ACÁMICA

TEMA DEL DÍA

Programación orientada a objetos

Python es lo que se denomina un lenguaje de programación orientado a objetos. Hoy vamos a conocer qué significa esta categoría y cuáles son las características de sus principales componentes: Clases y Objetos.

Agenda

Daily

Explicación: Clases y Objetos

Break.

Hands-on training

Dudas con el proyecto.

Cierre.

Daily

Daily

Sincronizando...

Toolbox

¿Cómo te ha ido?
¿Obstáculos?
¿Cómo seguimos?

Challenge

¿Cómo te ha ido?
¿Obstáculos?
¿Cómo seguimos?

Programación estructurada y Programación orientada a objetos

La Programación Estructurada o Procedimental y la Programación Orientada a Objetos son los dos paradigmas de programación más comunes y utilizados.

Conviven entre sí,
NO son excluyentes

¡Muchos lenguajes usan este paradigma!

Programación Estructurada:

- Es el paradigma con el que venimos y seguiremos trabajando.
- Hace foco en un flujo de código lineal.
- Generalmente definimos línea a línea el código, las variables y las acciones que usamos, solamente interrumpida por toma de decisiones con estructuras if/else

Programación orientada a objetos:

- Se complementará con la programación estructurada.
- Incluye la noción de Clases y Objetos.
- Agrupa funciones y variables en “paquetes” que llamamos Objetos.
- Las Clases son el molde que definirá los Objetos.

Clases y Objetos

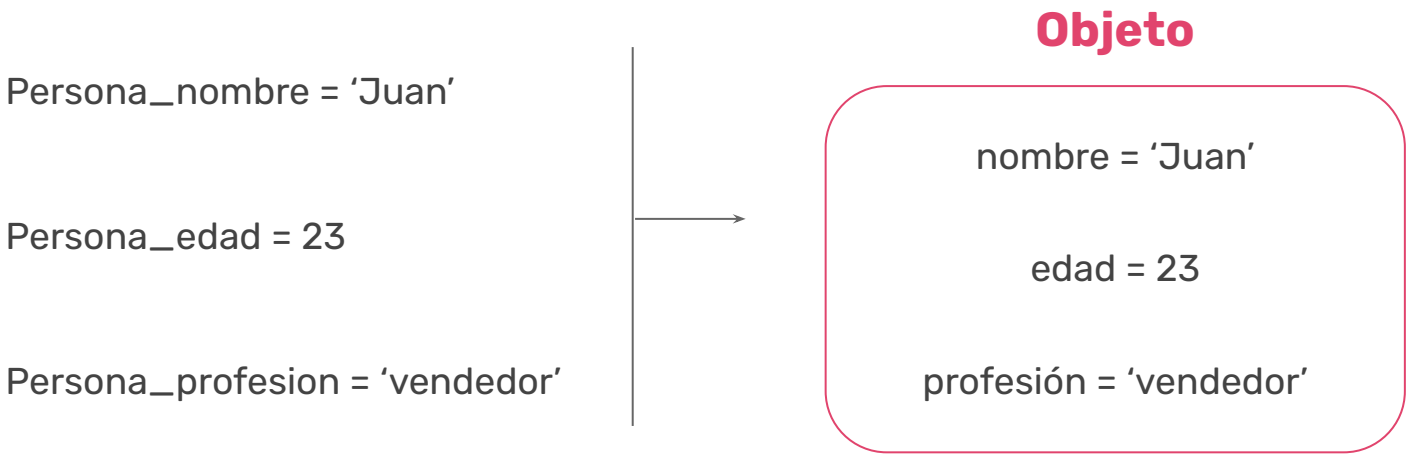
Utilizan clases y objetos

Lenguajes orientados a la programación estructurada

Lenguajes con programación orientada a objetos

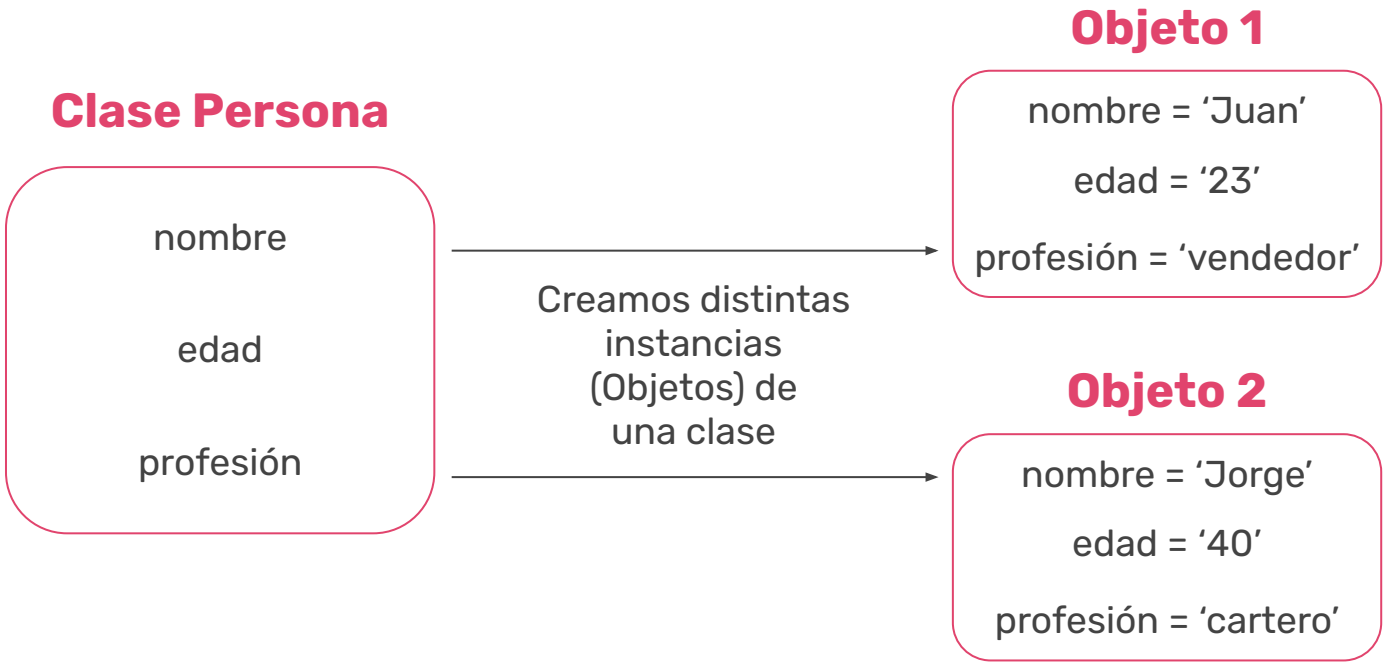
¿Qué es un objeto?

Un objeto es como un un paquete de variables y funciones que conviene tener agrupados por consistencia y comodidad.



¿Qué es una clase?

Los objetos suelen crearse a partir de unas *plantillas* a las que llamamos clases.



Creando una clase

Nombre que le doy a la clase.

```
[ ]: class Persona:
         def __init__(self, nombre, edad):
             self.nombre = nombre
             self.edad = edad
```

Creando una clase

Nombre que le doy a la clase.

```
[ ]: class Persona:
         def __init__(self, nombre, edad):
             self.nombre = nombre
             self.edad = edad
```

Atributos (variables) que van a tener los objetos de esta clase.

Creando una clase

Nombre que le doy a la clase.

```
[ ]: class Persona:
         def __init__(self, nombre, edad):
             self.nombre = nombre
             self.edad = edad
```

Cuando creemos una instancia de esta clase, vamos a tener que pasarle un nombre y una edad de la persona

Atributos (variables) que van a tener los objetos de esta clase.

Creando un objeto

```
[7]: p1 = Persona("Juan", 26)

     print(p1.name)
     print(p1.edad)
```

```
Juan
26
```

Creando un objeto

```
[7]: p1 = Persona("Juan", 26)

     print(p1.name)
     print(p1.edad)
```

```
Juan
26
```

Recordar: lo que hicimos fue crear una instancia de la clase Persona

```
[ ]: class Persona:
         def __init__(self, nombre, edad):
             self.nombre = nombre
             self.edad = edad
```

Creando un objeto

```
[7]: p1 = Persona("Juan", 26)

     print(p1.name)
     print(p1.edad)
```

```
Juan
26
```

Recordar: lo que hicimos fue crear una instancia de la clase Persona

```
[ ]: class Persona:
         def __init__(self, nombre, edad):
             self.nombre = nombre
             self.edad = edad
```

Métodos

Los métodos son funciones propias de una clase. Es decir, son funciones que definimos que actúan sobre un tipo de objeto determinado.

[15]:
```python
class Persona:
    """
    Esta es una clase donde se agregan todos los datos
    respecto a una persona
    """
    def __init__(self, nombre, edad):
        # Todo lo que definamos en __init__ se corre
        # al crear una instancia de la clase
        self.nombre = nombre
        self.edad = edad
    def mePresento(self):
        print("Hola, me llamo " + self.nombre)
```

Métodos

[15]:
```python
class Persona:
    """
    Esta es una clase donde se agregan todos los datos
    respecto a una persona
    """
    def __init__(self, nombre, edad):
        # Todo lo que definamos en __init__ se corre
        # al crear una instancia de la clase
        self.nombre = nombre
        self.edad = edad
    def mePresento(self):
        print("Hola, me llamo " + self.nombre)
```

Definimos una función que imprime un texto en pantallas

Métodos

[15]:
```
class Persona:
    """
    Esta es una clase donde se agregan todos los datos
    respecto a una persona
    """
    def __init__(self, nombre, edad):
        # Todo lo que definamos en __init__ se corre
        # al crear una instancia de la clase
        self.nombre = nombre
        self.edad = edad
    def mePresento(self):
        print("Hola, me llamo " + self.nombre)
```

[16]:
```
p1 = Persona("Juan", 26)
p1.mePresento()
```

```
Hola, me llamo Juan
```

Usamos el método

Métodos

Un método puede modificar en valor de algún atributo del objeto.

[34]:
```python
class Persona:
    """
    Esta es una clase donde se agregan todos los datos
    respecto a una persona
    """
    def __init__(self, nombre, edad):
        # Todo lo que definamos en __init__ se corre
        # al crear una instancia de la clase
        self.nombre = nombre
        self.edad = edad
    def cumplirAnios(self):
        self.edad = self.edad + 1
```

Cambia el valor del atributo edad, lo aumenta en 1.

Métodos

Un método puede modificar en valor de algún atributo del objeto.

[34]:
```python
class Persona:
    """
    Esta es una clase donde se agregan todos los datos
    respecto a una persona
    """
    def __init__(self, nombre, edad):
        # Todo lo que definamos en __init__ se corre
        # al crear una instancia de la clase
        self.nombre = nombre
        self.edad = edad
    def cumplirAnios(self):
        self.edad = self.edad + 1
```

[35]:
```python
p1 = Persona("Juan", 26)
p1.cumplirAnios()
p1.edad
```

[35]:
```
27
```

Clases: Beneficios

La programación orientada a objetos nos propone una **manera de trabajar** a la hora de escribir nuestro programa.

No sólo es práctico y ordenado, sino que también muchas veces nos ayuda a mantener la consistencia del código.

[1]:
```
class Departamento:
    def __init__(self, calle, altura, sup_total, sup_cubierta):
        self.calle = calle
        self.altura = altura
        self.sup_total = sup_total
        if sup_cubierta < sup_total:
            self.sup_cubierta = self.sup_cubierta
        else:
            self.sup_cubierta = self.sup_total
```

[3]:
```
depto_1 = Departamento('Humboldt',1122,50,455)
depto_1.sup_cubierta
```

[3]:
```
50
```

Clases: Beneficios

Más adelante veremos que el formato de clases y métodos propuestos por la librería scikit-learn es muy útil y es el más usado en la comunidad de data science.

¡BREAK!

Ph. Credit: Drew Coffmann

Hands-on training

Hands-on training

DS_Toolbox_10_Clases.ipynb

Primer Proyecto

Primer proyecto.

Ejercitación y dudas: 30 minutos.

Recursos

Programación orientada a objetos

Este artículo es bastante general y conceptual, con un enfoque muy parecido al de la Toolbox. Además, ¡está en castellano!

https://pythones.net/clases-y-metodos-python-oop/

Este artículo explica las Clases y Objetos con un objetivo bien particular: el de **crear tus propios clases en Python**. Si bien nosotros no necesitaremos crear nuestras propias clases, es útil saber cómo se hace. ¡También está en castellano!

https://programacion.net/articulo/como_funcionan_las_clases_y_objetos_en_python_1505

Para la próxima

- Termina el notebook de hoy.
- Lee la Toolbox 11

ACÁMICA