# **Módulo 6**

# **Programación orientada a objetos**

* Objetos propios.
  + Formulación de un objeto propio.
  + Atributos y métodos.
  + Constructor.
  + Herencias.
    - Simples y Múltiples
    - Orden de resolución de métodos - MRO
  + Polimorfismo.
  + Abstracción.

**Conceptos clave relacionados con los objetos en Python:**

* 45º65 76u0

**Métodos**

* Métodos Públicos
* Métodos Privados
* Métodos Protegidos

**self y cls**

* self
* cls

****Todo en Python es un objeto****

* **Creación de clases**
* **Herencia múltiple**
* **Métodos especiales**
* **Encapsulamiento**
* **Polimorfismo**
* **Variables de clase**

**Clases.**

Definición de una clase.

**Objetos**

isinstance().

**Atributos**

Estado de una clase/objeto.

Funciones relativas a atributos.

* **hasattr()**
* ﻿﻿**getattr()**
* **﻿﻿setattr()**
* **deltattr()**

**Métodos﻿﻿**

* Ámbitos en los métodos.
* Agregación de funciones a objetos (monkey patching).
* Métodos especiales
* El método \_\_init\_\_(). (Constructor)
* El método \_\_del\_\_().
* Los métodos \_\_str\_() y \_\_repr\_\_().
  + \_\_str\_\_
  + \_\_repr\_\_
* El método \_\_len\_\_().
  + \_\_len\_\_
* Los métodos \_\_getitem\_() y \_\_setitem\_\_().
  + \_\_getitem\_\_
  + **\_\_setitem\_\_**

# **Programación orientada a objetos**

La programación orientada a objetos (POO) constituye un paradigma de programación fundamentado en la utilización de objetos con el fin de representar y manipular datos de manera eficiente y estructurada. Estos objetos se componen de propiedades- atributos, que encapsulan características específicas, así como de métodos – funciones de un objeto, que posibilitan la interacción tanto entre objetos como con el entorno circundante.

La esencia de la POO radica en la conceptualización de entidades autónomas y reutilizables, denominadas objetos, que combinan tanto los datos que almacenan(atributos) como las operaciones que pueden ejecutar(métodos). Las propiedades o atributos definen las cualidades características de un objeto y describen su estado actual, mientras que los métodos o funciones contienen el conjunto de acciones que el objeto puede llevar a cabo.

Mediante la implementación de la POO, los programadores pueden concebir sistemas más organizados y comprensibles, al dividir la lógica del programa en módulos interconectados pero independientes. Esto promueve la reutilización de código y la modularidad, permitiendo un desarrollo más eficiente y mantenible. Además, la POO fomenta la abstracción, ya que los objetos representan conceptos del mundo real y sus interacciones, lo que simplifica la resolución de problemas complejos.

* Objetos propios.
  + Formulación de un objeto propio.
  + Atributos y métodos.
  + Constructor.
  + Herencias.
    - Simples y Múltiples
    - Orden de resolución de métodos - MRO
  + Polimorfismo.
  + Abstracción.

****Todo en Python es un objeto**** : En Python, todo es un objeto, incluyendo tipos básicos como números, cadenas y listas. Esto significa que todos los objetos tienen atributos y métodos, y pueden ser manipulados de la misma manera que los objetos creados por el usuario.

En Python, la programación orientada a objetos está totalmente integrada en el lenguaje. Aquí hay algunas particularidades de la POO en Python.

|  |
| --- |
| **Conceptos clave relacionados con los objetos en Python** |
| **Clase**: Son estructuras fundamentales en Python que definen la estructura y el comportamiento de los objetos. Es la plantilla o molde que define atributos (propiedades) y métodos (comportamientos) que los objetos de esa clase heredarán. Las clases se crean utilizando la palabra clave `class`, seguida del nombre de la clase y los dos puntos. Los métodos se definen dentro de la clase utilizando la sintaxis def.  **Objetos:** Son instancias específicas de una clase. Cada objeto tiene sus propios valores para las propiedades definidas en la clase y puede llamar a los métodos definidos en la clase.  Es una representación concreta de la plantilla definida por la clase. Los objetos tienen sus propios valores para las propiedades de la clase y pueden ejecutar los métodos definidos por ella.  **Atributos:** Los atributos son datos en objetos como variables, listas, tuplas, etc que pertenecen a un objeto y almacenan información sobre ese atributos del objeto.  **Métodos:** Los métodos son las funciones asociadas a una clase y sus instancias. Los métodos son funciones definidas en una clase que describen el comportamiento de los objetos de esa clase. Solo pueden ser usadas por los objetos de la clase o por herencia. Estos métodos pueden acceder y modificar los atributos del objeto, los privados y protegidos están destinados a un uso interno dentro de la clase.  **Métodos especiales :**  Python tiene una serie de métodos especiales que pueden ser definidos en una clase para personalizar el comportamiento de los objetos. Por ejemplo, el método `\_\_init\_\_` se utiliza para inicializar un objeto cuando se crea una instancia de la clase.  **Módulo:** Una módulo es una plantilla o definición que describe las propiedades y comportamientos que tendrán los objetos creados a partir de ella. Define los atributos (variables) y los métodos (funciones) que estarán disponibles en los objetos de esa módulo.  **Instancia:** Una instancia es un objeto específico creado a partir de una módulo. Representa una ocurrencia única de esa módulo y tiene sus propios valores para los atributos definidos en la módulo.  **Encapsulación:** Python utiliza convenciones de nomenclatura para implementar el encapsulamiento. Los atributos y métodos que no deben ser accedidos directamente desde fuera de la clase se denotan con un o dos guiones bajo al principio del nombre, como ‘\_atributo\_privado’ y ‘\_\_atributo\_privado’. La encapsulación es un principio de la POO que se refiere a la agrupación de atributos y métodos en un objeto. Los objetos encapsulan sus datos y proporcionan interfaces (métodos) para interactuar con ellos, ocultando los detalles internos de implementación. Es el mecanismo que permite ocultar los detalles internos de un objeto y exponer sólo la interfaz pública que otros objetos pueden utilizar para interactuar con él, ayuda a prevenir errores y a proteger los datos de un objeto.  **Herencia:** La herencia es un concepto que permite crear nuevas módulos basadas en módulos existentes. Permite que una módulo herede los atributos y métodos de otra módulo, extendiéndolos o modificándolos según sea necesario. La herencia permite reutilizar código y establecer relaciones jerárquicas entre las módulos.  **Herencia múltiple:** Python permite la herencia múltiple, lo que significa que una clase puede heredar de varias clases a la vez. Para hacer esto, se colocan las clases de las que se hereda entre paréntesis separadas por comas después del nombre de la clase.  **Polimorfismo:** El polimorfismo permite responder al llamado a un método o función que tenga el mismo nombre y parámetros pero ubicadas en diferentes módulos o clases. Esto significa que un mismo nombre de método puede tener diferentes implementaciones dependiendo del objeto en el que se llame. El polimorfismo promueve la flexibilidad y la reutilización de código.  Es la capacidad de los objetos de una clase de responder a un mismo mensaje o método de diferentes maneras. Esto permite que un objeto pueda ser utilizado de forma genérica, independientemente de su tipo específico.  **Variables de clase:**  Python permite la creación de variables de clase, que son compartidas por todas las instancias de la clase. Estas variables se definen fuera de cualquier método de la clase. |

## **Clases.**

Las clases son prototipos a partir de los cuales pueden crearse objetos que adquieren las propiedades, características y comportamientos definidos por las clases.

### **Definición de una clase.**

Por convención los nombres de clases utilizan el formato "CamelCase" (escribir las palabras juntas sin espacios y con la primera letra de cada palabra en mayúscula).

Sintaxis:

|  |
| --- |
| class MiGrupoPython(<superclase\_1>, <superclase\_2>... <superclase\_n>):  ...  ... |

Las clases en Python pueden "heredar" los componentes de otras clases definidas previamente a las que se conocen como "superclases" de la clase en cuestión.

Si no se indica una superclase, no es necesario usar los paréntesis y la clase heredaría las características de \_object\_.

|  |  |  |
| --- | --- | --- |
| Código Python   |  | | --- | | class MiGrupoPython (object):  def \_\_init\_\_(self):  print(f"objeto creado {self}")  #--------------------------es lo mismo que declarar la clase asi--------------------  class MiGrupoPython:  def \_\_init\_\_(self):  print(f"objeto creado {self}")  objeto1 = MiGrupoPython()  objeto2 = MiGrupoPython() |   Salida esperada por consola   |  | | --- | | objeto creado <\_\_main\_\_.MiGrupoPython object at 0x00000xxxxxxxxxxx>  objeto creado <\_\_main\_\_.MiGrupoPython object at 0x00000yyyyyyyyyyy> |   · |

**Ya tenemos un objeto propio**

**·**

## **Objetos:**

Los objetos son las implementaciones de una clase. A la creación de un objeto a partir de una clase, se le llama "instanciar".

Todos los elementos de Python son instancias de al menos una clase.

Para crear un objeto se utiliza la siguiente sintaxis:

<Nombre de la Clase>(<argumentos>)

﻿﻿Sintaxis:

|  |
| --- |
| Objeto = MiGrupoPython()  Objeto = MiGrupoPython(parámetro\_1, parámetro\_2... parámetro\_n)#si se requiere en el \_\_init\_\_ |

Si no se le asigna un nombre al objeto, este es desechado de inmediato por el intérprete de Python.

Ejemplo:

|  |  |  |
| --- | --- | --- |
| Código Python   |  | | --- | | class MiGrupoPython:  def \_\_init\_\_(self):  print(f"objeto creado {self}")  objeto1 = MiGrupoPython()  help(MiGrupoPython)  print (f"{MiGrupoPython=}")  print (f"{objeto1=}") |   Salida esperada por consola   |  | | --- | | objeto creado <\_\_main\_\_.MiGrupoPython object at 0x00000xxxxxxxxxxx>  Help on class MiGrupoPython in module \_\_main\_\_:  class MiGrupoPython(builtins.object)  | Methods defined here:  |  | \_\_init\_\_(self)  | Initialize self. See help(type(self)) for accurate signature.  |  | ----------------------------------------------------------------------  | Data descriptors defined here:  |  | \_\_dict\_\_  | dictionary for instance variables (if defined)  |  | \_\_weakref\_\_  | list of weak references to the object (if defined)  MiGrupoPython=<class '\_\_main\_\_.MiGrupoPython'>  objeto1=<\_\_main\_\_.MiGrupoPython object at 0x00000xxxxxxxxxxx> |   · |

·

|  |  |  |
| --- | --- | --- |
| Código Python   |  | | --- | | import time  import random  class MiGrupoPython2:  def \_\_init\_\_(self,ahora):  self.creado=ahora  self.actualizado=""  print(f"objeto creado {self} a las {self.creado}")  ######################################################################  objetos = {}  for valor in range(0,10):  ahora = time.strftime("%H:%M:%S:%MS", time.localtime())  print (f"{ahora=}")  cada\_obj=MiGrupoPython2(ahora)  objetos[cada\_obj]=f"obj {ahora}"  time.sleep(random.random())  # ~ print (objetos)  time.sleep(60)  ######################################################################  for cada\_objeto in objetos:  ahora = time.strftime("%H:%M:%S:%MS", time.localtime())  cada\_objeto.actualizado=ahora  print (f"{cada\_objeto} {cada\_objeto.\_\_dict\_\_}") |   Salida esperada por consola   |  | | --- | | ahora='17:30:14:30S'  objeto creado <\_\_main\_\_.MiGrupoPython2 object at 0x0000021B2A26CC50> a las 17:30:14:30S  . . .  ahora='17:30:18:30S'  objeto creado <\_\_main\_\_.MiGrupoPython2 object at 0x0000021B2A275CD0> a las 17:30:18:30S  ######################################################################  {<\_\_main\_\_.MiGrupoPython2 object at 0x0000021B2A26CC50>: 'obj 17:30:14:30S',  . . .  <\_\_main\_\_.MiGrupoPython2 object at 0x0000021B2A275CD0>: 'obj 17:30:18:30S'}  ######################################################################  <\_\_main\_\_.MiGrupoPython2 object at 0x0000021B2A26CC50> {'creado': '17:30:14:30S', 'actualizado': '17:31:19:31S'}  . . .  <\_\_main\_\_.MiGrupoPython2 object at 0x0000021B2A275D50> {'creado': '17:30:18:30S', 'actualizado': '17:31:19:31S'} |   · |

·

### **isinstance().**

En resumen, isinstance() es una función útil para verificar la relación de herencia entre objetos y clases en Python. Te permite determinar si un objeto es una instancia de una clase en particular o de una de sus subclases. Esto puede ser útil en situaciones en las que necesitas realizar operaciones específicas según el tipo de objeto que tengas.

Sintaxis:

|  |
| --- |
| **isinstance( objeto, <clase>)**  **isinstance( objeto, (<clase1>,<clase2>,<clase3> ) )**  **\\_\_\_\_\_\_\_\_\_\_Tupla\_\_\_\_\_\_\_\_\_/** |

·

****Ejemplo:****

|  |  |  |
| --- | --- | --- |
| Código Python   |  | | --- | | obj = 9  if isinstance(obj, int):  print(f"{obj=} pertenece a la clase int")  else:  print(f"{obj=} NO pertenece a la clase int")  print("\*"\*50)  #····································································#  obj = 9.5  if isinstance(obj, int):  print(f"{obj=} pertenece a la clase int")  else:  print(f"{obj=} NO pertenece a la clase int")  print("\*"\*50)  #····································································#  obj = 9.5  if isinstance(obj, (int,float)):#la coleccion de clases solo puede ser tupla (no listas))  print(f"{obj=} pertenece a la clase int o float")  else:  print(f"{obj=} NO pertenece a las clases int ni float")  print("\*"\*50)  #····································································# |   Salida esperada por consola   |  | | --- | | obj=9 pertenece a la clase int  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  obj=9.5 NO pertenece a la clase int  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  obj=9.5 pertenece a la clase int o float  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |   · |

## **Atributos**

Los atributos son objetos que pueden ser añadidos a una clase mediante un nombre y pueden ser creados mediante la siguiente sintaxis.

|  |
| --- |
| class MiGrupoPython():  mi\_atributo = "objeto" |

Para acceder al atributo se utiliza el operador de atributo, el cual corresponde a un punto ".".

|  |
| --- |
| print(MiGrupoPython.mi\_atributo ) |

Los atributos definidos en una clase son asignados a sus instancias con los valores definidos en dicha clase.

### **Estado de una clase/objeto.**

Por lo general es posible modificar el contenido e incluso añadir o eliminar los atributos en una clase u objeto. Al conjunto de valores y otros objetos que contiene un objeto o clase en un momento específico, se le conoce como "estado".

A diferencia de otros lenguajes de programación, las clases en Python pueden cambiar de estado, de tal forma que todas las instancias cuyos atributos no hayan sido modificados cambiarían su estado al estado de la clase.

Para sustituir el contenido de un atributo se utiliza el operador de asignación "\*=\*"

sintaxis:

|  |
| --- |
| MiGrupoPython.mi\_atributo = nuevo valor |

·

|  |  |  |
| --- | --- | --- |
| Código Python   |  | | --- | | class MiPrimerClase:  ingreso=0  def \_\_init\_\_(self,nombre,apellido,titulo=""):  self.gracia = f"{titulo}.{apellido},{nombre}".title()  print(f"objeto {MiPrimerClase.ingreso} creado")  MiPrimerClase.ingreso+=1  def info(self):  print (f"{self.gracia=}")  #····································································#  print("\*"\*50)  obj=MiPrimerClase("juan","perez","ing")  #····································································#  print("\*"\*50)  obj.info()  print("\*"\*50)  #····································································#  for valor in range (1,10):  obj=MiPrimerClase(f"juan{valor}",f"perez{valor}",f"ing{valor}")  print("\*"\*50)  #····································································#  print (f"total de alumnos: {MiPrimerClase.ingreso}")  print("\*"\*50)  #····································································# |   Salida esperada por consola   |  | | --- | | obj=9 pertenece a la clase int  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  obj=9.5 NO pertenece a la clase int  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  obj=9.5 pertenece a la clase int o float  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |   · |

·

## **Funciones relativas a atributos.**

****hasattr()****

Verifica si un objeto tiene un atributo o un método específico.

Esta función toma dos argumentos:

Primer argumento es el objeto de que necesitamos obtener información.

Segundo argumento es el nombre del atributo o método que quieres comprobar.

**Salida booleana**

Regresa True en caso de que exista y False en caso contrario.

sintaxis:

|  |  |  |  |
| --- | --- | --- | --- |
| **getattr** ( | objeto ,  nombre del objeto | 'atributo',  nombre del atributo o método buscado | ) |

·

|  |  |  |
| --- | --- | --- |
| Código Python   |  | | --- | | # verificar si esta instancia tiene ciertos atributos o métodos  class MiPrimerClase:  ingreso=1  def \_\_init\_\_(self,nombre,apellido,titulo=""):  self.gracia = f"{titulo}.{apellido},{nombre}".title()  print(f"objeto '{MiPrimerClase.ingreso}' creado")  MiPrimerClase.ingreso+=1  def info(self):  print (f"{self.gracia=}")  #····································································#  print("\*"\*50)  obj=MiPrimerClase("juan","perez","ing")  #····································································#  print("\*"\*50)  obj.info()  print("\*"\*50)  #····································································#  print (f"obj tiene el atributo/método gracia {hasattr(obj,'gracia')=}")  print (f"obj tiene el atributo/método ingreso {hasattr(obj,'ingreso')=}")  print (f"obj tiene el atributo/método info {hasattr(obj,'info')=}")  print (f"obj tiene el atributo/método salida {hasattr(obj,'salida')=}") |   Salida esperada por consola   |  | | --- | | \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  objeto '1' creado  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  self.gracia='Ing.Perez,Juan'  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  obj tiene el atributo/método gracia hasattr(obj,'gracia')=True  obj tiene el atributo/método ingreso hasattr(obj,'ingreso')=True  obj tiene el atributo/método info hasattr(obj,'info')=True  obj tiene el atributo/método salida hasattr(obj,'salida')=False  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |   · |

·

﻿﻿****getattr()****

**Es similar a get de la clase dict.**

**Regresa el contenido del atributo de una clase u objeto**

**Requiere 2 argumentos admite 3**

**﻿** Primer argumento es el objeto de que necesitamos obtener información.

Segundo argumento es el nombre del atributo o método que quieres comprobar.

Tercer argumento (opcional) es el string de salida si no encuentra el atributo o método en la clase.

De no haberse ingresado un tercer argumento, se levantará una excepción de tipo InstanceError.

sintaxis:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **getattr**( | objeto ,  nombre del objeto | 'atributo',  nombre del atributo o método buscado | nuevo argumento  string de salida si no existe para el atributo | ) |

·

|  |  |  |
| --- | --- | --- |
| Código Python   |  | | --- | | # verificar si esta instancia tiene ciertos atributos o métodos  class MiPrimerClase:  ingreso=1  def \_\_init\_\_(self,nombre,apellido,titulo=""):  self.gracia = f"{titulo}.{apellido},{nombre}".title()  print(f"objeto '{MiPrimerClase.ingreso}' creado")  MiPrimerClase.ingreso+=1  def info(self):  print (f"{self.gracia=}")  #····································································#  print("\*"\*50)  obj=MiPrimerClase("juan","perez","ing")  #····································································#  print (f"Desde el obj contenido del atributo gracia {getattr(obj,'gracia')=}")  print (f"Desde el obj contenido del atributo ingreso {getattr(obj,'ingreso')=}")  #····································································#  print("\*"\*50)  #print (f"Desde el obj contenido del atributo salida {getattr(obj,'salida')=}")  # obj, no tiene el atributo salida. Al **no** tener 3er parámetro da **error**  print (f"Desde el obj contenido del atributo gracia {getattr(obj,'salida', 'sin atributo ')=}")  # obj, no tiene el atributo salida. Al (si) tener 3er parámetro lo imprime 'sin atributo'  print("\*"\*50)  #····································································#  print (f"Desde el obj contenido del atributo gracia {getattr(obj,'info')=}")#método, no atributo  print("\*"\*50) |   Salida esperada por consola   |  | | --- | | #…..  Desde el obj contenido del atributo gracia getattr(obj,'gracia')='Ing.Perez,Juan'  Desde el obj contenido del atributo ingreso getattr(obj,'ingreso')=2  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  **print (f"Desde el obj contenido del atributo gracia {getattr(obj,'salida')=}")**  **^^^^^^^^^^^^^^^^^^^^^**  **AttributeError: 'MiPrimerClase' object has no attribute 'salida'**  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  Desde el obj contenido del atributo gracia getattr(obj,'salida', 'sin atributo ')='sin atributo '  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  **Desde el obj contenido del atributo gracia getattr(obj,'info')=<bound method MiPrimerClase.info of <\_\_main\_\_.MiPrimerClase object at 0x00000163303CD490>>**  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |   · |

·

****setattr()****

****Ten en cuenta que esta función no es necesaria en python.****

Permite reemplazar o crear un atributo dentro de un objeto o clase específica en , utilizando su nombre y el valor que se desea establecer como argumentos.

**Sustituye en el objeto ingresado como primer parámetro cuyo nombre correspondiente a un objeto o clase , como segundo argumento ingresado es el nombre del atributo a agregar/modificar y el tercer parámetro es el valor del atributo . En caso de que no exista un atributo con el nombre indicado, se añade el atributo con el valor indicado.**

**Requiere 3 argumentos:**

**Primer argumento es el objeto de que necesitamos agregar o modificar el atributo.**

**Segundo argumento es el nombre del atributo que se desea establecer o cambiar.**

**Tercer argumento es el valor que recibirá el atributo.**

sintaxis:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **setattr**( | objeto ,  nombre del objeto o clase | 'atributo',  nombre del atributo a agregar/modificar | nuevo valor para el atributo buscado o creado | ) |

·

|  |  |  |
| --- | --- | --- |
| Código Python   |  | | --- | | class MiPrimerClase:  ingreso=1  def \_\_init\_\_(self,nombre,apellido,titulo=""):  self.gracia = f"{titulo}.{apellido},{nombre}".title()  print(f"objeto '{MiPrimerClase.ingreso}' creado")  MiPrimerClase.ingreso+=1  def info(self):  print (f"{self.gracia=}")  #····································································#  print("\*"\*50)  obj=MiPrimerClase("juan","perez","ing")  #····································································#  print("\*"\*50)  obj.info()  print("\*"\*50)  setattr (obj,'ingreso',55)  setattr (obj,'gracia',"Dra. Ana Maria")  print("\*"\*50)  print (f"Desde el obj contenido del atributo gracia {getattr(obj,'gracia')=}")  print (f"Desde el obj contenido del atributo ingreso {getattr(obj,'ingreso')=}")  print (f"Desde el obj contenido del atributo ingreso {getattr(MiPrimerClase,'ingreso')=}") |   Salida esperada por consola   |  | | --- | | #setattr (obj,'gracia',"Dra. Ana Maria")  Desde el obj contenido del atributo gracia getattr(obj,'gracia')='Dra. Ana Maria'  #setattr (obj,'ingreso',55)  Desde el obj contenido del atributo ingreso getattr(obj,'ingreso')=55  # se modifico el atributo del objeto ^ ^  Desde el obj contenido del atributo ingreso getattr(MiPrimerClase,'ingreso')=2  # NO se modifico el atributo de la clase ^ ^ |   · |

·

﻿﻿****deltattr()****

****Ten en cuenta que esta función no es necesaria en python teniendo la función ‘del’.****

**Se utiliza para eliminar un atributo de un objeto. Toma dos argumentos: el objeto del que deseas eliminar el atributo y el nombre del atributo que deseas eliminar.**

**Requiere 2 argumentos admite 3**

**﻿** Primer argumento es el objeto de que necesitamos obtener modificar.

Segundo argumento es el nombre del atributo o método que quieres eliminar.

Tercer argumento (opcional) es el string de salida si no encuentra el atributo o método en la clase.

De no haberse ingresado un tercer argumento, se levantará una excepción de tipo AttributeError.

sintaxis:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **delattr**( | objeto ,  nombre del objeto | 'atributo',  nombre del atributo o método buscado | nuevo argumento  string de salida si no existe para el atributo | ) |

·

Elimina el atributo de una clase u objeto cuyo nombre es ingresado como primer argumento.

En caso de que no exista un atributo con el nombre correspondiente al objeto de tipo str ingresado como segundo argumento, regresará el valor ingresado como tercer argumento. De no haberse ingresado un tercer argumento, se levantará una excepción de tipo InstanceError.

·

|  |  |  |
| --- | --- | --- |
| Código Python   |  | | --- | | class MiPrimerClase:  ingreso=1  def \_\_init\_\_(self,nombre,apellido,titulo=""):  self.gracia = f"{titulo}.{apellido},{nombre}".title()  print(f"objeto '{MiPrimerClase.ingreso}' creado")  MiPrimerClase.ingreso+=1  def info(self):  print (f"{self.gracia=}")  #····································································#  print("\*"\*50)  obj=MiPrimerClase("juan","perez","ing")  #····································································#  print("\*"\*50)  obj.info()  print("\*"\*50)  print (f"Desde el obj contenido del atributo gracia {hasattr(obj,'gracia')=}")  print (f"Desde el obj contenido del atributo gracia {hasattr(obj,'ingreso')=}")  #····································································#  print("\*"\*50)  delattr (obj,'gracia')  #····································································#  print("\*"\*50)  print (f"Desde el obj contenido del atributo gracia {hasattr(obj,'gracia')=}")  #····································································#  print("\*"\*50)  delattr (MiPrimerClase,'ingreso')  #····································································#  print("\*"\*50)  print (f"Desde el obj contenido del atributo ingreso {hasattr(obj,'ingreso')=}")  print (f"Desde el obj contenido del atributo ingreso {hasattr(MiPrimerClase,'ingreso')=}")  #····································································#  print("\*"\*50) |   Salida esperada por consola   |  | | --- | | \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  Desde el obj contenido del atributo gracia hasattr(obj,'gracia')=True  Desde el obj contenido del atributo gracia hasattr(obj,'ingreso')=True  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  Desde el obj contenido del atributo gracia hasattr(obj,'gracia')=False  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  Desde el obj contenido del atributo ingreso hasattr(obj,'ingreso')=False  Desde el obj contenido del atributo ingreso hasattr(MiPrimerClase,'ingreso')=False  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |   · |

·

﻿﻿**Métodos﻿﻿:**

Los métodos son un tipo especial de atributo que corresponde a un objeto invocable (callable) muy parecido a una función. Hasta recién vimos como funciona el método info de MiPrimerClase

Para definir un método se utiliza la siguiente sintaxis:

|  |
| --- |
| class MiGrupoPython():  def mi\_metodo(self, parámetros):  pass |

La definición de un método siempre debe de incluir un parámetro inicial el cual no será utilizado, pero le indica a Python la naturaleza del objeto. Por convención, ese parámetro lleva el nombre self, aunque podría ser cualquier palabra.

Para invocar a un método se utiliza la siguiente sintaxis:

|  |
| --- |
| objeto = MiGrupoPython()  objeto.mi\_metodo(parámetros) |

·

|  |  |  |
| --- | --- | --- |
| Código Python   |  | | --- | | class MiPrimerClase:  ingreso=1  def \_\_init\_\_(self,nombre,apellido,titulo=""):  self.gracia = f"{titulo}.{apellido},{nombre}".title()  print(f"objeto '{MiPrimerClase.ingreso}' creado")  MiPrimerClase.ingreso+=1  def info(self):  print (f"{self.gracia=}")  #····································································#  print("\*"\*50)  obj=MiPrimerClase("juan","perez","ing")  #····································································#  print("\*"\*50)  obj.info()  print("\*"\*50)  print (f"Desde el obj contenido del atributo gracia {hasattr(obj,'gracia')=}")  print (f"Desde el obj contenido del atributo gracia {hasattr(obj,'ingreso')=}")  #····································································#  print("\*"\*50)  delattr (obj,'gracia')  #····································································#  print("\*"\*50)  print (f"Desde el obj contenido del atributo gracia {hasattr(obj,'gracia')=}")  #····································································#  print("\*"\*50)  delattr (MiPrimerClase,'ingreso')  #····································································#  print("\*"\*50)  print (f"Desde el obj contenido del atributo ingreso {hasattr(obj,'ingreso')=}")  print (f"Desde el obj contenido del atributo ingreso {hasattr(MiPrimerClase,'ingreso')=}")  #····································································#  print("\*"\*50) |   Salida esperada por consola   |  | | --- | | \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  Desde el obj contenido del atributo gracia hasattr(obj,'gracia')=True  Desde el obj contenido del atributo gracia hasattr(obj,'ingreso')=True  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  Desde el obj contenido del atributo gracia hasattr(obj,'gracia')=False  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  Desde el obj contenido del atributo ingreso hasattr(obj,'ingreso')=False  Desde el obj contenido del atributo ingreso hasattr(MiPrimerClase,'ingreso')=False  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |   · |

·

### **Ámbitos en los métodos.**

|  |  |  |
| --- | --- | --- |
| Código Python   |  | | --- | | class MiPrimerClase:  ingreso=1  def contador\_personas( no\_importa ):  MiPrimerClase.ingreso +=1  print (f"desde adentro de la clase: total de personas ingresadas {MiPrimerClase.ingreso}" )    #····································································#  print("\*"\*50)  obj1=MiPrimerClase()  obj1.contador\_personas()  #····································································#  print("\*"\*50)  obj2=MiPrimerClase()  obj2.contador\_personas()  #····································································#  print("\*"\*50)  obj3=MiPrimerClase()  obj3.contador\_personas()  #····································································#  print("\*"\*50)  obj4=MiPrimerClase()  obj4.contador\_personas()  #····································································#  print("\*"\*50)  print (f"desde afuera de la clase: total de personas ingresadas {MiPrimerClase.ingreso}" ) |   Salida esperada por consola   |  | | --- | | \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  desde adentro de la clase: total de personas ingresadas 2  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  desde adentro de la clase: total de personas ingresadas 3  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  desde adentro de la clase: total de personas ingresadas 4  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  desde adentro de la clase: total de personas ingresadas 5  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  desde afuera de la clase: total de personas ingresadas 5 |   · |

·

|  |  |  |
| --- | --- | --- |
| Código Python   |  | | --- | | class MiPub:  edad=1  nombre =""  def entrada( quien ):  print (f"{quien.edad=}")  if quien.edad >=18:  print (f"Bienvenido {quien.nombre}" )  else:  print (f"Lamentablemente {quien.nombre}, no puedo dejarte entras o me clausura la muni" )  #····································································#  print("\*"\*50)  Juan=MiPub()  Juan.nombre="Juan Manuel"  Juan.edad=20  Juan.entrada()  #····································································#  print("\*"\*50)  Ana=MiPub()  Ana.nombre="Ana Manuela"  Ana.edad=25  Ana.entrada()  #····································································#  print("\*"\*50)  Luca=MiPub()  Luca.nombre="Luca Garcia"  Luca.edad=18  Luca.entrada()  #····································································#  print("\*"\*50)  Luli=MiPub()  Luli.nombre="Luciana De Maria"  Luli.edad=15  Luli.entrada()  #····································································#  print("-"\*50)  Juan.entrada()  Ana.entrada()  Luca.entrada()  Luli.entrada()  print("-"\*50) |   Salida esperada por consola   |  | | --- | | \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  quien.edad=20  Bienvenido Juan Manuel  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  quien.edad=25  Bienvenido Ana Manuela  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  quien.edad=18  Bienvenido Luca Garcia  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  quien.edad=15  Lamentablemente Luciana De Maria, no puedo dejarte entras o me clausura la muni  --------------------------------------------------  quien.edad=20  Bienvenido Juan Manuel  quien.edad=25  Bienvenido Ana Manuela  quien.edad=18  Bienvenido Luca Garcia  quien.edad=15  Lamentablemente Luciana De Maria, no puedo dejarte entras o me clausura la muni  -------------------------------------------------- |   · |

·

Los métodos tiene un ámbito local, un ámbito global y el ámbito de la clase/objeto a la que pertenecen y al que se accede mediante self.

**Métodos Públicos:**

Los métodos públicos son aquellos que se pueden acceder desde cualquier parte del código, ya sea desde dentro de la clase, desde una subclase o desde fuera de la clase. Se definen de manera normal y su nombre no empieza con un guión bajo o doble guión bajo.

|  |  |  |
| --- | --- | --- |
| Código Python   |  | | --- | | class MiGrupoPython:  def metodo\_publico(self):  print("Este es un método público.")  objeto = MiGrupoPython()  objeto.metodo\_publico() |   Salida esperada por consola   |  | | --- | | Este es un método público. |   · |

**Métodos Privados:**

Los métodos privados se definen con un guión bajo al principio de su nombre (por ejemplo, \_metodo\_privado). Aunque Python no impide acceder a estos métodos desde fuera de la clase, se considera una convención no hacerlo. Los métodos privados generalmente se utilizan para implementar detalles internos de la clase.

|  |  |  |
| --- | --- | --- |
| Código Python   |  | | --- | | class MiGrupoPython:  def \_\_metodo\_privado(self):  print("Este es un método privado.")  objeto = MiGrupoPython()  # objeto.\_\_metodo\_privado() |   Salida esperada por consola   |  | | --- | | Este es un método privado. |   · |

·

**Métodos Protegidos:**

Los métodos protegidos se definen con un solo guión bajo al principio de su nombre (por ejemplo, \_metodo\_protegido). Aunque no hay restricciones reales en el acceso a estos métodos, se considera una convención que los métodos protegidos solo se accedan desde dentro de la clase y sus subclases.

|  |  |  |
| --- | --- | --- |
| Código Python   |  | | --- | | class MiGrupoPython:  def \_metodo\_protegido(self):  print("Este es un método protegido.")  class SubClase(MiGrupoPython):  def otro\_metodo(self):  self.\_metodo\_protegido() # Accediendo al método protegido desde la subclase  objeto = SubClase()  objeto.otro\_metodo() |   Salida esperada por consola   |  | | --- | | Este es un método protegido. |   · |

·

Es importante mencionar que Python confía en la responsabilidad del programador para respetar estas convenciones de acceso. Los métodos privados y protegidos no son bloqueados por el lenguaje, sino que se basan en acuerdos de diseño y buenas prácticas para mantener la integridad y el encapsulamiento de las clases.

**self y cls:**

Son dos parámetros especiales utilizados en los métodos de clases para referirse al objeto actual y a la propia clase, respectivamente.

**self:**

Es una convención que se utiliza para referirse al objeto actual dentro de los métodos de una clase. Es similar al uso de this en otros lenguajes de programación. Al definir un método dentro de una clase, el primer parámetro debe ser self, que hace referencia al objeto que está llamando al método. A través de self, se puede acceder a los atributos y métodos del objeto.

|  |  |  |
| --- | --- | --- |
| Código Python   |  | | --- | | class MiGrupoPython:  def \_\_init\_\_(self, valor):  self.valor = valor  def imprimir\_valor(self):  print(f"{self.valor=}")  objeto = MiGrupoPython(42)  objeto.imprimir\_valor() |   Salida esperada por consola   |  | | --- | | self.valor=42 |   · |

·

|  |  |  |
| --- | --- | --- |
| Código Python   |  | | --- | | class MiPub:  def \_\_init\_\_ (self,nombre,edad):  self.nombre =nombre.upper()  self.edad=round(edad)  def entrada( self ):  print (f"{self.edad=}")  if self.edad >=18:  print (f"Bienvenido {self.nombre}" )  else:  print (f"Lamentablemente {self.nombre}, no puedo dejarte entras o me clausura la muni" )  #····································································#  print("\*"\*50)  Juan=MiPub(nombre="Juan Manuel",edad=20)  Juan.entrada()  #····································································#  print("\*"\*50)  Ana=MiPub(edad=25,nombre="Ana Manuela")  Ana.entrada()  #····································································#  print("\*"\*50)  Luca=MiPub("Luca Garcia",18)  Luca.entrada()  #····································································#  print("\*"\*50)  Luli=MiPub("Luciana De Maria",15)  Luli.entrada()  #····································································#  print("-"\*50)  Juan.entrada()  Ana.entrada()  Luca.entrada()  Luli.entrada()  print("-"\*50) |   Salida esperada por consola   |  | | --- | | \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  self.edad=20  Bienvenido JUAN MANUEL  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  self.edad=25  Bienvenido ANA MANUELA  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  self.edad=18  Bienvenido LUCA GARCIA  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  self.edad=15  Lamentablemente LUCIANA DE MARIA, no puedo dejarte entras o me clausura la muni  --------------------------------------------------  self.edad=20  Bienvenido JUAN MANUEL  self.edad=25  Bienvenido ANA MANUELA  self.edad=18  Bienvenido LUCA GARCIA  self.edad=15  Lamentablemente LUCIANA DE MARIA, no puedo dejarte entras o me clausura la muni  --------------------------------------------------  self.edad=18  Bienvenido LUCA GARCIA  self.edad=15  Lamentablemente LUCIANA DE MARIA, no puedo dejarte entras o me clausura la muni  -------------------------------------------------- |   · |

·

|  |  |  |
| --- | --- | --- |
| Código Python   |  | | --- | | class MiPub:  def \_\_init\_\_ (self,nombre,edad):  self.nombre =nombre.upper()  self.edad=round(edad)  def entrada( self ):  print (f"{self.edad=}")  if self.edad >=18:  print (f"Bienvenido {self.nombre}" )  else:  print (f"Lamentablemente {self.nombre}, no puedo dejarte entras o me clausura la muni" )  print("-"\*50)  objetos=[["Juan Manuel",20],["Ana Manuela",25],["Luca Garcia",18],["Luciana De Maria",15]]  lista\_objetos= []  dic\_objetos = {}  for nombre,edad in objetos:  obj = MiPub(nombre,edad)  lista\_objetos.append(obj)  dic\_objetos[obj]=nombre  print (f"objeto {obj}-{nombre=} creado")  #for objetos\_desde\_coleccion in dic\_objetos.keys():  for objetos\_desde\_coleccion in lista\_objetos:  print("\*"\*50)  objetos\_desde\_coleccion.entrada() |   Salida esperada por consola   |  | | --- | | --------------------------------------------------  objeto <\_\_main\_\_.MiPub object at 0x00000A>-nombre='Juan Manuel' creado  objeto <\_\_main\_\_.MiPub object at 0x00000B>-nombre='Ana Manuela' creado  objeto <\_\_main\_\_.MiPub object at 0x00000C>-nombre='Luca Garcia' creado  objeto <\_\_main\_\_.MiPub object at 0x00000D>-nombre='Luciana De Maria' creado  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  self.edad=20  Bienvenido JUAN MANUEL  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  self.edad=25  Bienvenido ANA MANUELA  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  self.edad=18  Bienvenido LUCA GARCIA  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  self.edad=15  Lamentablemente LUCIANA DE MARIA, no puedo dejarte entras o me clausura la muni |   · |

·

**cls:**

Es una convención que se utiliza para referirse a la clase dentro de los métodos de clase (métodos que se definen utilizando el decorador @classmethod). Los métodos de clase trabajan con la clase misma en lugar de instancias específicas. **El parámetro cls se utiliza para acceder a los atributos y métodos de la clase.**

|  |  |  |
| --- | --- | --- |
| Código Python   |  | | --- | | class MiGrupoPython:  contador = 0  def \_\_init\_\_(self):  MiGrupoPython.contador += 1  @classmethod  def obtener\_contador(cls):  return cls.contador  objeto1 = MiGrupoPython()  objeto2 = MiGrupoPython()  print(f"{MiGrupoPython.obtener\_contador()=}") |   Salida esperada por consola   |  | | --- | | MiGrupoPython.obtener\_contador()=2 |   · |

··

## **Métodos especiales:**

En Python se pueden identificar a los atributos y métodos especiales (magic methods) de un objeto por que su nombre está encerrado entre dobles guiones bajos "\_\_".

Estos atributos se utilizan para definir comportamientos básicos de un objeto y son comunes con otras clases.

### El método \_\_init\_\_(). (Constructor)

Es el primer método que se ejecuta un vez instanciado un objeto. Los argumentos que se ingresan dentro del paréntesis al crear un objetos son transferidos a este método.

Ejemplo:

### El método \_\_del\_\_().

Es el método que se ejecuta cuando un objeto es desechado.

Ejemplo:

|  |  |  |
| --- | --- | --- |
| Código Python   |  | | --- | | class MiGrupoPython:  def \_\_init\_\_(self, valor, string):  self.edad = valor  self.nombre = string.title()  def imprimir\_edad(self):  print(f"{self.nombre} tiene {self.edad }")  def \_\_del\_\_(self):  print(f"Adios {self.nombre } ha sido eliminado antes de los {self.edad+1 }.")  # Crear una instancia de la clase  objeto1 = MiGrupoPython(49,"Juan")  objeto2 = MiGrupoPython(25,"Ana")  # Eliminar las referencias a los objetos  del objeto1  del objeto2 |   Salida esperada por consola   |  | | --- | | Adios Juan ha sido eliminado antes de los 50.  Adios Ana ha sido eliminado antes de los 26. |   · |

·

|  |  |  |
| --- | --- | --- |
| Código Python   |  | | --- | | class MiPub:  def \_\_init\_\_ (self,nombre,edad):  self.nombre =nombre.upper()  self.edad=round(edad)  self.donde=False  def \_\_del\_\_ (self):  print ("+"\*50)  print (f"fue un gusto {self.nombre}. Adios")  def entrada( self ):  print (f"{self.edad=}")  if self.edad >=18:  print (f"Bienvenido {self.nombre}" )  self.donde=True  else:  print (f"Lamentablemente {self.nombre}, no puedo dejarte entras o me clausura la muni" )  self.donde=False  def estado(self):  if self.donde is True:  print (f"{self.nombre} esta en el pub")  else:  print (f"{self.nombre} no se encuentra aqui")  print("-"\*50)  objetos=[["Juan Manuel",20],["Ana Manuela",25],["Luca Garcia",18],["Luciana De Maria",15]]  lista\_objetos= []  #dic\_objetos = {}# ~ for objetos\_desde\_coleccion in dic\_objetos.keys():  # carga  print("carga".center(40))  for nombre,edad in objetos:  obj = MiPub(nombre,edad)  lista\_objetos.append(obj)  #dic\_objetos[obj]=nombre  print (f"objeto {obj}-{nombre=} creado")  # entrada  print("estrada".center(40))  for objetos\_desde\_coleccion in lista\_objetos:  print("\*"\*50)  objetos\_desde\_coleccion.entrada()  # salida  print("del".center(40))  for objetos\_desde\_coleccion in lista\_objetos[::2]:  lista\_objetos.remove(objetos\_desde\_coleccion)  del objetos\_desde\_coleccion  # estado  print("estado".center(40))  for objetos\_desde\_coleccion in lista\_objetos:  print("\*"\*50)  objetos\_desde\_coleccion.estado()  # fin  print("fin".center(40)) |   Salida esperada por consola   |  | | --- | | --------------------------------------------------  carga  objeto <\_\_main\_\_.MiPub object at 0x00000253D1565750>-nombre='Juan Manuel' creado  objeto <\_\_main\_\_.MiPub object at 0x00000253D163F450>-nombre='Ana Manuela' creado  objeto <\_\_main\_\_.MiPub object at 0x00000253D163F3D0>-nombre='Luca Garcia' creado  objeto <\_\_main\_\_.MiPub object at 0x00000253D163F350>-nombre='Luciana De Maria' creado  estrada  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  self.edad=20  Bienvenido JUAN MANUEL  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  self.edad=25  Bienvenido ANA MANUELA  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  self.edad=18  Bienvenido LUCA GARCIA  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  self.edad=15  Lamentablemente LUCIANA DE MARIA, no puedo dejarte entras o me clausura la muni  del  ++++++++++++++++++++++++++++++++++++++++++++++++++  fue un gusto LUCA GARCIA. Adios  ++++++++++++++++++++++++++++++++++++++++++++++++++  fue un gusto JUAN MANUEL. Adios  estado  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  ANA MANUELA esta en el pub  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  LUCIANA DE MARIA no se encuentra aqui  fin  ++++++++++++++++++++++++++++++++++++++++++++++++++  fue un gusto ANA MANUELA. Adios  ++++++++++++++++++++++++++++++++++++++++++++++++++  fue un gusto LUCIANA DE MARIA. Adios |   · |

·

|  |  |  |
| --- | --- | --- |
| Código Python   |  | | --- | | class MiPub:  objetos=[["Juan Manuel",20],["Ana Manuela",25],["Luca Garcia",18],["Luciana De Maria",15]]  lista\_objetos= []  def \_\_init\_\_ (self,nombre,edad):  self.nombre =nombre.upper()  self.edad=round(edad)  self.donde=False  def \_\_del\_\_ (self):  print ("+"\*50)  print (f"fue un gusto {self.nombre}. Adios")  def entrada( self ):  print (f"{self.edad=}")  if self.edad >=18:  print (f"Bienvenido {self.nombre}" )  self.donde=True  else:  print (f"Lamentablemente {self.nombre}, no puedo dejarte entras o me clausura la muni" )  self.donde=False  def estado(self):  if self.donde is True:  print (f"{self.nombre} esta en el pub")  else:  print (f"{self.nombre} no se encuentra aqui")  print("-"\*50)  # carga  print("carga".center(40))  for nombre,edad in MiPub.objetos:  obj = MiPub(nombre,edad)  MiPub.lista\_objetos.append(obj)  #dic\_MiPub.objetos[obj]=nombre  print (f"objeto {obj}-{nombre=} creado")  # entrada  print("estrada".center(40))  for objetos\_desde\_coleccion in MiPub.lista\_objetos:  print("\*"\*50)  objetos\_desde\_coleccion.entrada()  # salida  print("del".center(40))  for objetos\_desde\_coleccion in MiPub.lista\_objetos[::2]:  MiPub.lista\_objetos.remove(objetos\_desde\_coleccion)  del objetos\_desde\_coleccion  # estado  print("estado".center(40))  for objetos\_desde\_coleccion in MiPub.lista\_objetos:  print("\*"\*50)  objetos\_desde\_coleccion.estado()  # fin  print("fin".center(40)) |   Salida esperada por consola   |  | | --- | | --------------------------------------------------  **carga**  objeto <\_\_main\_\_.MiPub object at 0x0000026129455750>-nombre='Juan Manuel' creado  objeto <\_\_main\_\_.MiPub object at 0x000002612952F450>-nombre='Ana Manuela' creado  objeto <\_\_main\_\_.MiPub object at 0x000002612952F3D0>-nombre='Luca Garcia' creado  objeto <\_\_main\_\_.MiPub object at 0x000002612952F350>-nombre='Luciana De Maria' creado  **estrada**  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  self.edad=20  Bienvenido JUAN MANUEL  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  self.edad=25  Bienvenido ANA MANUELA  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  self.edad=18  Bienvenido LUCA GARCIA  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  self.edad=15  Lamentablemente LUCIANA DE MARIA, no puedo dejarte entras o me clausura la muni  **del**  ++++++++++++++++++++++++++++++++++++++++++++++++++  fue un gusto LUCA GARCIA. Adios  ++++++++++++++++++++++++++++++++++++++++++++++++++  fue un gusto JUAN MANUEL. Adios  **estado**  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  ANA MANUELA esta en el pub  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  LUCIANA DE MARIA no se encuentra aqui  **fin**  ++++++++++++++++++++++++++++++++++++++++++++++++++  fue un gusto LUCIANA DE MARIA. Adios  ++++++++++++++++++++++++++++++++++++++++++++++++++  fue un gusto ANA MANUELA. Adios |   · |

·

·

Los métodos \_\_str\_() y \_\_repr\_\_().

\_\_str\_\_:

Devuelve una representación legible por humanos de un objeto como una cadena.

**Propósito:** Este método se utiliza para definir una representación legible por humanos del objeto. Debe devolver una cadena que describa el objeto de una manera que sea fácilmente comprensible para los humanos.

**Uso común:** str(objeto) o print(objeto) invocan este método para obtener una cadena representativa del objeto.

\_\_repr\_\_:

Devuelve una representación no ambigua del objeto que puede ser utilizada para recrear el objeto.

**Propósito:** Este método se utiliza para proporcionar una representación no ambigua del objeto que puede ser utilizada para recrear el objeto si se pasa como argumento a la función eval() o si se utiliza en el intérprete interactivo. Debe devolver una cadena que, preferiblemente, sea una expresión válida de Python que cree un objeto idéntico.

**Uso común:** repr(objeto) invoca este método para obtener una cadena representativa del objeto.

|  |  |  |
| --- | --- | --- |
| Código Python   |  | | --- | | class MiGrupoPython:  def \_\_init\_\_(self, valor, string):  self.edad = valor  self.nombre = string.title()  def \_\_str\_\_(self):  return f'\_\_str\_\_ instancia de MiGrupoPython con nombre {self.nombre } y edad {self.edad}'  def \_\_repr\_\_(self):  return f'\_\_repr\_\_ MiGrupoPython({self.nombre },{self.edad})'  # Crear dos instancias de la clase  objeto1 = MiGrupoPython(49,"Juan")  objeto2 = MiGrupoPython(25,"Ana")  print(f"{str(objeto1)=}")  print(repr(objeto1))  print(f"{str(objeto2)=}")  print(repr(objeto2)) |   Salida esperada por consola   |  | | --- | | str(objeto1)='\_\_str\_\_ instancia de MiGrupoPython con nombre Juan y edad 49'  \_\_repr\_\_ MiGrupoPython(Juan,49)  str(objeto2)='\_\_str\_\_ instancia de MiGrupoPython con nombre Ana y edad 25'  \_\_repr\_\_ MiGrupoPython(Ana,25) |   · |

·

El método \_\_len\_\_().

\_\_len\_\_: Permite que una clase defina su comportamiento de longitud cuando se usa la función len().

|  |  |  |
| --- | --- | --- |
| Código Python   |  | | --- | | class MiGrupoPython:  def \_\_init\_\_(self, elementos):  self.elementos = elementos  def \_\_len\_\_(self):  return len(self.elementos)  objeto = MiGrupoPython([1, 2, 3, 4, 5, 6, 7, 8, 9, 0])  print(f"\_\_len\_\_ {len(objeto)=}") |   Salida esperada por consola   |  | | --- | | \_\_len\_\_ len(objeto)=10 |   · |

·

Los métodos \_\_getitem\_\_() y \_\_setitem\_\_().

\_\_getitem\_\_: Permite acceder a elementos de un objeto como si fuera una secuencia o un diccionario.

\_\_setitem\_\_: Permite establecer el valor de elementos de un objeto.

|  |  |  |
| --- | --- | --- |
| Código Python   |  | | --- | | class MiGrupoPython:  def \_\_init\_\_(self):  self.datos = {}  def \_\_getitem\_\_(self, clave):  return self.datos[clave]  def \_\_setitem\_\_(self, clave, valor):  self.datos[clave] = valor  # Crear dos instancias de la clase  objeto1 = MiGrupoPython()  objeto2 = MiGrupoPython()  objeto1["Juan"]=49  objeto2["Ana"]=25  print(f"{objeto1['Juan']=}")  print(f"{objeto2['Ana']=}") |   Salida esperada por consola   |  | | --- | | objeto1['Juan']=49  objeto2['Ana']=25 |   · |

·

Estos son solo ejemplos básicos de cómo funcionan estos métodos especiales. Puedes personalizar su comportamiento para que se adapten a las necesidades de tu clase. Los métodos especiales son una parte fundamental de la programación orientada a objetos en Python y permiten que tus objetos se comporten de manera más natural y predecible en diversas situaciones.

## **Agregación de funciones a objetos (monkey patching).**

Monkey patching es una técnica que permite modificar o extender el comportamiento de una clase o módulo en tiempo de ejecución. Consiste en agregar, reemplazar o modificar métodos como objetos como si fueran un atributo sin modificar directamente su código fuente.

Debido a que los métodos utilizan el parámetro self, si se intenta añadir una función a una clase, está no funcionaría como si fuera un método.

|  |
| --- |
| Esta técnica es poderosa pero debe usarse con precaución. Ya que se modifica por afuera uno puede hacer que el código sea más difícil de entender y mantener si un colega o uno mismo después de un tiempo trata de entender porque el método escrito da otro resultado que el esperado. |

·

|  |  |  |
| --- | --- | --- |
| Código Python   |  | | --- | | class MiClase:  def trabajo(self, a, b):  resultado = a - b  return resultado  obj = MiClase()  resultado = obj.trabajo(10, 3)  print(f"Resultado original: {resultado}")  resultado = obj.trabajo(3, 10)  print(f"Resultado original: {resultado}")  print ("\ncreo el nuevo método")  def trabajo\_mejorado(self, a, b):  if a > b:  resultado = a - b  elif a < b:  resultado = b - a  else:  resultado = 0  return resultado  print ("asigno y sobre-escribo el método original con el nuevo método\n")  MiClase.trabajo = trabajo\_mejorado  resultado = obj.trabajo(10, 3)  print(f"Resultado nuevo método: {resultado}")  resultado = obj.trabajo(3, 10)  print(f"Resultado nuevo método: {resultado}") |   Salida esperada por consola   |  | | --- | | Resultado original: 7  Resultado original: -7  creo el nuevo método  asigno y sobre-escribo el método original con el nuevo método  Resultado nuevo método: 7  Resultado nuevo método: 7 |   · |

·

Es importante mencionar que el monkey patching puede ser útil en situaciones específicas, como corregir errores en bibliotecas de terceros o agregar funcionalidades adicionales a clases existentes. Pero genera al programador un código muy difícil de seguir.