Programación Orientada a Objetos en Python

Clases, Constructor y Atributos

Python Full Stack - Clase 16

🎯 Objetivos de Hoy

¿Qué aprenderemos?

- 🎭 Entender qué es la Programación Orientada a Objetos (OOP)
- 🏗️ Crear nuestras propias clases en Python
- 🔧 Implementar el método constructor `__init__`
- 📦 Definir y usar atributos de instancia
- 🔑 Comprender el significado de `self`
- 🎨 Crear múltiples instancias de una clase
- 💳 Construir una clase Usuario práctica

🎭 ¿Qué es la Programación Orientada a Objetos?

🎭 Programación Orientada a Objetos (OOP)

¿Cuándo necesito esto?

Escenario: "Tengo muchos usuarios en mi aplicación y necesito guardar el mismo tipo de información para cada uno"

OOP es un paradigma de programación que organiza el código en "objetos" que contienen datos (atributos) y funciones (métodos).

🎭 Analogía del Mundo Real

Piensa en **tipos de autos**:

- 🚗 **El tipo de auto (Clase)** = La categoría o modelo que define cómo será el auto
- 🚙 **Los autos individuales (Objetos/Instancias)** = Cada auto específico de ese tipo

🎭 Analogía del Mundo Real

```
Tipo de Auto: (Clase)
├── Características (Atributos)
│    ├── Marca
│    ├── Modelo
│    ├── Color
│    └── Precio
└── Funcionalidades (Métodos)
     ├── Encender
     ├── Acelerar
     └── Frenar

Auto 1 (Instancia)                Auto 2 (Instancia)
- Marca: Ford                   - Marca: Toyota
- Modelo: Corolla                - Modelo: Corolla
- Color: Rojo                     - Color: Azul
- Precio: $25,000                 - Precio: $25,000
```

Ambos autos son del mismo tipo (clase), pero cada uno tiene sus propias características únicas (atributos).

🎭 Ya has usado OOP sin saberlo

¿Recuerdas estos métodos?

```
# Listas
mi_lista = [1, 2, 3]
mi_lista.append(4)      # Método de la clase list
mi_lista.pop()          # Método de la clase list

# Strings
texto = "Hola"
texto.upper()           # Método de la clase str
texto.lower()           # Método de la clase str

# Diccionarios
persona = {"nombre": "María"}
persona.keys()          # Método de la clase dict
```

¡Todos estos son objetos con métodos! 🎉

🎭 Beneficios de OOP

¿Por qué usar OOP?

✅ **Organización:** Código más estructurado y fácil de entender
✅ **Reutilización:** Crear objetos una vez, usarlos muchas veces
✅ **Modularidad:** Cada clase tiene una responsabilidad específica
✅ **Mantenimiento:** Más fácil modificar y actualizar código
✅ **Escalabilidad:** Agregar nuevas funcionalidades sin romper lo existente

¿Qué es una Clase?

🏗️ ¿Qué es una Clase?

Definición Simple

Una **clase** es una plantilla o molde que define:

- **Atributos:** Datos que tendrá cada objeto
- **Métodos:** Acciones que puede realizar cada objeto

Analogía: La clase es como el tipo de auto (Sedán, SUV, etc.). Define qué características tendrán todos los autos de ese tipo, pero cada auto individual puede tener valores diferentes (color, precio, etc.).

🏗️ Ejemplo del Mundo Real: Clase Usuario

Imagina que necesitas registrar usuarios:

```
Tipo de Usuario (Clase Usuario)
├── Características (Atributos)
│   ├── Nombre
│   ├── Apellido
│   ├── Email
│   └── Edad
└── Funcionalidades (Métodos)
    ├── Saludar
    └── Actualizar email
```

Todos los usuarios siguen el mismo tipo (clase), pero cada uno tiene sus propios datos únicos.

🏗️ Sintaxis: Crear una Clase

La sintaxis más simple:

```
class Usuario:
    pass  # Por ahora está vacía, pero ya es una clase válida
```

Partes importantes:

- `class` → Palabra clave para definir una clase
- `Usuario` → Nombre de la clase (siempre con mayúscula inicial)
- `:` → Indica que viene el cuerpo de la clase
- `pass` → Placeholder (pronto agregaremos código aquí)

🏗️ Crear Instancias (Objetos)

Una instancia es un objeto específico creado a partir de una clase

```
# Definimos la clase
class Usuario:
    pass

# Creamos instancias (objetos)
francisco = Usuario()      # Primera instancia
carlos = Usuario()         # Segunda instancia
maria = Usuario()          # Tercera instancia
```

🏗️ Ejemplo Visual: Clase e Instancias

```
Clase Usuario (Tipo de Auto)
│
├── Instancia 1: francisco
│   └── (Datos únicos de francisco)
│
├── Instancia 2: carlos
│   └── (Datos únicos de carlos)
│
└── Instancia 3: maria
    └── (Datos únicos de maria)
```

Todas las instancias comparten la misma estructura, pero tienen datos diferentes.

💻 Ejemplo: Crear Clase e Instancias

```
# Definir la clase
class Usuario:
    pass

# Crear instancias
usuario1 = Usuario()
usuario2 = Usuario()
usuario3 = Usuario()

# Verificar que son objetos diferentes
print(usuario1)  # <__main__.Usuario object at 0x...>
print(usuario2)  # <__main__.Usuario object at 0x...>
print(usuario3)  # <__main__.Usuario object at 0x...>
```

Nota: Cada objeto tiene una dirección de memoria diferente.

🔧 El Constructor: `__init__`

🔧 ¿Qué es el Constructor?

El constructor inicializa los atributos de un objeto

Analogía: Es como cuando compras un auto nuevo. El vendedor completa los mismos datos para todos los autos del mismo tipo (marca, modelo, año), pero cada auto tiene sus propios valores específicos (número de serie, color, precio final).

El constructor `__init__` se ejecuta **automáticamente** cuando creas una nueva instancia.

🔧 Sintaxis del Constructor

```
class Usuario:
    def __init__(self):
        pass  # Aquí inicializaremos los atributos
```

Partes importantes:

- `def` → Define una función (método)
- `__init__` → Nombre especial del constructor (con doble guión bajo)
- `self` → Referencia al objeto que se está creando (SIEMPRE va primero)
- `pass` → Por ahora está vacío

🔧 ¿Por qué `self`?

`self` es como decir "yo mismo" o "este objeto"

```
class Usuario:
    def __init__(self):
        # self se refiere al objeto que se está creando
        self.nombre = "Sin nombre"  # Este objeto tendrá un nombre
```

Analogía:

Cuando te presentas dices "Mi nombre es María" → `self.nombre = "María"`

Es como si el objeto dijera "Mi nombre es..." → `self.nombre = ...`

🔧 Atributos de Instancia

Los atributos son variables que pertenecen a cada objeto

```
class Usuario:
    def __init__(self):
        # Cada usuario tendrá estos atributos
        self.nombre = "Francisco"
        self.apellido = "González"
        self.email = "francisco@email.com"
        self.limite_credito = 30000
        self.saldo_pagar = 0
```

Nota: El prefijo `self.` indica que el atributo pertenece a **este objeto específico**.

🔧 Acceder a Atributos

Usamos la notación de punto para acceder a atributos

```
class Usuario:
    def __init__(self):
        self.nombre = "Francisco"
        self.apellido = "González"
        self.email = "francisco@email.com"

# Crear instancia
francisco = Usuario()

# Acceder a atributos
print(francisco.nombre)      # Francisco
print(francisco.apellido)    # González
print(francisco.email)       # francisco@email.com
```

🔧 Problema: Todos los Usuarios Tienen los Mismos Datos

```
class Usuario:
    def __init__(self):
        self.nombre = "Francisco"
        self.apellido = "González"
        self.email = "francisco@email.com"

# Crear dos usuarios
usuario1 = Usuario()
usuario2 = Usuario()

print(usuario1.nombre)  # Francisco
print(usuario2.nombre)   # Francisco (¡mismo nombre!)
```

⚠️ **Problema:** Todos los usuarios tienen el mismo nombre.

✅ **Solución:** Pasar argumentos al constructor.

🔧 Constructor con Parámetros

Pasamos valores diferentes al crear cada instancia

```
class Usuario:
    def __init__(self, nombre, apellido, email):
        # Asignamos los valores recibidos a los atributos
        self.nombre = nombre
        self.apellido = apellido
        self.email = email
        # Valores por defecto (iguales para todos)
        self.limite_credito = 30000
        self.saldo_pagar = 0
```

Ahora cada usuario puede tener datos únicos.

🔧 Crear Instancias con Datos Personalizados

```
class Usuario:
    def __init__(self, nombre, apellido, email):
        self.nombre = nombre
        self.apellido = apellido
        self.email = email
        self.limite_credito = 30000
        self.saldo_pagar = 0

# Crear usuarios con datos diferentes
francisco = Usuario("Francisco", "González", "francisco@email.com")
carlos = Usuario("Carlos", "López", "carlos@email.com")

print(francisco.nombre)   # Francisco
print(carlos.nombre)   # Carlos (¡diferente!)
```

🔧 Desglose Paso a Paso

```python
# 1. Definimos la clase
class Usuario:
    def __init__(self, nombre, apellido, email):
        # 2. self se refiere al objeto que se está creando
        # 3. Asignamos los valores recibidos
        self.nombre = nombre
        self.apellido = apellido
        self.email = email

# 4. Creamos una instancia
#    Python llama automáticamente a __init__
francisco = Usuario("Francisco", "González", "francisco@email.com")
#    ↑      ↑      ↑           ↑          ↑
#    |      |      |           |          └─ email
#    |      |      |           └─ apellido
#    |      |      └─ nombre
#    |      └─ Método constructor
#    └─ Variable que guarda la instancia
```

🔧 Ejemplo Completo: Clase Usuario

```python
class Usuario:
    def __init__(self, nombre, apellido, email):
        # Atributos personalizados (diferentes para cada usuario)
        self.nombre = nombre
        self.apellido = apellido
        self.email = email

        # Atributos con valores por defecto (iguales para todos)
        self.limite_credito = 30000
        self.saldo_pagar = 0

# Crear usuarios
usuario1 = Usuario("María", "González", "maria@email.com")
usuario2 = Usuario("Pedro", "Martínez", "pedro@email.com")

# Ver información
print(f"{usuario1.nombre} {usuario1.apellido}")  # María González
print(f"{usuario2.nombre} {usuario2.apellido}")  # Pedro Martínez
print(usuario1.limite_credito)  # 30000
print(usuario2.limite_credito)  # 30000 (igual para todos)
```

💡 Conceptos Clave del Constructor

Concepto	Descripción	Ejemplo
`__init__`	Método especial que se ejecuta al crear un objeto	`def __init__(self):`
`self`	Referencia al objeto actual	Siempre primer parámetro
Parámetros	Valores que recibe el constructor	`nombre, apellido, email`
Atributos	Variables del objeto (con `self.`)	`self.nombre = nombre`
Valores por defecto	Valores iguales para todas las instancias	`self.limite_credito = 30000`

💡 Reglas Importantes

1. `self` siempre va primero

```
# ✅ Correcto
def __init__(self, nombre, apellido):

# ❌ Incorrecto
def __init__(nombre, self, apellido):
```

2. Usar `self.` para crear atributos

```
# ✅ Correcto
self.nombre = nombre

# ❌ Incorrecto (crea variable local, no atributo)
nombre = nombre
```

3. Parámetros opcionales al final

```
def __init__(self, nombre, apellido, email="sin@email.com"):
    # email tiene valor por defecto
```

💪 Ejercicio: Mi Primera Clase (`01-mi-primera-clase.py`)

1. Define una clase llamada `Perro`
2. Crea un `__init__` que reciba: `nombre` (string), `raza` (string), `edad` (int)
3. Asigna estos valores a atributos con `self`
4. Crea 3 instancias de `Perro` con datos diferentes
5. Imprime el nombre y raza de cada perro

Pistas:

- Recuerda que `self` siempre va primero
- Usa `self.atributo = valor` para crear atributos
- Crea instancias con `Perro(nombre, raza, edad)`

⏱️ **Tiempo:** 10 minutos

📦 Atributos de Instancia

📦 ¿Qué son los Atributos?

Los atributos son características o propiedades de un objeto

Analogía:
Si un objeto es una persona, los atributos serían:

- Nombre
- Edad
- Altura
- Color de pelo

Cada persona (instancia) tiene valores diferentes para estos atributos.

📦 Tipos de Atributos

1. Atributos definidos en el constructor

```
class Usuario:
    def __init__(self, nombre, apellido):
        self.nombre = nombre          # Definido con parámetro
        self.apellido = apellido      # Definido con parámetro
        self.activo = True            # Valor por defecto
```

2. Atributos agregados después

```
usuario = Usuario("María", "González")
usuario.telefono = "123456789"  # Agregado después
usuario.ciudad = "Santiago"      # Agregado después
```

📦 Acceder y Modificar Atributos

```
class Usuario:
    def __init__(self, nombre, apellido, email):
        self.nombre = nombre
        self.apellido = apellido
        self.email = email
        self.saldo_pagar = 0

# Crear instancia
usuario = Usuario("Diego", "Ramírez", "diego@email.com")

# Leer atributos
print(usuario.nombre)        # Diego
print(usuario.saldo_pagar)  # 0

# Modificar atributos
usuario.nombre = "Diego Andrés"
usuario.saldo_pagar = 500
print(usuario.nombre)        # Diego Andrés
print(usuario.saldo_pagar)   # 500
```

📦 Ejemplo Completo: Clase Usuario

```
class Usuario:
    def __init__(self, nombre, apellido, email):
        # Atributos básicos
        self.nombre = nombre
        self.apellido = apellido
        self.email = email

        # Atributos financieros (valores por defecto)
        self.limite_credito = 30000
        self.saldo_pagar = 0

# Crear usuarios
francisco = Usuario("Francisco", "González", "francisco@email.com")
carlos = Usuario("Carlos", "López", "carlos@email.com")

# Acceder a atributos
print(f"{francisco.nombre} tiene límite de ${francisco.limite_credito}")
# Francisco tiene límite de $30000

print(f"{carlos.nombre} debe ${carlos.saldo_pagar}")
# Carlos debe $0
```

💪 Ejercicio: Sistema de Estudiantes

Crea un archivo `02-estudiantes.py` con:

1. Define una clase `Estudiante`
2. El constructor debe recibir: `nombre` (string), `apellido` (string), `edad` (int)
    - `nombre` (string)
    - `apellido` (string)
    - `edad` (int)
3. Agrega atributos por defecto:
    - `curso` = "Python Full Stack"
    - `nota_promedio` = 0.0
    - `activo` = True

4. Crea 3 estudiantes con datos diferentes
5. Modifica la nota_promedio de cada uno
6. Imprime la información completa de cada estudiante

Pistas:

- Usa f-strings para imprimir: `f"{estudiante.nombre} tiene nota {estudiante.nota_promedio}"`
- Puedes modificar atributos con: `estudiante.nota_promedio = 8.5`

⏱️ **Tiempo:** 15 minutos

🎨 Caso Práctico: Usuario con Tarjeta

🎨 Caso Práctico: Usuario con Tarjeta

Vamos a construir un sistema de usuarios con tarjetas de crédito

Escenario:

Necesitamos crear usuarios que puedan hacer compras y acumular saldo en su tarjeta de crédito.

Requisitos:

- Cada usuario tiene nombre, apellido y email
- Cada usuario tiene un límite de crédito de $30,000
- Cada usuario comienza con saldo a pagar de $0

🎨 Clase Usuario Completa

```
class Usuario:
    def __init__(self, nombre, apellido, email):
        # Información personal
        self.nombre = nombre
        self.apellido = apellido
        self.email = email

        # Información financiera
        self.limite_credito = 30000
        self.saldo_pagar = 0
```

Esta será nuestra base para el ejercicio práctico.

🎨 Crear y Usar Usuarios

```
class Usuario:
    def __init__(self, nombre, apellido, email):
        self.nombre = nombre
        self.apellido = apellido
        self.email = email
        self.limite_credito = 30000
        self.saldo_pagar = 0

# Crear usuarios
usuario1 = Usuario("María", "González", "maria@email.com")
usuario2 = Usuario("Pedro", "Martínez", "pedro@email.com")

# Ver información
print(f"Usuario: {usuario1.nombre} {usuario1.apellido}")
print(f"Límite de crédito: ${usuario1.limite_credito}")
print(f"Saldo a pagar: ${usuario1.saldo_pagar}")

# Modificar saldo manualmente
usuario1.saldo_pagar = 500
print(f"Nuevo saldo: ${usuario1.saldo_pagar}")
```

💪 Ejercicio: Gestión de Usuarios

¡Hora de practicar!

Crea un archivo `03-gestion-usuarios.py` con:

1. Usa la clase `Usuario` del ejemplo anterior
2. Crea 3 usuarios diferentes:
    - Usuario 1: "María", "González", "maria@email.com"
    - Usuario 2: "Juan", "Pérez", "juan@email.com"
    - Usuario 3: "Catalina", "Torres", "catalina@email.com"
3. Asigna manualmente diferentes saldos a pagar:
    - Usuario 1: $1,500
    - Usuario 2: $3,200
    - Usuario 3: $0

4. Imprime para cada usuario:
    - Nombre completo
    - Email
    - Saldo a pagar
    - Límite disponible (límite - saldo)

Pistas:

- Para el límite disponible: `limite_credito - saldo_pagar`
- Usa f-strings para formatear la salida

⏱️ **Tiempo:** 15 minutos

📋 Resumen de Conceptos Clave

📋 Resumen: Clases

Concepto	Descripción	Ejemplo
Clase	Tipo o categoría que define la estructura de objetos	`class Usuario:`
Instancia	Objeto específico creado de una clase	`usuario = Usuario()`
Atributo	Variable que pertenece a un objeto	`self.nombre`
Constructor	Método especial que inicializa el objeto	`def __init__(self):`
self	Referencia al objeto actual	Siempre primer parámetro

📋 Resumen: Sintaxis Clave

```
# 1. Definir clase
class NombreClase:
    # 2. Constructor
    def __init__(self, param1, param2):
        # 3. Atributos
        self.atributo1 = param1
        self.atributo2 = param2
        self.atributo3 = valor_default

# 4. Crear instancias
objeto1 = NombreClase(valor1, valor2)
objeto2 = NombreClase(valor3, valor4)

# 5. Acceder a atributos
print(objeto1.atributo1)
objeto1.atributo1 = nuevo_valor
```

📋 Lo Más Importante

Conceptos que debes recordar:

1. ✅ **Clase** = **Tipo de auto**, Instancia = Auto individual creado
2. ✅ `__init__` se ejecuta automáticamente al crear un objeto
3. ✅ `self` siempre va primero y se refiere al objeto actual
4. ✅ **Atributos** se crean con `self.nombre_atributo = valor`
5. ✅ Cada instancia tiene sus propios valores de atributos
6. ✅ Usar **notación de punto** para acceder: `objeto.atributo`

1. Olvidar `self` como primer parámetro

```
# ❌ Error
def __init__(nombre, apellido):

# ✅ Correcto
def __init__(self, nombre, apellido):
```

2. No usar `self.` al crear atributos

```
# ❌ Error (crea variable local, no atributo)
def __init__(self, nombre):
    nombre = nombre

# ✅ Correcto
def __init__(self, nombre):
    self.nombre = nombre
```

3. Olvidar argumentos al crear instancia

```
# ❌ Error (falta argumento)
usuario = Usuario()
```

- Constructor que recibe: `nombre`, `apellido`, `email`
- Atributos por defecto: `limite_credito = 30000`, `saldo_pagar = 0`

2. Crea 5 usuarios diferentes con datos realistas
3. Simula compras manuales:
    - Usuario 1: 3 compras ($500, $800, $200)
    - Usuario 2: 2 compras ($1200, $300)
    - Usuario 3: 1 compra ($2500)
    - Usuario 4: 0 compras
    - Usuario 5: 4 compras ($100, $50, $75, $25)
4. Calcula y muestra para cada usuario:
    - Nombre completo
    - Saldo total acumulado
    - Límite disponible restante

📚 Recursos para Seguir Aprendiendo

📚 Recursos Adicionales

Documentación Oficial

- Python.org - Clases
- Python.org - Programación Orientada a Objetos

Tutoriales Interactivos

- Real Python - OOP
- W3Schools - Python Classes

Herramientas

- Python Tutor - Visualiza cómo se crean objetos paso a paso

📚 Práctica Recomendada

Ejercicios para casa:

1. ✅ Crear una clase `Producto` con nombre, precio y stock
2. ✅ Crear una clase `Libro` con título, autor y año
3. ✅ Crear una clase `CuentaBancaria` con número de cuenta y saldo
4. ✅ Experimentar con diferentes atributos
5. ✅ Crear múltiples instancias de cada clase

¡La práctica hace al maestro! 💪

🎯 Próximos Pasos

🎯 ¿Qué Viene en la Próxima Clase?

Métodos en Python

- ✅ Definir métodos en clases
- ✅ Usar `self` en métodos
- ✅ Métodos que modifican atributos
- ✅ Métodos que retornan valores
- ✅ Ejercicio práctico: Métodos en Usuario

Preparación:

- Repasa los conceptos de hoy
- Completa el ejercicio CORE
- Práctica creando más clases

🎯 Práctica para Casa

Ejercicio CORE: Usuarios con Tarjetas de Crédito

Tu tarea será:

- Crear una clase `Usuario` completa
- Implementar métodos para hacer compras
- Implementar métodos para pagar tarjeta
- Mostrar información del usuario

Este ejercicio combina todo lo aprendido hoy y es obligatorio completarlo.

💡 Consejos Finales

💡 Consejos y Mejores Prácticas

1. Nombres descriptivos

```
# ✅ Bueno
class Usuario:
    def __init__(self, nombre, apellido):

# ❌ Malo
class U:
    def __init__(self, n, a):
```

2. Organizar atributos lógicamente

```
# ✅ Bien organizado
def __init__(self, nombre, apellido, email):
    # Información personal
    self.nombre = nombre
    self.apellido = apellido
    # Información de contacto
    self.email = email
    # Información financiera
    self.limite_credito = 30000
```

3. Valores por defecto útiles

```
# ✅ Útil
self.saldo_pagar = 0  # Todos empiezan sin deuda

# ❌ Confuso
self.saldo_pagar = None  # ¿Qué significa None?
```

Error 1: Olvidar paréntesis al crear instancia

```
# ❌ Error
usuario = Usuario

# ✅ Correcto
usuario = Usuario("María", "González", "maria@email.com")
```

Error 2: Confundir clase con instancia

```
# ❌ No puedes hacer esto
Usuario.nombre = "María"  # Clase no tiene nombre

# ✅ Correcto
usuario = Usuario("María", "González", "maria@email.com")
usuario.nombre = "María"  # Instancia sí tiene nombre
```

Error 3: Atributos sin inicializar

```
# ❌ Error si intentas usar atributo no definido
usuario = Usuario("María", "González", "maria@email.com")
print(usuario.telefono)  # Error! telefono no existe
```

🎉 ¡Felicidades!

Ya sabes crear clases, constructores y atributos

¡Has dado un gran paso en tu aprendizaje de Python!

❓ Preguntas

¿Alguna duda sobre clases, constructor o atributos?

Recuerda:

- Las clases son tipos o categorías que definen la estructura de objetos
- `__init__` inicializa los atributos
- `self` se refiere al objeto actual
- Cada instancia tiene sus propios valores

🙌 ¡Excelente Trabajo!

¡Nos vemos en la siguiente clase con Métodos!

No olvides completar todos los ejercicios 📝

¡Sigue practicando! 💪