introduccion.py clase05.py

```
Programming ‘Language’ = {
    Introducción = [Python, Micropython]
}
```

introduccion.py clase05.py

```
Clase 05 = {
    Presentación = [Les damos la bienvenida al curso]
}
```

Clases = {

- 👉 **Clase 05** — **Conceptos Básicos de POO. Clases y objetos. Métodos y atributos.**
- **Clase 06** — Instalación de MicroPython en la placa ESP32. Introducción a las herramientas de desarrollo (Thonny, uPyCraft). Conexión y configuración de la placa.
- **Clase 07** — Control de Hardware Básico. Manejo de pines GPIO. Lectura de sensores y actuadores.
- **Clase 08** — Comunicación Serial. UART, I2C, SPI. Comunicación entre dispositivos.

Prog. Orientada a Objetos {

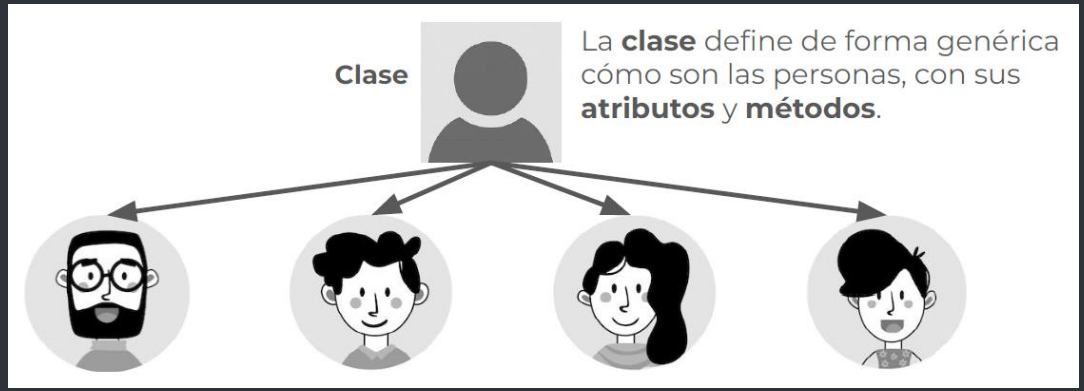
```
    # En el paradigma de programación orientada a objetos (POO) se utilizan
    entidades que representan elementos del problema a resolver y tienen
    atributos y comportamientos (pueden almacenar datos y realizar acciones).
    Estas entidades se denominan objetos, y Python proporciona soporte para
    este paradigma.

    # A diferencia de la programación estructurada, que está centrada en las
    funciones, la programación orientada (POO) se basa en la definición de
    clases y objetos.

    # La programación orientada a objetos surge en los 70s y tiene un gran
    auge en los 90. Uno de los lenguajes destacados de este nuevo paradigma es
    Java.

    # Por supuesto, el concepto de la POO excede a Java y Python ya que se
    aplica a muchos lenguajes.
}
```


Prog. Orientada a Objetos {

```
# Conceptos Clave

# Clase: Es una plantilla para crear objetos. Define un conjunto de atributos y métodos que los objetos creados a partir de la clase tendrán.
# Objeto: Es una instancia de una clase. Cada objeto puede tener diferentes valores para los atributos definidos en la clase.
# Atributo: Son datos que caracterizan al objeto, almacenan datos relacionados con su estado.
# Método: Es una función que pertenece a una clase y define un comportamiento que los objetos de la clase pueden realizar.
# Identidad: Cada objeto tiene una identidad que lo distingue de otros objetos, sin considerar su estado. Por lo general, esta identidad se crea mediante un identificador que deriva naturalmente de un problema (por ejemplo: un producto puede estar representado por un código, un automóvil por un número de modelo, etc).

}

# Otros conceptos que no veremos son Herencia, Encapsulamiento, polimorfismo y colaboración entre clases.
```


Clases {

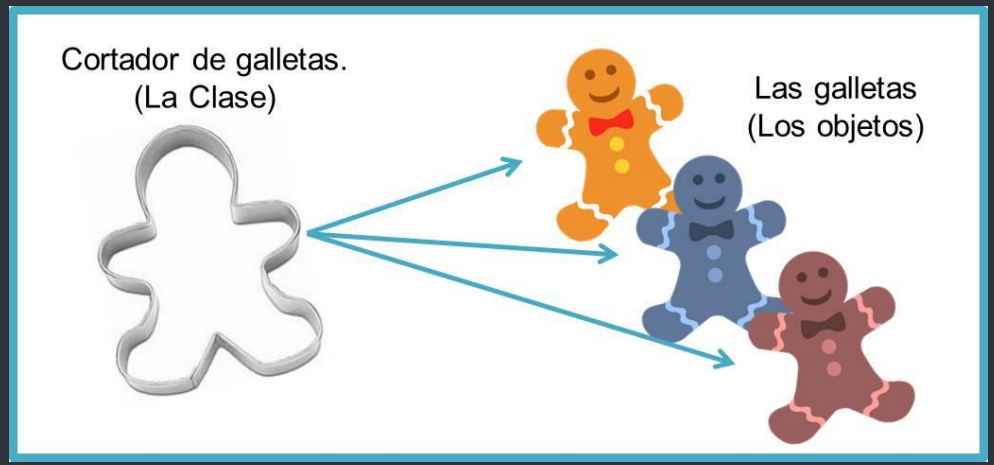
```
# Podemos pensar en las clases como plantillas. Definen de manera genérica
cómo van a ser los objetos de determinado tipo. Por ejemplo, una clase
para representar a las personas puede llamarse Persona y tener una serie
de propiedades como Nombre, Edad o Nro de DNI (similares a variables), y
una serie de comportamientos, como Hablar(), Caminar() o Comer(). Estos
comportamientos se implementan como métodos (similares a funciones). Los
objetos son personas concretas, cada una con sus características propias.
```



}

Clases {

```
# También podemos pensar en las clases como si fuera el molde cortador de
galletas. Todas las galletas recién cortadas con el cortador tendrán la
misma forma. Después las galletas (objetos) podrán o no ser diferentes de
su forma inicial.
```



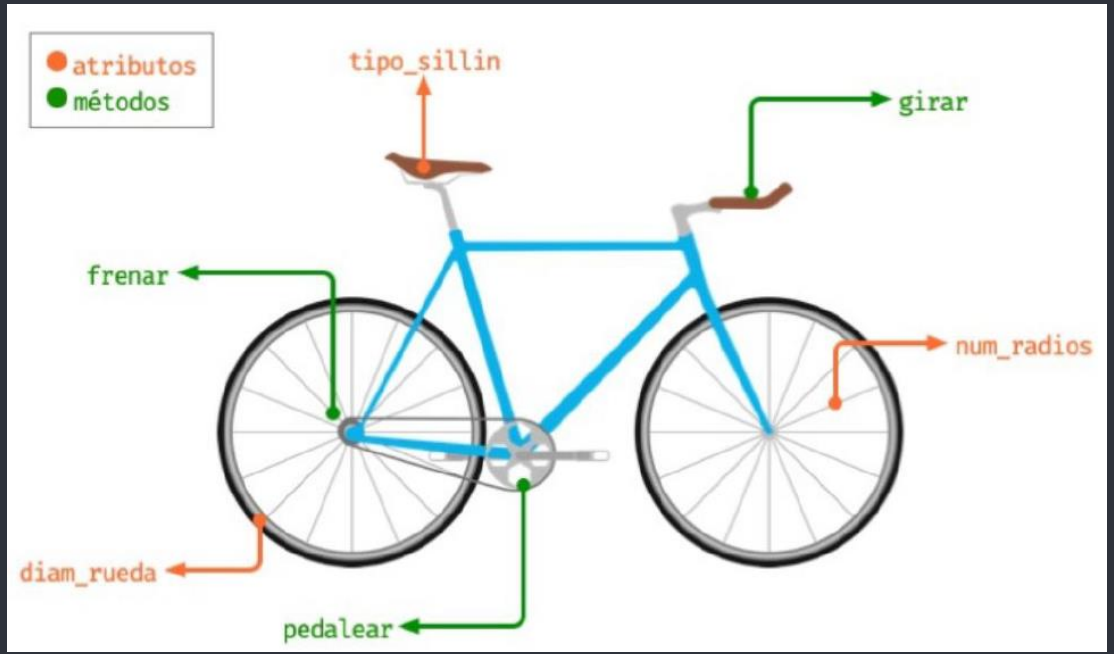
}

Clases {

```
# Una clase no es más que un concepto, sin entidad real. Para poder
utilizarlas en un programa hay que instanciarla, es decir, crear un nuevo
objeto concreto de la misma. Un objeto es una entidad concreta que se crea
a partir de la plantilla que es la clase. Este nuevo objeto tiene
"existencia" real, puesto que ocupa memoria y se puede utilizar en el
programa.

# Así un objeto puede ser una persona que se llama Ivana, de 37 años y DNI
nro 32456822, que en nuestro programa podría hablar, caminar o comer, que
son los comportamientos que están definidos en la clase.

# Una clase equivale a la generalización de un tipo específico de objetos.
Y una instancia es la concreción de una clase en un objeto.
```

}

Ejemplo {


atributos
métodos
tipo_sillin
girar
frenar
num_radios
diam_rueda
pedalear

}

Ejemplo {

```
# Los objetos de la clase Bicicleta comparten atributos y métodos:

# Los atributos tipo_sillin, num_radios y diam_rueda están presentes en
todos los objetos de la clase, pero posiblemente sus valores varían de un
objeto Bicicleta a otro.

# Los métodos frenar, girar y pedalear son compartidos por todas las
instancias que se crean a partir de la clase bicicleta. Pero en cada
instancia se invocan cuando sea necesario: no todas las instancias van a
frenar o acelerar al mismo tiempo.

# Resumiendo, estas características están presentes en todas las
bicicletas creadas a partir de la “plantilla” Bicicleta.
```

}

Definición de una clase {

```python
# Definimos la clase Persona
class Persona:

    # Atributo, presente en todos los objetos que pertenecen a la clase
    piernas = 2

# Instanciamos un objeto de la clase Persona
juan = Persona()

# Imprimimos un atributo del objeto
print(juan.piernas) # 2
```

Clase

Atributo

Instancia de objeto

Notación punto

}

Atributos {

Las variables dentro de la clase (**atributos de clase**) son compartidas por todos los objetos instanciados. Se definen dentro de la clase pero fuera de sus métodos.

Los objetos pueden tener sus propios atributos, llamados **atributos de instancia**. Una manera de crearlos es usar directamente la notación punto:

```python
class Persona:
    # Atributo de clase
    piernas = 2

juan = Persona()

# Atributo de instancia
juan.edad = 34
print(juan.edad) # 34
```

}

Ejemplo 2 {

```python
class Persona:
    piernas = 2

juan = Persona()
manuel = Persona()

print(juan.piernas) # 2
print(manuel.piernas) # 2

juan.edad = 34
manuel.edad = 26

print(juan.edad) # 34
print(manuel.edad) # 26
```

}

Métodos {

```
# Los métodos permiten a los objetos de una clase realizar acciones. Se
declaran con def:, como las funciones, dentro de la clase. Reciben
parámetros y uno de ellos, el primero (self) es obligatorio. self hace
referencia a la instancia propia perteneciente a la clase.

class Persona():
    piernas = 2 # Atributo DE CLASE

    def caminar(self): # Definimos un método
        print("Estoy caminando.")

juan = Persona() # Instanciamos un objeto
juan.caminar() # Invocamos el método caminar()
```

Atributo

Método

Notación punto

}

Ejemplo {

```python
class Persona():
    moviendo = False

    def caminar(self):
        self.moviendo = True

    def detener(self):
        self.moviendo = False

juan = Persona()

juan.caminar()
print(juan.moviendo) # True
juan.detener()
print(juan.moviendo) # False
```

}

Método constructor {

```
# Anteriormente vimos que si queremos tener atributos de instancia específicos, debíamos definirlos uno a uno luego de la instanciación del objeto. Para solucionar esto, podemos usar un constructor, el cual es un método que permite a la clase asignar valores a los atributos. Su primer parámetro es self, y los demás los requeridos para la inicialización.
class Persona():
    def constructor(self, cnombre, cedad, ccabello):
        self.nombre = cnombre
        self.edad = cedad
        self.cabello = ccabello
    def quiensos (self):
        print(f"Hola. Soy {self.nombre} y tengo {self.edad} años.")

persona1 = Persona()
persona1.constructor("Juan", 42, “Rubio")
persona1. quiensos()
```

}

Método __init__() {

```
    # En el ejemplo anterior tenemos una clase con dos métodos, y uno de sus
    métodos tiene tres atributos (nombre, edad y cabello). El valor de los
    atributos para cada objeto se establecen luego de la instancia, mediante
    el constructor. El método constructor puede tener cualquier nombre.

    # Es muy importante el uso de self. El constructor crea los atributos,
    cuyo nombre comienza por self y copia en ellos los valores pasados
    mediante los parámetros. Los atributos y los parámetros suelen tener el
    mismo nombre, pero esto no es obligatorio.

    # Python ofrece un método especial denominado __init__() que simplifica el
    proceso de instancia y asignación de valores a los atributos.

    # __init__() permite que en el momento de la instanciación se asignen
    valores a los atributos sin necesidad de llamar explícitamente al
    constructor.
}
```


Método constructor {

```python
class Persona():

    # Método constructor
    def __init__(self, cnombre, cedad, ccabello):
        self.nombre = cnombre
        self.edad = cedad
        self.cabello = ccabello

    # Método normal
    def quiensos(self):
        print(f"Hola. Soy {self.nombre} y tengo {self.edad} años
              y mi cabello es {self.cabello}.")

persona1 = Persona("Juan", 42, "Rubio") # Instanciamos
persona1. quiensos()
```

}

Ejemplo {

```python
class Cuadrado:

    def __init__(self, lado):
        self.lado = lado
    def calcular_area(self):
        return self.lado ** 2
    def calcular_perimetro(self):
        return self.lado * 4

cuad1 = Cuadrado(15)
print(f"El area del cuadrado es {cuad1.calcular_area()}")
print(f"El perimetro del cuadrado es {cuad1.calcular_perimetro()}")
cuad1.lado = 12
print(f"La nueva area del cuadrado es {cuad1.calcular_area()}")
print(f"El nuevo perimetro del cuadrado es {cuad1.calcular_perimetro()}")
```

}

Método __str__() {

```
# Para mostrar objetos, Python provee otro método especial, llamado
__str__, que debe devolver una cadena de caracteres con lo que queremos
mostrar. Este método se invoca cada vez que se llama a la función str, por
ejemplo, al imprimir el objeto. El método __str__ tiene un solo parámetro,
self.
class Persona():
    def __init__(self, cnombre, cedad, ccabello):
        self.nombre = cnombre
        self.edad = cedad
        self.cabello = ccabello
    def __str__(self):
        return f"Hola. Soy {self.nombre} y tengo {self.edad} años y mi
cabello es {self.cabello}."
persona1 = Persona("Juan", 42, "Rubio") # Instanciamos
print(persona1)
```

}

Método __del__() {

```python
# El método especial __del__ se invoca automáticamente cuando el objeto se elimina de la memoria. Se puede utilizar para realizar alguna acción especial cuando tiene lugar este evento. Su sintaxis es la que vemos en el ejemplo, y tiene como único parámetro self. Los objetos se borran con del, o se eliminan al finalizar el programa.

class Persona():
    def __init__(self, cnombre, cedad, ccabello):
        self.nombre = cnombre
        self.edad = cedad
        self.cabello = ccabello
    def __str__(self):
        return f"Hola. Soy {self.nombre} y tengo {self.edad} años y mi cabello es {self.cabello}."
    def __del__(self):
        print(f"Se ha eliminado el objeto {self.nombre}.")
persona1 = Persona("Juan", 42, "Rubio") # Instanciamos
print(persona1)
```

}

```
1
2
3  Aprender a programar
4  es aprender a pensar.
5
6
7
8
9
10
11 { Steve Jobs; }
12
13
14
```


{ Nos vemos en la proxima clase }