Programación Orientada a Objetos

WTPC 2019

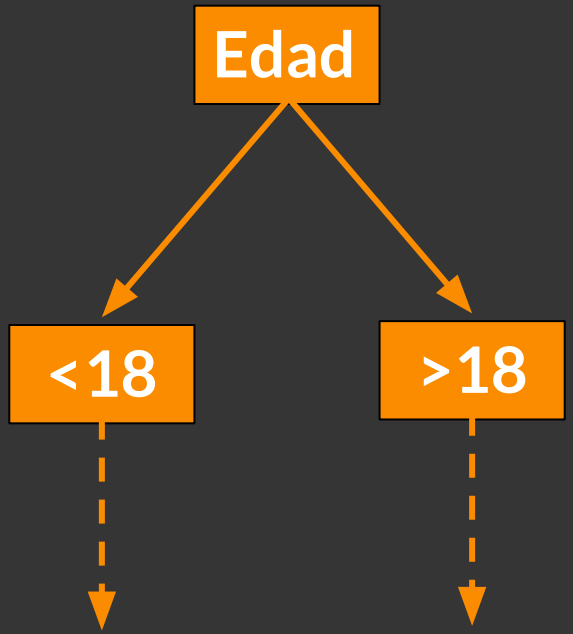
Estructurado

Órdenes a la computadora

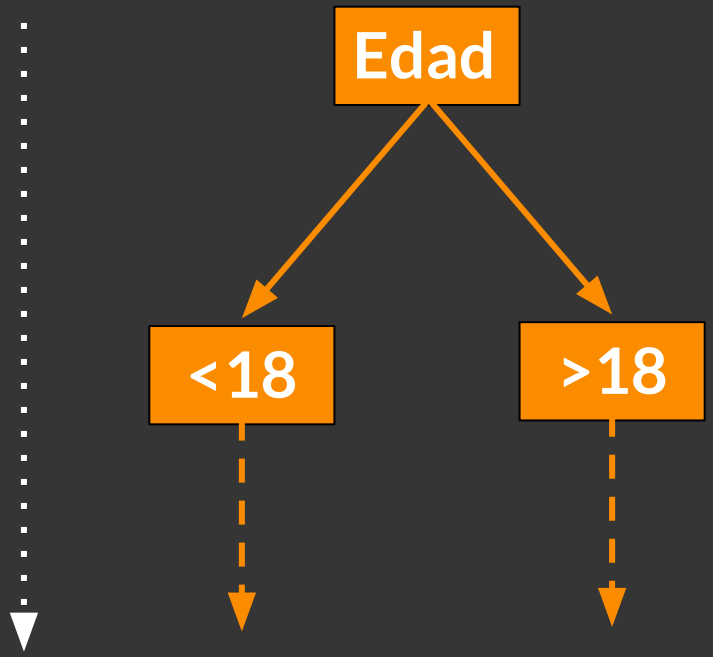
Lectura lineal

Segmentado en “funciones”

Estructurado

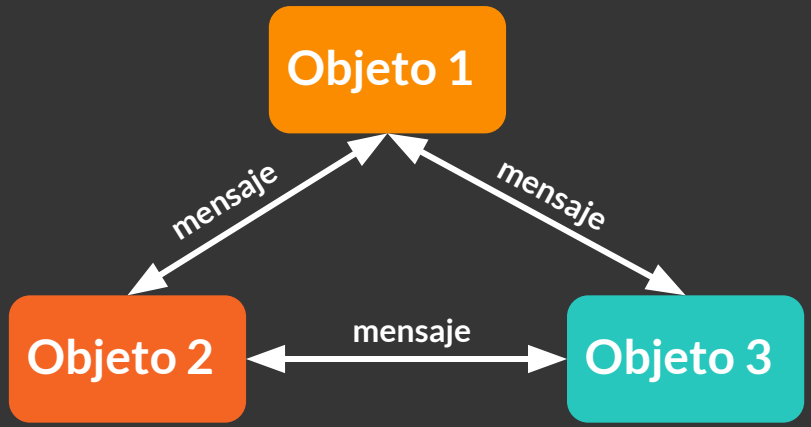


Estructurado

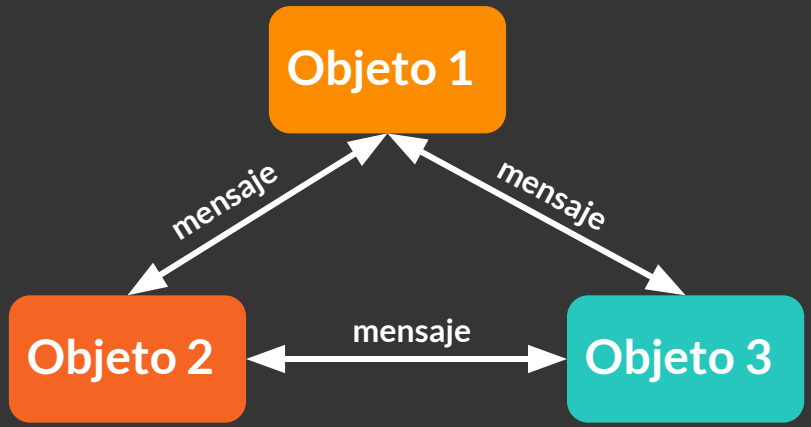



```
1 # file: esPrimo.py
2
3 numeroString = input('Número menor a 100: ')
4 numero = int(numeroString)
5 noEsPrimo = False
6
7 for primo in [2, 3, 5, 7]:
8   if numero % primo == 0:
9     noEsPrimo = True
10    break
11
12 if noEsPrimo:
13   print("El número {0} no es primo".format(numero))
14 else:
15   print("El número {0} es primo".format(numero))
```

Orientado a objetos



Orientado a objetos



Estado

Todas las propiedades de un objeto

Comportamiento

Cómo un objeto reacciona frente a una interacción.

Identidad

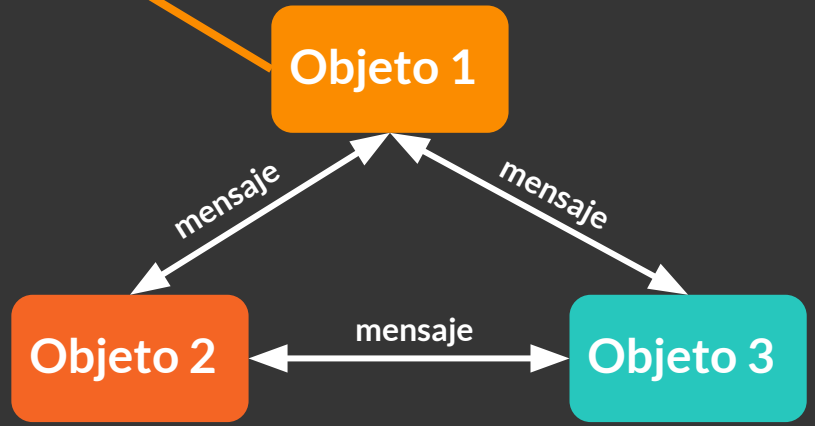
Distintos objetos pueden tener idénticos estados y el mismo comportamiento, pero cada uno tendrá su identidad.

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*

Clase: *blueprint* de los objetos

El concepto *abstracto* detrás del objeto *concreto*

mamífero

humano

Características

Composición

Encapsulamiento

Herencia

Polimorfismo

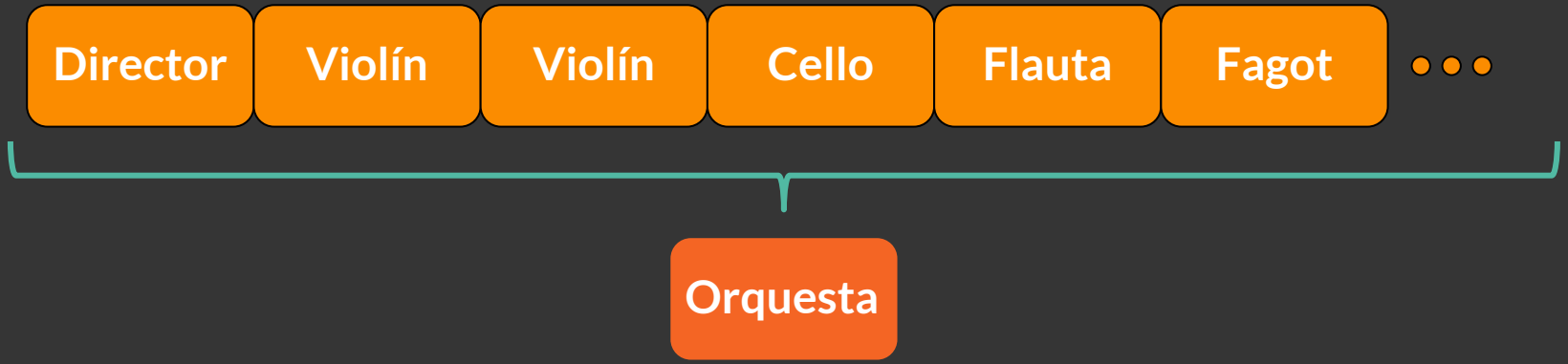
Composición

La forma natural de crear objetos es construyéndolos a partir de objetos ya existentes.

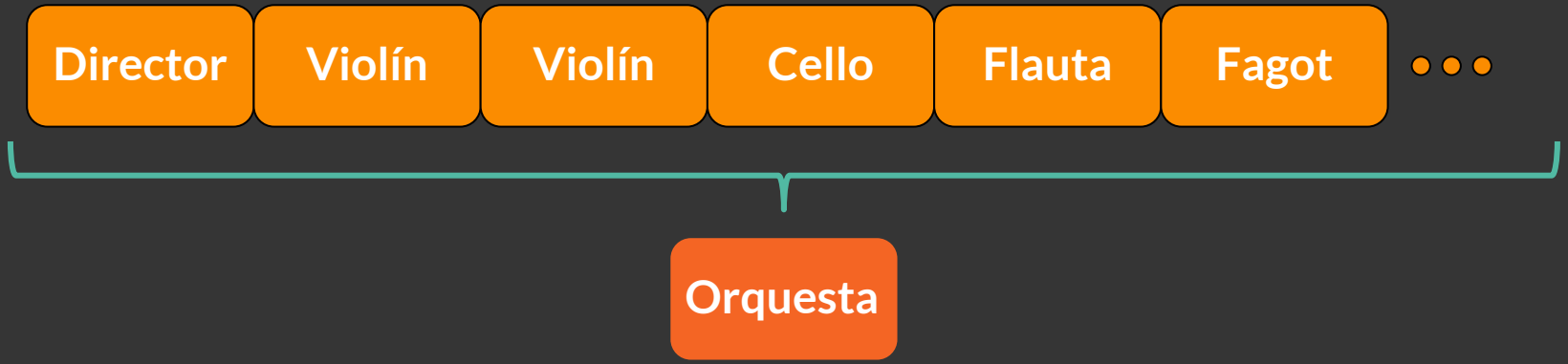
De esta manera, un sistema complejo se compone de subsistemas más simples.

Composición

Composición



Composición



Un objeto puede tener como atributos otros objetos

Encapsulamiento

Uno no siempre quiere que el usuario tenga acceso a todos los métodos o los atributos de una clase.

Se denomina encapsulamiento al ocultamiento del estado, es decir, de los datos miembro de un objeto de manera que sólo se pueda cambiar mediante las operaciones definidas para ese objeto.

Encapsulamiento

Prohibir el acceso a algunos métodos de una clase

Prohibir el acceso (directo) a los atributos de una clase

Permitir que la clase se modifique a sí misma con métodos

Encapsulamiento

Prohibir el acceso a algunos métodos de una clase

Prohibir el acceso (directo) a los atributos de una clase

Permitir que la clase se modifique a sí misma con métodos

En la práctica: elegir qué métodos y atributos son *privados*

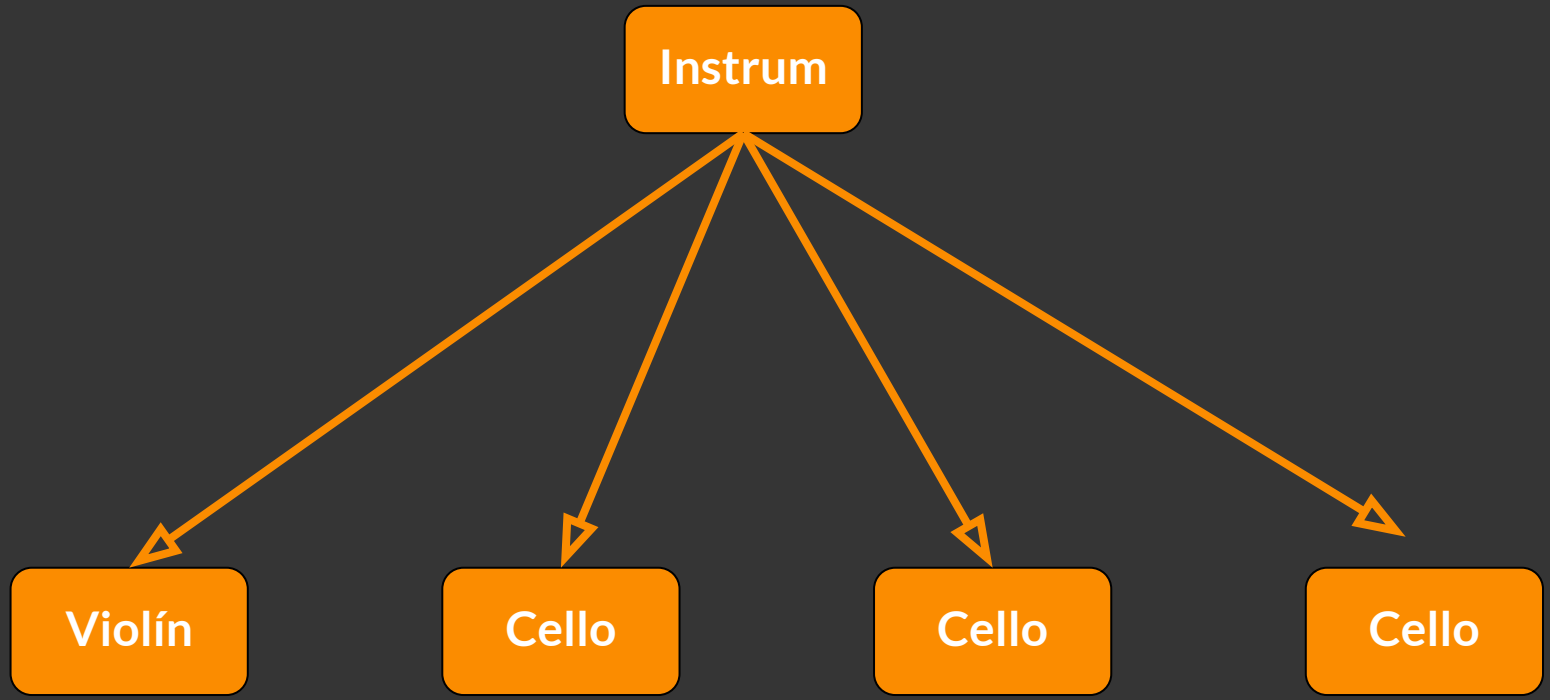
caja negra

Herencia

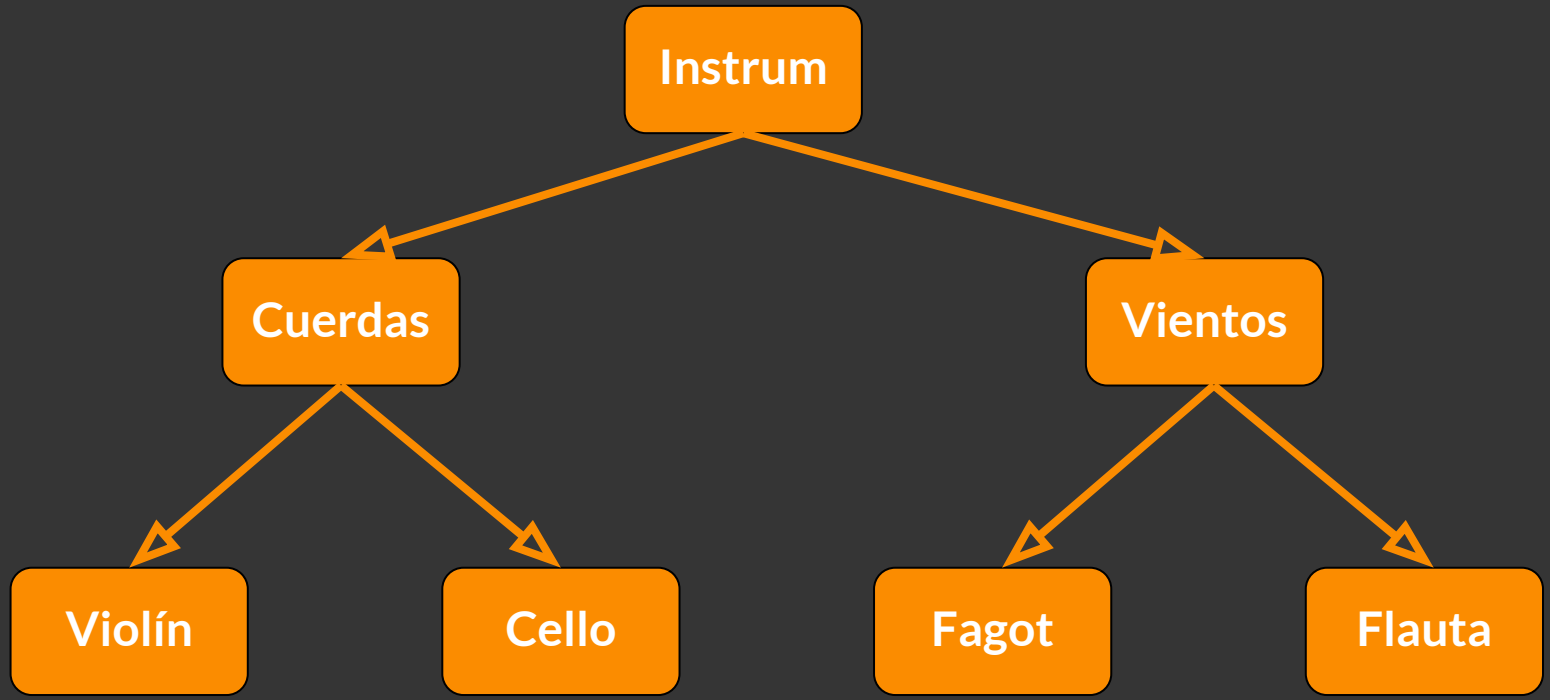
Crear nuevas clases partiendo de una clase preexistente, evitando el rediseño, la modificación y la verificación de la parte ya implementada.

La herencia facilita la creación de objetos a partir de otros ya existentes e implica que una subclase obtiene todo el comportamiento (métodos) y eventualmente los atributos (variables) de su superclase

Herencia



Herencia



El violín es un tipo de cuerdas.

Cuerdas es un tipo de instrumento.

Polimorfismo

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

Una única interfaz para entidades de distinto tipo. Es decir, mismo mensaje a objetos de tipos distintos.

Polimorfismo

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

Una única interfaz para entidades de distinto tipo

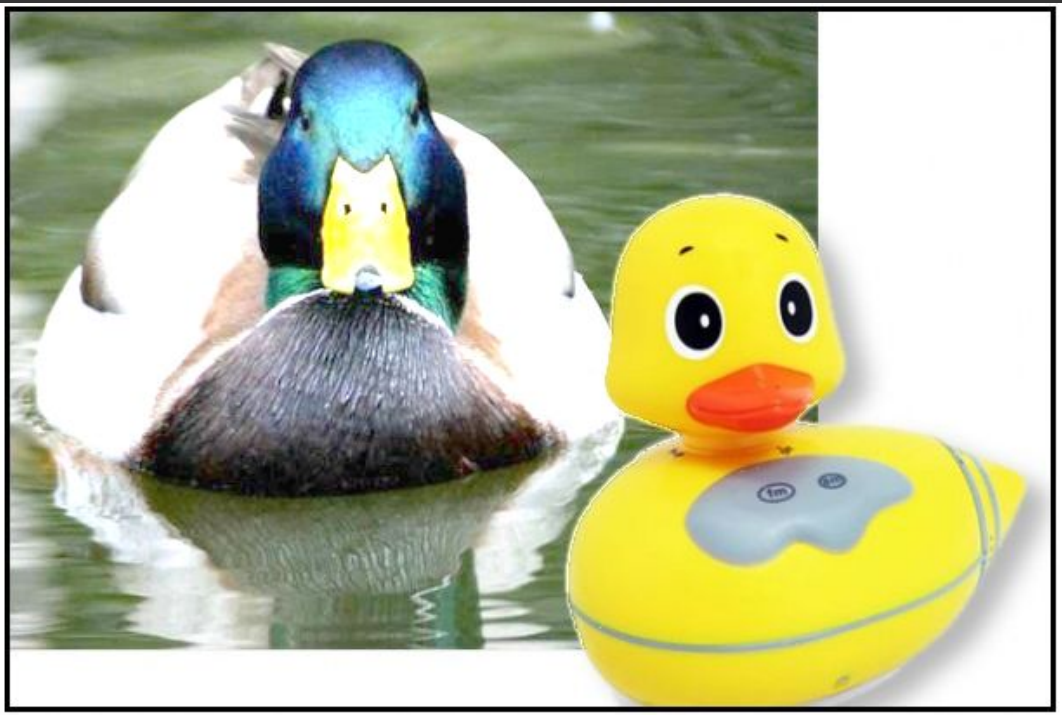
¿Siempre puedo reemplazar a un hijo por su padre?

Subtipado: Principio de Liskov

Creamos una subclase. La nueva clase derivada debe extender sin reemplazar funcionalidades de la clase padre.

El principio de sustitución de Liskov plantea que si un programa utiliza una clase base, entonces la referencia a la clase base debe poder ser reemplazada por la clase derivada, sin que eso afecte la funcionalidad de un programa.

Subtipado


LISKOV SUBSTITUTION PRINCIPLE
If It Looks Like A Duck, Quacks Like A Duck, But Needs Batteries - You Probably Have The Wrong Abstraction

OOP en Python

OOP en Python

```
1 class Flauta(object):
2   def __init__(self, nombre, color, vidaMedia):
3     self.nombre = nombre
4     self.color = color
5     self.vidaMedia = vidaMedia
6     self.notasTocadas = 0
7
8   def tocar(self, nota):
9     print("Flauta toca {0}".format(nota))
10    self.notasTocadas += 1
11
12  def estado(self):
13    if self.notasTocadas < self.vidaMedia:
14      print("La flauta está sana")
15    else:
16      print("La flauta está rota")
```

constructor

método

método

OOP en Python

```python
class Flauta(object):
  def __init__(self, nombre, color, vidaMedia):
    self.nombre = nombre
    self.color = color
    self.vidaMedia = vidaMedia
    self.notasTocadas = 0

  def tocar(self, nota):
    print("Flauta toca {0}".format(nota))
    self.notasTocadas += 1

  def estado(self):
    if self.notasTocadas < self.vidaMedia:
      print("La flauta está sana")
    else:
      print("La flauta está rota")
```

```
>>> from orquesta import Flauta
>>> primeraFlauta = Flauta("Primera
flauta", "plata", 10)
>>> primeraFlauta.tocar("la")
Flauta toca la
>>> primeraFlauta.tocar("sol")
Flauta toca sol
>>> primeraFlauta.tocar("mi")
Flauta toca mi
>>> primeraFlauta.estado()
La flauta está sana
>>> primeraFlauta.notasTocadas
3
```

OOP en Python: Herencia

```python
class Flauta(object):
  def __init__(self, nombre, color, vidaMedia):
    self.nombre = nombre
    self.color = color
    self.vidaMedia = vidaMedia
    self.notasTocadas = 0

  def tocar(self, nota):
    print("Flauta toca {0}".format(nota))
    self.notasTocadas += 1

  def estado(self):
    if self.notasTocadas < self.vidaMedia:
      print("La flauta está sana")
    else:
      print("La flauta está rota")
```

```python
class Violin(object):
  def __init__(self, nombre, color, vidaMedia):
    self.nombre = nombre
    self.color = color
    self.vidaMedia = vidaMedia
    self.notasTocadas = 0

  def tocar(self, nota):
    print("Violín toca {0}".format(nota))
    self.notasTocadas += 1

  def estado(self):
    if self.notasTocadas < self.vidaMedia:
      print("El violín está sano")
    else:
      print("El violín está roto")
```

OOP en Python: Herencia

```
2 class Instrumento(object):
3   def __init__(self, nombre, color, vidaMedia):
4     self.nombre = nombre
5     self.color = color
6     self.vidaMedia = vidaMedia
7     self.notasTocadas = 0
8
9   def tocar(self, nota):
10    self.notasTocadas += 1
```

```
13 class Flauta(Instrumento):
14   def tocar(self, nota):
15     print("Flauta toca {0}".format(nota))
16     self.notasTocadas += 1
17
18   def estado(self):
19     if self.notasTocadas < self.vidaMedia:
20       print("Sano")
21     else:
22       print("Roto")
```

```
25 class Violin(Instrumento):
26   def tocar(self, nota):
27     print("Violín toca {0}".format(nota))
28     self.notasTocadas += 1
29
30   def estado(self):
31     if self.notasTocadas < self.vidaMedia:
32       print("Sano")
33     else:
34       print("Roto")
```

OOP en Python: Herencia

```
2 class Instrumento(object):
3   def __init__(self, nombre, color, vidaMedia):
4     self.nombre = nombre
5     self.color = color
6     self.vidaMedia = vidaMedia
7     self.notasTocadas = 0
8
9   def tocar(self, nota):
10    self.notasTocadas += 1
```

```
13 class Flauta(Instrumento):
14   def tocar(self, nota):
15     print("Flauta toca {0}".format(nota))
16     self.notasTocadas += 1
17
18   def estado(self):
19     if self.notasTocadas < self.vidaMedia:
20       print("Sano")
21     else:
22       print("Roto")
```

```
>>> from orquesta import Flauta
>>> primeraFlauta = Flauta("Primera
flauta", "plata", 10)
>>> primeraFlauta.tocar("mi")
Flauta toca mi
>>> primeraFlauta.notasTocadas
>>> primeraFlauta.estado()
Sano
```

OOP en Python: Encapsulamiento

```python
class Flauta(object):
  def __init__(self, nombre, color, vidaMedia):
    self.nombre = nombre
    self.color = color
    self.vidaMedia = vidaMedia
    self.__notasTocadas = 0

  def tocar(self, nota):
    print("Flauta toca {0}".format(nota))
    self.__notasTocadas += 1

  def estado(self):
    if self.__notasTocadas < self.vidaMedia:
      print("Sano")
    else:
      print("Roto")
```

```
>>> from orquesta import Flauta
>>> primeraFlauta = Flauta("Primera flauta",
"plata", 10)
>>> primeraFlauta.tocar("mi")
Flauta toca mi
>>> primeraFlauta.estado()
Sano
>>> primeraFlauta.__notasTocadas

-------------------------------------------
AttributeError
Traceback (most recent call last)
<ipython-input-10-d2d7ea4a537a> in
<module>()
----> 1 primeraFlauta.__notasTocadas

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

OOP en Python: Encapsulamiento

```python
class Flauta(object):
  def __init__(self, nombre, color, vidaMedia):
    self.nombre = nombre
    self.color = color
    self.vidaMedia = vidaMedia
    self.__notasTocadas = 0

  def tocar(self, nota):
    print("Flauta toca {0}".format(nota))
    self.__notasTocadas += 1

  def estado(self):
    if self.__notasTocadas < self.vidaMedia:
      print("Sano")
    else:
      print("Roto")

  def getNotasTocadas(self):
    print(self.__notasTocadas)
```

```
>>> from orquesta import Flauta
>>> primeraFlauta = Flauta("Primera flauta",
"plata", 10)
>>> primeraFlauta.tocar("mi")
Flauta toca mi
>>> primeraFlauta.estado()
Sano
>>> primeraFlauta.getNotasTocadas()
1
```

Extra: Built-in methods

```
class Flauta(object):
  def __init__(self, nombre, color, vidaMedia):
    self.nombre = nombre
    self.color = color
    self.vidaMedia = vidaMedia
    self.__notasTocadas = 0

  def tocar(self, nota):
    print("Flauta toca {0}".format(nota))
    self.__notasTocadas += 1

  def estado(self):
    if self.__notasTocadas < self.vidaMedia:
      print("Sano")
    else:
      print("Roto")

  def getNotasTocadas(self):
    print(self.__notasTocadas)
```

```
>>> from orquesta import Flauta
>>> primeraFlauta = Flauta("Primera flauta",
"plata", 10)
>>> print(primeraFlauta)
<__main__.Flauta object at 0x7f949603e978>
```

Extra: Built-in methods

```
class Flauta(object):
  def __init__(self, nombre, color, vidaMedia):
    self.nombre = nombre
    self.color = color
    self.vidaMedia = vidaMedia
    self.__notasTocadas = 0

  def tocar(self, nota):
    print("Flauta toca {0}".format(nota))
    self.__notasTocadas += 1

  def estado(self):
    if self.__notasTocadas < self.vidaMedia:
      print("Sano")
    else:
      print("Roto")

  def getNotasTocadas(self):
    print(self.__notasTocadas)

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

```
>>> from orquesta import Flauta
>>> primeraFlauta = Flauta("Primera flauta",
"plata", 10)
>>> print(primeraFlauta)
Este instrumento es una flauta
```

Extra: Built-in methods

Son métodos “por defecto”, con un cierto comportamiento “por defecto”. Los dos que vimos hasta ahora fueron __init__ y __str__