Programación

Tema 6. Programación Orientada a Objetos

Javier González Villa

David Lázaro Urrutia

Departamento de Matemática Aplicada
y Ciencias de la Computación

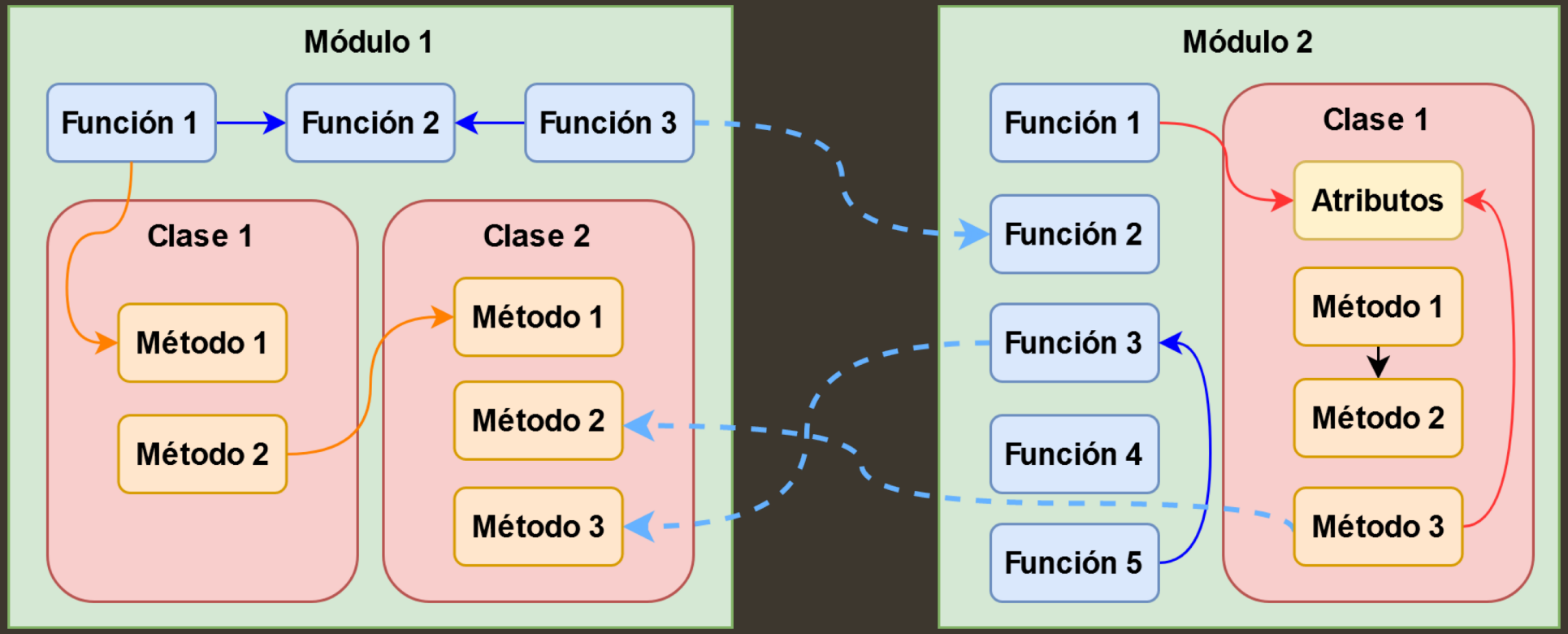
Contenidos

1. Programación Orientada a Objetos
    1. Clases
    2. Objetos
    3. Atributos
    4. Métodos
    5. Herencia y Polimorfismo
    6. Ejemplo

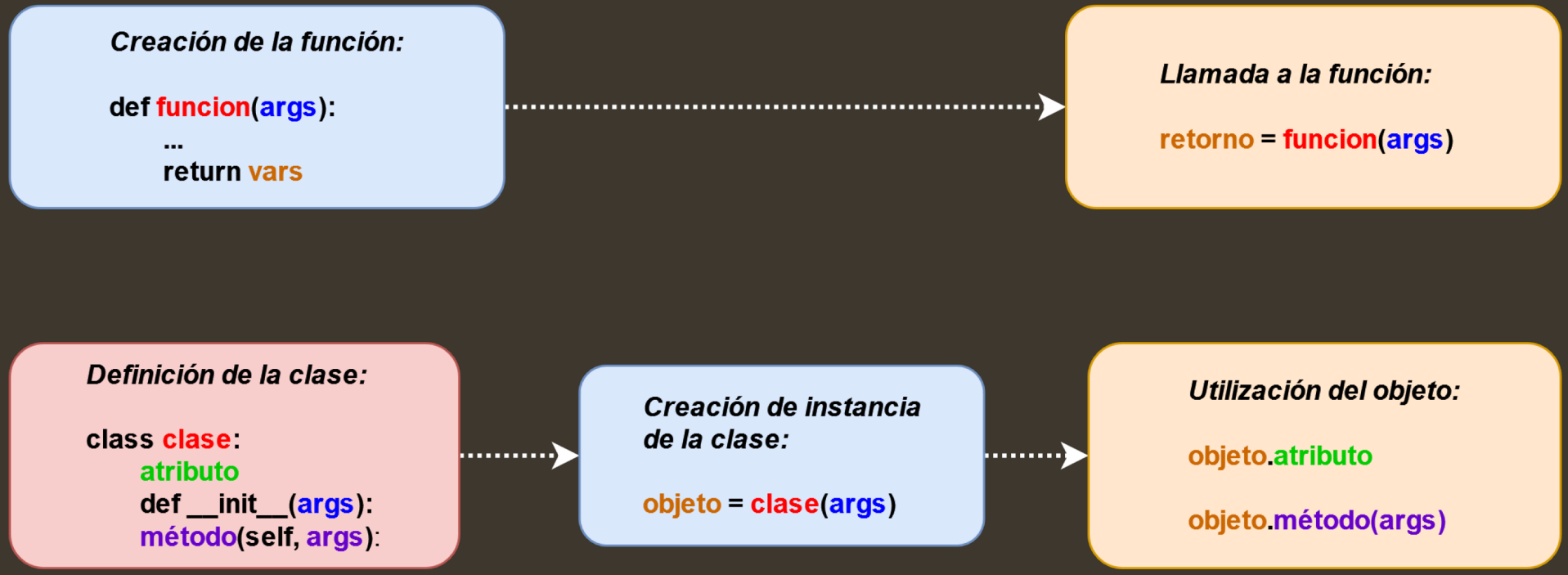
1. Programación orientada a Objetos

- Paradigma de programación que permite desarrollar aplicaciones complejas manteniendo un código más claro y manejable.
- Incrementa la reutilización de código.
- Permite realizar asignaciones como lo haríamos en la vida real.
- Permite la colaboración entre objetos para resolver problemas mas complejos.
- Permite implementar propiedades como herencia o polimorfismo.

1. Programación orientada a Objetos



1. Programación orientada a Objetos



1. Programación orientada a Objetos

Funciones	Objetos
Simples y rápidas.	Favorecen una mejor organización y representación.
Buenas para cálculos puntuales.	Útiles para problemas complejos.
Menos estructuradas.	Representan una estructura clara de los objetos de problema en la realidad.
Difíciles de ampliar y dificultan el mantenimiento del código.	Fáciles de ampliar con mejor mantenimiento del código.
Menor coste inicial a la hora de programar.	Códigos más complejos en estadios iniciales de la programación.

1.1. Clases

Clase: *tipo de dato definido por el desarrollador del cual pueden derivar objetos.*

- Estructura:

  class NombreClase:

  atributos

  constructor → **def __init__(self, argumentos)**

  métodos

- Nomenclatura: comenzando con mayúsculas cada palabra que nombra la clase sin espacios.

```
[63]: class Coche:
          marca='Audi'

          def __init__(self, matricula, valor):
              self.matricula = matricula
              self.valor = valor

          def consulta_matricula(self):
              return self.matricula

      mi_coche = Coche('1234ABC',18000)
      print(mi_coche.marca)
      print(mi_coche.consulta_matricula())
```

```
Audi
1234ABC
```

1.1. Clases

Clase: *tipo de dato definido por el desarrollador del cual pueden derivar objetos.*

- Documentación:

```
[63]: class Coche:
          marca='Audi'

          def __init__(self, matricula, valor):
              self.matricula = matricula
              self.valor = valor

          def consulta_matricula(self):
              return self.matricula

      mi_coche = Coche('1234ABC',18000)
      print(mi_coche.marca)
      print(mi_coche.consulta_matricula())
```

```
Audi
1234ABC
```

```
[ ]: class Coche:
         '''
         Clase que representa un objeto Coche.

         Atributos:
             matricula (str): Matricula del coche.
             valor (float): Valor del coche en euros.

         Métodos:
             consulta_matricula(): retorna el número de matricula.
         '''
```

1.2. Objetos

Objeto: *instancia o solicitud de una clase que puede ser* ***escalar*** *(divisible) o* ***no escalar****.*

Tipo	Notación	Ejemplo
Entero	int	1234
Punto Flotante	float	12.34
Complejo	complex	1+2j
Booleano	bool	True/False
Caracter	chr	'A'

Tipos Especiales: -inf, inf, None

```
[1]: type(1234)
[1]: int

[2]: type(12.34)
[2]: float

[3]: type(1+2j)
[3]: complex

[4]: type(True)
[4]: bool

[5]: type('a')
[5]: str

[10]: print(chr(65))
      A
```

1.2. Objetos

Objeto: *instancia o solicitud de una clase que puede ser* ***escalar*** *(divisible) o* ***no escalar****.*

- Estructura:

  Objeto = Clase(argumentos)

- Se pueden crear múltiples objetos de una misma clase sin que los valores de una interfieran en sí.
- Al crearlo, el primer método que se invoca de manera automática es el constructor.
- Los objetos son conscientes de su propia existencia (self).

```
[64]: class Coche:
          marca='Audi'

          def __init__(self, matricula, valor):
              self.matricula = matricula
              self.valor = valor
              print("Entro en el constructor!")

          def consulta_matricula(self):
              return self.matricula

      mi_coche = Coche('1234ABC',18000)

      Entro en el constructor!
```

1.3. Atributos

Atributo: *variables que solo existen dentro del objeto instancia de una clase.*

- Dinámicos:
  - Son creados bajo demanda al crea la instancia sin necesidad de haber sido especificados en la clase previamente.
- De clase:
  - Definidos de manera previa en la clase para facilitar su uso y la coherencia del objeto modelado entre las diferentes instancias.

Todos ellos pueden ser accedido y modificados en cualquier punto de la ejecución de nuestro código.

```
[65]: class Coche:
          marca='Audi'

          def __init__(self, matricula, valor):
              self.matricula = matricula
              self.valor = valor
              print("Entro en el constructor!")

          def consulta_matricula(self):
              return self.matricula

      mi_coche = Coche('1234ABC',18000)
      mi_coche.propietario = 'Yo'
      mi_coche.marca = 'Seat'
      print(mi_coche.propietario)
      print(mi_coche.marca)
```

```
Entro en el constructor!
Yo
Seat
```

1.4. Métodos

Método: *funciones que solo existen dentro del objeto instancia de una clase.*

- De clase:
  - Definidos de manera estática pueden ser llamados directamente desde la clase, pero no serán conocedores de los atributos de los objetos que se instancien.

- De instancia:
  - Son conscientes de la existencia del objeto a traves del primer atributo self y puede hacer uso de atributos y métodos del mismo.

```
[74]: class Mensajero:
          mensaje = 'No maten al mensajero!'

          def __init__(self, nombre, destino):
              self.nombre = nombre
              self.destino = destino
              print("Hola soy tu nuevo mensajero y me llamo "+str(self.nombre))

          def mensaje_dinamico(self):
              return self.destino

          def mensaje_estatico():
              return "Hola, soy un mensajero muy capaz"

      mi_mensajero1 = Mensajero('Luis','Madrid')
      print(mi_mensajero1.mensaje_dinamico())
      mi_mensajero2 = Mensajero('Sofia','Burgos')
      print(mi_mensajero2.mensaje_dinamico())

      Hola soy tu nuevo mensajero y me llamo Luis
      Madrid
      Hola soy tu nuevo mensajero y me llamo Sofia
      Burgos

[75]: mi_mensajero1.mensaje_estatico()

      ---------------------------------------------------------------------------
      TypeError                                 Traceback (most recent call last)
      Cell In [75], line 1
      ----> 1 mi_mensajero1.mensaje_estatico()

      TypeError: Mensajero.mensaje_estatico() takes 0 positional arguments but 1 was given

[76]: Mensajero.mensaje_estatico()

[76]: 'Hola, soy un mensajero muy capaz'
```

1.4. Métodos

Método: *funciones que solo existen dentro del objeto instancia de una clase.*

- Métodos especiales:
  - def __init__(self, argumentos) → Constructor
  - def __del__(self, argumentos) → Destructor
  - def __str__(self) → representación de string
  - def __len__(self) → retorna longitud del objeto
  - def __add__(self, objeto) → función del operador +
  - def __lt__(self, objeto) → función del operador <
  - …
- Métodos privados: utilizan el carácter "_" para definirlos.

```
[96]: class Prueba:
          def __init__(self, valor):
              self.valor = valor
              print("Instancia creada.")
          def __str__(self):
              return "Probando..."
          def __len__(self):
              return 1
          def __add__(self, otro):
              return self.valor+otro.valor
          def __del__(self):
              print("No me borres...")

[98]: prueba1 = Prueba(10)
      prueba2 = Prueba(20)

      Instancia creada.
      Instancia creada.

[100]: print(prueba1+prueba2)
       del(prueba1)

       30
       No me borres...
```

1.5. Herencia y Polimorfismo

Herencia: *permite crear clases derivadas que comparten métodos y atributos.*

- Estructura:

  class ClasePadre:

  definición_clase_padre

  class ClaseHijo(clase_padre):

  definición_clase_hijo

- Los hijos pueden reescribir los parámetros heredados o definir otros nuevos.
- Pueden existir clases abstractas sin implementación (pass) que fuercen la redefinición de métodos.

```
[61]: class Animal:
          def __init__(self, especie, edad):
              self.especie = especie
              self.edad = edad

          def hablar(self):
              pass

          def moverse(self):
              pass

          def describeme(self):
              print("Soy un Animal del tipo", type(self).__name__)

      class Perro(Animal):
          def hablar(self):
              print("Guau!")
          def moverse(self):
              print("Caminando con 4 patas")

      class Abeja(Animal):
          def hablar(self):
              print("Bzzzz!")
          def moverse(self):
              print("Volando")
          def picar(self):
              print("Picar!")

      mi_perro = Perro('mamífero', 10)
      mi_perro.describeme()

      Soy un Animal del tipo Perro
```

1.5. Herencia y Polimorfismo

Herencia: *permite crear clases derivadas que comparten métodos y atributos.*

- Redefinición de constructores:

```
class ClasePadre:
    def __init__(self, p_1, p_2, ..., p_N):
        self.p_1 = p_1
        self.p_2 = p_2
        ...
        self.p_N = p_N
```

```
class ClaseHijo(ClasePadre):
    def __init__(self, p_1, p_2, ..., p_N, p_N1):
        super().__init__(p_1, p_2, ..., p_N)
        self.p_N1 = p_N1
```

1.5. Herencia y Polimorfismo

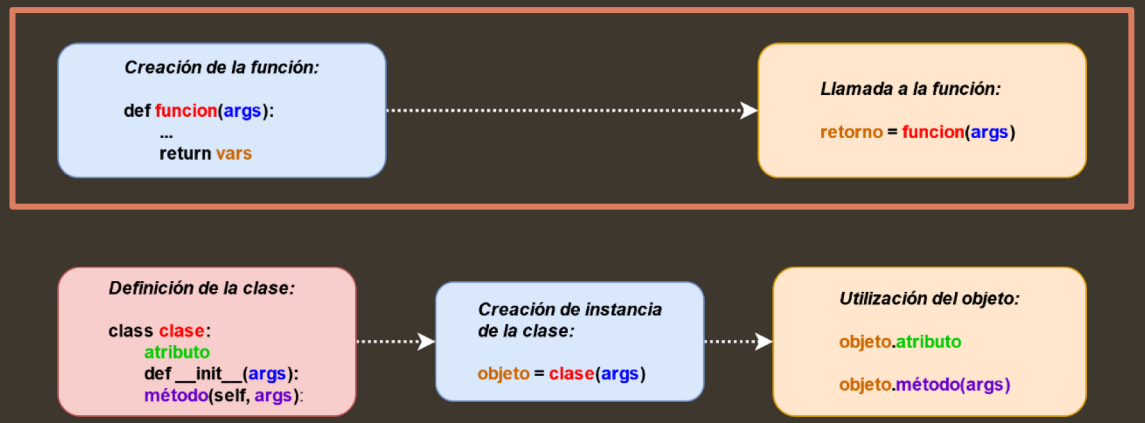
Polimorfismo: *permite a diferentes objetos ser accedidos utilizando el mismo interfaz.*

- Propiedad que es derivada de la herencia que permite acceder a objetos de diferente clase siempre que la clase padre (interfaz) sea compartida.
- La definición de una clase abstracta garantiza la existencia de los métodos polimórficos en todas las clases que lo heredan.
- Simplifican los procesos de llamada ya que las implementaciones particulares dependen de forma automática de la clase del objeto.

[62]:
```
class Animal:
    def hablar(self):
        pass

class Perro(Animal):
    def hablar(self):
        print("Guau!")

class Gato(Animal):
    def hablar(self):
        print("Miau!")

for animal in Perro(), Gato():
    animal.hablar()
```

```
Guau!
Miau!
```

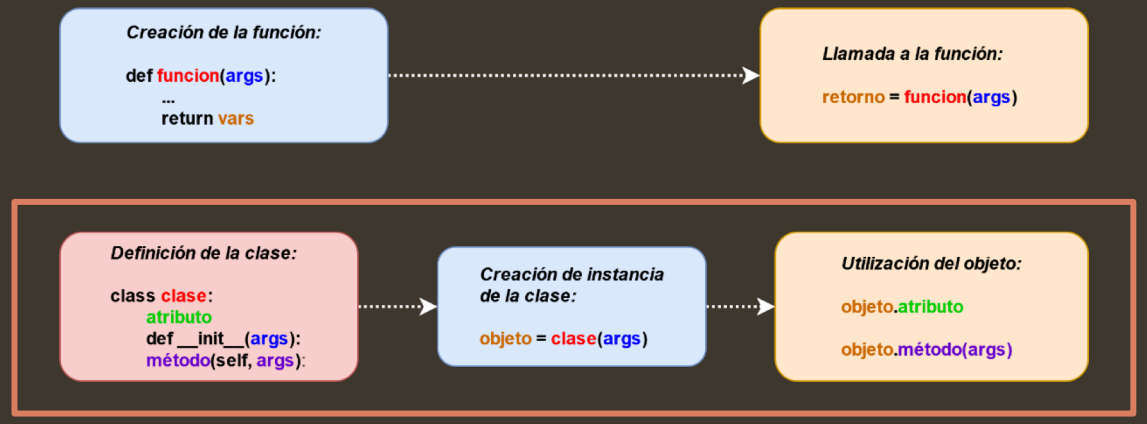
1.6. Ejemplo (Función)




```
[1]: def vel_MRUA(a, t, v0):
         '''
         Argumentos: a aceleración m/s2
                     t instante temporal s
                     v0 velocidad incial m/s
         Retorna: velocidad final vf
         '''
         vf = v0 + (a * t)
         return vf
```

```
[2]: retorno = vel_MRUA(3,10,0)
     print("Velocidad final: ",retorno, " m/s")

     Velocidad final:  30  m/s
```

1.6. Ejemplo (Clase)




```
[15]: class Coche:
          v0 = 0
          def __init__(self, a):
              self.a = a
          def vel_MRUA(self, t):
              return self.v0 + (self.a * t)
          def __str__(self):
              return "No pienso pintar nada!"
```

```
[20]: c = Coche(3)
      c2 = Coche(5)
      print(type(c))
      print(type(c2))

      <class '__main__.Coche'>
      <class '__main__.Coche'>
```

```
[21]: print("Velocidad final: ",c.vel_MRUA(10),' m/s')
      print(c)
      c.v0 = 15
      print("Velocidad inicial: ", c.v0, 'm/s')

      Velocidad final:  30  m/s
      No pienso pintar nada!
      Velocidad inicial:  15 m/s
```

1.6. Ejemplo (Contraste)

[3]:
```python
def area_losa(largo, ancho):
    return largo * ancho

def perimetro_losa(largo, ancho):
    return 2 * (largo + ancho)

largo = 5
ancho = 3

espesor = 0.2
material = "Hormigón"

print("Área:", area_losa(largo, ancho), "m²")
print("Perímetro:", perimetro_losa(largo, ancho), "m")
print("Espesor:", espesor, "m")
print("Material:", material)
```

```
Área: 15 m²
Perímetro: 16 m
Espesor: 0.2 m
Material: Hormigón
```

[4]:
```python
class Losa:
    def __init__(self, largo, ancho, espesor=0.2, material="Hormigón"):
        self.largo = largo
        self.ancho = ancho
        self.espesor = espesor
        self.material = material

    def area(self):
        return self.largo * self.ancho

    def perimetro(self):
        return 2 * (self.largo + self.ancho)

mi_losa = Losa(largo=5, ancho=3, espesor=0.25, material="Hormigón armado")

print("Área:", mi_losa.area(), "m²")
print("Perímetro:", mi_losa.perimetro(), "m")
print("Espesor:", mi_losa.espesor, "m")
print("Material:", mi_losa.material)
```

```
Área: 15 m²
Perímetro: 16 m
Espesor: 0.25 m
Material: Hormigón armado
```