Python TOTAL

clases

> Python es un lenguaje de Programación Orientado a Objetos (POO). Como tal, utiliza y manipula objetos, a través de sus métodos y propiedades. Las clases son las herramientas que nos permiten crear objetos, que "empaquetan" datos y funcionalidad juntos.

Podemos pensar a las clases como el "plano" o "plantilla" a partir del cual podemos crear objetos individuales, con propiedades y métodos asociados:

```
class Objeto:
    ...
    ...

nuevo_objeto = Objeto()
```

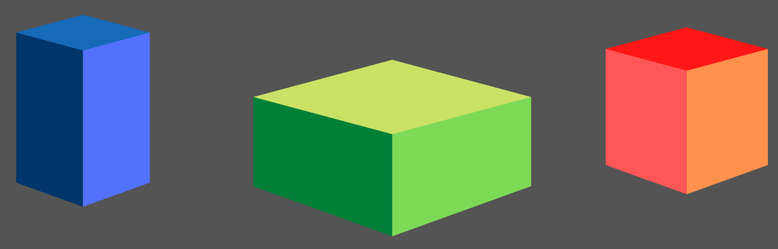
Crear una clase

Creación de un objeto a partir de la clase

este objeto es una *instancia* de la clase

```
class :
```

clase



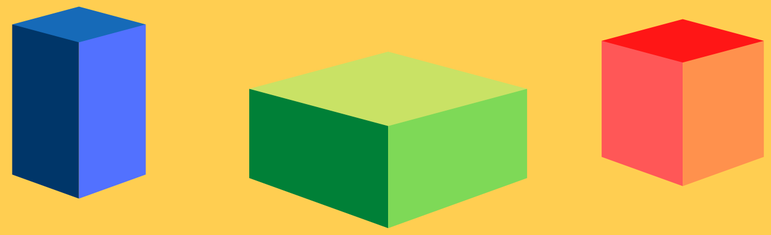
instancias (objetos)

atributos

Python TOTAL

Los atributos son variables que pertenecen a la clase. Existen **atributos de clase** (compartidos por todas las instancias de la clase), y **de instancia** (que son distintos en cada instancia de la clase).

`class` **Cubo**:



clase

instancias (objetos)

Cada objeto creado a partir de la clase puede compartir determinadas características

atributos de clase

(por ejemplo: cantidad de caras)

Cada objeto creado a partir de la clase puede contener características específicas

atributos de instancia

(por ejemplo: color, altura, ancho, largo...)

atributo de clase

```
class Cubo:
    caras = 6
    def __init__(self, color):
        self.color = color
```

atributo de instancia

los parámetros de la función __init__ se entregan a los atributos de instancia

Todas las clases tienen una función que se ejecuta al *instanciarla*, llamada **__init__()**, y que se utiliza para asignar valores a las propiedades del objeto que está siendo creado.
self: representa a la *instancia* del objeto que se va a crear

métodos

Python TOTAL

> Los objetos creados a partir de clases también contienen métodos. Dicho de otra manera, los métodos son funciones que pertenecen al objeto.

```
class Persona:

    especie = "humano"

    def __init__(self, nombre, edad):
        self.nombre = nombre
        self.edad = edad

    def saludar(self):
        print(f'Hola, mi nombre es {self.nombre}')

    def cumplir_anios(self, estado_humor):
        print(f'Cumplir {self.edad + 1} años me pone {estado_humor}')
```

```
juan = Persona("Juan", 37)
juan.saludar()
juan.cumplir_anios("feliz")

>> Hola, mi nombre es Juan
>> Cumplir 38 años me pone feliz
```

Cada vez que un atributo del objeto sea invocado (por ejemplo, en una función), debe incluirse *self*, que refiere a la *instancia* en cuestión, indicando la pertenencia de este atributo.

Python TOTAL

tipos de métodos

Los métodos estáticos y de clase anteponen un decorador específico, que indica a Python el tipo de método que se estará definiendo

	métodos de instancia	`@classmethod` métodos de clase	`@staticmethod` métodos estáticos
acceso a métodos y atributos de la clase	sí	sí	no
requiere una instancia	sí	no	no
acceso a métodos y atributos de la instancia	sí	no	no

Así como los métodos de instancia requieren del parámetro *self* para acceder a dicha instancia, los métodos de clase requieren del parámetro *cls* para acceder a los atributos de clase. Los métodos estáticos, dado que no pueden acceder a la instancia ni a la clase, no indican un parámetro semejante.

herencia

> La herencia es el proceso mediante el cual una clase puede tomar métodos y atributos de una clase superior, evitando repetición del código cuando varias clases tienen atributos o métodos en común.

Es posible crear una clase "*hija*" con tan solo pasar como parámetro la clase de la que queremos heredar:

```
class Personaje:

    def __init__(self, nombre, herramienta):
        self.nombre = nombre
        self.arma = arma

class Mago(Personaje):
    pass

hechicero = Mago("Merlín", "caldero")
```

Una clase "*hija*" puede sobreescribir los métodos o atributos, así como definir nuevos, que sean específicos para esta clase.

herencia extendida

> Las clases "hijas" que heredan de las clases superiores, pueden crear nuevos métodos o sobrescribir los de la clase "padre". Asimismo, una clase "hija" puede heredar de una o más clases, y a su vez transmitir herencia a clases "nietas".

Si varias superclases tienen los mismos atributos o métodos, la subclase sólo podrá heredar de una de ellas. En estos casos Python dará prioridad a la clase que se encuentre más a la izquierda.

Del mismo modo, si un mismo método se hereda por parte de la clase "padre", e "hija", la clase "nieta" tendrá preferencia por aquella más próxima ascendente (siguiendo nuestro esquema, la tomará de la clase "hija").

orden de búsqueda de un método:

un método dado se buscará primero en la propia clase, y de no hallarse, se explorará las superiores

...podemos continuar ampliando el diagrama de herencia con la misma lógica

Clase.__mro__

devuelve el orden de resolución de métodos

super().__init__(arg1, arg2,...)

hereda atributos de las superclases de manera compacta

Python TOTAL

polimorfismo

Python TOTAL

> El polimorfismo es el pilar de la POO mediante el cual un mismo método puede comportarse de diferentes maneras según el objeto sobre el cual esté actuando, en función de cómo dicho método ha sido creado para la clase en particular.

El método len( ) funciona en distintos tipos de objetos: listas, tuplas, strings, entre otros. Esto se debe a que para Python, lo importante no son los tipos de objetos, sino lo que pueden hacer: sus métodos.

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

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

hachiko = Perro()
garfield = Gato()

for animal in [hachiko, garfield]:
    animal.hablar()

>> Guau!
>> Miau!
```

métodos especiales

Python TOTAL

> Puedes encontrarlos con el nombre de métodos mágicos o *dunder methods* (del inglés: *dunder* = *double underscore*, o doble guión bajo). Pueden ayudarnos a sobrescribir métodos incorporados de Python sobre nuestras clases para controlar el resultado devuelto.

```
class Libro:
    def __init__(self, autor, titulo, cant_paginas):
        self.autor = autor
        self.titulo = titulo
        self.cant_paginas = cant_paginas

    def __str__(self):
        return f'Título: "{self.titulo}", escrito por {self.autor}'

    def __len__(self):
        return self.cant_paginas

libro1 = Libro("Stephen King", "It", 1032)
print(str(libro1))
print(len(libro1))

>> Título: "It", escrito por Stephen King
>> 1032
```

Práctica Clases 1

Crea una clase llamada `Personaje` y a continuación, crea un objeto a partir de ella, por ejemplo: `harry_potter`

Práctica Clases 2

Crea una clase llamada `Dinosaurio`, y tres instancias a partir de ella: `velociraptor`, `tiranosaurio_rex` y `braquiosaurio`.

Práctica Clases 3

Crea una clase llamada `PlataformaStreaming` y crea los siguientes objetos a partir de ella: `netflix`, `hbo_max`, `amazon_prime_video`

Práctica Atributos 1

Crea una clase llamada `Casa`, y asígnale atributos: `color`, `cantidad_pisos`.

Crea una instancia de Casa, llamada `casa_blanca`, de color "blanco" y cantidad de pisos igual a 4.

Práctica Atributos 2

Crea una clase llamada `Cubo`, y asígnale el atributo de clase:

- `caras = 6`

y el atributo de instancia:

- `color`

Crea una instancia `cubo_rojo`, de dicho color.

Práctica Atributos 3

Crea una clase llamada `Personaje`, y asígnale el siguiente atributo de clase:

- `real = False`

Crea una instancia llamada `harry_potter` con los siguientes atributos de instancia:

- `especie = "Humano"`
- `magico = True`
- `edad = 17`

Práctica Métodos 1

Crea una clase `Perro`. Dicho perro debe poder ladrar.

Crea el método `ladrar()` y ejecútalo en una instancia de Perro. Cada vez que ladre, debe mostrarse en pantalla `"Guau!"`.

Práctica Métodos 2

Crea una clase llamada `Mago`, y crea un método llamado `lanzar_hechizo()` (deberá imprimir "***¡Abracadabra!***").

Crea una instancia de Mago en el objeto `merlin`, y haz que lance un hechizo.

Práctica Métodos 3

Crea una instancia de la clase `Alarma`, que tenga un método llamado `postergar(cantidad_minutos)`. El método debe imprimir en pantalla el mensaje **"La alarma ha sido pospuesta `{cantidad_minutos}` minutos"**

Práctica Herencia 1

Crea una **clase** llamada `Persona`, que tenga los siguientes ***atributos de instancia***: `nombre`, `edad`. Crea otra clase, **Alumno**, que **herede** de la primera estos atributos.

Práctica Herencia 2

Crea una **clase** llamada `Mascota`, que tenga los siguientes ***atributos de instancia***: `edad`, `nombre`, `cantidad_patas`. Crea otra clase, **Perro**, que **herede** de la primera sus atributos.

Práctica Herencia 3

Crea una clase llamada `Vehiculo`, que contenga los métodos `acelerar()` y `frenar()` (puedes dejar el código de los métodos en blanco con `pass`). Crea una clase llamada `Automovil` que herede estos métodos de `Vehiculo`.

Ejercicio Cuenta Bancaria

Se debe crear un código que le permita a una persona realizar operaciones en su cuenta bancaria.

Primero vas a crear una clase llamada Persona, y Persona va a tener solo dos atributos: nombre y apellido. Luego, vas a crear una segunda clase llamada Cliente, y Cliente va a heredar de Persona, porque los clientes son personas, por lo que el Cliente va a heredar entonces los atributos de Persona, pero también va a tener atributos propios, como número de cuenta y balance, es decir, el saldo que tiene en su cuenta bancaria.

Pero eso no es todo: Cliente también va a tener tres métodos. El primero va a ser uno de los métodos especiales y es el que permite que podamos imprimir a nuestro cliente. Este método va a permitir que cuando el código pida imprimir Cliente, se muestren todos sus datos, incluyendo el balance de su cuenta. Luego, un método llamado Depositar, que le va a permitir decidir cuánto dinero quiere agregar a su cuenta. Y finalmente, un tercer método llamado Retirar, que le permita decidir cuánto dinero quiere sacar de su cuenta.

Una vez que hayas creado estas dos clases, tienes que crear el código para que tu programa se desarrolle, pidiéndole al usuario que elija si quiere hacer depósitos o retiros. El usuario puede hacer tantas operaciones como quiera hasta que decida salir del programa. Por lo tanto, nuestro código tiene que ir llevando la cuenta de cuánto dinero hay en el balance, y debes procurar, por supuesto, que el cliente nunca pueda retirar más dinero del que posee. Esto no está permitido.

Recuerda que ahora que sabes crear clases y objetos que son estables y que retienen información, no necesitas crear funciones que devuelvan el balance, ya que la instancia de cliente puede saber constantemente cuál es su saldo debido a que puede hacer sus operaciones llamando directamente a este atributo y no a una variable separada.

Para que tu programa funcione, puedes organizar tu código como quieras, hay muchas formas de hacerlo, pero mi recomendación es que básicamente, luego de crear las dos clases que te he mencionado, crees dos funciones una que se encarguen de crear al cliente pidiéndole al usuario toda la información necesaria y devolviendo, a través del return, un objeto cliente ya creado.

La otra función (que puede llamarse inicio, o algo por el estilo), es la función que organiza la ejecución de todo el código: primero llama a la función “crear cliente” y luego se encarga de mantener al usuario en un loop que le pregunte todo el tiempo si quiere depositar, retirar o salir del programa y demostrarle el balance, cada vez que haga una modificación.