

# ¿Para qué usamos Clases en Python?

## Clases en Python: qué son

Python es un lenguaje de programación orientado a objetos. Casi todo en Python es un objeto, con sus propiedades y métodos. Una clase es una especie de plantilla o molde para crear objetos, una estructura de programación que define un conjunto de métodos y atributos.

### ¿Para qué sirven?

Como hemos mencionado anteriormente, una clase define una plantilla para crear objetos, los cuales serán instancias de esa clase. Los objetos creados tienen las mismas propiedades y comportamientos definidos por la clase, pero pueden tener valores diferentes para los atributos que se definen en la clase.

## Crear una clase

Para definir una clase debemos usar la palabra clave **class** seguida del nombre de la clase y dos puntos (:). En la siguiente línea, que debe estar indentada, deberemos introducir las definiciones de métodos y atributos.

En este ejemplo sencillo, hemos definido la clase y hemos añadido atributos generales que pertenecen a la clase y por lo tanto serán comunes a todos los objetos que creemos con ella:

```
class SimpsonCharacters:
    city = "Springfield"
    type = "Cartoon"
```

### Convención general para los nombres de las clases

Según la guía de estilo de Python ([link](#)) los nombres de las clases se escriben con la letra inicial mayúscula y se usa Pascal Case.

```
class NombreClase():
```

## Añadir métodos o funciones a la clase

Además de atributos, las clases tienen un conjunto de funciones o métodos que realizan diferentes funcionalidades.

Los métodos se definen de la misma manera que fuera de las clases, pero tienen como primer parámetro o argumento el objeto al que se le aplicará el método, que suele llamarse **self** por convención.

### Error argumentos al definir métodos

Veamos el anterior ejemplo, en el que hemos introducido una función a nuestra clase pero no hemos añadido ningún argumento.

```
class SimpsonCharacters:
    city = "Springfield"
    type = "Cartoon"
```

```
def creator():
    print("Creator: Matt Groening")

barney = SimpsonCharacters()

barney.creator()
```

Si ejecutamos el código, obtendremos el siguiente error:

```
Traceback (most recent call last):
  File "/home/runner/Just-playin/main.py", line 11, in <module>
    print(barney.creator())
TypeError: SimpsonCharacters.creator() takes 0 positional arguments but 1 was given
```

Como veis, Python nos está avisando de que hay un problema con el número de argumentos proporcionado. La forma correcta de definir la función dentro de la clase será la siguiente:

```
class SimpsonCharacters:
    city = "Springfield"
    type = "Cartoon"

    def creator(self):
        print("Creator: Matt Groening")

barney = SimpsonCharacters()

barney.creator()
# Output: Creator: Matt Groening
```

## Crear objeto

Un objeto define una versión particular de una clase, con unos atributos particulares para ese objeto. El proceso de creación de un objeto a partir de una clase se llama instanciación. Es decir, el objeto es la instancia de la clase. La sintaxis para la creación es similar a llamar a una función. Proporcionamos el nombre del objeto, y lo igualamos al nombre de la clase terminado en paréntesis ().

En nuestro ejemplo, habíamos creado anteriormente una instancia llamada barney, y podemos crear más a partir de la clase. En este caso, aún no tenemos atributos que sean específicos para cada instancia, por lo que no pasaremos ningún argumento al instanciar los objetos.

```
class SimpsonCharacters:
    city = "Springfield"
    type = "Cartoon"

    def creator(self):
        print("Creator: Matt Groening")
```

```
barney = SimpsonCharacters()  
lisa = SimpsonCharacters()  
ralph = SimpsonCharacters()
```

## Imprimir los objetos instanciados

Partiendo del ejemplo anterior, hemos creado tres instancias para nuestra clase: barney, lisa y ralph.

Si intentamos imprimir una de estas instancias, obtendremos el siguiente output:

```
print(lisa)  
# Output: <__main__.SimpsonCharacters object at 0x728e0e40faf0>
```

Como veis, Python nos está indicando que lisa es un objeto de SimpsonCharacters (el nombre de nuestra clase).

## Ventajas y desventajas del uso de clases

Una de las ventajas del uso de clases en python es que podemos reutilizar nuestro código en distintas partes del programa, reduciendo duplicidad y ahorrando tiempo.

Además, permiten descomponer un programa en componentes más pequeños, facilitando la solución de problemas.

En cuanto a las desventajas, hemos de mencionar que las clases pueden añadir complejidad adicional a un programa, haciendo que sea más difícil de entender. Esto es más notorio cuando su uso es innecesario y una función simple podría haber hecho el mismo trabajo.

## Fuentes

Agradecimientos a:

- Guías Devcamp
- <https://docs.python.org/es/3/tutorial/classes.html>
- [https://www.w3schools.com/python/python\\_classes.asp](https://www.w3schools.com/python/python_classes.asp)
- <https://ellibrodepython.com/crear-clase-python>
- <https://blog.hubspot.es/website/clases-python#:~:text=Una%20clase%20en%20Python%20es,en%20un%20programa%20de%20computadora>

# ¿Qué método se ejecuta automáticamente cuando se crea una instancia de una clase?

## Método `__init__` o constructor: qué es

El método `__init__` (dunder init) es el que es llamado automáticamente por el intérprete de Python y se utiliza para realizar cualquier inicialización que sea necesaria para la instancia.

El método `__init__` se usa para asignar valores iniciales a los atributos de una instancia de la clase.

Anteriormente hemos visto los atributos de clase, que son comunes a todas las instancias de una clase. Sin embargo, existen también los atributos de instancia, que pertenecen a cada instancia particular de la clase.

Al llamar al método `__init__`, podemos establecer los valores de estos atributos y configurar la instancia de la clase para su uso posterior.

Veámoslo en nuestro anterior ejemplo. Imaginemos que cada instancia de nuestra clase tiene también dos atributos que consisten en el nombre y la edad del personaje. Deberemos definir la función `__init__` dentro de nuestra clase, tomando primero el argumento `self` y después los mencionados atributos. La sintaxis para definir los atributos dentro de la función es la siguiente:

```
class SimpsonCharacters:

    def __init__(self, name, age):
        self.name = name
        self.age = age

    city = "Springfield"
    type = "Cartoon"

    def creator(self):
        print("Creator: Matt Groening")
```

Así, podemos ver una diferencia clara entre `name` y `age`, que serán variables según el objeto que instanciamos, y `city` y `type`, que siempre serán los mismos para todos los objetos.

## Crear un objeto cuando existen atributos de instancia

Imaginemos que intentamos crear una instancia con la definición de clase anterior:

```
barney = SimpsonCharacters()
```

```
Traceback (most recent call last):
  File "/home/runner/Just-playin/main.py", line 15, in <module>
    barney = SimpsonCharacters()
```

```
TypeError: SimpsonCharacters.__init__() missing 2 required positional arguments:
'name' and 'age'
```

¡Ups! Parece que tenemos un problema. Por suerte, Python nos está chivando que faltan los dos argumentos requeridos por la función `__init__`.

Así, cuando hemos definido atributos de instancia, debemos proporcionarlos cada vez que queramos instanciar un objeto:

```
barney = SimpsonCharacters("Barney Gumble", 40)
lisa = SimpsonCharacters("Lisa Simpson", 8)
ralph = SimpsonCharacters("Ralph Wiggum", 8)
```

## Getters y setters

¿Qué son los métodos Getter y Setter?

- Getter: Método que permite acceder a un atributo de una clase determinada.
- Setter: Un método que te permite establecer o cambiar el valor de un atributo en una clase.

Más adelante veremos también como los getters y setters pueden ayudarnos a acceder a atributos privados. En esta sección veremos un proceso sencillo, pero tiene el inconveniente de ofrecer demasiado acceso. Por lo tanto, deberemos tener en cuenta que esto puede llevar a bugs.

### Getters

Para tener acceso a los valores de atributos que hemos definido, deberemos usar la siguiente sintaxis:

```
print(barney.age)
# Output: 40
print(lisa.name)
# Output: Lisa Simpson
```

También podemos acceder a los atributos de clase con la misma sintaxis:

```
print(ralph.type)
# Output: Cartoon
print(lisa.type)
# Output: Cartoon
```

### Acceder a funciones

Para acceder a las funciones dentro de la clase, usamos la misma sintaxis pero con paréntesis `()` al final. Aunque no es el caso, debemos recordar pasar también los argumentos si la función los requiere:

```
barney.creator()
# Output: Creator: Matt Groening
```

Es raro que los lenguajes de programación permitan este nivel de acceso, normalmente habría que entrar dentro de la clase y crear una función que devolviese los datos.

## Setters

Podemos establecer los valores de los atributos después de haber creado el objeto instanciado.

En el ejemplo hemos instanciado el objeto con el valor para age 8, pero posteriormente lo hemos actualizado a 9.

```
def __init__(self, name, age):
    self.name = name
    self.age = age

city = "Springfield"
type = "Cartoon"

def creator(self):
    print("Creator: Matt Groening")

lisa = SimpsonCharacters("Lisa Simpson", 8)

lisa.age = 9
print(lisa.age)
# Output: 9
```

## Problemas derivados del uso de atributos públicos

Por defecto, los atributos son públicos en Python, lo que significa que puede accederse a ellos desde cualquier lugar del programa. Esto parece algo positivo, pero tiene también la contraparte de que cualquiera puede acceder a ellos y modificarlos, incluso en los casos en los que no debería hacerse.

Por ejemplo, alguien podría modificar el valor del atributo de clase city para una de las instancias, cuando debería ser igual para todas las instancias de la clase:

```
lisa.city = "New York"
print(lisa.city)
# Output: New York
```

O podríamos tener un problema peor. Alguien podría modificar el valor de un atributo cambiando el tipo de datos. Las funciones que fuésemos a ejecutar según el tipo de dato original nos darían un error.

Por ejemplo, imaginemos que alguien cambia el valor de la variable age a un string y no nos damos cuenta. Si en otro punto del programa quisiéramos hacer crecer un año a nuestro personaje, obtendríamos este error:

```
# Una parte del programa
lisa.age = "kid"

# Otra parte del programa
lisa.age += 1
```

```
TypeError: can only concatenate str (not "int") to str
```

Como veremos más adelante, también existen atributos privados, a los cuales solo se puede acceder desde dentro de la clase en la que se definen. En Python, los atributos privados se definen mediante el prefijo «\_\_» seguido del nombre del atributo.

## Fuentes

Agradecimientos a:

- Guías Devcamp
- [https://www.w3schools.com/python/python\\_classes.asp](https://www.w3schools.com/python/python_classes.asp)
- <https://ellibrodepython.com/crear-clase-python>
- [https://www.udacity.com/blog/2021/11/\\_init\\_-in-python-an-overview.html](https://www.udacity.com/blog/2021/11/_init_-in-python-an-overview.html)
- <https://realpython.com/python-getter-setter/>

# ¿Qué es un método dunder?

## Métodos dunder o mágicos: qué son

En Python, los métodos dunder son métodos especiales que tienen un doble guión bajo al principio y al final de su nombre. Dunder es la abreviatura de double underscore.

Los métodos Dunder son métodos que permiten a las instancias de una clase interactuar con las funciones y operadores incorporados en el lenguaje. Típicamente, los métodos dunder no son invocados directamente por el programador, haciendo que parezca que son llamados por arte de magia. Por eso, a veces también se hace referencia a los métodos dunder como "métodos mágicos".

Sin embargo, los métodos dunder no se invocan por arte de magia. Simplemente son llamados implícitamente por el lenguaje, en momentos específicos que están bien definidos, y que dependen del método dunder en cuestión.

## Algunos de los métodos dunder más comunes

### `__init__`

Como hemos visto en el apartado correspondiente, es el método dunder que se utiliza para inicializar una instancia de una clase. Cuando se crea una instancia de una clase, el método **init** es llamado automáticamente por el intérprete de Python.

### `__str__`

Es un método dunder que se utiliza para devolver un string de una instancia de una clase. Este método se llama cuando se usa la función `str()` para imprimir strings de aspecto agradable para los usuarios finales, una representación legible de tu objeto.

En el siguiente ejemplo, tenemos una clase para una tienda, que tiene los atributos `product` y `sales`.

```
class Store:
    def __init__(self, product, sales):
        self.product = product
        self.sales = sales

    def __str__(self):
        return f'We have sold {self.sales} units of {self.product}.'

chocolate = Store('chocolate', 100)

print(str(chocolate))
# Output: We have sold 100 units of chocolate.
```

Hemos añadido el método **str** para que nos de un string de lectura agradable sobre los atributos de cada instancia.

Fijaos que no llamamos a la función mediante **str**, sino que usamos `str()` pasando como argumento nuestra instancia.



## `__repr__`

Así como `str_`, el método **repr** se utiliza también para convertir una representación de un objeto a un string. La diferencia es que **repr** debe representar sin ambigüedades el objeto, preferiblemente proporcionando una expresión que pueda utilizarse para reconstruir el objeto.

Los desarrolladores pueden utilizar `repr` porque necesitan depurar código y tienen que asegurarse de que saben lo que están viendo.

Siguiendo con el ejemplo anterior, podemos incluir la función **repr** dentro de la clase, para que apreciemos la diferencia con **str**:

```
class Store:
    def __init__(self, product, sales):
        self.product = product
        self.sales = sales

    def __str__(self):
        return f'We have sold {self.sales} units of {self.product}.'

    def __repr__(self):
        return f"Store('{self.product}', {self.sales})"

chocolate = Store('chocolate', 100)

print(str(chocolate))
# Output: We have sold 100 units of chocolate.
print(repr(chocolate))
# Output: Store('chocolate', 100)
```

De forma similar al método `str`, la función `__repr__` se llama mediante la sintaxis `repr()` tomando como argumento el objeto instanciado.

## `__add__`

El método **add** permite la definición de un comportamiento personalizado para el operador `+` en nuestra clase. Suma los atributos correspondientes de los objetos que contiene el resultado.

Una vez definido, podemos sumar dos objetos de una clase entre sí.

Vamos a añadirlo a nuestro anterior ejemplo:

```
class Store:
    def __init__(self, product, sales):
        self.product = product
        self.sales = sales

    def __str__(self):
        return f'We have sold {self.sales} units of {self.product}.'

    def __repr__(self):
        return f"Store('{self.product}', {self.sales})"
```

```
def __add__(self, other):
    return Store(self.product + other.product, self.sales + other.sales)

chocolate = Store('chocolate', 100)
cookies = Store('cookies', 50)

object_sum = chocolate + cookies

print(object_sum.sales)
# Output: 150
print(object_sum.product)
# Output: chocolatecookies
```

El uso de la palabra `other` como segundo argumento de la función `__add__` es solo una convención, el código funcionará si usamos también otra palabra.

También debemos tener en cuenta la utilidad que queramos darle a la función. Como podréis ver, sumar las ventas de los objetos puede tener sentido, pero sumar los nombres de los productos, no tanto.

## Listado métodos dunder

La siguiente tabla, tomada prestada directamente de <https://www.tutorialsteacher.com/python/magic-methods-in-python>, nos muestra un listado de métodos dunder importantes en Python 3.

Initialization and Construction	Description
<code>__new__(cls, other)</code>	To get called in an object's instantiation.
<code>__init__(self, other)</code>	To get called by the <code>__new__</code> method.
<code>__del__(self)</code>	Destructor method.

Unary operators and functions	Description
<code>__pos__(self)</code>	To get called for unary positive e.g. <code>+someobject</code> .
<code>__neg__(self)</code>	To get called for unary negative e.g. <code>-someobject</code> .
<code>__abs__(self)</code>	To get called by built-in <code>abs()</code> function.
<code>__invert__(self)</code>	To get called for inversion using the <code>~</code> operator.
<code>__round__(self,n)</code>	To get called by built-in <code>round()</code> function.
<code>__floor__(self)</code>	To get called by built-in <code>math.floor()</code> function.
<code>__ceil__(self)</code>	To get called by built-in <code>math.ceil()</code> function.
<code>__trunc__(self)</code>	To get called by built-in <code>math.trunc()</code> function.

Augmented Assignment	Description
<code>__iadd__(self, other)</code>	To get called on addition with assignment e.g. <code>a += b</code> .

<code>__isub__(self, other)</code>	To get called on subtraction with assignment e.g. <code>a -=b.</code>
<code>__imul__(self, other)</code>	To get called on multiplication with assignment e.g. <code>a *=b.</code>
<code>__ifloordiv__(self, other)</code>	To get called on integer division with assignment e.g. <code>a //=b.</code>
<code>__idiv__(self, other)</code>	To get called on division with assignment e.g. <code>a /=b.</code>
<code>__itruediv__(self, other)</code>	To get called on true division with assignment
<code>__imod__(self, other)</code>	To get called on modulo with assignment e.g. <code>a%=b.</code>
<code>__ipow__(self, other)</code>	To get called on exponentswith assignment e.g. <code>a **=b.</code>
<code>__ilshift__(self, other)</code>	To get called on left bitwise shift with assignment e.g. <code>a&lt;=&lt;b.</code>
<code>__irshift__(self, other)</code>	To get called on right bitwise shift with assignment e.g. <code>a &gt;&gt;=b.</code>
<code>__iand__(self, other)</code>	To get called on bitwise AND with assignment e.g. <code>a&amp;=b.</code>
<code>__ior__(self, other)</code>	To get called on bitwise OR with assignment e.g. <code>a =b.</code>
<code>__ixor__(self, other)</code>	To get called on bitwise XOR with assignment e.g. <code>a ^=b.</code>

Type Conversion Magic Methods	Description
<code>__int__(self)</code>	To get called by built-int <code>int()</code> method to convert a type to an int.
<code>__float__(self)</code>	To get called by built-int <code>float()</code> method to convert a type to float.
<code>__complex__(self)</code>	To get called by built-int <code>complex()</code> method to convert a type to complex.
<code>__oct__(self)</code>	To get called by built-int <code>oct()</code> method to convert a type to octal.
<code>__hex__(self)</code>	To get called by built-int <code>hex()</code> method to convert a type to hexadecimal.
<code>__index__(self)</code>	To get called on type conversion to an int when the object is used in a slice expression.
<code>__trunc__(self)</code>	To get called from <code>math.trunc()</code> method.

String Magic Methods	Description
<code>__str__(self)</code>	To get called by built-int <code>str()</code> method to return a string representation of a type.
<code>__repr__(self)</code>	To get called by built-int <code>repr()</code> method to return a machine

	readable representation of a type.
<code>__unicode__(self)</code>	To get called by built-in <code>unicode()</code> method to return an unicode string of a type.
<code>__format__(self, formatstr)</code>	To get called by built-in <code>string.format()</code> method to return a new style of string.
<code>__hash__(self)</code>	To get called by built-in <code>hash()</code> method to return an integer.
<code>__nonzero__(self)</code>	To get called by built-in <code>bool()</code> method to return True or False.
<code>__dir__(self)</code>	To get called by built-in <code>dir()</code> method to return a list of attributes of a class.
<code>__sizeof__(self)</code>	To get called by built-in <code>sys.getsizeof()</code> method to return the size of an object.

Attribute Magic Methods	Description
<code>__getattr__(self, name)</code>	Is called when the accessing attribute of a class that does not exist.
<code>__setattr__(self, name, value)</code>	Is called when assigning a value to the attribute of a class.
<code>__delattr__(self, name)</code>	Is called when deleting an attribute of a class.

Operator Magic Methods	Description
<code>__add__(self, other)</code>	To get called on add operation using + operator
<code>__sub__(self, other)</code>	To get called on subtraction operation using - operator.
<code>__mul__(self, other)</code>	To get called on multiplication operation using * operator.
<code>__floordiv__(self, other)</code>	To get called on floor division operation using // operator.
<code>__truediv__(self, other)</code>	To get called on division operation using / operator.
<code>__mod__(self, other)</code>	To get called on modulo operation using % operator.
<code>__pow__(self, other[, modulo])</code>	To get called on calculating the power using ** operator.
<code>__lt__(self, other)</code>	To get called on comparison using < operator.
<code>__le__(self, other)</code>	To get called on comparison using <= operator.
<code>__eq__(self, other)</code>	To get called on comparison using == operator.
<code>__ne__(self, other)</code>	To get called on comparison using != operator.
<code>__ge__(self, other)</code>	To get called on comparison using >= operator.

## Fuentes

Agradecimientos a:

- Guías Devcamp
- <https://barcelonageeks.com/dunder-o-metodos-magicos-en-python>
- <https://mathspp.com/blog/pydents/dunder-methods>
- <https://blog.hubspot.es/website/clases-python#:~:text=Una%20clase%20en%20Python%20es,en%20un%20programa%20de%20computadora>
- <https://pythondiario.com/2018/06/metodos-magicos-programacion-en-python.html>
- <https://www.tutorialsteacher.com/python/magic-methods-in-python> [<https://mathspp.com/blog/pydents/dunder-methods>]

# ¿Qué es un decorador de python?

## Decoradores - qué son

Los decoradores son funciones que modifican el comportamiento de otras funciones.

Al utilizar la palabra decorar estamos indicando que queremos modificar el comportamiento de una función ya existente, pero sin tener que modificar su código. Una analogía tomada prestada directamente de

<https://codigofacilito.com/articulos/decoradores-python> indica que podemos ver el tema como un pastel, donde en ocasiones la base del pastel no es suficiente para una fiesta y debemos añadir elementos extras, quizás glaseado, velas, aderezos etc ... de esta forma el pastel tendrá mejor aspecto y sabrá mejor. En la analogía la base del pastel no será más que nuestra función a decorar y los elementos extras los decoradores.

Un decorador no es más que una función la cual toma como input una función y a su vez retorna otra función.

## Cómo crear un decorador

Veamos un ejemplo sencillo. Queremos decorar la función `ventas_totales` usando `ejemplo_decorador()`.

La función `ejemplo_decorador()` define una nueva función que envuelve la función que se pasa como entrada. Concretamente, imprime dos strings, antes y después de la llamada a la función.

Para decorar una función basta con colocar en la parte superior el decorador con el prefijo `@`.

```
def ejemplo_decorador(funcion):
    def nueva_funcion(a, b):
        print("Antes de llamar a la función")
        c = funcion(a, b)
        print("Después de llamar a la función")
        return c
    return nueva_funcion

@ejemplo_decorador
def ventas_totales(a, b):
    print("Se está ejecutando la función ventas_totales()")
    return a + b

print(ventas_totales(100, 200))
```

Y obtendremos como output:

```
Antes de llamar a la función
Se está ejecutando la función ventas_totales()
Después de llamar a la función
300
```

## Algunos decoradores integrados en Python

Hemos visto como podemos definir nuestros propios decoradores, pero hay algunos que ya están integrados en Python.

Algunos de los más comunes son:

**@staticmethod:** Se utiliza en una clase para indicarte que el método que está decorando es un método estático (pueden ejecutarse sin crear instancias de la clase)

```
class Magikarp:
    @staticmethod
    def salpicadura():
        print('No pasó nada...')

Magikarp.salpicadura()
# Output: No pasó nada...
```

**@classmethod:** Se usa para indicarte que el método que está decorando es un método de clase. Los métodos de clase tampoco requieren que la clase sea instanciada antes de que puedan ser llamados, pero al contrario que los métodos estáticos, sí tienen acceso a los atributos de clase.

**@property:** se utiliza para etiquetar un método como definidor de propiedades.

## Decorador @property

### Conocimiento previo: atributos privados y protegidos

Como hemos mencionado en el apartado de getters y setters de atributos, el hecho de tener acceso a todos los datos dentro de una clase se puede considerar una mala práctica, y puede dar lugar a errores.

Los atributos a los que solo se debe acceder dentro de la clase, ni siquiera las clases anidadas, se llaman atributos privados y la convención común es añadir dos guiones bajos antes de su nombre.

Los atributos protegidos, por otro lado, se establecen añadiendo un guion bajo delante del nombre del atributo. Son similares a los atributos privados, porque no deben ser accedidos directamente desde fuera de la clase. Sin embargo, se puede acceder a los atributos protegidos desde las subclases.

Es importante destacar que el uso de atributos privados o protegidos nos ayuda a que nuestros datos no sean accidentalmente sobrescritos, pero no protegen frente a las infracciones intencionadas.

Volvamos a nuestro ejemplo de clase Store y protejamos los atributos.

```
class Store:
    def __init__(self, product, sales):
        self._product = product
        self._sales = sales

    def __str__(self):
        return f'We have sold {self._sales} units of {self._product}.'
```

```
def __repr__(self):
    return f"Store('{self._product}', {self._sales})"
```

```
chocolate = Store('chocolate', 100)
```

Si intentásemos acceder a alguno de los valores usando la sintaxis habitual, obtendríamos un error.

```
print(chocolate.sales)
# Output: AttributeError: 'Store' object has no attribute 'sales'. Did you mean:
'_sales'?
```

Por supuesto, siempre podríamos intentar acceder al atributo añadiendo simplemente el guion bajo, pero esto sería una mala práctica, ya que el código nos está indicando que no lo hagamos:

```
# Bad practice
print(chocolate._sales)
# Output: 100
```

## Uso del decorador @property

El decorador de propiedad puede ser usado sobre un método, que hará que actúe como si fuera un atributo.

Así, podemos agragar getters y setters "behind the scenes" y poder utilizar la sintaxis utilizada para acceder a los atributos cuando eran públicos.

Para añadir una propiedad, deberemos añadir @property dentro de la clase. El código introducido en el ejemplo nos permite acceder a la variable sales directamente:

```
class Store:
    def __init__(self, product, sales):
        self._product = product
        self._sales = sales

    def __str__(self):
        return f'We have sold {self._sales} units of {self._product}.'

    def __repr__(self):
        return f"Store('{self._product}', {self._sales})"

    @property
    def sales(self):
        return self._sales
```

Así, si intentamos acceder a sales, podemos obtener su valor:

```
chocolate = Store('chocolate', 100)

print(chocolate.sales)
# Output: 100
```



Pero si intentamos cambiar el valor del atributo, seguiremos obteniendo un error:

```
chocolate.sales = 200
# Output: AttributeError: can't set attribute 'sales'
```

Imaginemos que queremos ofrecer la habilidad de acceder a los datos de product y sales, pero solo queremos dar la opción de sobrescribir el valor de sales y no de product.

Como véis, hemos añadido el decorador @property para ambos atributos, pero la función setter (el decorador @sales.setter) solo ha sido añadido para el atributo sales.

```
class Store:
    def __init__(self, product, sales):
        self._product = product
        self._sales = sales

    def __str__(self):
        return f'We have sold {self._sales} units of {self._product}.'

    def __repr__(self):
        return f"Store('{self._product}', {self._sales})"

    @property
    def sales(self):
        return self._sales

    @sales.setter
    def sales(self, sales):
        self._sales = sales

    @property
    def product(self):
        return self._product

chocolate = Store('chocolate', 100)
```

Por lo tanto, vamos a poder consultar ambos atributos, pero solo podremos cambiar el valor de sales.

```
print(chocolate.sales)
#Output: 100
chocolate.sales = 200
print(chocolate.sales)
#Output: 200
print(chocolate.product)
#Output: chocolate
chocolate.product = 'candy'
print(chocolate.product)
#Output: AttributeError: can't set attribute 'product'
```

Como conclusión, cuando vemos un decorador de propiedad, tenemos que pensar que vamos a querer acceder a esos datos en algún momento. Si sólo vemos atributos con `_` delante del nombre pero no decoradores de propiedad, esto significa que los datos son para el uso interno de la clase.

## Fuentes

Agradecimientos a:

- Guías Devcamp
- <https://geekflare.com/es/python-decorators/>
- <https://www.geeksforgeeks.org/decorators-in-python/>
- <https://codigofacilito.com/articulos/decoradores-python>
- <https://ellibrodepython.com/decoradores-python> [<https://www.geeksforgeeks.org/decorators-in-python/>]

# ¿Qué es el polimorfismo?

## Conocimiento previo: Herencia

Antes de ver en qué consiste el polimorfismo en Python, aclaremos primero el concepto de herencia.

Como concepto general, la herencia es la capacidad de crear versiones especializadas de las clases. Es un proceso mediante el cual se puede crear una clase hija que hereda de una clase padre, compartiendo sus métodos y atributos. La clase hija puede sobrescribir los métodos o atributos, o definir unos nuevos.

Para implementar la herencia, la sintaxis consiste en añadir paréntesis después del nombre de la clase hija al definirla, y dentro se pasa el nombre de la clase padre.

En el siguiente ejemplo tenemos una clase padre que es el genérico `Animal`, con un atributo y una función. Hemos definido también la clase hija `Horse`, que heredará el atributo y el método de la clase `Animal`:

```
class Animal:
    def __init__(self, weight):
        self.weight = weight

    def feed(self):
        self.weight += 1
        print(f"That was delicious. New weight is {self.weight} kg")

class Horse(Animal):
    def speak(self):
        print('¡Hiiii!')
```

Vamos a instanciar un objeto y acceder a métodos y atributos de la clase hija y la clase padre:

```
bojack_horseman = Horse(400)

bojack_horseman.speak()
# Output: ¡Hiiii!
print(bojack_horseman.weight)
# Output: 400
bojack_horseman.feed()
# Output: That was delicious. New weight is 401 kg
```

Hemos visto que la instancia de la clase hija tiene acceso a la clase padre.

Pero, ¿y al contrario? Creemos una instancia de la clase padre `Animal` e intentemos realizar las mismas operaciones:

```
anonymous_animal = Animal(5)

anonymous_animal.speak()
# Output: AttributeError: 'Animal' object has no attribute 'speak'
print(anonymous_animal.weight)
# Output: 5
```

```
anonymous_animal.feed()  
# Output: That was delicious. New weight is 6 kg
```

Como podréis ver, la instancia de la clase padre tiene acceso a métodos y atributos de la clase padre, pero obtenemos un error si intentamos acceder a la clase hija.

## Polimorfismo

El término polimorfismo tiene origen en las palabras poly (muchos) y morfo (formas), y aplicado a la programación hace referencia a que los objetos pueden tomar diferentes formas, lo cual significa que un objeto se puede comportar de manera diferente en distintos contextos.

Para implementar el polimorfismo, simplemente podemos definir diferentes métodos que compartan el mismo nombre pero que realicen operaciones distintas.

Veamoslo con nuestro anterior ejemplo. Hemos creado un método llamado speak en la clase padre Animal, y haremos que lance un error porque no queremos que nadie llame a la clase Animal, ya que es una clase abstracta. Tiene el único propósito de mantener y almacenar el comportamiento compartido. Sólo las clases hijas van a llamar a esta clase.

Hemos añadido la función speak() a cada una de nuestras clases hijas, por lo que sobrescribirán el comportamiento de la función en la clase padre.

```
class Animal:  
    def __init__(self, weight):  
        self.weight = weight  
  
    def feed(self):  
        self.weight += 1  
        print(f"That was delicious. New weight is {self.weight} kg")  
  
    def speak(self):  
        raise NotImplementedError('Subclass must implement speak method')  
  
class Horse(Animal):  
    def speak(self):  
        print('¡Hiiii!')  
  
class Cat(Animal):  
    def speak(self):  
        print('Miau')  
  
class Dog(Animal):  
    def speak(self):  
        print('Guau Guau')
```

Veamos como se comporta la función speak() para cada instancia, tomando una forma distinta, es decir, en este caso imprimiendo un texto distinto, para cada clase hija.

```
bojack_horseman = Horse(400)  
princess_carolyn = Cat(4)  
mr_peanutbutter = Dog(10)
```

```

bojack_horseman.speak()
# Output: ¡Hiiii!
princess_carolyn.speak()
# Output: Miau
mr_peanutbutter.speak()
# Output: Guau Guau

anon = Animal(5)
anon.speak()
# Output: NotImplementedError: Subclass must implement speak method

```

### Otra opción: construir funciones polimórficas

En lugar de crear una clase abstracta y tener varias clases que hereden de ella, tenemos la opción de crear clases independientes y crear además una función donde ocurra el polimorfismo.

Vamos a reorganizar el código de nuestro anterior ejemplo, de forma que tenemos 3 clases, cada una con su función dunder init y speak.

Además, creamos la función speaker, de forma que cuando la llamemos tomando como argumento cada una de nuestras instancias, llamará a su vez a la función speak:

```

class Horse():
    def __init__(self, weight):
        self.weight = weight

    def speak(self):
        print('¡Hiiii!')

class Cat():
    def __init__(self, weight):
        self.weight = weight

    def speak(self):
        print('Miau')

class Dog():
    def __init__(self, weight):
        self.weight = weight

    def speak(self):
        print('Guau Guau')

def speaker(animal_obj):
    return animal_obj.speak()

bojack_horseman = Horse(400)
princess_carolyn = Cat(4)
mr_peanutbutter = Dog(10)

speaker(bojack_horseman)

```

```
# Output: ¡Hiiii!  
speaker(princess_carolyn)  
# Output: Miau  
speaker(mr_peanutbutter)  
# Output: Guau Guau
```

Aunque creo que el ejemplo de los animales es útil para entender el concepto, tal vez no sea muy útil para aplicarlo a la vida real. Puede que el siguiente ejemplo os pueda dar una pista de cómo podrían usarse estas herramientas para dar solución a requerimientos más realistas:

```
class Sociedad():  
    def __init__(self, ventas, costes):  
        self.ventas = ventas  
        self.costes = costes  
  
    def beneficio_despues_impuestos(self):  
        return (self.ventas - self.costes) * 0.7  
  
class Cooperativa():  
    def __init__(self, ventas, costes):  
        self.ventas = ventas  
        self.costes = costes  
  
    def beneficio_despues_impuestos(self):  
        return (self.ventas - self.costes) * 0.85  
  
def calculador_beneficio(empresa):  
    return empresa.beneficio_despues_impuestos()  
  
floristeria = Sociedad(1000, 500)  
academias = Cooperativa(1000, 500)  
  
print(calculador_beneficio(floristeria))  
# Output: 350.0  
print(calculador_beneficio(academias))  
# Output: 425.0
```

### ¿Cuándo usar clases abstractas vs funciones polimórficas?

Cuando tenemos mucho comportamiento compartido, será una buena idea hacer uso de la herencia. Por ejemplo, si la clase padre tiene 5 funciones que las clases hijas también deberían compartir de otro modo.

Si no tenemos mucho comportamiento compartido, pero simplemente queremos ser capaces de llamar la misma función, utilizar el polimorfismo con un enfoque basado en funciones es una buena manera de hacerlo.

## Fuentes

Agradecimientos a:

- Guías Devcamp

- <https://desarrolloweb.com/articulos/polimorfismo-programacion-orientada-objetos-concepto.html>
- <https://ellibrodepython.com/programacion-orientada-a-objetos-python>
- [https://www.w3schools.com/python/python\\_polymorphism.asp](https://www.w3schools.com/python/python_polymorphism.asp)
- <https://blog.hubspot.es/website/clases-python#:~:text=Una%20clase%20en%20Python%20es,en%20un%20programa%20de%20computadora>

# ¿Qué es una API?

API es el acrónimo de "application programming interface" o interfaz de programación de aplicaciones.

Una API es un servicio similar a un sitio web o un servidor con el que puedes comunicarte, pero en lugar de recibir una página web, recibes datos.

## ¿Cómo funcionan las API?

La arquitectura de las API suele explicarse en términos de cliente y servidor. La aplicación que envía la solicitud se llama cliente, y la que envía la respuesta se llama servidor. Las API pueden funcionar de cuatro maneras diferentes, según el momento y el motivo de su creación.

- API de SOAP: utilizan el protocolo simple de acceso a objetos. El cliente y el servidor intercambian mensajes mediante XML. Se trata de una API menos flexible que era más popular en el pasado.
- API de RPC: se denominan llamadas a procedimientos remotos. El cliente completa una función (o procedimiento) en el servidor, y el servidor devuelve el resultado al cliente.
- API de WebSocket: otro desarrollo moderno de la API web que utiliza objetos JSON para transmitir datos
- API de REST: las API más populares y flexibles que se encuentran en la web actualmente. El cliente envía las solicitudes al servidor como datos. El servidor utiliza esta entrada del cliente para iniciar funciones internas y devuelve los datos de salida al cliente.

## Clasificación según ámbito de uso

Hay cuatro tipos de API según el ámbito de uso:

API privadas: son internas de una empresa y solo se utilizan para conectar sistemas y datos dentro de la empresa.

API públicas: están abiertas al público y pueden cualquier persona puede utilizarlas. Puede haber o no alguna autorización y coste asociado a este tipo de API.

API de socios: solo pueden acceder a ellas los desarrolladores externos autorizados para ayudar a las asociaciones entre empresas.

API compuestas: combinan dos o más API diferentes para abordar requisitos o comportamientos complejos del sistema. #

## ¿Qué es un punto final de API?

Un punto final de API es la ubicación de la API en la que un sistema interactúa con una API web. También es el punto de comunicación entre dos sistemas.

Es la URL específica que se utiliza para acceder a un recurso proporcionado por una aplicación web desde una API. El punto final se refleja como un buscador uniforme de recursos (URL), similar a la URL de un sitio web, donde los datos se transmiten de un programa a otro.



La URL del punto final es la ubicación exacta del recurso solicitado en un servidor API, permitiendo así que dos programas interactúen. En el punto final, la API accederá a los recursos que necesite de un servidor para realizar una tarea específica, como la obtención de ciertos datos o información.

Las API envían solicitudes para acceder a los datos desde un servidor y reciben una respuesta. La ubicación de la respuesta es el punto final, y es una parte importante de cualquier documentación porque indica a los desarrolladores cómo realizar solicitudes de API.

## APIs públicas para practicar

Con el fin de practicar y familiarizarte con las API, desde <https://ed.team/blog/las-mejores-apis-publicas-para-practicar> nos proporcionan 6 APIs públicas y gratuitas:

- API de Marvel: <http://developer.marvel.com/>;
- PokéAPI: <https://pokeapi.co/>;
- COVID Tracking: <https://covidtracking.com/data/>;
- Nomics: <https://nomics.com/>;
- OpenWeather APIs: <https://openweathermap.org/api>;
- JSON placeholder: <https://jsonplaceholder.typicode.com/>

Vamos a ver un ejemplo de dos de ellas, para comprobar qué nos encontraremos al consultarlas

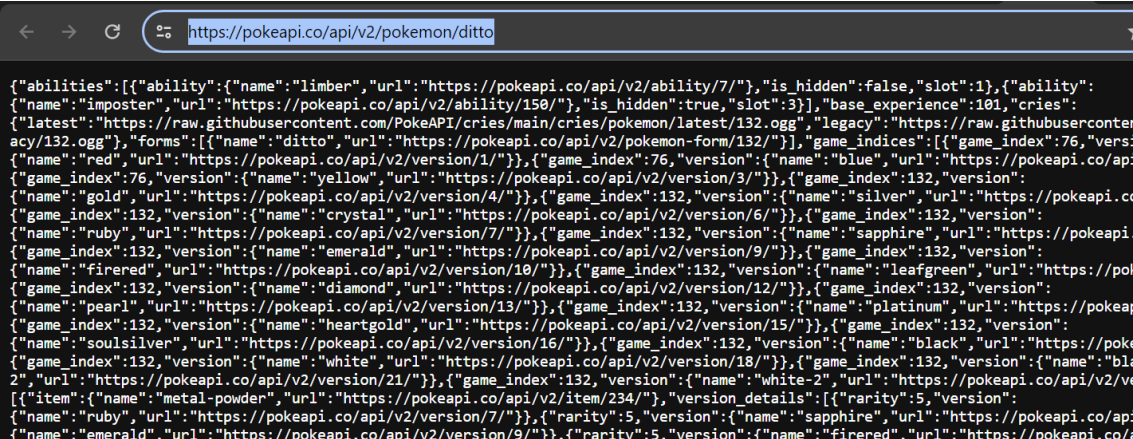
### PokéAPI

Si como yo eres una buena millennial y no puedes resistirte al universo Pokémon, puedes acceder a la API mediante el enlace anteriormente proporcionado.

En la documentación de la API (<https://pokeapi.co/docs/v2>) encontramos indicaciones sobre su uso. Se nos indica además que la API es solo para consumo, por lo que solo está disponible el método HTTP GET. Hablaremos de los verbos HTTP en la siguiente sección.

Si accedemos al recurso de prueba que nos indican

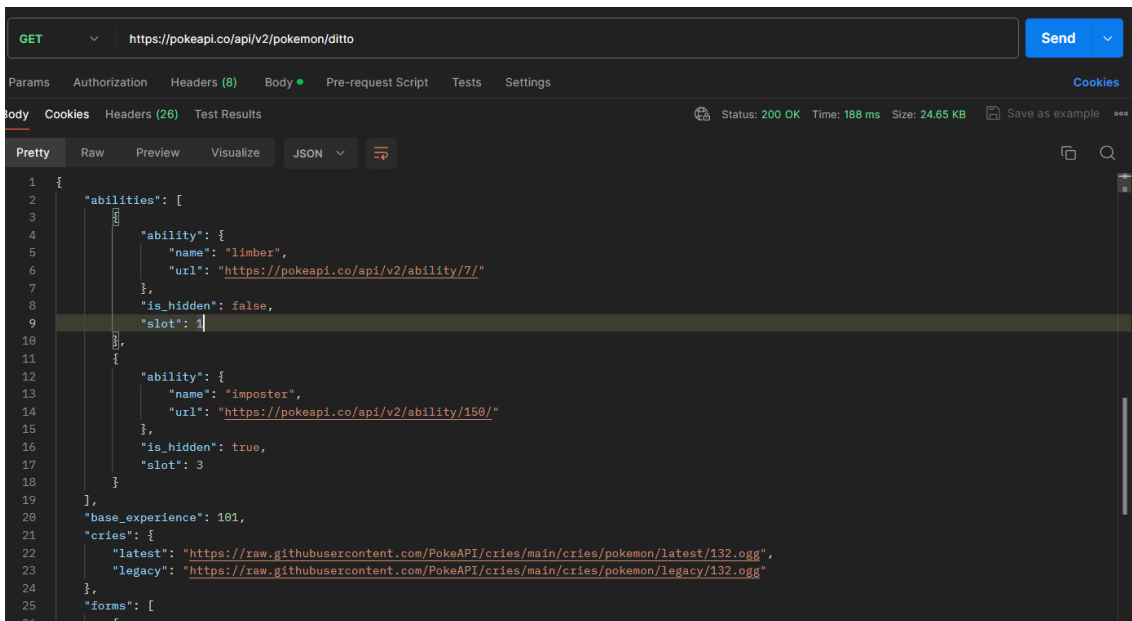
(<https://pokeapi.co/api/v2/pokemon/ditto>), encontraremos el siguiente código:



```
{
  "abilities": [
    {
      "ability": {
        "name": "limber",
        "url": "https://pokeapi.co/api/v2/ability/7/"
      },
      "is_hidden": false,
      "slot": 1
    },
    {
      "ability": {
        "name": "imposter",
        "url": "https://pokeapi.co/api/v2/ability/150/"
      },
      "is_hidden": true,
      "slot": 3
    }
  ],
  "base_experience": 101,
  "cries": {
    "latest": "https://raw.githubusercontent.com/PokeAPI/cries/main/cries/pokemon/latest/132.ogg",
    "legacy": "https://raw.githubusercontent.com/PokeAPI/cries/main/cries/pokemon-form/132.ogg"
  },
  "forms": [
    {
      "name": "ditto",
      "url": "https://pokeapi.co/api/v2/pokemon-form/132/"
    }
  ],
  "game_indices": [
    {
      "game_index": 76,
      "version": {
        "name": "red",
        "url": "https://pokeapi.co/api/v2/version/1/"
      }
    },
    {
      "game_index": 76,
      "version": {
        "name": "blue",
        "url": "https://pokeapi.co/api/v2/version/2/"
      }
    },
    {
      "game_index": 76,
      "version": {
        "name": "yellow",
        "url": "https://pokeapi.co/api/v2/version/3/"
      }
    },
    {
      "game_index": 132,
      "version": {
        "name": "gold",
        "url": "https://pokeapi.co/api/v2/version/4/"
      }
    },
    {
      "game_index": 132,
      "version": {
        "name": "silver",
        "url": "https://pokeapi.co/api/v2/version/5/"
      }
    },
    {
      "game_index": 132,
      "version": {
        "name": "crystal",
        "url": "https://pokeapi.co/api/v2/version/6/"
      }
    },
    {
      "game_index": 132,
      "version": {
        "name": "ruby",
        "url": "https://pokeapi.co/api/v2/version/7/"
      }
    },
    {
      "game_index": 132,
      "version": {
        "name": "sapphire",
        "url": "https://pokeapi.co/api/v2/version/8/"
      }
    },
    {
      "game_index": 132,
      "version": {
        "name": "emerald",
        "url": "https://pokeapi.co/api/v2/version/9/"
      }
    },
    {
      "game_index": 132,
      "version": {
        "name": "firered",
        "url": "https://pokeapi.co/api/v2/version/10/"
      }
    },
    {
      "game_index": 132,
      "version": {
        "name": "leafgreen",
        "url": "https://pokeapi.co/api/v2/version/11/"
      }
    },
    {
      "game_index": 132,
      "version": {
        "name": "diamond",
        "url": "https://pokeapi.co/api/v2/version/12/"
      }
    },
    {
      "game_index": 132,
      "version": {
        "name": "pearl",
        "url": "https://pokeapi.co/api/v2/version/13/"
      }
    },
    {
      "game_index": 132,
      "version": {
        "name": "platinum",
        "url": "https://pokeapi.co/api/v2/version/14/"
      }
    },
    {
      "game_index": 132,
      "version": {
        "name": "heartgold",
        "url": "https://pokeapi.co/api/v2/version/15/"
      }
    },
    {
      "game_index": 132,
      "version": {
        "name": "soulsilver",
        "url": "https://pokeapi.co/api/v2/version/16/"
      }
    },
    {
      "game_index": 132,
      "version": {
        "name": "black",
        "url": "https://pokeapi.co/api/v2/version/17/"
      }
    },
    {
      "game_index": 132,
      "version": {
        "name": "white",
        "url": "https://pokeapi.co/api/v2/version/18/"
      }
    },
    {
      "game_index": 132,
      "version": {
        "name": "black-2",
        "url": "https://pokeapi.co/api/v2/version/19/"
      }
    },
    {
      "game_index": 132,
      "version": {
        "name": "white-2",
        "url": "https://pokeapi.co/api/v2/version/20/"
      }
    },
    {
      "game_index": 132,
      "version": {
        "name": "metal-powder",
        "url": "https://pokeapi.co/api/v2/item/234/"
      }
    }
  ],
  "version_details": [
    {
      "rarity": 5,
      "version": {
        "name": "ruby",
        "url": "https://pokeapi.co/api/v2/version/7/"
      }
    },
    {
      "rarity": 5,
      "version": {
        "name": "sapphire",
        "url": "https://pokeapi.co/api/v2/version/8/"
      }
    },
    {
      "rarity": 5,
      "version": {
        "name": "firered",
        "url": "https://pokeapi.co/api/v2/version/10/"
      }
    }
  ]
}
```

Esto es código JSON, y presentado de esta forma es muy difícil de leer.

Podemos usar Postman, que es una herramienta que explicaremos en una de las siguientes secciones, para obtener un resultado más legible:



## JSON placeholder

JSON placeholder es una API gratuita que te ofrece datos ficticios como fotos, publicaciones, comentarios, datos de usuarios falsos...

Podemos acceder a través del enlace <https://jsonplaceholder.typicode.com/>. Al contrario que la anterior API, esta soporta todos los métodos HTTP:

# Routes

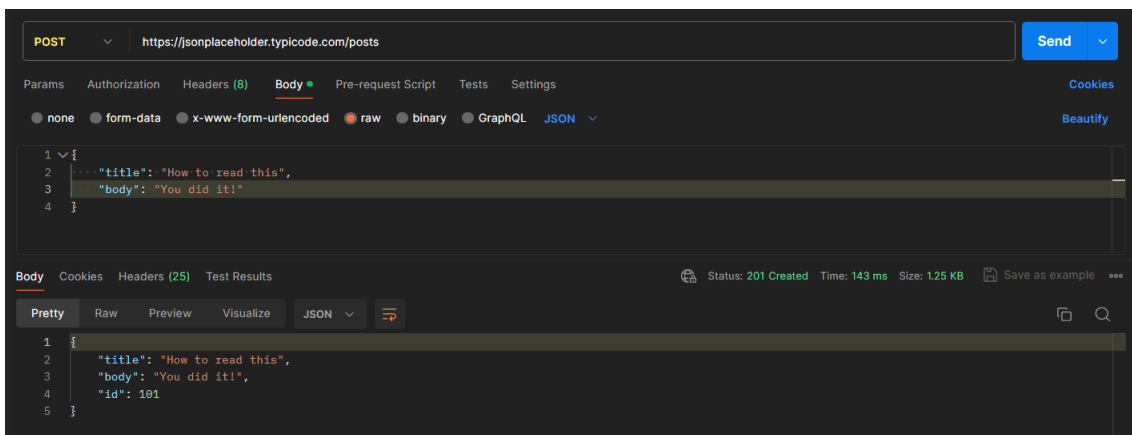
All HTTP methods are supported. You can use http or https for your requests.

GET	<u>/posts</u>
GET	<u>/posts/1</u>
GET	<u>/posts/1/comments</u>
GET	<u>/comments?postId=1</u>
POST	/posts
PUT	/posts/1
PATCH	/posts/1
DELETE	/posts/1

**Note:** see [guide](#) for usage examples.

Aunque en la propia guía se nos indica que los recursos no se actualizaran realmente en el servidor, sino que se fingirá como si así fuese.

A continuación tenemos el ejemplo de crear un nuevo post mediante el verbo POST en Postman:



## Fuentes

Agradecimientos a:

- Guías Devcamp
- <https://aws.amazon.com/es/what-is/api/&#x20;>
- <https://www.xataka.com/basics/api-que-sirve/&#x20;>
- <https://ed.team/blog/las-mejores-apis-publicas-para-practicar>

- <https://mailchimp.com/es/resources/what-is-an-api-endpoint/>

# ¿Cuáles son los tres verbos de API?

## ¿Qué es una API REST?

Como hemos mencionado previamente, una API REST es una interfaz de comunicación entre sistemas de información que usa el protocolo de transferencia de hipertexto (hypertext transfer protocol o HTTP, por su siglas en inglés) para obtener datos o ejecutar operaciones sobre dichos datos en diversos formatos, como pueden ser XML o JSON.

## ¿Qué son los verbos HTTP?

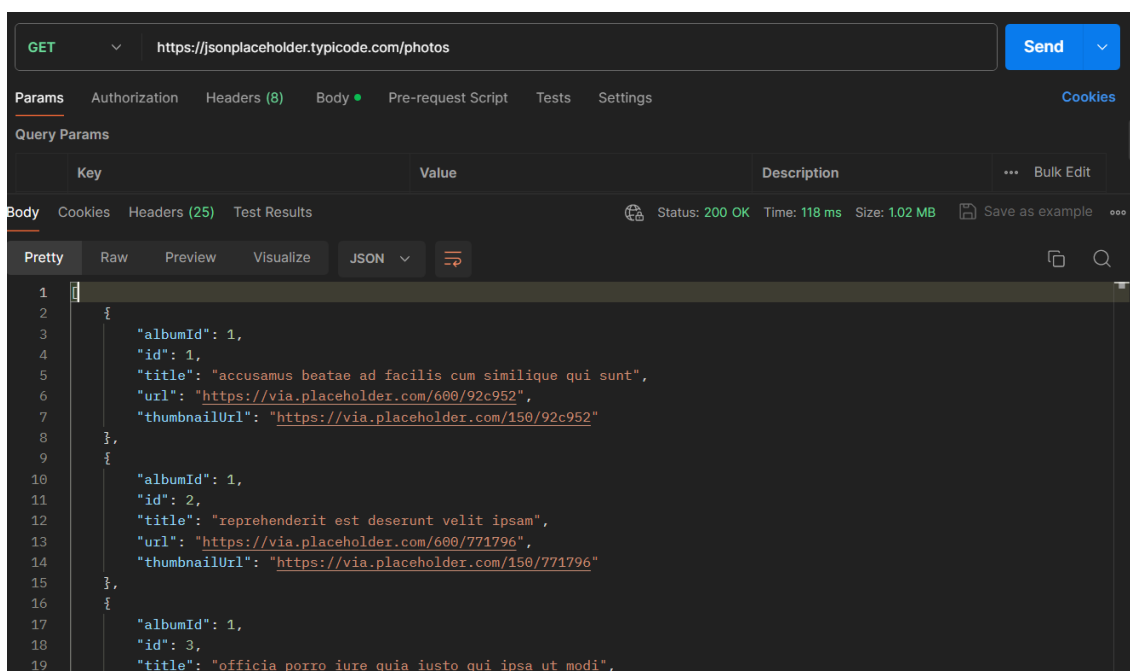
Son aquellos verbos propios del protocolo HTTP que fueron tomados para definir operaciones muy puntuales y específicas sobre los recursos de la API.

Los verbos Http involucrados en un sistema REST son GET, POST, PUT, PATCH y DELETE.

## GET

GET es el que usamos para consultar un recurso. Una de las principales características de una petición GET es que no debe causar efectos secundarios en el servidor, no deben producir nuevos registros, ni modificar los ya existentes. A esta cualidad la llamamos idempotencia, cuando una acción ejecutada un número indefinido de veces, produce siempre el mismo resultado.

Ejemplo de uso de GET con <https://jsonplaceholder.typicode.com/> y Postman:



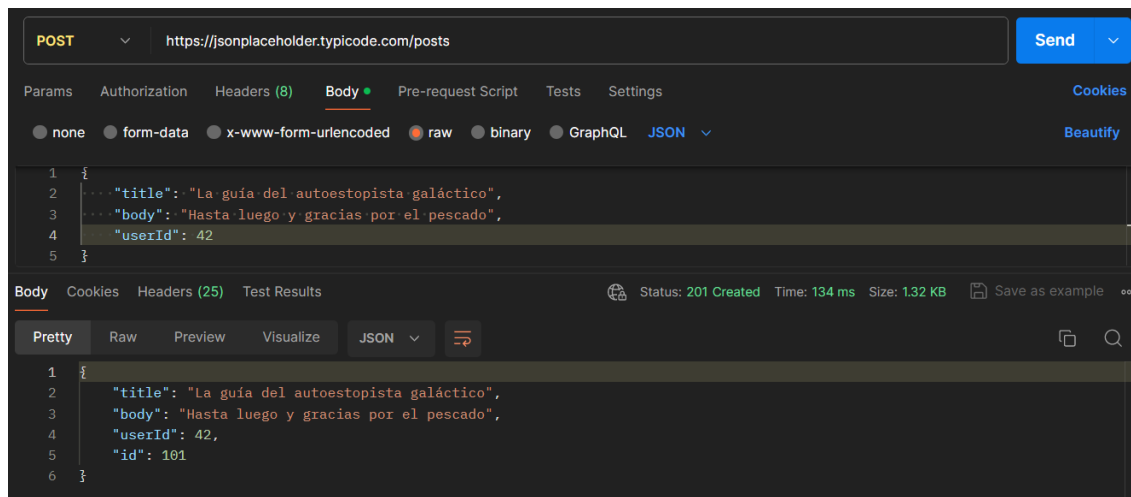
## POST

Las peticiones con POST son sólo para crear recursos nuevos. Cada llamada con POST debería producir un nuevo recurso.

Normalmente, la acción POST se dirige a un recurso que representa una colección, para indicar que el nuevo recurso debe agregarse a dicha colección, por ejemplo POST /cursos para agregar un nuevo recurso a la colección cursos.

Si queremos crear un nuevo post, podríamos tener una URI /posts. Lo que es importante en estos casos, es recordar que la URI no debe decir qué acción estamos ejecutando, no debe indicar de /posts/create etc. El verbo PUT dice qué haremos, y la URI, sobre qué recurso se harán las modificaciones.

Ejemplo de uso de POST:

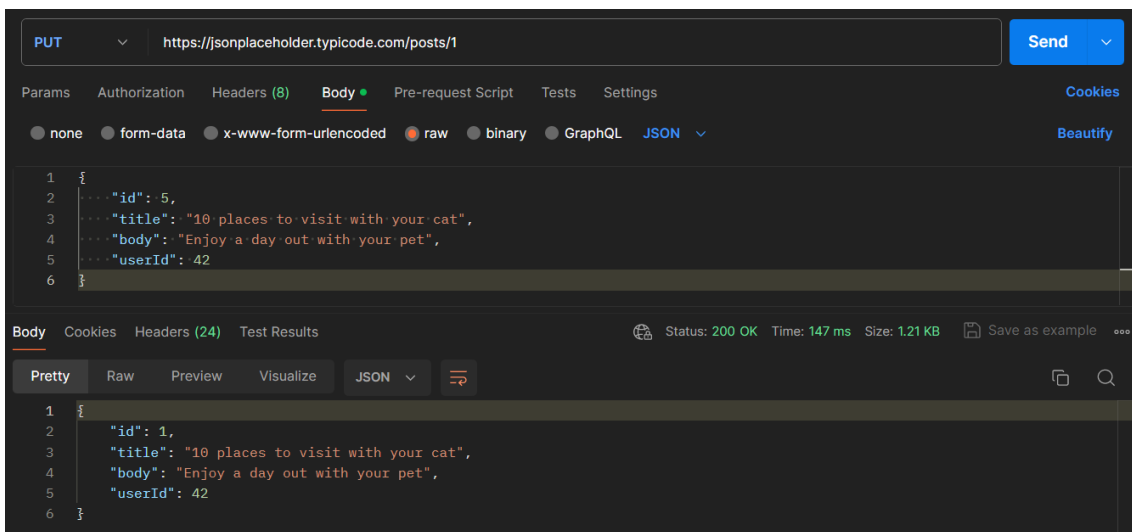


## PUT y PATCH

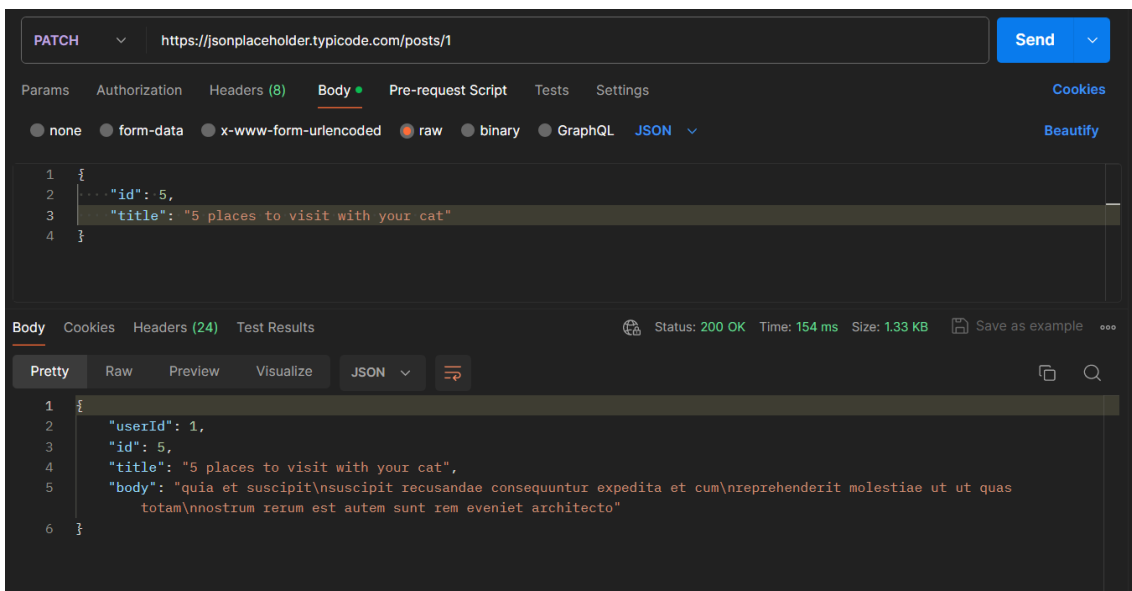
Los verbos PUT/PATCH son muy similares ya que ambos se usan para modificar un recurso existente. PUT se diferencia de PATCH, en que el primero indica que vamos a sustituir por completo un recurso, mientras que PATCH habla de actualizar algunos elementos del recurso mismo, sin sustituirlo por completo.

En la práctica, ambos verbos se usan para actualizar un recurso, sin importar si lo sustituimos parcial o totalmente.

Ejemplo de uso de PUT, sustituyendo un recurso:



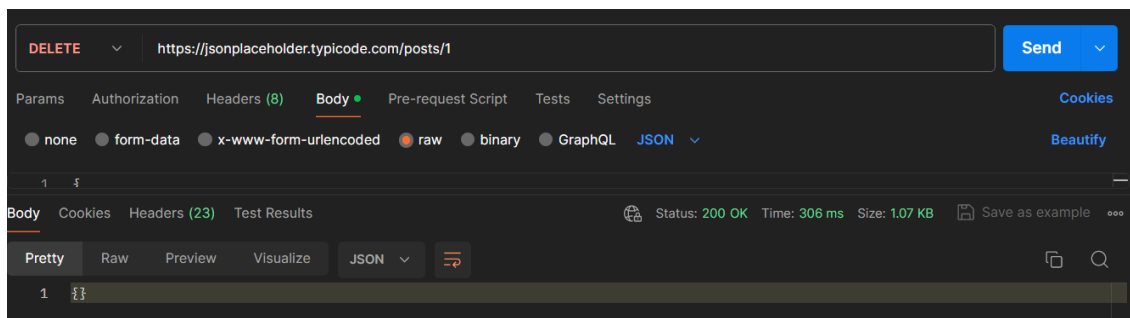
Ejemplo de uso de PATCH, actualizando un recurso de forma parcial:



## DELETE

DELETE es el verbo que usamos para eliminar registros, puede usarse para eliminar un recurso individual o para eliminar una colección completa.

Ejemplo de eliminación de un recurso:



## Fuentes

Agradecimientos a:

- Guías Devcamp
- 
- <https://blog.hubspot.es/website/que-es-api-rest>



# ¿Qué es Postman?

Postman es una plataforma API para construir y utilizar APIs. Se trata de una aplicación que dispone de herramientas nativas para diversos sistemas operativos como Windows, Mac y Linux.

Cuenta con una versión libre de pago y con tres planes (básico, profesional y empresarial) que pueden consultarse en su web oficial <https://www.postman.com/>.

## ¿Para qué sirve Postman?

- Testear colecciones o catálogos de APIs tanto para Frontend como para Backend.
- Organizar en carpetas, funcionalidades y módulos los servicios web.
- Permite gestionar el ciclo de vida (conceptualización y definición, desarrollo, monitoreo y mantenimiento) de nuestra **API**.
- Generar documentación de nuestras APIs.
- Trabajar con entornos (calidad, desarrollo, producción) y de este modo es posible compartir a través de un entorno **cloud** la información con el resto del equipo involucrado en el desarrollo.

## Métodos más utilizados

Los métodos o verbos más utilizados para tomar acción ante nuestras peticiones son los que hemos visto en el apartado anterior:

- **GET**: Obtener información
- **POST**: Agregar información
- **PUT**: Reemplazar la información
- **PATCH**: Actualizar alguna información
- **DELETE**: Borrar información

## Posibles errores

Si la respuesta dada se encuentra en el rango de "200" significa que la petición ha salido sin problemas. El rango 400 hace referencia a errores con el cliente, y los 500 tienen que ver con fallos en el servidor.

- 200 - OK
- 201 - Created
- 204 - No Content
- 400 - Bad Request
- 401 - Unauthorized
- 403 - Forbidden
- 404 - Not Found
- 500 - Internal Server Error

En la siguiente tabla que he tomado prestada de <https://blog.hubspot.es/website/que-es-api-rest> se ofrecen ejemplos de errores y el porqué de los mismos:

VERBO HTTP	URL DEL RECURSO	ACCIÓN	HTTP STATUS
GET	/libros	Obtener una lista de libros	200 - OK

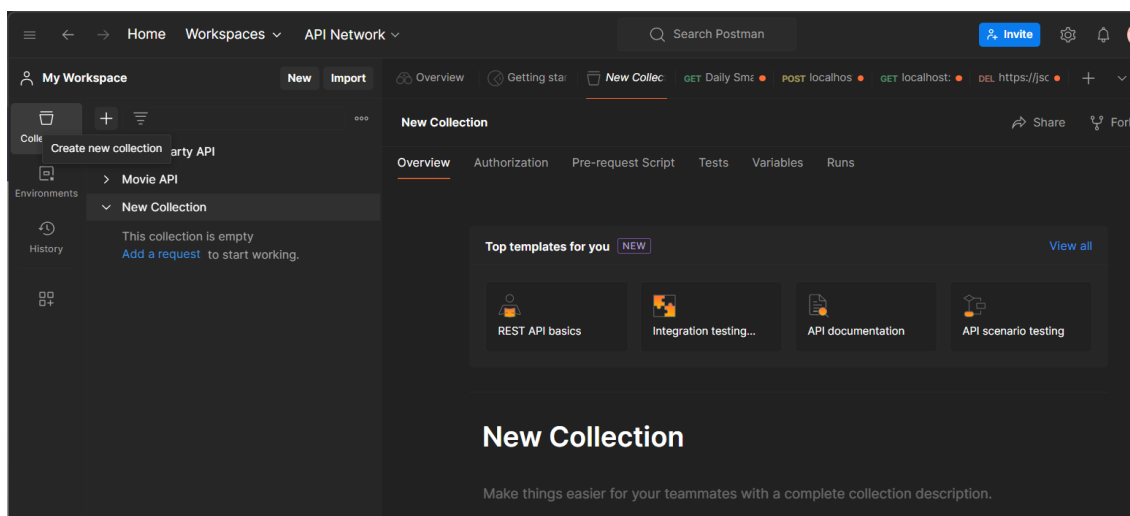
			204 - Not Content: cuando no hay libros a listar
GET	/libros/{id-libro}	Obtener detalle de un libro	200 - OK 404 - Not Found: cuando no existe el libro buscado
POST	/libros	Crear un recurso nuevo libro	201 - Created
PUT	/libros/{id-libro}	Modificar un recurso libro completamente	200 - OK 400 - Bad request: cuando algún parámetro enviado no cumple con las reglas
PATCH	/libros/{id-libro}	Modificar un recurso libro parcialmente	200 - OK 201 - Created: si el recurso a modificar no existe se crea en automático 400 - Bad request: cuando algún parámetro enviado no cumple con las reglas
DELETE	/libros/{id-libro}	Eliminar un recurso libro	201 - Not Content: es el status standard a regresar en un

## Instalación

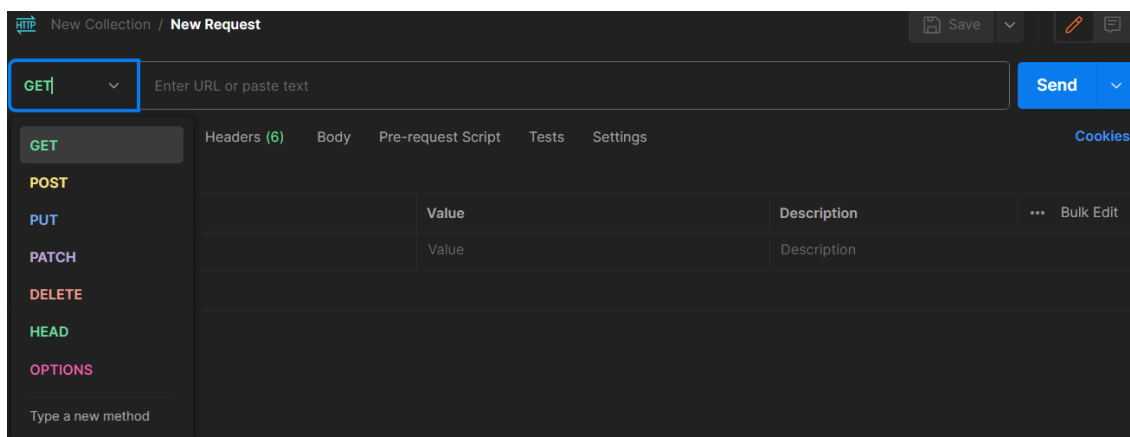
La descarga e instalación de la aplicación dependerá de tu sistema operativo, por lo que la mejor opción será que accedas al enlace <https://www.postman.com/downloads/> y sigas las indicaciones que correspondan en tu caso.

## Primeros pasos

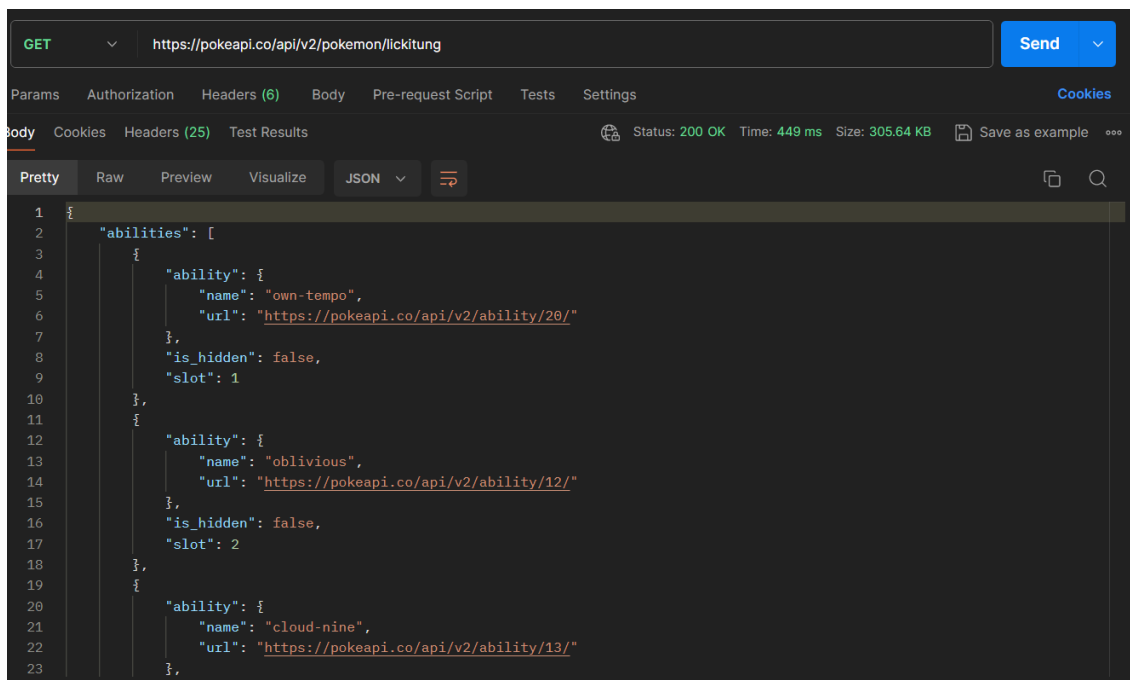
Una vez instalada la aplicación, podemos empezar creando una nueva colección. Dentro de ésta, podemos acceder a Add a request para empezar a utilizar el método.



La lista de los verbos API a utilizar se encuentra en el siguiente apartado.



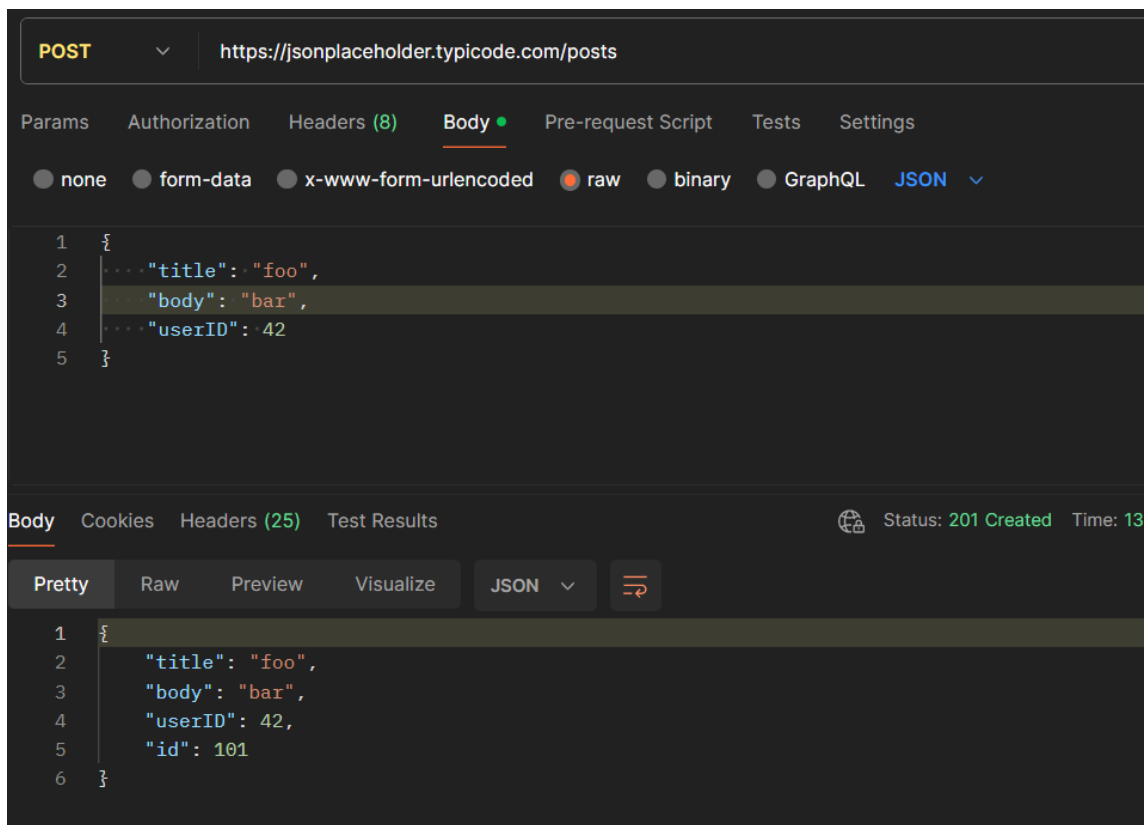
Debemos seleccionar el método que queramos, copiar la URL correspondiente a la API, y pinchar en Send si no tenemos más información que añadir. Los resultados nos aparecerán en el apartado body de la parte inferior:



```
1 {
2   "abilities": [
3     {
4       "ability": {
5         "name": "own-tempo",
6         "url": "https://pokeapi.co/api/v2/ability/20/"
7       },
8       "is_hidden": false,
9       "slot": 1
10    },
11    {
12      "ability": {
13        "name": "oblivious",
14        "url": "https://pokeapi.co/api/v2/ability/12/"
15      },
16      "is_hidden": false,
17      "slot": 2
18    },
19    {
20      "ability": {
21        "name": "cloud-nine",
22        "url": "https://pokeapi.co/api/v2/ability/13/"
23      },
24    }
25  ]
26 }
```

Si debemos introducir código, por ejemplo porque queremos añadir un nuevo recurso mediante el verbo POST, deberemos acceder al apartado Body de la parte superior, seleccionar raw y JSON.

A continuación introduciremos la info correspondiente y entonces pulsaremos Send.



## Ventajas de Postman

Una de las ventajas del uso de Postman es que cuenta con una comunidad grande de usuarios. Además, es posible colaborar con otros miembros de un equipo.

Su interfaz es sencilla e intuitiva, y cuenta con una extensión para el navegador Google Chrome.

Es posible integrarlo con otras herramientas, y es posible agregar scripts de JavaScript para añadir validaciones, automatizar o configurar pruebas.

## Alternativa: SoapUI

SoapUI es una herramienta que ha sido desarrollada en Java y es mayormente utilizada para llevar a cabo pruebas de sistemas cuya arquitectura sea REST o SOAP. Esta soporta múltiples protocolos como JMS, JDBC, HTTP, REST, SOAP.

Al igual que Postman, cuenta con una versión libre y otra de pago. Posee una gran comunidad de usuarios y su curva de aprendizaje es de nivel medio.

A diferencia de Postman, con SoapUI trabajamos en base a proyectos, los cuales pueden ser independientes o compuestos.

## Fuentes

Agradecimientos a:

- Guías Devcamp
- <https://www.postman.com/product/what-is-postman>
- <https://openwebinars.net/blog/que-es-postman/&#x20;>
- <https://formadoresit.es/que-es-postman-cuales-son-sus-principales-ventajas/>

## ¿Es MongoDB una base de datos SQL o NoSQL?

### ¿Que son las bases de datos SQL?

Una base de datos SQL es una base de datos relacional, escrita en el lenguaje de consulta estructurado SQL, que es el acrónimo de Structured Query Language.

Las bases de datos SQL tiene una estructura de tabla (similar a Excel) que consta de columnas y filas.

Tienen herramientas para evitar la duplicidad de registros, y al llevar mucho tiempo en el mercado tienen mayor soporte.

Además, dispone de un sistema estándar (SQL) para las operaciones con la base de datos, como inserción, actualización o consultas.

### ¿Qué son las bases de datos NoSQL?

Una base de datos No SQL es una base de datos no relacional que no cuenta con un identificador que relacione un conjunto de datos con otro. En las bases de datos No SQL la información es organizada generalmente como documentos y no requieren que los datos estén estructurados para poder manipularlos.

Las bases de datos NoSQL son más versátiles y las opciones open source no requieren pago de licencia.

Además, permiten guardar datos de cualquier tipo sin requerir una verificación, y realizan consultas utilizando lenguaje JSON (JavaScript Object Notation).

### Diferencias entre SQL y NoSQL

Las bases de datos NoSQL ofrecen mayor rendimiento que las SQL, ya que necesitan menos recursos de hardware.

Las bases de datos relacionales SQL son más fiables que las NoSQL, porque si un proceso tiene algún error no se lleva a cabo.

En cuanto a la consistencia, o la capacidad de garantizar la integridad de los datos, es pobre en las bases de datos NoSQL. Las SQL son sin embargo muy consistentes.

Con respecto al volumen de almacenamiento, las bases de datos SQL son indicadas para una cantidad de datos no muy grande. Las NoSQL pueden manejar grandes volúmenes de datos.

Por otro lado, las NoSQL son escalables porque se puede aumentar su capacidad fácilmente. Las SQL pueden serlo pero con un costo económico más elevado. Las bases de datos No SQL utilizan un escalado horizontal (aumentar el número de servidores) mientras que las SQL utilizan un escalado vertical (aumentar los recursos de un servidor).

### ¿Cuándo utilizar una u otra opción?

La decisión de utilizar bases de datos SQL o NoSQL dependerá de los requisitos de la aplicación. Como norma general:

- Si tienes datos que dependen en gran medida de las relaciones, entonces es probable que la mejor opción sea utilizar una base de datos relacional SQL.
- Si tienes un conjunto de datos no estructurados que necesitan flexibilidad, entonces la mejor opción será optar por una base de datos NoSQL.

### Ejemplos de bases de datos SQL

Estas son algunas de las bases de datos SQL más conocidas:

- MySQL: principal gestor de bases de datos en SQL. Es una herramienta open source en código abierto.
- Oracle: una de las herramientas más potentes del mercado, destinada principalmente a medianas o grandes empresas.
- Microsoft Access: herramienta de Microsoft para la creación de bases de datos SQL
- SQLite: es una biblioteca de C que provee una base de datos ligera basada en disco que no requiere un proceso de servidor separado y permite acceder a la base de datos usando una variación no estándar del lenguaje de consulta SQL

### Ejemplos de bases de datos NoSQL

Algunas de las bases de datos NoSQL más reconocidas son las siguientes:

- ApacheCassandra: es una base de datos de tipo clave-valor. Está diseñada para almacenar grandes cantidades de datos y realizar distribuciones a través de varios nodos.
- CouchDB: su fuerte es la accesibilidad y compatibilidad web con diferentes tipos de dispositivos.
- Redis: su diseño principal está basado en el almacenamiento de tablas de hashes aunque no es restrictiva sólo hacia este modelo. También tiene la posibilidad de ser utilizada como una BBDD persistente.
- MongoDB: importante gestor de datos que almacena documentos en un formato parecido al JSON a alta velocidad. Es ideal para proyectos en los que se requiera alto nivel de escalabilidad. Le dedicaremos el siguiente apartado.

## MongoDB

### ¿Qué es MongoDB?

MongoDB es una base de datos NoSQL que ofrece una gran escalabilidad y flexibilidad, y un modelo de consultas e indexación avanzado.

Esta base de datos almacena datos en documentos flexibles similares a JSON, por lo que los campos pueden variar entre documentos y la estructura de datos puede cambiarse con el tiempo.

Es una base de datos distribuida en su núcleo, por lo que la alta disponibilidad, la escalabilidad horizontal y la distribución geográfica están integradas y son fáciles de usar.

MongoDB es de uso gratuito. Las versiones lanzadas antes del 16 de octubre de 2018 se publican bajo licencia AGPL. Todas las versiones posteriores al 16 de octubre de 2018, incluidos los parches lanzados para versiones anteriores, se publican bajo Licencia pública del lado del servidor (SSPL) v1.

### Instalación

La descarga e instalación de MongoDB dependerá del sistema operativo que utilices, por lo que puedes acceder al siguiente link y seguir las indicaciones que apliquen para tu caso:

<https://www.mongodb.com/docs/manual/installation/>

Además, deberás escoger entre instalar MongoDB Community edition y MongoDB Enterprise.

¿En qué se diferencian las licencias MongoDB Community y Enterprise? MongoDB Community es la edición gratuita de MongoDB, mientras que MongoDB Enterprise solo está disponible con la suscripción MongoDB Enterprise Advanced e incluye un completo soporte para sus despliegues en MongoDB.

### MongoDB Shell, mongosh

Mongo shell es una interfaz JavaScript interactiva para MongoDB. Puedes utilizar mongo shell para consultar y actualizar datos, así como para realizar operaciones administrativas.

MongoDB Shell (mongosh) no se instala con MongoDB Server, por lo tanto deberás seguir las instrucciones el siguiente enlace para descargar e instalar mongosh por separado:

```
{% embed url="https://www.mongodb.com/docs/mongodb-shell/install/" %}
```

### Primeros pasos con MongoDB

Para acceder a MongoDB, primero debemos tener el Daemon de MongoDB ejecutándose.

Para ello, es necesario que en nuestra línea de comando nos movamos al directorio que contenga el archivo "mongod".

```
cd "C:\Program Files\MongoDB\Server\7.0\bin"
```

Este es el directorio en mi caso, pero dependerá de la ubicación de la instalación.

A continuación deberemos introducir el comando:

```
mongod
```

Ahora abrir una nueva pestaña de cmd sin cerrar la anterior, e introducir:

```
mongosh
```

Este debería ser el aspecto de tu terminal:



```

Current Mongosh Log ID: 65faf2ce88d2662449f61447
Connecting to:      mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000&appName=mongosh+2
.1.5
Using MongoDB:      7.0.6
Using Mongosh:      2.1.5

For mongosh info see: https://docs.mongodb.com/mongodb-shell/

-----
  The server generated these startup warnings when booting
  2024-03-10T16:19:04.483+01:00: Access control is not enabled for the database. Read and write access to data and conf
  igation is unrestricted
  -----

test> _

```

Ahora que lo tenemos, veamos algunas operaciones sencillas y su output.

Dado que el código introducido para determinadas acciones puede ser largo, es común escribir el código en un editor de texto para que sea más cómodo, y a continuación pegarlo en el shell.

Para ver las bases de datos que tenemos en el sistema:

```
show dbs
```

```

test> show dbs
admin          180.00 KiB
config         72.00 KiB
local          40.00 KiB
mongoCourse    72.00 KiB
test>

```

Crear nueva base de datos o acceder a una existente:

```
use nombreBaseDatos
```

```

test> use movies
switched to db movies
movies>

```

Crear nuevo usuario:

```

db.createUser({
  user: 'leslieknope',
  pwd: 'waffles',
  customData: { startDate: new Date() },
  roles: [
    { role: 'clusterAdmin', db: 'admin' },
    { role: 'readAnyDatabase', db: 'admin' },
    'readWrite'
  ]
})

```

```

movies> db.createUser({
...   user: 'leslieknope',
...   pwd: 'waffles',
...   customData: { startDate: new Date() },
...   roles: [
...     { role: 'clusterAdmin', db: 'admin' },
...     { role: 'readAnyDatabase', db: 'admin' },
...     'readWrite'
...   ]
... })
{ ok: 1 }
movies>

```

Ver todos los usuarios de la base de datos:

```
db.getUsers()
```

Eliminar un usuario:

```
db.dropUser('leslieknope')
```

Crear una colección (es la forma de organizar la base de datos. Las colecciones almacenan documentos, que son los elementos a introducir en la base de datos).

```
db.createCollection('movies')
```

```

movies> db.createCollection('movies')
{ ok: 1 }

```

Ver colecciones:

```
show collections
```

Insertar un documento en la base de datos:

```

db.movies.insertOne({
  "title": "The princess bride",
  "year": 1987,
  "reviews": [
    {1: "Magical movie"},
    {2: "So boring"}
  ]
})

```

La función insertOne() espera un objeto.

Recuerda que al tratarse de una BBDD NoSQL cada elemento puede tener la estructura que tú elijas. Ejemplo de otro documento con una estructura diferente:

```

db.movies.insertOne({
  "title": "The Bob's Burgers movie",
  "year": 2022,
  "directors": [
    {"name": "Loren Bouchard"},
    {"name": "Bernard Derriman"}
  ],
  "reviews": 99,
  "cast": [
    {"Tina": "Dan Mintz"},

```

```
    {"Gene": "Eugene Mirman"},  
    {"Louise": "Kristen Schaal"}  
  ]  
})
```

```
movies> db.books.insertOne(  
...   "title": "The princess bride",  
...   "year": 1987,  
...   "reviews": [  
...     {1: "Magical movie"},  
...     {2: "So boring"}  
...   ]  
... })  
{  
  acknowledged: true,  
  insertedId: ObjectId('65faf89a88d2662449f61448')  
}  
movies>
```

```
movies> db.books.insertOne(  
...   "title": "The Bob's Burgers movie",  
...   "year": 2022,  
...   "directors": [  
...     {"name": "Loren Bouchard"},  
...     {"name": "Bernard Derriman"}  
...   ],  
...   "reviews": 99,  
...   "cast": [  
...     {"Tina": "Dan Mintz"},  
...     {"Gene": "Eugene Mirman"},  
...     {"Louise": "Kristen Schaal"}  
...   ]  
... })  
{  
  acknowledged: true,  
  insertedId: ObjectId('65fafcce88d2662449f61449')  
}  
movies>
```

Sin embargo, Intenta ser lo más consistente posible para que tu base de datos sea utilizable.

Asimismo, si cometemos algún error de sintaxis MongoDB nos lo notificará.

```
Uncaught:  
SyntaxError: Missing semicolon. (1:13)
```

Insertar varios documentos:

```
db.movies.insertMany([
  {
    "title": "Life of Brian",
    "year": 1979,
  },
  {
    "title": "Mapa de los sonidos de Tokio",
    "director": "Isabel Coixet",
  },
  {
    "title": "Everything Everywhere All at Once",
  }
])
```

```
movies> db.movies.insertMany([
...   {
...     "title": "Life of Brian",
...     "year": 1979,
...   },
...   {
...     "title": "Mapa de los sonidos de Tokio",
...     "director": "Isabel Coixet",
...   },
...   {
...     "title": "Everything Everywhere All at Once",
...   }
... ])
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId('65faff2888d2662449f6144a'),
    '1': ObjectId('65faff2888d2662449f6144b'),
    '2': ObjectId('65faff2888d2662449f6144c')
  }
}
movies>
```

Consultar todos los documentos:

```
db.movies.find()
```

```

movies> db.movies.find()
[
  {
    _id: ObjectId('65faff2888d2662449f6144a'),
    title: 'Life of Brian',
    year: 1979
  },
  {
    _id: ObjectId('65faff2888d2662449f6144b'),
    title: 'Mapa de los sonidos de Tokio',
    director: 'Isabel Coixet'
  },
  {
    _id: ObjectId('65faff2888d2662449f6144c'),
    title: 'Everything Everywhere All at Once'
  },
  {
    _id: ObjectId('65faff8888d2662449f6144d'),
    title: "The Bob's Burgers movie",
    year: 2022,
    directors: [ { name: 'Loren Bouchard' }, { name: 'Bernard Derriman' } ],
    reviews: 99,
    cast: [
      { Tina: 'Dan Mintz' },
      { Gene: 'Eugene Mirman' },
      { Louise: 'Kristen Schaal' }
    ]
  },
  {
    _id: ObjectId('65faff9388d2662449f6144e'),
    title: 'The princess bride',
    year: 1987,
    reviews: [ { '1': 'Magical movie' }, { '2': 'So boring' } ]
  }
]

```

Consultar documentos específicos:

```
db.movies.find( {title: "The princess bride"} )
```

```
db.movies.find( {year: 2022} )
```

Si hay varias coincidencias, MongoDB devolverá todas. Existe una función para que solo nos devuelva un resultado:

```
db.movies.findOne( { year: 2022 } )
```

Si MongoDB no encuentra ningún resultado, no devolverá nada, tampoco un mensaje de error.

En relación a las búsquedas, debemos mencionar de que se considera una mala práctica realizar una búsqueda que nos devuelva todos los atributos de los elementos, ya que normalmente querremos escoger qué atributos queremos ver.

Para esto existen las proyecciones, que son limitaciones que ponemos a nuestra búsqueda para que nos devuelva la información que escojamos. La sintaxis consiste en añadir un segundo objeto a la función find, en la que con 0s y 1s escogeremos qué atributos queremos.

```

db.movies.find(
  {
    title: "The princess bride"
  }
)

```

```

    },
    {
      _id: 0,
      title: 1,
      year: 1
    }
  )

```

El atributo `_id` se nos devolverá por defecto, así que si no lo queremos deberemos explícitamente asignarle el número 0.

```

movies> db.movies.find(
...   {
...     title: "The princess bride"
...   },
...   {
...     _id: 0,
...     title: 1,
...     year: 1
...   }
... )
[ { title: 'The princess bride', year: 1987 } ]
movies>

```

Eliminar un documento:

```
db.movies.deleteOne({title: "Mapa de los sonidos de Tokio"})
```

```

movies> db.movies.deleteOne({title: "Mapa de los sonidos de Tokio"})
{ acknowledged: true, deletedCount: 1 }
movies>

```

Podemos utilizar la función `deleteMany()` para eliminar todos los documentos que cumplan con el criterio establecido. Por ejemplo, para borrar todas las películas del 2022:

```
db.movies.deleteMany(year: 2022)
```

### Otros conceptos a explorar

A continuación mencionaremos otras funcionalidades de MongoDB para que puedas investigar y probar en tu base de datos particular:

- Función `$slice`
- Búsqueda de elementos anidados
- Función `$exists`
- Uso de regular expressions

## Fuentes

Agradecimientos a:

- <https://www.mongodb.com/es/nosql-explained>
- <https://ayudaleyprotecciondatos.es/bases-de-datos/sql/>
- [https://www.mongodb.com/docs/manual/?\\_ga=2.128054348.1193247927.1710508974-1202821968.1710082371](https://www.mongodb.com/docs/manual/?_ga=2.128054348.1193247927.1710508974-1202821968.1710082371)
- <https://www.purestorage.com/es/knowledge/what-is-mongodb.html#:~:text=MongoDB%20es%20muy%20flexible%20y,bases%20de%20datos%20relacionales%20tradicionales>
- <https://ilimit.com/blog/base-de-datos-sql-nosql/#que-es-una-base-de-datos-sql>
- <https://www.grapheverywhere.com/bases-de-datos-nosql-marcas-tipos-ventajas/>