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()
```

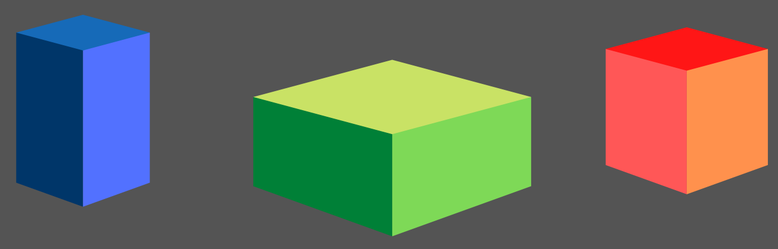
Crear una clase

Creación de un objeto a partir de la clase

este objeto es una *instancia* de la clase

```
class :
```

clase



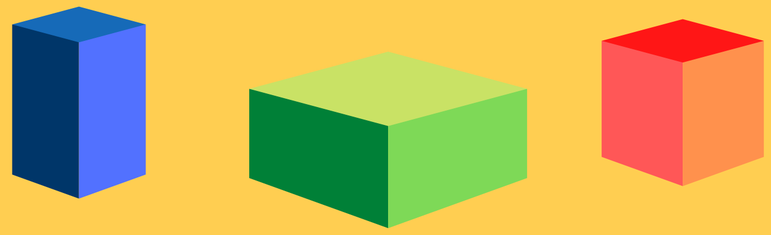
instancias (objetos)

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
```