Hoja informativa: Clases y objetos

¿Qué son las clases y los objetos?

Un objeto es una **instancia de una clase**.

Ejemplo. Imagina una fábrica que produce perros de juguete. Para fabricar los juguetes, se utiliza un molde con el aspecto básico de un perro: cuatro patas, un cuerpo, una cabeza y una cola.

Primero se diseña el molde y luego se usa para producir todos los perros. Los perros pueden tener una apariencia diferente: blancos, marrones, con manchas, etc. Pero todos ellos tienen un aspecto común: cuatro patas, un cuerpo, una cabeza y una cola.

En esta analogía, el molde de un perro es una **clase** de Python. El perro que se creó a base de este molde es un **objeto**, o una **instancia de una clase**.

Básicamente, una clase no hace nada por sí misma, solo describe cuál será el futuro objeto generado.

En Python, muchas entidades se describen mediante objetos. Por ejemplo, si en una prueba automatizada necesitas acceder a una página web específica, podrías desarrollar una clase que represente esa página, permitiéndote luego interactuar con ella como si fuese un objeto.

Cómo crear una clase

Para crear una clase, haz lo siguiente:

- Declara una nueva clase utilizando la palabra clave `class`.
- Inventa un nombre para tu clase, por ejemplo, `User`.

Cómo nombrar clases

Hay algunas buenas de nomenclatura para las clases.

Con mayúscula inicial. Es buena práctica escribir con mayúscula inicial el nombre de una clase para diferenciarlos de otras variables: `User`. No utilizar mayúsculas no provocará un error, pero contradice la buena práctica aceptada.

Solo números y caracteres latinos. Por ejemplo, `User` o `User2`. Cualquier otro carácter puede provocar un error.

No empieces el nombre con un número. `2User` no es un nombre válido: el programa se bloqueará.

No uses espacios. Si el nombre tiene más de una palabra, escribe en mayúscula la primera letra de cada palabra. Por ejemplo, `MyUser`. Esta convención de nomenclatura es conocida como **PascalCase** debido a su uso en el lenguaje de programación Pascal y fue muy popular en su momento.

Variables de clase

Una clase es un molde para objetos. Debe tener parámetros específicos. Por ejemplo, queremos que todos los perros tengan una cola y cuatro patas.

Para eso, necesitaremos **variables** de clase, o **atributos**.

Las variables de clase contienen los parámetros que compartirán todos los objetos.

Por ejemplo, necesitaremos las variables `is_registered`, `name` y `login` para nuestros usuarios:

```
class User():
    is_registered = True
    name = 'Mark'
    login = 'supermarkus94'
```

Como hemos definido estas variables dentro de la clase, todos los usuarios (objetos de esta clase) tendrán dichos nombres y logins.

Si necesitas cambiar el nombre del usuario o la usuaria en algún momento del programa, puedes anular este parámetro. Utiliza la siguiente sintaxis: `NombreDeClase.nombre_de_variable`. En nuestro caso, `User.name`.

Reemplacemos el nombre "Mark" por "Bob":

```
print(User.name) # Salida "Mark"

User.name = 'Bob' # Anular la variable de clase

print(User.name) # Salida "Bob"
```

Ten en cuenta que el nombre se cambiará en consecuencia para todos los usuarios al mismo tiempo.

Cómo crear objetos

Esta es nuestra clase `User` con todas las variables:

```
class User():
       is_registered = True
       name = 'Mark'
       login = 'supermarkus94'
```

Puedes crear un objeto utilizando esta sintaxis: `object_name = ClassName()`. En nuestro caso, `user = User()`.

Puedes crear varios objetos de usuario:

```
class User():
       is_registered = True
       name = 'Mark'
       login = 'supermarkus94'

user_1 = User() # Crear objetos
user_2 = User()
user_3 = User()
```

El problema es que todos estos objetos serán idénticos. Tendrán el mismo nombre y login.

El método `__init__`

Para generar diferentes objetos, necesitas un **constructor**: el método `init`.

En Python, las clases tienen un método especial llamado `__init__`, que actúa como un constructor. Este método se ejecuta automáticamente cuando se crea una nueva instancia (objeto) de la clase. Sirve para inicializar los atributos del objeto con valores específicos o predeterminados.

Para definir el método `__init__`, empieza con la palabra clave `def`, seguida del nombre del método y un conjunto de paréntesis:

```
class User():
    is_registered = True
    name = 'Mark'
    login = 'supermarkus94'

    def __init__(...): # Definición del método constructor
          # Resto del código
```

La llamada al método aún no está completa; también necesitas un **argumento** `self` obligatorio.

El argumento `self` te permite hacer referencia a un objeto y asignarle valores únicos. Por ejemplo, un nombre o login concreto.
Este es una especie de argumento complementario: no contiene ningún valor y solo pasa parámetros a los objetos.

```
  def __init__(self, ..): # self es el primer argumento obligatorio
        # Resto del código
```

El argumento `self` va primero; después puedes enumerar todos los demás atributos que quieres asignar a tus objetos. En nuestro caso, el nombre y login: `name`, `login`.

```
  def __init__(self, name, login): # Argumento self obligatorio, seguido de name y login
        # Resto del código
```

En realidad, puedes nombrar este argumento como quieras, pero en la mayoría de los casos simplemente se llama `self`. Lo más importante es colocarlo primero.

Especificarás estos valores (name y login) para cada usuario individualmente al crear objetos.

Es necesario indicar los campos para almacenar estos valores. Para ello, utiliza la notación de puntos con `self`. Por ejemplo, `self.user_name = name` significa que "el objeto tendrá un campo llamado `user_name`, y este campo almacenará el valor `name`".

Entonces, tenemos un constructor que crea un nuevo usuario o usuaria con parámetros específicos: name y login. Al mismo tiempo, cada objeto tendrá el parámetro `is_registered = True`.

```
class User():  # Declara una clase
    is_registered = True

    def __init__(self, name, login):
        self.user_name = name
        self.user_login = login
```

Ahora, podemos crear tantos usuarios distintos como sea necesario. Puedes asignarles nombres y logins entre paréntesis:

```
class User():
    is_registered = True

    def __init__(self, name, login):
        self.user_name = name
        self.user_login = login


user_1 = User('Mark','supermarkus94')
user_2 = User('Bob', 'bobisthebest')
user_3 = User('Ted', 'tediousted')
```

También puedes comprobar qué parámetros tiene un objeto. Por ejemplo, puedes comprobar que la variable `user_1` tiene el nombre Mark.

Puedes mostrar valores de parámetros en la consola mediante `print()`. Utiliza la notación de puntos con el nombre del objeto y el parámetro:

```
print(user_1.user_name) # Salida: Mark
print(user_1.is_registered) # Salida: True (es lo mismo para todos los objetos)
```

¿Qué es un método?

📎 Un método es una función que está asociada a un objeto.

Cómo declarar un método

Declarar un método en Python es muy parecido a crear una función normal, pero hay algunas reglas específicas que debes seguir:

- Usa la palabra clave `def` para definir el método.
- Elige un nombre para el método siguiendo las convenciones de nombres de Python.
- El argumento `self` obligatorio. Este argumento permite que el método acceda a los atributos y otros métodos del objeto.
- Escribe el bloque de código que determina qué hará el método.

Vamos a crear un método llamado `describe()`. Devolverá la descripción del usuario o la usuaria: su nombre y login.

Para acceder a los atributos del objeto User, como el nombre y login, el método describe() necesita usar el argumento self.

Aquí tienes un ejemplo de cómo podrías hacerlo:

```
class User():
    is_registered = True

    def __init__(self, name, login):
        self.user_name = name
        self.user_login = login

    def describe(self):
        return f'Nombre: {self.user_name}, login: {self.user_login}'
```

El método se ha creado, pero para que funcione, debes llamarlo, como una función.

Cómo llamar a un método

Lo que tienes que hacer:

1. Nombre del Objeto: Empieza escribiendo el nombre del objeto en el que deseas llamar al método. Este objeto debe ser una instancia de la clase que contiene el método que deseas usar.

2. Punto Separador: Después del nombre del objeto, coloca un punto `.`. Este punto indica que vas a acceder a uno de los métodos o atributos del objeto. Por ejemplo, `.insert`.

3. Paréntesis: Finalmente, coloca un par de paréntesis '()' después del nombre del método. Si el método requiere argumentos, introdúcelos dentro de estos paréntesis, separados por comas.

```
class User():
    is_registered = True

    def __init__(self, name, login):
        self.user_name = name
        self.user_login = login

    def describe(self):
        return f'Nombre: {self.user_name}, login: {self.user_login}'

user_1 = User('Mark', 'supermarkus94') # Crear un objeto user_1
print(user_1.describe()) # Llamar al método para este objeto

# Salida: "Nombre: Mark, login, supermarkus94"
```

Cómo interactúan las clases

Muy a menudo, hay múltiples clases en un programa. Pueden interactuar entre sí, y una clase puede hacer referencia a las variables y los métodos de otra clase.

Aquí está tu clase `User`:

```
class User():
    is_registered = True

    def __init__(self, name, login):
        self.user_name = name
        self.user_login = login

    def describe(self):
        return f'Nombre: {self.user_name}, login {self.user_login}'
```

Ahora, imaginemos que tenemos una segunda clase llamada `Group`. Esta clase tiene un atributo `name` para el nombre del grupo y una lista vacía members para almacenar los usuarios que pertenecen a ese grupo.

La clase `Group` también tiene un método `add_member()` para agregar usuarios al grupo. Este método también verifica que el usuario no se repita en la lista.

```
class Group():
    def __init__(self,name): # El grupo tiene un nombre
        self.name = name
        self.members = [] # Inicializar la lista de miembros como vacía

    def add_member(self, user): # El método agrega un usuario o usuaria
        if user not in self.members: # Verificar que el usuario no está ya en el grupo
            print('Nuevo miembro agregado')
            self.members.append(user) # Agregar un nuevo usuario o usuaria al grupo
```

Si necesitamos mostrar la información de todos los usuarios en un grupo, podemos hacerlo incluso si el método para mostrar esta información (`describe()`) pertenece a la clase `User`. ¿Cómo? Podemos

acceder al método `describe()` de cada objeto `User` dentro de nuestra clase `Group`.

```
class Group():
    def __init__(self,name): # El grupo tiene un nombre
        self.name = name
        self.members = [] # Información sobre los miembros del grupo

    def add_member(self, user): # Recorremos cada miembro en la lista
        if user not in self.members: # Llamamos al método 'describe()' del usuario
            print('Nuevo miembro agregado')
            self.members.append(user) # Agregar un nuevo usuario o usuaria al grupo

    def print_member_descriptions(self):  # El método muestra la información de todos los miembros del grupo
        print(f'Información sobre los miembros del grupo {self.name}:')
        for member in self.members: # Para cada usuario o usuaria,
            print(member.describe()) # mostrar la información
```

Usamos esta sintaxis para acceder al método de otra clase:

```
instancia_de_clase.otro_nombre_de_método_de_clase()
self.dog.bark()
```

Ahora podemos agregar usuarios y usuarias al grupo y mostrar su información:

```
class User():
    is_registered = True

    def __init__(self, name, login):
        self.user_name = name
        self.user_login = login

    def describe(self):
        return f'Nombre: {self.user_name}, iniciar sesión {self.user_login}'


class Group():
    def __init__(self, name):
        self.name = name
        self.members = []

    def add_member(self, user):
        if user not in self.members:
            print(f'Se ha agregado un nuevo miembro al grupo {self.name}')
            self.members.append(user)

    def print_member_descriptions(self):
        print(f'Información sobre los miembros del grupo {self.name}:')
        for member in self.members:
            print(member.describe())

user1 = User('Mark', 'supermarkus94') # Crear un usuario o usuaria
user2 = User('Bob', 'bobisthebest') # Crear otro usuario o usuaria
group1 = Group('Dog Lovers') # Crear un grupo de amantes de los perros
group1.add_member(user1) # Agregar un nuevo usuario o usuaria al grupo
group1.add_member(user2) # Agregar un segundo usuario o usuaria al grupo
group1.print_member_descriptions() # Mostrar información sobre los usuarios y usuarias
```