

UNIVERSIDAD PRIVADA FRANZ TAMAYO

DEFENSA HITO 2 – TAREA FINAL

Estudiante: Univ. Juan Manuel Tola Zabaleta

Asignatura: Programación de Sistemas Embebidos

Carrera: Ingeniería de Sistemas

Paralelo: PSE (2)

Docente: Lic. William Roddy Barra Paredes

Fecha: 05/04/2022

Manejo de conceptos

1. ¿Qué es un sistema embebido?

R. Un sistema embebido, embarcado o empotrado es un sistema de computación basado en un micróprocesador o un micro controlador diseñado para realizar una o algunas pocas funciones dedicadas, frecuentemente en un sistema de computación en tiempo real.

2. ¿Mencione 5 ejemplos de sistemas embebidos?

- R.**
- Sistemas de calefacción central
 - Sistemas GPS
 - Rastreadores de fitness
 - Dispositivos médicos
 - Sistemas de automoción

3. ¿Mencione las diferencias o similitudes entre un sistema operativo, un sistema móvil y un sistema embebido?

R. La principal diferencia entre un ordenador tradicional y un sistema embebido está en la optimización del espacio y sus características técnicas. También en que elimina todo aquello que no es necesario para realizar su función. La diferencia entre un sistema operativo móvil (SO) y un sistema operativo de computadora tiene que ver con cómo las compañías tecnológicas individuales han implementado varias versiones de los sistemas operativos que proporcionan los entornos fundamentales para las aplicaciones de software tradicionales, así como las nuevas aplicaciones móviles.

4. ¿A que se referían los términos MCU y MPU? Explique cada una de ellas.

R.

- MCU, micro controlador es un circuito integrado programable, capaz de ejecutar las órdenes grabadas en su memoria. Está compuesto de varios bloques funcionales que cumplen una tarea específica. Un micro controlador incluye en su interior las tres principales unidades funcionales de una computadora: unidad central de procesamiento, memoria y periféricos de entrada/ salida.

- MPU, el microprocesador es el circuito integrado central más complejo de un sistema informático; a modo de ilustración, se le suele llamar por analogía el «cerebro» de un ordenador.

Es el encargado de ejecutar todos los programas, desde el sistema operativo hasta las aplicaciones de usuario; solo ejecuta instrucciones en lenguaje binario, realizando operaciones aritméticas y lógicas simples, tales como sumar, restar, multiplicar, dividir, las lógicas binarias y accesos a memoria. Entre sus componentes, puede contener una o más unidades centrales de procesamiento (CPU) constituidas, esencialmente, por registros, una unidad de control, una unidad aritmético lógica (ALU) y una unidad de cálculo en coma flotante (conocida antiguamente como coprocesador matemático).

5. ¿Cuáles son los pilares de POO?.

- **Abstracción**, puede definirse como las características específicas de un objeto, aquellas que lo distinguen de los demás tipos de objetos y que logran definir límites conceptuales respecto a quien está haciendo dicha abstracción del objeto.
- **Encapsulamiento**, es un principio por el cual, toda persona que use una clase (mediante la instanciación) no va a tener la oportunidad de cambiar los atributos de la clase. Esto significa, por tanto, que las propiedades de la misma estarán ocultas.
- **Herencia**, es un concepto de la programación orientada a objetos. El **cual es** un mecanismo que permite derivar una clase a otra clase. En otras palabras, tendremos unas clases que serán hijos, y otras clases que serán padres.
- **Polimorfismo**, es la capacidad que tienen ciertos lenguajes para hacer que, al enviar el mismo mensaje (o, en otras palabras, invocar al mismo método) desde distintos objetos, cada uno de esos objetos pueda responder a ese mensaje (o a esa invocación) de forma distinta.

6. ¿Mencione los componentes en lo que se basa POO?. Y explicar cada una de ellas.

Los componentes en los que se basa POO son 3 fundamentales: métodos, eventos y atributos.

- **Métodos**, son aquellas funciones que permite efectuar el objeto y que nos rinden algún tipo de servicio durante el transcurso del programa.

Determinan a su vez como va a responder el objeto cuando recibe un mensaje.

- **Eventos**, son aquellas acciones mediante las cuales el objeto reconoce que se está interactuando con él.

De esta forma el objeto se activa y responde al evento según lo programado en su código.

- **Atributos**, características que aplican al objeto solo en el caso en que el sea visible en pantalla por el usuario; entonces sus atributos son el aspecto que refleja, tanto en color, tamaño, posición, si está o no habilitado.

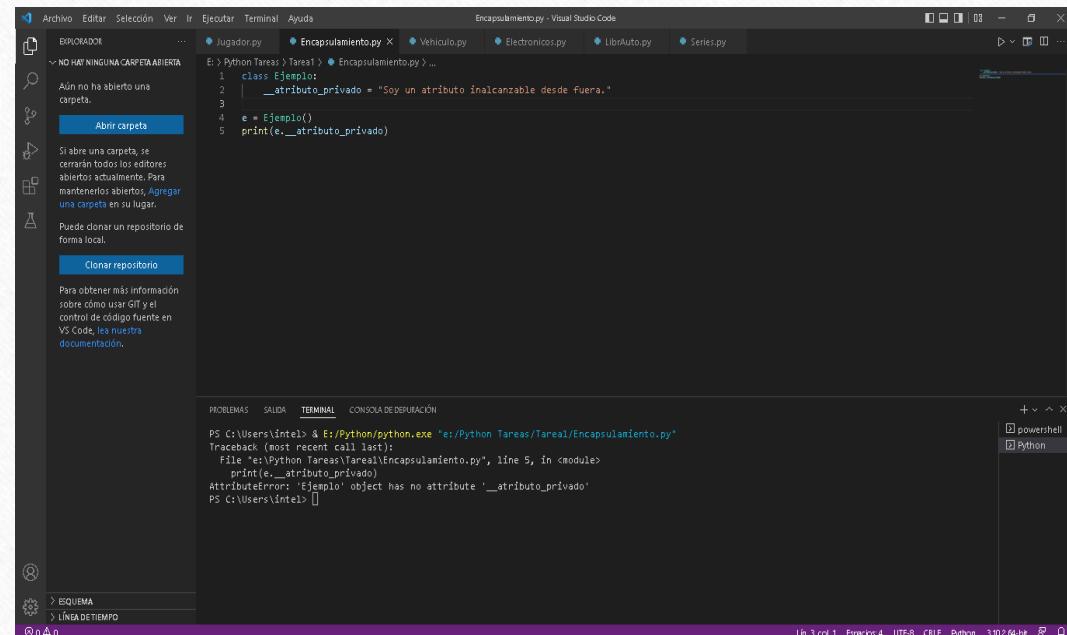
7. Defina los siguientes:

- **Multiplataforma**, es un término utilizado frecuentemente en informática para indicar la capacidad o características de poder funcionar o mantener una interoperabilidad de forma similar en diferentes sistemas operativos o plataformas. Por ejemplo la posibilidad de utilizar un programa o software determinado en sistemas Windows y Linux.
- **Multiparadigma**, la programación multiparadigma es una práctica que emerge como resultado de la existencia de los paradigmas orientado a objetos, procedural, declarativo y funcional buscando mejorar la producción en el desarrollo de proyectos.
- **Multipropósito**, resulta más cómodo adquirir y configurar un dispositivo que satisfaga todas sus necesidades que comprar un dispositivo para cada función. Esto resulta más evidente para el usuario doméstico.
- **Un lenguaje interpretado** es un lenguaje de programación para el que la mayoría de sus implementaciones ejecuta las instrucciones directamente, sin una previa compilación del programa a instrucciones en lenguaje máquina.

8. Defina a que se refiere cuando se habla de encapsulación y muestre un ejemplo (Código en Python).

La encapsulación consiste en denegar el acceso a los atributos y métodos internos de la clase desde el exterior. En Python no existe, pero se puede simular precediendo atributos y métodos con dos barras bajas __ como indicando que son "especiales".

En el caso de los atributos quedarían así:



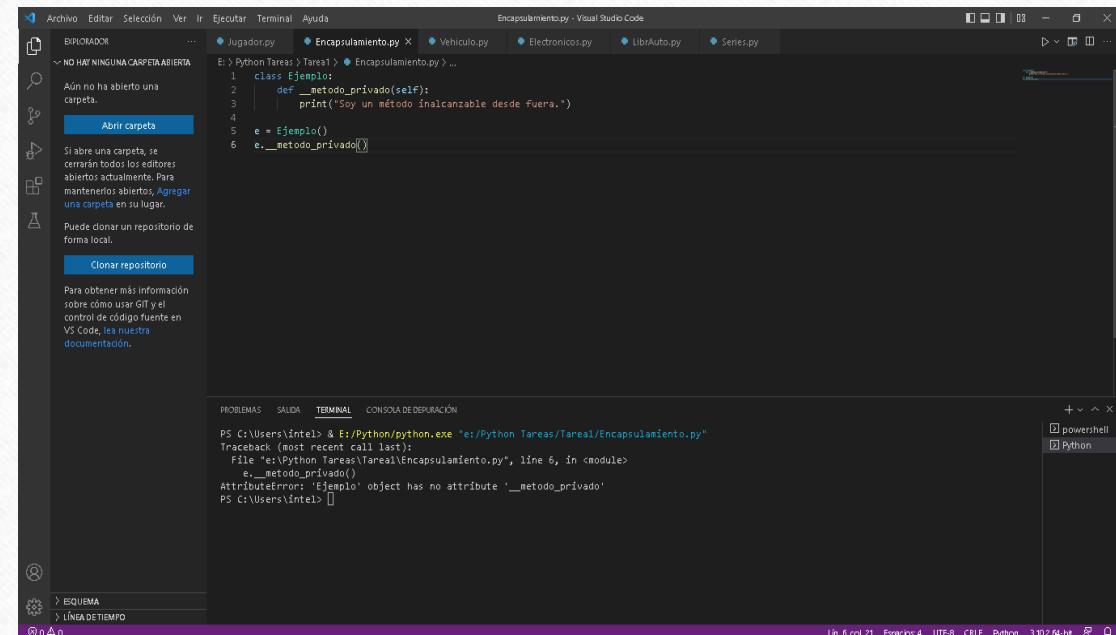
```
Archivo Editar Selección Ver Ir Ejecutar Terminal Ayuda Encapsulamiento.py - Visual Studio Code

EXPLORADOR Archivo Editar Selección Ver Ir Ejecutar Terminal Ayuda Encapsulamiento.py - Visual Studio Code
... Jugador.py Encapsulamiento.py Vehículo.py Electrónicos.py LibroAuto.py Series.py

E:\Python Tareas\Tarea1> & E:/Python/python.exe "E:/Python Tareas/Tarea1/Encapsulamiento.py"
Traceback (most recent call last):
  File "E:/Python Tareas\Tarea1\Encapsulamiento.py", line 5, in <module>
    print(e._atributo_privado)
AttributeError: 'Ejemplo' object has no attribute '_atributo_privado'
PS C:\Users\Intel>

PROBLEMAS SALIDA TERMINAL CONSOLES DE DEPURACIÓN
PS C:\Users\Intel> & E:/Python/python.exe "E:/Python Tareas/Tarea1/Encapsulamiento.py"
Traceback (most recent call last):
  File "E:/Python Tareas\Tarea1\Encapsulamiento.py", line 6, in <module>
    e._metodo_privado()
AttributeError: 'Ejemplo' object has no attribute '_metodo_privado'
PS C:\Users\Intel>
```

Y en los métodos:



```
Archivo Editar Selección Ver Ir Ejecutar Terminal Ayuda Encapsulamiento.py - Visual Studio Code
... Jugador.py Encapsulamiento.py Vehículo.py Electrónicos.py LibroAuto.py Series.py

E:\Python Tareas\Tarea1> & E:/Python/python.exe "E:/Python Tareas/Tarea1/Encapsulamiento.py"
Traceback (most recent call last):
  File "E:/Python Tareas\Tarea1\Encapsulamiento.py", line 6, in <module>
    e._metodo_privado()
AttributeError: 'Ejemplo' object has no attribute '_metodo_privado'
PS C:\Users\Intel>

PROBLEMAS SALIDA TERMINAL CONSOLES DE DEPURACIÓN
PS C:\Users\Intel> & E:/Python/python.exe "E:/Python Tareas/Tarea1/Encapsulamiento.py"
Traceback (most recent call last):
  File "E:/Python Tareas\Tarea1\Encapsulamiento.py", line 6, in <module>
    e._metodo_privado()
AttributeError: 'Ejemplo' object has no attribute '_metodo_privado'
PS C:\Users\Intel>
```

¿Qué sentido tiene esto en Python?

Ninguno, porque se pierde toda la gracia de lo que en esencia es el lenguaje:

flexibilidad y polimorfismo sin control (Sea como sea para acceder a esos datos se deberían crear métodos públicos que hagan de interfaz. En otros lenguajes les llamaríamos *getters* y *setters* y es lo que da lugar a las *propiedades*, que no son más que atributos protegidos con interfaces de acceso:

The screenshot shows a Visual Studio Code interface with the following details:

- File Explorer (Left):** Shows a message "NO HAY NINGUNA CARPETA ABIERTA" and buttons for "Abrir carpeta" and "Clonar repositorio".
- Code Editor (Center):** Displays a Python file named "Encapsulamiento.py". The code defines a class "Ejemplo" with private attribute "_atributo_privado" and private method "__metodo_privado". It also has public methods "atributo_publico" and "metodo_publico" which return or call the private members respectively.
- Terminal (Bottom):** Shows the output of running the script with "python.exe". The output is:

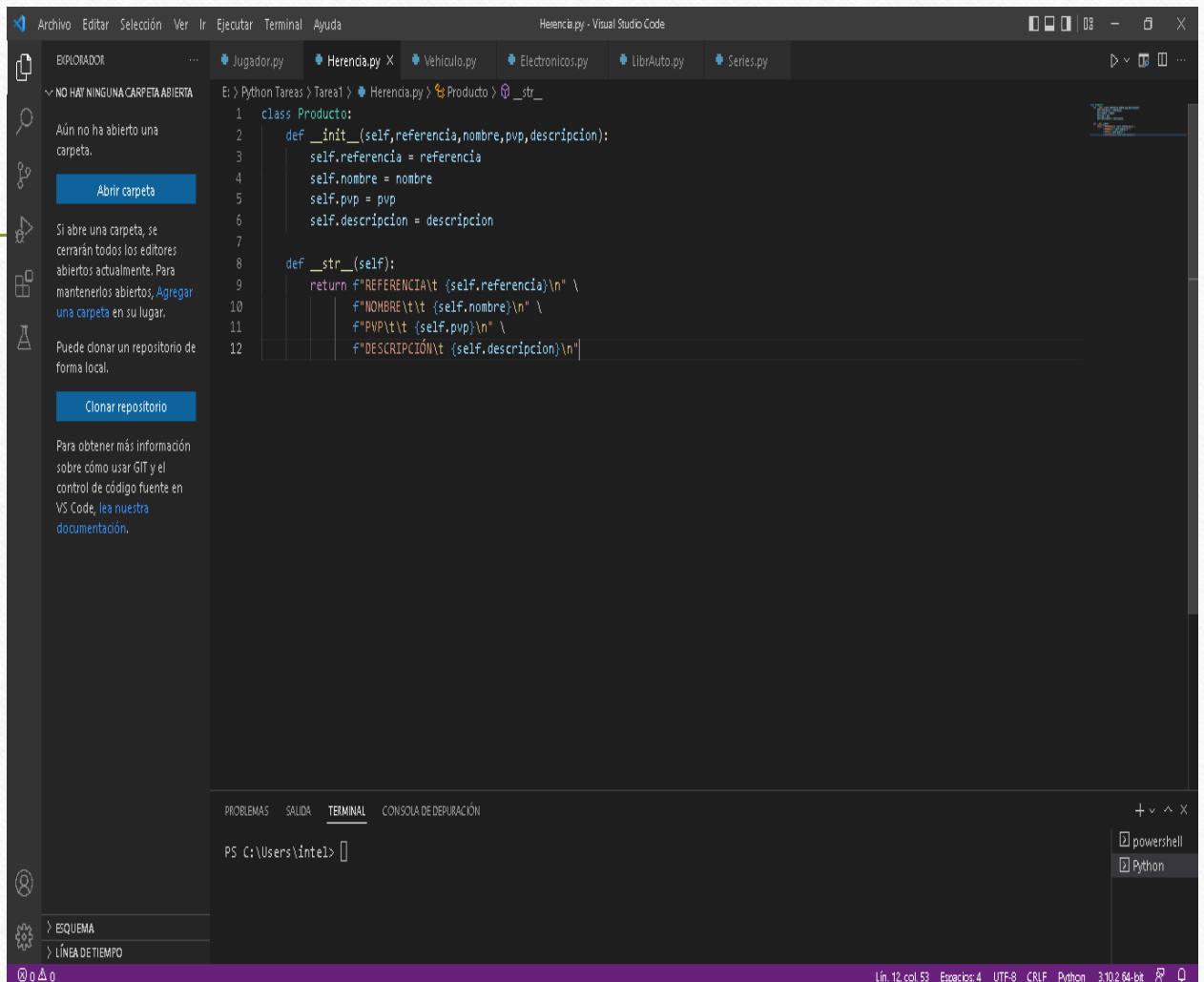
```
PS C:\Users\intel> & E:/Python/python.exe "E:/Python Tareas/Tarea1/Encapsulamiento.py"
Soy un atributo inalcanzable desde fuera.
Soy un método inalcanzable desde fuera.
PS C:\Users\intel>
```
- Bottom Status Bar:** Shows "Lín. 15 col. 19 Espacios:4 UTF-8 CRLF Python 3.10.2 64-bit".

9. Defina a que se refiere cuando se habla de herencia y muestre un ejemplo (Código en Python).

La herencia es la capacidad que tiene una clase de heredar los atributos y métodos de otra, algo que nos permite reutilizar código y hacer programar mucho más óptimos.

Superclases

Así pues la idea de la herencia es identificar una clase base (la superclase) con los atributos comunes y luego crear las demás clases heredando de ella (las subclases) extendiendo sus campos específicos. En nuestro caso esa clase sería el Producto en sí mismo:



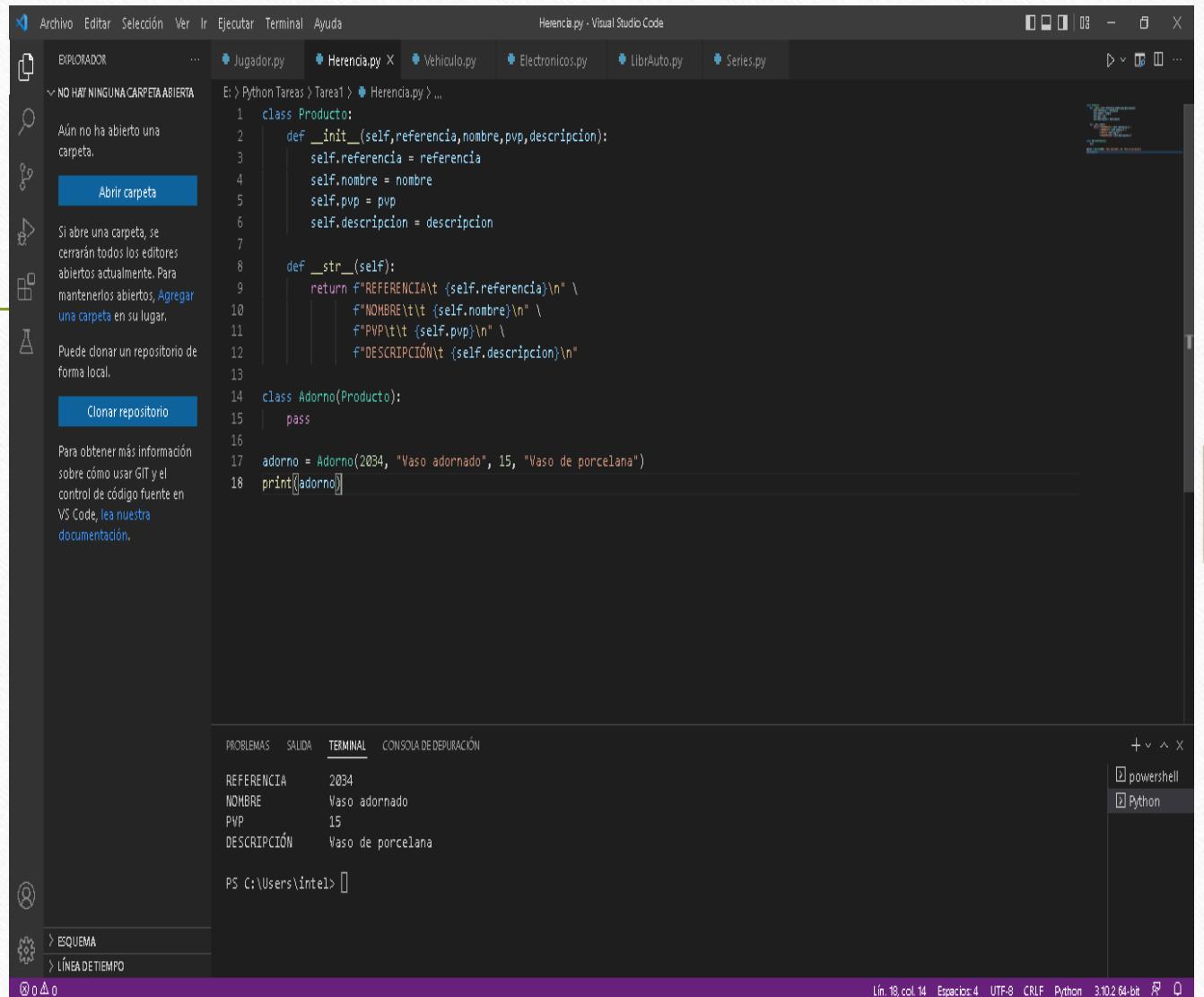
The screenshot shows a Visual Studio Code interface with the following details:

- File Explorer:** Shows files: Jugador.py, Herencia.py (highlighted), Vehiculo.py, Electronicos.py, LibAuto.py, and Series.py.
- Code Editor:** Displays the content of Herencia.py:

```
class Producto:  
    def __init__(self,referencia,nombre,pvp,descripcion):  
        self.referencia = referencia  
        self.nombre = nombre  
        self.pvp = pvp  
        self.descripcion = descripcion  
  
    def __str__(self):  
        return f'REFERENCIA\t {self.referencia}\n' +  
               f'NOMBRE\t {self.nombre}\n' +  
               f'PVP\t {self.pvp}\n' +  
               f'DESCRIPCION\t {self.descripcion}\n'
```
- Terminal:** Shows the command prompt: PS C:\Users\Intel>.
- Status Bar:** Shows the following information: Lín. 12, col. 53 Espacio: 4 UTF-8 CRLF Python 3.10.2 64-bit

Subclases

Para heredar los atributos y métodos de una clase en otra sólo tenemos que pasarla entre paréntesis durante la definición:

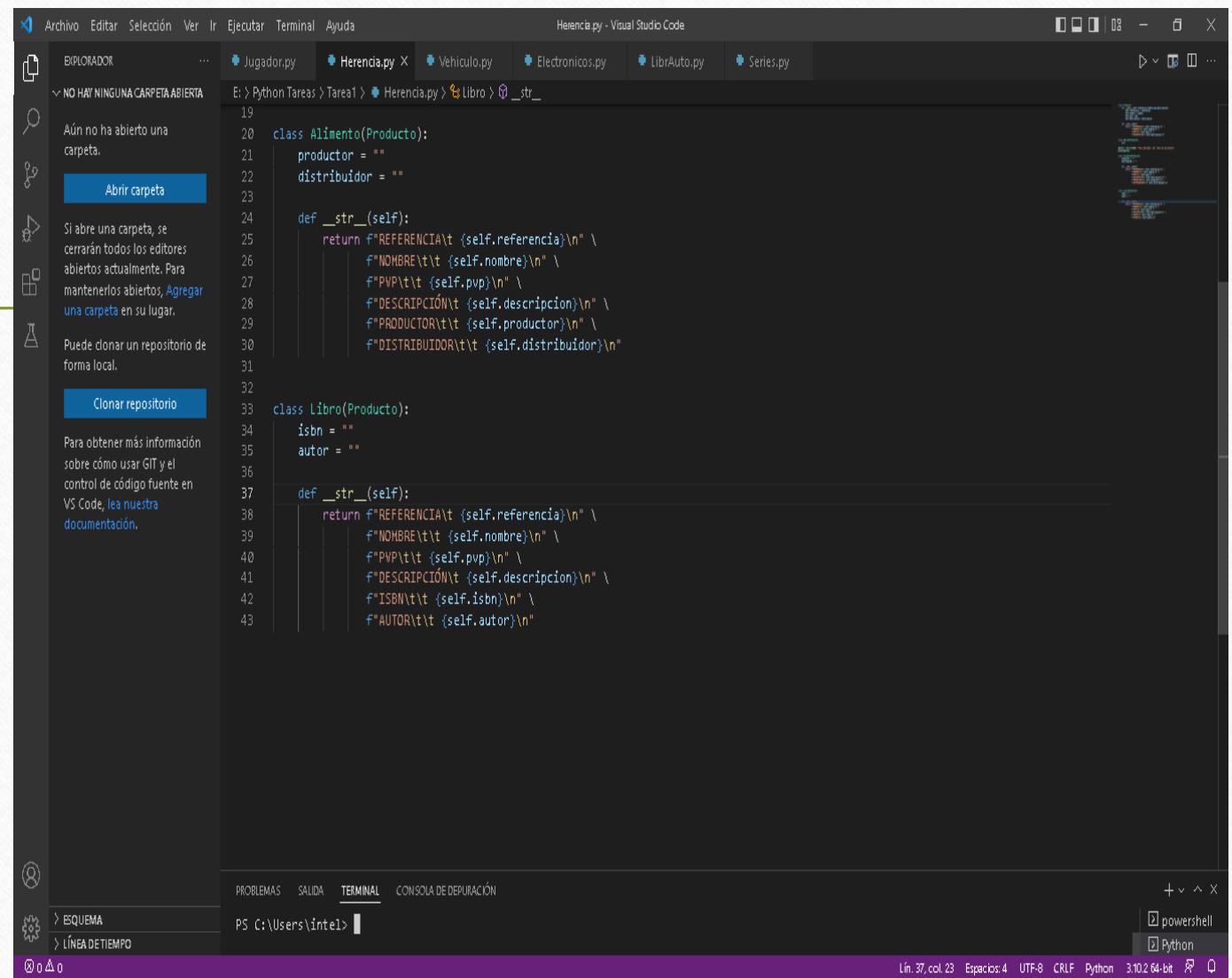


The screenshot shows a Visual Studio Code interface with the following details:

- File Explorer:** Shows a folder structure: E:\Python Tareas > Tarea1 > Herencia.py.
- Code Editor:** Displays the Python code for the `Herencia.py` file. It defines two classes: `Producto` and `Adorno`, both inheriting from `Producto`. The `__init__` method initializes attributes like `referencia`, `nombre`, `pvp`, and `descripcion`. The `__str__` method returns a string representation of the object.
- Terminal:** Shows the output of running the code: `adorno = Adorno(2034, "Vaso adornado", 15, "Vaso de porcelana")` and `print(adorno)`.
- Output:** Shows the variables `REFERENCIA`, `NOMBRE`, `PVP`, and `DESCRIPCIÓN` with their corresponding values: 2034, Vaso adornado, 15, and Vaso de porcelana.
- Status Bar:** Shows the status: Lin. 18 col. 14 Espacio: 4 UTF-8 CRLF Python 3.10.2 64-bit 0.

Como se puede apreciar es posible utilizar el comportamiento de una superclase sin definir nada en la subclase.

Respecto a las demás subclases como se añaden algunos atributos, podríamos definirlas de esta forma:



The screenshot shows a Visual Studio Code interface with the following details:

- File Explorer:** Shows a folder named "BIBLIODOR" with a message: "NO HAY NINGUNA CARPETA ABIERTA". It includes buttons for "Abrir carpeta", "Si abre una carpeta, se cerrarán todos los editores abiertos actualmente. Para mantenerlos abiertos, Agregar una carpeta en su lugar.", and "Clonar repositorio".
- Terminal:** Shows the command "PS C:\Users\intel>".
- Code Editor:** Displays two Python files: "Alimento.py" and "Libro.py".
- Code Content:**

```
19
20 class Alimento(Producto):
21     productor = ""
22     distribuidor = ""
23
24     def __str__(self):
25         return f"REFERENCIA\t {self.referencia}\n" \
26                f"OMBRE\t {self.nombre}\n" \
27                f"PVP\t {self.pvp}\n" \
28                f"DESCRIPCIÓN\t {self.descripcion}\n" \
29                f"PRODUCTOR\t {self.productor}\n" \
30                f"DISTRIBUIDOR\t {self.distribuidor}\n"
31
32
33 class Libro(Alimento):
34     isbn = ""
35     autor = ""
36
37     def __str__(self):
38         return f"REFERENCIA\t {self.referencia}\n" \
39                f"OMBRE\t {self.nombre}\n" \
40                f"PVP\t {self.pvp}\n" \
41                f"DESCRIPCIÓN\t {self.descripcion}\n" \
42                f"ISBN\t {self.isbn}\n" \
43                f"AUTOR\t {self.autor}\n"
```
- Status Bar:** Shows "Lín. 37, col. 23 Espacio: 4 UTF-8 CRLF Python 3.10.2 64-bit".

Ahora para utilizarlas simplemente tendríamos que establecer los atributos después de crear los objetos:

The screenshot shows a Visual Studio Code interface with the following details:

- File Explorer:** Shows a folder structure with files: Jugador.py, Herencia.py (the active file), Vehiculo.py, Electronicos.py, LibrAuto.py, and Series.py.
- Code Editor:** Displays the content of the `Herencia.py` file. The code defines a `__str__` method for a `Alimento` class and creates instances of `Alimento` and `Libro`.
- Terminal:** Shows the command `PS C:\Users\intel> & E:/Python/python.exe "e:/Python Tareas/Tarea1/Herencia.py"` and the resulting output:

REFERENCIA	2034
NOMBRE	Vaso adornado
PVP	15
DESCRIPCIÓN	Vaso de porcelana

REFERENCIA	2035
NOMBRE	Botella de Aceite de Oliva
PVP	5
DESCRIPCIÓN	250 ML
PRODUCTOR	La Aceitera
DISTRIBUIDOR	Distribuciones SA

REFERENCIA	2036
NOMBRE	Cocina Mediterránea
PVP	9
DESCRIPCIÓN	Recetas sanas y buenas
ISBN	0-123456-78-9
AUTOR	Doña Marcela
- Output Panel:** Shows the results of the terminal command, indicating the execution of the Python script.

10. Defina los siguientes:

- Que es una Clase**

Una clase es la descripción de un conjunto de objetos similares; consta de métodos y de datos que resumen las características comunes de dicho conjunto. En un lenguaje de programación orientada a objetos se pueden definir muchos objetos de la misma clase de la misma forma que, en la vida real, haríamos galletas (objeto) con el mismo molde (clase) solo que, para entenderlo mejor, cada galleta tendría igual forma pero es posible que tenga distinto sabor, textura, olor, color, etc.

- Que es un Objeto**

Un objeto es una instancia de una clase. La clase puede tener un método initialize llamado constructor que se va a llamar cada vez que se cree un objeto de esa clase. El constructor se utiliza generalmente para inicializar los atributos de los objetos.

- Defina que es una instancia**

Se llama instancia a todo objeto que derive de algún otro. De esta forma, todos los objetos son instancias de algún otro, menos la clase Object que es la madre de todas.

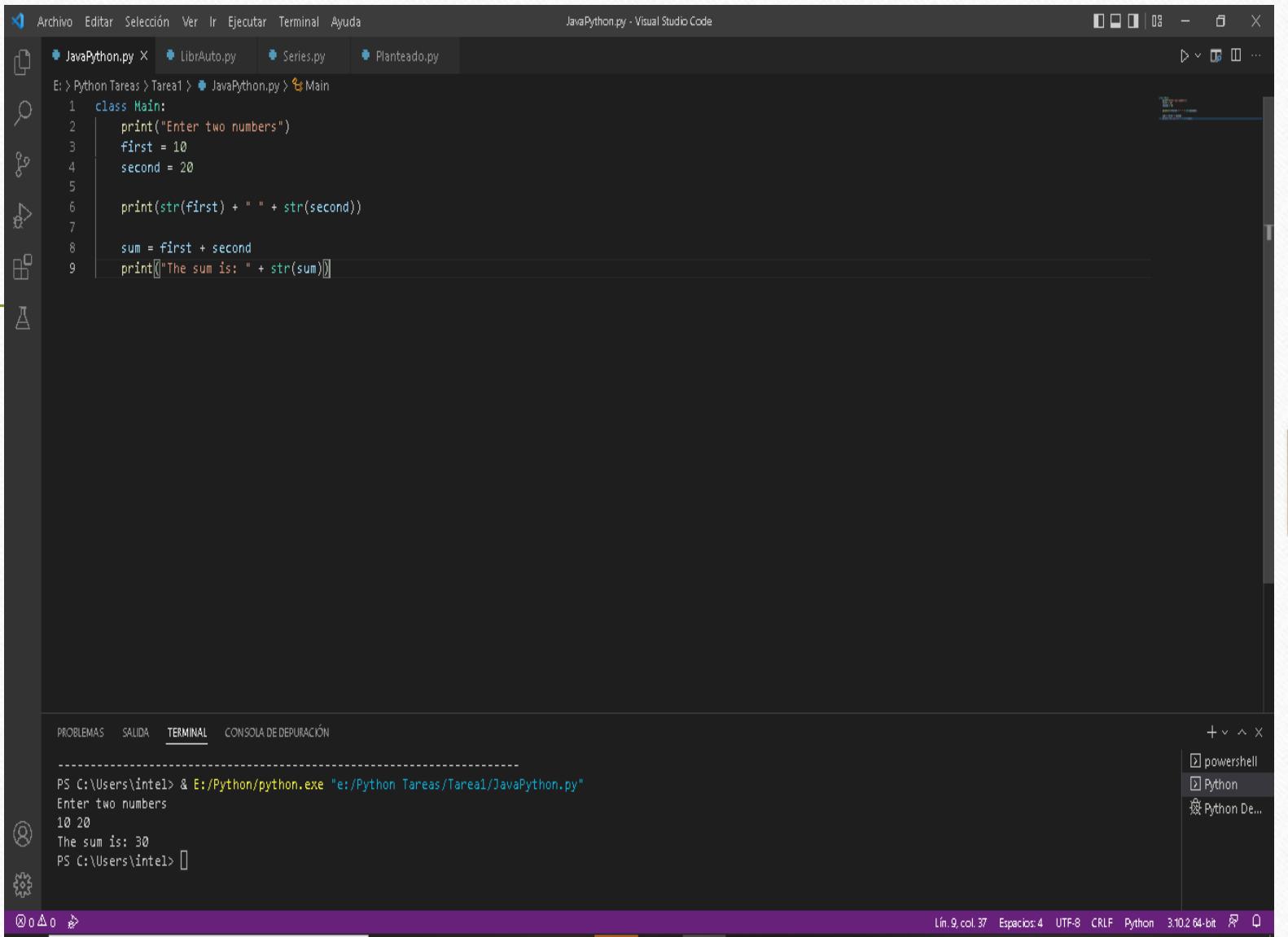
Parte practica

11. Llevar el siguiente código JAVA a Python

```
class Main {  
  
    public static void main(String[] args) {  
  
        System.out.println("Enter two numbers");  
        int first = 10;  
        int second = 20;  
  
        System.out.println(first + " " + second);  
  
        // add two numbers  
        int sum = first + second;  
        System.out.println("The sum is: " + sum);  
    }  
}
```

- Adjuntar el código python generado.
- Adjuntar la imagen(captura) del correcto funcionamiento.

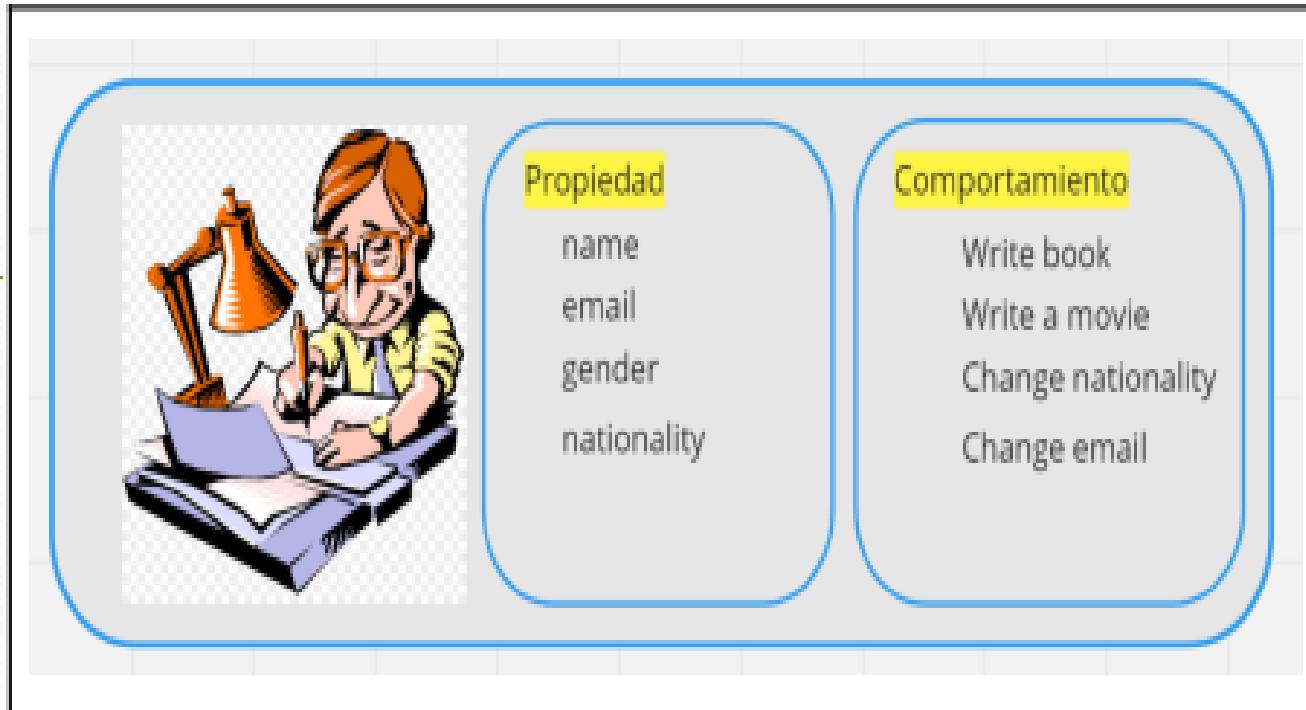
```
class Main:  
    print("Enter two numbers")  
    first = 10  
    second = 20  
  
    print(str(first) + " " + str(second))  
  
    sum = first + second  
    print("The sum is: " + str(sum))
```



The screenshot shows a Visual Studio Code interface with the following details:

- Editor:** The main editor window displays the Python script `Main.py`. The code defines a class `Main` with a constructor that prints "Enter two numbers", initializes `first` to 10 and `second` to 20, and then prints the sum of `first` and `second`.
- Terminal:** The terminal at the bottom shows the output of running the script in a PowerShell window. It prompts for input ("Enter two numbers"), receives "10 20", and then prints the sum ("The sum is: 30").
- Sidebar:** The sidebar on the right lists other files in the workspace: `JavaPython.py`, `LibrAuto.py`, `Series.py`, and `Planteado.py`.
- Status Bar:** The status bar at the bottom indicates the file is `JavaPython.py`, the encoding is `UTF-8`, and the line count is `3102 64-bit`.

12. Crear el código JAVA y Python para el siguiente análisis



- Adjuntar el código python generado.
- Adjuntar la imagen(captura) del correcto funcionamiento.

```
class Persona():
    nombre = None
    email = None
    genero = None
    nacionalidad = None

    def __init__(self, nombre, email, genero, nacionalidad):
        self.nombre = nombre
        self.email = email
        self.genero = genero
        self.nacionalidad = nacionalidad

    def __str__(self):
        return f'\nNombre: {self.nombre}, \nEmail: {self.email}, \nGenero: {self.genero}, \nNacionalidad: {self.nacionalidad}'

Per1 = Persona('JuanManuel', 'JuanManuel@gmail.com', 'Masculino', 'Boliviano')
print(Per1)

class Libro():
    Write_book = None
    Write_a_movie = None
    Change_Nacionality = None
    Change_Email = None

    def __init__(self, libro, pelicula, cambiarNacionalidad, cambiarEmail):
        self.Write_book = libro
        self.Write_a_movie = pelicula
        self.Change_Nacionality = cambiarNacionalidad
        self.Change_Email = cambiarEmail

    def __str__(self):
        return f'\nLibro: {self.Write_book}, \nPelicula: {self.Write_a_movie}, \nChange_Nacionality: {self.Change_Nacionality}, \nChange_Email: {self.Change_Email}'

libr1 = Libro('Señor de los Anillos', 'AssasingCreed', 'Español', 'JuanManuelTolaZabaleta@gmail.com')
print(libr1)
```

Archivo Editar Selección Ver Ir Ejecutar Terminal Ayuda PropiedadComportamiento.py - Visual Studio Code

E: > Python Tareas > Tarea1 > PropiedadComportamiento.py > ...

```
1  from pyexpat import native_encoding
2
3
4  class Persona():
5      nombre = None
6      email = None
7      genero = None
8      nacionalidad = None
9
10 def __init__(self, nombre, email, genero, nacionalidad):
11     self.nombre = nombre
12     self.email = email
13     self.genero = genero
14     self.nacionalidad = nacionalidad
15
16 def __str__(self):
17     return f'\nNombre: {self.nombre}, \nEmail: {self.email}, \nGenero: {self.genero}, \nNacionalidad: {self.nacionalidad}'
18
19
20
21 Per1 = Persona('JuanManuel', 'JuanManuel@gmail.com', 'Masculino', 'Boliviano')
22 print(Per1)
23
24 class Libro():
25     Write_book = None
26     Write_a_movie = None
27     Change_Nacionality = None
```

PROBLEMAS SALIDA TERMINAL CONSOLA DE DEPURACIÓN

Nombre: JuanManuel,
Email: JuanManuel@gmail.com,
Genero: Masculino,
Nacionalidad: Boliviano

Libro: Señor de los Anillos,
Película: AssasinsCreed,
Change_Nacionality: Español,
Change_Email: JuanManuelTolaZabaleta@gmail.com
PS C:\Users\intel> []

Lín. 22, col. 12 Espacios: 4 UTF-8 CRLF Python 3.10.2 64-bit

13. Crear un programa Python que genere los primeros N números de la serie Fibonacci.

- El programa tiene que leer un valor por consola.

 - **Ejem: N = 8**
- Para el valor leído anteriormente, la salida debería ser:
 - **0,1,1,2,3,5,8,13,**
- Adjuntar el código Python generado.
- Adjuntar la imagen(captura) del correcto funcionamiento.

```

def fibonacci(num):
    arr = [0, 1]
    if num == 1:
        print('0')
    elif num == 2:
        print('[0, 1]')
    else:
        while (len(arr) < num):
            arr.append(0)
        if (num == 0 or num == 1):
            return 1
        else:
            arr[0] = 0
            arr[1] = 1
            for i in range(2, num):
                arr[i] = arr[i - 1] + arr[i - 2]
            print(arr)

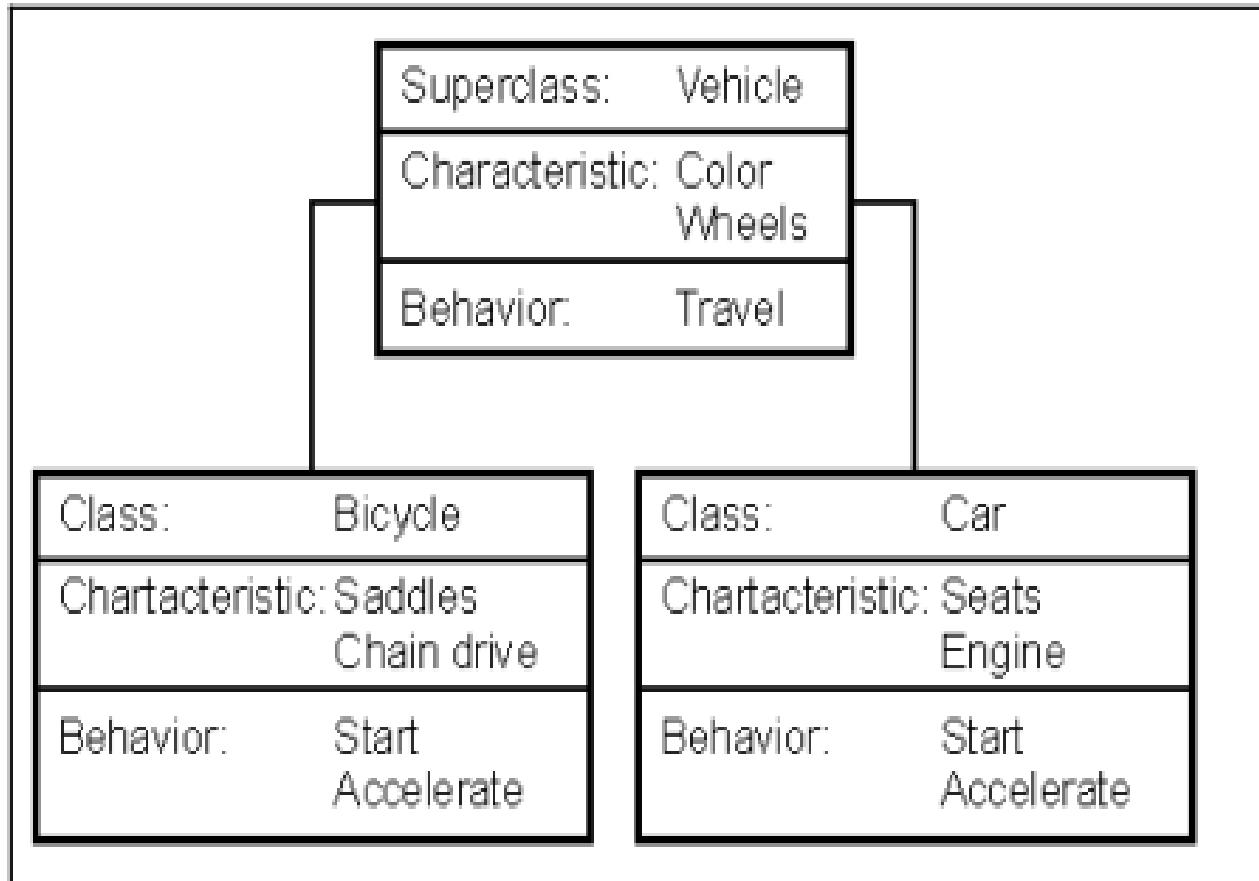
```

`fibonacci(num = int(input("Ingrese el valor de N: ")))`

The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** Shows files like Jugador.py, Vehiculo.py, Electronicos.py, LibrAuto.py, and Series.py.
- Code Editor:** Displays the `Series.py` code. The code defines a `fibonacci` function that handles base cases (0 and 1), prints them directly, and then uses a loop to calculate subsequent Fibonacci numbers by summing the previous two in an array. It also handles the case where `num` is 0 or 1.
- Terminal:** Shows the command `python Series.py` being run, followed by the user input `Ingrese el valor de N: 8`, and the output `[0, 1, 1, 2, 3, 5, 8, 13]`.
- Status Bar:** Shows the file path `E:\Python Tareas>Tarea1>Series.py`, line count `Lín. 19, col. 55`, character count `Espacios: 4`, encoding `UTF-8`, line endings `CRLF`, and Python version `3.10.2 64-bit`.

14. POO - Crear las clases necesarias para resolver el siguiente planteamiento.



- Adjuntar el código python generado.
- Adjuntar la imagen(captura) del correcto funcionamiento.

```
class Vehiculo():
    color = None
    wheels = None
    def __init__(self, color, wheels):
        self.color = color
        self.wheels = wheels

    def __str__(self):
        return "Color Vehiculo: {}, Cantidad Ruedas {}".format( self.color, self.wheels )

class Car(Vehiculo):
    def __init__(self, color, wheels, seats, engine):
        Vehiculo.__init__(self, color, wheels)
        self.seats = seats
        self.engine = engine

    def __start__(self):
        print("Encendiendo Vehiculo")

    def __accelerate__():
        print("Acelerando Vehiculo")

    def __str__(self):
        return Vehiculo.__str__(self) + ", {} km/h, {} cc".format(self.seats, self.engine, self.__start__(), self.__accelerate__())

c = Car("Verde", 4, 120, 1200)
print(c)
Car.__start__
Car.__accelerate__

class Bicycle(Vehiculo):
    def __init__(self, color, wheels, saddles, chain):
        Vehiculo.__init__(self, color, wheels)
        self.saddles = saddles
        self.chain = chain
```

```
def __startb__(self):
    print("Iniciando Bicicleta")

def __accelerateb__(self):
    print("Acelerando bicicleta")

def __str__(self):
    return Vehiculo.__str__(self) + ", {} sillás, {} cond.".format(self.saddles, self.chain, self.__startb__(), self.__accelerateb__())

b = Bicycle("Purpura", 2, 2, 10)
print(b)
Bicycle.__startb__
Bicycle.__accelerateb__
```

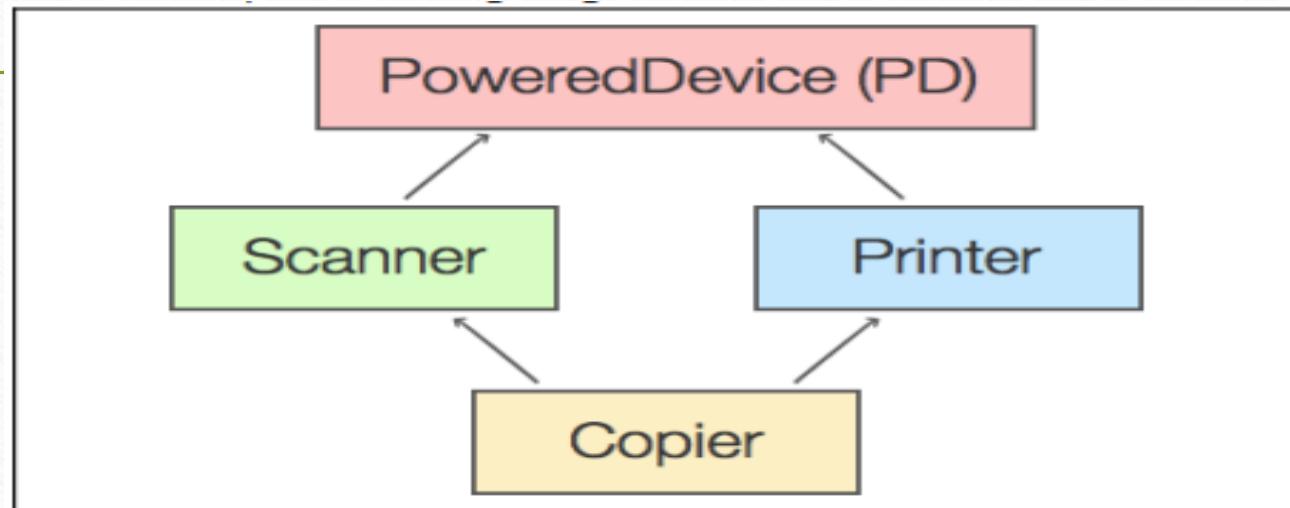
Vehiculo.py - Visual Studio Code

```
E: > Python Tareas > Tarea1 > ● Vehiculo.py > ...
1  class Vehiculo():
2      color = None
3      wheels = None
4      def __init__(self, color, wheels):
5          self.color = color
6          self.wheels = wheels
7
8      def __str__(self):
9          return "Color Vehiculo: {}, Cantidad Ruedas {}".format( self.color, self.wheels )
10
11
12 class Car(Vehiculo):
13
14     def __init__(self, color, wheels, seats, engine):
15         Vehiculo.__init__(self, color, wheels)
16         self.seats = seats
17         self.engine = engine
18
19     def __start__(self):
20         print("Encendiendo Vehiculo")
21
22     def __accelerate__(self):
23         print("Acelerando Vehiculo")
PROBLEMAS SALIDA TERMINAL CONSOLA DE DEPURACIÓN
PS C:\Users\intel> & E:/Python/python.exe "e:/Python Tareas/Tarea1/Vehiculo.py"
Encendiendo Vehiculo
Acelerando Vehiculo
Color Vehiculo: Verde, Cantidad Ruedas 4 , 120 km/h, 1200 cc
Iniciando Bicicleta
Acelerando bicicleta
Color Vehiculo: Purpura, Cantidad Ruedas 2 , 2 sillitas, 10 cond.
PS C:\Users\intel>
```

Lín. 52, col. 9 Espacios: 4 UTF-8 CRLF Python 3.10.2 64-bit

15. POO - Crear las clases necesarias para resolver el siguiente planteamiento.

- En la actualidad tenemos equipos electrónicos como impresoras, scanners, fotocopiadoras, etc. Sin embargo también existen equipos electrónicos multifunción, como por ejemplo es posible tener una impresora y un scanner al mismo tiempo.



- Analizar que cosas características debería de tener cada entidad (clase)
 - Identificar atributos
 - Identificar métodos
 - etc
- Adjuntar el código python generado.
- Adjuntar la imagen(captura) del correcto funcionamiento.

```
class PoweredDevice:  
    _Modelo = None  
    _Serie = "  
  
    def __init__(self, M, S):  
        self._Modelo = M  
        self._Serie = S  
  
    def __str__(self):  
        return f'Modelo: {self._Modelo}\nSerie: {self._Serie}'  
  
    def Proces(self):  
        print("Iniciando Proceso")  
  
class Scanner(PoweredDevice):  
    _Tamaño = None  
  
    def __init__(self, _Modelo, _Serie, _T):  
        PoweredDevice.__init__(self, _Modelo, _Serie)  
        self._Tamaño = _T  
  
class Printer(PoweredDevice):  
    _capacidad = None  
  
    def __init__(self, _Modelo, _Serie, _Capacidad):  
        PoweredDevice.__init__(self, _Modelo, _Serie)  
        self._capacidad = _Capacidad  
  
class Copier(Printer):  
    _Fax = None  
  
    def __init__(self, _Modelo, _Serie, _Tamaño, _capacidad, _F):  
        Scanner.__init__(self, _Modelo, _Serie, _Tamaño)  
        Printer.__init__(self, _Modelo, _Serie, _capacidad)  
        self._Fax = _F  
    def __str__(self):  
        return f'Modelo: {self._Modelo}\nSerie: {self._Serie}\nTamaño: {self._Tamaño}\nCapacidad: {self._capacidad}\nFax: {self._Fax}{self.Proces()}'  
  
Prt = Copier("L735", "1000", "Industrial jj", "Ea", "Si")  
print(Prt)
```

Archivo Editar Selección Ver Ir Ejecutar Terminal Ayuda

Electronics.py - Visual Studio Code

E: > Python Tareas > Tarea1 > Electronics.py > Scanner > __init__

```
1  class PoweredDevice:
2      _Modelo = None
3      _Serie = ''
4
5      def __init__(self, M, S):
6          self._Modelo = M
7          self._Serie = S
8
9      def __str__(self):
10         return f'Modelo: {self._Modelo}\nSerie: {self._Serie}'
11
12     def Proces(self):
13         print("Iniciando Proceso")
14
15 class Scanner(PoweredDevice):
16     _Tamaño = None
17
18     def __init__(self, _Modelo, _Serie, _T):
19         PoweredDevice.__init__(self, _Modelo, _Serie)
20         self._Tamaño = _T
21
22 class Printer(PoweredDevice):
23     _capacidad = None
24
25     def __init__(self, _Modelo, _Serie, _Capacidad):
26         PoweredDevice.__init__(self, _Modelo, _Serie)
27         self._capacidad = _Capacidad
```

PROBLEMAS SALIDA TERMINAL CONSO LA DEPURACIÓN

PS C:\Users\intel> & E:/Python/python.exe "e:/Python Tareas/Tarea1/Problema.py"

Iniciando Proceso

Modelo: L735

Serie: 1000

Tamaño: Industrial jj

Capacidad: Ea

Fax: SiNone

PS C:\Users\intel>

+ ✓ ^ ×

powershell

Python

Python De...

Lín. 19, col. 54 Espacios: 4 UTF-8 CRLF Python 3.10.2 64-bit

16. Ejercicio de planteamiento.

- Identificar un problema cualquiera del mundo real.
- Mostrar el uso de encapsulación.
- Mostrar el usos de herencia simple.
- Mostrar el usos de herencia múltiple.
 - Adjuntar el código Python generado.
 - Adjuntar la imagen (captura) del correcto funcionamiento.
 - Sugerencia: Usar parámetros de tipo string, integer y array-

```
class PedirComida:  
    _Comida = None  
    _Precio = ""  
  
    def __init__(self, M, S):  
        self._Comida = M  
        self._Precio = S  
  
    def __str__(self):  
        return f'Modelo: {self._Comida}\nPrecio: {self._Precio}'  
  
    def Proces(self):  
        print("Iniciando Pedido")  
  
class TamañoC(PedirComida):  
    _Tamaño = None  
  
    def __init__(self, _Comida, _Precio, _T):  
        PedirComida.__init__(self, _Comida, _Precio)  
        self._Tamaño = _T  
  
class Destino(PedirComida):  
    _direccion = None  
  
    def __init__(self, _Comida, _Serie, _Capacidad):  
        PedirComida.__init__(self, _Comida, _Serie)  
        self._direccion = _Capacidad  
  
class PedidoF(Destino):  
    _Recibo = None  
  
    def __init__(self, _Comida, _Precio, _Tamaño, _capacidad, _F):  
        TamañoC.__init__(self, _Comida, _Precio, _Tamaño)  
        Destino.__init__(self, _Comida, _Precio, _capacidad)  
        self._Recibo = _F  
    def __str__(self):  
        return f'Comida: {self._Comida}\nPrecio: {self._Precio}\nTamaño: {self._Tamaño}\nDireccion:  
{self._direccion}\nRecibo: {self._Recibo}{self.Proces()}'  
  
Prt = PedidoF("Pizza", "150", "Familiar", "Ciudad Satelite", "Si")  
print(Prt)
```

Archivo Editar Selección Ver Ir Ejecutar Terminal Ayuda

Problema.py - Visual Studio Code

E: > Python Tareas > Tarea1 > Problema.py > ...

```
25     def __init__(self, _Comida, _Serie, _Capacidad):
26         PedirComida.__init__(self, _Comida, _Serie)
27         self._direccion = _Capacidad
28
29     class PedidoF(Destino):
30         _Recibo = None
31
32         def __init__(self, _Comida, _Precio, _Tamaño, _capacidad, _F):
33             TamañoC.__init__(self, _Comida, _Precio, _Tamaño)
34             Destino.__init__(self, _Comida, _Precio, _capacidad)
35             self._Recibo = _F
36         def __str__(self):
37             return f'Comida: {self._Comida}\nPrecio: {self._Precio}\nTamaño: {self._Tamaño}\nDireccion: {self._direccion}\nRecibo: {self._Recibo}({self.Proces()})'
38
39 Prt = PedidoF("Pizza", "150", "Familiar", "Ciudad Satelite", "Si")
40 print(Prt)
```

PROBLEMAS SALIDA TERMINAL CON SOLA DE DEPURACIÓN

PS C:\Users\intel> & E:/Python/python.exe "e:/Python Tareas/Tarea1/Problema.py"

Iniciando Pedido

Comida: Pizza

Precio: 150

Tamaño: Familiar

Direccion: Ciudad Satelite

Recibo: SiNone

PS C:\Users\intel>

+ ✓ ×

powershell

Python

Python Dev...

Lín. 40, col. 11 Espacios: 4 UTF-8 CRLF Python 3.10.2 64-bit

ENLACE DE DRIVE PARA EL VIDEO

<https://drive.google.com/file/d/1vP9cZ-zDE-l5Ade5goPCxXKVA58Sa-QD/view?usp=sharing>
