name, age, gender):
        self.name = name #instance attribute
        self.age = age  #instance attribute
        self.gender = gender

    #instance method
    def info(self):
        print("Name:",self.name,"\nAge:",self.age, "\nGender:", self.gender)

    #instance method
    def course_taught(self, course):
        return "{} teaches {}.".format(self.name, course)

#Instantiate (or create) the Employee object
mireilla = Employee("Mireilla", 46, "Female")
john = Employee("John",55, "Male")

# call the instance methods
print(mireilla.info(), john.info())
print(mireilla.course_taught("Programming and System Development"),
      john.course_taught("Machine Learning"))
```



```
Name:  Mireilla
Age:  46
Gender:  Female
Name:  John
Age:  55
Gender:  Male
None None
Mireilla teaches Programming and System Development. John teaches Machine
Learning.
```

Instance methods, defined inside the class, are used to get content of the instance and perform operations with the attribute of our objects.

Python Object Inheritance

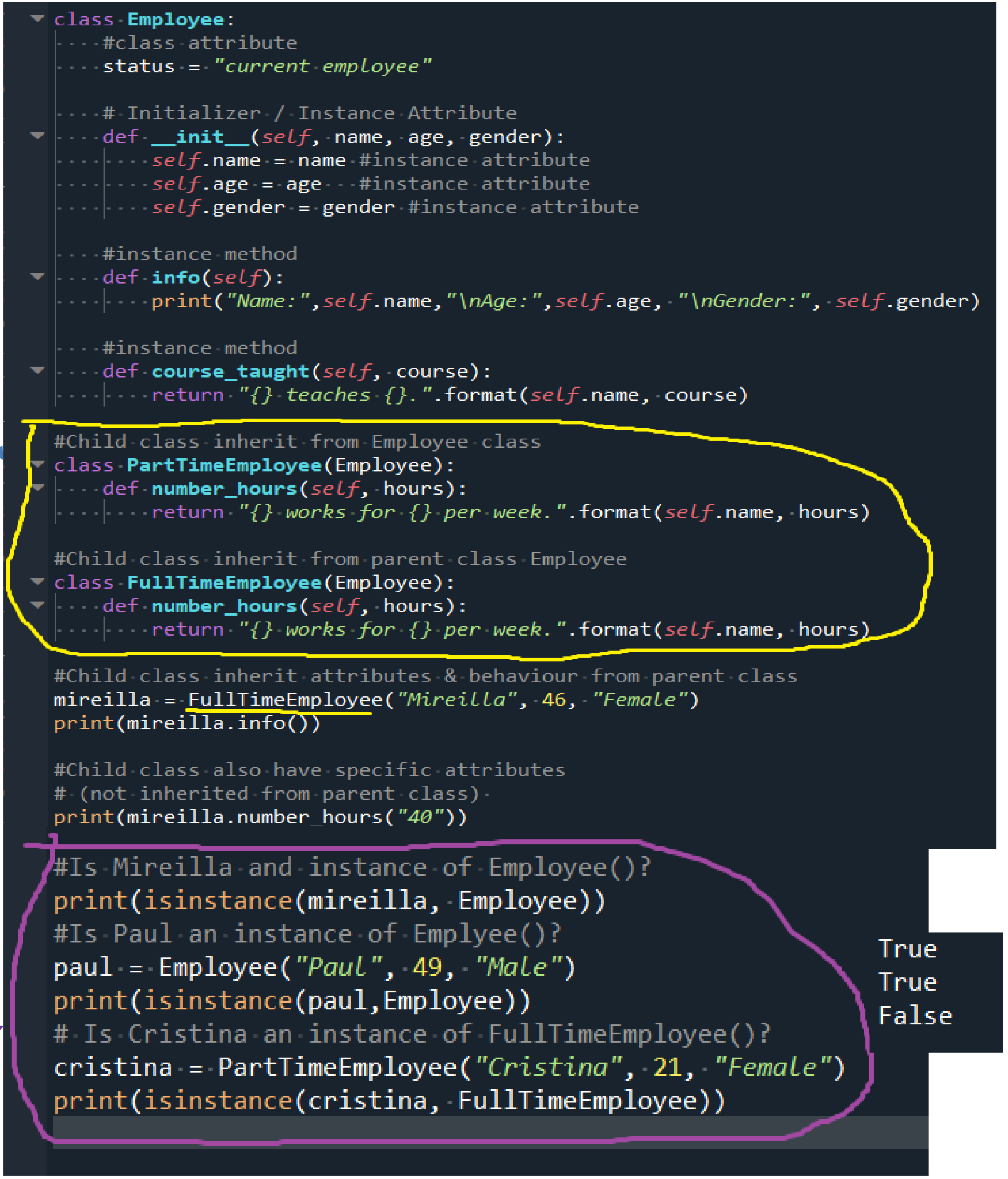
Inheritance is when one class accepts the attributes and methods of another class. It helps prevent code duplication.

Child class: newly created class.

Parent class: a class from which a child class derives.

Child class overrides or extends functionalities: It will take (inherit) all the attributes and behaviour of a parent class and can define further behaviours.

To determine if the instance is an instance of a specific parent class, use isinstance()



```python
class Employee:
    #class attribute
    status = "current employee"

    # Initializer / Instance Attribute
    def __init__(self, name, age, gender):
        self.name = name #instance attribute
        self.age = age   #instance attribute
        self.gender = gender #instance attribute

    #instance method
    def info(self):
        print("Name:",self.name,"\nAge:",self.age, "\nGender:", self.gender)

    #instance method
    def course_taught(self, course):
        return "{} teaches {}.".format(self.name, course)

#Child class inherit from Employee class
class PartTimeEmployee(Employee):
    def number_hours(self, hours):
        return "{} works for {} per week.".format(self.name, hours)

#Child class inherit from parent class Employee
class FullTimeEmployee(Employee):
    def number_hours(self, hours):
        return "{} works for {} per week.".format(self.name, hours)

#Child class inherit attributes & behaviour from parent class
mireilla = FullTimeEmployee("Mireilla", 46, "Female")
print(mireilla.info())

#Child class also have specific attributes
# (not inherited from parent class)
print(mireilla.number_hours("40"))

#Is Mireilla and instance of Employee()?
print(isinstance(mireilla, Employee))
#Is Paul an instance of Emplyee()?
paul = Employee("Paul", 49, "Male")
print(isinstance(paul,Employee))
# Is Cristina an instance of FullTimeEmployee()?
cristina = PartTimeEmployee("Cristina", 21, "Female")
print(isinstance(cristina, FullTimeEmployee))
```

```
True
True
False
```