CLASES Y OBJETOS EN PYTHON

Una **clase** es una plantilla para la creación de objetos según un modelo definido previamente. Las clases se utilizan para la definición de atributos (variables) y métodos (funciones).

Un **objeto** sería una instancia de esa clase, es decir, un objeto sería la llamada a una clase. Por ejemplo cuando se importa el módulo random de Python. Este módulo en sí es una clase, al llamar a dicha clase estamos creando una instancia (objeto) de dicha clase. Cuando llamamos al método randint() realmente estamos llamando al método randint de la clase random a través de un objeto creado para dicho fin.

Viendo esta relación, podemos llegar fácilmente a la conclusión de que para crear un objeto, debemos crear previamente la clase, la cual queremos instanciar.

Definición de una clase

Para crear una clase hay que hacer uso de la palabra class a continuación el nombre que le asignes y por último dos puntos (:). Ej:

class Restaurante:

El código debe estar indentado, en caso contrario se producirá un error. Después del nombre de la clase no lleva paréntesis, a menos que quisiéramos que herede características de otra clase.

Atributos de una clase

Dentro de la clase podemos definir atributos o propiedades de la clase . Ej:

```
class Restaurante:

   nombre = "Mi Restaurante"
   cuit = "30-12345678-9"
   categoria = 4
   concepto = "Temático“
```

- Para asignar atributos sólo debemos declarar variables dentro de la clase, estas serían las características principales.
- No hay límites en cuanto a los atributos.
- Cuando definimos atributos tenemos que estar pendientes de:
  - Asignarle siempre un valor ya que en caso contrario el interprete disparará una excepción.
  - Los nombres de los atributos deberán ser lo más sencillos y descriptivos posibles.

Instanciar una clase - Objeto de la clase

Una clase no se puede manipular directamente, es por eso que se debe instanciar un objeto de la clase para así modificar los atributos que posea. Para instanciar lo único que debemos hacer es asignarle a una variable el nombre de la clase seguido de paréntesis.

```
rest_1= Restaurante()
```

Crear una instancia de la clase y emitir atributos

```
# ejemplo 1
class Restaurante:

    nombre = "Mi Restaurante"
    cuit = "30-12345678-9"
    categoria = 4
    concepto = "Temático“

rest1 = Restaurante()
print(rest1.nombre)
print(rest1.cuit)
print(rest1.categoria)
print(rest1.concepto)
print(f"El rest se llama {rest1.nombre} su cuit es {rest1.cuit} de categoria {rest1.categoria} y concepto
 {rest1.concepto}")
```

```
>>> rest1 = Restaurante()
>>> print(f"El rest se llama {rest1.nombre} su cuit es {rest1.cuit} de categoria {rest1.categoria} y concepto {rest1.concepto}")
El rest se llama Mi Restaurante su cuit es 30-12345678-9 de categoria 4 y concepto Temático
```

Emitimos los valores de los atributos de la clase

Modificar atributos

```
# ejemplo 2
class Restaurante:

    nombre = "Mi Restaurante"
    cuit = "30-12345678-9"
    categoria = 4
    concepto = "Temático"

rest1 = Restaurante()
print(rest1.nombre)
rest1.nombre = "Rest_1"
rest1.cuit = "30-11111111-8"
rest1.categoria = 3
rest1.concepto = "Comida rápida"
print(f"El restaurante se llama {rest1.nombre} su cuit es {rest1.cuit} de categori
a {rest1.categoria} y concepto {rest1.concepto}")
```

Si podemos referenciar un atributo haciendo **rest1.nombre**, entonces podemos tratar los atributos como si fueran variables y la podemos modificar.

```
>>> rest1 = Restaurante()
>>> rest1.nombre = "Rest_1"
>>> rest1.cuit = "30-11111111-8"
>>> rest1.categoria = 3
>>> rest1.concepto = "Comida rápida"
>>> print(f"El restaurante se llama {rest1.nombre} su cuit es {rest1.cuit} de categoria {rest1.categoria} y concepto {rest1.concepto}")
El restaurante se llama Rest_1 su cuit es 30-11111111-8 de categoria 3 y concepto Comida rápida
```

Métodos

Todos los métodos tienen como primer parámetro el identificador **self**

Los métodos son esencialmente funciones dentro de las clases y pertenecerán a la clase. Para invocarlos debemos escribir el nombre de la variable para después poder hacer uso del método. Para definir un método usamos "def", luego el nombre, dos puntos y luego (**self**).

```
# Ejemplo 3
class Restaurante:
    #método con parámetros
    def agregar_restaurante(self, nombre, cuit, categoria, concepto):
        print('Agregando restaurante....')
        self.nombre = nombre
        self.cuit = cuit
        self.categoria = categoria
        self.concepto = concepto
    #método
    def mostrar_info(self):
        print('Emitiendo info de restaurante....')
        print(f'Nombre: {self.nombre}')
        print(f'Cuit: {self.cuit}')
        print(f'Categoría: {self.categoria}')
        print(f'Concepto: {self.concepto}')

rest1 = Restaurante()
rest1.agregar_restaurante('Rest_1',
                '30-11111111-8',3,'Comida rápida')
rest1.mostrar_info()
rest2 = Restaurante()
rest2.agregar_restaurante('Rest_2',
                '30-22222222-7',2,'Para llevar')
rest2.mostrar_info()
```

Para que el método funcione dentro de una clase debe cumplir con:

- Extra indentado: todo bloque debe esta extraindentado dentro de la clase para que el interprete de Python lo entienda.
- Siempre debe poseer un argumento self para que cuando sea invocado, Python le pase el objeto instanciado y así pueda operar con los valores actuales de la instancia.
- Si no se incluye el self , Python emitirá una excepción.

```
>>> rest1 = Restaurante()
>>> rest1.agregar_restaurante('Rest_1','30-11111111-8',3,'Comida rápida')
Agregando restaurante....
>>> rest1.mostrar_info()
Emitiendo info de restaurante....
Nombre: Rest_1
Cuit: 30-11111111-8
Categoría: 3
Concepto: Comida rápida
>>> rest2 = Restaurante()
>>> rest2.agregar_restaurante('Rest_2','30-22222222-7',2,'Para llevar')
Agregando restaurante....
>>> rest2.mostrar_info()
Emitiendo info de restaurante....
Nombre: Rest_2
Cuit: 30-22222222-7
Categoría: 2
Concepto: Para llevar
```

Métodos constructor de la clase: __init__

def __init__ es la definición de una función como cualquier otra.

El nombre __init__ , Python lo reserva para los métodos constructores.

- Un método constructor de una clase se ejecuta automáticamente cuando se crea un objeto. El objetivo es inicializar los atributos de un objeto.
- Es imposible olvidarse de llamarlo porque se llama automáticamente.
- Se ejecuta inmediatamente después de la creación del objeto.
- No puede retornar datos.
- Puede recibir parámetros normalmente para inicializar los atributos.
- Es opcional, de todos modos es común declararlo.
- Se escribe con dos guiones bajos, la palabra init y a continuación otros dos guiones bajos.

Métodos constructor de la clase : __init__

```
# abstracción y constructores, se refiere a qué datos son necesarios en la clase
 y los objetos
# Ejemplo 4
class Restaurante:

    def __init__ (self, nombre, cuit, categoria, concepto):
        print('Agregando restaurante....')
        self.nombre = nombre
        self.cuit = cuit
        self.categoria = categoria
        self.concepto = concepto

    def mostrar_info(self):
        print('Emitiendo info de restaurante....')
        print(f'Nombre: {self.nombre}')
        print(f'Cuit: {self.cuit}')
        print(f'Categoría: {self.categoria}')
        print(f'Concepto: {self.concepto}')

rest1 = Restaurante('Rest_1','30-11111111-8',3,'Comida rápida')
rest1.mostrar_info()

rest2 = Restaurante('Rest_2','30-22222222-7',2,'Para llevar')
rest2.mostrar_info()
```

```
>>> rest1 = Restaurante('Rest_1','30-11111111-8',3,'Comida rápida')
Agregando restaurante....
>>> #rest1.agregar_restaurante('Rest_1','30-11111111-8',3,'Comida rápida')
>>>
>>> rest1.mostrar_info()
Emitiendo info de restaurante....
Nombre: Rest_1
Cuit: 30-11111111-8
Categoría: 3
Concepto: Comida rápida
>>>
>>> rest2 = Restaurante('Rest_2','30-22222222-7',2,'Para llevar')
Agregando restaurante....
>>> # rest2.agregar_restaurante('Rest_2','30-22222222-7',2,'Para llevar')
>>>
>>> rest2.mostrar_info()
Emitiendo info de restaurante....
Nombre: Rest_2
Cuit: 30-22222222-7
Categoría: 2
Concepto: Para llevar
```

Al implementar el constructor pasándole los parámetros inicializamos el objeto con los datos en el momento de crearlo.

Métodos constructor de la clase: __init__

```
# ejemplo 5
class Restaurante:

    def __init__(self):
        self.nombre = ''
        self.cuit = ''
        self.categoria = 0
        self.concepto = ''

    def agregar_restaurante(self):
        print('Alta restaurante....')
        self.nombre = input("Ingrese el nombre del restaurante\n")
        self.cuit = input("Ingrese el número de cuit\n")
        self.categoria = int(input("Ingrese la categoría (1 a 5)\n"))
        self.concepto = input(
            "Ingrese el concepto:\nGourmet, Especialidad, Familiar, Buffet,\nComida rápida, Buffet, Para llevar\n")

    def mostrar_info(self):
        print('Información del restaurante....')
        print(f'Nombre: {self.nombre}')
        print(f'Cuit: {self.cuit}')
        print(f'Categoría: {self.categoria}')
        print(f'Concepto: {self.concepto}')

rest1 = Restaurante()
rest1.agregar_restaurante()
rest1.mostrar_info()
```

```
Alta restaurante....
Ingrese el nombre del restaurante
Rest 1
Ingrese el número de cuit
30-22222222-7
Ingrese la categoría (1 a 5)
3
Ingrese el concepto:
Gourmet, Especialidad, Familiar, Buffet,
Comida rápida, Buffet, Para llevar
Buffet
>>> rest1.mostrar_info()
Información del restaurante....
Nombre: Rest 1
Cuit: 30-22222222-7
Categoría: 3
Concepto: Buffet
```

Tenemos al constructor creado entonces agregamos el método agregar_restaurante y lo invocamos inmediatamente después de crear el objeto.

Llamado de métodos desde otro método de la misma clase

```
# Ejemplo 6
class Restaurante:
    def __init__(self):
        self.nombre = ''
        self.cuit = ''
        self.categoria = 0
        self.concepto = ''
    def agregar_restaurante(self):
        print('Alta restaurante....')
        self.nombre = input("Ingrese el nombre del restaurante\n")
        self.cuit = input("Ingrese el número de cuit\n")
        self.categoria = int(input("Ingrese la categoría (1 a 5)\n"))
        self.concepto = input(
            "Ingrese el concepto:\nGourmet, Especialidad, Familiar, Buffet,\nComida rápida, Buffet, Para llevar\n")
    def mostrar_info(self):
        print('Información del restaurante....')
        print(f'Nombre: {self.nombre}')
        print(f'Cuit: {self.cuit}')
        print(f'Categoría: {self.categoria}')
        print(f'Concepto: {self.concepto}')

    def main(self):
        self.mostrar_info()

rest1 = Restaurante()
rest1.agregar_restaurante()
rest1.main()
```

Es importante saber que, llamar un método desde otro método, solo se puede hacer dentro de la misma clase.

```
Alta restaurante....
Ingrese el nombre del restaurante
Rest 2
Ingrese el número de cuit
30-33333333-3
Ingrese la categoría (1 a 5)
4
Ingrese el concepto:
Gourmet, Especialidad, Familiar, Buffet,
Comida rápida, Buffet, Para llevar
Especialidad
>>> rest1.main()
Información del restaurante....
Nombre: Rest 2
Cuit: 30-33333333-3
Categoría: 4
Concepto: Especialidad
```

Colaboración entre clases

```
class Cliente: # ejemplo 7
    def __init__(self,nombre):
        self.nombre=nombre
        self.monto=0

    def factura(self, monto):
        self.monto = self.monto + monto

    def impuesto(self):
        self.monto = self.monto + (self.monto * 0.21)

    def retornar_monto(self):
        return self.monto

    def imprimir(self):
        print(self.nombre,": El importe a abonar es de $",self.mon
to)

class Restaurante:

    def __init__(self):
        self.nombre = ''
        self.cuit = ''
        self.categoria = 0
        self.concepto = ''

def agregar_restaurante(self):
        print('Alta restaurante....')
        self.nombre = input("Ingrese el nombre del restaurante\n")
        self.cuit = input("Ingrese el número de cuit\n")
        self.categoria = int(input("Ingrese la categoría (1 a 5)\n"))
        self.concepto = input(
            "Ingrese el concepto:\nGourmet, Especialidad, Familiar,
Buffet,\nComida rápida, Buffet, Para llevar\n")

def mostrar_info(self):
        print('Información del restaurante....')
        print(f'Nombre: {self.nombre}')
        print(f'Cuit: {self.cuit}')
        print(f'Categoría: {self.categoria}')
        print(f'Concepto: {self.concepto}')

    def un_cliente(self):
        self.cliente1=Cliente("Pepe")
        self.cliente1.factura(1000)
        self.cliente1.impuesto()
        self.cliente1.retornar_monto()
        self.cliente1.imprimir()

    def main(self):
        self.mostrar_info()
        self.un_cliente()

rest1 = Restaurante()
rest1.agregar_restaurante()
rest1.main()
```

La clase Cliente colabora con la clase Restaurante

```
Información del restaurante....
Nombre: Rest 4
Cuit: 30-9999999-4
Categoría: 4
Concepto: Familiar
Pepe : El importe a abonar es de $ 1210.0
```

Encapsulamiento

```
class Restaurante: # ejemplo 8

    def __init__(self,nombre,cuit,categoria,concepto):
        self.nombre = nombre  #Default Public
        self.cuit = cuit      #Default Public
        self.__categoria = categoria #Private
        self._concepto = concepto   #Protected

    def mostrar_info(self):
        print('Información del restaurante....')
        print(f'Nombre: {self.nombre}')
        print(f'Cuit: {self.cuit}')
        print(f'Categoría: {self.__categoria}')
        print(f'Concepto: {self._concepto}')

    def main(self):
        self.mostrar_info()

    def get_categoria(self, categoria):
        return self.__categoria

    def set_categoria(self, categoria):
        self.__categoria = categoria

rest1 = Restaurante('Rest 5', '30-55555555', 2, 'Buffet')
rest1.main()
rest1.set_categoria(5)
rest1.main()
categoria = rest1.get_categoria(4)
print(categoria)
rest1.main()
```

Default Public:
quiere decir que se puede modificar en cualquier lugar de la aplicación.

Protected:
un guión bajo dice que está protegido de cambios, sólo puede ser accesible desde una clase derivada.

Private:
doble guión bajo, sólo es accesible dentro de la clase, no se puede modificar sólo por algún método getters y setters. También se pueden encapsular métodos.

```
Información del restaurante....
Nombre: Rest 5
Cuit: 30-55555555
Categoría: 2
Concepto: Buffet
>>> rest1.set_categoria(5)
>>> rest1.main()
Información del restaurante....
Nombre: Rest 5
Cuit: 30-55555555
Categoría: 5
Concepto: Buffet
>>> categoria = rest1.get_categoria(4)
>>> print(categoria)
5
```

get: obtiene un valor
set : agrega un valor

Herencia

En Python, dos clases, además de poder tener una relación de colaboración, también pueden tener una relación de herencia. A través de la **herencia** se pueden crear nuevas clases a partir de una clase o de una jerarquía de clases (comprobadas y verificadas), evitando el rediseño, verificación y modificación de la parte implementada. Implica que una subclase tiene todo el comportamiento (métodos) y los atributos (variables) de su superclase, además, de poder agregar los suyos propios. Por medio de la herencia, una clase, extiende su funcionalidad, y permite la reutilización y la extensibilidad.

La clase de la que se hereda suele llamarse clase base, superclase, clase padre, clase ancestro (depende del lenguaje de programación)

En los lenguaje que tienen un sistema de tipos muy fuerte y restrictivo con el tipo de datos de las variables, la herencia suele ser un requisito fundamental para poder emplear el **polimorfismo**.

Herencia simple	Herencia Múltiple
Una clase sólo puede heredar de una clase base y de ninguna otra.	Una clase puede heredar las características de varias clases base, es decir, puede tener varios padres. En este aspecto hay discrepancias entre los diseñadores de lenguajes, algunos prefieren no admitir la herencia múltiple por los conflictos entre métodos y variables con igual nombre, y eventualmente con comportamientos diferentes pueda crear un desajuste que va en contra de los principios de la POO. Por ello la mayoría de los lenguajes admiten herencia simple, en contraste, unos pocos admiten la herencia múltiple, entre ellos C++, Python o Eiffel.

Herencia simple

```
class Restaurante: # ejemplo 9
    def __init__(self,nombre,cuit,categoria,concepto):
        self.nombre = nombre
        self.cuit = cuit
        self.__categoria = categoria
        self.__concepto = concepto
    def mostrar_info(self):
        print('Información....')
        print(f'Nombre: {self.nombre}')
        print(f'Cuit: {self.cuit}')
        print(f'Categoría: {self.__categoria}')
        print(f'Concepto: {self.__concepto}')

    def main(self):
        self.mostrar_info()

    def get_categoria(self, categoria):
        return self.__categoria

    def set_categoria(self, categoria):
        self.__categoria = categoria

class Hotel(Restaurante):
    def __init__(self,nombre, cuit, categoria, concepto):
        super().__init__(nombre, cuit, categoria, concepto)

hotel = Hotel('Hotel POO', '30-12341234-9', 5, 'Boutique')
hotel.mostrar_info()

rest1 = Restaurante('Rest 5', '30-55555555', 2, 'Buffet')
rest1.main()
```

Agregando entre paréntesis el nombre de la clase, estamos especificándole que hereda dicha clase. Entonces podemos decir que la clase Hotel heredará los atributos de Restaurante.

```
>>> hotel = Hotel('Hotel POO', '30-12341234-9', 5, 'Boutique')
>>> hotel.mostrar_info()
Información....
Nombre: Hotel POO
Cuit: 30-12341234-9
Categoría: 5
Concepto: Boutique
>>> rest1 = Restaurante('Rest 5', '30-55555555', 2, 'Buffet')
>>> rest1.main()
Información....
Nombre: Rest 5
Cuit: 30-55555555
Categoría: 2
Concepto: Buffet
```

Con el método **super()** hacemos referencia a la clase heredada. De otra forma podemos llamar directamente a la clase en lugar de al método **super().**

Herencia simple

```
class Restaurante: # ejemplo 10
    def __init__(self,nombre,cuit,categoria,concepto):
        self.nombre = nombre
        self.cuit = cuit
        self.__categoria = categoria
        self.__concepto = concepto
    def mostrar_info(self):
        print('Información....')
        print(f'Nombre: {self.nombre}')
        print(f'Cuit: {self.cuit}')
        print(f'Categoría: {self.__categoria}')
        print(f'Concepto: {self.__concepto}')

    def main(self):
        self.mostrar_info()
    def get_categoria(self, categoria):
        return self.__categoria
    def set_categoria(self, categoria):
        self.__categoria = categoria

class Hotel(Restaurante):
    def __init__(self,nombre, cuit, categoria, concepto, pileta):
        super().__init__(nombre, cuit, categoria, concepto)
        self.pileta = pileta

    def get_pileta(self):
        return self.pileta
hotel = Hotel('Hotel POO', '30-12341234-9', 5, 'Boutique‘, ‘Si’)
hotel.mostrar_info()
rest1 = Restaurante('Rest 5', '30-55555555', 2, 'Buffet')
rest1.main()
```

```
>>> hotel = Hotel('Hotel POO', '30-12341234-9', 5, 'Boutique')
>>> hotel.mostrar_info()
Información....
Nombre: Hotel POO
Cuit: 30-12341234-9
Categoría: 5
Concepto: Boutique
>>> rest1 = Restaurante('Rest 5', '30-55555555', 2, 'Buffet')
>>> rest1.main()
Información....
Nombre: Rest 5
Cuit: 30-55555555
Categoría: 2
Concepto: Buffet
```

Agrego atributo y método exclusivo de la clase Hotel. Pero no sale en la emisión porque debo redefinir el método mostrar_info() para Hotel, eso se llama polimorfismo….

Herencia Múltiple

```
class Restaurante: # ejemplo 11
    def __init__(self, nombre, cuit, categoria, concepto):
        self.nombre = nombre
        self.cuit = cuit
        self.categoria = categoria
        self.concepto = concepto
    def mostrar_info(self):
        print('Información....')
        print(f'Nombre: {self.nombre}')
        print(f'Cuit: {self.cuit}')
        print(f'Categoría: {self.categoria}')
        print(f'Concepto: {self.concepto}')
    def main(self):
        self.mostrar_info()

class Gerente:
    def __init__(self, dni, apellido):
        self.dni = dni
        self.apellido = apellido
    def marcacion(self):
        print(f"Apellido: {self.apellido}: Marca asistencia 1 vez.")
```

```
class Hotel(Restaurante, Gerente):

    def __init__(self, nombre, cuit, categoria, concepto, dni, apellido, pileta):
        super().__init__(nombre, cuit, categoria, concepto)
        self.pileta = pileta
        Restaurante.__init__(self, nombre, cuit, categoria, concepto)
        Gerente.__init__(self, dni, apellido)

    def get_pileta(self):
        return self.pileta

    def mostrar_info(self):
        print(f'Nombre: {self.nombre}, Cuit: {self.cuit}, Categoría: {self.categoria}, Concepto: {self.concepto}, Gerente: {self.apellido}, Pileta: {self.pileta}')

gerente = Gerente(12341234, 'Python3')
print(gerente.marcacion())
rest1 = Restaurante('Rest 5', '30-55555555', 2, 'Buffet')
rest1.main()
hotel = Hotel('Hotel Python', '30-12341234-9', 5,
              'Boutique', 111111111, 'Python3', 'Si')
hotel.mostrar_info()
```

Inicializo el constructor de la clase con los parámetros e inicializo los constructores de las clases heredadas

```
>>>
>>> hotel.mostrar_info()
Nombre: Hotel Python, Cuit: 30-12341234-9, Categoría: 5, Concepto: Boutique, Gerente: Python3, Pileta: Si
>>>
```

Polimorfismo

- Es la cualidad de los objetos de responder de distintos modo al mismo mensaje.

- Es el cambio de comportamiento de un objeto de tal manera que una referencia a una clase acepta directivas de objeto de dicha clase y de sus clases derivadas

Tipos de Polimorfismo	
Sobrecarga:	**Paramétrico:**
Se encuentra cuando, varias clases independientes entre sí , cuentan con un método con el mismo nombre y la misma funcionalidad.	Es la capacidad para definir varias funciones utilizando el mismo nombre pero usando parámetros diferentes.

Polimorfismo

```
class Gerente: # ejemplo 12

    def marcacion(self):
        print("Marca asistencia 1 vez.")

class Encargado:

    def marcacion(self):
        print("Marca asistencia 2 veces.")

class Mozo:

    def marcacion(self):
        print("Marca asistencia 2 veces y firma planilla.")

def marcacionTrabajador(trabajador):
    trabajador.marcacion()

class Restaurante:

    def __init__(self,nombre,cuit,categoria,concepto):
        self.nombre = nombre
        self.cuit = cuit
        self.__categoria = categoria
        self.__concepto = concepto
def mostrar_info(self):
        print('Información....')
        print(f'Nombre: {self.nombre}')
        print(f'Cuit: {self.cuit}')
        print(f'Categoría: {self.__categoria}')
        print(f'Concepto: {self.__concepto}')
```

```
    def main(self):
        self.mostrar_info()

    def get_categoria(self, categoria):
        return self.__categoria

    def set_categoria(self, categoria):
        self.__categoria = categoria

rest1 = Restaurante('Rest 5', '30-55555555', 2, 'Buffet')
rest1.main()
mTrabajador = Mozo()
marcacionTrabajador(mTrabajador)
mTrabajador2 = Gerente()
marcacionTrabajador(mTrabajador2)
```

El método marcacion esta definido en las clases Gerente, Encargado y Mozo. La función marcacionTrabajador recibe el objeto y llama al método marcacion según el objeto creado.

```
Información....
Nombre: Rest 5
Cuit: 30-55555555
Categoría: 2
Concepto: Buffet
>>> mTrabajador = Mozo()
>>> marcacionTrabajador(mTrabajador)
Marca asistencia 2 veces y firma planilla.
>>> mTrabajador2 = Gerente()
>>> marcacionTrabajador(mTrabajador2)
Marca asistencia 1 vez.
>>>
```

Polimorfismo

```
class Restaurante: # ejemplo 13
    def __init__(self, nombre, cuit, categoria, concepto):
        self.nombre = nombre
        self.cuit = cuit
        self.categoria = categoria
        self.concepto = concepto
    def mostrar_info(self):
        print('Información....')
        print(f'Nombre: {self.nombre}')
        print(f'Cuit: {self.cuit}')
        print(f'Categoría: {self.categoria}')
        print(f'Concepto: {self.concepto}')
    def main(self):
        self.mostrar_info()
    def get_categoria(self, categoria):
        return self.__categoria
    def set_categoria(self, categoria):
        self.__categoria = categoria
class Hotel(Restaurante):
    def __init__(self, nombre, cuit, categoria, concepto, pileta):
        super().__init__(nombre, cuit, categoria, concepto)
        self.pileta = pileta
    def get_pileta(self):
        return self.pileta
    def mostrar_info(self):
        print(
            f'Nombre: {self.nombre} Cuit: {self.cuit} Categoría: {self.categoria} Precio: {self.concepto} Pileta: {self.pileta}')
hotel = Hotel('Hotel POO','30-12341234-9',5,'Boutique','Si')
hotel.mostrar_info()
rest1 = Restaurante('Rest 5', '30-55555555', 2, 'Buffet')
rest1.main()
```

```
>>> hotel = Hotel('Hotel POO','30-12341234-9',5,'Boutique','Si')
>>> hotel.mostrar_info()
Nombre: Hotel POO Cuit: 30-12341234-9 Categoría: 5 Precio: Boutique Pileta: Si
>>> rest1 = Restaurante('Rest 5', '30-55555555', 2, 'Buffet')
>>> rest1.main()
Información....
Nombre: Rest 5
Cuit: 30-55555555
Categoría: 2
Concepto: Buffet
```

Redefino el método mostrar_info() para la clase Hotel

Variables de clase

En algunos momentos necesitaremos almacenar datos que sean compartidos por todos los objetos de la misma clase, en esos momentos necesitamos emplear variables de clase. Para definir una variable de clase lo hacemos dentro de la clase pero fuera de sus métodos. La variable de clase es compartida por todos los objetos.

```
class Restaurante: # ejemplo 14

    restaurantes = []

    def __init__(self, nombre, cuit, categoria, concepto):
        self.nombre = ''
        self.cuit = ''
        self.categoria = 0
        self.concepto = ''
        self.restaurantes.append(nombre)

    def mostrar_info(self):
        print('Información....')
        print(f'Nombre: {self.nombre}')
        print(f'Cuit: {self.cuit}')
        print(f'Categoría: {self.categoria}')
        print(f'Concepto: {self.concepto}')

rest1 = Restaurante('Rest 1', '30-11111111-9', 2, 'Buffet')
rest2 = Restaurante('Rest 2', '30-22222222-7', 3, 'Especialidad')
rest3 = Restaurante('Rest 3', '30-33333333-5', 5, 'Temático')
rest4 = Restaurante('Rest 4', '30-44444444-3', 4, 'Comida rápida')

print(Restaurante.restaurantes)
```

```
>>> rest1 = Restaurante('Rest 1', '30-11111111-9', 2, 'Buffet')
>>> rest2 = Restaurante('Rest 2', '30-22222222-7', 3, 'Especialidad')
>>> rest3 = Restaurante('Rest 3', '30-33333333-5', 5, 'Temático')
>>> rest4 = Restaurante('Rest 4', '30-44444444-3', 4, 'Comida rápida')
>>> print(Restaurante.restaurantes)
['Rest 1', 'Rest 2', 'Rest 3', 'Rest 4']
```

Método str

Python nos permite redefinir el método que se debe ejecutar, esto se hace *definiendo en la clase.* En este ejemplo, devolvemos un string con el método __str__ que mostrará el dato que le indiquemos al emitir dicho objeto. Para generar dicho string, debemos concatenar valores fijos como el paréntesis o la coma, dobles comillas y convertir a string los atributos que no lo son .

```
class Restaurante: # ejemplo 15

    def __init__(self, nombre, cuit, categoria, concepto):
        self.nombre = nombre
        self.cuit = cuit
        self.categoria = categoria
        self.concepto = concepto

    def __str__(self):
        cadena=self.nombre+', número de cuit:'+self.cuit+', de categoría: '+str(self.categoria)+', tipo: '+ s
elf.concepto
        return cadena

rest1 = Restaurante('Rest 1', '30-11111111-9', 2, 'Buffet')
rest2 = Restaurante('Rest 2', '30-22222222-7', 3, 'Especialidad')
rest3 = Restaurante('Rest 3', '30-33333333-5', 5, 'Temático')
rest4 = Restaurante('Rest 4', '30-44444444-3', 4, 'Comida rápida')
print(rest1)
print(rest2)
print(rest3)
print(rest4)
```

```
>>> rest1 = Restaurante('Rest 1', '30-11111111-9', 2, 'Buffet')
>>> rest2 = Restaurante('Rest 2', '30-22222222-7', 3, 'Especialidad')
>>> rest3 = Restaurante('Rest 3', '30-33333333-5', 5, 'Temático')
>>> rest4 = Restaurante('Rest 4', '30-44444444-3', 4, 'Comida rápida')
>>> print(rest1)
Rest 1, número de cuit:30-11111111-9, de categoría: 2, tipo: Buffet
>>> print(rest2)
Rest 2, número de cuit:30-22222222-7, de categoría: 3, tipo: Especialidad
>>> print(rest3)
Rest 3, número de cuit:30-33333333-5, de categoría: 5, tipo: Temático
>>> print(rest4)
Rest 4, número de cuit:30-44444444-3, de categoría: 4, tipo: Comida rápida
>>>
```

Method Resolution Order (MRO)

Ese es el orden en el cual el método debe heredar en la presencia de herencia múltiple.

```
>>> Restaurante.__mro__
(<class '__main__.Restaurante'>, <class 'object'>)
>>> Gerente.__mro__
(<class '__main__.Gerente'>, <class 'object'>)
>>> Hotel.__mro__
(<class '__main__.Hotel'>, <class '__main__.Restaurante'>, <class '__main__.Gerente'>, <class 'object'>)
>>>
```

vars()

Podemos ver las clases.

```
>>> vars()
{'__name__': '__main__', '__doc__': None, '__package__': None, '__loader__': <class '_frozen_importlib.BuiltinImporter'>, '__spec__': None, '__annotations__': {}, '__builtins__': <module 'builtins' (built-in)>, 'Gerente': <class '__main__.Gerente'>, 'Encargado': <class '__main__.Encargado'>, 'Mozo': <class '__main__.Mozo'>, 'marcacionTrabajador': <function marcacionTrabajador at 0x000002C57626FD30>, 'Cliente': <class '__main__.Cliente'>, 'Restaurante': <class '__main__.Restaurante'>, 'Hotel': <class '__main__.Hotel'>}
```

dir()

Podemos ver los atributos y métodos de una clase.

```
>>> dir(Restaurante)
['__class__', '__delattr__', '__dict__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '__gt__', '__hash__', '__init__', '__init_subclass__', '__le__', '__lt__', '__module__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__setattr__', '__sizeof__', '__str__', '__subclasshook__', '__weakref__', 'agregar_restaurante', 'get_categoria', 'main', 'mostrar_info', 'restaurantes', 'set_categoria', 'un_cliente']
```

FIN