Programación Orientada a Objetos

Rodrigo Lugones
rodrigolugones@gmail.com

Paradigmas de programación

Filosofía o enfoque fundamental para escribir código y estructurar programas.

Ejemplos:

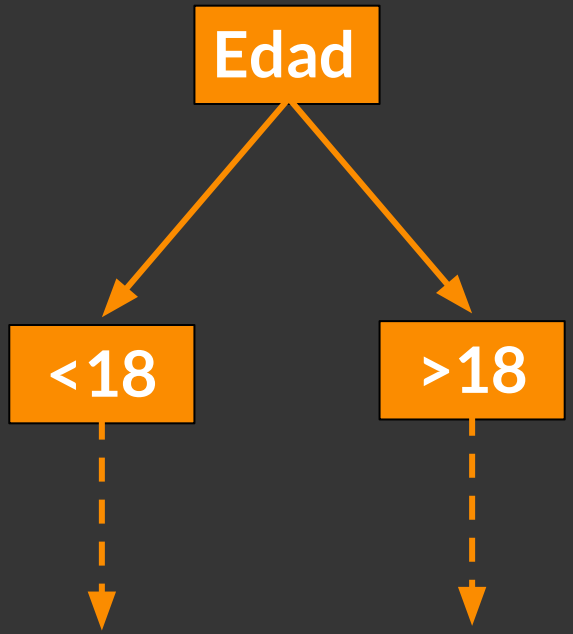
- Imperativa
- Orientada a objetos
- Funcional
- etc

Paradigmas de programación

Influyen en la forma de pensar el código
Facilitan la comunicación
Guían la elección de herramientas

Si bien algunos lenguajes de programación son más adecuados para ciertos paradigmas, pero no restringen.

Imperativo

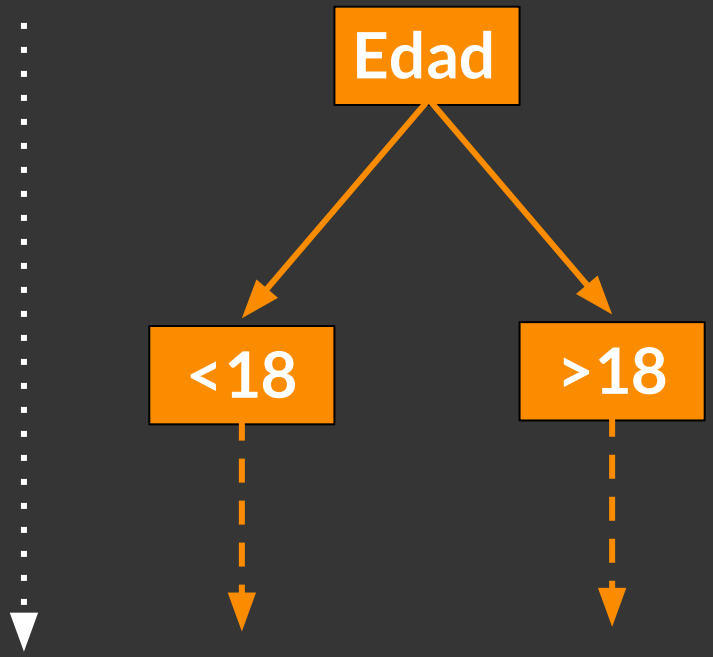


Órdenes a la computadora

Lectura lineal

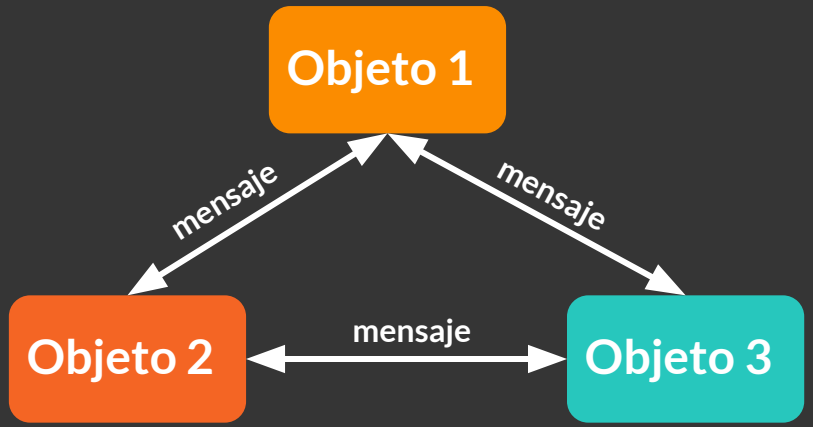
Segmentado en “funciones”

Imperativo



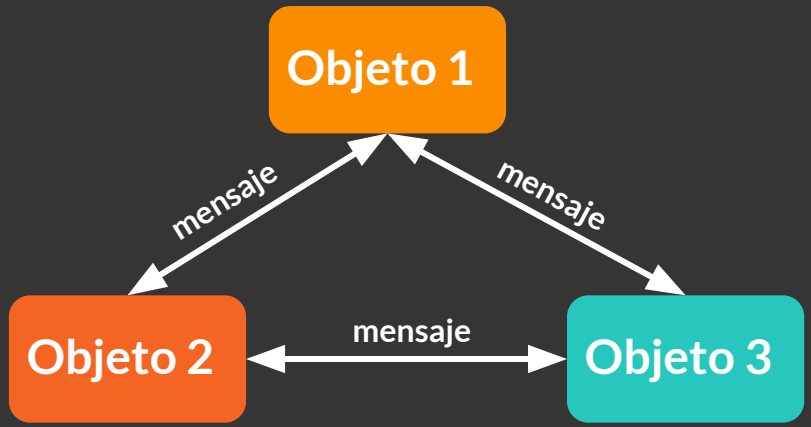

```python
# file: esPrimo.py

numeroString = input('Número menor a 100: ')
numero = int(numeroString)
noEsPrimo = False

for primo in [2, 3, 5, 7]:
  if numero % primo == 0:
    noEsPrimo = True
    break

if noEsPrimo:
  print("El número {0} no es primo".format(numero))
else:
  print("El número {0} es primo".format(numero))
```

Orientado a objetos



Creación de objetos que interactúan entre sí

Orientado a objetos



Estado

Condición interna de un objeto.
Se define por los valores de sus **atributos**.

Comportamiento

Define cómo actúa y cómo interactúa con otros objetos.
Se implementa a través de sus **métodos**.

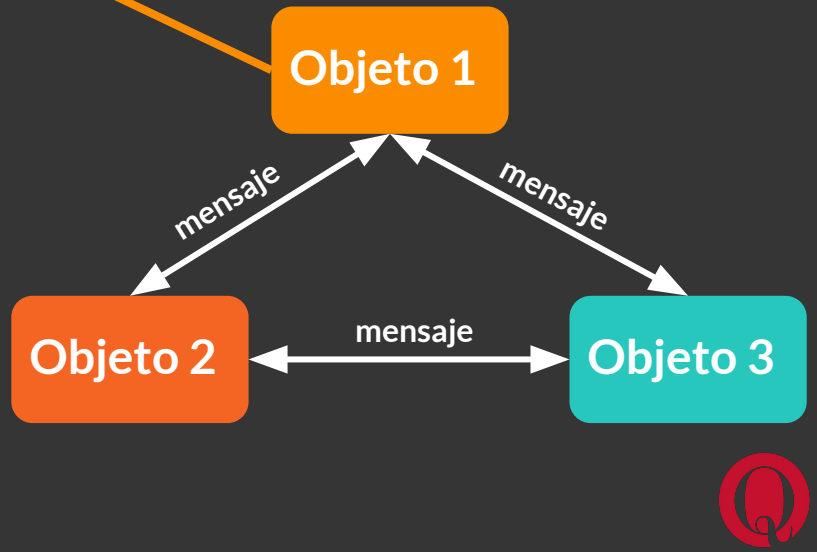
Identidad

Propiedad que distingue a un objeto del resto, incluso si tienen el mismo estado y comportamiento.
Identificador único e inmutable asignado cuando se crea.

Objeto

Atributos: datos

Métodos: procedimientos



El objeto puede, a través de sus métodos, modificarse a sí mismo

Clase: *blueprint* de los objetos

El concepto *abstracto* detrás del objeto *concreto*

Clase: *blueprint* de los objetos

El concepto ***abstracto*** detrás del objeto ***concreto***

humano

Rodrigo Lugones

mamífero

humano

(subclase)

Pilares fundamentales de OOP

Composición

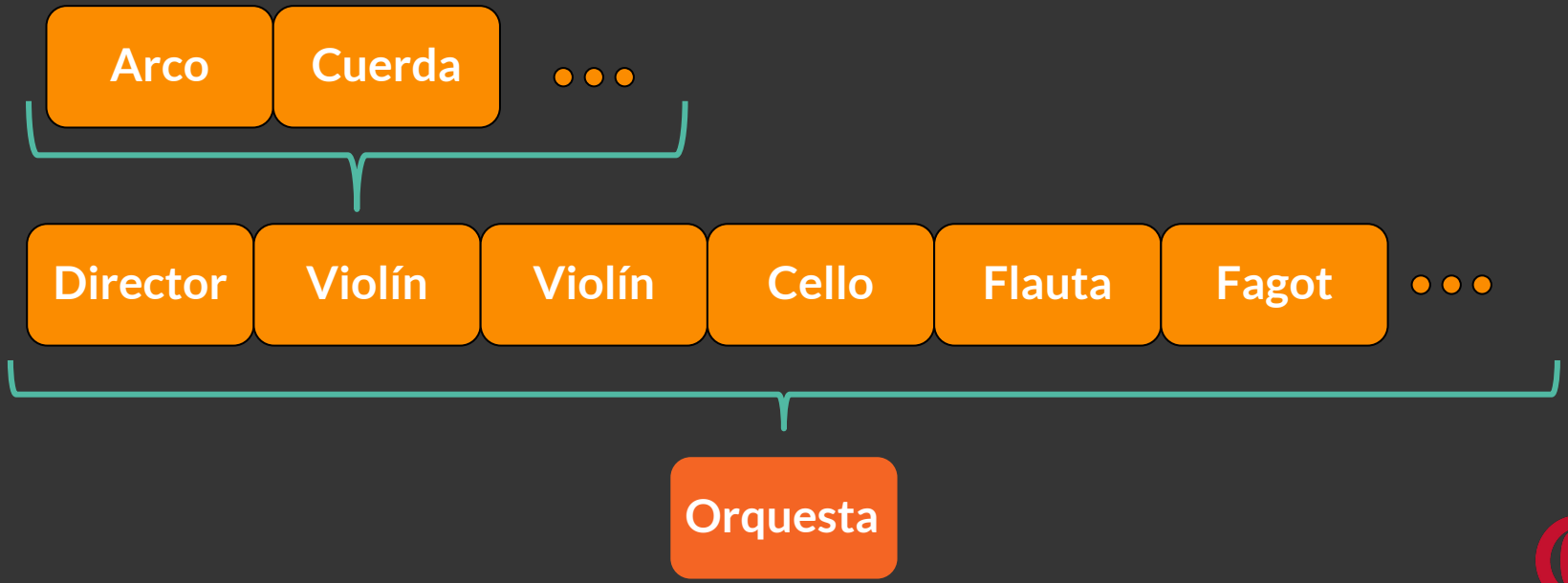
Encapsulamiento

Herencia

Polimorfismo

1) Composición

Un objeto está conformado por otros objetos



2) Encapsulamiento

Ocultar los detalles internos de un objeto y controlar el acceso a sus datos.

Tratamos al objeto como una caja negra.

Permite prohibir el acceso a atributos y/o métodos de una clase.

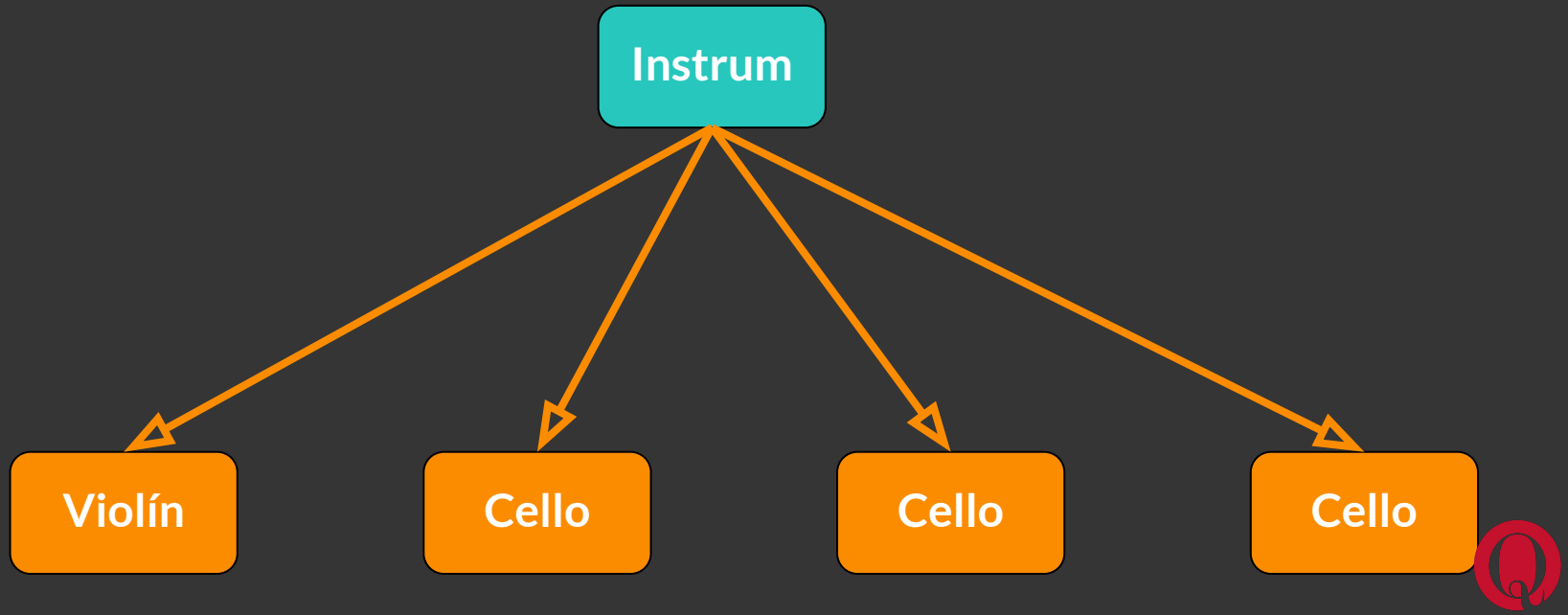
Se puede acceder a los atributos de un objeto (mostrar o modificar) sólo a través de métodos.

En la práctica, elegimos qué métodos y atributos son *privados*.

Ej: Un músico interactúa con el violín a través de las cuerdas y el arco, sin necesidad de conocer los detalles de su construcción.

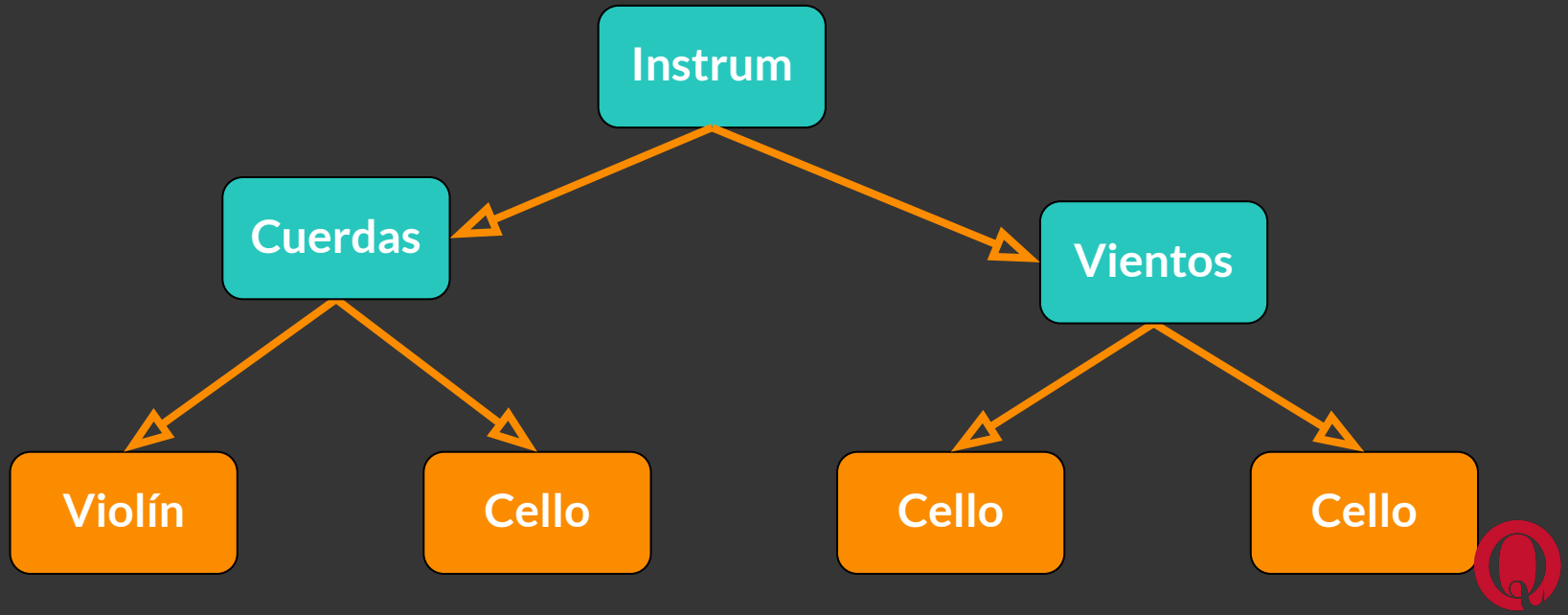
Herencia

Crear nuevas clases (subclases) a partir de clases existentes (superclases), heredando sus características.



Herencia

Crear nuevas clases (subclases) a partir de clases existentes (superclases), heredando sus características.



Polimorfismo

Objetos pueden responder de distinta manera a un mismo mensaje.

Flexibilidad y adaptabilidad del código.

Una única interfaz para entidades de distinto tipo

Ej: Cualquier instrumento puede tocar una nota*, pero una flauta y un violín suenan distinto

OOP en Python

OOP en Python

```python
class Flauta(object):
    def __init__(self, nombre, color, vidaMedia, limpia=True):
        self.nombre = nombre
        self.color = color
        self.vidaMedia = vidaMedia
        self.__notasTocadas = 0
        self.estado = "sano"
        self.limpia = limpia

    def tocar(self, nota):
        if self.estado == "sano":
            print("Flauta toca {0}".format(nota))
            self.__notasTocadas += 1
            self.actualizar_estado()
        else:
            print("Flauta esta rota, no se puede tocar")

    def actualizar_estado(self):
        if self.__notasTocadas >= self.vidaMedia:
            self.estado = "roto"
```

constructor

método

método

OOP en Python

```python
class Flauta(object):
    def __init__(self, nombre, color, vidaMedia, limpia=True):
        self.nombre = nombre
        self.color = color
        self.vidaMedia = vidaMedia
        self.__notasTocadas = 0
        self.estado = "sano"
        self.limpia = limpia

    def tocar(self, nota):
        if self.estado == "sano":
            print("Flauta toca {0}".format(nota))
            self.__notasTocadas += 1
            self.actualizar_estado()
        else:
            print("Flauta esta rota, no se puede tocar")

    def actualizar_estado(self):
        if self.__notasTocadas >= self.vidaMedia:
            self.estado = "roto"
```

```
In [1]: from orquesta import Flauta

In [2]: primeraFlauta = Flauta("Primera flauta", "plata", 2)

In [3]: primeraFlauta.tocar("la")
Flauta toca la

In [4]: primeraFlauta.tocar("re")
Flauta toca re

In [5]: primeraFlauta.tocar("sol")
Flauta esta rota, no se puede tocar

In [6]: primeraFlauta.notasTocadas
Out[6]: 2
```

OOP en Python: Herencia

```python
class Flauta(object):
    def __init__(self, nombre, color, vidaMedia, limpia=True):
        self.nombre = nombre
        self.color = color
        self.vidaMedia = vidaMedia
        self.__notasTocadas = 0
        self.estado = "sano"
        self.limpia = limpia

    def tocar(self, nota):
        if self.estado == "sano":
            print("Flauta toca {0}".format(nota))
            self.__notasTocadas += 1
            self.actualizar_estado()
        else:
            print("Flauta esta rota, no se puede tocar")

    def actualizar_estado(self):
        if self.__notasTocadas >= self.vidaMedia:
            self.estado = "roto"
```

```python
class Violín(object):
    def __init__(self, nombre, color, vidaMedia):
        self.nombre = nombre
        self.color = color
        self.vidaMedia = vidaMedia
        self.notasTocadas = 0
        self.estado = "sano"

    def tocar(self, nota):
        if self.estado == "sano":
            print("Violín toca {0}".format(nota))
            self.notasTocadas += 1
            self.actualizar_estado()
        else:
            print("Violín esta roto, no se puede tocar")

    def actualizar_estado(self):
        if self.notasTocadas >= self.vidaMedia:
            self.estado = "roto"
```

OOP en Python: Herencia

```python
class Instrumento(object):
    def __init__(self, nombre, color, vidaMedia):
        self.nombre = nombre
        self.color = color
        self.vidaMedia = vidaMedia
        self.notasTocadas = 0
        self.estado = "sano"

    def tocar(self, nota):
        pass

    def actualizar_estado(self):
        if self.notasTocadas >= self.vidaMedia:
            self.estado = "roto"
```

```python
class Flauta(Instrumento):
    def __init__(self, nombre, color, vidaMedia, limpia):
        super().__init__(nombre, color, vidaMedia)
        self.limpia = limpia

    def tocar(self, nota):
        if self.estado == "sano":
            print("Flauta toca {0}".format(nota))
            self.notasTocadas += 1
            self.actualizar_estado()
        else:
            print("Flauta esta rota, no se puede tocar")
```

```python
class Violín(Instrumento):
    def tocar(self, nota):
        if self.estado == "sano":
            print("Violín toca {0}".format(nota))
            self.notasTocadas += 1
            self.actualizar_estado()
        else:
            print("Violín esta roto, no se puede tocar")
```

OOP en Python: Composición

```python
class Orquesta(object):
    def __init__(self, nombre, instrumentos):
        self.nombre = nombre
        self.instrumentos = instrumentos

    def agregar_instrumento(self, instrumento):
        self.instrumentos.append(instrumento)

    def tocar(self, nota):
        for instrumento in self.instrumentos:
            instrumento.tocar(nota)
```

```
In [1]: from orquesta import *

In [2]: violin1 = Violín("Violín 1", "café", 10)

In [3]: flauta1 = Flauta("Flauta 1", "blanco", 5, True)

In [4]: orquesta_unq = Orquesta("Orquesta UNQ", [violin1, flauta1])

In [5]: orquesta_unq.tocar("Do")
Violín toca Do
Flauta toca Do

In [6]: orquesta_unq.instrumentos
Out[6]: [<orquesta.Violín at 0x7f7d4055f3d0>, <orquesta.Flauta at 0x7f7d4055f910>]
```

OOP en Python: Encapsulamiento

```python
class Flauta(object):
    def __init__(self, nombre, color, vidaMedia, limpia=True):
        self.nombre = nombre
        self.color = color
        self.vidaMedia = vidaMedia
        self.__notasTocadas = 0
        self.estado = "sano"
        self.limpia = limpia

    def tocar(self, nota):
        if self.estado == "sano":
            print("Flauta toca {0}".format(nota))
            self.__notasTocadas += 1
            self.actualizar_estado()
        else:
            print("Flauta esta rota, no se puede tocar")

    def actualizar_estado(self):
        if self.__notasTocadas >= self.vidaMedia:
            self.estado = "roto"
```

```
In [1]: from orquesta import *

In [2]: flauta1 = Flauta("Flauta 1", "blanco", 1)

In [3]: flauta1.__notasTocadas
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In [3], line 1
----> 1 flauta1.__notasTocadas

AttributeError: 'Flauta' object has no attribute '__notasTocadas'

In [4]: flauta1.tocar("do")
Flauta toca do

In [5]: flauta1.tocar("do")
Flauta esta rota, no se puede tocar
```

Extra: Built-in methods

```python
class Flauta(object):
    def __init__(self, nombre, color, vidaMedia, limpia=True):
        self.nombre = nombre
        self.color = color
        self.vidaMedia = vidaMedia
        self.__notasTocadas = 0
        self.estado = "sano"
        self.limpia = limpia

    def tocar(self, nota):
        if self.estado == "sano":
            print("Flauta toca {0}".format(nota))
            self.__notasTocadas += 1
            self.actualizar_estado()
        else:
            print("Flauta esta rota, no se puede tocar")

    def actualizar_estado(self):
        if self.__notasTocadas >= self.vidaMedia:
            self.estado = "roto"
```

```
In [1]: from orquesta import Flauta

In [2]: flauta1 = Flauta("Flauta 1", "blanco", 1)

In [3]: print(flauta1)
<orquesta.Flauta object at 0x7f2687816ed0>
```

Extra: Built-in methods

```python
class Flauta(object):
    def __init__(self, nombre, color, vidaMedia, limpia=True):
        self.nombre = nombre
        self.color = color
        self.vidaMedia = vidaMedia
        self.__notasTocadas = 0
        self.estado = "sano"
        self.limpia = limpia

    def tocar(self, nota):
        if self.estado == "sano":
            print("Flauta toca {0}".format(nota))
            self.__notasTocadas += 1
            self.actualizar_estado()
        else:
            print("Flauta esta rota, no se puede tocar")

    def actualizar_estado(self):
        if self.__notasTocadas >= self.vidaMedia:
            self.estado = "roto"

    def __str__(self):
        return "Este instrumento es una flauta"
```

```
In [1]: from orquesta import Flauta

In [2]: flauta1 = Flauta("Flauta 1", "blanco", 1)

In [3]: print(flauta1)
Este instrumento es una flauta
```

Programación Orientada a Objetos

¿Preguntas?