

Unidad 1

CIENCIA DE DATOS CON PYTHON: RECOLECCIÓN, ALMACENAMIENTO Y PROCESO



Fuente: adobestock/84383512



Autor:

Amaury Giovanni Méndez Aguirre

ÍNDICE

Introducción.	3
Ciencia de datos con Python_ recolección, almacenamiento y proceso.	4
¿Qué es Python?	5
Instalación de un intérprete de Python	5
Comenzando a programar	7
Comentar un código.	7
Variables	8
Creación de variables.	10
Ejercicios	11
Programación estructurada vs Programación orientada a objetos.	12
Herencia	17
Conclusiones	18
Bibliografía.	19

INTRODUCCIÓN

Python es un lenguaje que se ha popularizado en los últimos años por su sencillez y facilidad tanto en su uso como en su aprendizaje. Por tal motivo vas a encontrar los aspectos relevantes y fundamentales del lenguaje y su forma de programar con base en la metodología orientada a objetos, preparándote el camino para usar las librerías especializadas en este lenguaje para la ciencia de datos. Vas a aprender cómo desarrollar programas en Python, cómo programar orientado a objetos, qué son las comparaciones y condiciones en programación, bucles o ciclos y los tipos de datos más usados como las listas y los diccionarios. A lo largo de este curso, irás encontrando ejemplos de código que podrás ir desarrollando para entrenar tus habilidades de programación, así que empecemos.



Ciencia de datos con Python_ recolección, almacenamiento y proceso



¿Qué es Python?

Como lenguaje de programación, Python tiene una estructura o sintaxis muy sencilla en comparación con otros lenguajes de programación como C++ o Java. Esto no le resta potencia a la hora de ejecutar tareas complejas en varios escenarios con menos líneas de código y en menor tiempo, como en el caso de la Ciencia de Datos, donde es preferido junto al lenguaje R precisamente por esta razón (García, 2018)

Desde un punto de vista más técnico, Python es un lenguaje de programación interpretado, orientado a objetos y de alto nivel. Fue creado por Guido van Rossum en 1991, con la idea de ser un lenguaje fácil, intuitivo, y de código abierto. Es uno de los lenguajes que más ha crecido, debido a su compatibilidad con la mayoría de los sistemas operativos y la capacidad de integrarse con otros lenguajes de programación. Aplicaciones como Dropbox, Reddit e Instagram son escritas en Python (Python para Data Science, 2022)

La programación estructurada presenta simplemente una secuencia de instrucciones en donde su potencial se evidencia en bloques de código que pueden reutilizarse indefinidamente, llamados **funciones**. Para la programación orientada a objetos (POO), esto es simplemente una característica de su potencial, puesto que estos “objetos” pueden ser creados y ser independientes de su bloque de código original. Esta diferencia será clave durante todo el módulo y mostrará el potencial de programar en Python.

Instalación de un intérprete de Python

Para empezar a programar en Python, vamos a optar por instalar un entorno de desarrollo conocido como **Jupyter Lab**. Una vez instalado, podremos correr nuestro primer programa (se recomienda que se digite la instrucción completa y no que se emplee la herramienta de copiar y pegar):



Función:

En programación, una función es un bloque de código que puede reutilizarse sin la necesidad de volver a definir su contenido.

Jupyter Lab:

Es un entorno de desarrollo con interfaz web para la programación en Python. El proceso de instalación se puede ver en el siguiente video: <https://youtu.be/tzxcKmFj24A>

```
print("Hola Mundo!")
```

suma

```
print(2 + 3)  
>> 5
```

multiplicación

```
print(3 * 4)  
>> 12
```

división

```
print(10 / 3)  
>> 3.3333333333333335
```

división parte entera

```
print(10 // 3)  
>> 3
```

resto de la división

```
print(20 % 6)  
>> 2
```

hacer mayúsculas un texto

```
print("programación para ciencia de datos".upper( ))  
>> PROGRAMACIÓN PARA CIENCIA DE DATOS
```

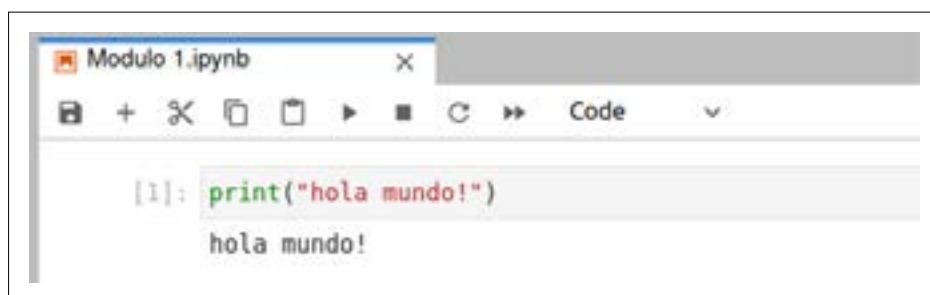
unir textos

```
print("programación" + "en Python")  
>> Programaciónen Python
```

separar palabras en un texto

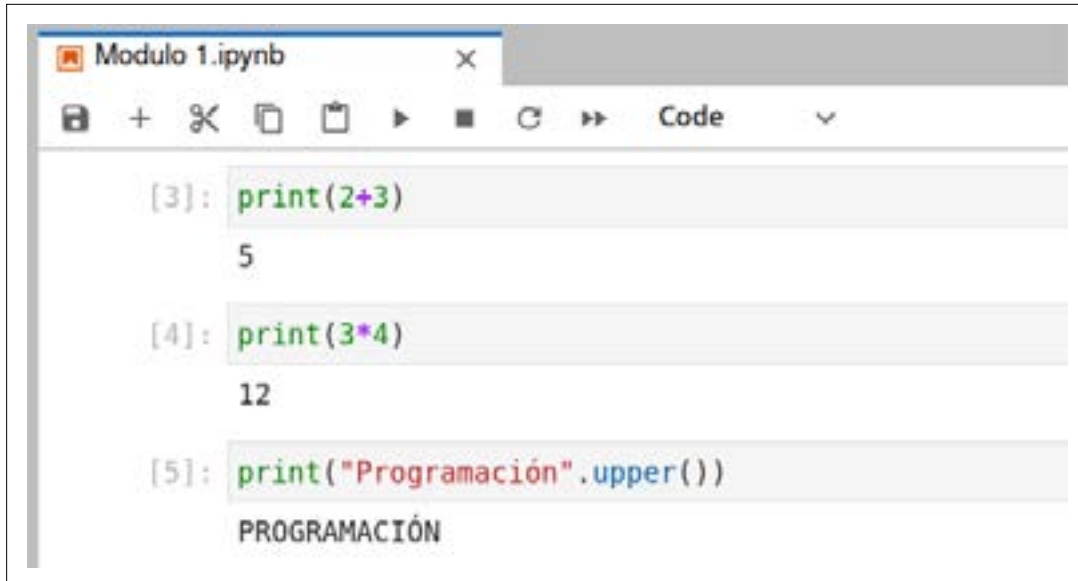
```
print("Ciencia de Datos".split( ))  
>> [ 'Ciencia', 'de', 'Datos' ]
```

Figura 1



Fuente: propia

Figura 2



```
[3]: print(2+3)
5

[4]: print(3*4)
12

[5]: print("Programación".upper())
PROGRAMACIÓN
```

Fuente: propia

La función **print()** recibe como argumento un texto que está delimitado por las comillas dobles y es condición necesaria para que Python lo interprete como un texto. Esta función ya se encuentra precargada al iniciar cualquier intérprete de Python y no es necesario importarla, como veremos más adelante con otras funciones.

Comenzando a programar

Para empezar a programar, debes entender que un lenguaje de programación en esencia puede realizar operaciones matemáticas, pero también puede realizar otro tipo de operaciones o instrucciones con diversos tipos de datos que no solo son numéricos. Veamos algunos ejemplos y ejercicios que puedes ir desarrollando dentro de Jupyter Lab: Observa de estos ejemplos que, así como se pueden usar los símbolos aritméticos para realizar operaciones matemáticas, también se pueden usar símbolos lingüísticos como las comillas, la coma y el punto, y en este último su uso se hace inherente a la POO, como al usar los métodos **.upper()** y **.split()** de la clase *string*, pero este es un tema que trataremos más adelante.

¿Te atreves a realizar operaciones como $(3 + 4 * 7 * (5 - 2) / (10 ** 2))$?

Comentar un código

En Python, al igual que en otros lenguajes de programación, puedes realizar pequeños o grandes comentarios para explicar líneas de código, dejar información importante o algún otro uso que el programador desee dejar y para esto se puede realizar bien sea un comentario de una sola línea o comentarios de varias líneas.

Para comentar una sola línea, basta con utilizar el símbolo numeral #, y para comentar varias líneas se suele emplear la comilla simple o las comillas dobles, tres veces al inicio y tres veces al final. Veámos como:

```
#Comentario de una sola línea

"""
Comentarios
de varias líneas
según requiera o desee el programador
"""
```

Variables

Una variable es simplemente un espacio en memoria que se usará temporalmente para almacenar algún tipo de dato. Veámos un ejemplo:

```
edad = 17

print( edad )

>> 17
```

Observa que se ha usado el símbolo igual (=) para realizar este almacenamiento temporal del número 17, en una variable creada con el nombre edad. El número 17 resulta ser entonces un tipo de dato que en este caso es un número entero. En este ejemplo, para mostrar el contenido de la variable, se ha usado la función **print()** y como resultado se muestra el número almacenado.



Variable:

Es simplemente un espacio en memoria que se usará temporalmente para almacenar algún tipo de dato.

Para indagar en Python por cuál tipo de dato ha sido almacenado en la **variable**, podemos utilizar una función denominada **type()** la cual cumple este propósito:

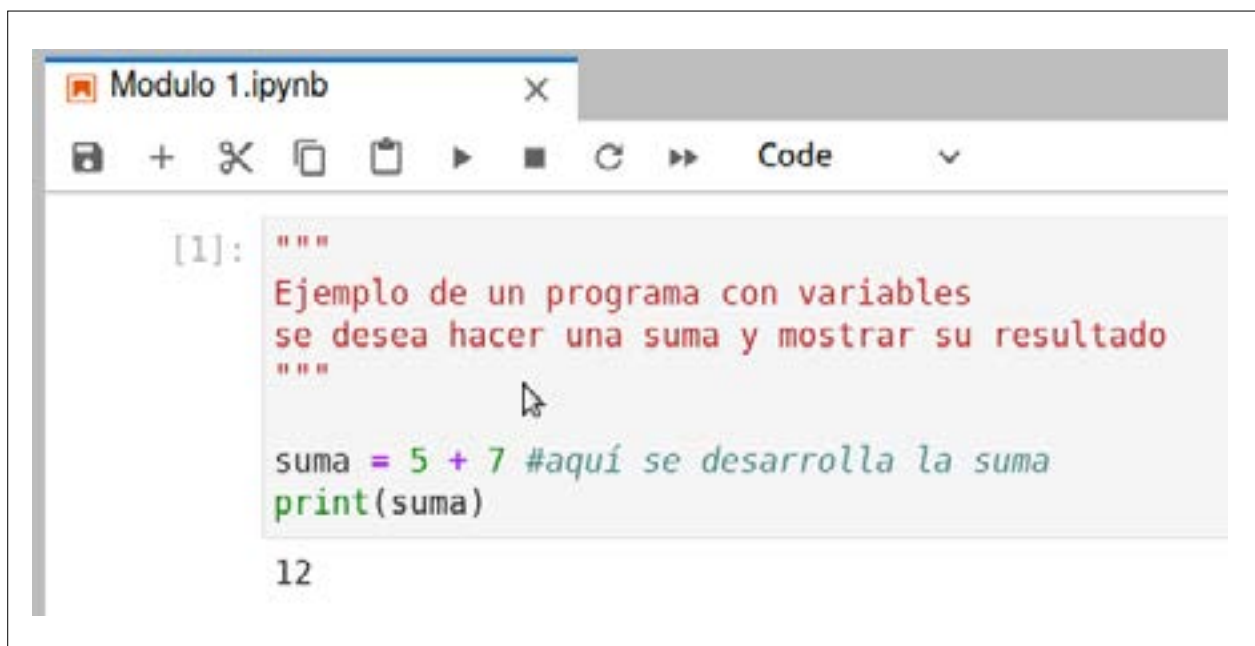
```
type( edad )

>> int
```


Algo muy importante para resaltar, es que las variables pueden almacenar resultados de distintas operaciones aritméticas o de objetos como se muestra a continuación:

```
suma = 5 + 7  
  
print( suma )  
  
>> 12  
  
nombre = "Ciencia de Datos".upper( )  
  
print( nombre )  
  
>> CIENCIA DE DATOS
```

Figura 3



```
[1]: """  
Ejemplo de un programa con variables  
se desea hacer una suma y mostrar su resultado  
"""  
  
suma = 5 + 7 #aquí se desarrolla la suma  
print(suma)  
  
12
```

Fuente: propia

Creación de variables

Las variables pueden ser creadas bajo ciertas reglas:

Puede empezar por una letra, (a - z, A - Z) o con el guión bajo, ejemplo:

```
ancho = 15  
Altura = 20  
_radio = 3.5
```

Puede contener después otras letras, números y el guión bajo, ejemplo:

```
primer_nombre = "Giovanni"  
_horas_dia2 = 4
```

Los nombres de las variables son case sensitive, lo que quiere decir que la variable altura, será diferente de la variable Altura, o de la variable ALTURA

Debido a que Python emplea algunas palabras para ejecutar instrucciones de código, no todas las palabras pueden ser usadas para crear variables, y a esto se le conoce como palabras reservadas.



Visitar página

Encuentre la lista detallada en el siguiente enlace

https://www.w3schools.com/python/python_ref_keywords.asp



Ejercicios

Resuelve con Python estas operaciones matemáticas

1. $y = x^2 + 4x + 10$ donde $x = 5$
2. $y = 3x^2 + 5x - 50$ donde $x = -15$
3. $y = 2x^3 - 4x + 10 - 6 * 7$ donde $x = 35$
4. $y = ((10 + 30) ** 4) - (600 / 200)$
5. $y = x^m - 7k + 10$ donde $x = 5, m = 3, k = 8$

Un posible código para el ejercicio 1 sería el siguiente:

```
x = 5

y = x**2 + 4*x + 10

print("y =", y)

>> y = 55
```

Ahora continúa con los demás ejercicios.

Programación estructurada vs Programación orientada a objetos

Considera las siguientes líneas de código como un ejemplo de la programación estructurada o secuencial, donde el código se ejecutará línea tras línea

```
edad = 20

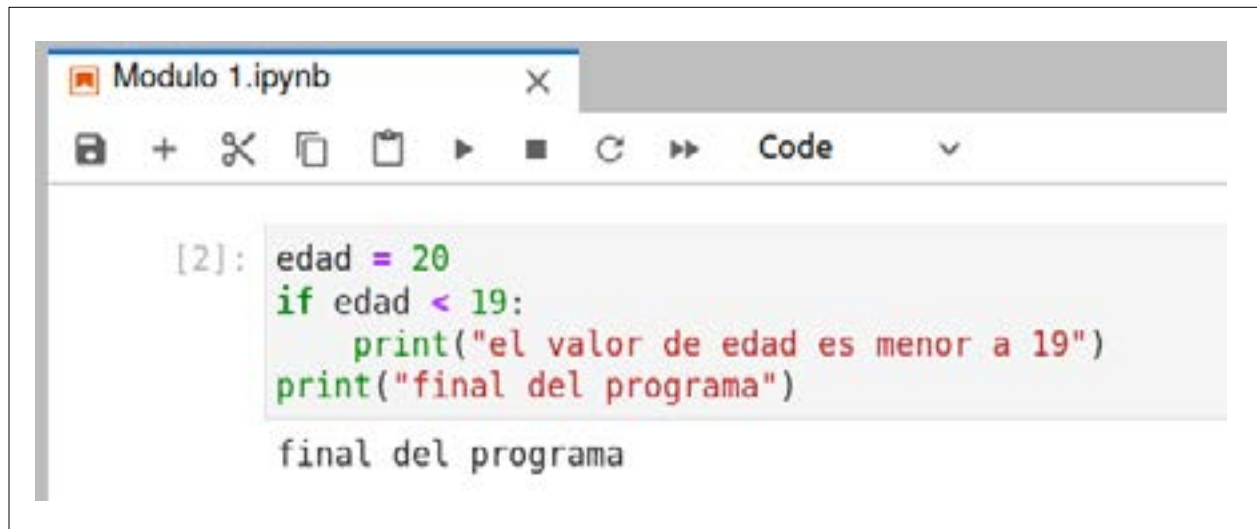
if edad < 19:

    print("el valor de edad es menor a 19")

print("final del programa")

>> final del programa
```

Figura 4



Fuente: propia

En este ejemplo, se ha definido una característica en forma de variable con la palabra edad, y se le ha asignado un valor numérico de tipo entero de 20. El resultado del programa será el mensaje "final del programa" puesto que la condición impuesta de un valor menor al número 19 no se cumple, pero esto lo estudiaremos más a fondo posteriormente. Por ahora considera que un programa estructurado o secuencial simplemente toma instrucciones de forma jerárquica y las va ejecutando una por una.

En el siguiente ejemplo, analiza un problema donde padre e hijo tienen cada uno un auto y se desea describir sus características como se muestra a continuación:

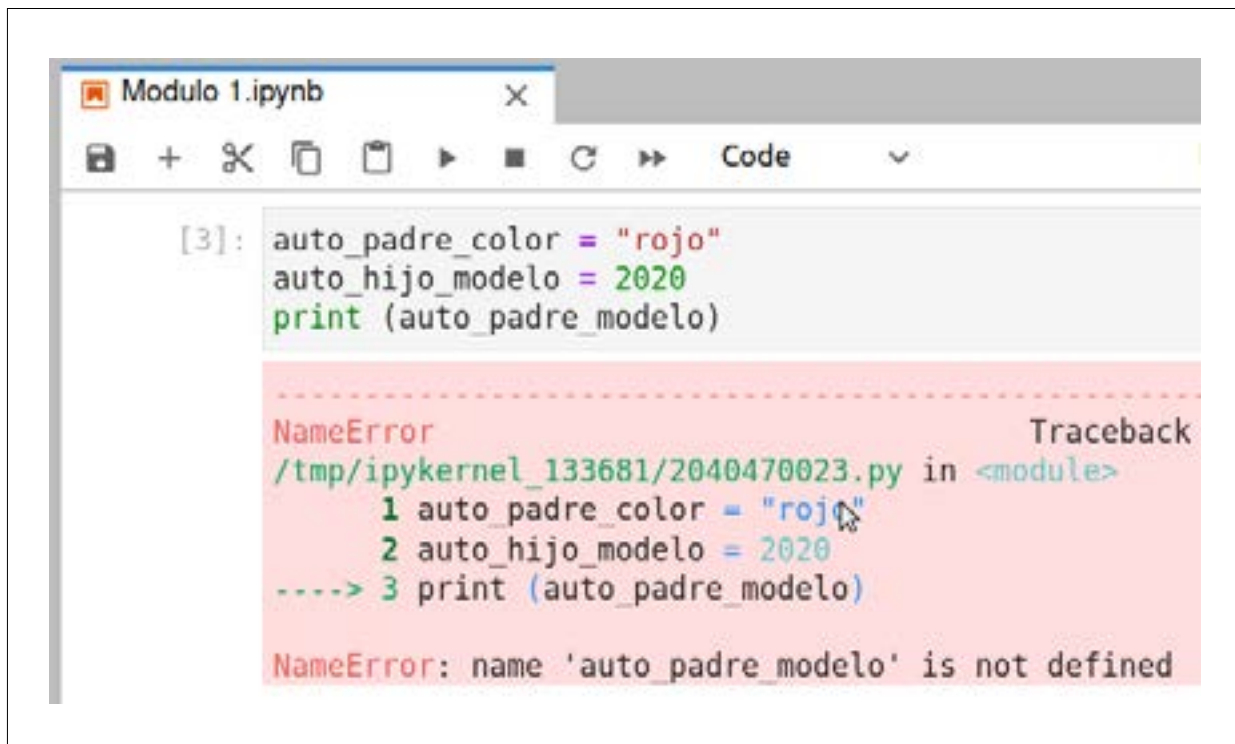
```
auto_padre_color = "rojo"

auto_hijo_modelo = 2020

print (auto_padre_modelo)

>> NameError: name 'auto_padre_modelo' is not defined
```

Figura 5



Fuente: propia

En este ejemplo, Python nos mostrará un error en el cual nos dice que la variable **auto_padre_modelo** no ha sido definida, y tiene razón!, puesto que aunque la variable ha sido definida para el **auto_hijo**, no lo ha sido para el auto padre y viceversa con la característica del color. ¿Te imaginas cuántas variables tendríamos que crear para describir las características de cada auto, no solo para este ejemplo, sino por ejemplo, para cada auto nuevo que ingrese a nuestro programa? Y si en algún momento se definiera una nueva característica para los autos, como un nuevo sistema de navegación, ¿Cuántas nuevas variables tendríamos que crear? Bueno, es aquí donde la programación orientada a objetos cobra sentido al ahorrarnos la creación de nuevas variables por medio de la capacidad de crear nuevos objetos a partir de una plantilla que se conoce con el nombre de Clase. De hecho, al crear variables donde definimos nombres, estamos creando nuevos objetos de tipo texto (**str**) y éstos están heredando todos sus atributos y métodos. Veamos un ejemplo:

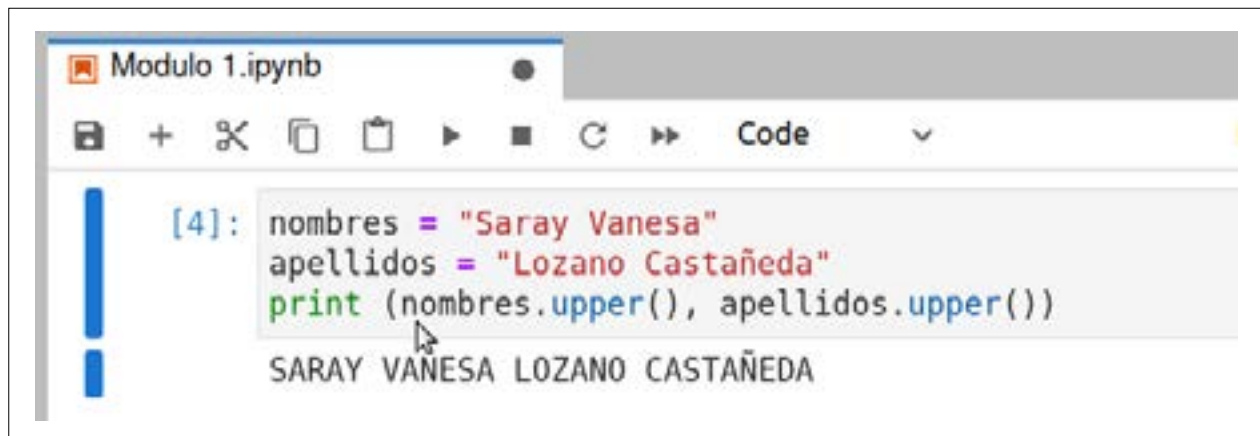
```
nombres = "Saray Vanesa"

apellidos = "Lozano Castañeda"

print ( nombres.upper( ), apellidos.upper( ) )

>> SARAY VANESA LOZANO CASTAÑEDA
```

Figura 6



The screenshot shows a Jupyter Notebook window titled 'Modulo 1.ipynb'. The code cell contains the following Python code:

```
[4]: nombres = "Saray Vanesa"
     apellidos = "Lozano Castañeda"
     print (nombres.upper(), apellidos.upper())
```

The output of the code cell is displayed below the code:

```
SARAY VANESA LOZANO CASTAÑEDA
```

Fuente: propia

Observa que al crear las variables nombres y apellidos, ambas tienen la capacidad de usar el método `.upper()` que transforma el texto en mayúsculas

En código, una "Clase" se crea de la siguiente manera (es muy importante que al digitar el siguiente ejemplo, se respeten los espacios, pues es una característica de Python denominada indentación)

```
class Auto: #definimos el nombre de la clase Auto

    """creamos los atributos de la clase, por defecto
    llevan la palabra reservada self"""

    def __init__(self, modelo, color):

        self.modelo = modelo

        self.color = color

    def ver_modelo(self):

        print(self.modelo)

    def ver_color(self):

        print(self.color)


#Creamos los objetos a partir de la clase Auto

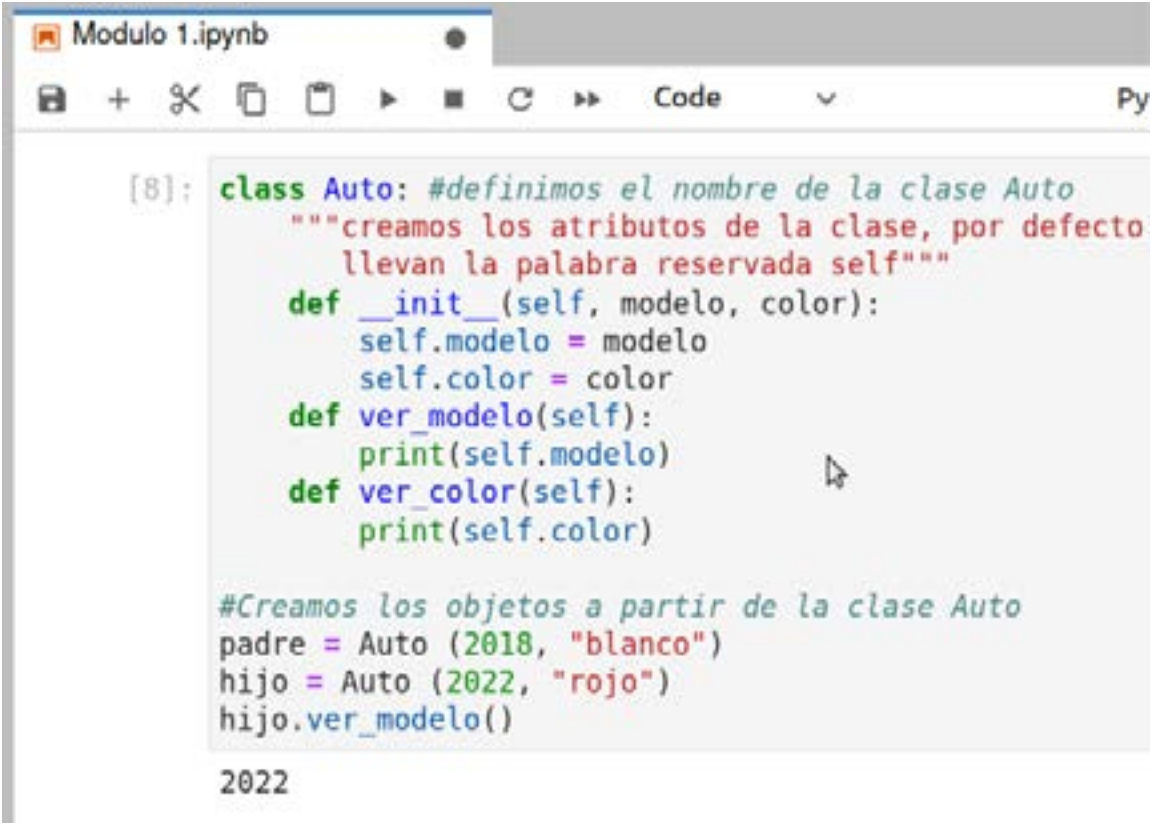
padre = Auto (2018, "blanco")

hijo = Auto (2022, "rojo")

hijo.ver_modelo()

>> 2022
```

Figura 7



```
[8]: class Auto: #definimos el nombre de la clase Auto
      """creamos los atributos de la clase, por defecto
      llevan la palabra reservada self"""
      def __init__(self, modelo, color):
          self.modelo = modelo
          self.color = color
      def ver_modelo(self):
          print(self.modelo)
      def ver_color(self):
          print(self.color)

      #Creamos los objetos a partir de la clase Auto
      padre = Auto (2018, "blanco")
      hijo = Auto (2022, "rojo")
      hijo.ver_modelo()

      2022
```

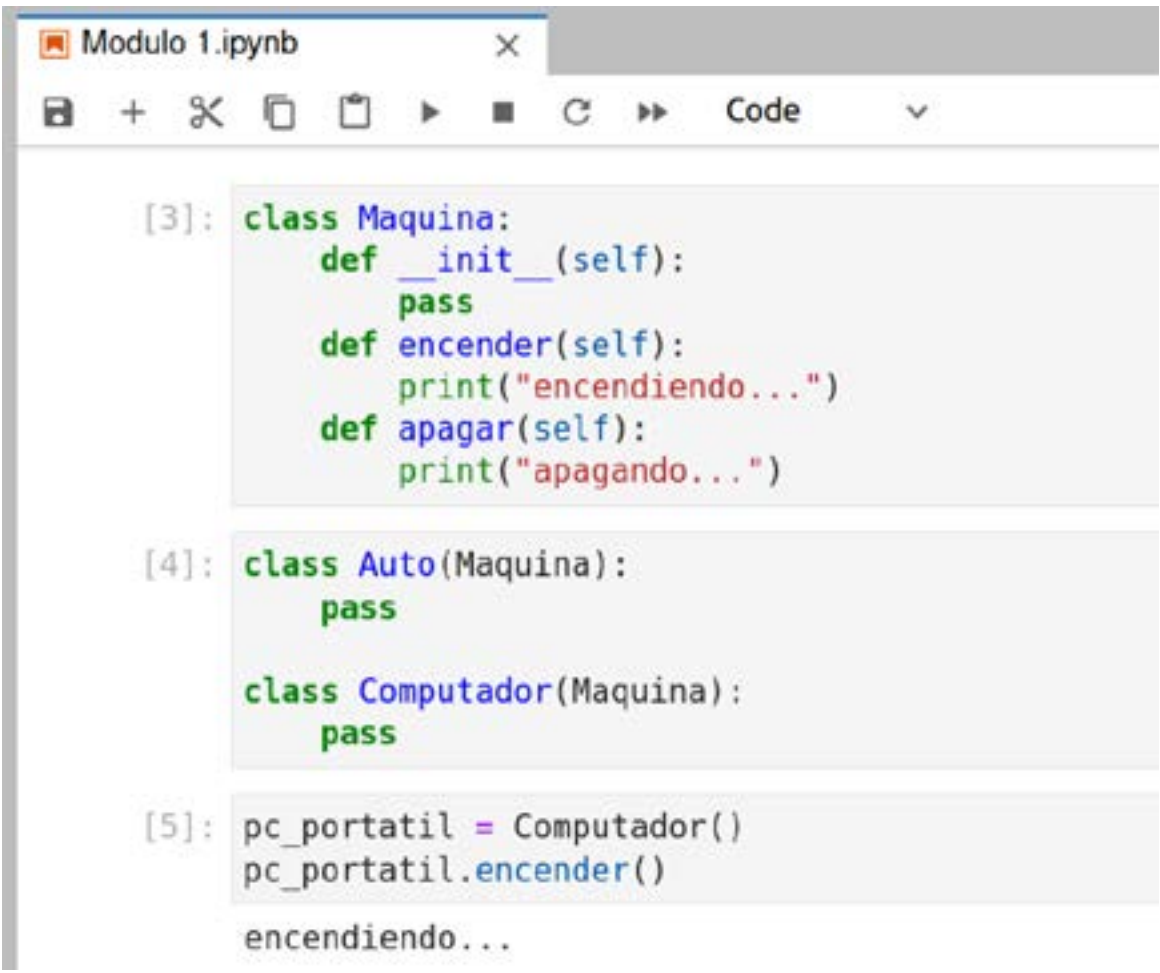
Fuente: propia

Observa que en este momento, la instrucción **hijo.ver_modelo()** da como resultado la impresión del número 2022, puesto que es un método que se definió en la clase Auto. Así mismo, si ahora se ejecutara la instrucción **padre.ver_modelo()**, el resultado debería ser el número 2018. En este código vemos que se ha creado la clase Auto con tres métodos: **__init__(self)** que es un método por defecto, un método que nos sirve para construir inicialmente los objetos de la clase y pasar atributos desde su creación, como sucede al decir que el auto del padre será de modelo 2018 y de color blanco. El método **ver_modelo(self)** que tiene como propósito mostrar como mensaje el valor contenido en el atributo modelo, y el método **ver_color(self)**. En la creación de clases y métodos, la palabra reservada **self** es obligatoria para que Python desarrolle los métodos correctamente. Como sugerencia, piensa en los métodos como acciones que debe realizar el objeto y de ahí que los nombres para mostrar información puedan ser palabras como mostrar o ver o imprimir.

Herencia

En programación, puedes crear clases de objetos que puedan heredar tanto los atributos como los métodos para no tener que definir nuevamente alguno de éstos. En el siguiente ejemplo verás la creación de una clase máquina y a partir de ésta, la creación de otras clases

Figura 8



```
[3]: class Maquina:
    def __init__(self):
        pass
    def encender(self):
        print("encendiendo...")
    def apagar(self):
        print("apagando...")

[4]: class Auto(Maquina):
    pass

    class Computador(Maquina):
        pass

[5]: pc_portatil = Computador()
pc_portatil.encender()

encendiendo...
```

Fuente: propia

En este ejemplo, se destacan algunas cosas:

1. La palabra reservada **pass** se usa para dejar parte del código sin definir, como sucede en el método **`__init__(self)`**.
2. Al heredar, se usa el nombre de la clase de la cual se va a heredar, como sucede en el caso de la clase **`class Computador (Maquina)`** donde le estamos programando a la clase **`class Computador`**, todos los atributos y métodos que tenga la clase **`class Maquina`**, y así la clase **`Computador`** hereda el método **`encender`** de **`Maquina`** y por eso al crear el objeto **`pc_portatil`**, éste puede hacer uso de ese método por medio de la instrucción **`pc_portatil.encender()`**

Conclusiones

Python es un lenguaje orientado a objetos, por lo cual debe entenderse que al programar, se debe pensar en que cada instrucción probablemente tenga algún tipo de comportamiento especial que puede ser reutilizado, como por ejemplo, la acción de encendido o apagado de una máquina como un auto, o como una estufa, una nevera, un televisor, entre otros, o el abrir y cerrar de una puerta o ventana. Estas acciones son conocidas como métodos, y las características que puedan tener, como el material, tamaño o color, son conocidos como atributos.

Las variables cumplen condiciones para ser declaradas o creadas y pueden contener cualquier tipo de dato existente en el lenguaje Python. Estas reglas son:

- a. Pueden comenzar por una letra (a-z) (A-Z)
- b. Pueden comenzar con un guión bajo. ejemplo: **`_nombre`**
- c. Son case sensitive, lo cual indica que **`manzana`** es diferente de **`Manzana`** o **`MANZANA`** y todas sus posibles combinaciones de mayúsculas y minúsculas
- d. No pueden emplearse las palabras reservadas de Python como variables



Lectura recomendada

Para finalizar, te invito a realizar la siguiente lectura complementaria:

[¿Qué hay de nuevo en Python?: I](#)

BIBLIOGRAFÍA

García, J. (2018). Ciencia De Datos. Técnicas Analíticas Y Aprendizaje Estadístico. Un Enfoque Práctico - Jesús García.pdf [6ng22yvvd2lv]. Idoc. pub. Recuperado de <https://idoc.pub/documents/idocpub-6ng22yvvd2lv>.

Python Tutorial. W3schools.com. (2022). Retrieved 31 August 2022, from <https://www.w3schools.com/python/default.asp>.