Programming language(Python)

Class: the word is used “class + name”

Atributos: Son las propiedades que recibe el objeto

```
EXPLORADOR                    objeto.py ●   Curso.py
PYTHON                        OPP > Project > objeto.py > ...
  > .dist                     1  #Para la creacion de una clase se usa de la palabra reservada "class +Nombre"
  OPP \ Project               2  class Persona():
    Curso.py                  3      # Atributos, propiedades o caracteristicas del objeto
    objeto.py                 4      apellido = ""
  > tipos-de-datos            5      nombre = ""
                              6      edad = 0
                              7      despierta = False
                              8
```

Methods: its functions that receive the variables are assigned with the reserved word“def” to define that function and y “self” refers to the class itself

```
#Funcionalidades o metodo
def despertar(self):
    #self(minusculas) que hace referencia a la instancia pertenenciente a la clase
    self.despierta = True
    print("Buen dia")
```

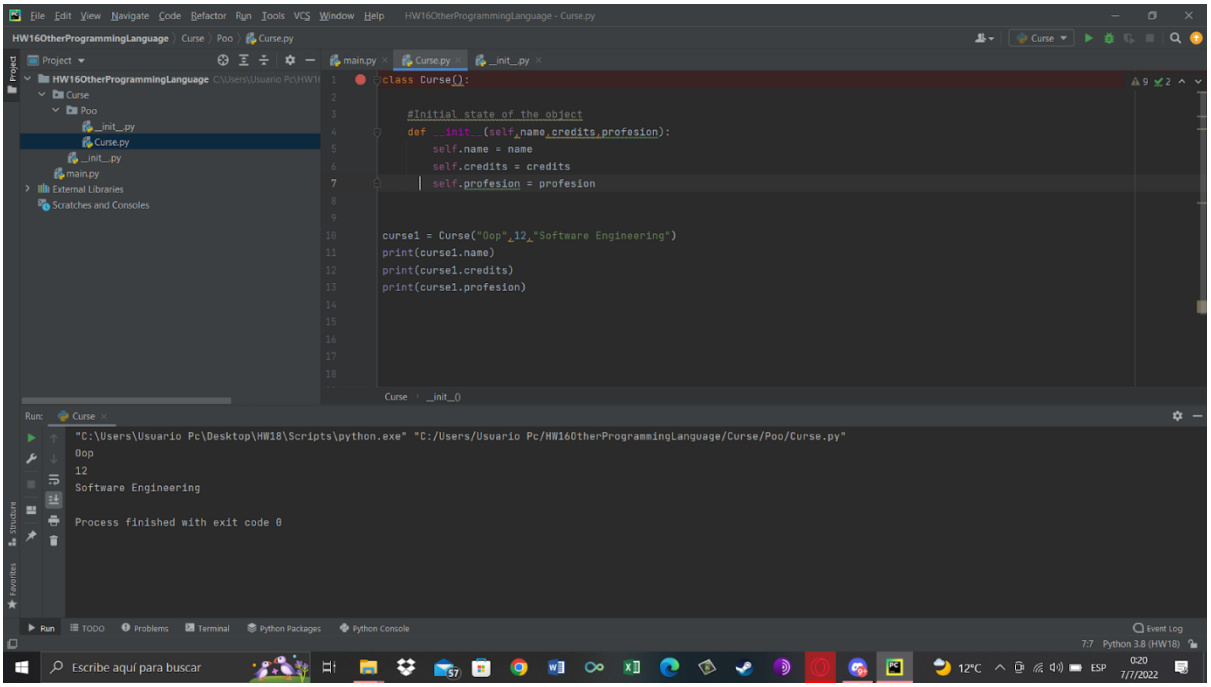
Polymorphism: Python being a weakly typed language, that is to say that when passing a parameter it is not necessary to indicate the data type, since it implicitly interprets

```
OPP > Project > Polimorfismo.py > describirPersona
1  class Estudiante():
2
3      def describir(self):
4          print("Soy un buen estudiante")
5
6  class Docente():
7
8      def describir(self):
9          print("Me dedico a enseñar a cursos")
10
11
12 class Trabajador():
13
14     def describir(self):
15         print("Trabajo dentro de una gran empresa")
16
17
18 def describirPersona(persona):
19     persona.describir()
20
21 doc1=Trabajador()
22 describirPersona(doc1)
```

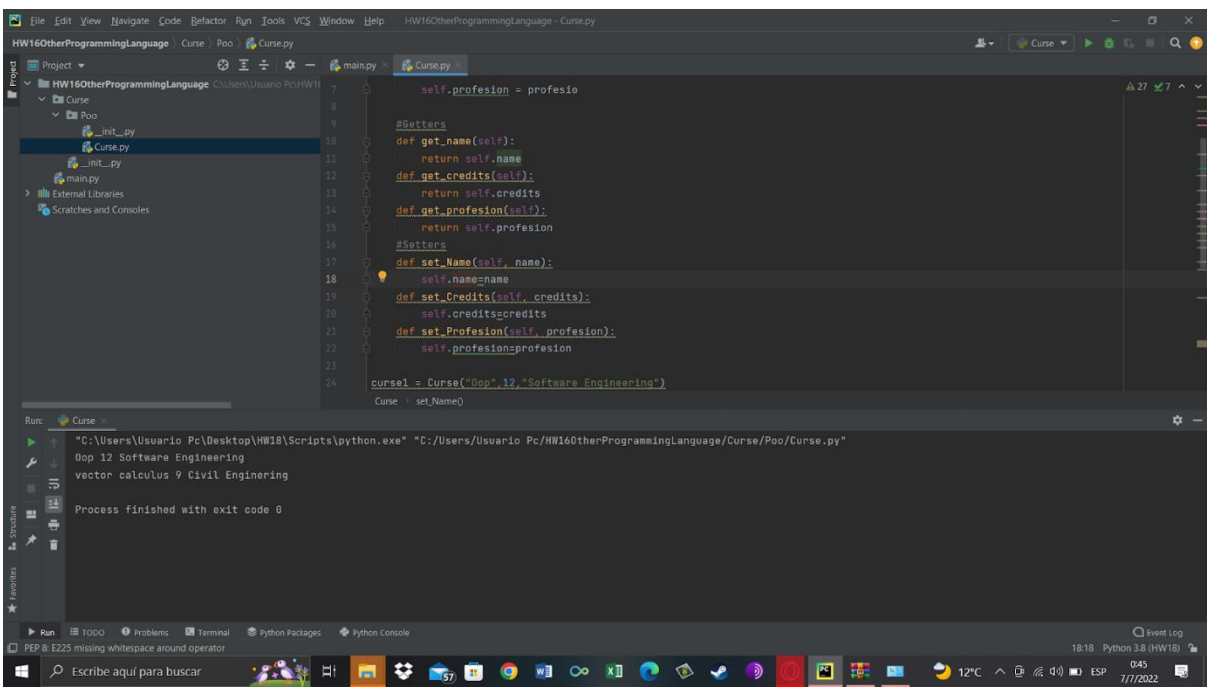
Encapsulation:

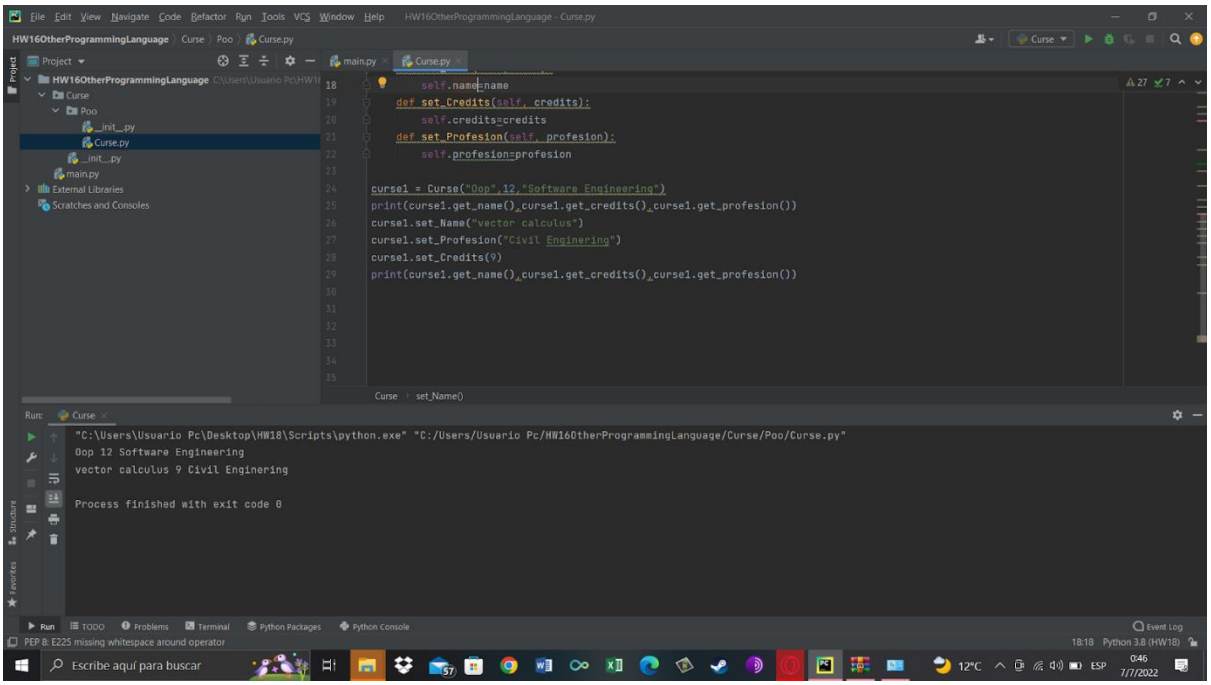
```
OPP > Project > Curso.py > Curso > __init__
class Curso:
    """
    nombre = "Matematicas"
    creditos = 5
    carrera = "Ingenieria Civil"
    """

    #estado inicial del objeto
    def __init__(self,nom,cre,pro):
        self.nombre = nom
        self.creditos = cre
        self.carrera = pro
        self.__imparticion = "Presencial" #propiedad encapsulada

    def mostrarDatos(self):
        dat="Nombre: {0} / Creditos: {1} / Modo de particion: {2}"
        print(dat.format(self.nombre,self.creditos,self.__imparticion))

curso1 = Curso("matematicas",5,"ingenieria Civil")
print(curso1.nombre)
curso1.mostrarDatos()
```

Constructor: The constructor function is used to initialize all the attributes of the class. The name of this constructor function is the same as the name of the class. The concept of a constructor function is the same in Python, but the name of the constructor function is __init__() for all classes.

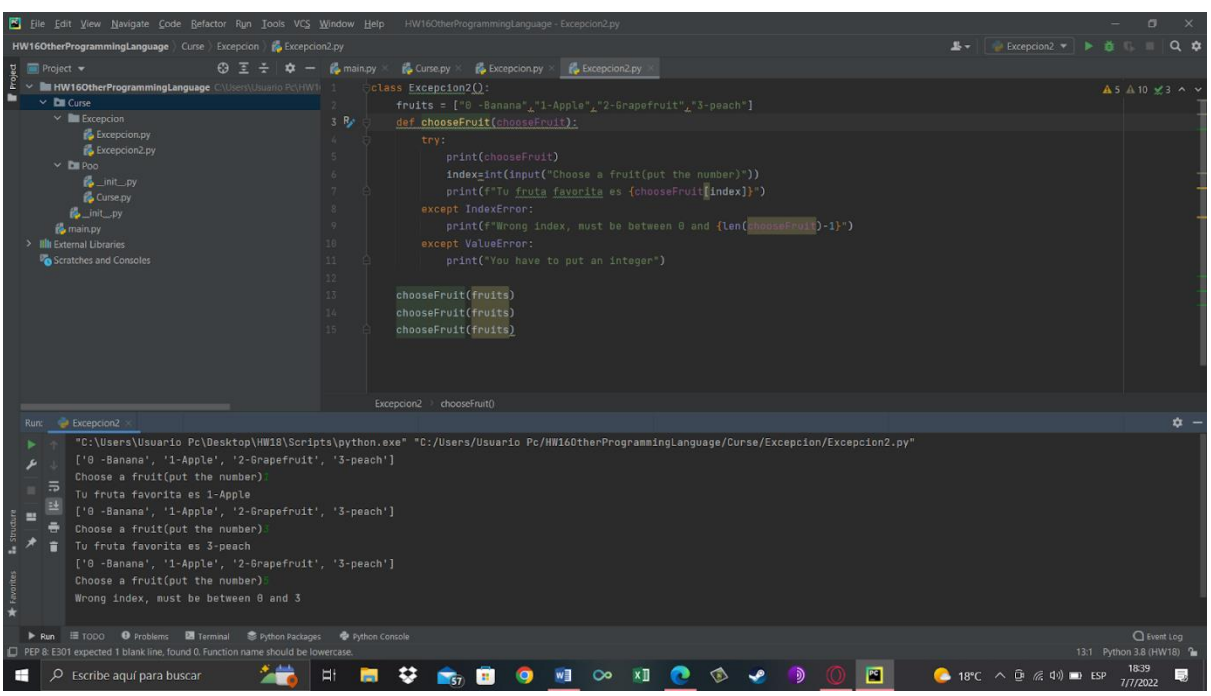


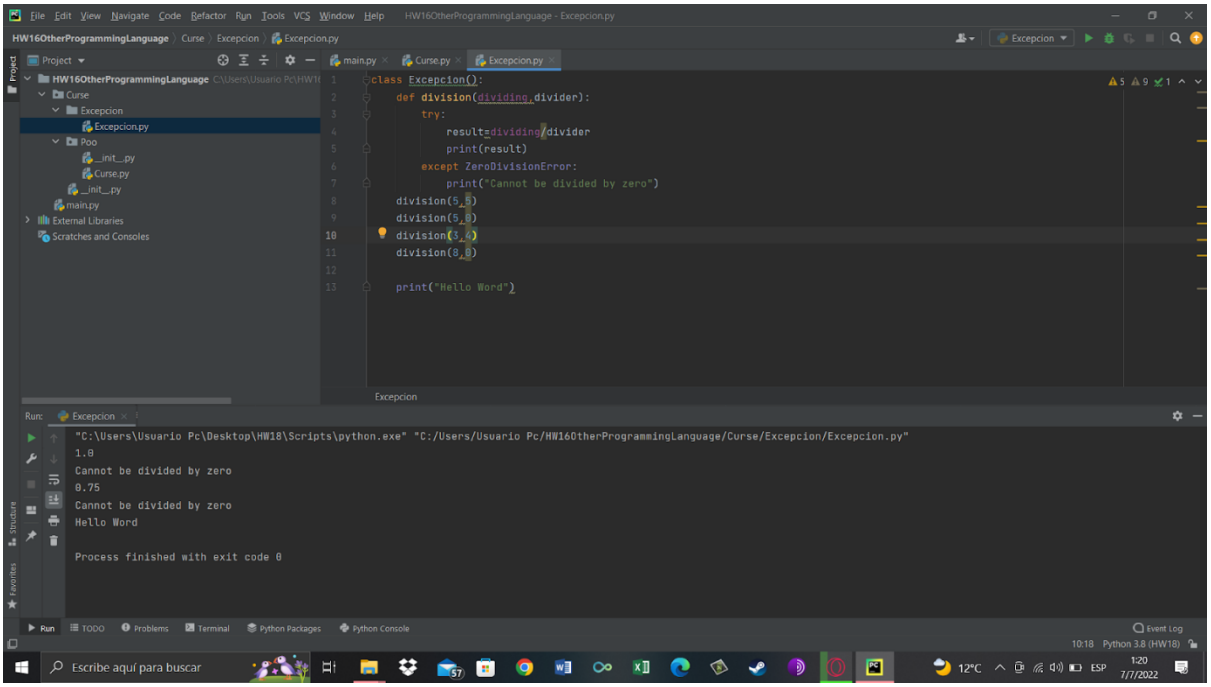
Geetters and Setters: "Getters" and "Setters" are used in OOP to guarantee the principle of data encapsulation. These are typically used in Python: If we want to add validation logic to get and set a value.





Exception: Python uses a special object called an exception to handle any errors that may occur during the execution of a program. When an error occurs during the execution of a program, Python creates an exception.



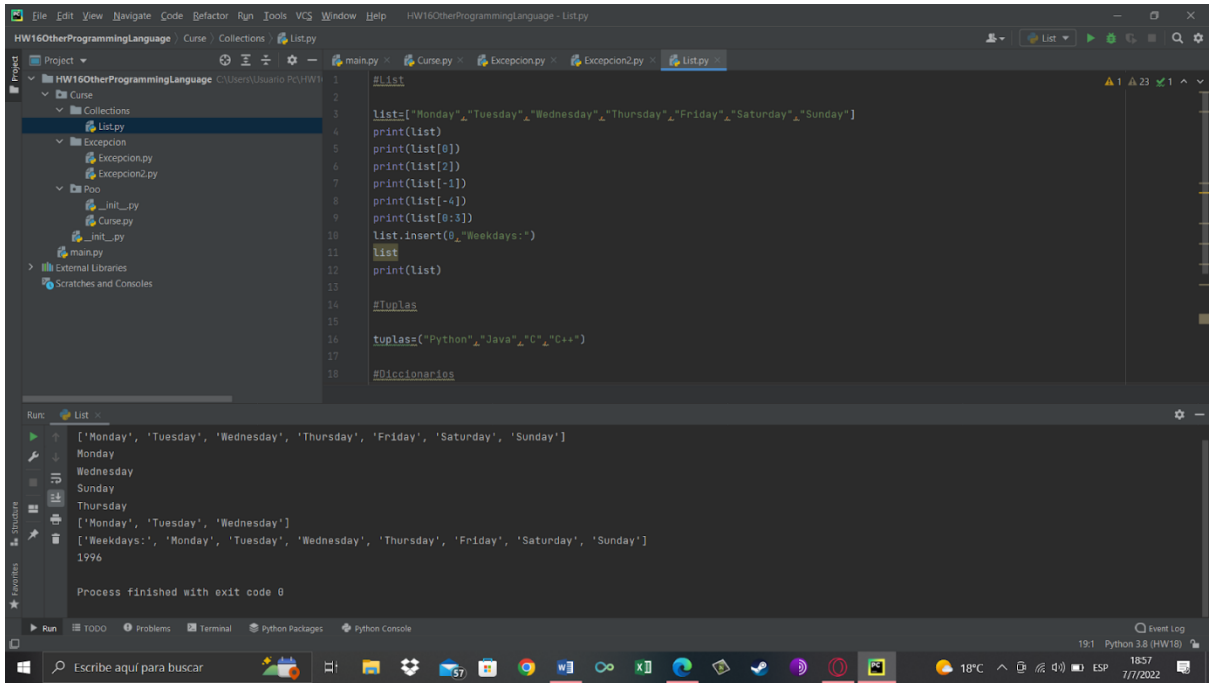


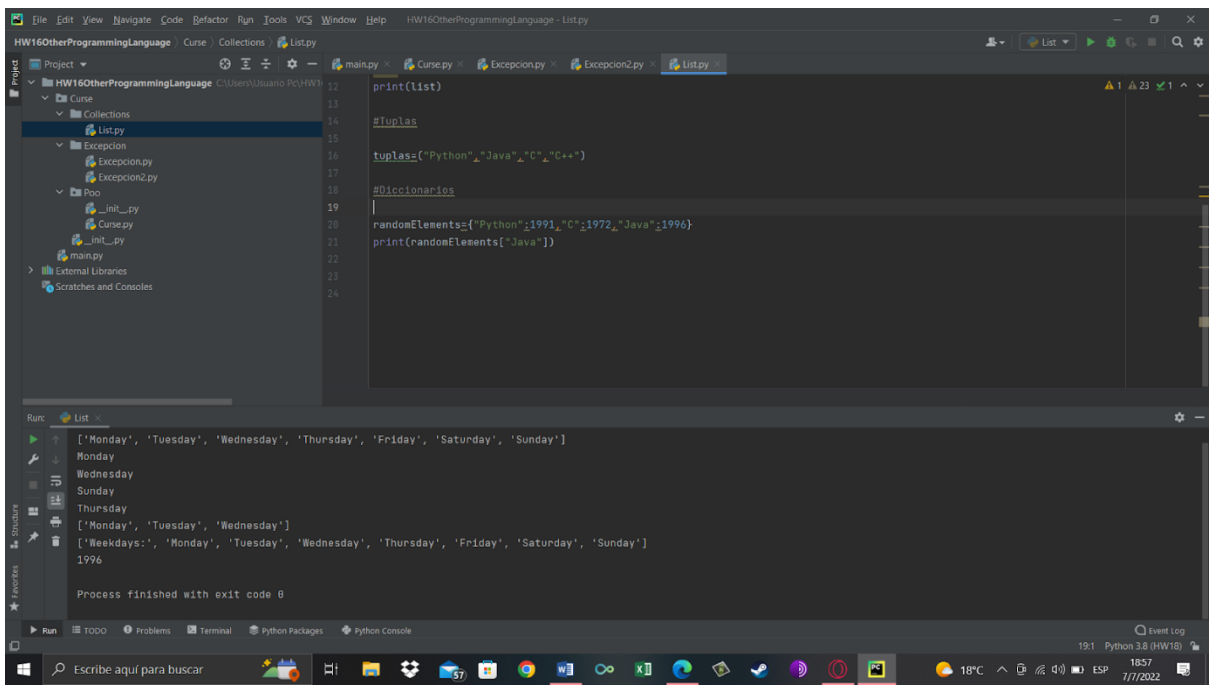
Collection:

Lists:A list is an ordered set of objects. By objects we mean any of the data types already mentioned, including other lists.

Tuples: Tuples, like lists, are ordered collections. However, unlike these, they are immutable. That is, once the elements are assigned, they cannot be altered. In functional terms, it could be said that tuples are a subset of lists, since they support index operations to access their elements, but not assignment operations.

Dictionaries: Dictionaries, unlike lists and tuples, are unordered collections of objects. In addition, its elements have a peculiarity: they always make up a key-value pair. That is, when we add a value to a dictionary, it is assigned a unique key with which it can then be accessed (because the position is no longer a determinant).





Abstraction

Abstraction allows us to create abstract classes and methods. An abstract class must satisfy two basic conditions:

1.- An object will not be created directly from the class.

2.- It must contain at least one abstract method.

This serves as the base (main class) for the other subclasses. For example, an abstract class could be "Tool". They can inherit from her the "hammer", "screwdriver" or "tweezers". These are all tools and it makes sense to create such objects. However, it doesn't make sense to create a "tool" object because it's a very general concept.

```python
from abc import ABC, abstractmethod

class Character(ABC):
    @abstractmethod
    def __init__(self, name):
        self.name = name
        self.level = 0
        self.inventory = []

    @abstractmethod
    def attack(self, target):
        pass
```

Interfaces

An interface is a common means for unrelated objects to communicate with each other. These are definitions of methods and values on which the objects agree to cooperate.

```python
from app.repository.user_repository import UserRepository
from app.business.store_manager import StoreManager
from app.repository.database import Database
from app.repository.s3 import S3
from app.repository.file_system import FileSystem
from app.models.user import User


user = User("Ronny","Ibarra",21)

s3Repository = S3("321312321","sdf32423","MyBucket")
StoreManager.storeUser(user,s3Repository)

print("\n")

fileSystemRepository = FileSystem("/home/users")
```

PROBLEMAS 15 SALIDA CONSOLA DE DEPURACIÓN TERMINAL JUPYTER Python

```
<root><name>Ronny</name><lastName>Ibarra</lastName><age>21</age></root>
Closing file



--->Storing user...

Opening database connection: localhost:admin@admin123
Storing user in the database Ronny

INSERT INTO USER VALUES('Ronny','Ibarra',21)
Closing connection
PS C:\Users\ronny\OneDrive\Documents\Lenguaje Poo>
```