



Universidad de las Fuerzas Armadas ESPE  
Unidad de Educación a Distancia

---



## INGENIERÍA EN TECNOLOGÍAS DE LA INFORMACIÓN

**ASIGNATURA:** Programación Orienta a Objetos

**NRC:** 16362

**DOCENTE:** Jaramillo Montaña Luis Enrique

**ESTUDIANTE:** Fajardo Collaguazo Roberto Manuel

**PERIODO:** 2023 -2024

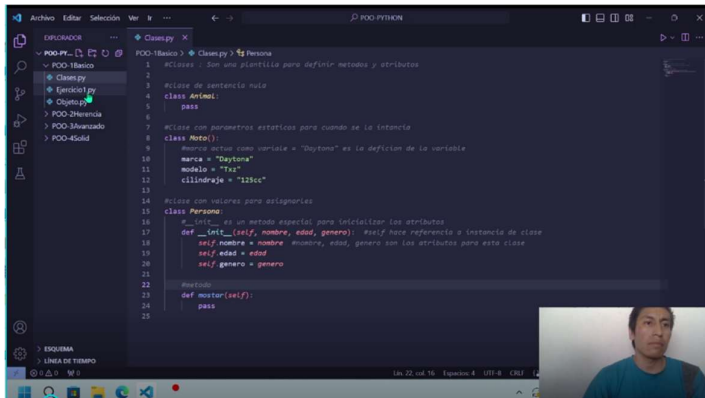
**TEMA:** POO CON PYTHON

1. Link de enlace: <https://github.com/FajardoRM/POO-Java.git>

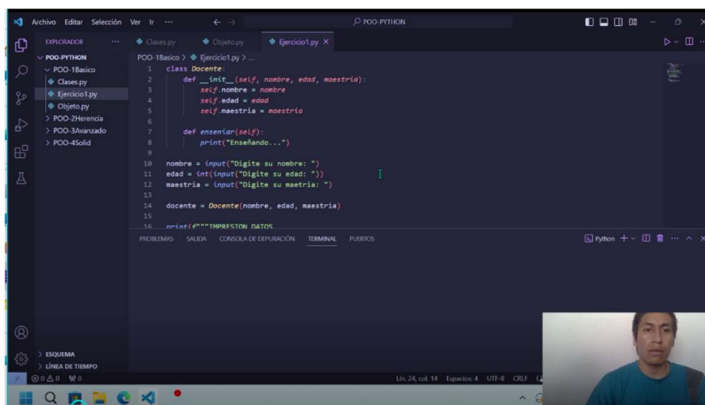
## 2. Capturas de Pantalla del Código POO de Python

---- LO BÁSICO DE POO ----

Clases y objetos:

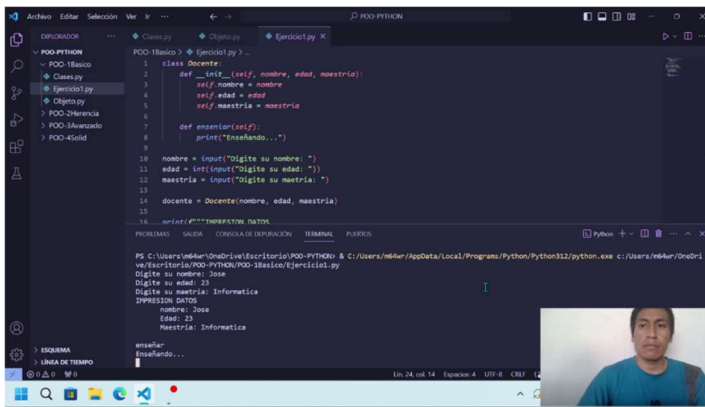


```
1 # Auto.py
2 # Auto es una plantilla para definir metodos y atributos
3
4 # Auto es una plantilla para definir metodos y atributos
5
6 # Auto es una plantilla para definir metodos y atributos
7
8 # Auto es una plantilla para definir metodos y atributos
9
10 # Auto es una plantilla para definir metodos y atributos
11
12 # Auto es una plantilla para definir metodos y atributos
13
14 # Auto es una plantilla para definir metodos y atributos
15
16 # Auto es una plantilla para definir metodos y atributos
17
18 # Auto es una plantilla para definir metodos y atributos
19
20 # Auto es una plantilla para definir metodos y atributos
21
22 # Auto es una plantilla para definir metodos y atributos
23
24 # Auto es una plantilla para definir metodos y atributos
25
```



```
1 # Docente.py
2 # Docente es una plantilla para definir metodos y atributos
3
4 # Docente es una plantilla para definir metodos y atributos
5
6 # Docente es una plantilla para definir metodos y atributos
7
8 # Docente es una plantilla para definir metodos y atributos
9
10 # Docente es una plantilla para definir metodos y atributos
11
12 # Docente es una plantilla para definir metodos y atributos
13
14 # Docente es una plantilla para definir metodos y atributos
15
16 # Docente es una plantilla para definir metodos y atributos
17
18 # Docente es una plantilla para definir metodos y atributos
19
20 # Docente es una plantilla para definir metodos y atributos
21
22 # Docente es una plantilla para definir metodos y atributos
23
24 # Docente es una plantilla para definir metodos y atributos
25
```

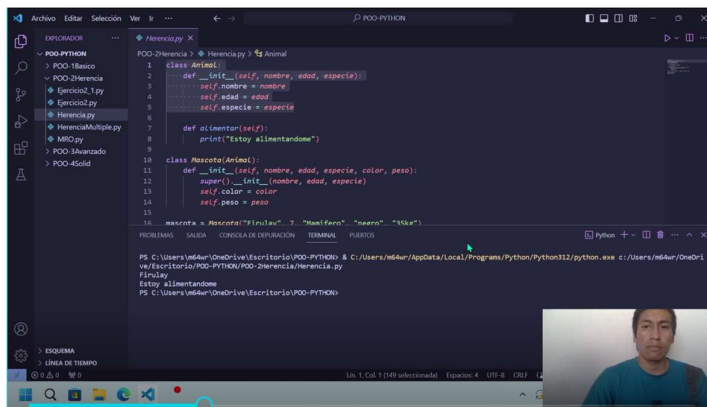
Ejercicio 1:



```
1 # Docente.py
2 # Docente es una plantilla para definir metodos y atributos
3
4 # Docente es una plantilla para definir metodos y atributos
5
6 # Docente es una plantilla para definir metodos y atributos
7
8 # Docente es una plantilla para definir metodos y atributos
9
10 # Docente es una plantilla para definir metodos y atributos
11
12 # Docente es una plantilla para definir metodos y atributos
13
14 # Docente es una plantilla para definir metodos y atributos
15
16 # Docente es una plantilla para definir metodos y atributos
17
18 # Docente es una plantilla para definir metodos y atributos
19
20 # Docente es una plantilla para definir metodos y atributos
21
22 # Docente es una plantilla para definir metodos y atributos
23
24 # Docente es una plantilla para definir metodos y atributos
25
```

---- ENTENDIENDO HERENCIA ----

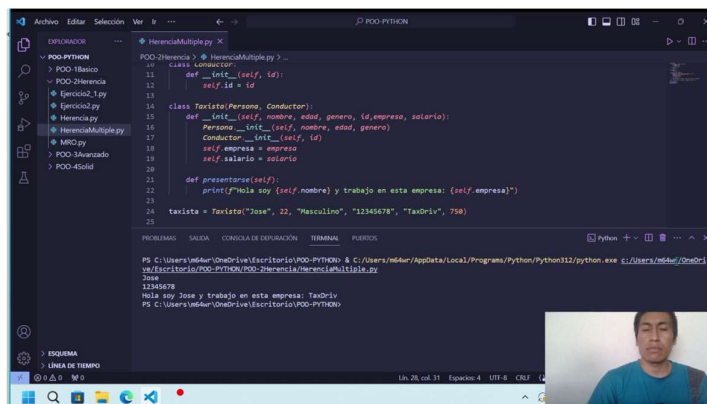
## Herencia:



```
1 class Animal:
2     def __init__(self, nombre, edad, especie):
3         self.nombre = nombre
4         self.edad = edad
5         self.especie = especie
6
7     def alimentarse(self):
8         print("Estoy alimentandome")
9
10 class Mascota(Animal):
11     def __init__(self, nombre, edad, especie, color, peso):
12         super().__init__(nombre, edad, especie)
13         self.color = color
14         self.peso = peso
15
16 mascota = Mascota("Fifi", 3, "Mascota", "rojo", "15kg")
17
18 mascota.alimentarse()
```

PS C:\Users\mdur\OneDrive\Escritorio\POO-PYTHON> python.exe c:\Users\mdur\OneDrive\Escritorio\POO-PYTHON\Herencia.py  
Estoy alimentandome  
PS C:\Users\mdur\OneDrive\Escritorio\POO-PYTHON>

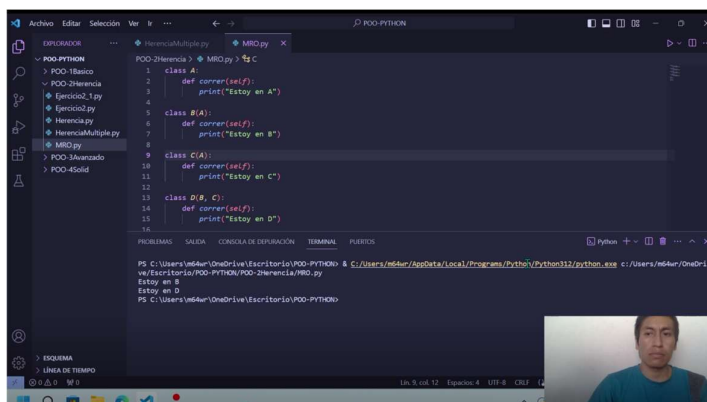
## Herencia multiple:



```
10 class Conductor:
11     def __init__(self, id):
12         self.id = id
13
14 class Taxista(Persona, Conductor):
15     def __init__(self, nombre, edad, genero, id, empresa, salario):
16         Persona.__init__(self, nombre, edad, genero)
17         Conductor.__init__(self, id)
18         self.empresa = empresa
19         self.salario = salario
20
21     def presentarse(self):
22         print(f"Hola soy {self.nombre} y trabajo en esta empresa: {self.empresa}")
23
24 taxista = Taxista("Jose", 22, "Masculino", "12345678", "Taxista", 750)
25
26 taxista.presentarse()
```

PS C:\Users\mdur\OneDrive\Escritorio\POO-PYTHON> python.exe c:\Users\mdur\OneDrive\Escritorio\POO-PYTHON\HerenciaMultiple.py  
Hola soy Jose y trabajo en esta empresa: Taxista  
PS C:\Users\mdur\OneDrive\Escritorio\POO-PYTHON>

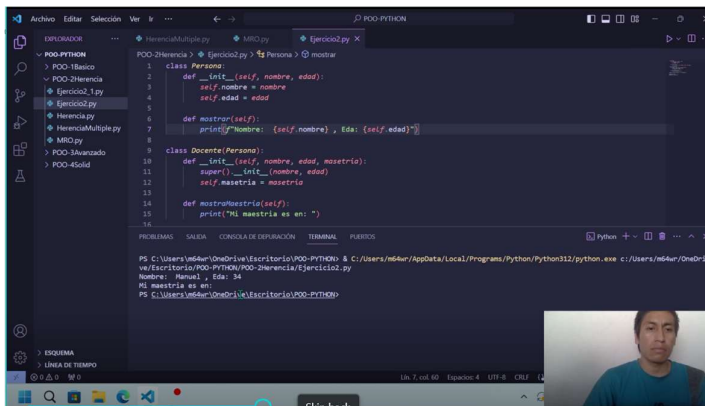
## MRO (Méthod Resolution Order)



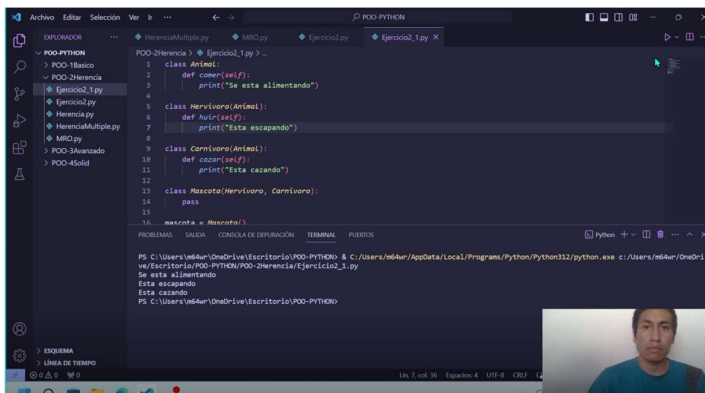
```
1 class A:
2     def correr(self):
3         print("Estoy en A")
4
5 class B(A):
6     def correr(self):
7         print("Estoy en B")
8
9 class C(A):
10     def correr(self):
11         print("Estoy en C")
12
13 class D(B, C):
14     def correr(self):
15         print("Estoy en D")
16
17 D.correr()
```

PS C:\Users\mdur\OneDrive\Escritorio\POO-PYTHON> python.exe c:\Users\mdur\OneDrive\Escritorio\POO-PYTHON\MRO.py  
Estoy en D  
PS C:\Users\mdur\OneDrive\Escritorio\POO-PYTHON>

## Ejercicio 2:



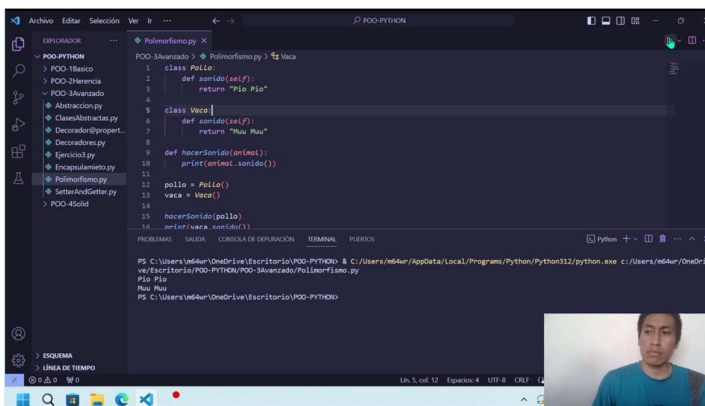
```
1 class Persona:
2     def __init__(self, nombre, edad):
3         self.nombre = nombre
4         self.edad = edad
5
6     def mostrar(self):
7         print(f"Nombre: {self.nombre}, Edad: {self.edad}")
8
9 class Docente(Persona):
10     def __init__(self, nombre, edad, materia):
11         super().__init__(nombre, edad)
12         self.materia = materia
13
14     def mostrarMateria(self):
15         print(f"La materia es en: ")
16
17 PS C:\Users\m64ur\OneDrive\Escritorio\POO-PYTHON> python.exe c:\Users\m64ur\OneDrive\Escritorio\POO-PYTHON\ejercicio2.py
Nombre: Manuel, Edad: 34
La materia es en:
```



```
1 class Animal:
2     def __init__(self, nombre, edad):
3         self.nombre = nombre
4         self.edad = edad
5
6     def mostrar(self):
7         print(f"Nombre: {self.nombre}, Edad: {self.edad}")
8
9 class Mascota(Animal):
10     def __init__(self, nombre, edad, mascota):
11         super().__init__(nombre, edad)
12         self.mascota = mascota
13
14     def mostrarMascota(self):
15         print(f"La mascota es en: ")
16
17 PS C:\Users\m64ur\OneDrive\Escritorio\POO-PYTHON> python.exe c:\Users\m64ur\OneDrive\Escritorio\POO-PYTHON\ejercicio2.py
Nombre: Manuel, Edad: 34
La mascota es en:
```

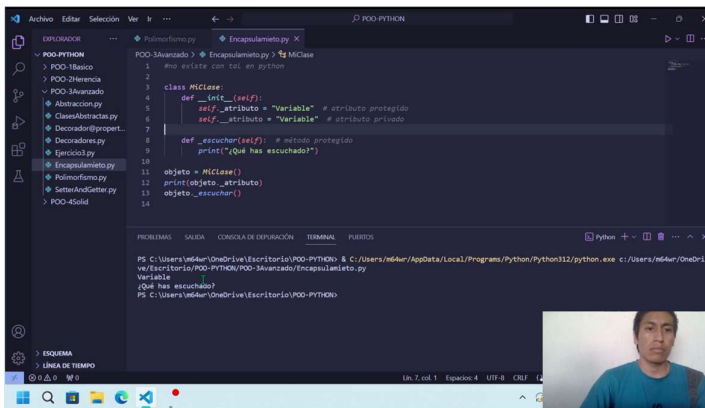
## --- CONCEPTOS AVANZADOS ---

### Polimorfismo:



```
1 class Pollo:
2     def __init__(self, nombre, edad):
3         self.nombre = nombre
4         self.edad = edad
5
6     def mostrar(self):
7         print(f"Nombre: {self.nombre}, Edad: {self.edad}")
8
9 class Vaca(Pollo):
10     def __init__(self, nombre, edad, mascota):
11         super().__init__(nombre, edad)
12         self.mascota = mascota
13
14     def mostrarMascota(self):
15         print(f"La mascota es en: ")
16
17 PS C:\Users\m64ur\OneDrive\Escritorio\POO-PYTHON> python.exe c:\Users\m64ur\OneDrive\Escritorio\POO-PYTHON\ejercicio2.py
Nombre: Manuel, Edad: 34
La mascota es en:
```

### Encapsulamiento:



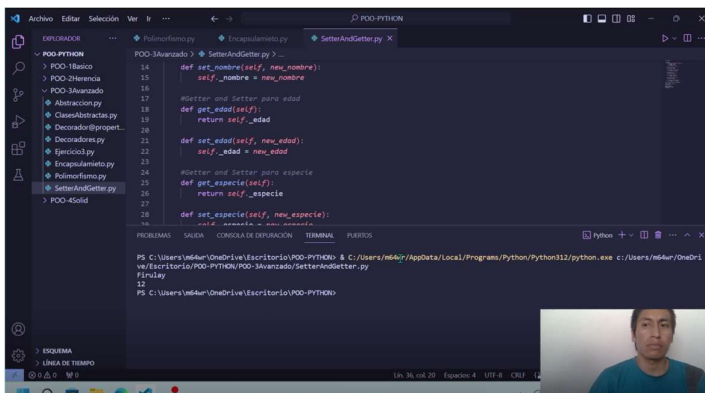
The screenshot shows a VS Code editor with a file named `Encapsulamiento.py`. The code defines a class `MCClass` with a decorator `@property` applied to a method `escuchar`. The terminal output shows the execution of the script, which prints the output of the `escuchar` method.

```
1 # No existe con los en python
2
3 class MCClass:
4     def __init__(self):
5         self._atributo = "Variable" # atributo protegido
6         self.__atributo = "Variable" # atributo privado
7
8     def _escuchar(self): # método protegido
9         print("¿Qué has escuchado?")
10
11 objeto = MCClass()
12 print(objeto._atributo)
13 objeto._escuchar()
14
```

Terminal output:

```
PS C:\Users\mdur\OneDrive\Escritorio\POO-PYTHON> & C:\Users\mdur\AppData\Local\Programs\Python\Python312\python.exe c:\Users\mdur\OneDrive\Escritorio\POO-PYTHON\POO-Avanzado\Encapsulamiento.py
Variable
¿Qué has escuchado?
```

## Getter y Setter:



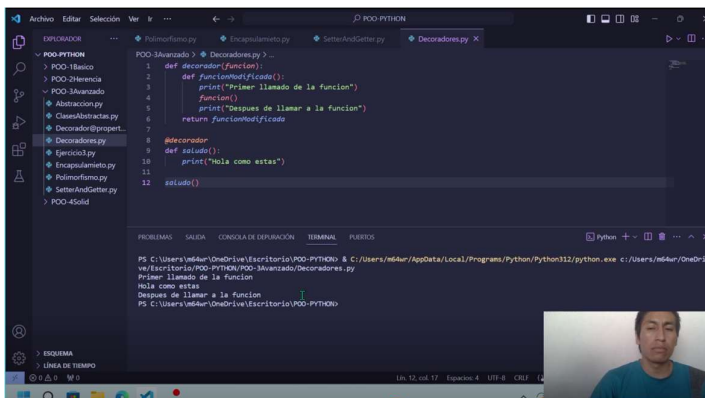
The screenshot shows a VS Code editor with a file named `SettersAndGetters.py`. The code defines a class `MCClass` with two properties: `edad` and `especie`. Each property has a getter and a setter method. The terminal output shows the execution of the script, which prints the output of the `get_edad` and `set_edad` methods.

```
16 def get_edad(self, new_edad):
17     self._edad = new_edad
18
19 # Getter and Setter para edad
20 def get_edad(self):
21     return self._edad
22
23 def set_edad(self, new_edad):
24     self._edad = new_edad
25
26 # Getter and Setter para especie
27 def get_especie(self):
28     return self._especie
29
30 def set_especie(self, new_especie):
31     self._especie = new_especie
32
```

Terminal output:

```
PS C:\Users\mdur\OneDrive\Escritorio\POO-PYTHON> & C:\Users\mdur\AppData\Local\Programs\Python\Python312\python.exe c:\Users\mdur\OneDrive\Escritorio\POO-PYTHON\POO-Avanzado\SettersAndGetters.py
19
20
21
22
23
24
25
26
27
28
29
30
31
32
```

## Decoradores:



The screenshot shows a VS Code editor with a file named `Decoradores.py`. The code defines a decorator `@decorador(funcion)` and a function `saludo`. The terminal output shows the execution of the script, which prints the output of the `saludo` function.

```
1 def decorador(funcion):
2     def func_wrapper():
3         print("Primer llamado de la funcion")
4         funcion()
5         print("Después de llamar a la funcion")
6         return funcion()
7
8     return func_wrapper
9
10 @decorador
11 def saludo():
12     print("Hola como estas")
13
14 saludo()
15
```

Terminal output:

```
PS C:\Users\mdur\OneDrive\Escritorio\POO-PYTHON> & C:\Users\mdur\AppData\Local\Programs\Python\Python312\python.exe c:\Users\mdur\OneDrive\Escritorio\POO-PYTHON\POO-Avanzado\Decoradores.py
Primer llamado de la funcion
Hola como estas
Después de llamar a la funcion
```

## Decoradores @property

```

1 class Alumno:
2     def __init__(self, nombre, edad):
3         self._nombre = nombre
4         self._edad = edad
5
6     @property
7     def nombre(self):
8         return self._nombre
9
10    @nombre.setter
11    def nombre(self, nuevo_nombre):
12        self._nombre = nuevo_nombre
13
14    @property
15    def edad(self):
16        return self._edad
17
18    @edad.setter
19    def edad(self, nueva_edad):
20        self._edad = nueva_edad
21
22
23
24
25
26

```

PS C:\Users\m64ur\OneDrive\Escritorio\POO-PYTHON> python.exe c:\Users\m64ur\OneDrive\Escritorio\POO-PYTHON\Decorador@property.py

PS C:\Users\m64ur\OneDrive\Escritorio\POO-PYTHON>

Abstracción:

```

1 class Moto:
2     def __init__(self):
3         self._estado = "apagado"
4
5     def encender(self):
6         self._estado = "encendido"
7
8     def conducir(self):
9         if self._estado == "apagado":
10             self._encender()
11         print("Conducir la moto")
12
13 moto = Moto()
14 moto.conducir()
15
16
17
18
19
20

```

PS C:\Users\m64ur\OneDrive\Escritorio\POO-PYTHON> python.exe c:\Users\m64ur\OneDrive\Escritorio\POO-PYTHON\Abstraccion.py

Conducir la moto

PS C:\Users\m64ur\OneDrive\Escritorio\POO-PYTHON>

Clases abstractas:

```

1 class Docente:
2     def __init__(self, nombre, edad, genero, actividad):
3         self._nombre = nombre
4         self._edad = edad
5         self._genero = genero
6         self._actividad = actividad
7
8     def trabajar(self):
9         pass
10
11     def hacerActividad(self):
12         print(f"Actualmente estoy trabajando en el rubro de: {self._actividad}")
13
14 estudiante = Estudiante("Manuel", 27, "Masculino", "Programación")
15 docente = Docente("Rosa", 34, "Femenino", "Pedagogía")
16
17 estudiante.presentar()
18 docente.hacerActividad()
19
20
21
22
23
24

```

PS C:\Users\m64ur\OneDrive\Escritorio\POO-PYTHON> python.exe c:\Users\m64ur\OneDrive\Escritorio\POO-PYTHON\ClasesAbstractas.py

Hola, me llamo Manuel y tengo 27 años

Estoy estudiando: Programación

Hola, me llamo Rosa y tengo 34 años

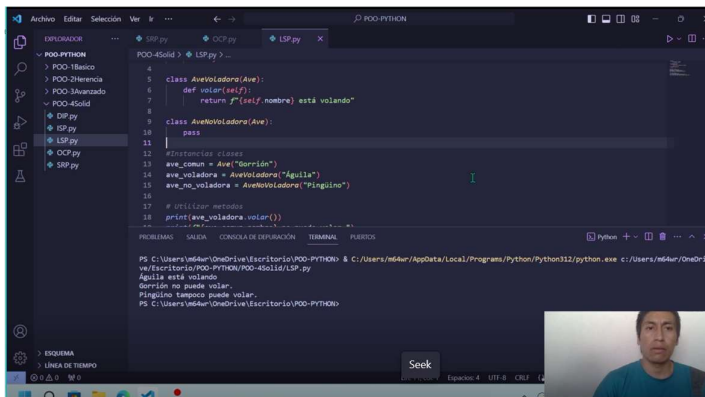
Actualmente estoy trabajando en el rubro de: Pedagogía

PS C:\Users\m64ur\OneDrive\Escritorio\POO-PYTHON>

Ejercicio 3:



## LSP (Liskov's Substitution Principle)



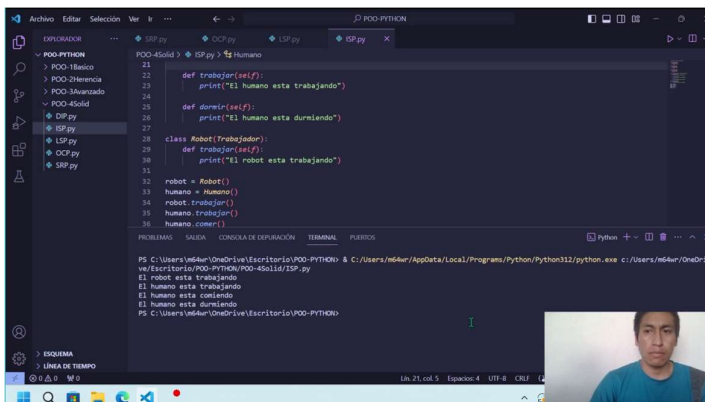
The screenshot shows a VS Code editor with a file named `LSP.py` open. The code defines a base class `AveVoladora(Ave)` with a `volar()` method that returns a string indicating the bird is flying. It also defines a subclass `AveNoVoladora(Ave)` which inherits from `AveVoladora` but does not override the `volar()` method. The terminal shows the execution of the script, which prints the output of the `volar()` method for both classes.

```
1 class AveVoladora(Ave):
2     def volar(self):
3         return f"{self.nombre} está volando"
4
5 class AveNoVoladora(Ave):
6     pass
7
8 # Instancias clases
9 ave_comun = Ave("Gorrión")
10 ave_voladora = AveVoladora("Águila")
11 ave_no_voladora = AveNoVoladora("Pinguino")
12
13 # Utilizar métodos
14 print(ave_voladora.volar())
15
16 # Problemas
```

Terminal output:

```
PS C:\Users\mdur\OneDrive\Escritorio\POO-PYTHON> C:\Users\mdur\AppData\Local\Programs\Python\Python312\python.exe c:/Users/mdur/OneDrive/Escritorio/POO-PYTHON/POO-4Solid/LSP.py
Águila está volando
Gorrión no puede volar.
Pinguino tampoco puede volar.
PS C:\Users\mdur\OneDrive\Escritorio\POO-PYTHON>
```

## ISP (Interface Segregation Principle)



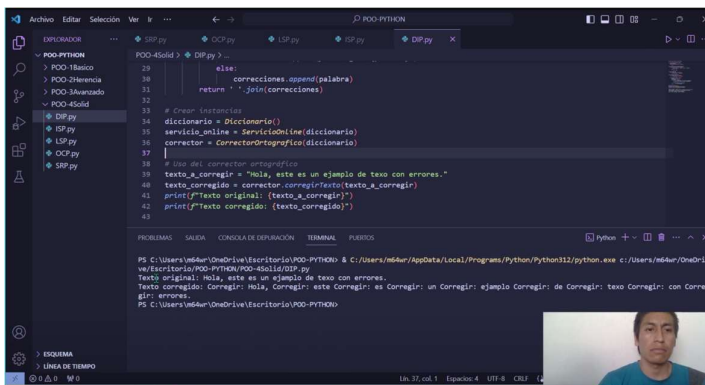
The screenshot shows a VS Code editor with a file named `ISP.py` open. The code defines a base class `Humano` with two methods: `trabajar(self)` and `dormir(self)`. It also defines a subclass `Robot(Humano)` which inherits from `Humano` but only implements the `trabajar()` method. The terminal shows the execution of the script, which prints the output of the `trabajar()` method for both classes.

```
1 class Humano:
2     def trabajar(self):
3         print("El humano está trabajando")
4     def dormir(self):
5         print("El humano está durmiendo")
6
7 class Robot(Humano):
8     def trabajar(self):
9         print("El robot está trabajando")
10
11 robot = Robot()
12 humano = Humano()
13 robot.trabajar()
14 humano.dormir()
15 humano.comer()
16
17 # Problemas
```

Terminal output:

```
PS C:\Users\mdur\OneDrive\Escritorio\POO-PYTHON> C:\Users\mdur\AppData\Local\Programs\Python\Python312\python.exe c:/Users/mdur/OneDrive/Escritorio/POO-PYTHON/POO-4Solid/ISP.py
El robot está trabajando
El humano está trabajando
El humano está durmiendo
El humano está comiendo
PS C:\Users\mdur\OneDrive\Escritorio\POO-PYTHON>
```

## DIP (Dependency Inversion Principle)



The screenshot shows a VS Code editor with a file named `DIP.py` open. The code defines a base class `Corrector` with a `corregir(texto)` method. It also defines a subclass `CorrectorOnline(Corrector)` which inherits from `Corrector` and implements the `corregir()` method. The terminal shows the execution of the script, which prints the output of the `corregir()` method for both classes.

```
1 class Corrector:
2     def corregir(texto):
3         return texto
4
5 class CorrectorOnline(Corrector):
6     def corregir(texto):
7         return texto.replace(" ", " ")
8
9 # Crear instancias
10 diccionario = Diccionario()
11 servicio_online = ServicioOnline(diccionario)
12 corrector = CorrectorOnline(servicio_online)
13
14 # Uso del corrector
15 texto_a_corregir = "Hola, este es un ejemplo de texto con errores."
16 texto_corregido = corrector.corregir(texto_a_corregir)
17 print(f"Texto original: {texto_a_corregir}")
18 print(f"Texto corregido: {texto_corregido}")
19
20 # Problemas
```

Terminal output:

```
PS C:\Users\mdur\OneDrive\Escritorio\POO-PYTHON> C:\Users\mdur\AppData\Local\Programs\Python\Python312\python.exe c:/Users/mdur/OneDrive/Escritorio/POO-PYTHON/POO-4Solid/DIP.py
Texto original: Hola, este es un ejemplo de texto con errores.
Texto corregido: Hola, Corregir: este Corregir: es Corregir: un Corregir: ejemplo Corregir: de Corregir: texto Corregir: con Corre
gir: errores
PS C:\Users\mdur\OneDrive\Escritorio\POO-PYTHON>
```