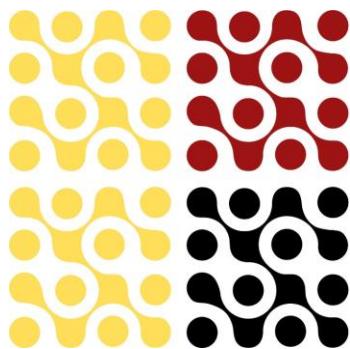


1

UNIDAD

DIPLOMATURA EN PYTHON APLICADO A LA CIENCIA DE DATOS

Elementos de Python



INSTITUTO
Data Science

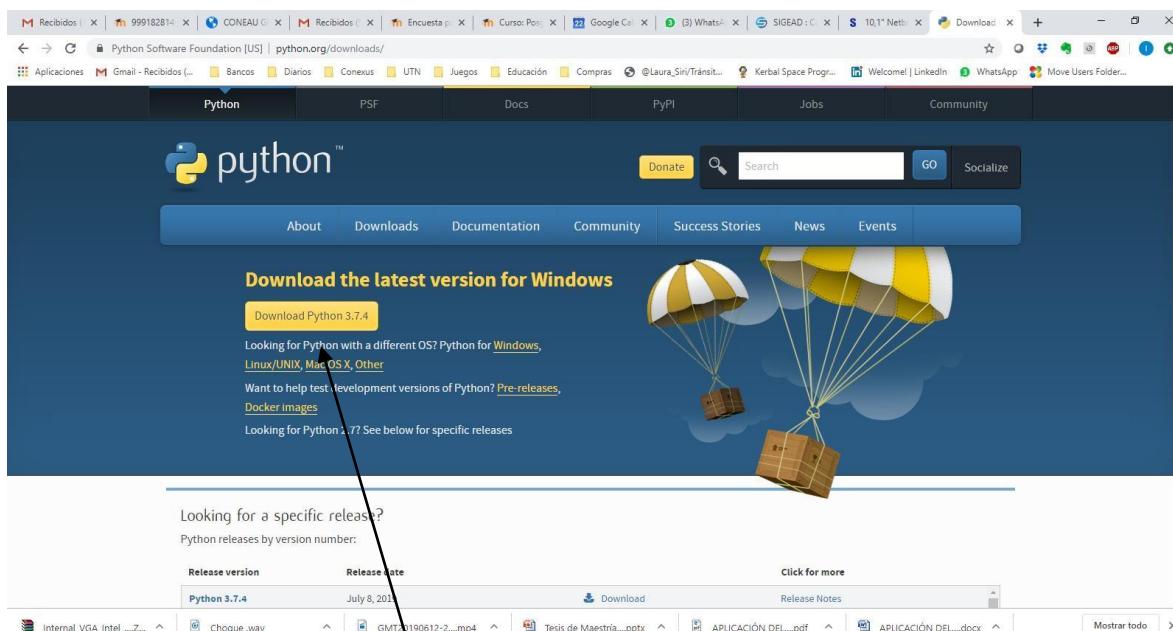
Unidad 1: Elementos de Python

Descarga e instalación

Python:

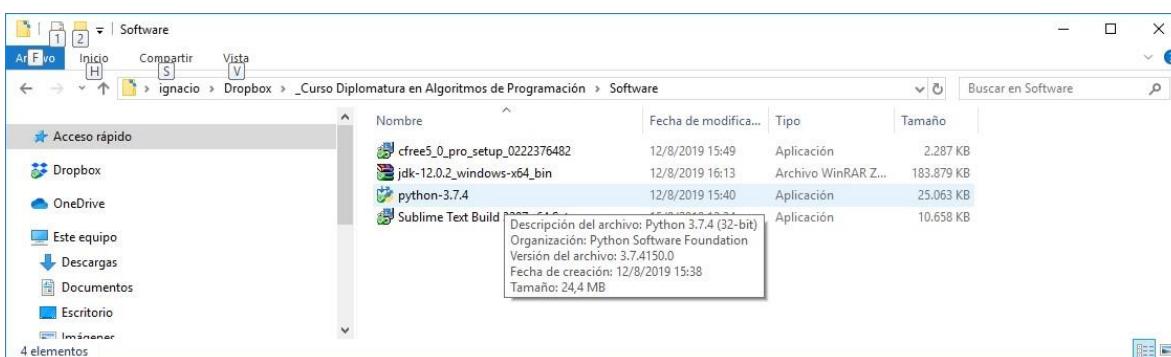
Vamos a empezar por descargar de Internet los archivos necesarios para instalar Python.
Vamos a:

<https://www.python.org/downloads/>



Y, por supuesto seleccionamos

Nos descargamos entonces:



Lo ejecutamos y nos aparece:

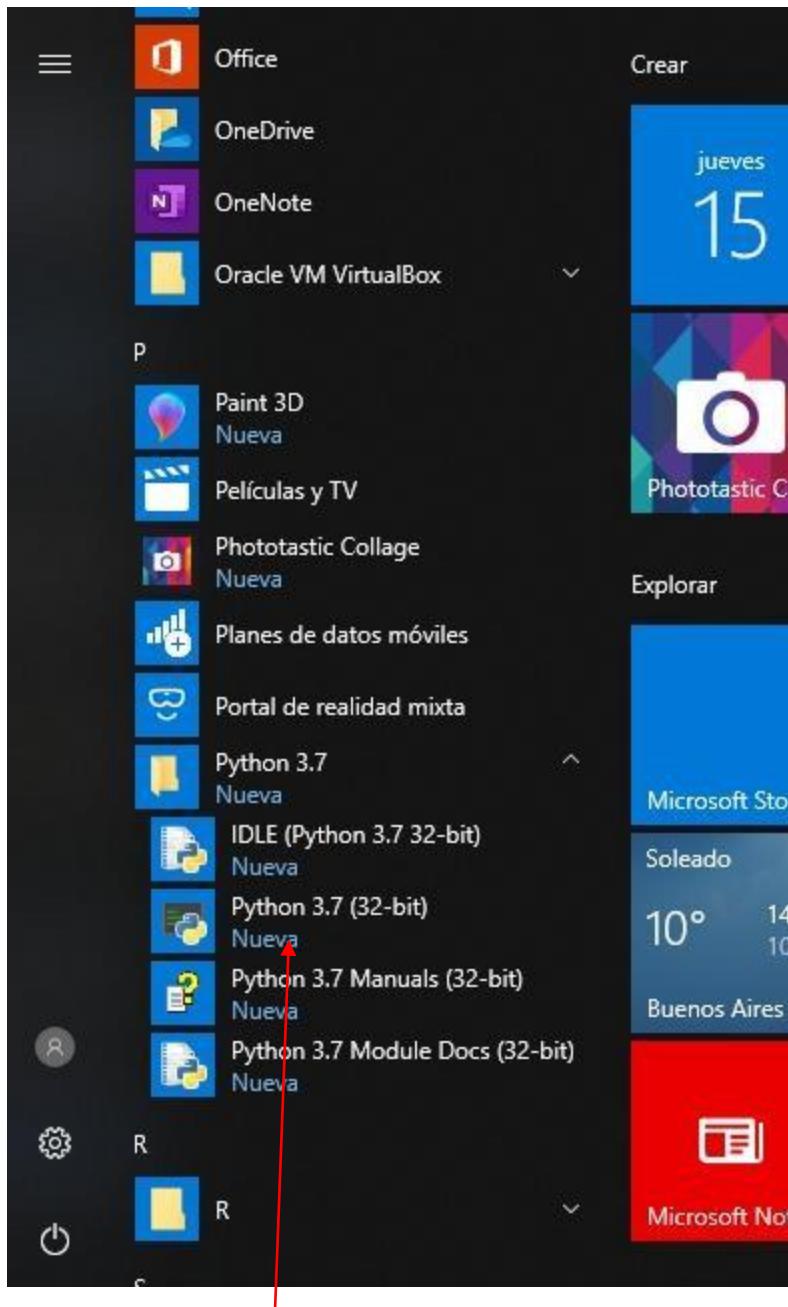


Tomamos la opción de instalar ahora y vamos respondiendo las preguntas que nos hace utilizando, en lo posible, las opciones por defecto.

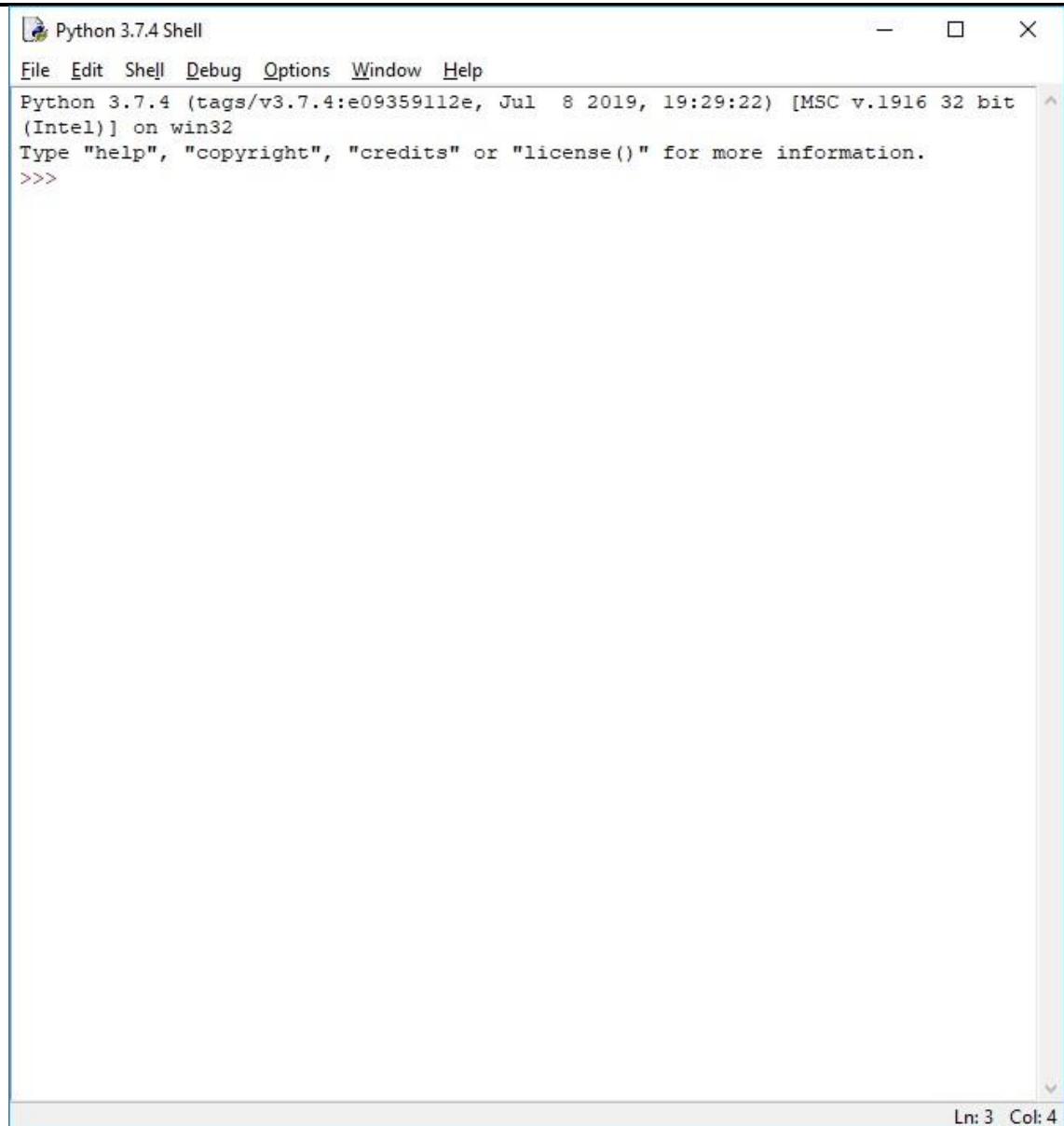
Al terminar nos encontramos con:



Entre las muchas cosas que fuimos instalando a lo largo del curso nos aparece ahora también:



Si seleccionamos Python 3.7 nos aparece:



The screenshot shows a window titled "Python 3.7.4 Shell". The menu bar includes "File", "Edit", "Shell", "Debug", "Options", "Window", and "Help". The main text area displays the Python 3.7.4 startup message: "Python 3.7.4 (tags/v3.7.4:e09359112e, Jul 8 2019, 19:29:22) [MSC v.1916 32 bit (Intel)] on win32", followed by "Type "help", "copyright", "credits" or "license()" for more information.", and a ">>>" prompt. A status bar at the bottom right indicates "Ln: 3 Col: 4".

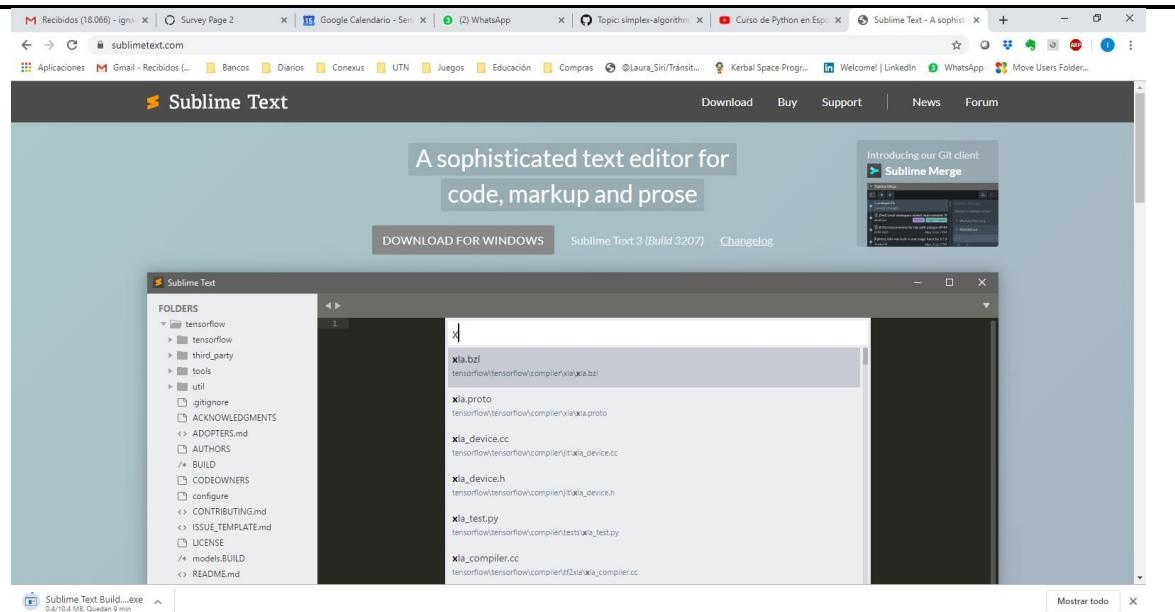
Ya podríamos empezar a trabajar.

Sin embargo, esta forma de trabajar no está exenta de incomodidades. Mejor incorporar un buen editor de código.

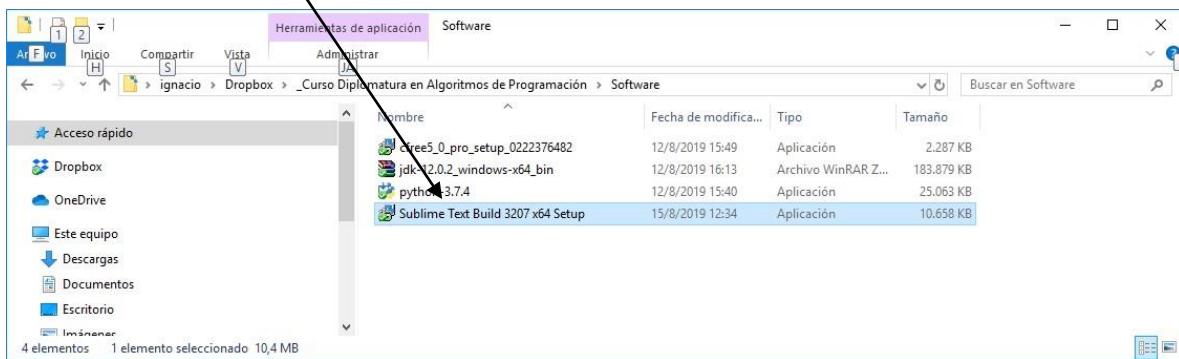
Vamos a descargar sublime, para eso vamos a:

www.sublimetext.com

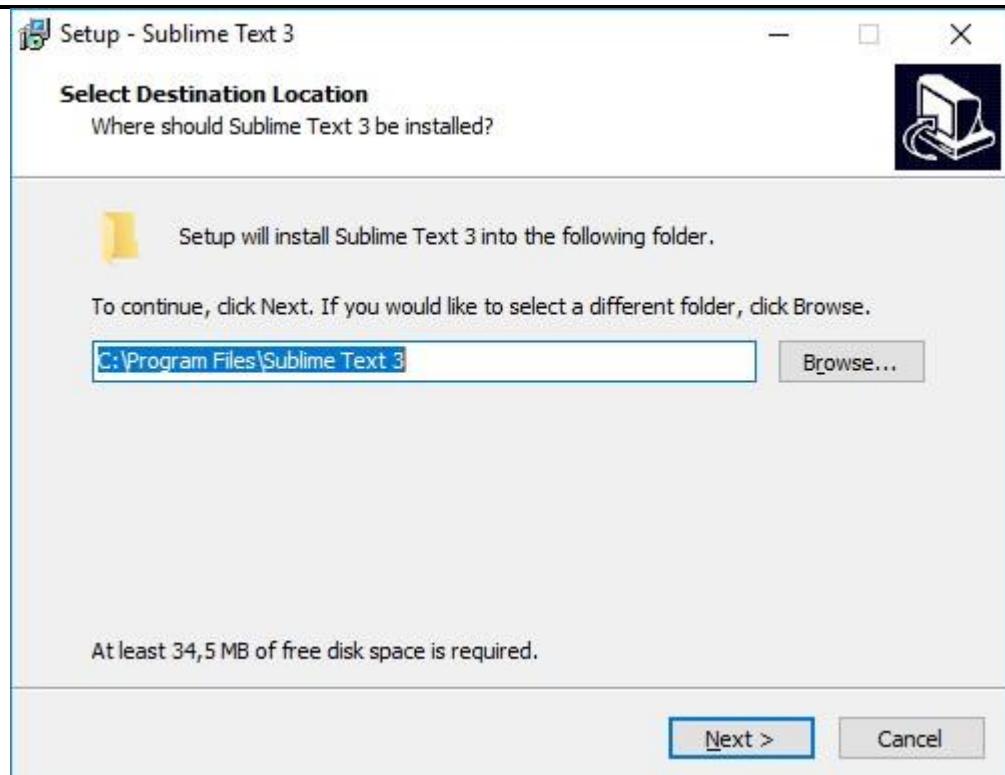
Y obtenemos:



Seleccionamos la opción de descargar para Windows (en mi caso, ustedes verán) para finalmente bajarnos:

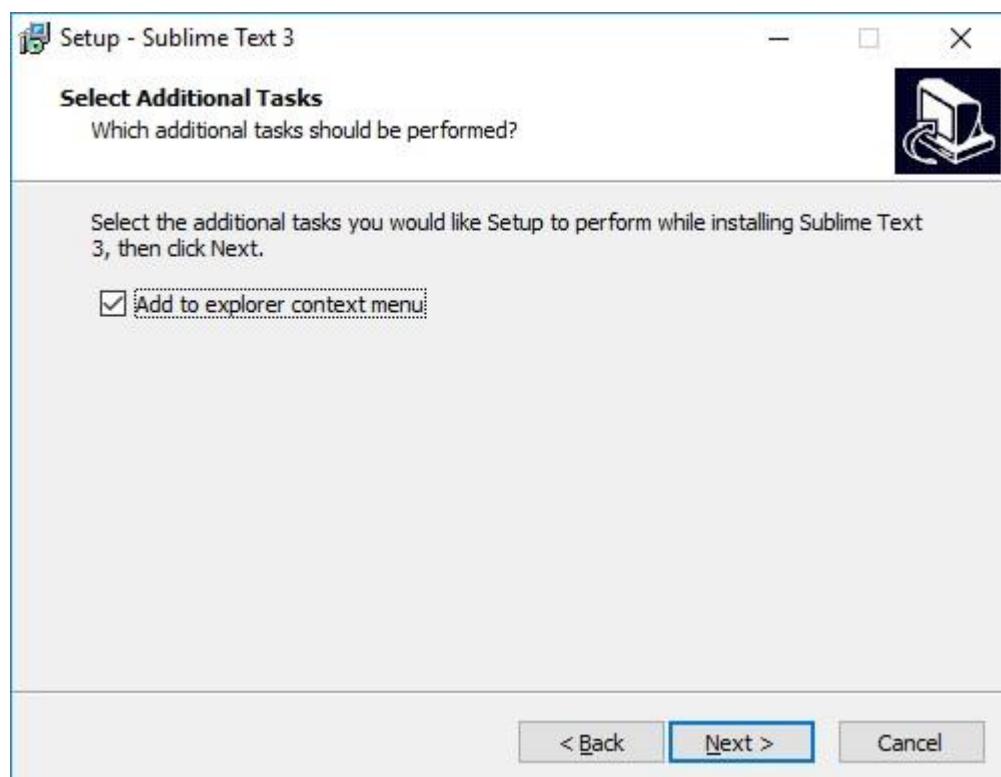


Lo ejecutamos:



Como siempre vamos tomando las opciones por defecto para avanzar:

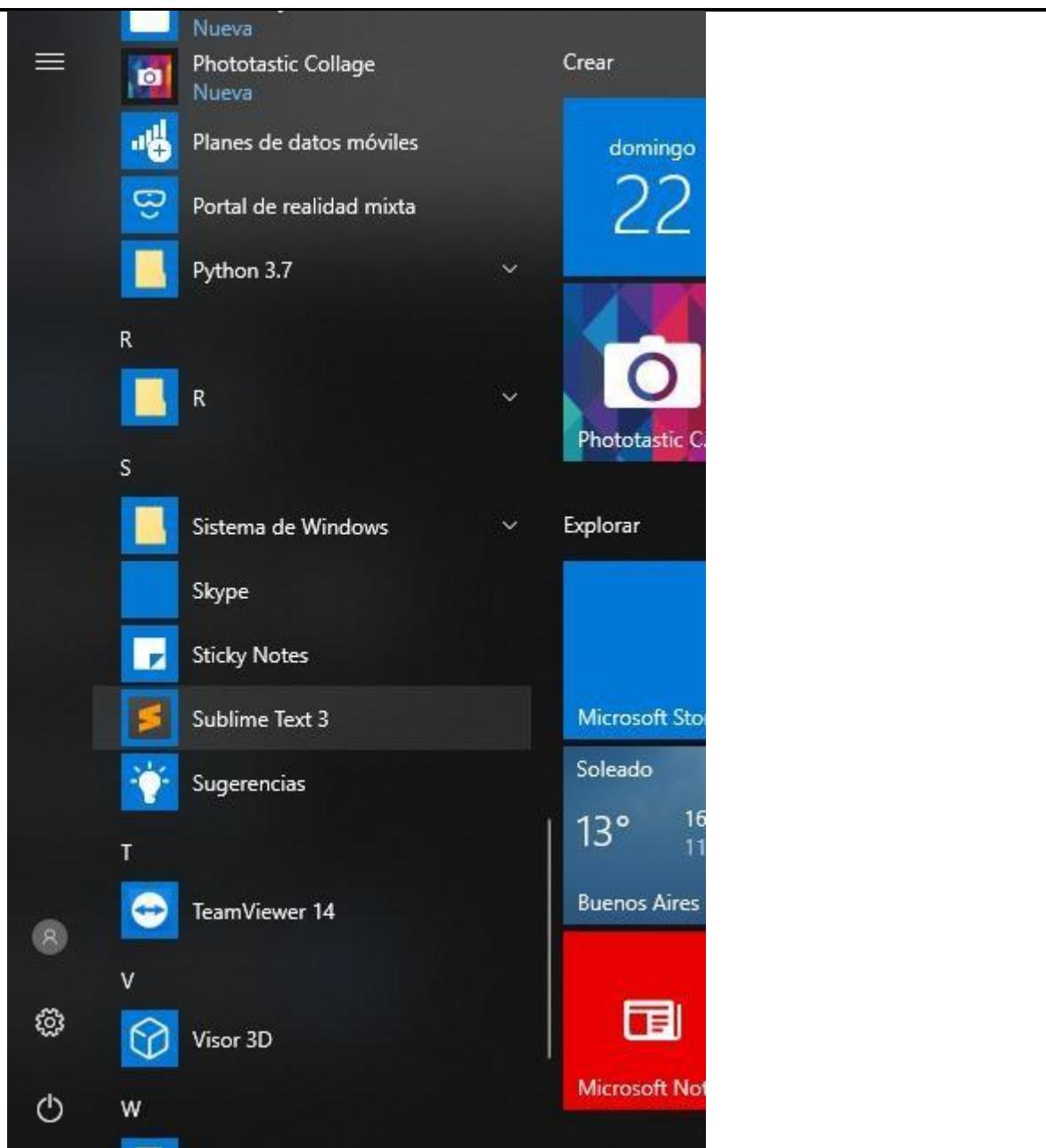
En particular no hay que olvidarse de seleccionar:



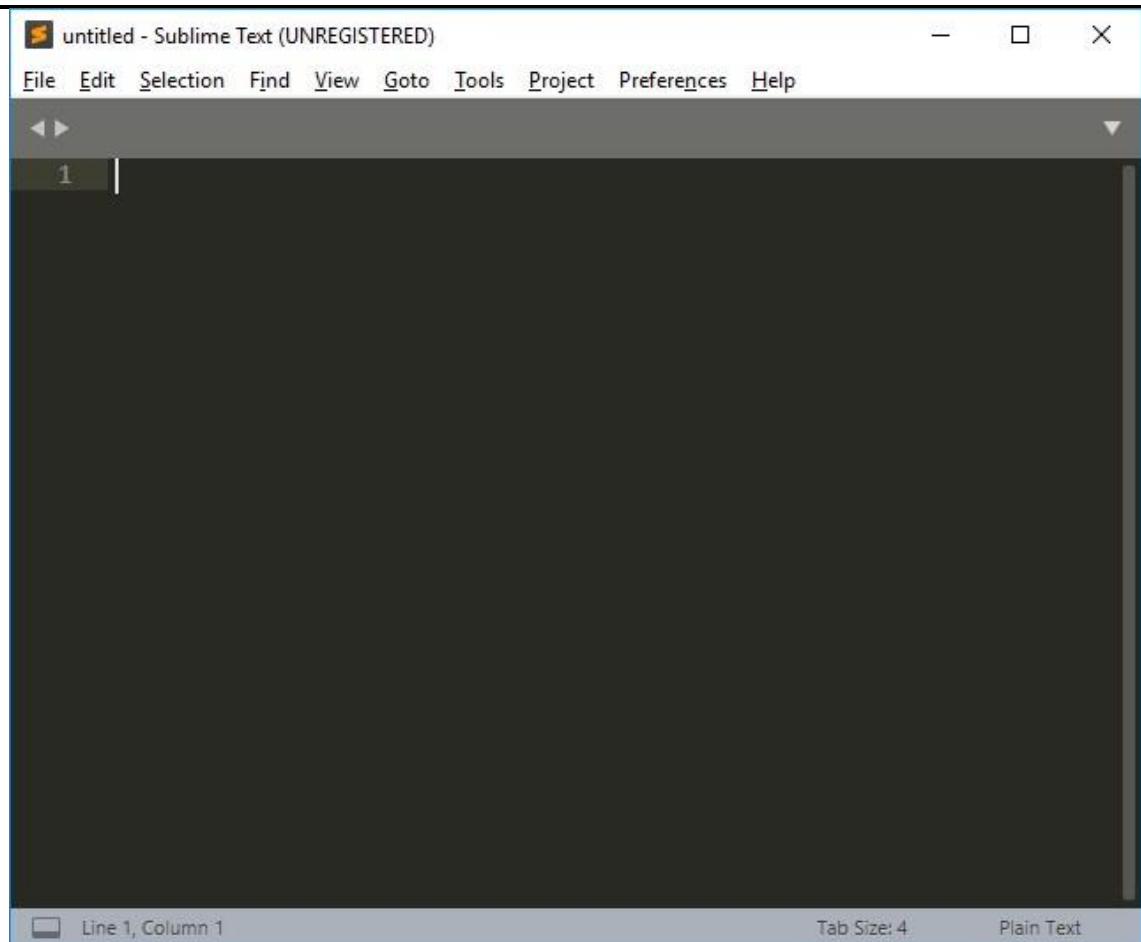
Hasta que llegamos a:



Vamos al menú de Windows donde encontramos nuestro nuevo amigo listo para usar:



Lo lanzamos y obtenemos:

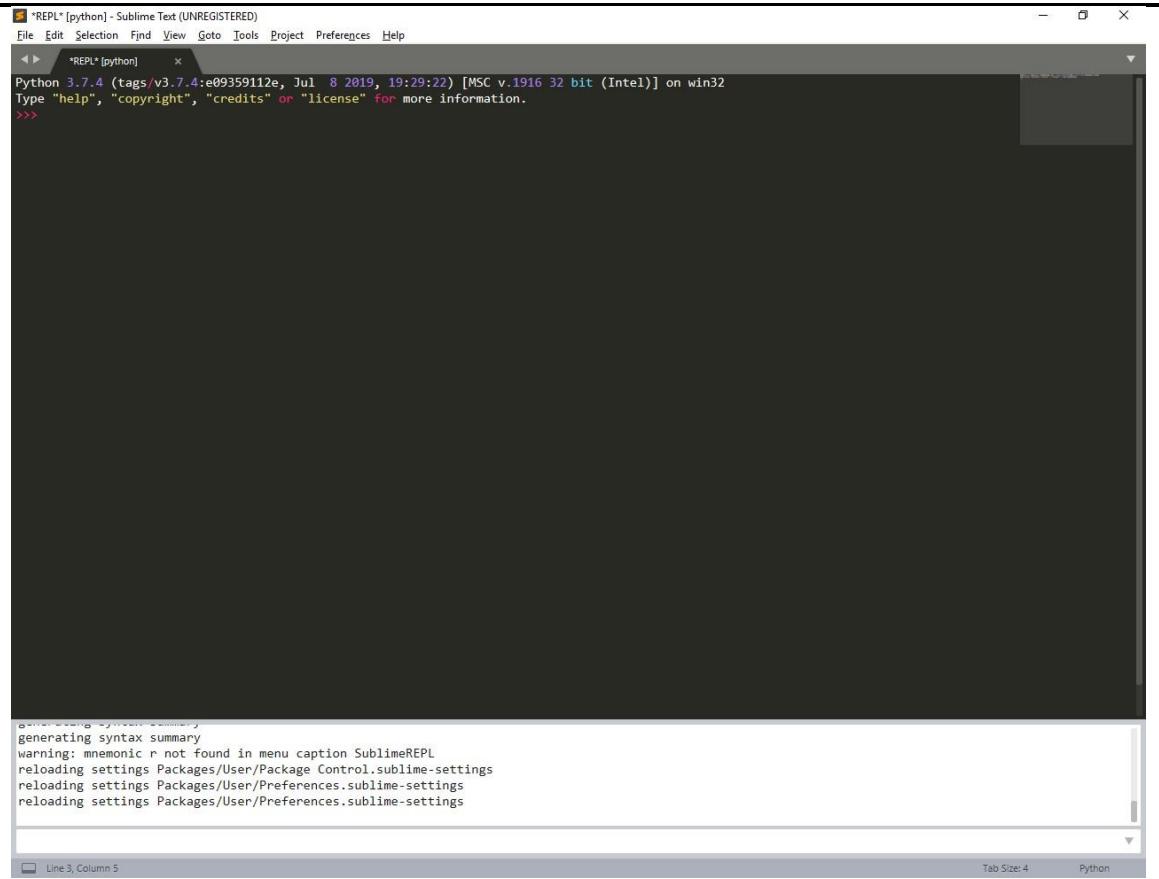


Ahora necesitamos agregarle algunos paquetes que harán nuestra vida más cómoda aún permitiendo la integración de SublimeText con la interface de Python (y otros lenguajes como nuestro viejo amigo R)

Para eso hacemos:

- Ctrl shift P
- Install package control
- Install sublimeREPL

Y obtenemos:



Ahora si estamos listos para arrancar con la aventura de aprender Python.

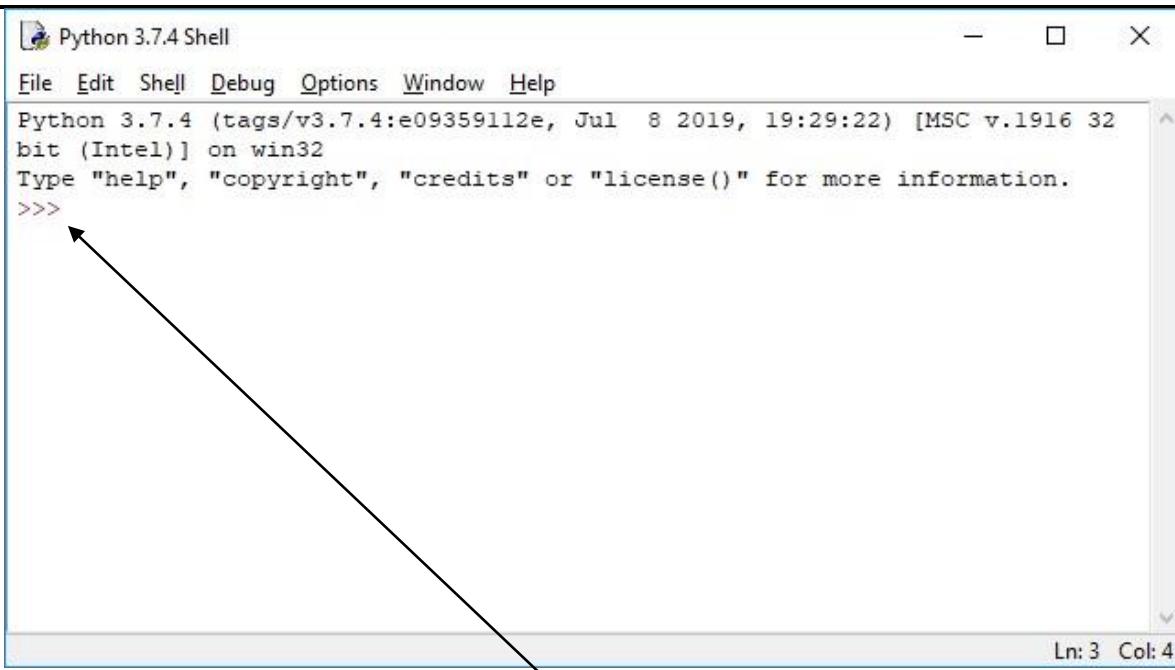
Una buena noticia es que Python es un buen lenguaje para empezar a programar los que hayan llegado hasta aquí sin experiencia previa en programación.

Cosas que vamos a ver:

- Sintaxis básica del lenguaje
- Algoritmos, listas y tramos
- Conexión con bases de datos.
- Gráficos
- Procesos y tareas
- Programación en red

Sintaxis básica:

Empezamos por abrir el IDLE:



Nos encontramos por primera vez con el prompt que nos informa donde vamos a introducir nuestros comandos.

Cuando entramos a un lugar es bueno, en primer lugar aprender a salir. Para salir de Python nos bastará con teclear **ctrl-q** y listo, ya estamos afuera.

Volvemos ahora a entrar para empezar a utilizarlo. La primera instrucción que vamos a aprender es **print**. Como se pueden imaginar **print** se encarga de sacar por pantalla el contenido que pongamos entre paréntesis.

Por ejemplo:



The screenshot shows a Python 3.7.4 Shell window. The title bar reads "Python 3.7.4 Shell". The menu bar includes File, Edit, Shell, Debug, Options, Window, and Help. The shell area displays the following text:

```
Python 3.7.4 (tags/v3.7.4:e09359112e, Jul  8 2019, 19:29:22) [MSC v.1916 32 bit  
(Intel)] on win32  
Type "help", "copyright", "credits" or "license()" for more information.  
>>> print("Hola curso")  
Hola curso  
>>> |
```

In the bottom right corner of the shell window, there is a status bar with "Ln: 5 Col: 4".

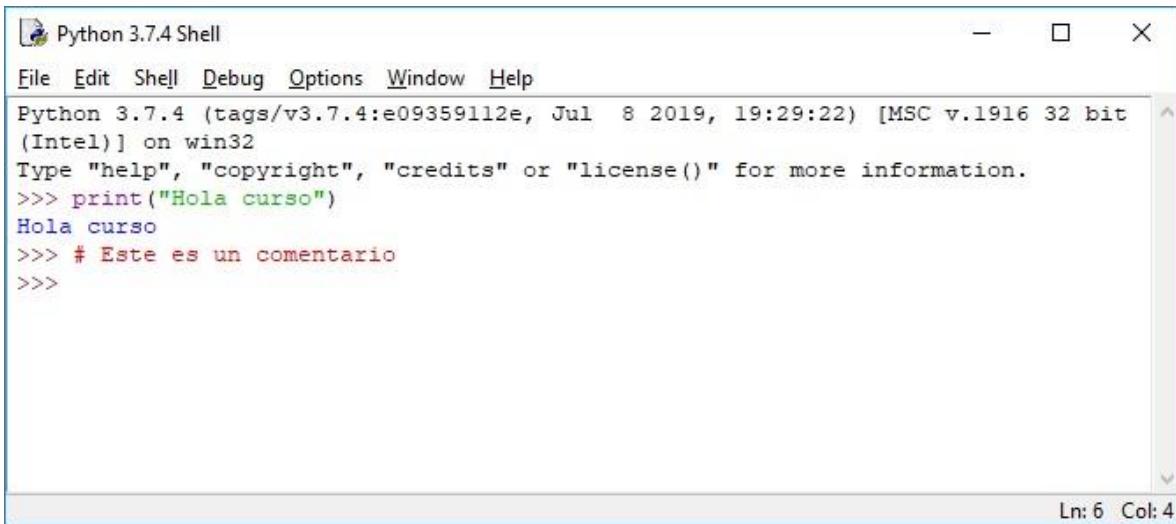
Conviene notar que los finales de instrucción no llevan punto y coma ni ningún otro indicador de fin de instrucción.

Se pueden escribir dos instrucciones en una misma línea separándolas por punto y coma pero se desaconseja hacerlo porque van en detrimento de la legibilidad del código.

Y, hablando de legibilidad del código, algo que ayuda mucho a leer el código ajeno, o incluso el propio después de cierto tiempo, es la introducción de comentarios.

Para que Python interprete una línea como un comentario en vez de una instrucción debemos hacer que esa línea comience con #

Por ejemplo:



The screenshot shows a Python 3.7.4 Shell window. The code entered is:

```

Python 3.7.4 (tags/v3.7.4:e09359112e, Jul  8 2019, 19:29:22) [MSC v.1916 32 bit
(Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> print("Hola curso")
Hola curso
>>> # Este es un comentario
>>>

```

The window has a status bar at the bottom right indicating "Ln: 6 Col: 4".

Y vamos notando con delicia como utiliza diferentes colores para que sepamos como interpretar el código:

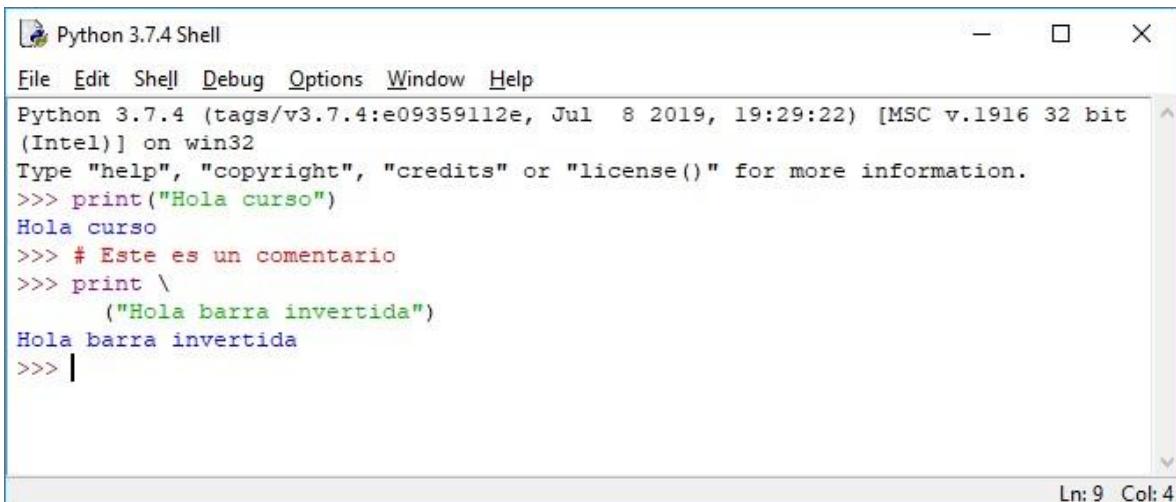
- Las instrucciones, como print en violeta
- Los comentarios en rojo
- Las constantes de texto en verde

Sigamos descubriendo Python.

También podemos separar una línea de instrucciones para que continúe en otro renglón si sentimos que con eso ganamos en claridad. La forma de hacerlo es con la barra invertida:

\

Por ejemplo:



The screenshot shows a Python 3.7.4 Shell window. The code entered is:

```

Python 3.7.4 (tags/v3.7.4:e09359112e, Jul  8 2019, 19:29:22) [MSC v.1916 32 bit
(Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> print("Hola curso")
Hola curso
>>> # Este es un comentario
>>> print \
    ("Hola barra invertida")
Hola barra invertida
>>> |

```

The window has a status bar at the bottom right indicating "Ln: 9 Col: 4".

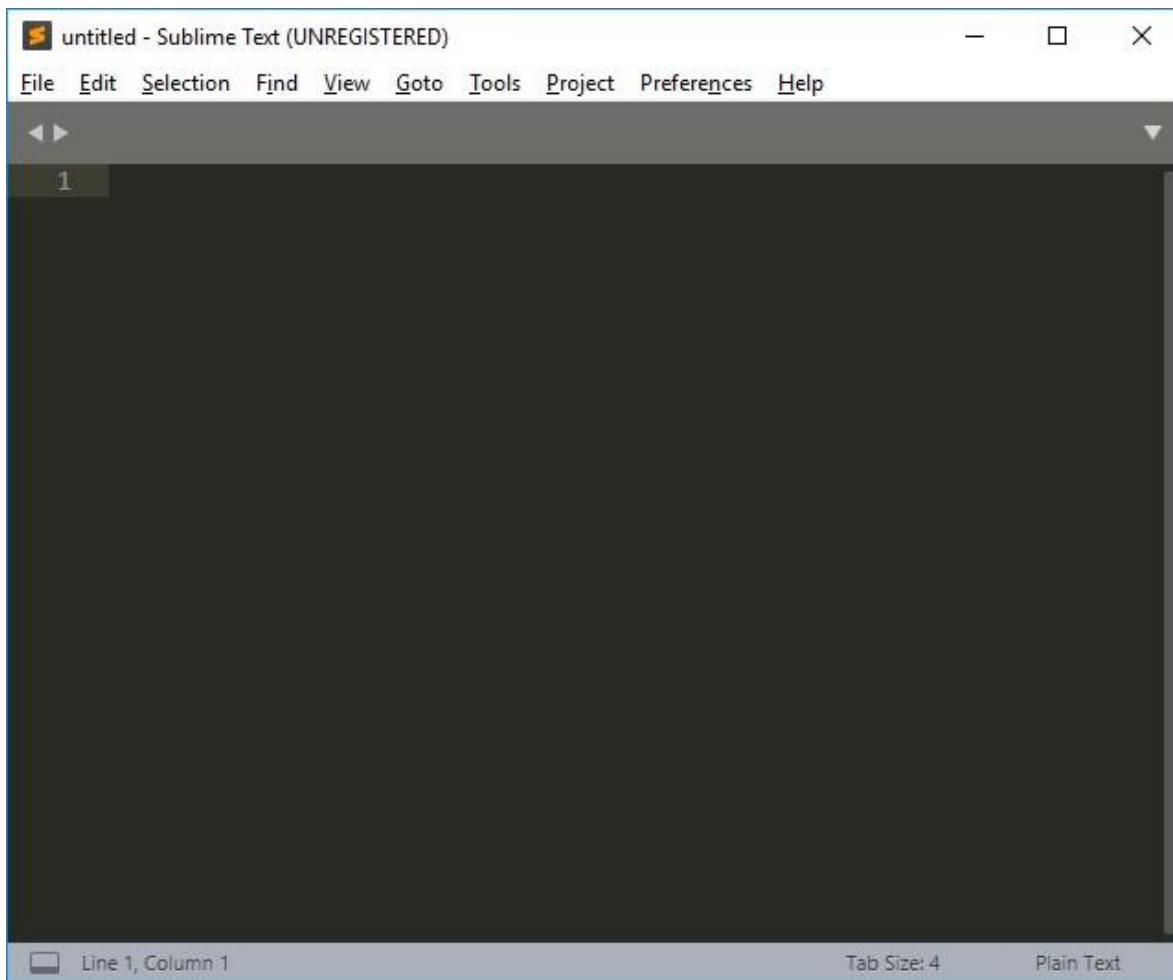
La instrucción aparece en dos renglones. El primer renglón termina con la barra invertida. Para facilitar la lectura Python nos agrega espacios en blanco en la línea que es continuación de la anterior.

El resultado al ejecutar es el mismo pues imprime “Hola barra invertida”

El concepto de indentación, que acabamos de ver en acción es muy importante en Python pues, a diferencia de otros lenguajes es mandatorio, es decir, es obligatorio para que las instrucciones funcionen. Esto puede parecer una traba, pero, a la larga, es beneficioso para la comprensibilidad del código.

Ya veremos que SublimeText se encargará de ayudarnos a indentar automáticamente por lo que no representará esfuerzo alguno para nosotros y si, puro beneficio.

Y ya que lo mencionamos empecemos a usar SublimeText:

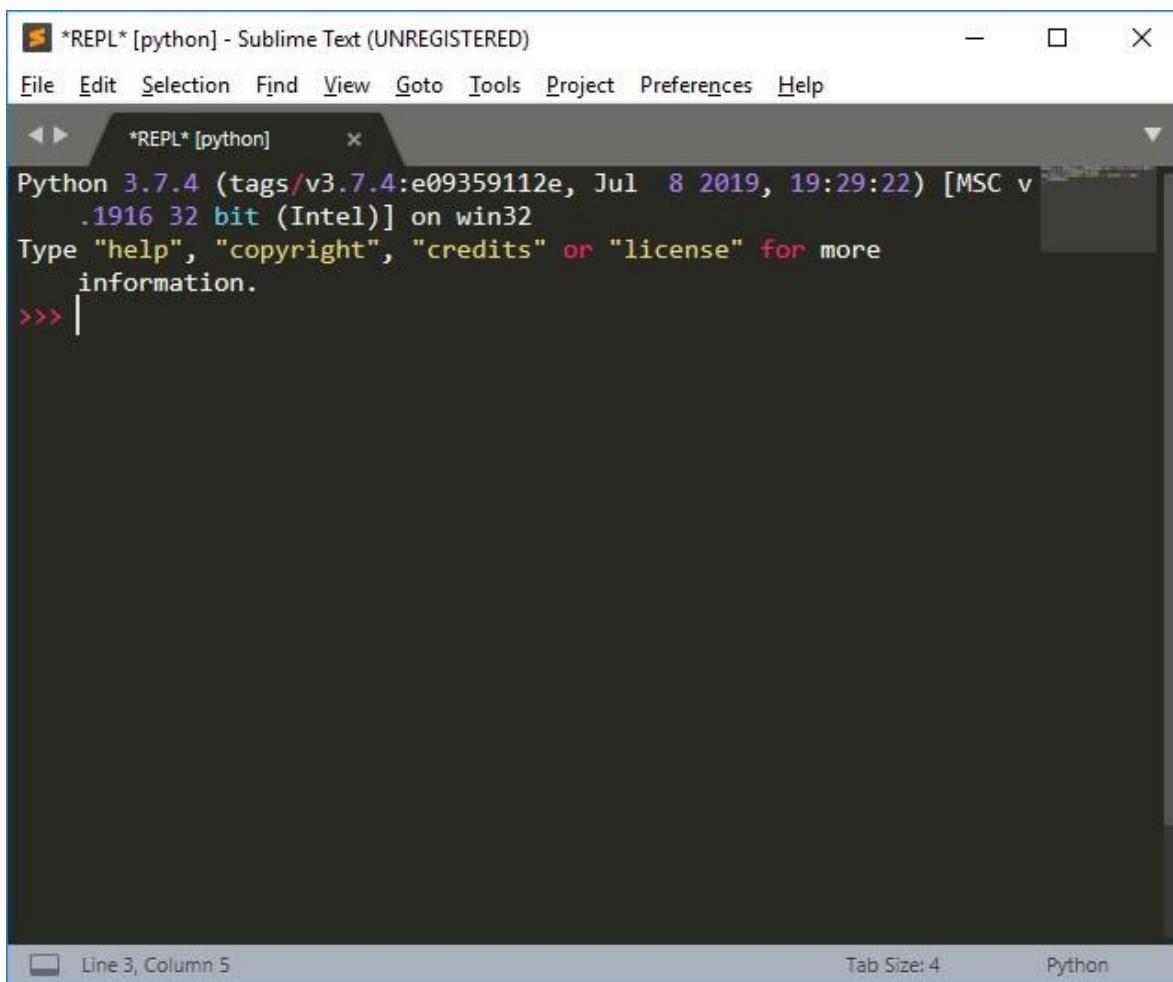


Así SublimeText trabaja como un editor de uso general. Si queremos combinar la magia de SublimeText con la consola de Python vamos al menú y seleccionamos las opciones:

Tools > SublimeREPL > Python > Python

MBA Ignacio Urteaga

Y:



The screenshot shows a Sublime Text window titled "*REPL* [python] - Sublime Text (UNREGISTERED)". The menu bar includes File, Edit, Selection, Find, View, Goto, Tools, Project, Preferences, and Help. The status bar at the bottom indicates "Line 3, Column 5" and "Tab Size: 4". The main area displays the Python 3.7.4 startup message and a command prompt line starting with ">>>".

```
Python 3.7.4 (tags/v3.7.4:e09359112e, Jul  8 2019, 19:29:22) [MSC v.1916 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more
information.
>>> |
```

Ya tenemos ambas herramientas combinadas.

Probamos lo que ya hicimos y notamos que el código de colores es distinto del que nos brindaba la consola.



The screenshot shows a Sublime Text window titled '*REPL* [python] - Sublime Text (UNREGISTERED)'. The menu bar includes File, Edit, Selection, Find, View, Goto, Tools, Project, Preferences, and Help. Below the menu is a toolbar with back, forward, and close buttons. The main area displays a Python REPL session:

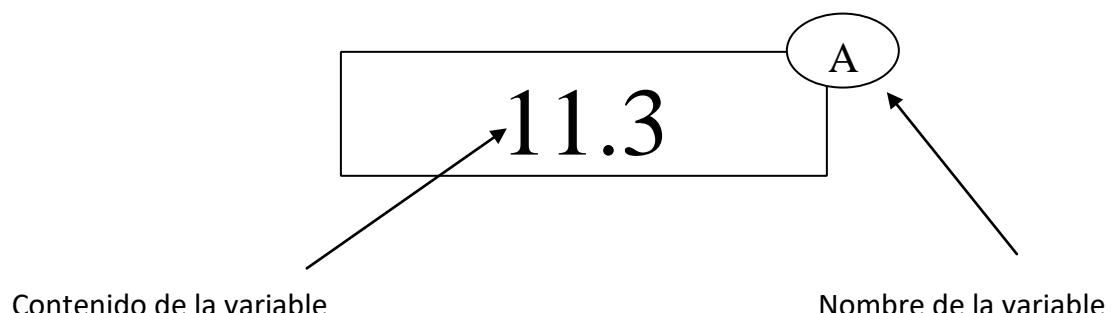
```
Python 3.7.4 (tags/v3.7.4:e09359112e, Jul  8 2019, 19:29:22) [MSC v.1916 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more
information.
>>> print("Hola SublimeText")
Hola SublimeText
>>> |
```

The status bar at the bottom indicates Line 5, Column 5, Tab Size: 4, and Python.

Variables y tipos de datos

El concepto de variable recorre todos los lenguajes de programación. Podemos pensar en una variable como una caja, que tiene una etiqueta que es el nombre de la variable. Dentro de la caja podemos guardar un valor. Nuestra caja también tiene una forma definida que nos limita el tipo de información que podemos guardar dentro de la caja.

Vamos a ponerlo en un esquema:

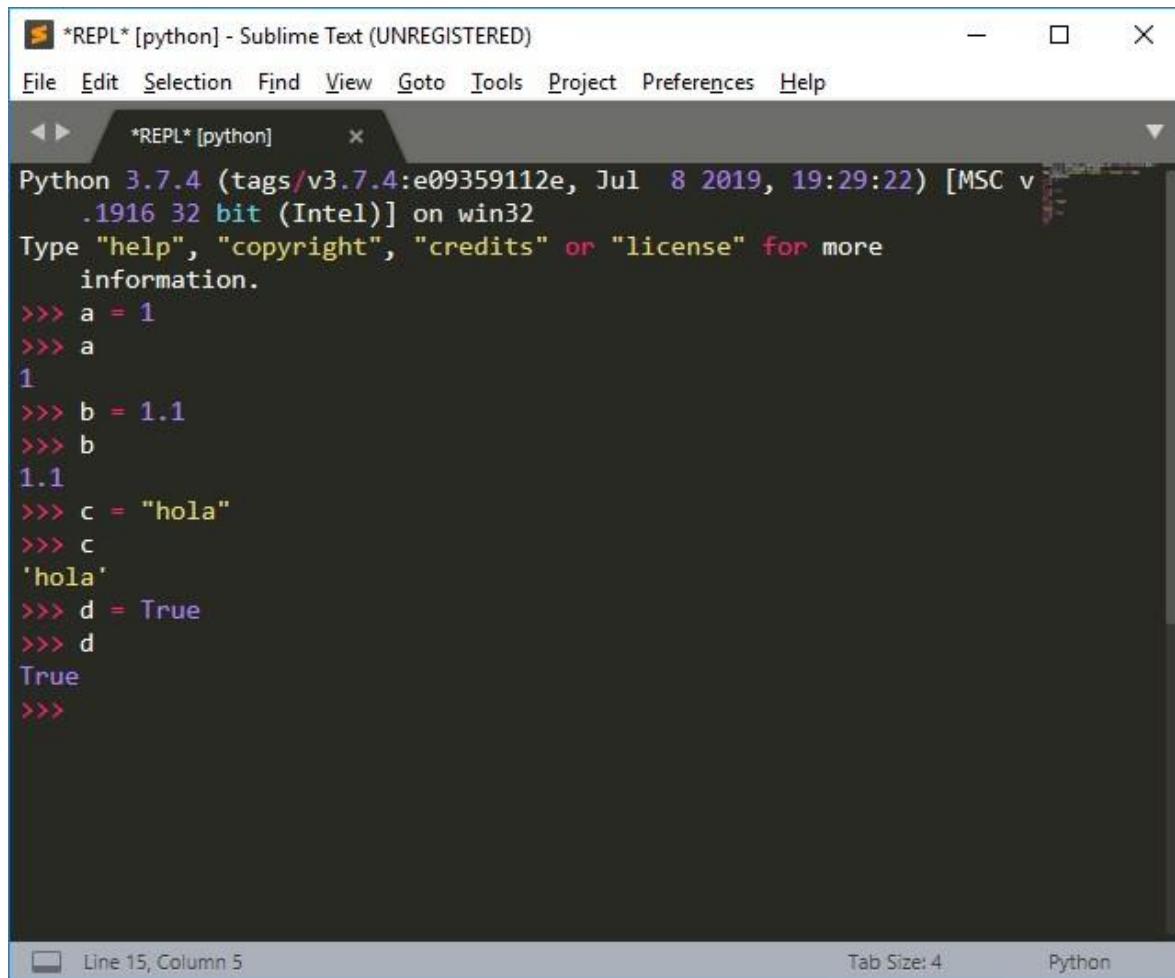


Esta variable guarda un número que tiene punto decimal: 11.3

Dentro de Python vamos a poder tener distintos tipos de variables. Para empezar vamos a utilizar:

- Numéricos
 - Enteros (int)
 - Flotantes (float)
 - Complejos
- Textos
- Lógicas

En Python no necesitamos declarar la variable antes de usarlas. Veamos ejemplos de cada una:



The screenshot shows a Sublime Text window titled '*REPL* [python] - Sublime Text (UNREGISTERED)'. The menu bar includes File, Edit, Selection, Find, View, Goto, Tools, Project, Preferences, and Help. The main area displays a Python REPL session:

```
Python 3.7.4 (tags/v3.7.4:e09359112e, Jul  8 2019, 19:29:22) [MSC v.1916 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.

>>> a = 1
>>> a
1
>>> b = 1.1
>>> b
1.1
>>> c = "hola"
>>> c
'hola'
>>> d = True
>>> d
True
>>>
```

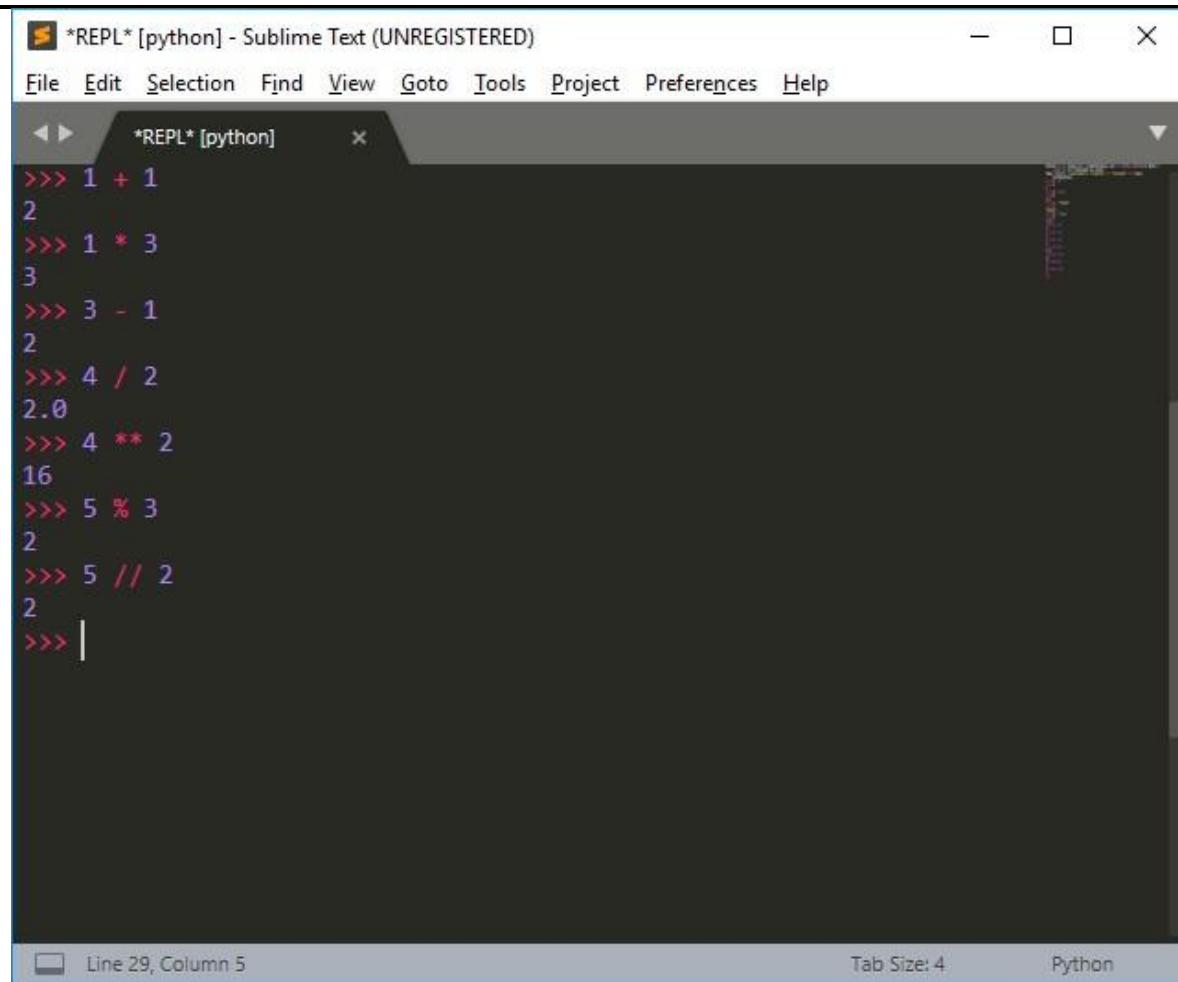
At the bottom of the window, status bars indicate 'Line 15, Column 5' and 'Tab Size: 4'.

Una vez que aprendimos a generar variables tenemos que aprender a hacer algo con ellas. Para eso vamos a necesitar los operadores:

- Aritméticos
 - Más +
 - Menos -
 - Multiplicación

-
- * División / Módulo %
 - Potencia ** División entera //
 - Comparación Igual que == Diferente de != Mayor que > Menor que < Mayor o igual que >= Menor o igual que <=
 - Lógicos
 - AND
 - OR
 - NOT
 - Asignación =
 - Especiales IS IS NOT IN NOT IN

Vamos a hacer algunos ejemplos para estar seguros de entendernos:

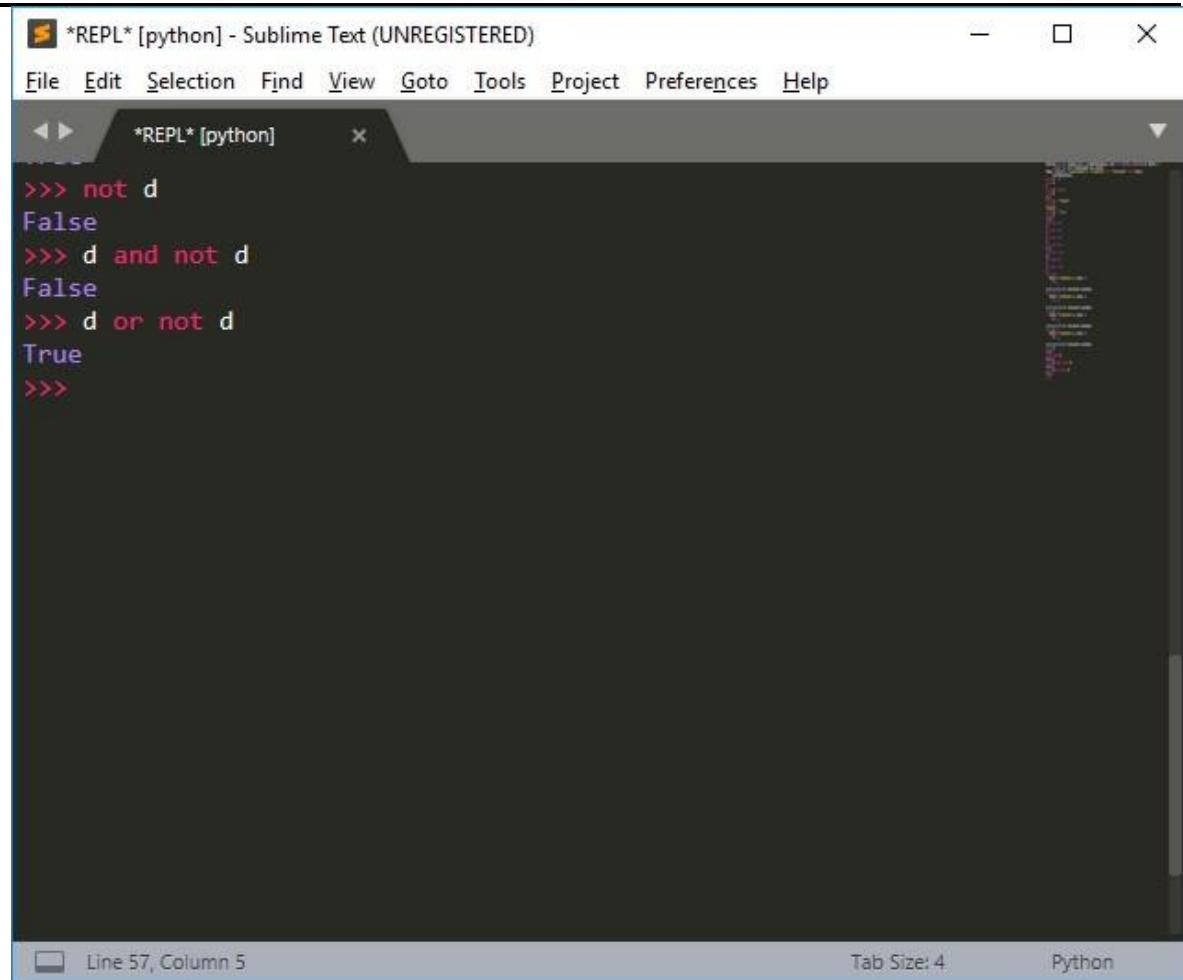


The screenshot shows a Sublime Text window titled '*REPL* [python] - Sublime Text (UNREGISTERED)'. The menu bar includes File, Edit, Selection, Find, View, Goto, Tools, Project, Preferences, and Help. The main pane displays a Python REPL session:

```
>>> 1 + 1
2
>>> 1 * 3
3
>>> 3 - 1
2
>>> 4 / 2
2.0
>>> 4 ** 2
16
>>> 5 % 3
2
>>> 5 // 2
2
>>> |
```

The status bar at the bottom left shows 'Line 29, Column 5'. The status bar at the bottom right shows 'Tab Size: 4' and 'Python'.

Esto es lo fácil. Vamos a probar algo más interesante con los operadores lógicos:



The screenshot shows a Sublime Text window titled '*REPL* [python] - Sublime Text (UNREGISTERED)'. The menu bar includes File, Edit, Selection, Find, View, Goto, Tools, Project, Preferences, and Help. The main area displays a Python REPL session:

```
>>> not d
False
>>> d and not d
False
>>> d or not d
True
>>>
```

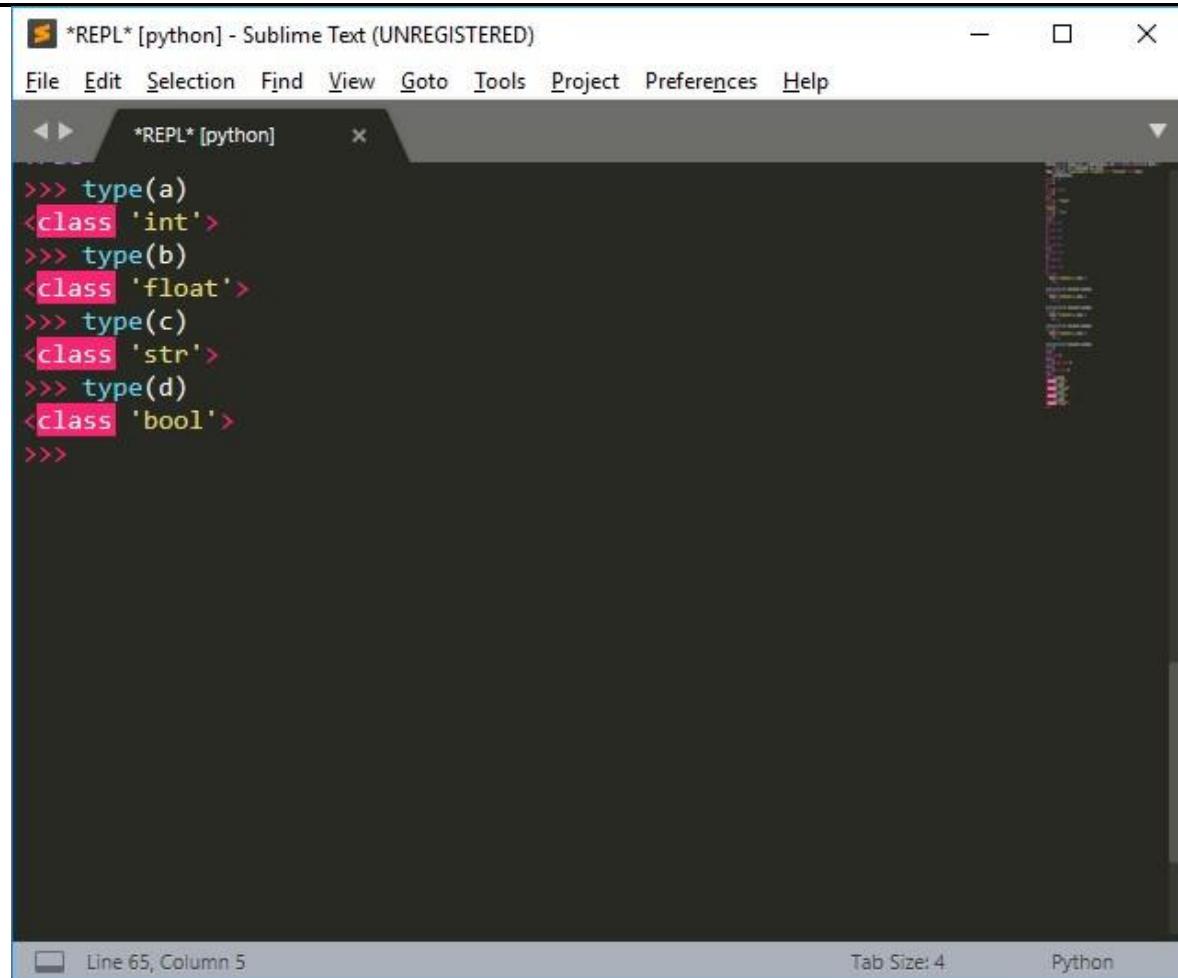
The status bar at the bottom indicates 'Line 57, Column 5' and 'Tab Size: 4'. A small Python logo icon is also present.

La asignación ya la usamos. Sirve para reemplazar el contenido de una variable por un nuevo valor si la variable ya existe o para crearla si no existe aún.

Para jugar con los operadores especiales vamos a necesitar algunos conceptos que no tenemos todavía.

Vamos a jugar ahora con nuestra primera función. Una función podemos visualizarla como una máquina a la que le vamos a dar unas entradas (también llamadas parámetros) para obtener una salida (que también se conoce como resultado)

La función que vamos a ver es la función Type y tiene por misión informarnos de que tipo es una variable.



The screenshot shows a Sublime Text window titled '*REPL* [python] - Sublime Text (UNREGISTERED)'. The menu bar includes File, Edit, Selection, Find, View, Goto, Tools, Project, Preferences, and Help. The main area displays a Python REPL session:

```
<stdin> >>> type(a)
<class 'int'>
<stdin> >>> type(b)
<class 'float'>
<stdin> >>> type(c)
<class 'str'>
<stdin> >>> type(d)
<class 'bool'>
<stdin> >>>
```

The status bar at the bottom indicates Line 65, Column 5, Tab Size: 4, and Python.

Vemos que para a nos dice class int. ¿Qué es esto de clase? Nos está informando que, desde la perspectiva de Python a es algo más que lo que otro lenguaje consideraría una simple variable, es un objeto.

Necesitamos entonces entender que cosa es un objeto. ¿De dónde vienen los objetos?

La programación orientada a objetos representó una gran ganancia en la productividad del programador, la confiabilidad de la aplicación, y rendimiento. A finales de los años ochenta y principios de los noventa, la mayoría de los lenguajes de programación se convirtieron en un modelo orientado a objetos, y surgieron muchos lenguajes nuevos importantes, como Java, que estaban orientados a objetos nativos.

La revolución de la programación orientada a objetos preparó el escenario para el primer desafío serio a las base de datos relacionales, que surgió a mediados de la década de 1990: había una falta de concordancia entre las representaciones orientadas a objetos de sus datos dentro de los programas y la representación relacional dentro de la base de datos.

Así, los primeros lenguajes de programación orientados a objetos, comenzando con Simula y Smalltalk, presentaron una forma alternativa de diseñar programas en la que las estructuras de datos y sus operaciones tenían una importancia primordial. La programación

orientada a objetos es ampliamente reconocida como un método para producir código altamente confiable y reutilizable.

Los primeros lenguajes orientados a objetos fueron C++, C#, Python, Ruby y Java. Además otros lenguajes introdujeron conceptos de objetos como Visual Basic.NET.

Un objeto entonces tendrá datos como si fuera una variable que indicarán en qué estado se encuentra ese objeto. Pero, un objeto, tendrá también comportamientos, esto es, sabrá hacer cosas.

También como las variables (a las que podemos ver como el tipo más sencillo de objetos) podremos crear y destruir objetos. Para crear un objeto hay que indicar cuál es el molde que vamos a usar. A ese molde que nos permite crear muchos objetos que tendrán los mismos comportamientos pero distintos estados lo llamamos la clase.

Ya volveremos sobre esto más adelante.

Ejecución condicional

Vamos a aprender a guiar el flujo de un programa. En principio, cuando tenemos varias instrucciones una debajo de otra lo que ocurre es que las va ejecutando una tras otra desde arriba hacia abajo.

¿Qué pasa si queremos que, dependiendo de una condición, algunas de las líneas no se ejecuten?

Para eso vamos a recurrir a la instrucción if

La sintaxis de if es:

If condición:

Bloque de instrucciones para ejecutar si la condición es verdadera

Else:

Bloque de instrucciones para ejecutar si la condición es falsa

Ejemplo:

The screenshot shows a Sublime Text window titled '*REPL* [python] - Sublime Text (UNREGISTERED)'. The menu bar includes File, Edit, Selection, Find, View, Goto, Tools, Project, Preferences, and Help. Below the menu is a toolbar with back, forward, and close buttons. The main editor area contains the following Python code:

```
>>>
>>>
>>>
>>> if a<b:
...     print("a es menor que b")
... else:
...     print("b es menor o igual que a")
...
a es menor que b
>>> |
```

The status bar at the bottom left shows 'Line 151, Column 5' and the status bar on the right shows 'Tab Size: 4' and 'Python'.

Funciones y Generadores

Vamos a llamar funciones a un bloque de instrucciones que pueden o no devolver un valor. Las funciones pueden, a su vez, llamar a otras funciones para dividir así una tarea compleja en varias tareas simples. De este modo vamos a facilitar que el código que escribimos sea legible y que no escribamos nunca dos veces el mismo código.

Cuando una función está contenida dentro de una clase (se acuerdan de clases y objetos) entonces la llamamos un método. (ya volveremos más adelante sobre esto)

Vamos a ver que las funciones podrán también recibir uno o varios valores al ser llamadas a hacer su trabajo.

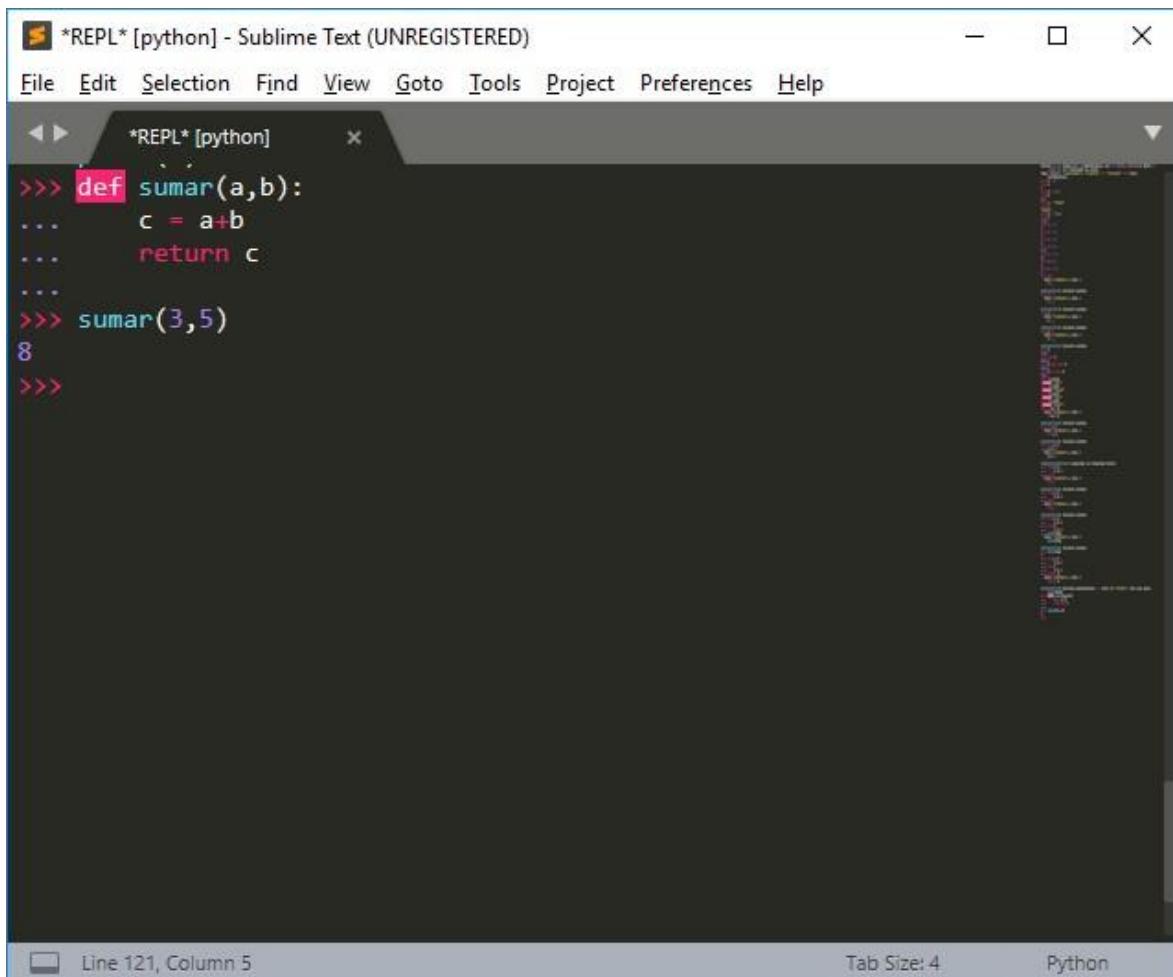
La sintaxis para definir una función es:

Def nombre_función():

 Instrucciones

 [return]

Pongamos un ejemplo haciendo una función que sirva para sumar sus dos parámetros y devolvernos esa suma:



The screenshot shows a Sublime Text window titled "*REPL* [python] - Sublime Text (UNREGISTERED)". The menu bar includes File, Edit, Selection, Find, View, Goto, Tools, Project, Preferences, and Help. The main area displays the following Python code in a REPL session:

```
>>> def sumar(a,b):
...     c = a+b
...     return c
...
>>> sumar(3,5)
8
>>>
```

The status bar at the bottom indicates "Line 121, Column 5", "Tab Size: 4", and "Python".

Vamos a combinar ahora lo que hemos aprendido de las funciones con la ejecución condicional:

The screenshot shows a Sublime Text window titled '*REPL* [python] - Sublime Text (UNREGISTERED)'. The menu bar includes File, Edit, Selection, Find, View, Goto, Tools, Project, Preferences, and Help. The main pane displays a Python script and its execution results:

```
< > *REPL* [python] x
>>> def condicion(a,b):
...     if a<b:
...         print(a," es menor que ",b)
...     else:
...         print(b," es menor que ",a)
...
...
>>> condicion(1,2)
1 es menor que 2
>>> |
```

The status bar at the bottom indicates "Line 130, Column 5", "Tab Size: 4", and "Python".

Antes de seguir incrementando la complejidad vamos a aprender a generar un programa y ejecutarlo luego.

Para eso en vez de usar el Plugin REPL vamos a usar SublimeText directamente. Lo abrimos y ya podemos empezar a volcar nuestro código con la tranquilidad de que no lo estará interpretando cada vez que completemos una línea

The screenshot shows a code editor window titled "PrimerPrograma". The code in the editor is:

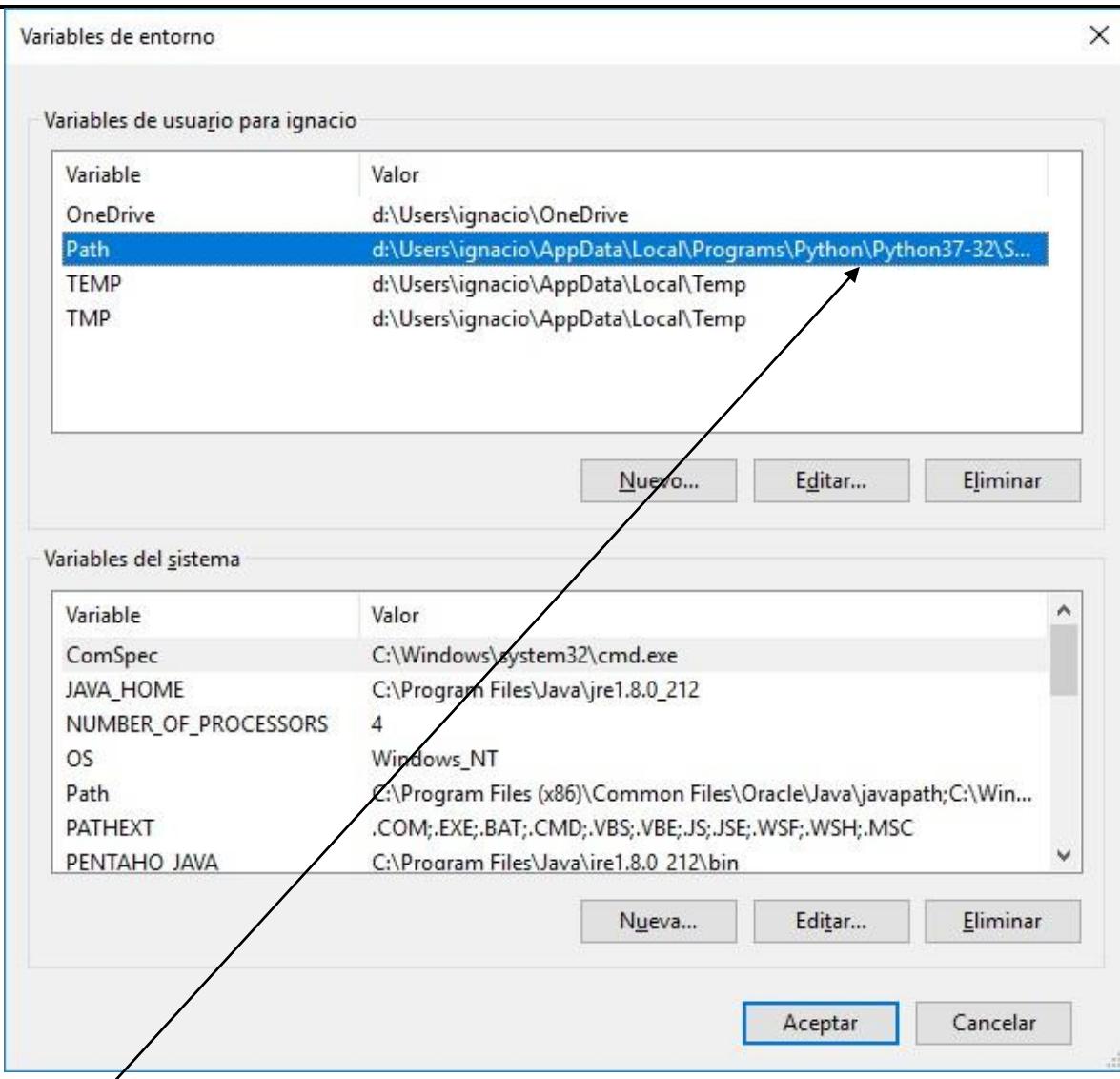
```
1 # Este será nuestro primer programa en Python
2 a = 1
3 b = 5
4 if a<b:
5     print(b-a)
6 else:
7     print(a-b)
8
9
```

The status bar at the bottom indicates "Line 9, Column 1", "Tab Size: 4", and "Plain Text".

Y para guardarlo vamos a File, Save As, elegimos la carpeta y ponemos el nombre del archivo. Muy fácil y directo.

¿Cómo lo ejecutamos?

Primero tenemos que asegurarnos que el directorio donde se encuentra Python está agregado al path.



Si, lo está!

Ahora vamos a ir a la carpeta donde hemos grabado nuestro programa:

```

C:\Windows\system32\cmd.exe
D:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python>python PrimerPrograma.py
4

D:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python>

```

Si miramos como 1 era menor que 5 nos va a imprimir $5 - 1$ que da exactamente los cuatro que nos muestra. Hemos tenido éxito en ejecutar nuestro primer programa!

Ahora bien, puedo ganar algo de tiempo si ejecuto desde dentro del mismo SublimeText. Para eso primero debo asegurarme de guardar mi archivo con la extensión .Py

Después puedo, usando Ctrl B ejecutarlo y:

D:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python\PrimerPrograma.py

File Edit Selection Find View Goto Tools Project Preferences Help

PrimerPrograma.Py

```
1 # Este será nuestro primer programa en Python
2 a = 1
3 b = 5
4 if a<b:
5     print(b-a)
6 else:
7     print(a-b)
8
9
```

4
[Finished in 0.2s]

Line 9, Column 1 Tab Size: 4 Python

Tratemos ahora de hacer que nuestro programa detenga la ejecución como para que podamos introducir un valor desde el teclado. Para eso tenemos la instrucción `input`

The screenshot shows a Python code editor window. The title bar reads "D:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python\PruebaInput.py". The menu bar includes File, Edit, Selection, Find, View, Goto, Tools, Project, Preferences, and Help. A tab labeled "PruebaInput.py" is open, showing the following code:

```
1 # prueba del input
2
3 a = input("ingrese un valor ")
4
5 print("valor ingresado: ",a)
6
7
```

The status bar at the bottom indicates "Line 1, Column 1" and "Tab Size: 4".

Lo ejecutamos desde la línea de comandos:

The screenshot shows a Windows command prompt window titled "cmd.exe". The path is "D:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python>Python PruebaInput.py". The user types "ingrese un valor abc" and presses Enter. The output shows "valor ingresado: abc". The command prompt then returns to the directory.

El valor abc parece haber entrado sin problemas

Pruero con un valor numérico, por ejemplo ingresando 3:

The screenshot shows a Windows command prompt window titled "cmd.exe". The path is "D:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python>Python PruebaInput.py". The user types "ingrese un valor 3" and presses Enter. The output shows "valor ingresado: 3". The command prompt then returns to the directory.

Todo parece funcionar bien. ¿Es el valor de a un número? Para ver de qué tipo es una variable podemos usar la función type:

The screenshot shows a Python code editor window. The title bar reads "D:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python\PruebaInput.py". The menu bar includes File, Edit, Selection, Find, View, Goto, Tools, Project, Preferences, and Help. The code area contains the following Python script:

```
1 # prueba del input
2
3 a = input("ingrese un valor ")
4
5 print(type(a))
6
7
```

The status bar at the bottom indicates "Line 6, Column 1", "Tab Size: 4", and "Python".

Al ejecutar ingresando abc obtenemos:

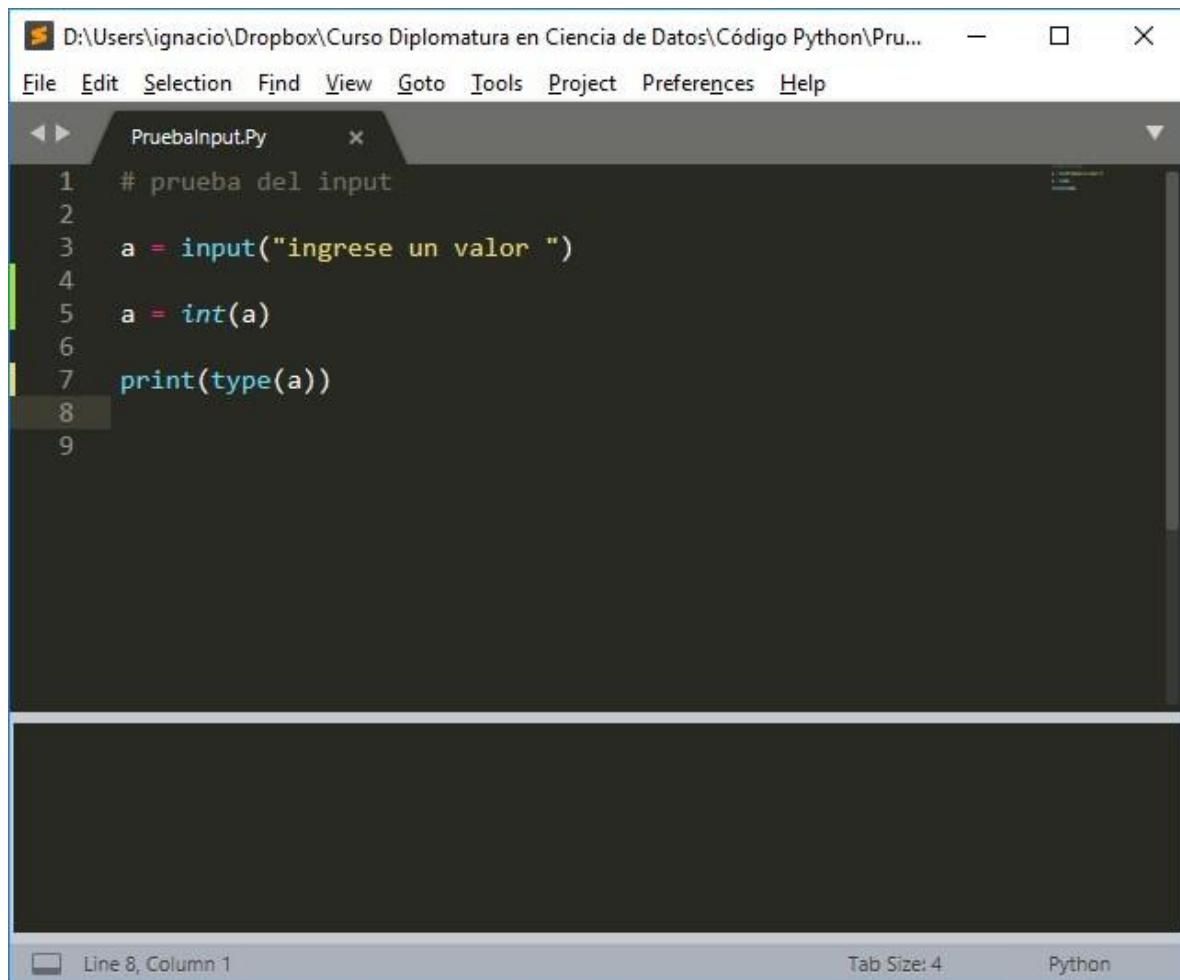
The screenshot shows a Windows command prompt window titled "cmd.exe". The command "D:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python>Python PruebaInput.py" is run. The output shows the user input "ingrese un valor abc" followed by the type of the variable: "<class 'str'>". The command prompt then returns to the directory "D:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python>".

Ahora volvemos a probar con el 3:

The screenshot shows a Windows command prompt window titled "cmd.exe". The command "D:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python>Python PruebaInput.py" is run. The output shows the user input "ingrese un valor 3" followed by the type of the variable: "<class 'str'>". The command prompt then returns to the directory "D:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python>".

El 3 también lo reconoce como una cadena de caracteres o string. Si queremos convertirlo a número tenemos que usar la función int.

La función int es la primera de las funciones que vamos a estudiar que se usan para convertir (si se puede) los valores de un tipo en otro. En nuestro caso vamos a convertir un string en un entero:



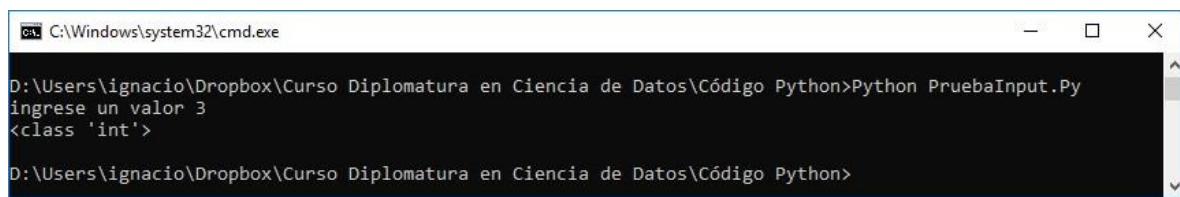
```
D:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python\PruebaInput.py

File Edit Selection Find View Goto Tools Project Preferences Help

◀ ▶ PruebaInput.py ×
1 # prueba del input
2
3 a = input("ingrese un valor ")
4
5 a = int(a)
6
7 print(type(a))
8
9

Line 8, Column 1 Tab Size: 4 Python
```

Lo ejecutamos:



```
C:\Windows\system32\cmd.exe

D:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python>Python PruebaInput.py
ingrese un valor 3
<class 'int'>

D:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python>
```

Pero la malsana curiosidad nos empuja siempre a probar el límite de lo posible para eso, con toda intención vamos a introducir abc cuando nos pide un valor para ver que hace:

```
C:\Windows\system32\cmd.exe
D:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python>Python PruebaInput.Py
ingrese un valor abc
Traceback (most recent call last):
  File "PruebaInput.Py", line 5, in <module>
    a = int(a)
ValueError: invalid literal for int() with base 10: 'abc'

D:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python>
```

Lo que acabamos de ver es un mensaje de error. Es bueno acostumbrarse porque, programando, los veremos a menudo.

Es útil que aprendamos a leerlos:

- Indica en qué archivo ocurre el error
- Indica en qué línea está la instrucción que llevó a error
- Trata de describir el error ocurrido

Vamos entonces a combinar las cosas que ya sabemos en un nuevo programa:

- Ejecución condicional
- Funciones definidas por nosotros
- Función int para convertir a enteros
- Introducir valores desde el teclado

```

D:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python\Seg...
File Edit Selection Find View Goto Tools Project Preferences Help
SegundoPrograma.Py x
4 def sumar(a,b):
5     c = a + b
6     return c
7
8 # acá empieza la ejecución de mi programa,
9 # la parte de la función "no hace nada más que dejarla lista"
10
11 a1 = int(input("Número 1 "))
12 b1 = int(input("Número 2 "))
13
14 if a1<b1:
15     a1 = a1 * 2
16     print(sumar(a1,b1))
17 else:
18     print(sumar(a1,b1))
19
20
21
Line 1, Column 1 Tab Size: 4 Python

```

Vamos a asegurarnos que entendemos lo que está pasando:

```

C:\Windows\system32\cmd.exe
D:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python>Python SegundoPrograma.py
Número 1 1
Número 2 3
5

D:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python>Python SegundoPrograma.py
Número 1 3
Número 2 1
4

D:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python>

```

En la primera vez que ejecuto el programa ingreso el 1 y el 3. El 1 es menor que el 3 y, por lo tanto, al 1 lo multiplica por dos antes de sumarlo. Luego nos muestra el resultado de la suma y, por supuesto, da 5.

En la segunda vez que ejecuto el programa ingreso el 3 y el 1. El 3 no es menor que el 1 y, por lo tanto los suma tal y como están y me da 4.

Generadores:

Un generador es una función que podemos usar para producir una serie de valores, potencialmente infinita sin ocupar infinito espacio en la memoria ni infinito tiempo de proceso.

¿Cómo funciona este milagro?

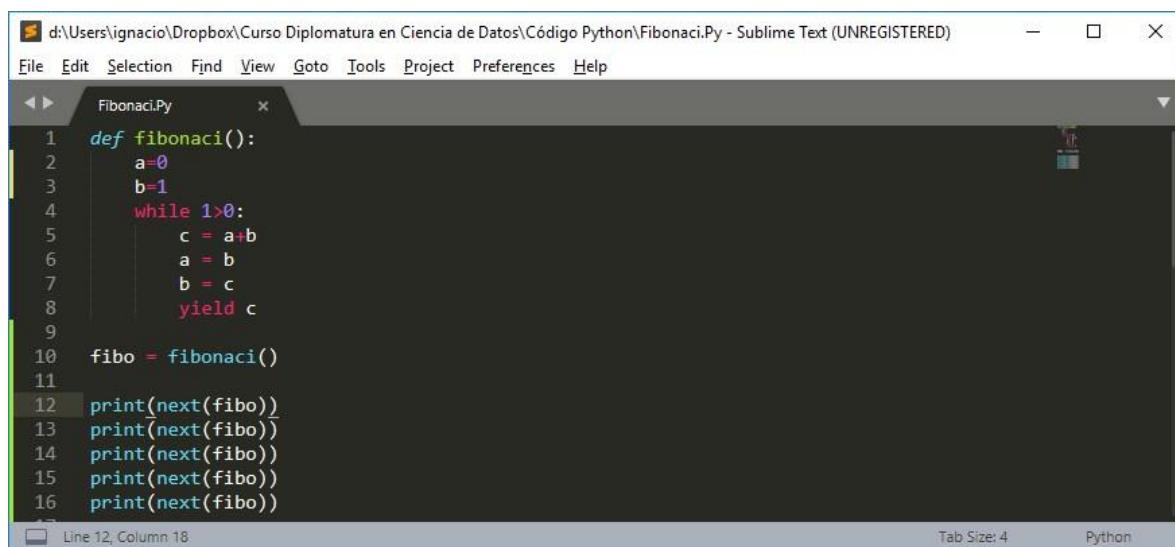
De a poco. El truco consiste en generar el siguiente elemento y luego devolver el control al programa que invoca la función a la vez que almacena el estado interno para no olvidarse y generar el siguiente valor.

Parece inevitable usar como ejemplo la sucesión de Fibonacci.

Para los que eso no signifique nada vamos a contarla en forma sencilla: La sucesión de Fibonacci se forma poniendo como siguiente paso a la suma de los dos anteriores.

Arranca con 0, sigue con 1, luego $0+1 = 1$, luego $1+1 = 2$, luego $1+2=3$ luego $2+3=5$ y sigue...

La sucesión de Fibonacci queda entonces 1, 2, 3, 5, 8, 13, 21, ...



```
d:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python\Fibonacci.py - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
Fibonacci.py
1 def fibonacci():
2     a=0
3     b=1
4     while 1>0:
5         c = a+b
6         a = b
7         b = c
8         yield c
9
10 fibo = fibonacci()
11
12 print(next(fibo))
13 print(next(fibo))
14 print(next(fibo))
15 print(next(fibo))
16 print(next(fibo))
Line 12, Column 18
Tab Size: 4
Python
```

Recorramos el código:

Empieza definiendo la función fibonacci.

No tiene parámetros

El while $1>0$ abre un ciclo infinito. Casi me dio miedo escribirlo.

Guarda en c el siguiente elemento de la sucesión

Guarda en a el contenido de b

Guarda en b el contenido de c

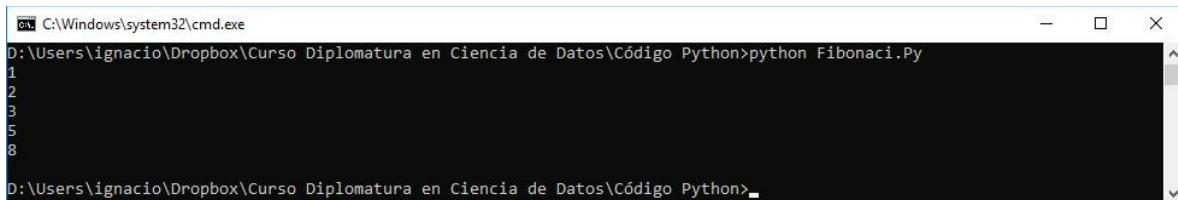
Acá viene lo diferente: devuelve el contenido de c pero no lo hace con return sino con yield.

Esto le permite guardar el estado de la función que ya no arrancará con a=0 y b=1 sino que “recordará” el contenido que tenía cuando pidamos el elemento siguiente.

Ya fuera de la definición de la función lo primero que hacemos es crear un objeto fibo que nos servirá para llamar a la función fibonaci guardando el estado.

Cada vez que hacemos print(next(fibo)) forzamos a la función a calcular el siguiente valor y a imprimirla.

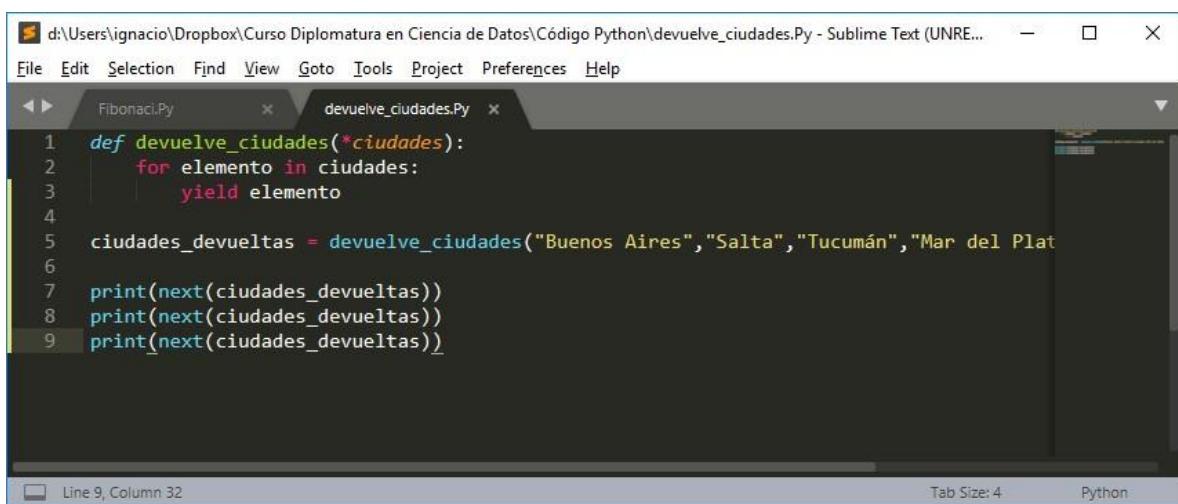
Lo ejecutamos:



```
C:\Windows\system32\cmd.exe
D:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python>python Fibonaci.Py
1
2
3
5
8
D:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python>
```

Ahora vamos a introducir la instrucción yield from que es un agregado relativamente reciente al lenguaje Python.

Lo primero que vamos a necesitar es entender que significa un * delante de un argumento en una definición de una función:

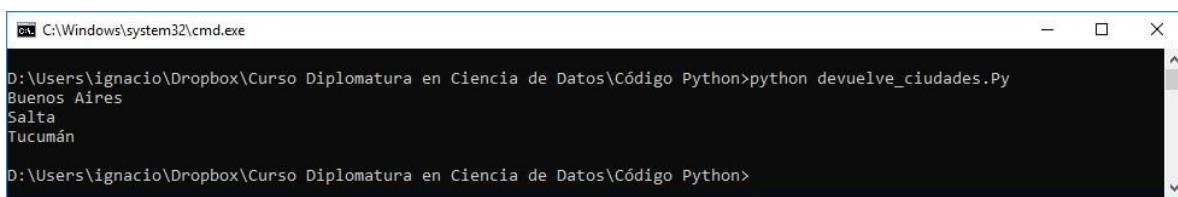


```
File Edit Selection Find View Goto Tools Project Preferences Help
Fibonaci.Py devuelve_ciudades.Py
1 def devuelve_ciudades(*ciudades):
2     for elemento in ciudades:
3         yield elemento
4
5 ciudades_devueltas = devuelve_ciudades("Buenos Aires","Salta","Tucumán","Mar del Plat
6
7 print(next(ciudades_devueltas))
8 print(next(ciudades_devueltas))
9 print(next(ciudades_devueltas))

Line 9, Column 32
Tab Size: 4
Python
```

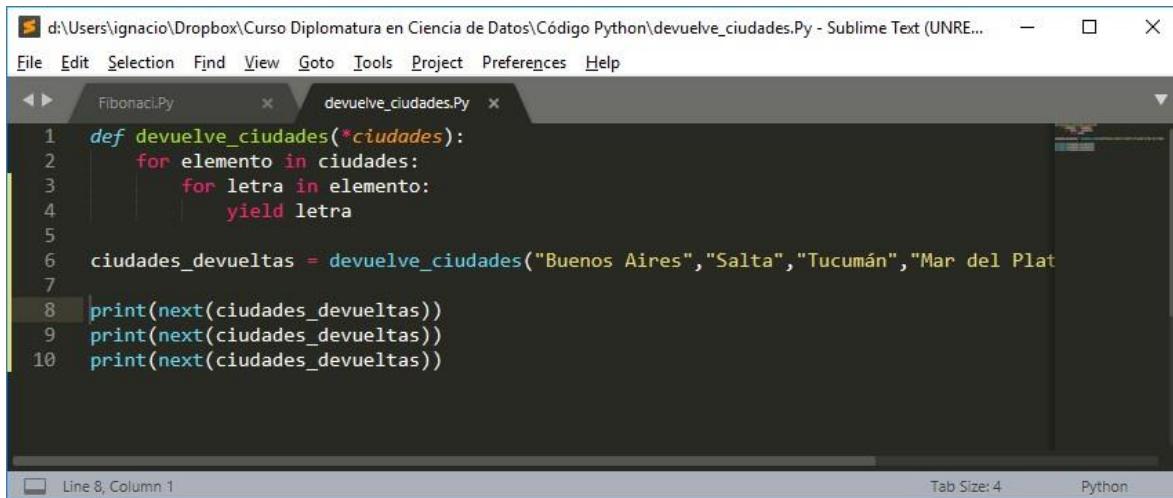
Utilizamos *ciudades en la definición de la función para que Python sepa que puede esperar una cantidad variable de argumentos.

Al ejecutar obtenemos:



```
C:\Windows\system32\cmd.exe
D:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python>python devuelve_ciudades.Py
Buenos Aires
Salta
Tucumán
D:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python>
```

Ahora queremos recorrer las letras de las ciudades. Por suerte para nosotros en Python nos encontramos con que las cadenas de caracteres también podemos recorrerlas con un ciclo for:



```

d:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python\devuelve_ciudades.Py - Sublime Text (UNRE...)

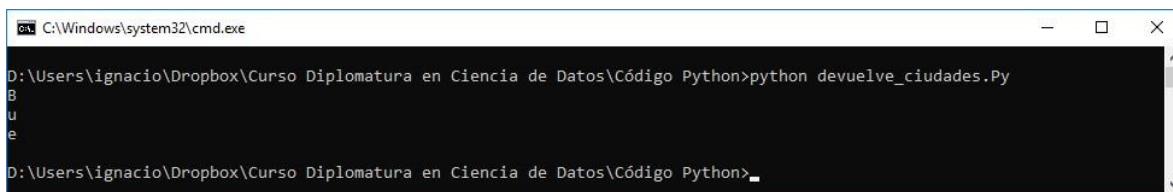
File Edit Selection Find View Goto Tools Project Preferences Help
Fibonacci.Py devuelve_ciudades.Py

1 def devuelve_ciudades(*ciudades):
2     for elemento in ciudades:
3         for letra in elemento:
4             yield letra
5
6 ciudades_devueltas = devuelve_ciudades("Buenos Aires", "Salta", "Tucumán", "Mar del Plat"
7
8 print(next(ciudades_devueltas))
9 print(next(ciudades_devueltas))
10 print(next(ciudades_devueltas))

Line 8, Column 1
Tab Size: 4
Python

```

Y obtenemos:



```

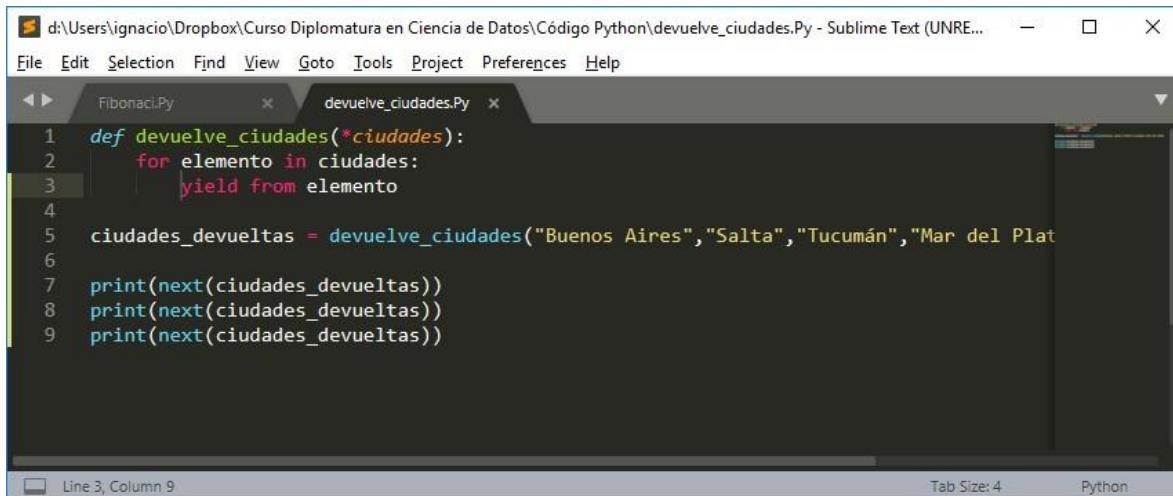
C:\Windows\system32\cmd.exe

D:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python>python devuelve_ciudades.py
B
u
e

D:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python>

```

Una forma más simple de hacer esto, evitando el ciclo anidado que recorre el elemento es:



```

d:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python\devuelve_ciudades.Py - Sublime Text (UNRE...)

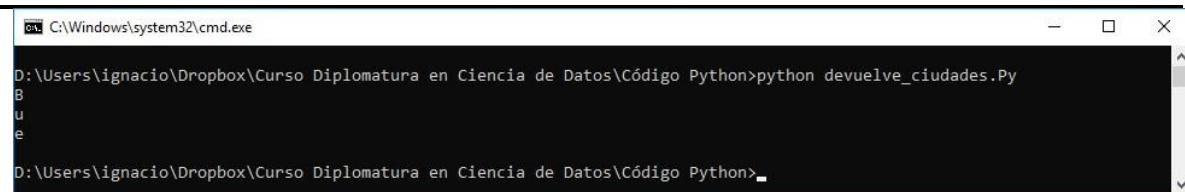
File Edit Selection Find View Goto Tools Project Preferences Help
Fibonacci.Py devuelve_ciudades.Py

1 def devuelve_ciudades(*ciudades):
2     for elemento in ciudades:
3         yield from elemento
4
5 ciudades_devueltas = devuelve_ciudades("Buenos Aires", "Salta", "Tucumán", "Mar del Plat"
6
7 print(next(ciudades_devueltas))
8 print(next(ciudades_devueltas))
9 print(next(ciudades_devueltas))

Line 3, Column 9
Tab Size: 4
Python

```

Y, por supuesto, al ejecutar obtenemos lo mismo:



A screenshot of a Windows Command Prompt window titled 'C:\Windows\system32\cmd.exe'. The window shows the command 'python devuelve_ciudades.Py' being run from the directory 'D:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python'. The output of the script is displayed, showing the letters 'B', 'u', and 'e' on separate lines.

Listas, tuplas y diccionarios

Listas:

Las listas nos permiten guardar una colección de valores de distinto tipo. A diferencia de lo que ocurre con los vectores en otros lenguajes de programación las listas pueden tener un elemento de tipo numérico, otro de tipo texto, otro de tipo booleano, sin limitaciones.

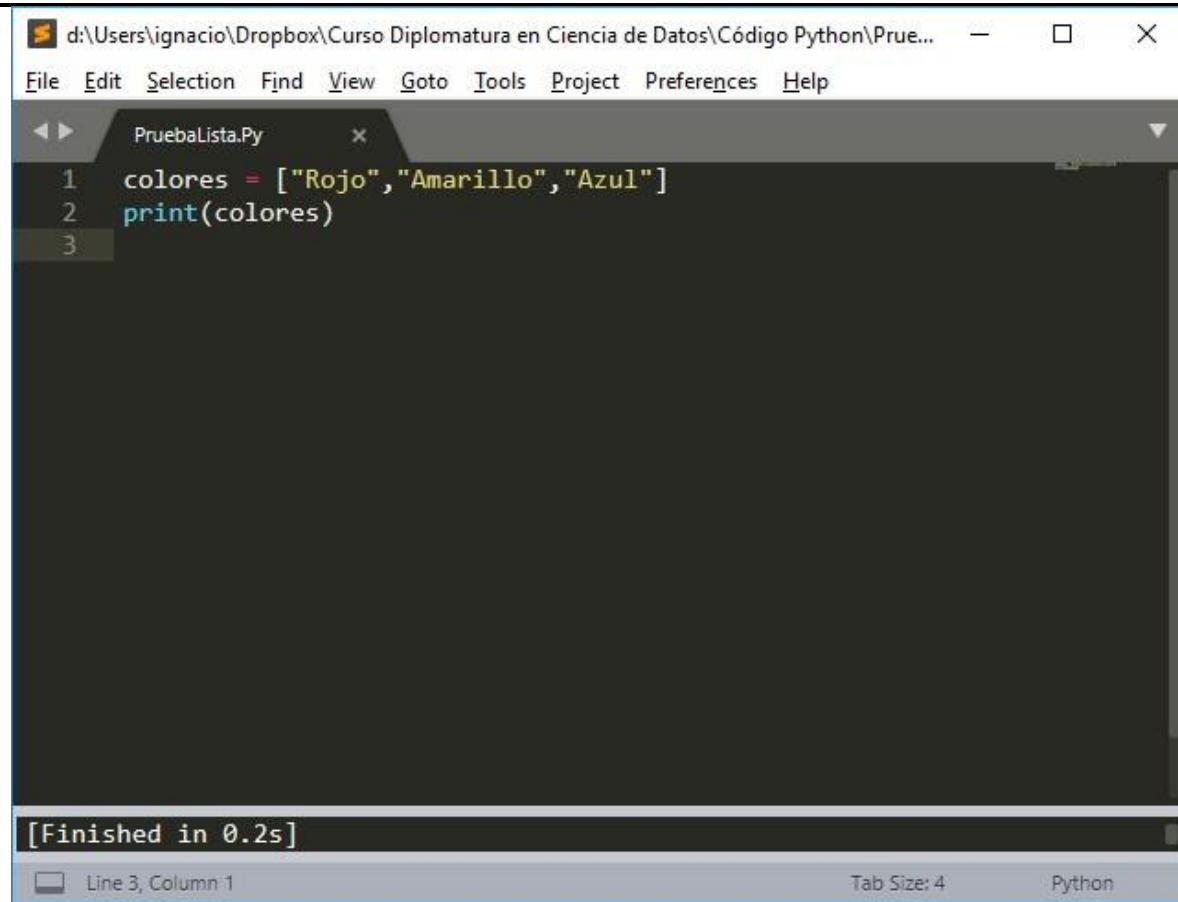
Veremos que resulta fácil expandir la lista durante la ejecución de un programa. Esto es muy práctico pues no necesitamos, de antemano, conocer el tamaño máximo que esta lista alcanzará.

Obviamente toda esta flexibilidad no se obtiene gratis. Las estructuras que otros lenguajes usan para manejar vectores y matrices son más eficientes. De hecho existen extensiones de Python que le permiten incorporar estructuras comparables a vectores y matrices.

Por ahora vamos a concentrarnos en las listas.

Para generar una lista debemos primero elegir el nombre del objeto con el que la utilizaremos. Ese nombre sigue las mismas convenciones que los nombres de variables (que, después de todo, también son objetos)

Vamos entonces a armar una lista que se llame colores y vamos a guardar en ella los elementos "Rojo", "Amarillo" y "Azul"

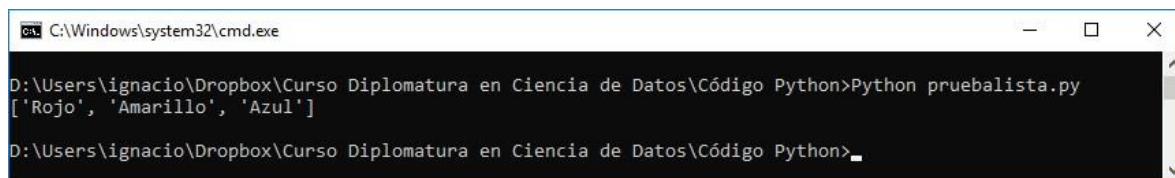


The screenshot shows a Python code editor window titled "PruebaLista.py". The code in the editor is:

```
1 colores = ["Rojo", "Amarillo", "Azul"]
2 print(colores)
3
```

At the bottom of the editor, a status bar displays "[Finished in 0.2s]". Below the editor, the status bar also shows "Line 3, Column 1", "Tab Size: 4", and "Python".

Y cuando lo ejecuto:



The screenshot shows a Windows command prompt window. The command entered is "python pruebalista.py". The output displayed is:

```
C:\Windows\system32\cmd.exe
D:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python>Python pruebalista.py
['Rojo', 'Amarillo', 'Azul']
D:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python>
```

Una cosa que acabamos de probar, sin darnos cuenta, pero que vale la pena subrayar es que `print`, que sabía imprimir variables y textos ahora descubrimos que también sabe imprimir listas.

`Print` es entonces un método que los distintos objetos tienen y, por lo tanto, todos saben ejecutar. Cada objeto necesitará definir su `print`. En el caso de la lista el método `print` agrega el corchete que se abre, todos los elementos separados por comas y el corchete que se cierra.

Hagamos algo más ambicioso:

- Generemos una lista de cinco números
- Generemos una lista que contenga las dos listas anteriores
- Imprimamos la lista de listas y veamos que pasa

The screenshot shows a Python code editor window titled "PruebaLista2.Py". The code is as follows:

```
1 colores = ["Rojo", "Amarillo", "Azul"]
2 numeros = [1, 2, 3, 4, 5]
3 lista = [colores, numeros]
4 print(lista)
5
6
```

The status bar at the bottom indicates "[Finished in 0.2s]". Below the status bar, it says "Line 5, Column 1". On the right, it shows "Tab Size: 4" and "Python".

Y lo ejecutamos:

The screenshot shows a command prompt window with the following text:

```
C:\Windows\system32\cmd.exe
D:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python>Python pruebalista2.py
[['Rojo', 'Amarillo', 'Azul'], [1, 2, 3, 4, 5]]
D:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python>
```

¿Qué pasó?

El método `print`, aplicado a una lista puso el corchete que se abre y luego empezó a aplicar el método `print` a cada elemento. Como el primer elemento era también una lista abrió un segundo corchete y luego siguió aplicando el método `print` a los elementos del primer elemento.

Esta es una de las cosas piolas de los objetos. Al aplicar el mismo método vamos haciendo cosas distintas en distintos objetos pero que siempre tienen el mismo propósito, en este caso, mostrar el contenido del objeto.

Si queremos referirnos sólo a un elemento de la lista utilizamos el índice que identifica a ese elemento:

The screenshot shows a code editor window titled 'PruebaLista2.Py'. The code is as follows:

```
1 colores = ["Rojo", "Amarillo", "Azul"]
2 numeros = [1, 2, 3, 4, 5]
3 lista = [colores, numeros]
4 print(lista)
5 print(lista[1])
6
```

The status bar at the bottom indicates '[Finished in 0.2s]'. The bottom right corner shows 'Tab Size: 4' and 'Python'.

Y ejecutamos:

The screenshot shows a command prompt window with the following output:

```
C:\Windows\system32\cmd.exe
D:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python>Python pruebalista2.py
[['Rojo', 'Amarillo', 'Azul'], [1, 2, 3, 4, 5]]
[1, 2, 3, 4, 5]
D:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python>
```

Y Python nos vuelve a sorprender. Esperábamos ver la lista de colores pero nos mostró la de números. Eso se debe a que Python numera los índices desde 0 y no desde 1.

Y, si seguimos desarrollando nuestro ejemplo:

The screenshot shows a Python code editor window. The file is named 'PruebaLista2.py'. The code contains:

```
1 colores = ["Rojo", "Amarillo", "Azul"]
2 numeros = [1, 2, 3, 4, 5]
3 lista = [colores, numeros]
4 print(lista)
5 print(lista[1])
6 print(lista[1][1])
```

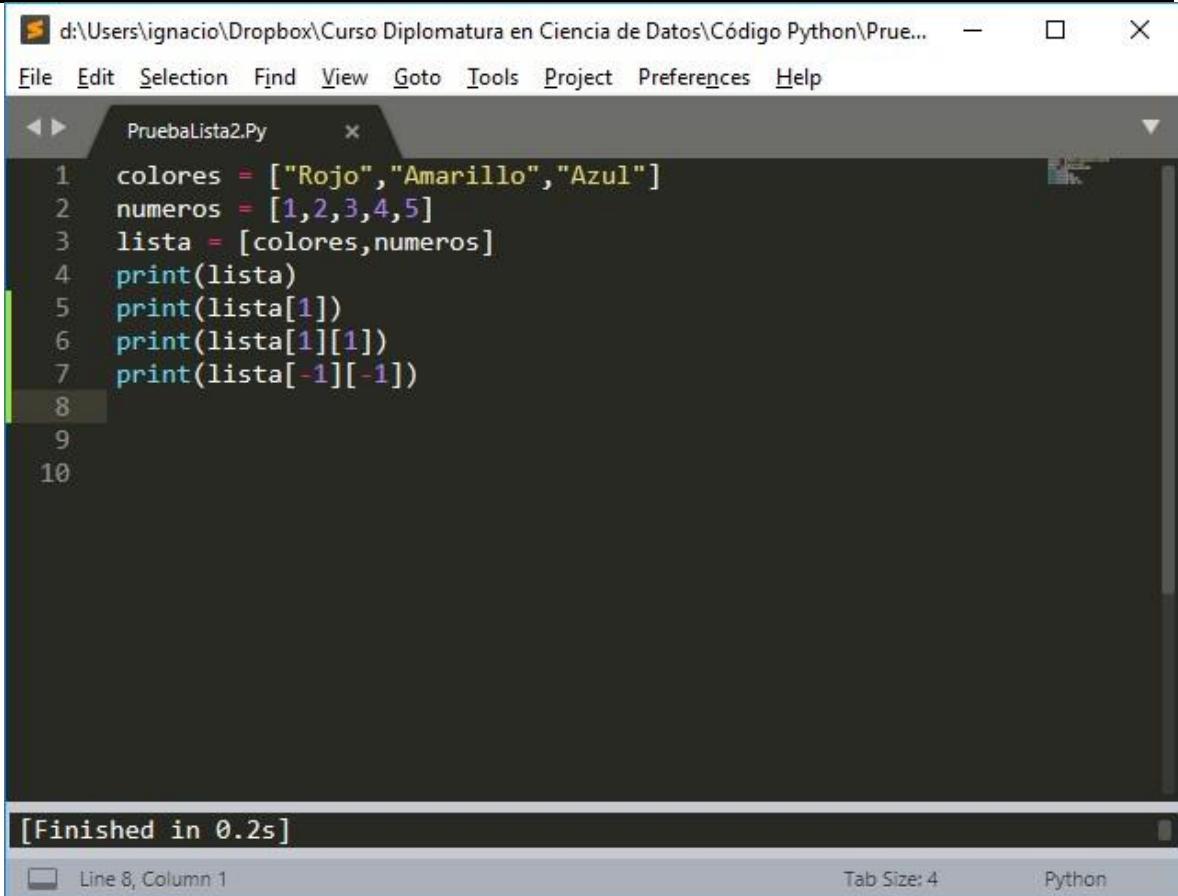
The status bar at the bottom indicates '[Finished in 0.2s]'. The bottom right corner shows 'Python'.

Y ejecutamos:

The screenshot shows a Windows command prompt window. The command 'cmd.exe' is run, and the script 'PruebaLista2.py' is executed. The output is:

```
C:\Windows\system32\cmd.exe
[['Rojo', 'Amarillo', 'Azul'], [1, 2, 3, 4, 5]]
[1, 2, 3, 4, 5]
2
D:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python>
```

Python nos permite, utilizando índices negativos acceder a los elementos de una lista pero contando desde el final:

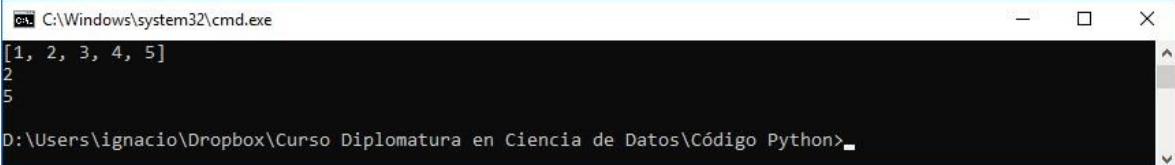


The screenshot shows a Python code editor window titled "PruebaLista2.Py". The code is as follows:

```
1 colores = ["Rojo", "Amarillo", "Azul"]
2 numeros = [1, 2, 3, 4, 5]
3 lista = [colores, numeros]
4 print(lista)
5 print(lista[1])
6 print(lista[1][1])
7 print(lista[-1][-1])
8
9
10
```

The status bar at the bottom indicates "[Finished in 0.2s]". The bottom right corner shows "Python".

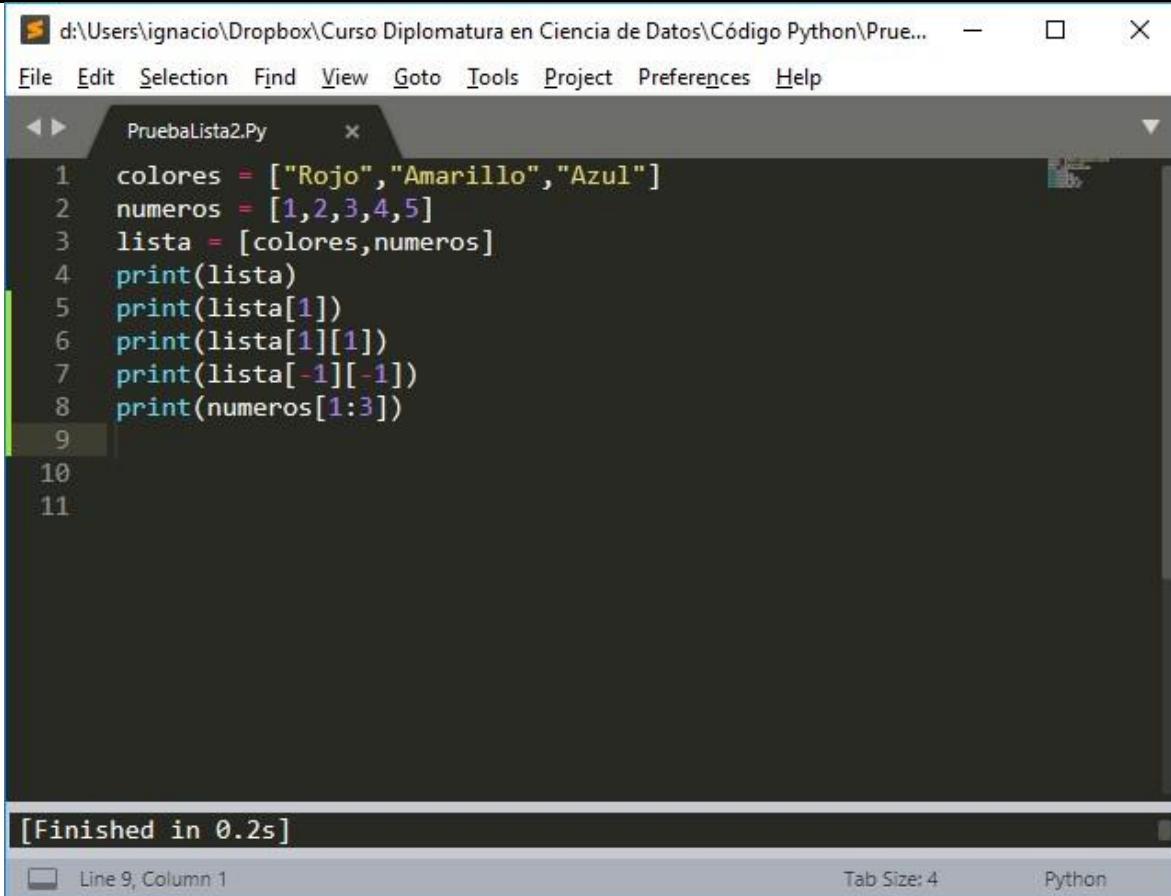
Y ejecutamos:



The screenshot shows a Windows Command Prompt window (cmd.exe) with the following output:

```
C:\Windows\system32\cmd.exe
[1, 2, 3, 4, 5]
2
5
D:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python>
```

También podemos extraer porciones de una lista utilizando los índices:

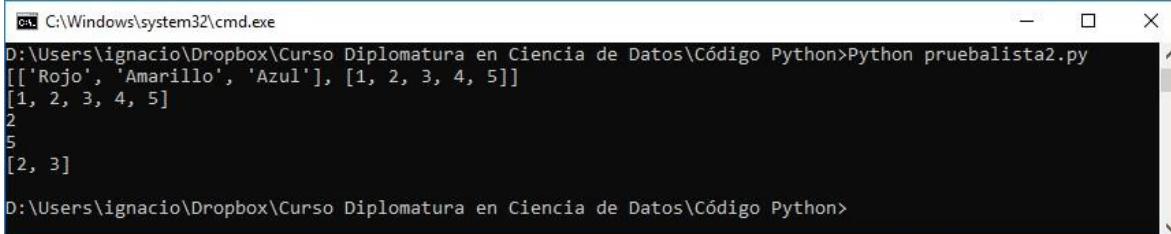


```

d:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python\Prue...
File Edit Selection Find View Goto Tools Project Preferences Help
PruebaLista2.Py x
1 colores = ["Rojo", "Amarillo", "Azul"]
2 numeros = [1, 2, 3, 4, 5]
3 lista = [colores, numeros]
4 print(lista)
5 print(lista[1])
6 print(lista[1][1])
7 print(lista[-1][-1])
8 print(numeros[1:3])
9
10
11
[Finished in 0.2s]
Line 9, Column 1 Tab Size: 4 Python

```

Si ejecutamos:



```

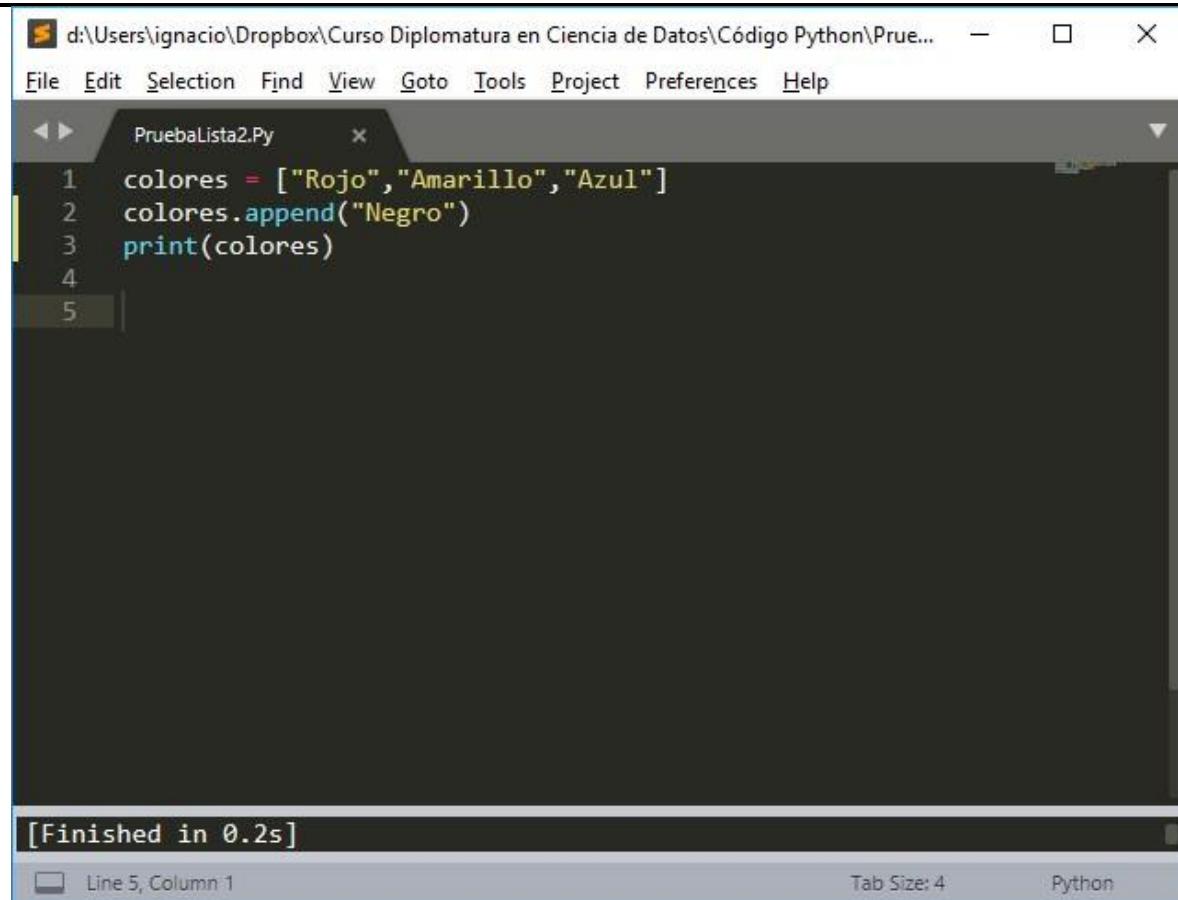
C:\Windows\system32\cmd.exe
D:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python>Python pruebalista2.py
[['Rojo', 'Amarillo', 'Azul'], [1, 2, 3, 4, 5]]
[1, 2, 3, 4, 5]
2
5
[2, 3]
D:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python>

```

Conviene notar que al pedir los elementos del 1 al 3 el elemento 1 es incluido pero el 3 no.

Tenemos formas abreviadas de referirnos a una parte de una lista. Si ponemos [:2] entonces nos quedamos con los dos primeros elementos y si ponemos [2:] nos quedamos todos los elementos menos los dos primeros.

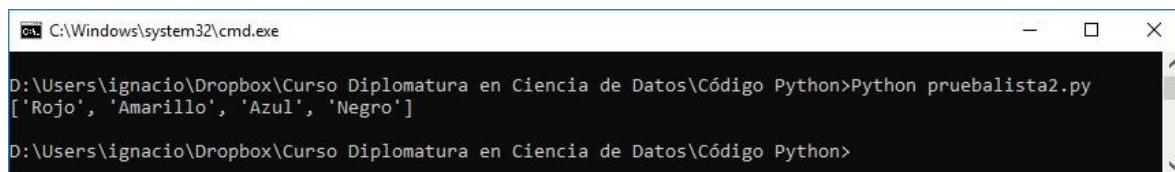
Para agregar elementos a una lista podemos utilizar el método append:



```
PruebaLista2.Py
1 colores = ["Rojo", "Amarillo", "Azul"]
2 colores.append("Negro")
3 print(colores)
4
5

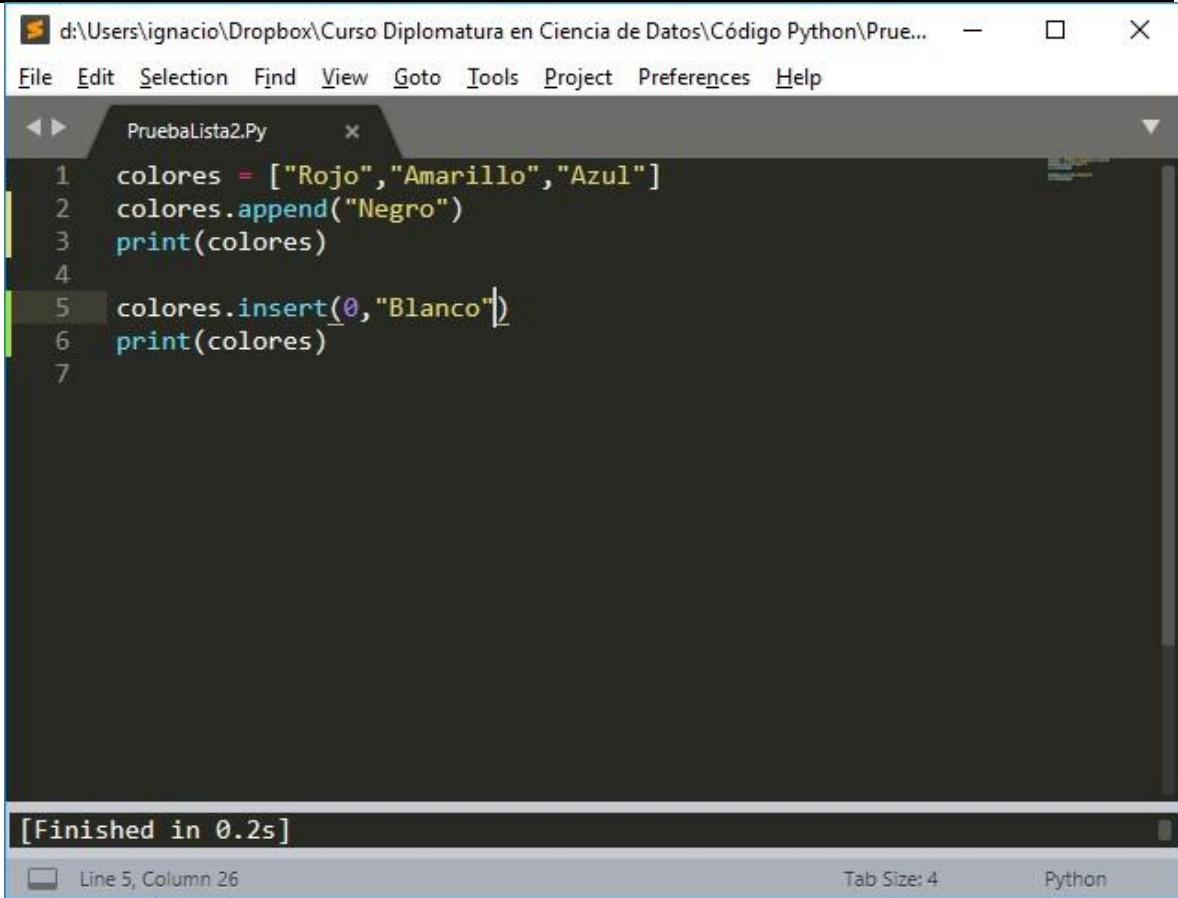
[Finished in 0.2s]
```

Al ejecutar:



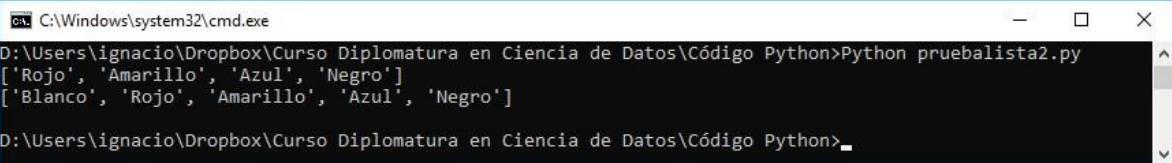
```
C:\Windows\system32\cmd.exe
D:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python>Python pruebalista2.py
['Rojo', 'Amarillo', 'Azul', 'Negro']
D:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python>
```

Y si queremos agregar un elemento a una lista, pero no al final sino en un punto intermedio debemos utilizar el método insert:



```
d:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python\Prue... ━ ━ X
File Edit Selection Find View Goto Tools Project Preferences Help
◀ ▶ PruebaLista2.Py ×
1 colores = ["Rojo", "Amarillo", "Azul"]
2 colores.append("Negro")
3 print(colores)
4
5 colores.insert(0,"Blanco")
6 print(colores)
7
[Finished in 0.2s]
Line 5, Column 26 Tab Size: 4 Python
```

El cero indica la cantidad de elementos que quedarán antes del que estoy insertando. Al elegir un cero el nuevo elemento encabezará la lista:



```
C:\Windows\system32\cmd.exe
D:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python>Python pruebalista2.py
['Rojo', 'Amarillo', 'Azul', 'Negro']
['Blanco', 'Rojo', 'Amarillo', 'Azul', 'Negro']
D:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python>
```

Para extender una lista agregando los elementos de otra podemos usar el método extend:

The screenshot shows a Jupyter Notebook cell titled "PruebaLista2.Py". The code is as follows:

```
1 colores = ["Rojo", "Amarillo", "Azul"]
2 colores.append("Negro")
3 print(colores)
4
5 colores.insert(0,"Blanco")
6 print(colores)
7
8 colores.extend(colores)
9 print(colores)
10
11
```

The output at the bottom of the cell is "[Finished in 0.2s]". The status bar at the bottom right indicates "Line 10, Column 1", "Tab Size: 4", and "Python".

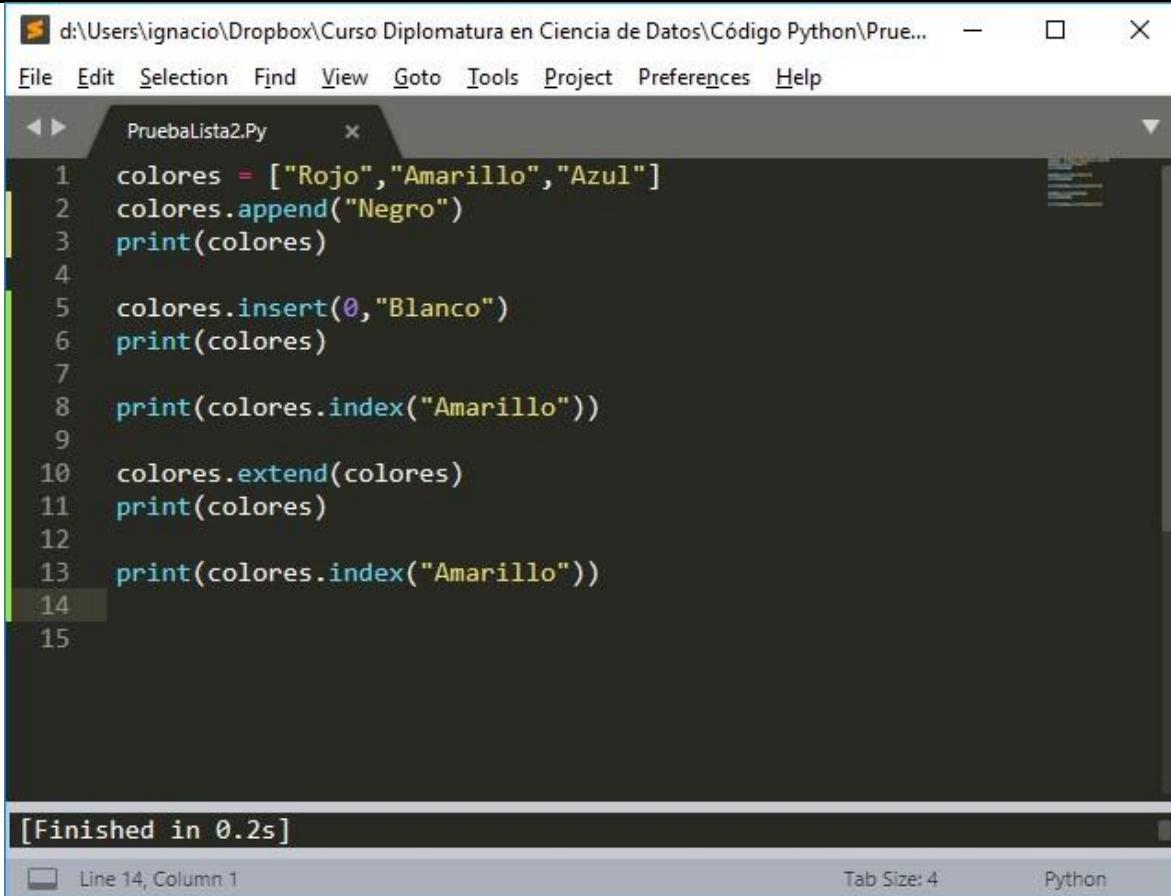
Al ejecutar nos queda:

The screenshot shows a Windows Command Prompt window with the title "cmd C:\Windows\system32\cmd.exe". The command entered is "D:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python>Python pruebalista2.py". The output shows three lists being printed:

```
['Rojo', 'Amarillo', 'Azul', 'Negro']
['Blanco', 'Rojo', 'Amarillo', 'Azul', 'Negro']
['Blanco', 'Rojo', 'Amarillo', 'Azul', 'Negro', 'Blanco', 'Rojo', 'Amarillo', 'Azul', 'Negro']
```

The prompt at the bottom is "D:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python>".

Si tenemos necesidad de averiguar en qué posición está un elemento usamos el método index:

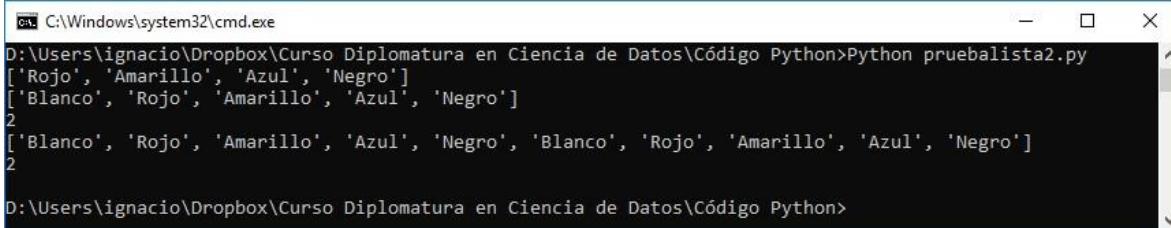


```

d:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python\Prue...
File Edit Selection Find View Goto Tools Project Preferences Help
PruebaLista2.Py x
1 colores = ["Rojo", "Amarillo", "Azul"]
2 colores.append("Negro")
3 print(colores)
4
5 colores.insert(0,"Blanco")
6 print(colores)
7
8 print(colores.index("Amarillo"))
9
10 colores.extend(colores)
11 print(colores)
12
13 print(colores.index("Amarillo"))
14
15
[Finished in 0.2s]
Line 14, Column 1 Tab Size: 4 Python

```

Ejecutamos y nos encontramos con que:



```

C:\Windows\system32\cmd.exe
D:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python>Python pruebalista2.py
['Rojo', 'Amarillo', 'Azul', 'Negro']
['Blanco', 'Rojo', 'Amarillo', 'Azul', 'Negro']
2
['Blanco', 'Rojo', 'Amarillo', 'Azul', 'Negro', 'Blanco', 'Rojo', 'Amarillo', 'Azul', 'Negro']
2
D:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python>

```

En el primer caso “Amarillo” está en la posición 2. No nos quedan dudas. Pero, ¿Qué pasa luego?

Hay dos elementos que son “Amarillo”. Index nos devuelve el primero que encuentra.

También es útil poder verificar si un elemento se encuentra en una lista. Para eso contamos con el método in:

The screenshot shows a Sublime Text window with a dark theme. A file named 'PruebaLista2.py' is open, containing the following code:

```
1 colores = ["Rojo", "Amarillo", "Azul"]
2 |
3 print("Rojo" in colores)
4 print("Violeta" in colores)
5
6
7
```

Below the code, the output of the execution is displayed in a light gray terminal-like area:

```
True
False
[Finished in 0.1s]
```

At the bottom of the window, status bars indicate 'Line 2, Column 1', 'Tab Size: 4', and 'Python'.

Aquí, en vez de ejecutar desde el sistema operativo hemos ejecutado dentro de SublimeText lo que se hace con Ctrl B

Verificamos que Rojo se encontraba y que Violeta no se encontraba.

Otra opción que tenemos es la de remover elementos de una lista. Para eso tenemos el método remove:

The screenshot shows a Python code editor window. The menu bar includes File, Edit, Selection, Find, View, Goto, Tools, Project, Preferences, and Help. The title bar shows the path d:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python\Prue... and the file name PruebaLista2.py. The code in the editor is:

```
1 colores = ["Rojo", "Amarillo", "Azul"]
2
3 colores.remove("Rojo")
4
5 print(colores)
6
7
8
9
10
```

The output window below shows the execution results:

```
['Amarillo', 'Azul']
[Finished in 0.1s]
```

At the bottom, status information includes Line 3, Column 23, Tab Size: 4, and Python.

Es útil notar que tenemos que poner el contenido del elemento dentro de remove y no el índice.

¿Y si un elemento se encuentra varias veces?

Probamos y:

```
d:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python\Prue... ━ ━ X
File Edit Selection Find View Goto Tools Project Preferences Help
PruebaLista2.Py ×
1 colores = ["Rojo", "Amarillo", "Azul"]
2
3 colores.extend(colores)
4
5 colores.remove("Rojo")
6
7 print(colores)
8
9
10
11
12
13

['Amarillo', 'Azul', 'Rojo', 'Amarillo', 'Azul']
[Finished in 0.1s]

Line 3, Column 11 Tab Size: 4 Python
```

Sólo se elimina la primera ocurrencia.

También contamos con el método pop para eliminar el último elemento:

The screenshot shows a Python code editor window titled "PruebaLista2.Py". The code in the editor is:

```
1 colores = ["Rojo", "Amarillo", "Azul"]
2
3 colores.extend(colores)
4
5 colores.remove("Rojo")
6
7 print(colores)
8
9 colores.pop()
10
11 print(colores)
12
```

Below the code editor, the terminal output is displayed:

```
['Amarillo', 'Azul', 'Rojo', 'Amarillo', 'Azul']
['Amarillo', 'Azul', 'Rojo', 'Amarillo']
[Finished in 0.1s]
```

At the bottom of the editor window, status bars show "Line 9, Column 14; Build finished", "Tab Size: 4", and "Python".

Tenemos también los operadores + y * que podemos aplicar entre listas.

El operador + produce el mismo resultado que el método extend.

El operador * repite la misma lista tantas veces como el número que va a la derecha del operador lo indique:

The screenshot shows a Python code editor window titled "PruebaLista2.Py". The code is as follows:

```
1 colores1 = ["Rojo", "Amarillo", "Azul"]
2
3 colores2 = ["Rojo", "Verde", "Violeta"]
4
5 colores = colores1+colores2
6
7 print(colores)
8
9 print(colores1*2)
10
```

The output window below the editor shows the results of running the script:

```
['Rojo', 'Amarillo', 'Azul', 'Rojo', 'Verde', 'Violeta']
['Rojo', 'Amarillo', 'Azul', 'Rojo', 'Amarillo', 'Azul']
[Finished in 0.1s]
```

At the bottom of the editor window, it says "Line 10, Column 1" and "Python".

Con esto podemos dejar en paz las listas por un tiempo y empezar a trabajar con otra estructura no menos importante, las tuplas.

Tuplas:

Las tuplas son listas inmutables. Eso quiere decir que una vez creada no es posible agregar o quitar elementos.

Nos permiten crear una nueva tupla extrayendo los elementos de una tupla pre-existente.

Las tuplas permiten usar el método `index` para buscar elementos y también admiten buscar si un elemento existe en la tupla o no.

¿Y por qué no usar listas?

Las tuplas ocupan menos lugar y son más rápidas pues se organizan internamente de un modo distinto que las listas.

En cuanto a la sintaxis, para distinguir las tuplas de las listas usaremos () para las tuplas y [] para las listas.

Por ejemplo:

The screenshot shows a Python code editor window with the following details:

- Title Bar:** d:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python\Prue...
- Menu Bar:** File Edit Selection Find View Goto Tools Project Preferences Help
- Code Area:** A tab labeled "PruebaTupla.py" contains the following Python code:

```
1 tupla_colores = ("Rojo", "Amarillo", "Azul")
2
3 print(tupla_colores)
4
5 print(tupla_colores*2)
6
```
- Output Area:** Displays the execution results:

```
('Rojo', 'Amarillo', 'Azul')
('Rojo', 'Amarillo', 'Azul', 'Rojo', 'Amarillo', 'Azul')
[Finished in 0.1s]
```
- Status Bar:** Shows "Line 1, Column 14" on the left, "Tab Size: 4" in the center, and "Python" on the right.

Si nos interesa generar una lista a partir de una tupla podemos recurrir a la instrucción list:

The screenshot shows a Python code editor window with the following details:

- Title Bar:** d:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python\Prue...
- Menu Bar:** File Edit Selection Find View Goto Tools Project Preferences Help
- Code Area:** A tab labeled "PruebaTupla.py" contains the following Python code:

```
1 tupla_colores = ("Rojo", "Amarillo", "Azul")
2
3 lista_colores = list(tupla_colores)
4
5 print(lista_colores)
6
```
- Output Area:** Displays the execution results:

```
['Rojo', 'Amarillo', 'Azul']
[Finished in 0.1s]
```
- Status Bar:** Shows "Line 5, Column 21" on the left, "Tab Size: 4" in the center, and "Python" on the right.

Y, por supuesto, podemos convertir al revés:

The screenshot shows a Python code editor window. The title bar says "d:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python\Prue...". The menu bar includes File, Edit, Selection, Find, View, Goto, Tools, Project, Preferences, and Help. The code editor tab is titled "PruebaTupla.py". The code itself is:

```
1 lista_colores = ["Rojo", "Amarillo", "Azul"]
2 tupla_colores = tuple(lista_colores)
3 print(lista_colores)
4 print(tupla_colores)
5
```

The output window below shows the results of running the script:

```
['Rojo', 'Amarillo', 'Azul']
('Rojo', 'Amarillo', 'Azul')
[Finished in 0.5s]
```

At the bottom, it indicates "Line 1, Column 1" and "Tab Size: 4" and "Python".

Un método que podemos aplicar a las tuplas es count y se usa exactamente para contar cuantas veces aparece un determinado valor entre los elementos de la tupla:

The screenshot shows a Python code editor window with the same setup as the previous one. The code editor tab is still "PruebaTupla.py". The code has been modified at line 4:

```
1 lista_colores = ["Rojo", "Amarillo", "Azul"]
2 tupla_colores = tuple(lista_colores)
3 print(lista_colores)
4 print(tupla_colores.count("Rojo"))
5
```

The output window shows the results:

```
['Rojo', 'Amarillo', 'Azul']
1
[Finished in 0.1s]
```

At the bottom, it indicates "Line 4, Column 35" and "Tab Size: 4" and "Python".

Y, por las dudas probamos, count funciona también con las listas:

The screenshot shows a Python code editor window titled 'PruebaTupla.py'. The code defines a list 'lista_colores' containing three elements: 'Rojo', 'Amarillo', and 'Azul'. It then converts this list into a tuple 'tupla_colores' using the built-in tuple() function. Finally, it prints both the list and the tuple, followed by the count of the element 'Rojo' in the list.

```
1 lista_colores = ["Rojo", "Amarillo", "Azul"]
2 tupla_colores = tuple(lista_colores)
3 print(lista_colores)
4 print(lista_colores.count("Rojo"))
5

['Rojo', 'Amarillo', 'Azul']
1
[Finished in 0.3s]
```

Para averiguar cuántos elementos tenemos en una tupla recurrimos a len:

The screenshot shows a Python code editor window titled 'PruebaTupla.py'. The code defines a list 'lista_colores' containing three elements: 'Rojo', 'Amarillo', and 'Azul'. It then converts this list into a tuple 'tupla_colores' using the built-in tuple() function. After printing the list and tuple, it uses the len() function to print the length of both the list and the tuple, which both output the value 3.

```
1 lista_colores = ["Rojo", "Amarillo", "Azul"]
2 tupla_colores = tuple(lista_colores)
3 print(len(lista_colores))
4 print(len(tupla_colores))
5

3
3
[Finished in 0.1s]
```

Y, como podíamos esperar len funciona también para las listas.

Si queremos crear una tupla con un único elemento (se las conoce como tuplas unitarias) recurrimos a:

```

d:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python\Prue...
File Edit Selection Find View Goto Tools Project Preferences Help
PruebaTupla.py x
1 tupla_colores = ("Rojo",)
2 lista_colores = ["Rojo",]
3 print(len(tupla_colores))
4 print(len(lista_colores))
5

1
1
[Finished in 0.3s]

Line 4, Column 26 Tab Size: 4 Python

```

Y, otra vez, funciona tanto para tuplas como para listas.

Algo muy interesante nos permite distribuir el contenido de una tupla entre varias variables, lo que conocemos como el desempaquetado de una tupla:

```

d:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python\Prue...
File Edit Selection Find View Goto Tools Project Preferences Help
PruebaTupla.py x
1 tupla_colores = ("Rojo", "Amarillo", 2)
2 lista_colores = ["Rojo", "Amarillo", 2]
3 a,b,c = tupla_colores
4 print(a,b,c)
5 a,b,c = lista_colores
6 print(a,b,c)

Rojo Amarillo 2
Rojo Amarillo 2
[Finished in 0.1s]

Line 6, Column 13 Tab Size: 4 Python

```

Y, verificamos, que nos funciona tanto con tuplas como con listas.

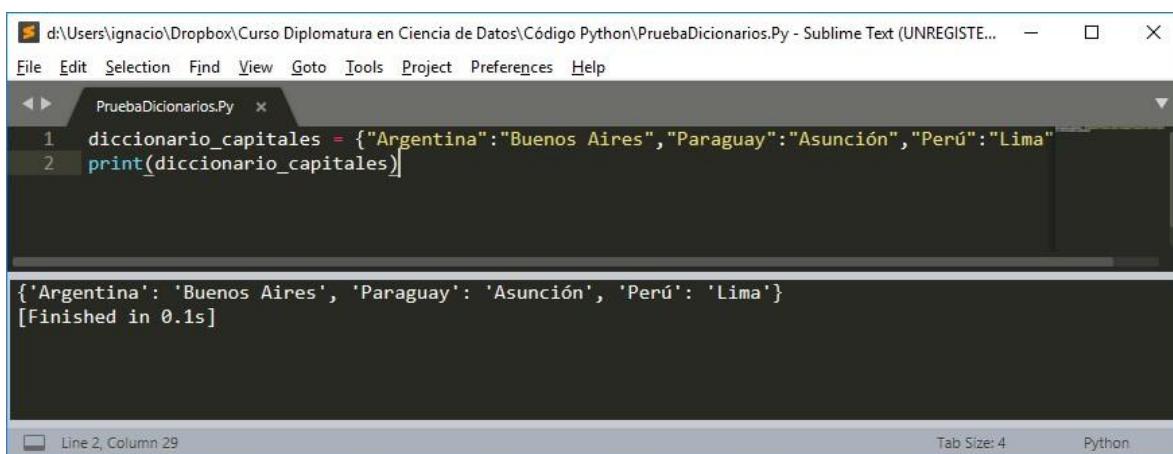
Diccionarios:

Los diccionarios nos permiten almacenar datos de diferente tipo, como las listas y las tuplas que ya hemos visto.

Sin embargo, introducen una novedad pues cada elemento se introduce asociado a una clave.

En las tuplas y listas el orden entre los elementos era parte de la información almacenada. Esto no cuenta para los diccionarios.

Las claves no pueden repetirse dentro del diccionario. Jugarán el mismo rol que en las listas y tuplas jugaba la posición y nos permitirán identificar y encontrar los elementos.



A screenshot of a Sublime Text editor window. The title bar says "d:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python\PruebaDiccionarios.Py - Sublime Text (UNREGISTERED)". The menu bar includes File, Edit, Selection, Find, View, Goto, Tools, Project, Preferences, and Help. The status bar at the bottom shows "Line 2, Column 29", "Tab Size: 4", and "Python". The code in the editor is:

```
1 diccionario_capitales = {"Argentina": "Buenos Aires", "Paraguay": "Asunción", "Perú": "Lima"}  
2 print(diccionario_capitales)
```

The output in the console below the editor is:

```
{'Argentina': 'Buenos Aires', 'Paraguay': 'Asunción', 'Perú': 'Lima'}  
[Finished in 0.1s]
```

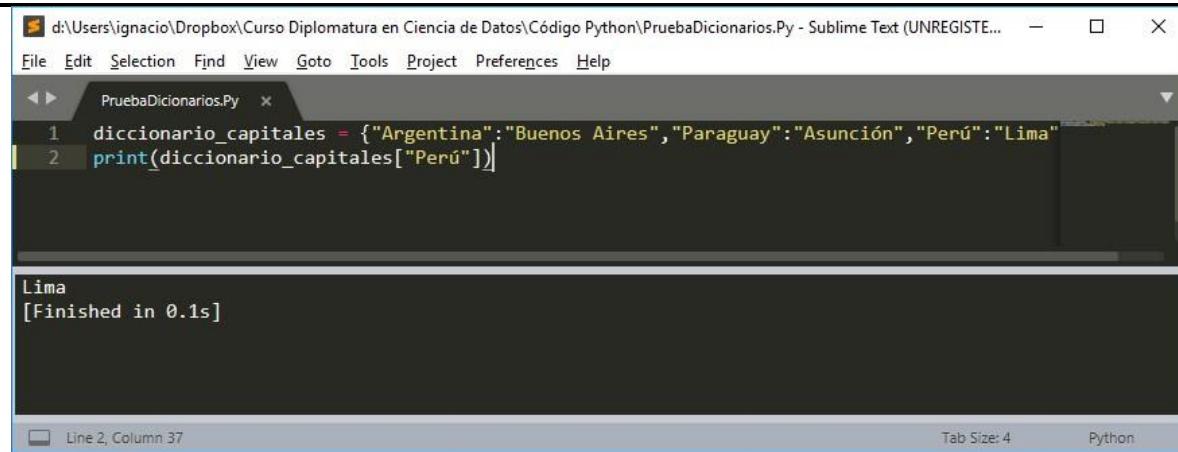
Cada parámetro clave:valor se separa del siguiente por una coma, tal y como separábamos los elementos de listas y tuplas.

Encerrando todos los pares clave valor usamos las {} cuando antes usamos los () para las tuplas y los [] para las listas.

Finalmente usamos el signo : para separar cada clave de su valor.

Tanto las claves como sus valores pueden ser de cualquier tipo.

Para acceder al valor almacenado en la clave:



```
d:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python\PruebaDiccionarios.py - Sublime Text (UNREGISTERED)

File Edit Selection Find View Goto Tools Project Preferences Help

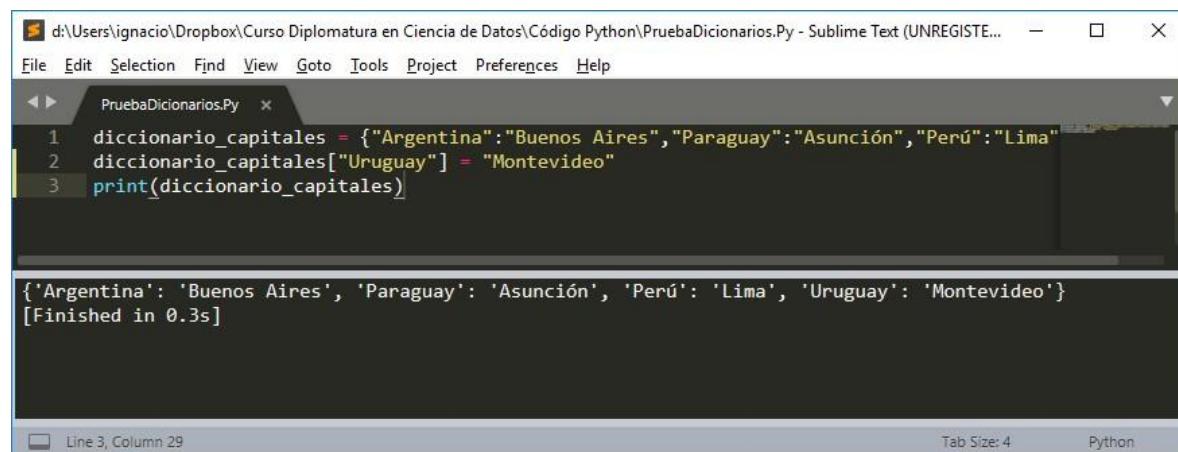
PruebaDiccionarios.py x

1 diccionario_capitales = {"Argentina": "Buenos Aires", "Paraguay": "Asunción", "Perú": "Lima"}
2 print(diccionario_capitales["Perú"])

Lima
[Finished in 0.1s]

Line 2, Column 37 Tab Size: 4 Python
```

Para agregar elementos usamos:



```
d:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python\PruebaDiccionarios.py - Sublime Text (UNREGISTERED)

File Edit Selection Find View Goto Tools Project Preferences Help

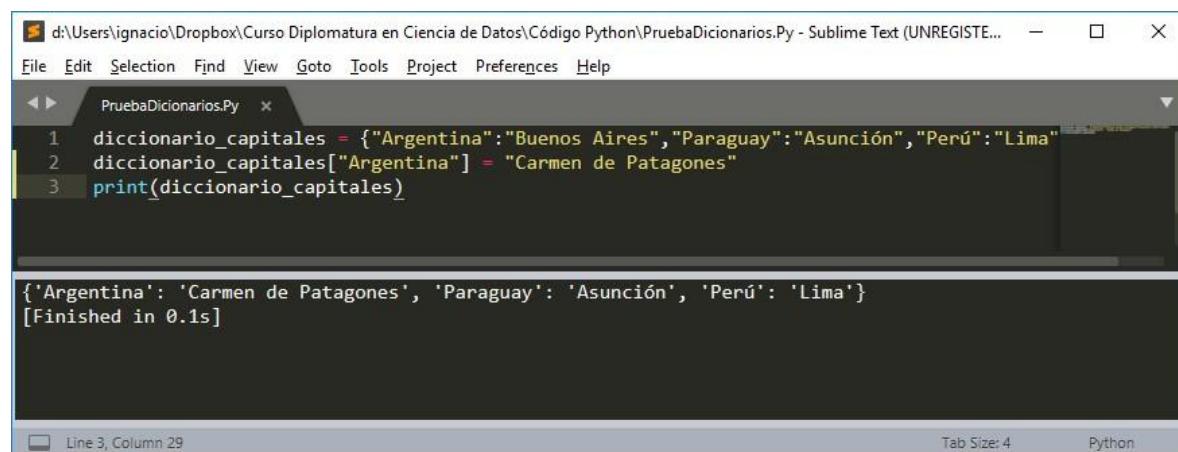
PruebaDiccionarios.py x

1 diccionario_capitales = {"Argentina": "Buenos Aires", "Paraguay": "Asunción", "Perú": "Lima"}
2 diccionario_capitales["Uruguay"] = "Montevideo"
3 print(diccionario_capitales)

{'Argentina': 'Buenos Aires', 'Paraguay': 'Asunción', 'Perú': 'Lima', 'Uruguay': 'Montevideo'}
[Finished in 0.3s]

Line 3, Column 29 Tab Size: 4 Python
```

Para corregir el valor de un elemento, por ejemplo, si Argentina hubiera mudado su capital a Carmen de Patagones:



```
d:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python\PruebaDiccionarios.py - Sublime Text (UNREGISTERED)

File Edit Selection Find View Goto Tools Project Preferences Help

PruebaDiccionarios.py x

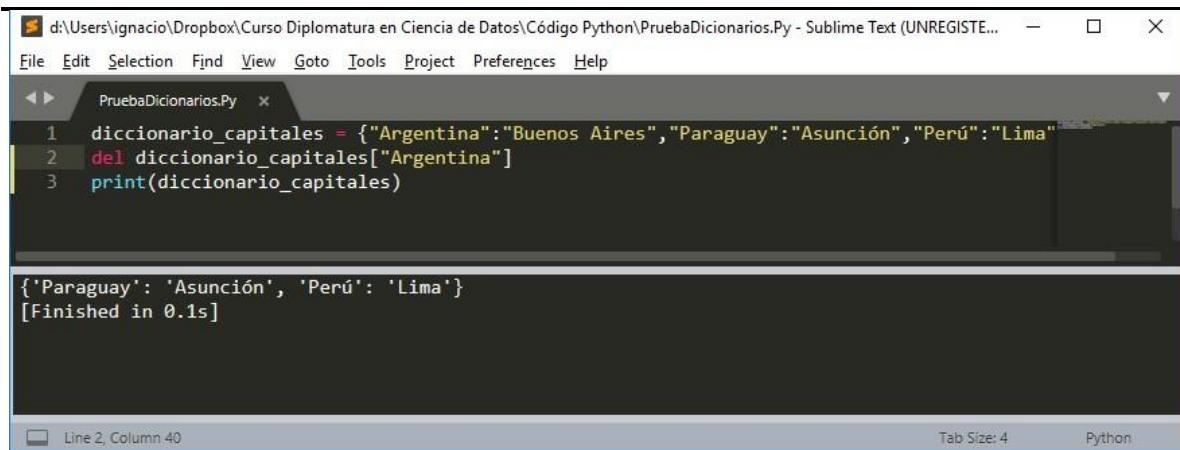
1 diccionario_capitales = {"Argentina": "Buenos Aires", "Paraguay": "Asunción", "Perú": "Lima"}
2 diccionario_capitales["Argentina"] = "Carmen de Patagones"
3 print(diccionario_capitales)

{'Argentina': 'Carmen de Patagones', 'Paraguay': 'Asunción', 'Perú': 'Lima'}
[Finished in 0.1s]

Line 3, Column 29 Tab Size: 4 Python
```

Cuando asignamos un valor a una misma clave ese valor reemplaza el anterior. Nunca se crean claves repetidas.

Para eliminar elementos de un diccionario usamos el método del:



```
d:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python\PruebaDiccionarios.Py - Sublime Text (UNREGISTERED)

File Edit Selection Find View Goto Tools Project Preferences Help

PruebaDiccionarios.Py x

1 diccionario_capitales = {"Argentina": "Buenos Aires", "Paraguay": "Asunción", "Perú": "Lima"}
2 del diccionario_capitales["Argentina"]
3 print(diccionario_capitales)

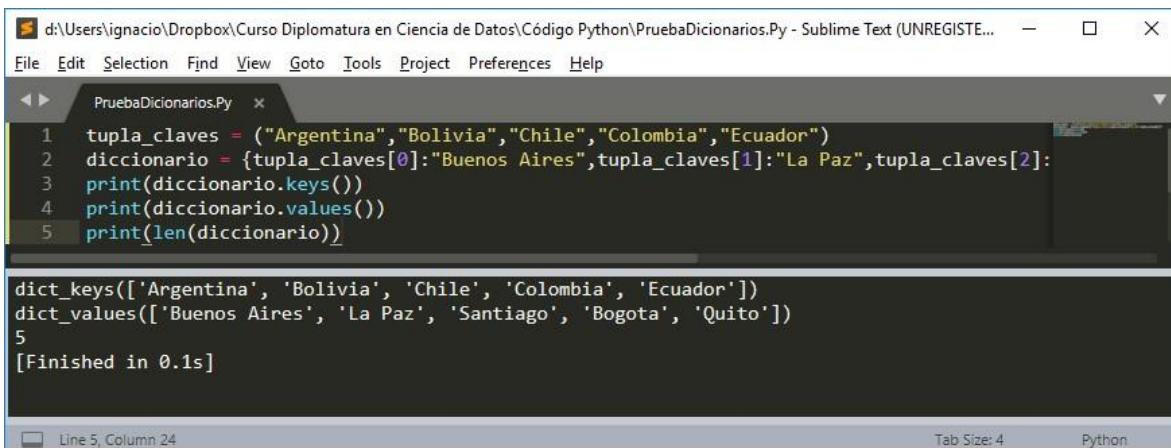
{'Paraguay': 'Asunción', 'Perú': 'Lima'}
[Finished in 0.1s]

Line 2, Column 40 Tab Size: 4 Python
```

Otros tres métodos útiles para los diccionarios son:

- Keys: que nos devuelve las claves
- Values: que nos devuelve los valores
- Len: que nos devuelve la longitud

Por ejemplo:



```
d:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python\PruebaDiccionarios.Py - Sublime Text (UNREGISTERED)

File Edit Selection Find View Goto Tools Project Preferences Help

PruebaDiccionarios.Py x

1 tupla_claves = ("Argentina", "Bolivia", "Chile", "Colombia", "Ecuador")
2 diccionario = {tupla_claves[0]: "Buenos Aires", tupla_claves[1]: "La Paz", tupla_claves[2]: "Santiago", tupla_claves[3]: "Bogota", tupla_claves[4]: "Quito"}
3 print(diccionario.keys())
4 print(diccionario.values())
5 print(len(diccionario))

dict_keys(['Argentina', 'Bolivia', 'Chile', 'Colombia', 'Ecuador'])
dict_values(['Buenos Aires', 'La Paz', 'Santiago', 'Bogota', 'Quito'])
5
[Finished in 0.1s]

Line 5, Column 24 Tab Size: 4 Python
```

Condicionales:

Como ya hemos comentado más arriba la ejecución de los programas va recorriendo el código pasando de cada instrucción a la siguiente. Este orden se llama gravitación.

Por supuesto que hay formas de alterar ese orden. Una de esas formas es el uso de instrucciones condicionales como if:

```

d:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python\PruebaCondicional.py - Sublime Text (UNREGISTERED)

File Edit Selection Find View Goto Tools Project Preferences Help

▶ PruebaCondicional.py x
1 a = 3
2 if a>1:
3     print("mayor que 1")
4 else:
5     print("menor o igual que 1")
6 print("Todo el mundo por acá")
7

mayor que 1
Todo el mundo por acá
[Finished in 0.1s]

```

Line 7, Column 1 Tab Size: 4 Python

Primero se ejecuta `a=3` y guardamos 3 dentro de la variable `a`.

Luego el programa pregunta si el contenido de `a` es mayor que 1. Esto resulta cierto entonces ejecuta el bloque indentado que está a continuación del `if`.

Para ver qué ocurre si `a` no es mayor a 1:

```

d:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python\PruebaCondicional.py - Sublime Text (UNREGISTERED)

File Edit Selection Find View Goto Tools Project Preferences Help

▶ PruebaCondicional.py x
1 a = 0
2 if a>1:
3     print("mayor que 1")
4 else:
5     print("menor o igual que 1")
6 print("Todo el mundo por acá")
7

menor o igual que 1
Todo el mundo por acá
[Finished in 0.1s]

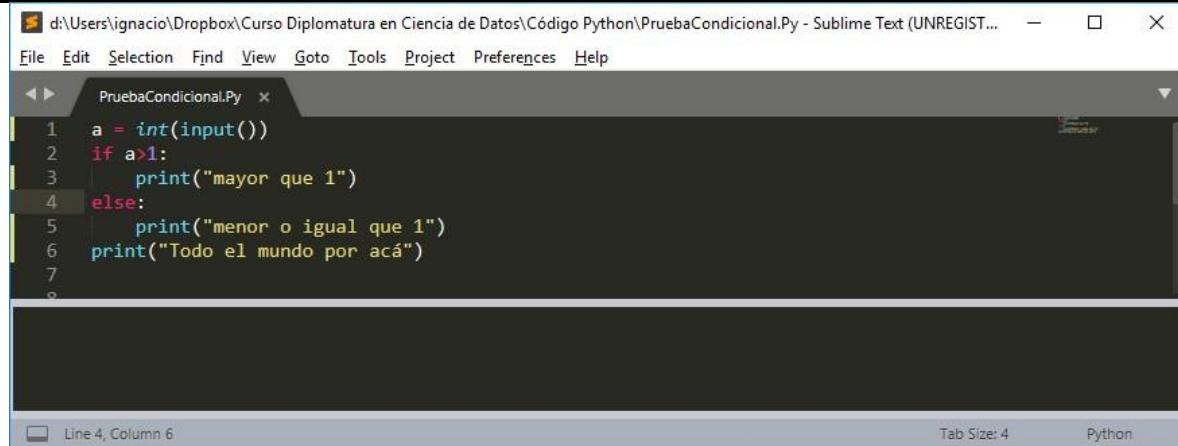
```

Line 1, Column 6 Tab Size: 4 Python

Vemos que en ambos casos imprime “`Todo el mundo por acá`”

A diferencia de lo que ocurre en otros lenguajes la forma de marcar los bloques en Python es con la indentación que así se vuelve parte esencial del código.

Vamos a hacer un pequeño cambio para que el dato que vamos a comparar con 1 sea introducido desde el teclado:



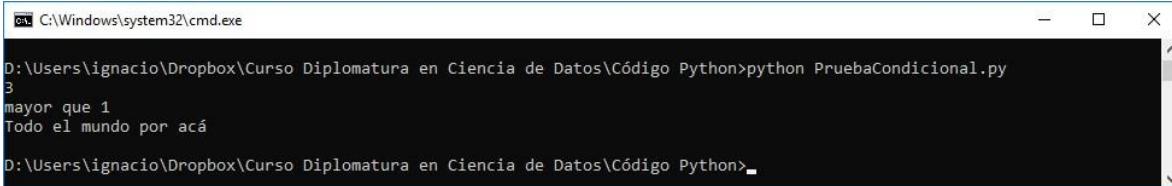
```

d:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python\PruebaCondicional.py - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
PruebaCondicional.py x
1 a = int(input())
2 if a>1:
3     print("mayor que 1")
4 else:
5     print("menor o igual que 1")
6 print("Todo el mundo por acá")
7
8
Line 4, Column 6
Tab Size: 4
Python

```

Por defecto el resultado de input es una cadena de caracteres. Para compararlo con 1 necesitamos que sea un valor numérico. Por eso utilizamos la función int para convertirlo a un entero.

El resultado al ejecutar es:



```

C:\Windows\system32\cmd.exe
D:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python>python PruebaCondicional.py
3
mayor que 1
Todo el mundo por acá
D:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python>

```

Si queremos ejecutar una serie de decisiones en cascada podemos recurrir a la instrucción elif.

Por ejemplo, para calificar los cursos utilizo la siguiente tabla:

Nota en escala 10	Concepto que figura en el certificado
10	Sobresaliente
Menor a 10 pero mayor o igual a 8.5	Excelente
Menor a 8.5 pero mayor o igual a 7.5	Muy bueno
Menor a 7.5 pero mayor o igual a 6.5	Bueno
Mayor o igual a 5.5 pero menor que 6.5	Aprobó
Menor a 5.5	Participó

Ahora vamos a generar una función que llamaremos evaluar para que haga este trabajo como forma de exemplificar el uso de elif:

```

d:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python\Prueb elif.py - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
Prueb elif.py
1 a = float(input("Nota: "))
2
3 def evaluar(n):
4     if n == 10:
5         c = "Sobresaliente"
6         return c
7     elif n>= 8.5:
8         c = "Excelente"
9         return c
10    elif n>= 7.5:
11        c = "Muy bueno"
12        return c
13    elif n>= 6.5:
14        c = "Bueno"
15        return c
16    elif n >= 5.5:
17        c = "Aprobado"
18        return c
19    elif n >= 0:
20        c = "Participó"
21        return c
22    else:
23        c = "Indefinido"
24        return c
25
26
27 print(evaluar(a))

```

Line 6, Column 17 Tab Size: 4 Python

Y voy probando la ejecución:

```

C:\Windows\system32\cmd.exe
D:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python>python PruebaElif.py
Nota: 10
Sobresaliente
D:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python>

```

```

C:\Windows\system32\cmd.exe
D:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python>python PruebaElif.py
Nota: 9.9
Excelente
D:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python>

```

```

C:\Windows\system32\cmd.exe
D:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python>python PruebaElif.py
Nota: 8.4
Muy bueno
D:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python>

```

```

C:\Windows\system32\cmd.exe
D:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python>python PruebaElif.py
Nota: 6.4
Aprobado
D:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python>

```

```
C:\Windows\system32\cmd.exe
D:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python>python PruebaElif.py
Nota: 5.4
Participó
D:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python>
```

```
C:\Windows\system32\cmd.exe
D:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python>python PruebaElif.py
Nota: 0
Participó
D:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python>
```

Y varios casos patológicos:

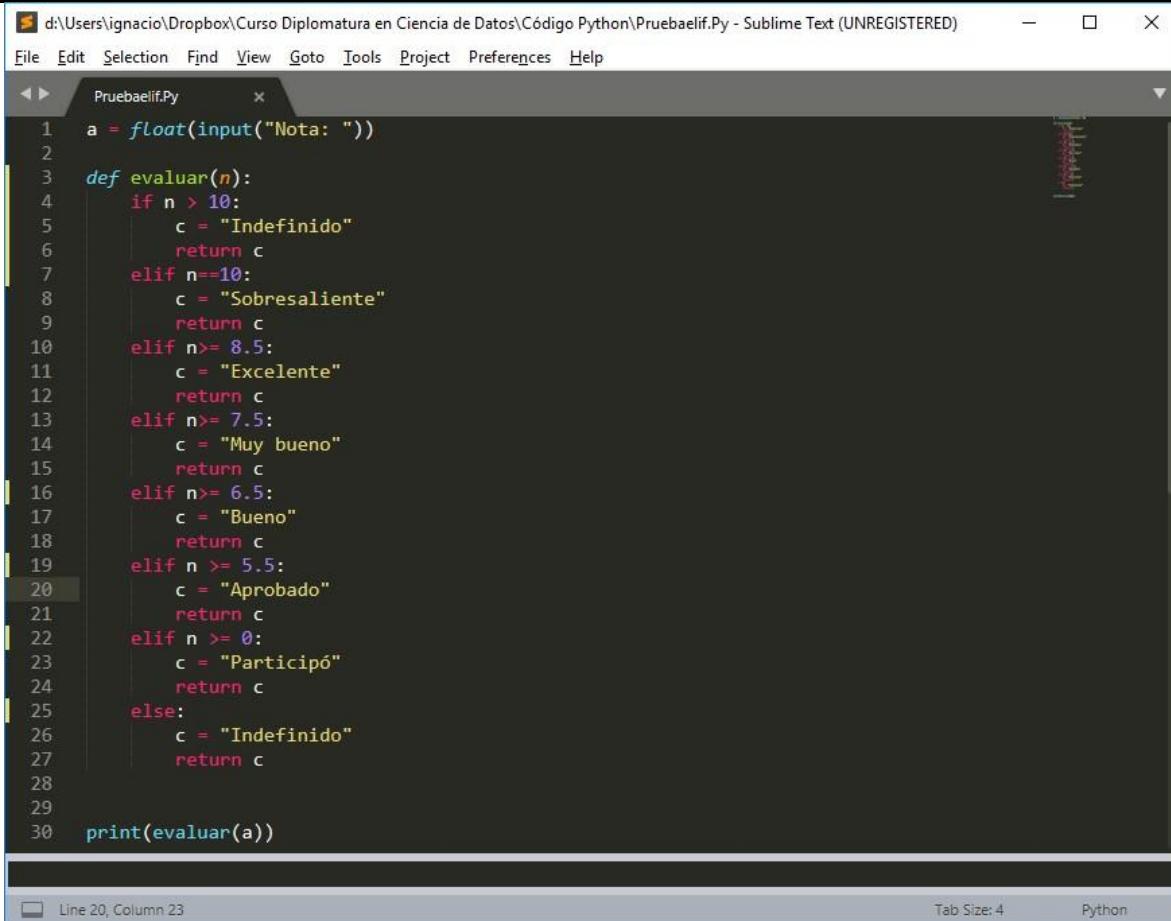
```
C:\Windows\system32\cmd.exe
D:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python>python PruebaElif.py
Nota: -1
Indefinido
D:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python>
```

```
C:\Windows\system32\cmd.exe
D:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python>python PruebaElif.py
Nota: 11
Excelente
D:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python>
```

```
C:\Windows\system32\cmd.exe
Nota: se copió
Traceback (most recent call last):
  File "PruebaElif.py", line 1, in <module>
    a = float(input("Nota: "))
ValueError: could not convert string to float: 'se copió'
D:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python>
```

Conviene hacer notar que para probar el programa necesitamos mínimamente que cada pieza de código se ejecute al menos una vez como hemos tratado de hacer aquí.

Sin embargo el comportamiento no deseado que ocurrió con el valor 11 necesita ser corregido:



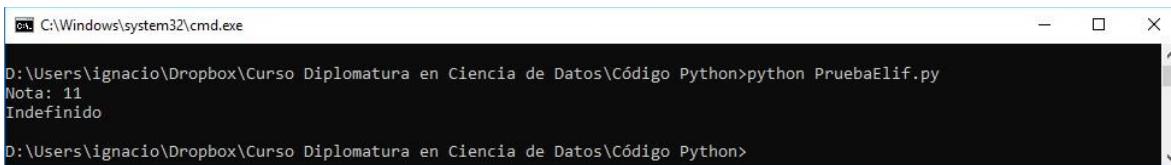
```

d:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python\Prueb elif.py - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
Prueb elif.py
1 a = float(input("Nota: "))
2
3 def evaluar(n):
4     if n > 10:
5         c = "Indefinido"
6         return c
7     elif n==10:
8         c = "Sobresaliente"
9         return c
10    elif n>= 8.5:
11        c = "Excelente"
12        return c
13    elif n>= 7.5:
14        c = "Muy bueno"
15        return c
16    elif n>= 6.5:
17        c = "Bueno"
18        return c
19    elif n >= 5.5:
20        c = "Aprobado"
21        return c
22    elif n >= 0:
23        c = "Participó"
24        return c
25    else:
26        c = "Indefinido"
27        return c
28
29
30 print(evaluar(a))

```

Line 20, Column 23 Tab Size: 4 Python

Y al ejecutar el caso patológico



```

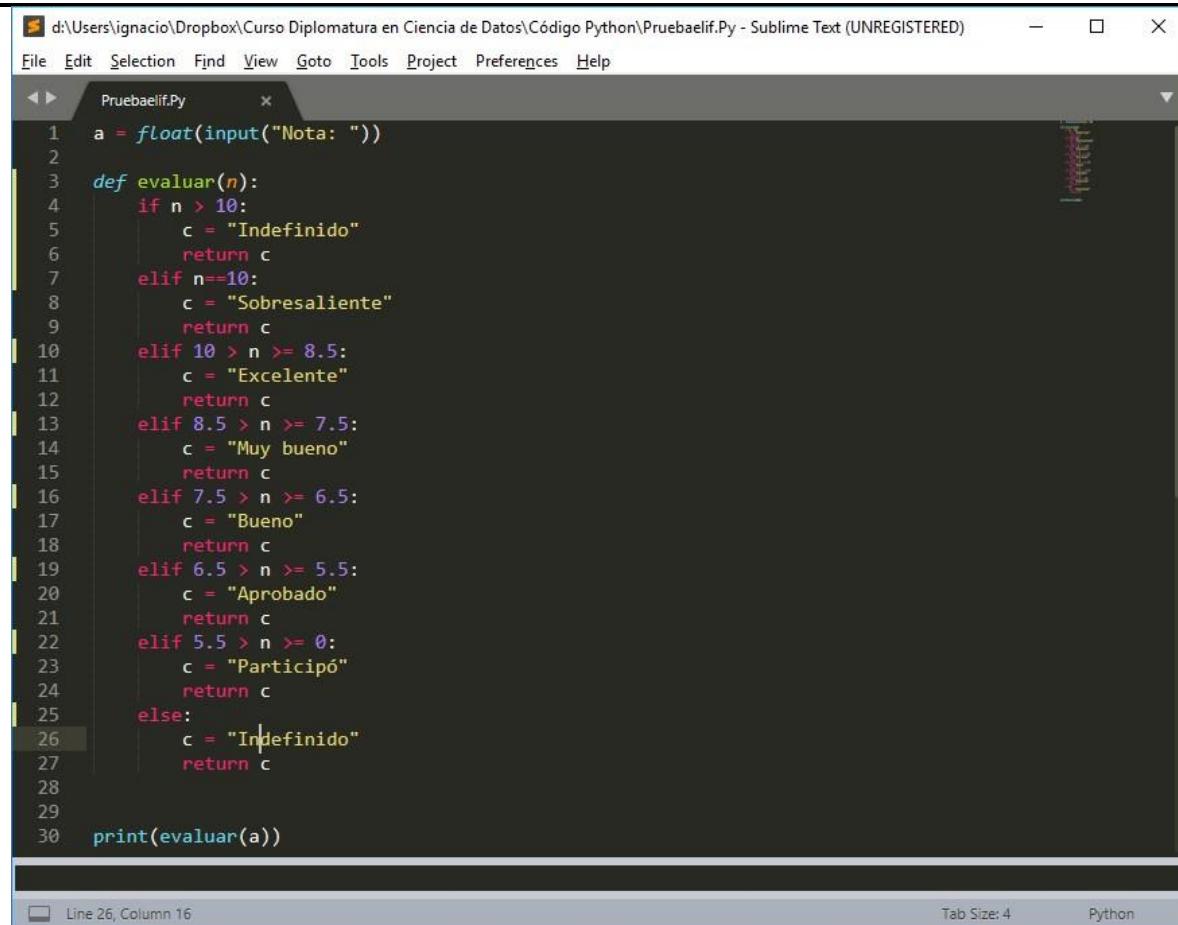
C:\Windows\system32\cmd.exe
D:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python>python PruebaElif.py
Nota: 11
Indefinido
D:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python>

```

Empezamos a descubrir que probar un programa es algo más difícil de lo que nos habíamos imaginado...

Una facilidad que nos ofrece Python es la concatenación de operadores dentro de la instrucción if

Para el programa anterior podríamos haber escrito:



```

d:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python\Pruebaelif.py - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
Pruebaelif.py
1 a = float(input("Nota: "))
2
3 def evaluar(n):
4     if n > 10:
5         c = "Indefinido"
6         return c
7     elif n==10:
8         c = "Sobresaliente"
9         return c
10    elif 10 > n >= 8.5:
11        c = "Excelente"
12        return c
13    elif 8.5 > n >= 7.5:
14        c = "Muy bueno"
15        return c
16    elif 7.5 > n >= 6.5:
17        c = "Bueno"
18        return c
19    elif 6.5 > n >= 5.5:
20        c = "Aprobado"
21        return c
22    elif 5.5 > n >= 0:
23        c = "Participó"
24        return c
25    else:
26        c = "Indefinido"
27        return c
28
29
30 print(evaluar(a))

```

Line 26, Column 16 Tab Size: 4 Python

Al escribirlo de esta forma el programa se sigue comportando igual pero resulta más sencillo de leer.

Para que considere cumplida la condición concatenada cada una de las desigualdades debe cumplirse.

O sea escribir:

$$A < B < C$$

Es equivalente a:

$$A < B \text{ AND } B < C$$

Con lo cual nuestro programa quedaría:

```

d:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python\Prueb elif.py - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
Prueb elif.py
1 a = float(input("Nota: "))
2
3 def evaluar(n):
4     if n > 10:
5         c = "Indefinido"
6         return c
7     elif n==10:
8         c = "Sobresaliente"
9         return c
10    elif 10 > n and n >= 8.5:
11        c = "Excelente"
12        return c
13    elif 8.5 > n and n >= 7.5:
14        c = "Muy bueno"
15        return c
16    elif 7.5 > n and n >= 6.5:
17        c = "Bueno"
18        return c
19    elif 6.5 > n and n >= 5.5:
20        c = "Aprobado"
21        return c
22    elif 5.5 > n and n >= 0:
23        c = "Participó"
24        return c
25    else:
26        c = "Indefinido"
27        return c
28
29
30 print(evaluar(a))

```

Line 25, Column 10 Tab Size: 4 Python

Otro operador útil es `in` y sirve para ver si un valor dado se encuentra en una lista o tupla:

```

d:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python\Pruebaln.py - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
Pruebaln.py
1 a = int(input("Número: "))
2
3 if a in (1,3,5):
4     print("Dentro")
5 else:
6     print("Fuera")
7
8

```

Line 8, Column 5 Tab Size: 4 Python

Lo ejecutamos:

```

C:\Windows\system32\cmd.exe
D:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python>python PruebaIn.py
Número: 4
Fuera
D:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python>

```

Y:

```
C:\Windows\system32\cmd.exe
D:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python>python PruebaIn.py
Número: 3
Dentro
D:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python>
```

Ciclos definidos e indefinidos

Ciclos:

La otra estructura de control muy popular son los ciclos. Sirven para repetir un conjunto de instrucciones una cantidad de veces definida o hasta que se cumpla una condición.

Vamos a encontrarnos con dos tipos de ciclos:

- Definidos
- Indefinidos

Los ciclos definidos son aquellos para los cuales, viendo la instrucción con la que se los invoca se puede saber cuántas veces se ejecutarán.

Los ciclos indefinidos son los que se repiten hasta que se satisface alguna condición. Estos ciclos pueden ejecutarse una vez, muchas veces, ninguna vez o pueden no detenerse nunca.

Empecemos por los ciclos definidos:

La sintaxis es:

`for variable in objeto a recorrer`

La variable en cuestión se va a ir moviendo a lo largo del objeto. El objeto puede ser una cadena de caracteres, una lista, una tupla o un rango

Empecemos por recorrer una lista:

```
d:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python\PruebaCicloDefinido.py - Sublime Text (UNREGIS... — □ ×
File Edit Selection Find View Goto Tools Project Preferences Help
PruebaCicloDefinido.py x
1 for i in [1,3,9]:
2     print(i)
3
[Finished in 0.1s]
Line 2, Column 13 Tab Size: 4 Python
```

Y obtenemos:

```
C:\Windows\system32\cmd.exe
D:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python>python PruebaCicloDefinido.Py
1
3
9

D:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python>
```

Vamos ahora a recorrer una cadena de caracteres:

```
d:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python\PruebaCicloDefinido.Py - Sublime Text (UNREGIS... — □ X
File Edit Selection Find View Goto Tools Project Preferences Help
PruebaCicloDefinido.Py x
1 for i in "caracteres":
2     print(i)
3
4
[Finished in 0.1s]
Line 2, Column 13 Tab Size: 4 Python
```

Y nos sale:

```
C:\Windows\system32\cmd.exe
D:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python>python PruebaCicloDefinido.Py
c
a
r
a
c
t
e
r
e
e
s

D:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python>
```

Probamos con un rango:

The screenshot shows a Sublime Text window with the following details:

- Title Bar:** d:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python\PruebaCicloDefinido.Py - Sublime Text (UNREGIS...)
- Menu Bar:** File Edit Selection Find View Goto Tools Project Preferences Help
- File List:** PruebaCicloDefinido.Py
- Code Area:**

```
1 a = int(input("Cuantas veces? "))
2 for i in range(a):
3     print(i)
4
5
```
- Status Bar:** [Finished in 0.1s]
- Bottom Status:** Line 2, Column 19 | Tab Size: 4 | Python

Y obtenemos:

```
C:\Windows\system32\cmd.exe
D:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python>python PruebaCicloDefinido.py
Cuantas veces? 5
0
1
2
3
4

D:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python>
```

Ahora podemos tratar de los ciclos indefinidos. La sintaxis es:

while condición:

Hagamos un ejemplo:

```

d:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python\PruebaCicloInDefinido.py - Sublime Text (UNREGISTE... — □ ×
File Edit Selection Find View Goto Tools Project Preferences Help
▶ PruebaCicloInDefinido.py ×
1 a = int(input("Cuantas veces? "))
2 while a>0:
3     print(a)
4     a = a -1
5
[Finished in 0.1s]
Line 5, Column 5
Tab Size: 4 Python

```

Va a ir restando de a 1 desde el valor de a que introduzcamos por teclado:

```

C:\Windows\system32\cmd.exe
D:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python>python PruebaCicloInDefinido.py
Cuantas veces? 5
5
4
3
2
1
D:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python>

```

Si necesitamos que, con independencia de la condición que encabeza el ciclo forcemos la salida del mismo podemos usar la instrucción break:

```

d:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python\PruebaWhileBreak.py - Sublime Text (UNREGISTE... — □ ×
File Edit Selection Find View Goto Tools Project Preferences Help
▶ PruebaWhileBreak.py ×
1 a = int(input("Cuantas veces? "))
2 while a>0:
3     print(a)
4     a = a -1
5     if a==3:
6         break
7
Line 6, Column 14
Tab Size: 4 Python

```

Y al ejecutar:

```

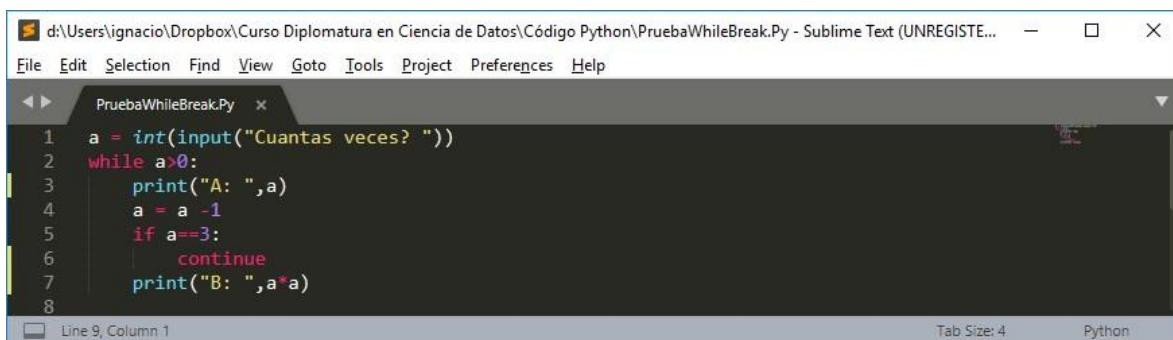
C:\Windows\system32\cmd.exe
D:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python>python PruebaWhileBreak.py
Cuantas veces? 6
6
5
4
D:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python>

```

La variable a comienza con un valor de 6 Pasa
a 5, 4 y a 3.

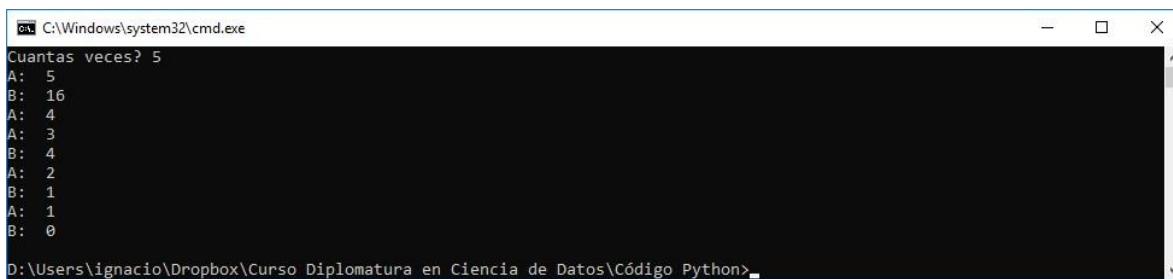
Cuando es igual a 3 cumple la condición del if y ejecuta el break.
El break interrumpe el ciclo y transfiere el control a la siguiente instrucción fuera del ciclo.
Como no hay instrucciones fuera del ciclo el programa termina.

Otra forma de controlar el flujo dentro de un ciclo es la instrucción continue que se encarga de volver a iniciar el ciclo:



```
d:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python\PruebaWhileBreak.py - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
PruebaWhileBreak.py
1 a = int(input("Cuantas veces? "))
2 while a>0:
3     print("A: ",a)
4     a = a -1
5     if a==3:
6         continue
7     print("B: ",a*a)
8
Line 9, Column 1
Tab Size: 4
Python
```

Cuando ejecutamos obtenemos:



```
C:\Windows\system32\cmd.exe
Cuantas veces? 5
A: 5
A: 16
A: 4
A: 3
A: 4
A: 2
A: 1
A: 1
B: 0
D:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python>
```

Introducimos el 5.

Inicia el ciclo, lo imprime, le resta 1 y queda 4. 4 no es igual a 3 por lo que no va al continue sino que imprime $4*4$ lo que da 16.

Como a sigue siendo mayor que cero lo imprime y aparece el 4. Le resta 1 y quedan 3.

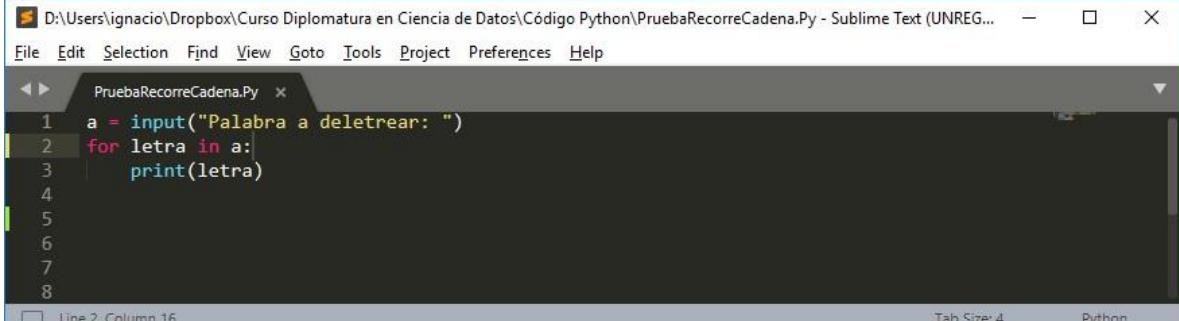
Ahora si va al continue por lo que no llega a imprimir $3*3$!

3 sigue siendo mayor que 0 por lo que retoma el ciclo, imprime el 3, le resta 1 y quedan 2.

Como 2 es distinto de 3 no entra en el if e imprime $2*2$ que es 4.

Y sigue repitiendo el ciclo hasta que la variable a se hace menor que cero.

También podemos usar los ciclos para recorrer cadenas de caracteres:



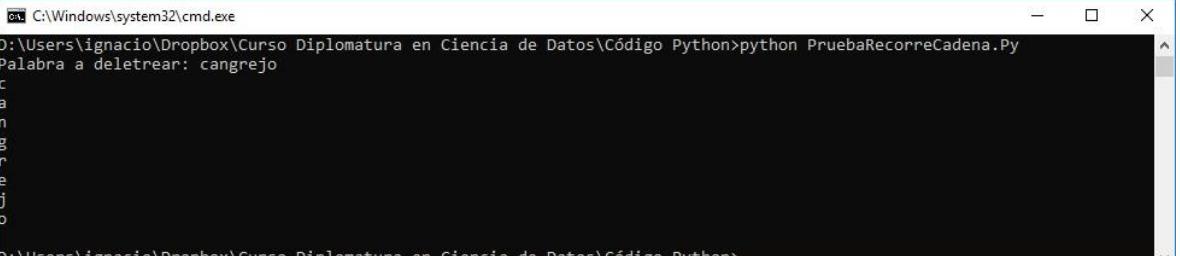
```
D:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python\PruebaRecorreCadena.Py - Sublime Text (UNREGISTERED)

File Edit Selection Find View Goto Tools Project Preferences Help

▶ PruebaRecorreCadena.Py ×
1 a = input("Palabra a deletrear: ")
2 for letra in a:
3     print(letra)
4
5
6
7
8

Line 2, Column 16                                     Tab Size: 4          Python
```

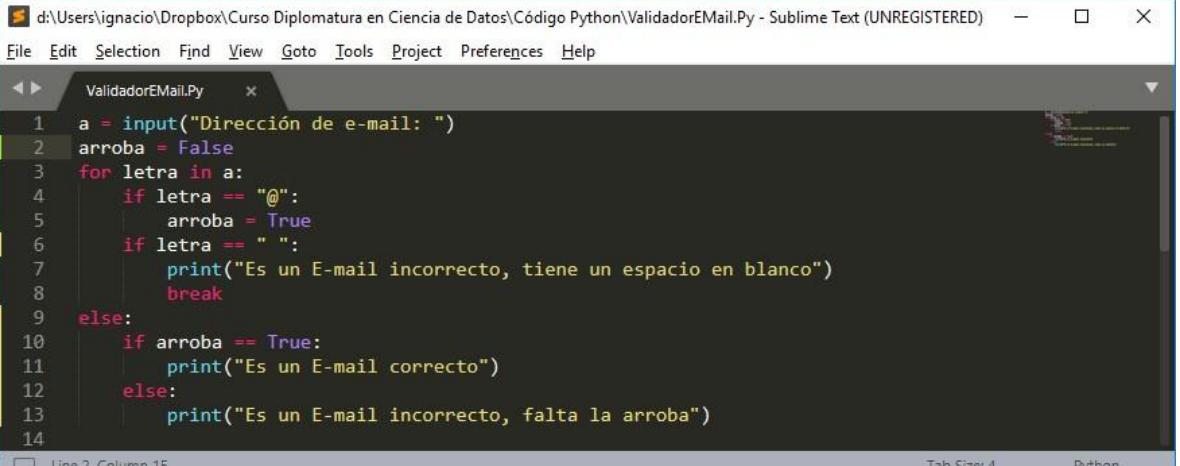
Y lo ejecutamos:



```
C:\Windows\system32\cmd.exe
D:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python>python PruebaRecorreCadena.py
Palabra a deletrear: cangrejo
c
a
n
g
r
e
j
o

D:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python>
```

Vamos a combinar estos conceptos para validar un E-mail:



```
d:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python\ValidadorEMail.Py - Sublime Text (UNREGISTERED)

File Edit Selection Find View Goto Tools Project Preferences Help

▶ ValidadorEMail.Py ×
1 a = input("Dirección de e-mail: ")
2 arroba = False
3 for letra in a:
4     if letra == "@":
5         arroba = True
6     if letra == " ":
7         print("Es un E-mail incorrecto, tiene un espacio en blanco")
8         break
9     else:
10        if arroba == True:
11            print("Es un E-mail correcto")
12        else:
13            print("Es un E-mail incorrecto, falta la arroba")
14

Line 2, Column 15                                     Tab Size: 4          Python
```

Lo ejecutamos varias veces:

```
C:\Windows\system32\cmd.exe
D:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python>python ValidadorEMail.py
Dirección de e-mail: abc@def
Es un E-mail correcto

D:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python>python ValidadorEMail.py
Dirección de e-mail: abcdef
Es un E-mail incorrecto, falta la arroba

D:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python>python ValidadorEMail.py
Dirección de e-mail: abc def
Es un E-mail incorrecto, tiene un espacio en blanco

D:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python>python ValidadorEMail.py
Dirección de e-mail: abc@d f
Es un E-mail incorrecto, tiene un espacio en blanco

D:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python>
```

Manejo de excepciones

Una excepción es un error que recién aparece en tiempo de ejecución. Tenemos un programa correctamente escrito, o sea su sintaxis es correcta, pero, al ejecutar, ocurre algo inesperado.

Puede tratarse de una división por cero, de no poder establecer control sobre un recurso que se necesita (por ejemplo memoria) o muchas, muchas otras circunstancias.

Las excepciones pueden manejarse. Esto es, en vez de que la excepción interrumpa la ejecución de nuestro programa volcando, a los ojos del usuario, un mensaje de error no elegante y escasamente comprensible podemos decidir seguir otro curso de acción.

Nos proponemos ver cómo hacemos para realizar este manejo de excepciones en Python.

Hagamos saltar un error:

```
d:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python\PruebaExcepciones.py - Sublime Text (UNRE... - File Edit Selection Find View Goto Tools Project Preferences Help
Fibonacci.py PruebaExcepciones.py
1 a = 5
2 b = 0
3 c = a/b
4 print("Última línea")
5 |
```

Line 5, Column 1 Tab Size: 4 Python

Y al ejecutar:

```
C:\Windows\system32\cmd.exe
D:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python>python PruebaExcepciones.Py
Traceback (most recent call last):
  File "PruebaExcepciones.Py", line 3, in <module>
    c = a/b
ZeroDivisionError: division by zero
D:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python>
```

Debemos notar dos cosas:

1. Nos da información sobre la línea y el archivo donde se produjo el error y el tipo de error del que se trata: división por cero. El código por el que Python identifica este error es: *ZeroDivisionError*
2. La ejecución del programa se detiene y NUNCA ejecuta la línea siguiente (y las que hubieran podido seguir) a la que cometió el error.

Para conseguir que la excepción producida por la división por cero quede manejada tenemos que utilizar un bloque try / except:

```
d:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python\PruebaExcepciones.Py - Sublime Text (UNRE... — □ ×
File Edit Selection Find View Goto Tools Project Preferences Help
Fibonaci.Py PruebaExcepciones.Py
1 a = 5
2 b = 0
3
4 try:
5     c = a/b
6 except ZeroDivisionError:
7     print("No se puede dividir por cero")
8
9 print("Ultima linea")
10
Line 1, Column 1 Tab Size: 4 Python
```

Antes de la instrucción que puede causar el error viene la instrucción try que abre el bloque que queremos intentar.

El bloque except deberá incluir los códigos de los errores que queremos controlar y, a continuación las instrucciones que queremos que se ejecuten si se diera la excepción.

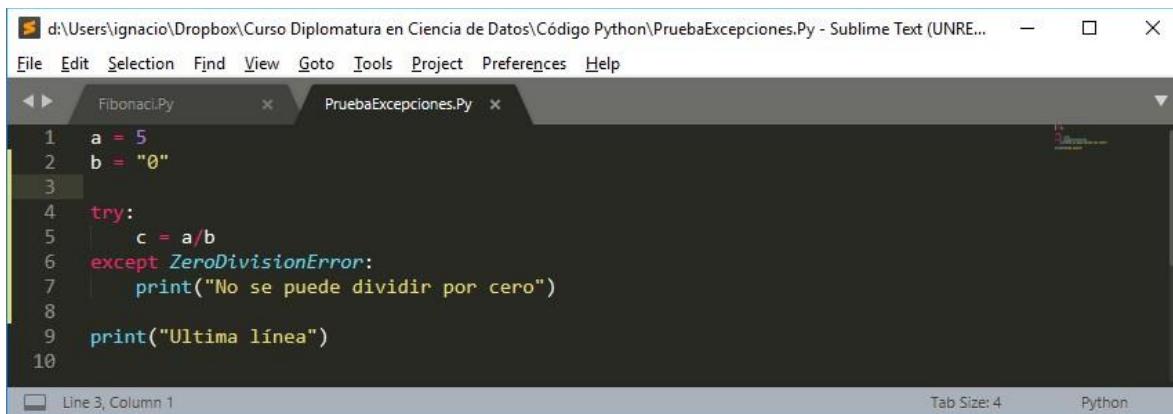
Cuando ejecutamos:

```
C:\Windows\system32\cmd.exe
ZeroDivisionError: division by zero
D:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python>python PruebaExcepciones.Py
No se puede dividir por cero
Ultima linea
D:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python>
```

Vemos que nos aparece el mensaje que está dentro del except y que luego aparece el mensaje correspondiente a que el programa sigue ejecutando normalmente las líneas que siguen tras la excepción.

Pero, para que esto funcione necesitamos que el código de error sea el correcto.

Imaginemos que nuestro programa dijera:



```

d:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python\PruebaExcepciones.py - Sublime Text (UNRE...)

File Edit Selection Find View Goto Tools Project Preferences Help

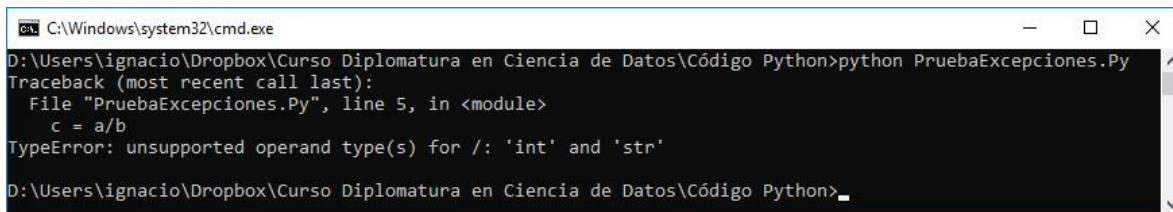
Fibonaci.py x PruebaExcepciones.py x

1 a = 5
2 b = "0"
3
4 try:
5     c = a/b
6 except ZeroDivisionError:
7     print("No se puede dividir por cero")
8
9 print("Ultima linea")
10

Line 3, Column 1 Tab Size: 4 Python

```

Y al ejecutar:



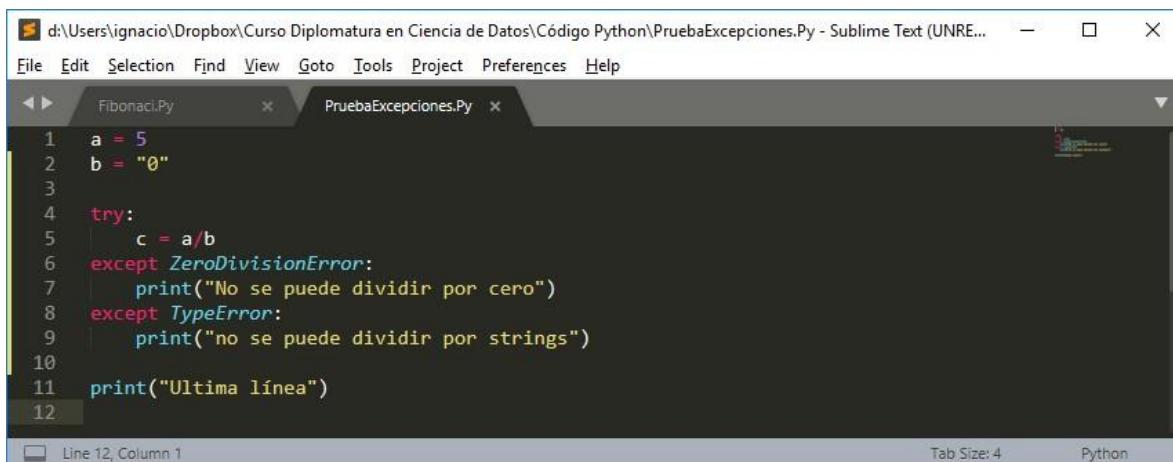
```

C:\Windows\system32\cmd.exe
D:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python>python PruebaExcepciones.py
Traceback (most recent call last):
  File "PruebaExcepciones.py", line 5, in <module>
    c = a/b
TypeError: unsupported operand type(s) for /: 'int' and 'str'
D:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python>

```

Como no estamos controlando el error TypeError nuestro manejo de excepción no aplica y el programa termina de manera anormal.

Si lo queremos controlar también:



```

d:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python\PruebaExcepciones.py - Sublime Text (UNRE...)

File Edit Selection Find View Goto Tools Project Preferences Help

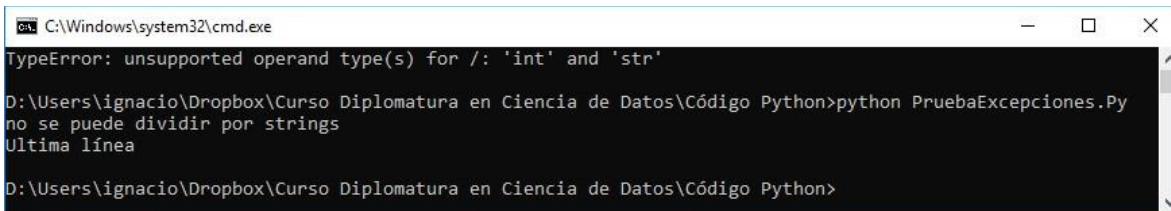
Fibonaci.py x PruebaExcepciones.py x

1 a = 5
2 b = "0"
3
4 try:
5     c = a/b
6 except ZeroDivisionError:
7     print("No se puede dividir por cero")
8 except TypeError:
9     print("no se puede dividir por strings")
10
11 print("Ultima linea")
12

Line 12, Column 1 Tab Size: 4 Python

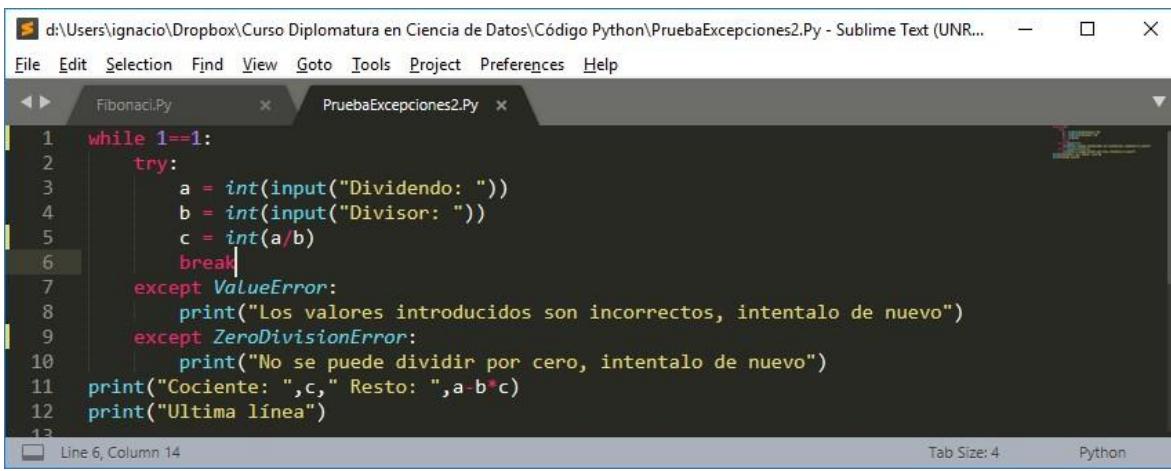
```

Y cuando ejecutamos:



```
C:\Windows\system32\cmd.exe
TypeError: unsupported operand type(s) for /: 'int' and 'str'
D:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python>python PruebaExcepciones.Py
no se puede dividir por strings
Ultima linea
D:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python>
```

Cuando necesitamos reintentar el código porque nuestro usuario introdujo un dato erróneo que necesitamos luego para realizar el propósito del programa utilizamos un ciclo infinito:

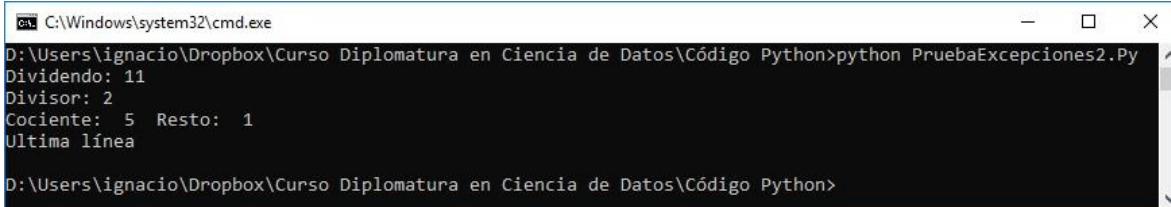


```
File Edit Selection Find View Goto Tools Project Preferences Help
Fibonacci.py x PruebaExcepciones2.py x
1 while 1==1:
2     try:
3         a = int(input("Dividendo: "))
4         b = int(input("Divisor: "))
5         c = int(a/b)
6         break
7     except ValueError:
8         print("Los valores introducidos son incorrectos, intentalo de nuevo")
9     except ZeroDivisionError:
10        print("No se puede dividir por cero, intentalo de nuevo")
11    print("Cociente: ",c, " Resto: ",a-b*c)
12    print("Ultima linea")
13
Line 6, Column 14 Tab Size: 4 Python
```

La única salida del ciclo infinito (while 1==1) es por el break. La única forma de ejecutar el break es que a, b y c se puedan calcular sin que aparezca ninguna excepción.

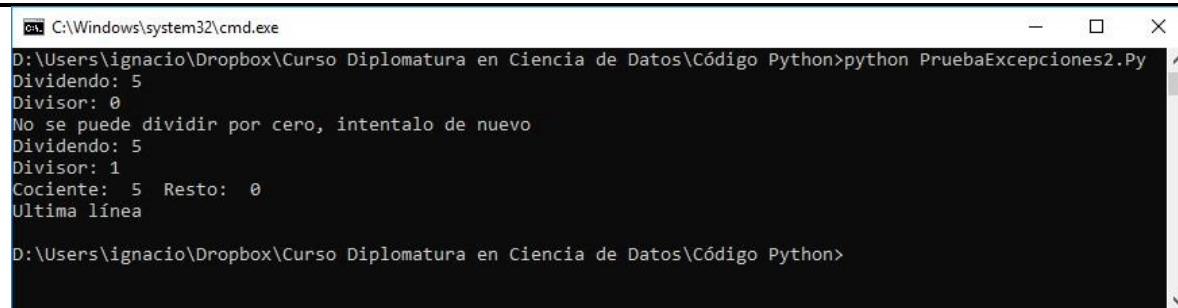
Si una de las dos excepciones que conocemos se producen nos da el mensaje correspondiente y, como seguimos dentro del ciclo infinito nos pide ingresar los datos de nuevo.

El resultado de una ejecución normal es:



```
C:\Windows\system32\cmd.exe
D:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python>python PruebaExcepciones2.Py
Dividendo: 11
Divisor: 2
Cociente: 5 Resto: 1
Ultima linea
D:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python>
```

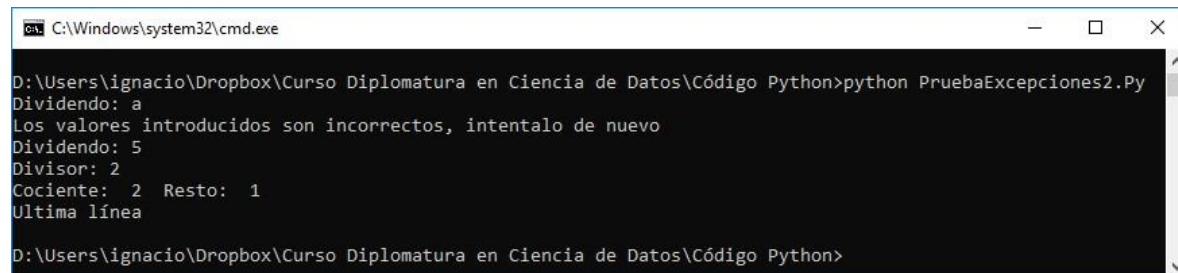
Vamos ahora a ver qué pasa con las ejecuciones con valores que hacen saltar las excepciones:



```
C:\Windows\system32\cmd.exe
D:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python>python PruebaExcepciones2.py
Dividendo: 5
Divisor: 0
No se puede dividir por cero, intentalo de nuevo
Dividendo: 5
Divisor: 1
Cociente: 5 Resto: 0
Ultima linea

D:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python>
```

Vemos que ejecutó sin problema los dos inputs pero que saltó la excepción de división por cero y nos llevó a cargar los valores de nuevo.



```
C:\Windows\system32\cmd.exe
D:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python>python PruebaExcepciones2.py
Dividendo: a
Los valores introducidos son incorrectos, intentalo de nuevo
Dividendo: 5
Divisor: 2
Cociente: 2 Resto: 1
Ultima linea

D:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python>
```

Si introduzco algo que no puede convertirse a número, en este caso la letra a no espera a pedirme el divisor, salta la excepción.

Si la excepción saltara al escribir el divisor nos volvería a pedir tanto el dividendo como el divisor.

Se podría tratar de que una vez admitida una pieza de información correcta, por ejemplo el dividendo, no se volviera a pedir el dato si el problema surge del valor cargado en el divisor. Dejamos esta adaptación del programa como ejercicio para los lectores.

Para asegurarnos que alguna operación se realiza con independencia de las excepciones que hayan surgido tenemos la instrucción `finally`:

```

1 while 1==1:
2     try:
3         a = int(input("Dividendo: "))
4         b = int(input("Divisor: "))
5         c = int(a/b)
6         break
7     except ValueError:
8         print("Los valores introducidos son incorrectos, intentalo de nuevo")
9     except ZeroDivisionError:
10        print("No se puede dividir por cero, intentalo de nuevo")
11    finally:
12        print("Siempre pasamos por aquí")
13
14 print("Cociente: ",c," Resto: ",a-b*c)
15 print("Ultima linea")
16

```

Line 14, Column 39 Tab Size: 4 Python

Y al ejecutar nos queda:

```

C:\Windows\system32\cmd.exe

D:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python>python PruebaExcepciones2.py
Dividendo: 5
Divisor: 1
Siempre pasamos por aquí
Cociente: 5 Resto: 0
Ultima linea

D:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python>

```

Si no tenemos la instrucción finally y tampoco los except y surge alguna condición de excepción la ejecución del programa termina de un modo anormal aunque hayamos incluido la instrucción try.

Por eso, si no queremos que la ejecución se caiga, con independencia de cuales pudieran ser las excepciones que ocurran no podemos dejar de incluir la instrucción finally.

No podemos incluir un try sin que tenga al menos un except o un finally.

También somos capaces de lanzar nuestras propias excepciones. Para eso vamos a utilizar la instrucción raise.

La instrucción raise nos va a servir para pasar información de la ocurrencia de una circunstancia excepcional desde el interior de una función hacia el programa que la ha llamado.

Vamos a importar la clase math, que viene predefinida en Python para que podamos calcular raíces cuadradas mediante el método sqrt:

```

d:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python\PruebaExcepciones2.Py - Sublime Text (UNR... — □ ×
File Edit Selection Find View Goto Tools Project Preferences Help
Fibonacci.Py PruebaExcepciones2.Py
1 import math
2
3 def interior(a):
4     if a<0:
5         raise TypeError("negativo")
6     else:
7         b = math.sqrt(a)
8     return b
9
10 c = float(input("Ingrese número: "))
11 try:
12     d = interior(c)
13     print("La raiz cuadrada de ",c, " es ",d)
14 except TypeError:
15     print("No hay raices cuadradas reales de números negativos")
16
17 print("Fin de programa")
Line 9, Column 1 Tab Size: 4 Python

```

Lo ejecutamos con 4 y:

```

C:\Windows\system32\cmd.exe
D:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python>python PruebaExcepciones2.Py
Ingrese número: 4
La raiz cuadrada de  4.0  es  2.0
Fin de programa
D:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python>

```

Ahora lo ejecutamos con -4

```

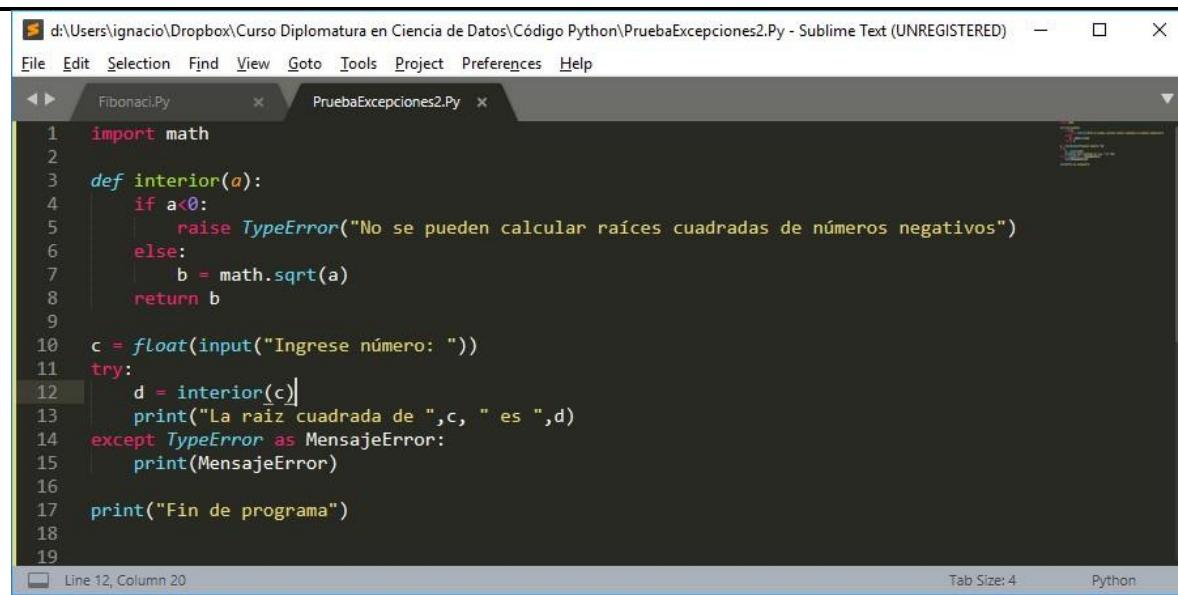
C:\Windows\system32\cmd.exe
La raiz cuadrada de  4.0  es  2.0
Fin de programa
D:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python>python PruebaExcepciones2.Py
Ingrese número: -4
No hay raices cuadradas reales de números negativos
Fin de programa
D:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python>

```

Vemos que se lanza la excepción desde dentro de la función, luego la capturamos desde el programa que llamó la función y mostramos el aviso que hemos preparado para este caso.

La función no termina normalmente de ejecutar pero el programa si lo hace.

Todavía podemos recuperar el mensaje de error que hemos generado desde adentro de la propia función. Para eso necesitamos darle un nombre a la excepción mediante la palabra reservada as:



A screenshot of the Sublime Text editor showing two tabs: 'Fibonaci.Py' and 'PruebaExcepciones2.Py'. The 'PruebaExcepciones2.Py' tab contains the following Python code:

```
import math

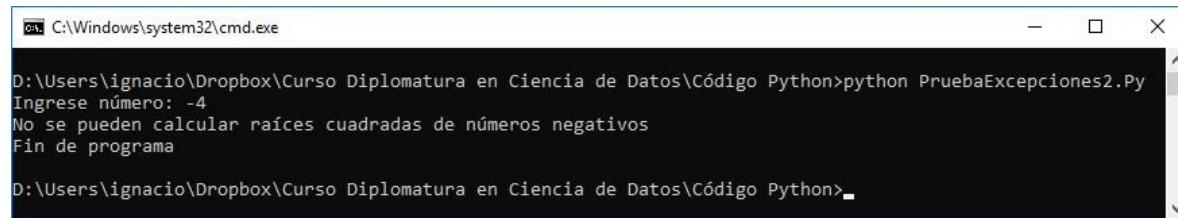
def interior(a):
    if a<0:
        raise TypeError("No se pueden calcular raíces cuadradas de números negativos")
    else:
        b = math.sqrt(a)
    return b

c = float(input("Ingrese número: "))
try:
    d = interior(c)
    print("La raíz cuadrada de ",c, " es ",d)
except TypeError as MensajeError:
    print(MensajeError)

print("Fin de programa")
```

The code defines a function 'interior' that calculates the square root of a number. It includes a check for negative numbers and a try-except block to handle the 'TypeError' that occurs when trying to take the square root of a negative number. The main part of the program prompts the user for a number, calls the function, and prints the result or an error message.

Lo ejecutamos con -4



A screenshot of a Windows Command Prompt window titled 'C:\Windows\system32\cmd.exe'. The command 'python PruebaExcepciones2.py' is run, followed by the input '-4'. The output shows the program catching the 'TypeError' and printing the error message 'No se pueden calcular raíces cuadradas de números negativos'. The command prompt then ends with 'Fin de programa'.

```
D:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python>python PruebaExcepciones2.py
Ingrese número: -4
No se pueden calcular raíces cuadradas de números negativos
Fin de programa
```

Vemos que nuestro programa principal nos imprime el mensaje de error que generamos dentro de la función.

Ese mensaje no viaja por la salida de la función sino por la excepción misma.

Ámbito de las variables

Las variables no son todas accesibles desde todas partes. Tienen un ámbito de acción dentro del cual están definidas. Tratemos de explorar el concepto:

```

d:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python\ambitos.py

File Edit Selection Find View Goto Tools Project Preferences Help

oop.py ambitos.py

1 a = 1
2
3 def suma(b,c):
4     a = 3
5     return b+c
6
7 print(a)
8 print(suma(a,a))
9 print(a)
10

1
2
1
[Finished in 0.1s]

```

Line 10, Column 1 Tab Size: 4 Python

La variable a comienza valiendo 1.

La definición de la función no afecta el valor de a, por lo menos hasta que no se ejecuta.

De hecho lo verificamos en que el primer número que se muestra por pantalla es un 1.

Luego nos muestra en el resultado de la función suma el 1+1 que da 2.

Luego tendría que mostrarnos el valor de a. ¿Habrá cambiado a 3? Tenemos certeza que cuando ejecutamos la función suma pasó por la línea que tenía el a=3 Sin embargo nos vuelve a aparecer un 1.

Es como si la variable a de adentro de la función fuera una variable distinta. A ver:

```

d:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python\ambitos.py

File Edit Selection Find View Goto Tools Project Preferences Help

oop.py ambitos.py

1 a = 1
2
3 def suma(b,c):
4     a = 3
5     print("Adentro ",a)
6     return b+c
7
8 print("antes ",a)
9 print("suma ",suma(a,a))
10 print("después ",a)
11
12

antes 1
Adentro 3
suma 2
después 1
[Finished in 0.1s]

```

Line 7, Column 1 Tab Size: 4 Python

Antes de entrar la función a vale uno.

Dentro de la función la otra a vale 3

Luego viene el 2 correspondiente a la suma Y,
finalmente, la a de afuera vale 3.

¿Y si no hubiera puesto a=3 dentro de la función?

```
d:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python\ambit... ━ ━ X
File Edit Selection Find View Goto Tools Project Preferences Help
oop.py ambitos.py
1 a = 1
2
3 def suma(b,c):
4     print("Adentro ",a)
5     return b+c
6
7
8 print("antes ",a)
9 print("suma ",suma(a,a))
10 print("después ",a)
11
12
antes 1
Adentro 1
suma 2
después 1
[Finished in 0.1s]
```

Line 4, Column 5 Tab Size: 4 Python

En ese caso, como no se encuentra una a con un valor definido dentro de la función toma el valor que tiene la a de afuera.

¿Y si dentro de la función definimos una nueva variable, por ejemplo, d?

```

d:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python\ambitos.py

File Edit Selection Find View Goto Tools Project Preferences Help
oop.py ambitos.py

1 a = 1
2
3 def suma(b,c):
4     d=4
5     print("Adentro ",a)
6     return b+c
7
8 print("antes ",a)
9 print("suma ",suma(a,a))
10 print("después ",a)
11 print("d vista afuera ",d)

File "d:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Dat

antes Traceback (most recent call last):
  File "d:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código
    Python\ambitos.py", line 11, in <module>
      1
Adentro 1
suma 2
después 1
      print("d vista afuera ",d)
NameError: name 'd' is not defined
[Finished in 0.1s with exit code 1]
[shell_cmd: python -u "d:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de
Datos\Código Python\ambitos.py"]
[dir: d:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código
Python]
[path: C:\Program Files (x86)\Common Files\Oracle\Java\javapath;C:\Windows\sys
tem32;C:\Windows;C:\Windows\System32\Wbem;C:\Windows\System32\WindowsPowerShell\v1
.0\;C:\Program Files (x86)\Microsoft SQL Server\140\Tools\Binn\;C:\Program
Files\Microsoft SQL Server\140\Tools\Binn\;C:\Program Files (x86)\Microsoft SQL
Server\140\DTSP\Binn\;C:\Program Files\Microsoft SQL
Server\140\DTSP\Binn\;C:\Program Files\Microsoft SQL Server\Client
SDK\ODBC\130\Tools\Binn\;C:\Program Files (x86)\Microsoft SQL Server\Client
SDK\ODBC\130\Tools\Binn\;C:\Program Files (x86)\Microsoft SQL Server\140\Tools\B

Line 11, Column 27
Tab Size: 4
Python

```

Al ejecutar obtenemos un mensaje de error que nos refiere que, vista desde afuera de la función la variable d no está definida.

¿Cómo interpretamos esto?

Una variable definida dentro del cuerpo principal del programa es visible en las funciones.

Una variable definida dentro de una función sólo es visible en la misma función y no existe fuera de ella.

Manejo de archivos y directorios

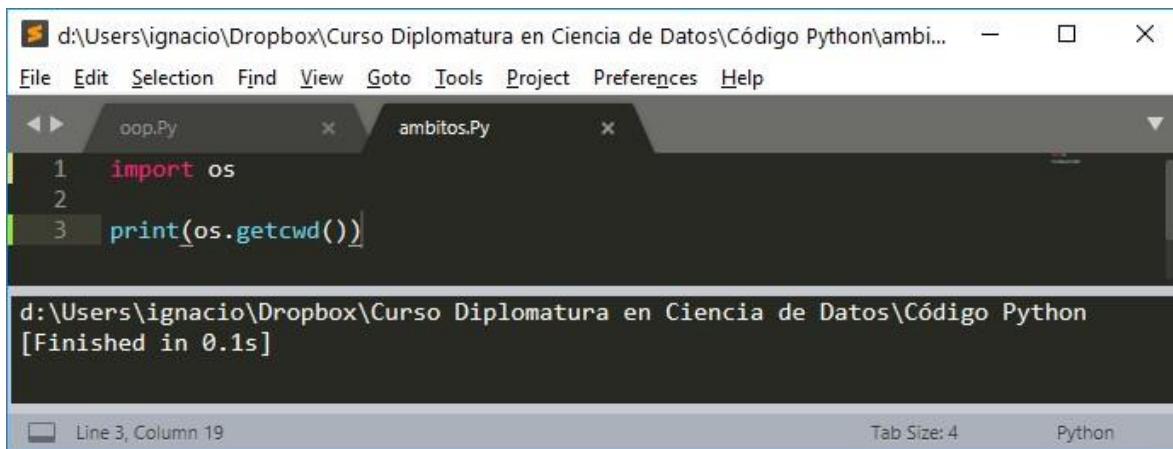
Vamos a explorar las posibilidades que nos brinda el paquete os
Para tenerlo disponible nos alcanza con poner al principio de nuestro programa:

```
import os
```

Los métodos con los cuales podemos obtener información son:

```
os.getcwd()
```

Devuelve dentro de un string el directorio actual:



A screenshot of a Python code editor window. The title bar says "d:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python\ambitos.py". The menu bar includes File, Edit, Selection, Find, View, Goto, Tools, Project, Preferences, and Help. There are two tabs open: "oop.py" and "ambitos.py". The code in "ambitos.py" is:

```
1 import os
2
3 print(os.getcwd())
```

The output window below shows the result of running the script:

```
d:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python
[Finished in 0.1s]
```

At the bottom, status bars indicate "Line 3, Column 19", "Tab Size: 4", and "Python".

```
os.listdir()
```

Obtiene el contenido del directorio actual como una lista de strings.

Vamos a imprimirlo como lista y luego a recorrerlo como lista e imprimir cada elemento:

```

d:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python\ambitos.py - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
oop.py ambitos.py
1 import os
2
3 print(os.listdir())
4
5 for l in os.listdir():
6     print(l)
7
['ambitos.py', 'Archivos.py', 'Datos.txt', 'devuelve_ciudades.py', 'Ejemplo_str.py', 'Fibonaci.py', 'oop.py', 'Ordenar.py', 'ParaOrdenar.csv',
'PrimerPrograma.py', 'PruebaCicloDefinido.py', 'PruebaCicloIndefinido.py', 'PruebaCondicional.py', 'PruebaCSV.py', 'PruebaCSV2.py',
'PruebaDiccionarios.py', 'PruebaElseif.py', 'PruebaExcepciones.py', 'PruebaExcepciones2.py', 'PruebaIn.py', 'PruebaInput.py', 'PruebaLista.py',
'PruebaLista2.py', 'PruebaRecorreCadena.py', 'PruebaTupla.py', 'PruebaWhileBreak.py', 'SegundoPrograma.py', 'ValidadorEMail.py', '__pycache__']
ambitos.py
Archivos.py
Datos.txt.txt
devuelve_ciudades.py
Ejemplo_str.py
Fibonaci.py
oop.py
Ordenar.py
ParaOrdenar.csv
PrimerPrograma.py
PruebaCicloDefinido.py
PruebaCicloIndefinido.py
PruebaCondicional.py
PruebaCSV.py
PruebaCSV2.py
PruebaDiccionarios.py
PruebaElseif.py
PruebaExcepciones.py
PruebaExcepciones2.py
PruebaIn.py
PruebaInput.py
PruebaLista.py
PruebaLista2.py
PruebaRecorreCadena.py
PruebaTupla.py
PruebaWhileBreak.py
SegundoPrograma.py
ValidadorEMail.py
__pycache__
[Finished in 0.1s]

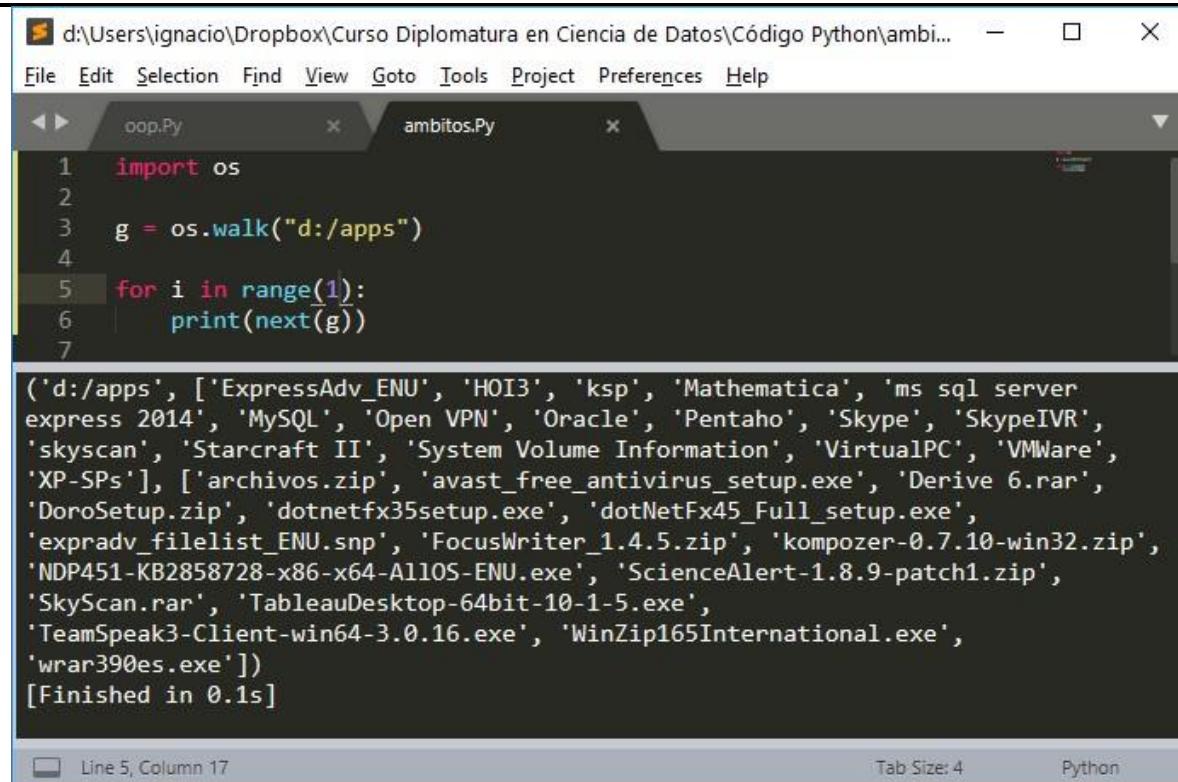
```

2 lines, 2 characters selected Tab Size: 4 Python

Vemos que, en primer lugar y entre [] nos pasa la lista de los archivos del directorio actual y luego, a continuación, recorre la lista de los nombres de archivo a uno por renglón tal como nos proponíamos.

`os.walk("directorio de inicio")`

Devuelve un generador que recorre uno a uno los archivos y directorios que se encuentran a partir de la carpeta anunciada.



```

d:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python\ambitos.py

File Edit Selection Find View Goto Tools Project Preferences Help

ambitos.py * oop.py *

1 import os
2
3 g = os.walk("d:/apps")
4
5 for i in range(1):
6     print(next(g))
7

('d:/apps', ['ExpressAdv_ENU', 'HOI3', 'ksp', 'Mathematica', 'ms sql server express 2014', 'MySQL', 'Open VPN', 'Oracle', 'Pentaho', 'Skype', 'SkypeIVR', 'skyscan', 'Starcraft II', 'System Volume Information', 'VirtualPC', 'VMWare', 'XP-SPs'], ['archivos.zip', 'avast_free_antivirus_setup.exe', 'Derive 6.rar', 'DoroSetup.zip', 'dotnetfx35setup.exe', 'dotNetFx45_Full_setup.exe', 'expradv_filelist_ENU.snp', 'FocusWriter_1.4.5.zip', 'kompozer-0.7.10-win32.zip', 'NDP451-KB2858728-x86-x64-A110S-ENU.exe', 'ScienceAlert-1.8.9-patch1.zip', 'SkyScan.rar', 'TableauDesktop-64bit-10-1-5.exe', 'TeamSpeak3-Client-win64-3.0.16.exe', 'WinZip165International.exe', 'wRAR390es.exe'])
[Finished in 0.1s]

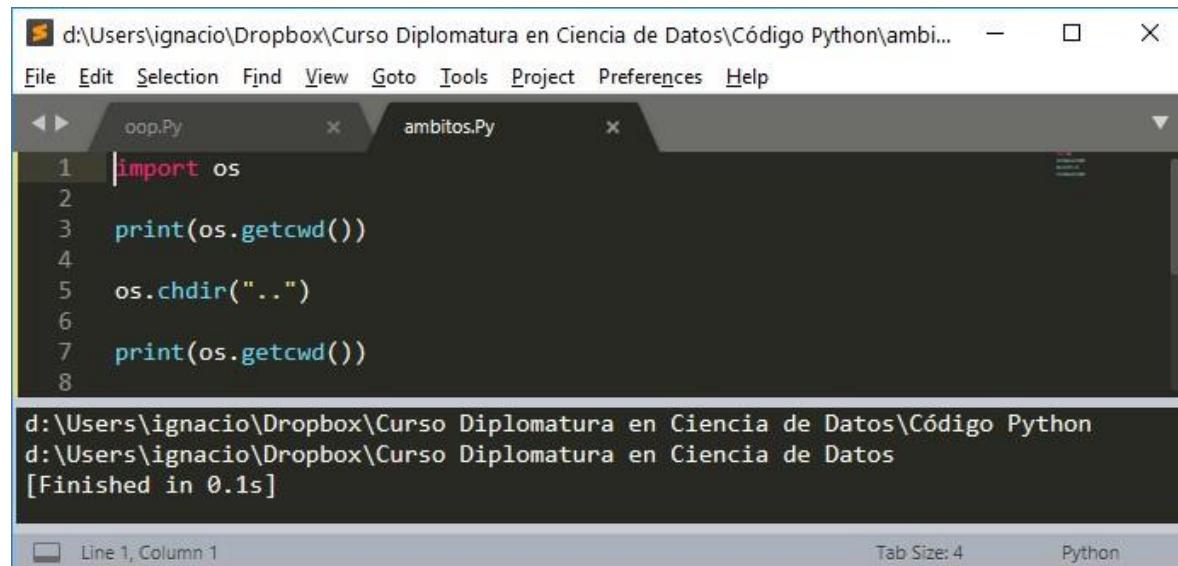
Line 5, Column 17 Tab Size: 4 Python

```

Los métodos con los cuales podemos realizar cambios son:

`os.chdir("path absoluto o relativo")`

Mueve el directorio actual



```

d:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python\ambitos.py

File Edit Selection Find View Goto Tools Project Preferences Help

ambitos.py * oop.py *

1 import os
2
3 print(os.getcwd())
4
5 os.chdir("..")
6
7 print(os.getcwd())
8

d:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python
d:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos
[Finished in 0.1s]

Line 1, Column 1 Tab Size: 4 Python

```

`os.makedirs("directorio a crear")` Crea un directorio.

```

d:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python\ambitos.py
File Edit Selection Find View Goto Tools Project Preferences Help
< > oop.py x ambitos.py x
1 import os
2
3 print(os.getcwd())
4
5 os.makedirs("abc")
6
7 os.chdir("abc")
8
9 print(os.getcwd())
10
11
12
d:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python
d:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python\abc
[Finished in 0.1s]

```

Line 9, Column 19 Tab Size: 4 Python

`shutil.copy2("path de origen","path de destino")`

Copia un conjunto de archivos o directorios

```

d:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python\ambitos.py
File Edit Selection Find View Goto Tools Project Preferences Help
< > oop.py x ambitos.py x
1 import os
2 import shutil
3
4 shutil.copy2("ambitos.py","abc")
5
6 os.chdir("abc")
7
8 print(os.listdir())
9

```

```

['ambitos.py']
[Finished in 0.1s]

```

Line 4, Column 28 Tab Size: 4 Python

La diferencia con `shutil.copy` es que el 2 copia los metadatos de los archivos mientras que el `shutil.copy` no los incluye.

`os.remove("nombre de archivo")`

Borra un archivo.

The screenshot shows a Python code editor window with a dark theme. The title bar says "d:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python\ambitos.py". The menu bar includes File, Edit, Selection, Find, View, Goto, Tools, Project, Preferences, and Help. There are two tabs open: "oop.py" and "ambitos.py". The "ambitos.py" tab contains the following code:

```
1 import os
2
3 os.remove("ambitos.py")
4
5 print(os.listdir())
['abc', 'Archivos.py', 'Datos.txt.txt', 'devuelve_ciudades.py', 'Ejemplo
str.py', 'Fibonaci.py', 'oop.py', 'Ordenar.py', 'ParaOrdenar.csv',
'PrimerPrograma.py', 'PruebaCicloDefinido.py', 'PruebaCicloInDefinido.py',
'PruebaCondicional.py', 'PruebaCSV.py', 'PruebaCSV2.py',
'PruebaDicionarios.py', 'Prueba elif.py', 'PruebaExcepciones.py',
'PruebaExcepciones2.py', 'PruebaIn.py', 'PruebaInput.py', 'PruebaLista.py',
'PruebaLista2.py', 'PruebaRecorreCadena.py', 'PruebaTupla.py',
'PruebaWhileBreak.py', 'SegundoPrograma.py', 'ValidadorEMail.py', '__pycache__']
[Finished in 0.1s]
```

At the bottom, it says "Line 5, Column 1" and "Tab Size: 4" and "Python".

```
shutil.rmtree("directorio")
```

Borra un directorio y todo su contenido

The screenshot shows a Python code editor window with a dark theme. The title bar says "d:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python\ambitos.py". The menu bar includes File, Edit, Selection, Find, View, Goto, Tools, Project, Preferences, and Help. There are two tabs open: "oop.py" and "ambitos.py". The "ambitos.py" tab contains the following code:

```
1 import os
2 import shutil
3
4 shutil.rmtree("abc")
5
6 print(os.listdir())
['ambitos.py', 'Archivos.py', 'Datos.txt.txt', 'devuelve_ciudades.py', 'Ejemplo
str.py', 'Fibonaci.py', 'oop.py', 'Ordenar.py', 'ParaOrdenar.csv',
'PrimerPrograma.py', 'PruebaCicloDefinido.py', 'PruebaCicloInDefinido.py',
'PruebaCondicional.py', 'PruebaCSV.py', 'PruebaCSV2.py', 'PruebaDicionarios.py',
'Prueba elif.py', 'PruebaExcepciones.py', 'PruebaExcepciones2.py', 'PruebaIn.py',
'PruebaInput.py', 'PruebaLista.py', 'PruebaLista2.py', 'PruebaRecorreCadena.py',
'PruebaTupla.py', 'PruebaWhileBreak.py', 'SegundoPrograma.py', 'ValidadorEMail.py', '__pycache__']
[Finished in 0.1s]
```

At the bottom, it says "Line 3, Column 1" and "Tab Size: 4" and "Python".

Módulo fs

Se puede encontrar toda la información relacionada con el módulo fs en:

<https://docs.pyfilesystem.org/en/latest/>

La API PyFilesystem suele ser más simple de utilizar que los módulos os e io.

La principal ventaja que ofrece PyFilesystem es que permite ejecutar el mismo código sobre distintos sistemas operativos. Por ejemplo permite escribir una función para identificar archivos duplicados y no importará si luego la queremos ejecutar sobre un directorio en un disco rígido, o en un archivo zip o en un servidor FTP o en un servicio S3 de Amazon.

Para incorporar fs a nuestro Python nos alcanza con hacer desde una ventana de comando de Windows:

```
pip install fs
```

Y, si tenemos conexión a internet y los privilegios de administración correctos todo debiera fluir sin problemas.

Vamos ahora a crear una instancia de sistema de archivos que apunte a nuestro disco local para empezar a trabajar con ella:

The screenshot shows a code editor with two tabs: 'oop.py' and 'ambitos.py'. The 'ambitos.py' tab is active, displaying the following Python code:

```
from fs.osfs import OSFS
home_fs = OSFS("~/")
print(home_fs.listdir("/"))
```

The output of the script is displayed in the terminal window below:

```
['.idlerc', '.kettle', '.pentaho', '.swt', '.VirtualBox', '3D Objects',
 'AppData', 'Contacts', 'Desktop', 'Documents', 'Downloads', 'Dropbox',
 'Favorites', 'Links', 'MicrosoftEdgeBackups', 'Music', 'NTUSER.DAT',
 'ntuser.dat.LOG1', 'ntuser.dat.LOG2',
 'NTUSER.DAT{47a6a179-a514-11e7-a94e-ec0d9a05c860}.TxR.0.regtrans-ms',
 'NTUSER.DAT{47a6a179-a514-11e7-a94e-ec0d9a05c860}.TxR.1.regtrans-ms',
 'NTUSER.DAT{47a6a179-a514-11e7-a94e-ec0d9a05c860}.TxR.2.regtrans-ms',
 'NTUSER.DAT{47a6a179-a514-11e7-a94e-ec0d9a05c860}.TxR.blf',
 'NTUSER.DAT{47a6a17a-a514-11e7-a94e-ec0d9a05c860}.TM.blf', 'NTUSER.DAT{47a6a17a-a514-11e7-a94e-ec0d9a05c860}.TMContainer000000000000000000000001.regtrans-ms',
 'NTUSER.DAT{47a6a17a-a514-11e7-a94e-ec0d9a05c860}.TMContainer000000000000000000000002.regtrans-ms',
 'ntuser.ini', 'OneDrive', 'Pictures', 'Saved Games', 'Searches', 'Videos',
 'VirtualBox VMs']
[Finished in 0.5s]
```

The status bar at the bottom of the terminal window indicates 'Line 3, Column 28', 'Tab Size: 4', and 'Python'.

Una utilidad muy destacada de fs es permitir la visualización de un árbol de archivos y directorios en forma gráfica:

The screenshot shows a Python code editor interface. At the top, there's a menu bar with File, Edit, Selection, Find, View, Goto, Tools, Project, Preferences, and Help. Below the menu, there are two tabs: 'oop.py' and 'ambitos.py'. The 'ambitos.py' tab is active, displaying the following code:

```
1 from fs.osfs import OSFS
2 home_fs = OSFS(".")
3 home_fs.tree()
```

Below the code, the output of the execution is shown in a large text area:

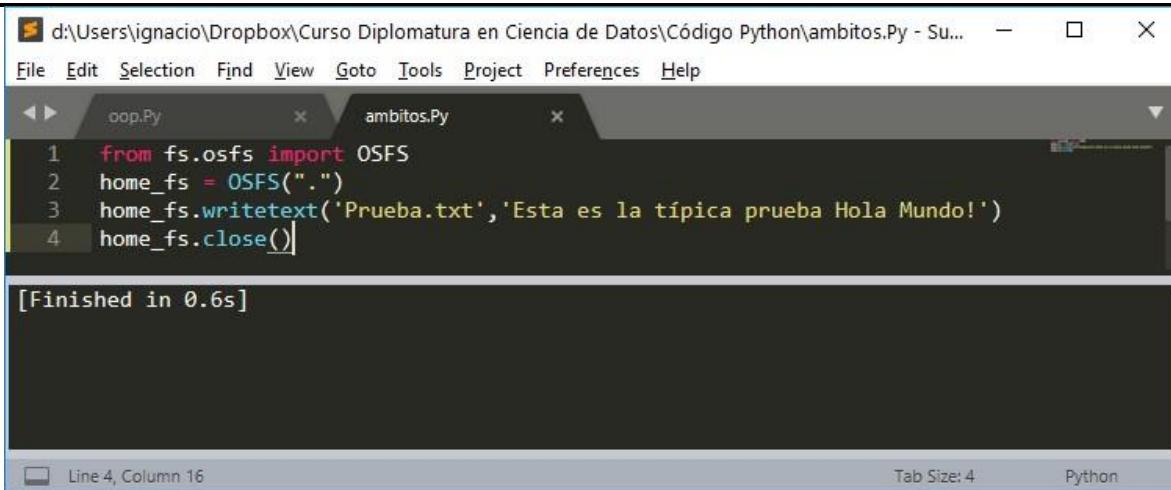
```
-- __pycache__
`-- csv.cpython-37.pyc
-- ambitos.py
-- Archivos.py
-- Datos.txt.txt
-- devuelve_ciudades.py
-- Ejemplo_str.py
-- Fibonaci.py
-- oop.py
-- Ordenar.py
-- ParaOrdenar.csv
-- PrimerPrograma.py
-- PruebaCicloDefinido.py
-- PruebaCicloIndefinido.py
-- PruebaCondicional.py
-- PruebaCSV.py
-- PruebaCSV2.py
-- PruebaDiccionarios.py
-- PruebaElif.py
-- PruebaExcepciones.py
-- PruebaExcepciones2.py
-- PruebaIn.py
-- PruebaInput.py
-- PruebaLista.py
-- PruebaLista2.py
-- PruebaRecorreCadena.py
-- PruebaTupla.py
-- PruebaWhileBreak.py
`-- SegundoPrograma.py
`-- ValidadorEMail.py
[Finished in 0.5s]
```

At the bottom of the editor window, there are status indicators: 'Line 3, Column 15', 'Tab Size: 4', and 'Python'.

close()

Para algunos sistemas de archivos es necesario cerrar la conexión para liberar recursos. Este NO es el caso de los sistemas de archivos que utilizan los discos locales. Se vuelve más relevante, por ejemplo en FTP.

Por ejemplo:

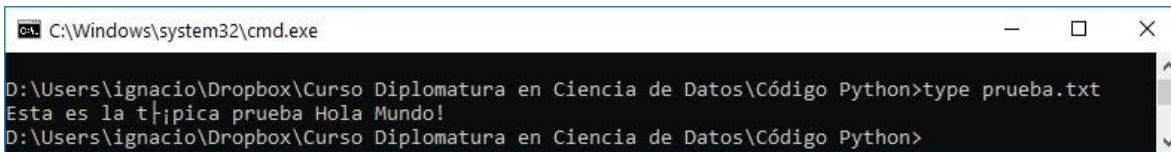


```
d:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python\ambitos.Py - Su... — X
File Edit Selection Find View Goto Tools Project Preferences Help
oop.py x ambitos.py x
1 from fs.osfs import OSFS
2 home_fs = OSFS(".")
3 home_fs.writetext('Prueba.txt', 'Esta es la típica prueba Hola Mundo!')
4 home_fs.close()

[Finished in 0.6s]

Line 4, Column 16 Tab Size: 4 Python
```

¿Y, que pasó?



```
C:\Windows\system32\cmd.exe — X
D:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python>type prueba.txt
Esta es la típica prueba Hola Mundo!
D:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python>
```

Conviene notar el desafío que implican los caracteres acentuados.

scandir()

Sirve para obtener la información del directorio local. Tiene ventajas de performance y brinda más información que listdir()

```
from fs.osfs import OSFS
home_fs = OSFS(".")
for entrada in home_fs.scandir(path = '.'):
    print(entrada.name)
```

ambitos.py
Archivos.py
Datos.txt.txt
devuelve_ciudades.py
Ejemplo_str.py
Fibonaci.py
oop.py
Ordenar.py
ParaOrdenar.csv
PrimerPrograma.py
Prueba.txt
PruebaCicloDefinido.py
PruebaCicloIndefinido.py
PruebaCondicional.py
PruebaCSV.py
PruebaCSV2.py
PruebaDicionarios.py

is_dir()

Es una propiedad de las entradas que produce scandir() Es verdadera si la entrada corresponde a un directorio y falsa en otro caso.

is_file

Es una propiedad de las entradas que produce scandir() Es verdadera si la entrada corresponde a un archivo y falsa en otro caso.

A screenshot of a terminal window titled "ambitos.Py - Su..." showing the output of a Python script. The script uses the `os.scandir` function to iterate through files in the current directory and print their names based on whether they are directories or files.

```
2
3 for entrada in os.scandir(path = '.'):
4     if entrada.is_dir():
5         print("Directorio: ",entrada.name)
6     elif entrada.is_file():
7         print("Archivo: ",entrada.name)
8     else:
9         print("Ni archivo ni directorio: ",entrada.name)
10

Archivo: ambitos.Py
Archivo: Archivos.Py
Archivo: Datos.txt.txt
Archivo: devuelve_ciudades.Py
Archivo: Ejemplo_str.py
Archivo: Fibonaci.Py
Archivo: oop.Py
Archivo: Ordenar.Py
Archivo: ParaOrdenar.csv
Archivo: PrimerPrograma.Py
Archivo: Prueba.txt
Archivo: PruebaCicloDefinido.Py
Archivo: PruebaCicloIndefinido.Py
Archivo: PruebaCondicional.Py
Archivo: PruebaCSV.Py
Archivo: PruebaCSV2.Py
Archivo: PruebaDicionarios.Py
Archivo: PruebaElif.Py
Archivo: PruebaExcepciones.Py
Archivo: PruebaExcepciones2.Py
Archivo: PruebaIn.Py
Archivo: PruebaInput.Py
Archivo: PruebaLista.Py
Archivo: PruebaLista2.Py
Archivo: PruebaRecorreCadena.Py
Archivo: PruebaTupla.Py
Archivo: PruebaWhileBreak.Py
Archivo: SegundoPrograma.Py
Archivo: ValidadorEMail.Py
Directorio: __pycache__
[Finished in 0.2s]
```

Para obtener el tamaño de un archivo podemos usar:

The screenshot shows a terminal window with two tabs: 'oop.py' and 'ambitos.py'. The 'ambitos.py' tab is active, displaying the following Python script:

```
2
3     for entrada in os.scandir(path = '.'):
4         if entrada.is_dir():
5             print("Directorio: ",entrada.name)
6         elif entrada.is_file():
7             print("Archivo: ",entrada.name,os.path.getsize(entrada.name))
8         else:
9             print("Ni archivo ni directorio: ",entrada.name)
10
```

Below the script, the terminal displays the output of running the script:

```
Archivo: ambitos.py 265
Archivo: Archivos.py 139
Archivo: Datos.txt.txt 19
Archivo: devuelve_ciudades.py 302
Archivo: Ejemplo str.py 40
Archivo: Fibonaci.py 251
Archivo: oop.py 750
Archivo: Ordenar.py 544
Archivo: ParaOrdenar.csv 156
Archivo: PrimerPrograma.py 106
Archivo: Prueba.txt 37
Archivo: PruebaCicloDefinido.py 76
Archivo: PruebaCicloIndefinido.py 76
Archivo: PruebaCondicional.py 126
Archivo: PruebaCSV.py 169
Archivo: PruebaCSV2.py 266
Archivo: PruebaDiccionarios.py 297
Archivo: PruebaElif.py 496
Archivo: PruebaExcepciones.py 189
Archivo: PruebaExcepciones2.py 364
Archivo: PruebaIn.py 94
Archivo: PruebaInput.py 88
Archivo: PruebaLista.py 54
Archivo: PruebaLista2.py 152
Archivo: PruebaRecorreCadena.py 77
Archivo: PruebaTupla.py 150
Archivo: PruebaWhileBreak.py 124
Archivo: SegundoPrograma.py 395
Archivo: ValidadorEMail.py 337
Directorio: __pycache__
[Finished in 0.1s]
```

Walking

Devuelve una lista que permite recorrer los archivos de un directorio con una condición de filtro.

The screenshot shows a Windows desktop environment. In the foreground, there is a terminal window titled 'ambitosPy...' displaying the output of a Python script. The script uses the `os.walk` function to traverse the 'c:/' directory and prints the current path and a counter 'k'. Below this, the terminal continues to show the results of the walk operation, listing numerous files and directories found in the Windows 10 upgrade resources. In the background, a code editor window titled 'oop.py' is visible, containing the Python code for the script.

```

import os
k=0
for l in os.walk('c:/'):
    print(k,l)
    k=k+1

```

```

'webservices.dll'])
46584 ('c:/Windows10Upgrade\\dll2', [], ['webservices.dll'])
46585 ('c:/Windows10Upgrade\\resources', ['amd64', 'i386', 'ux'],
['hwcompatShared.txt'])
46586 ('c:/Windows10Upgrade\\resources\\amd64', [], ['BiosBlocks.xml',
'hwcompat.txt', 'hwexclude.txt', 'nxquery.cat', 'nxquery.inf', 'NXQuery.sys'])
46587 ('c:/Windows10Upgrade\\resources\\i386', [], ['BiosBlocks.xml',
'hwcompat.txt', 'hwexclude.txt', 'nxquery.cat', 'nxquery.inf', 'NXQuery.sys'])
46588 ('c:/Windows10Upgrade\\resources\\ux', ['EULA', 'Microsoft.WinJS'],
['block.png', 'bluelogo.png', 'bullet.png', 'default.css', 'default.htm',
'default_eos.css', 'default_eos.htm', 'default_oobe.css', 'default_oobe.htm',
'default_posteos.css', 'default_posteos.htm', 'default_tens.htm', 'eula.css',
'loading.gif', 'lock.png', 'logo.png', 'marketing.png',
'MicrosoftLogo.scale-100.png', 'Network.scale-100.png', 'pass.png',
'Plug.scale-100.png', 'Power.scale-100.png', 'Save.scale-100.png',
'ShieldError.scale-100.png'])
46589 ('c:/Windows10Upgrade\\resources\\ux\\EULA', [], ['EULA_ar-sa.htm',
'EULA_bg-bg.htm', 'EULA_cs-cz.htm', 'EULA_da-dk.htm', 'EULA_de-de.htm',
'EULA_el-gr.htm', 'EULA_en-gb.htm', 'EULA_en-us.htm', 'EULA_es-es.htm',
'EULA_es-mx.htm', 'EULA_et-ee.htm', 'EULA_fi-fi.htm', 'EULA_fr-ca.htm',
'EULA_fr-fr.htm', 'EULA_he-il.htm', 'EULA_hr-hr.htm', 'EULA_hu-hu.htm',
'EULA_it-it.htm', 'EULA_ja-jp.htm', 'EULA_ko-kr.htm', 'EULA_lt-lt.htm',
'EULA_lv-lv.htm', 'EULA_nb-no.htm', 'EULA_nl-nl.htm', 'EULA_pl-pl.htm',
'EULA_pt-br.htm', 'EULA_pt-pt.htm', 'EULA_ro-ro.htm', 'EULA_ru-ru.htm',
'EULA_sk-sk.htm', 'EULA_sl-si.htm', 'EULA_sr-latn-cs.htm', 'EULA_sv-se.htm',
'EULA_th-th.htm', 'EULA_tr-tr.htm', 'EULA_uk-ua.htm', 'EULA_zh-cn.htm',
'EULA_zh-hk.htm', 'EULA_zh-tw.htm'])
46590 ('c:/Windows10Upgrade\\resources\\ux\\Microsoft.WinJS', ['css', 'js'], [])
46591 ('c:/Windows10Upgrade\\resources\\ux\\Microsoft.WinJS\\css', [],
['oobe-desktop.css', 'oobe-desktopRS2.css', 'ui-dark.css'])
46592 ('c:/Windows10Upgrade\\resources\\ux\\Microsoft.WinJS\\js', [], ['base.js',
'ui.js'])
[Finished in 31.9s]

```

Cada uno de los elementos que devuelve `walk` es un tupla con tres elementos:

La raíz de la entrada que estoy recorriendo

Una tupla con los nombres de los archivos que cuelgan de esa entrada

Una tupla con los nombres de los directorios que cuelgan de esa entrada

Cuando agota la recorrida de la entrada salta recursivamente por sus directorios.

The screenshot shows a Windows desktop environment. In the foreground, there is a terminal window titled 'ambitos.Py' which displays the output of a Python script. The script uses the 'os.walk' function to traverse a directory tree and print file and directory paths. Below the terminal window, the taskbar is visible with several icons for common applications like File Explorer, Google Chrome, and Microsoft Word.

```
1 import os
2 k=0
3
4 for root, dirs, files in os.walk(".", topdown=True):
5     k=k+1
6     print("Vamos x ",k," root ",root)
7     for name in files:
8         print(os.path.join(root, name))
9     for name in dirs:
10        print(os.path.join(root, name))

Vamos x 1 root .
.\ambitos.Py
.\Archivos.Py
.\Datos.txt.txt
.\devuelve_ciudades.Py
.\Ejemplo str.py
.\Fibonaci.Py
.\oop.Py
.\Ordenar.Py
.\ParaOrdenar.csv
.\PrimerPrograma.Py
.\Prueba.txt
.\PruebaCicloDefinido.Py
.\PruebaCicloIndefinido.Py
.\PruebaCondicional.Py
.\PruebaCSV.Py
.\PruebaCSV2.Py
.\PruebaDiccionarios.Py
.\Prueba elif.Py
.\PruebaExcepciones.Py
.\PruebaExcepciones2.Py
.\PruebaIn.Py
.\PruebaInput.Py
.\PruebaLista.Py
.\PruebaLista2.Py
.\PruebaRecorreCadena.Py
.\PruebaTupla.Py
.\PruebaWhileBreak.Py
.\SegundoPrograma.Py
.\ValidadorEMail.Py
.\__pycache__
Vamos x 2 root .\__pycache__
.\__pycache__\csv.cpython-37.pyc
```

Globing

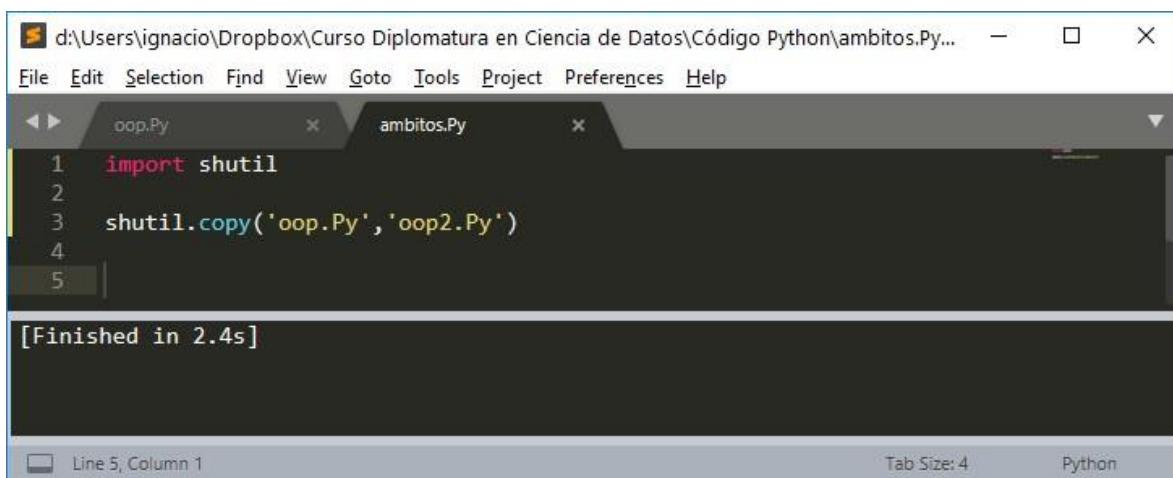
Permite realizar una recorrida de archivos y directorios como walk pero agregando una condición de filtro.

Mover y copiar archivos

Necesitaremos importar shutil:

```
move()  
copy()  
movedir()  
copydir()
```

Por ejemplo:



The screenshot shows a code editor window titled "ambitos.py". The code in the editor is:

```
1 import shutil  
2  
3 shutil.copy('oop.py', 'oop2.py')  
4  
5
```

Below the code editor, a message box displays "[Finished in 2.4s]". At the bottom of the window, it says "Line 5, Column 1" and "Tab Size: 4". The status bar also indicates "Python".

Manejo de archivos planos

open()

Abre una conexión al archivo que nos permitirá manipular el contenido.

Podemos abrirlo para leer, escribir o agregar.

El uso de open() es:

```
open(nombre del archivo, modo de apertura)
```

El modo de apertura es 'r' para lectura, 'w' para escritura y 'a' para agregar.

Cuando terminamos de trabajar con el archivo necesitamos cerrarlo con close:

The screenshot shows a code editor window with two tabs: 'oop.py' and 'ambitos.py'. The 'oop.py' tab is active and displays the following Python code:

```
1 archivo = open('oop2.Py')
2 texto = archivo.read()
3 print(texto)
4 archivo.close()

def arrancar(self):
    self.enMarcha = True

def parar(self):
    self.enMarcha = False
    self.apagarLuces()

def encenderLuces(self):
    if self.enMarcha:
        self.__luces = True

def apagarLuces(self):
    self.__luces = False

def estadoLuces(self):
    if self.__luces:
        return "Encendidas"
    else:
        return "Apagadas"

miCoche = Coche()
print("Luces: ", miCoche.estadoLuces())
miCoche.arrancar()
print("Luces: ", miCoche.estadoLuces())
miCoche.__luces = True
print("Luces: ", miCoche.estadoLuces())
miCoche.encenderLuces()
print("Luces: ", miCoche.estadoLuces())
miCoche.parar()
print("Luces: ", miCoche.estadoLuces())

[Finished in 0.2s]
```

The status bar at the bottom of the editor shows 'Line 1, Column 1' and 'Tab Size: 4'. The language mode is set to 'Python'.

Vemos que con `read()` leemos el contenido del archivo completo y lo guardamos en una variable que se llama `texto`.

Hagamos un ejemplo de escritura:

```
d:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código... - □ X
File Edit Selection Find View Goto Tools Project Preferences Help
◀ ▶ oop.Py × ambitos.Py ×
1 archivo = open('prueba.txt','w')
2
3 for i in range(10):
4     archivo.write('Paso '+str(i)+"\n")
5
6 archivo.close
7
8 archivo = open('prueba.txt','r')
9 texto = archivo.read()
10 print(texto)
11 archivo.close
12
Paso 0
Paso 1
Paso 2
Paso 3
Paso 4
Paso 5
Paso 6
Paso 7
Paso 8
Paso 9
[Finished in 0.1s]

Line 4, Column 37 Tab Size: 4 Python
```

Búsqueda de patrones

Uno de los problemas más comunes con que nos solemos encontrar al desarrollar cualquier programa informático, es el de procesamiento de texto. Esta tarea puede resultar bastante trivial para el cerebro humano, ya que nosotros podemos detectar con facilidad que es un número y que una letra, o cuales son palabras que cumplen con un determinado patrón y cuáles no; pero estas mismas tareas no son tan fáciles para una computadora.

Es por esto, que el procesamiento de texto siempre ha sido uno de los temas más relevantes en las ciencias de la computación. Luego de varias décadas de investigación se logró desarrollar un poderoso y versátil lenguaje que cualquier computadora puede utilizar para reconocer patrones de texto; este lenguaje es lo que hoy en día se conoce con el nombre

de expresiones regulares; las operaciones de validación, búsqueda, extracción y sustitución de texto ahora son tareas mucho más sencillas para las computadoras gracias a las expresiones regulares.

¿Qué son las Expresiones Regulares?

Las expresiones regulares, a menudo llamadas también regex, son secuencias de caracteres que forman un patrón de búsqueda, las cuales son formalizadas por medio de una sintaxis específica. Los patrones se interpretan como un conjunto de instrucciones, que luego se ejecutan sobre un texto de entrada para producir un subconjunto o una versión modificada del texto original.

Las expresiones regulares pueden incluir patrones de coincidencia literal, de repetición, de composición, de ramificación, y otras sofisticadas reglas de reconocimiento de texto.

Las expresiones regulares deberían formar parte del arsenal de cualquier buen programador ya que un gran número de problemas de procesamiento de texto pueden ser fácilmente resueltos con ellas.

Para realizar búsquedas con expresiones regulares debemos primero generar un patrón y luego utilizar el método `match` del patrón.

Las operaciones que podemos hacer sobre un patrón son

`match()`:

El cual determina si la regex tiene coincidencias en el comienzo del texto.

`search()`:

El cual escanea todo el texto buscando cualquier ubicación donde haya una coincidencia.

`findall()`:

El cual encuentra todos los subtextos donde haya una coincidencia y nos devuelve estas coincidencias como una lista.

`finditer()`:

El cual es similar al anterior pero en lugar de devolvernos una lista nos devuelve un generador.

Por ejemplo para `match`:

A screenshot of a Jupyter Notebook interface. The top bar shows the path: d:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código... and the menu: File Edit Selection Find View Goto Tools Project Preferences Help. Below the menu, there are two tabs: oop.py and ambitos.py. The oop.py tab is active, displaying the following Python code:

```
1 import re
2
3 patron = re.compile('a')
4
5 print(patron.match('amigo'))
6 print(patron.match('pepa'))
7 print(patron.match('Pepe'))
8
9
10
```

The output area below the code shows the results of the execution:

```
<re.Match object; span=(0, 1), match='a'>
None
None
[Finished in 0.1s]
```

At the bottom left, it says Line 8, Column 1. At the bottom right, it says Tab Size: 4 and Python.

Por ejemplo para search:

A screenshot of a Jupyter Notebook interface, identical to the one above but with different code. The top bar, tabs, and code are the same. The output area shows the results of the execution:

```
<re.Match object; span=(0, 1), match='a'>
<re.Match object; span=(3, 4), match='a'>
None
[Finished in 0.1s]
```

At the bottom left, it says Line 8, Column 1. At the bottom right, it says Tab Size: 4 and Python.

Por ejemplo para findall

The screenshot shows a terminal window with the following content:

```
d:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código... - □ X
File Edit Selection Find View Goto Tools Project Preferences Help
◀ ▶ oop.py × ambitos.py ×
1 import re
2
3 patron = re.compile('a')
4
5 print(patron.findall('amigo'))
6 print(patron.findall('pepa'))
7 print(patron.findall('Papa'))
8
9
10
['a']
['a']
['a', 'a']
[Finished in 0.1s]
```

Line 8, Column 1 Tab Size: 4 Python

Además de especificar un carácter el patrón puede reconocer una serie de caracteres:

The screenshot shows a code editor window with two tabs: 'oop.py' and 'ambitos.py'. The 'ambitos.py' tab is active, displaying the following Python code:

```
1 import re
2
3 patron = re.compile('[abg]')
4
5 print(patron.findall('amigo'))
6 print(patron.findall('pepa'))
7 print(patron.findall('Papa'))
8
9
10
```

The output window below the code shows the results of the regular expression matches:

```
['a', 'g']
['a']
['a', 'a']
[Finished in 0.3s]
```

At the bottom of the editor window, status bars indicate 'Line 3, Column 26' and 'Tab Size: 4'.

Con un signo ^ delante decimos que caracteres no pueden estar:

The screenshot shows a code editor window with two tabs: 'oop.py' and 'ambitos.py'. The 'ambitos.py' tab is active, displaying the following Python code:

```
1 import re
2
3 patron = re.compile('[^abcdefghijklmnoprstuvwxyz]')
4
5 print(patron.findall('amigo'))
6 print(patron.findall('pepa'))
7 print(patron.findall('Papa'))
8
9
10
```

The output window below the code shows the results of the regular expression matches:

```
[]
[]
['P']
[Finished in 0.3s]
```

At the bottom of the editor window, status bars indicate 'Line 3, Column 50' and 'Tab Size: 4'.

El patrón puede contener:

Literales:

Cualquier carácter que no tiene un significado especial.

Un carácter sin significado especial se encuentra a si mismo cuando se realiza una búsqueda:

Los caracteres que tienen un significado especial son las secuencias de escape y los metacaracteres.

Secuencias de escape:

La sintaxis de las expresiones regulares nos permite utilizar las secuencias de escape que ya conocemos de otros lenguajes de programación para esos casos especiales como ser finales de línea, tabs, barras diagonales, etc. Las principales secuencias de escape que podemos encontrar, son:

Secuencia de escape Significado

\n	Nueva línea (new line). El cursor pasa a la primera posición de la línea siguiente.
\t	Tabulador. El cursor pasa a la siguiente posición de tabulación.
\\\	Barra diagonal inversa
\v	Tabulación vertical.
\ooo	Carácter ASCII en notación octal.
\xhh	Carácter ASCII en notación hexadecimal.
\xhhhh	Carácter Unicode en notación hexadecimal.

Clases de caracteres: Se pueden especificar clases de caracteres encerrando una lista de caracteres entre corchetes [], la que encontrará uno cualquiera de los caracteres de la lista. Si el primer símbolo después del "[" es "^", la clase encuentra cualquier carácter que no está en la lista.

Metacaracteres: Los metacaracteres son caracteres especiales que son la esencia de las expresiones regulares. Como son sumamente importantes para entender la sintaxis de las expresiones regulares y existen diferentes tipos, voy a dedicar una sección a explicarlos un poco más en detalle.

Metacaracteres

Metacaracteres - delimitadores

Esta clase de metacaracteres nos permite delimitar dónde queremos buscar los patrones de búsqueda. Ellos son:

Metacaracter	Descripción
^	inicio de línea. \$ fin de línea.
\A	inicio de texto.

\Z	fin de texto.
.	cualquier caracter en la línea.
\b	encuentra límite de palabra.
\B	encuentra distinto a límite de palabra.

Metacaracteres - clases predefinidas

Estas son clases predefinidas que nos facilitan la utilización de las expresiones regulares.
Ellos son:

Metacaracter	Descripción
\w	un carácter alfanumérico (incluye "_").
\W	un carácter no alfanumérico.
\d	un carácter numérico.
\D	un carácter no numérico.
\s	cualquier espacio (lo mismo que [\t\n\r\f]).
\S	un no espacio.

Metacaracteres - iteradores

Cualquier elemento de una expresión regular puede ser seguido por otro tipo de metacaracteres, los iteradores. Usando estos metacaracteres se puede especificar el número de ocurrencias del carácter previo, de un metacaracter o de una subexpresión. Ellos son:

Metacaracter	Descripción
*	cero o más, similar a {0,}.
+	una o más, similar a {1,}.
?	cero o una, similar a {0,1}.
{n}	exactamente n veces.
{n,}	por lo menos n veces.
{n,m}	por lo menos n pero no más de m veces.
*?	cero o más, similar a {0,}?.
+?	una o más, similar a {1,}?.
??	cero o una, similar a {0,1}?.
{n}?	exactamente n veces.
{n,}?	por lo menos n veces.
{n,m}?	por lo menos n pero no más de m veces.

En estos metacaracteres, los dígitos entre llaves de la forma {n,m}, especifican el mínimo número de ocurrencias en n y el máximo en m.

Metacaracteres - alternativas

Se puede especificar una serie de alternativas para una plantilla usando "|" para separarlas, entonces do|re|mi encontrará cualquier "do", "re", o "mi" en el texto de entrada.

Las alternativas son evaluadas de izquierda a derecha, por lo tanto la primera alternativa que coincide plenamente con la expresión analizada es la que se selecciona. Por ejemplo: si se buscan foo|foot en "barefoot", sólo la parte "foo" da resultado positivo, porque es la primera alternativa probada, y porque tiene éxito en la búsqueda de la cadena analizada.

Ejemplo:

foo(bar|foo) --> encuentra las cadenas 'foobar' o 'foofoo'.

Metacaracteres - subexpresiones

La construcción (...) también puede ser empleada para definir subexpresiones de expresiones regulares.

Ejemplos:

(foobar){10} --> encuentra cadenas que contienen 8, 9 o 10 instancias de 'foobar'

foob([0-9]|a+)r --> encuentra 'foob0r', 'foob1r' , 'foobar', 'foobaar', 'foobaar' etc.

Metacaracteres - memorias (backreferences)

Los metacaracteres \1 a \9 son interpretados como memorias. \ encuentra la subexpresión previamente encontrada #.

Ejemplos:

(.)\1+ --> encuentra 'aaaa' y 'cc'.

(.+)\1+ --> también encuentra 'abab' y '123123'

([""]?) (\d+)\1 --> encuentra ""13" (entre comillas dobles), o '4' (entre comillas simples) o 77 (sin comillas) etc.

Expresiones Regulares con Python

Luego de esta introducción, llegó el tiempo de empezar a jugar con las expresiones regulares y Python.

Como no podría ser de otra forma tratándose de Python y su filosofía de todas las baterías incluídas; en la librería estandar de Python podemos encontrar el módulo re, el cual nos proporciona todas las operaciones necesarias para trabajar con las expresiones regulares.

Por lo tanto, en primer lugar lo que debemos hacer es importar el modulo re.

Clases y objetos

Podemos categorizar los lenguajes de programación existentes en dos grandes bloques:

- Orientados a procedimientos
- Orientados a objetos

Orientados a procedimientos	Orientados a objetos
La longitud del código crece linealmente o peor con la complejidad	La longitud del código crece menos que linealmente con la complejidad
La reutilización del código requiere un esfuerzo consciente del programador	La reutilización del código viene empujada por el paradigma
Difícil de entender y mantener por alguien distinto del autor o por el mismo autor si ya se olvidó lo que hizo	Gracias a que se encapsula la funcionalidad junto a las cosas es más simple de entender y por lo tanto mantener por un programador que no sea el autor original.

Vamos a intentar trasladar la naturaleza de las cosas que nos encontramos en nuestra experiencia cotidiana al desarrollo de software para beneficiarnos de nuestra familiaridad con esa experiencia.

Las cosas con las que nos encontramos todos los días tienen:

- Estados
- Propiedades
- Comportamientos

Por ejemplo un automóvil tiene:

Estados:

- Estacionado en el garaje
- Tanque de nafta al 75%
- Motor apagado
- Batería cargada al 100%
- Km recorridos desde la fábrica 52.000
- ...

Propiedades:

- Toyota
- Corolla
- 2019
- ...

Comportamientos: -

Arrancar

- Frenar
- Estacionar
- ...

Si todo esto lo ponemos dentro de un paquete que podamos utilizar cada vez que nuestro programa use un coche entonces este código existirá en un solo lugar, será compacto y fácil de entender porque estaremos hablando de una sola cosa a la vez...

A este paquete de estados, propiedades y comportamientos lo vamos a llamar objeto. Al molde con el que creamos objetos del mismo tipo lo vamos a llamar clase.

Ya que aclaramos los objetos de objetos y clase vamos a aprovechar y poner en blanco sobre negro algunos conceptos más:

- Ejemplar o instanciar
- Modularización
- Encapsulamiento
- Herencia
- Polimorfismo

Ejemplar o instanciar:

Consiste en crear un objeto de una dada clase.

Luego, cada objeto, podrá tener propiedades y estados que los hagan diferentes.

Modularización:

Lo normal y recomendable es que una aplicación compleja esté integrada por varias clases que se reparten las distintas partes de la funcionalidad que queremos reproducir con nuestro programa.

Gracias a esto podremos hacer que en otros programas podamos usar las clases de este programa en otros nuevos que aún tengamos que escribir.

Encapsulamiento:

El funcionamiento interno de cada clase no es visible desde afuera. Por lo tanto podremos cambiar una clase por otra sin preocuparnos más que de las conexiones explícitas que hagamos entre ellas. (métodos de acceso y herencia)

Herencia:

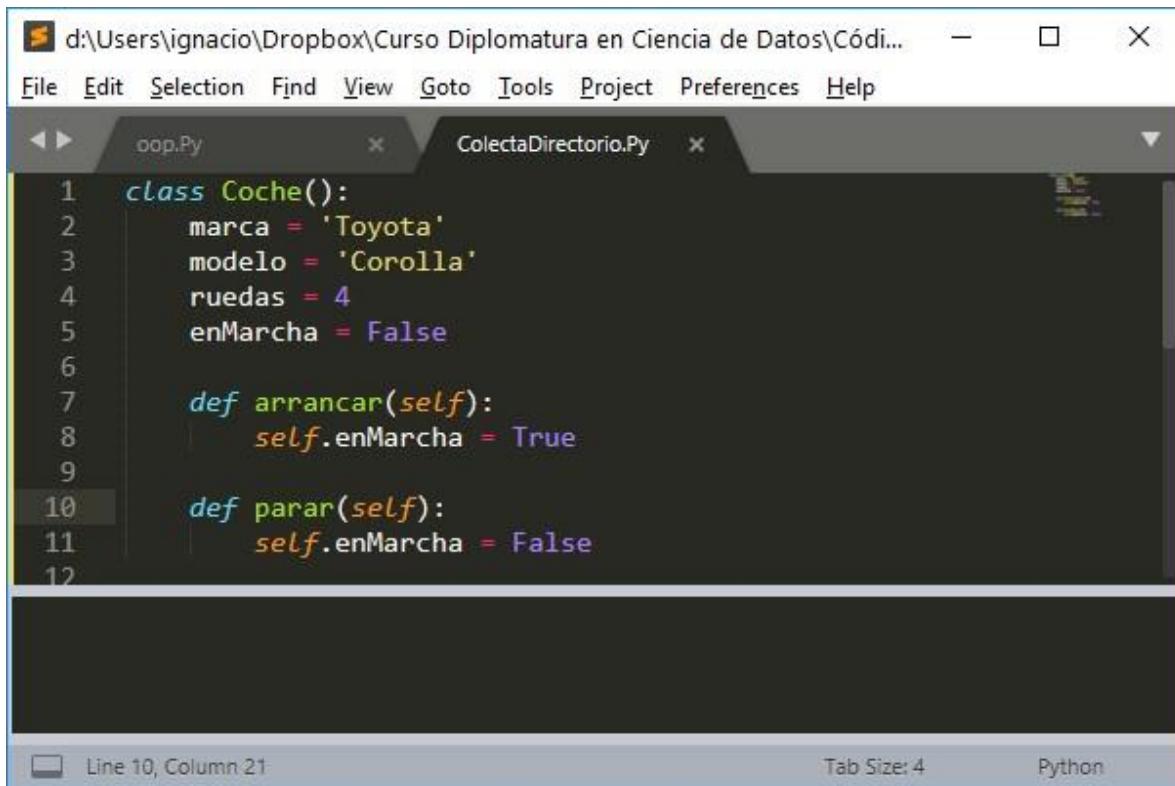
Una clase padre es heredada por una clase hija cuando usamos al padre para definir al hijo. En principio el hijo tendrá los mismos estados, propiedades y comportamientos que el padre. Luego podremos agregarle más estados, propiedades y comportamientos.

Incluso podremos hacer que los comportamientos de un hijo sean internamente diferentes de los comportamientos del padre.

Polimorfismo:

¿Cómo reflejamos todo esto en código Python?

Lo primero será crear una clase:



The screenshot shows a code editor window with two tabs: 'oop.py' and 'ColectaDirectorio.py'. The 'oop.py' tab is active and displays the following Python code:

```
1 class Coche():
2     marca = 'Toyota'
3     modelo = 'Corolla'
4     ruedas = 4
5     enMarcha = False
6
7     def arrancar(self):
8         self.enMarcha = True
9
10    def parar(self):
11        self.enMarcha = False
12
```

The code defines a class 'Coche' with attributes 'marca', 'modelo', 'ruedas', and 'enMarcha', and methods 'arrancar' and 'parar'. The code editor interface includes a status bar at the bottom showing 'Line 10, Column 21', 'Tab Size: 4', and 'Python'.

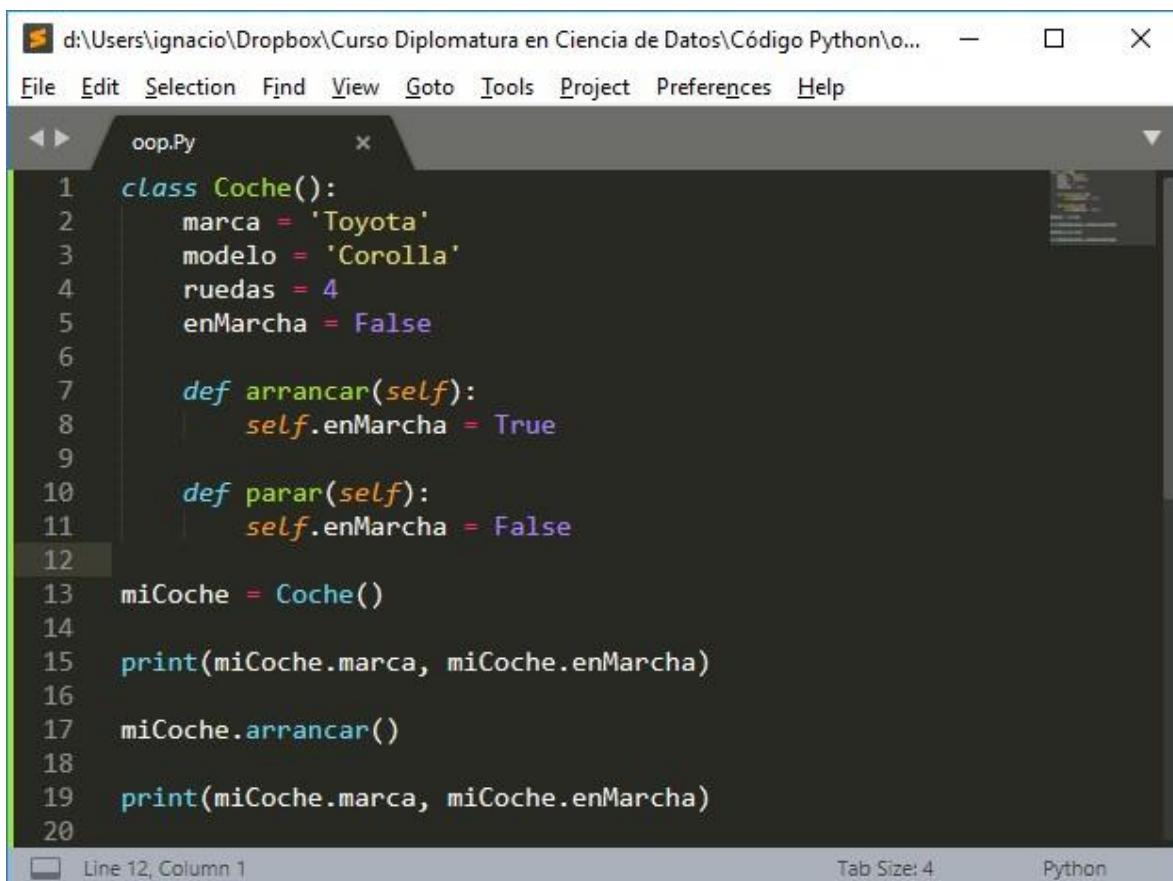
Marca, modelo, ruedas y enMarcha son propiedades de la clase coche. Todos los objetos que creamos siguiendo el molde de la clase coche tendrán estas propiedades y arrancarán en los valores que hemos especificado.

También estamos indicando dos comportamientos que la clase coche va a saber hacer:

- Arrancar
- Parar

A estos comportamientos los llamamos métodos. En este caso arrancar cambia la propiedad enMarcha a True y parar la cambia a False.

Para crear nuestro primer objeto de la clase Coche alcanza con hacer:



```

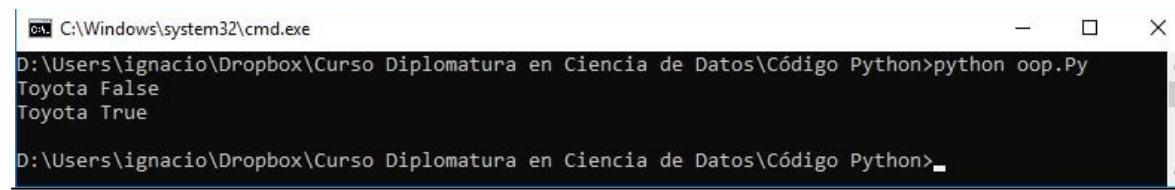
d:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python\oop.py
File Edit Selection Find View Goto Tools Project Preferences Help
    oop.py
1  class Coche():
2      marca = 'Toyota'
3      modelo = 'Corolla'
4      ruedas = 4
5      enMarcha = False
6
7      def arrancar(self):
8          self.enMarcha = True
9
10     def parar(self):
11         self.enMarcha = False
12
13 miCoche = Coche()
14
15 print(miCoche.marca, miCoche.enMarcha)
16
17 miCoche.arrancar()
18
19 print(miCoche.marca, miCoche.enMarcha)
20

```

Line 12, Column 1 Tab Size: 4 Python

Luego podemos tratar de ver como consultamos las propiedades del coche y como usamos sus comportamientos.

Al ejecutarlo obtendremos:



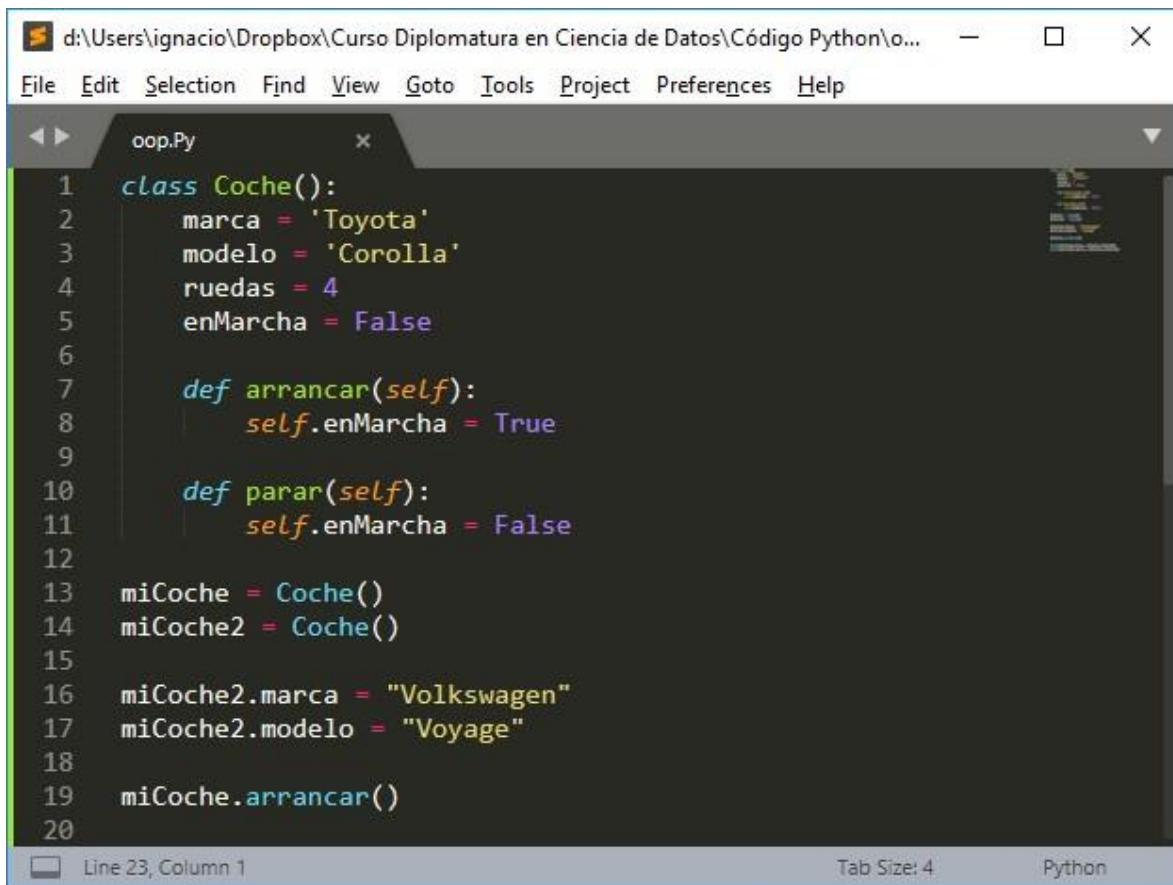
```

C:\Windows\system32\cmd.exe
D:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python>python oop.py
Toyota False
Toyota True
D:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python>

```

Conviene hacer notar que para que los métodos puedan cambiar el valor de la propiedad enMarcha debemos referirnos a ella como self.enMarcha.

Introduzcamos ahora un segundo coche para verificar lo que pasa con sus propiedades y comportamientos:

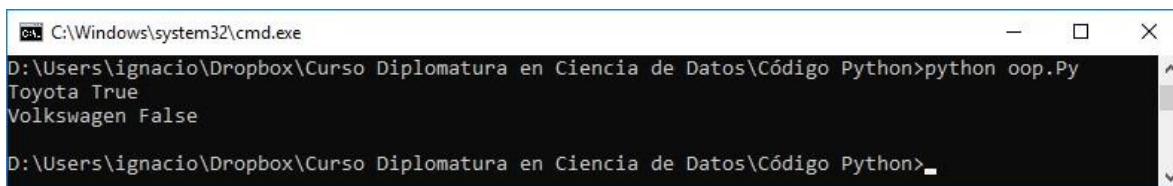


```

d:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python\oop.py
File Edit Selection Find View Goto Tools Project Preferences Help
oop.py
1 class Coche():
2     marca = 'Toyota'
3     modelo = 'Corolla'
4     ruedas = 4
5     enMarcha = False
6
7     def arrancar(self):
8         self.enMarcha = True
9
10    def parar(self):
11        self.enMarcha = False
12
13    miCoche = Coche()
14    miCoche2 = Coche()
15
16    miCoche2.marca = "Volkswagen"
17    miCoche2.modelo = "Voyage"
18
19    miCoche.arrancar()
20
Line 23, Column 1
Tab Size: 4
Python

```

Al ejecutar verificamos que cada coche es independiente:



```

C:\Windows\system32\cmd.exe
D:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python>python oop.py
Toyota True
Volkswagen False
D:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python>

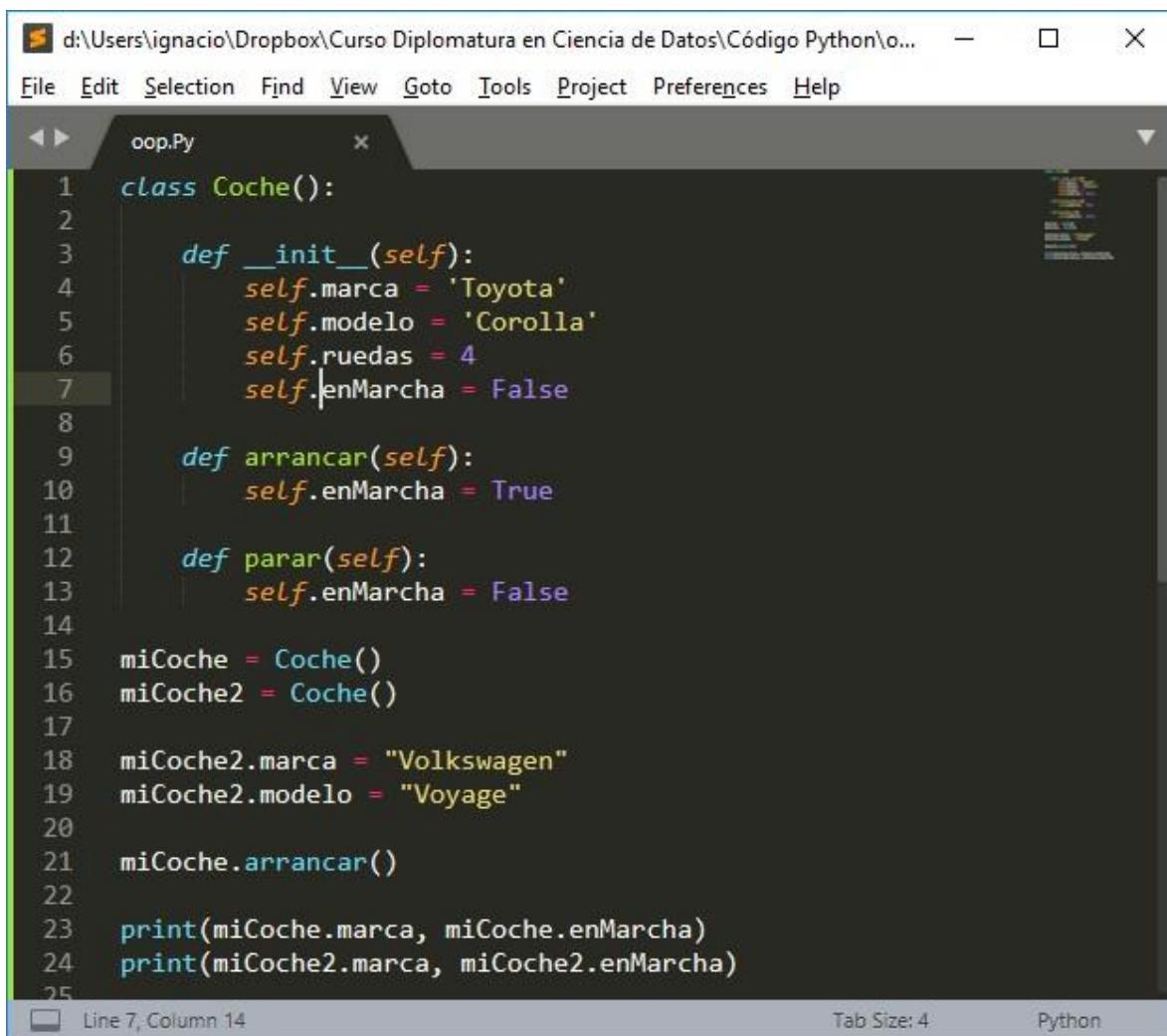
```

Constructor:

El constructor es el método que especificamos para asignarle un estado inicial al objeto. Todos los objetos de esa clase se crearán en ese estado inicial.

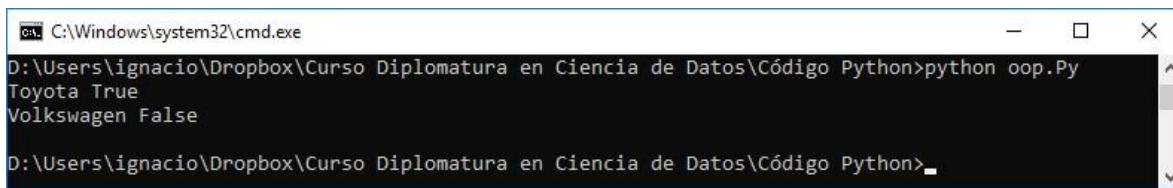
Para definir el constructor vamos a utilizar la instrucción def. El nombre de la función constructora es siempre `__init__`

Por ejemplo, para la clase coche:



```
d:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python\oop.py
File Edit Selection Find View Goto Tools Project Preferences Help
oop.py
1 class Coche():
2
3     def __init__(self):
4         self.marca = 'Toyota'
5         self.modelo = 'Corolla'
6         self.ruedas = 4
7         self.enMarcha = False
8
9     def arrancar(self):
10        self.enMarcha = True
11
12    def parar(self):
13        self.enMarcha = False
14
15 miCoche = Coche()
16 miCoche2 = Coche()
17
18 miCoche2.marca = "Volkswagen"
19 miCoche2.modelo = "Voyage"
20
21 miCoche.arrancar()
22
23 print(miCoche.marca, miCoche.enMarcha)
24 print(miCoche2.marca, miCoche2.enMarcha)
25
Line 7, Column 14
Tab Size: 4
Python
```

Y al ejecutar obtenemos lo mismo que antes:

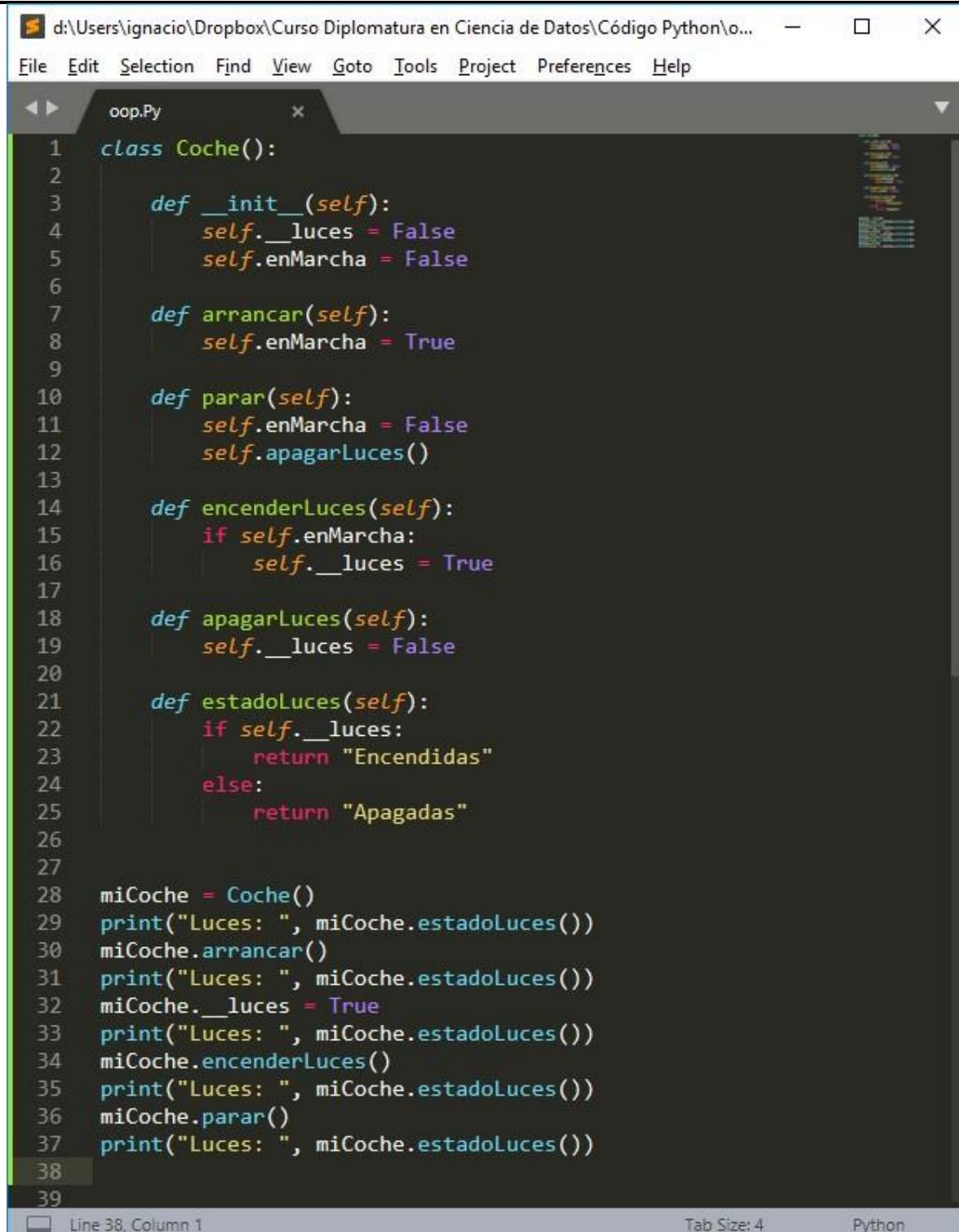


```
C:\Windows\system32\cmd.exe
D:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python>python oop.py
Toyota True
Volkswagen False
D:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python>
```

A veces no queremos que el estado de un objeto pueda ser modificado directamente desde el exterior sino que queremos reservarnos el control forzando a que cualquier cambio se haga por el lado de los métodos.

¿Nunca se quedaron sin baterías por dejar las luces encendidas?

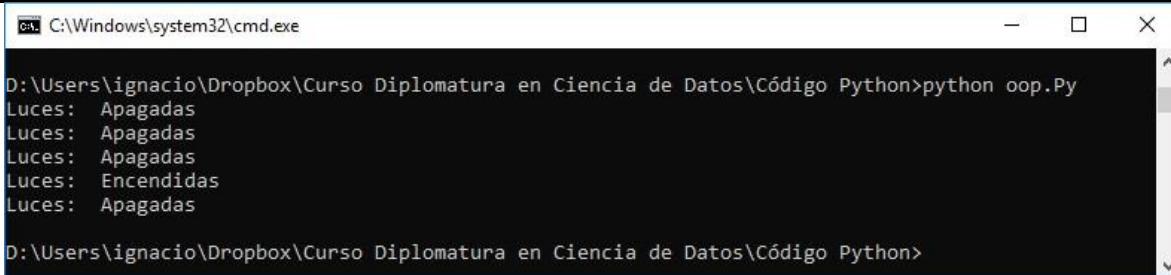
Vamos a querer que esto no pase nunca más. Para lograrlo vamos a asegurarnos que cada vez que paramos el motor se apagan las luces y que no puedan jamás encenderse si el motor no está encendido:



```
d:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python\oop.py
File Edit Selection Find View Goto Tools Project Preferences Help
oop.py
1 class Coche():
2
3     def __init__(self):
4         self.__luces = False
5         self.enMarcha = False
6
7     def arrancar(self):
8         self.enMarcha = True
9
10    def parar(self):
11        self.enMarcha = False
12        self.apagarLuces()
13
14    def encenderLuces(self):
15        if self.enMarcha:
16            self.__luces = True
17
18    def apagarLuces(self):
19        self.__luces = False
20
21    def estadoLuces(self):
22        if self.__luces:
23            return "Encendidas"
24        else:
25            return "Apagadas"
26
27
28 miCoche = Coche()
29 print("Luces: ", miCoche.estadoLuces())
30 miCoche.arrancar()
31 print("Luces: ", miCoche.estadoLuces())
32 miCoche.__luces = True
33 print("Luces: ", miCoche.estadoLuces())
34 miCoche.encenderLuces()
35 print("Luces: ", miCoche.estadoLuces())
36 miCoche.parar()
37 print("Luces: ", miCoche.estadoLuces())
38
39
```

Line 38, Column 1 Tab Size: 4 Python

Al llamar a la propiedad `__luces` no podrá ser modificada desde afuera del programa. Veamos ahora la ejecución y discutamos lo que va pasando:



```
C:\Windows\system32\cmd.exe
D:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python>python oop.py
Luces: Apagadas
Luces: Apagadas
Luces: Apagadas
Luces: Apagadas
Luces: Encendidas
Luces: Apagadas

D:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python>
```

El primer print nos muestra el estado inicial del coche.

Luego aplicamos el método arrancar el cual no enciende las luces por lo que cuando hacemos el segundo print siguen apagadas.

Luego intentamos hacer trampa y encender las luces directo de la propiedad. Volvemos a consultar por tercera vez el estado de las luces y siguen apagadas.

Finalmente encendemos las luces de la manera correcta usando el método que creamos a tal efecto y vemos que se encienden.

Luego apagamos el motor y preguntamos por las luces. Al ver el quinto print confirmamos que las luces están ahora apagadas por lo que no se nos agotará la batería. Misión cumplida.

Dejamos para el lector verificar que si trata de encender las luces con el motor apagado no lo consigue.

Así como pudimos encapsular las propiedades de una clase podemos también encapsular los métodos. Para eso le agregamos, del mismo modo, dos guiones bajos en el inicio del nombre del método.

Herencia

Ya hemos visto que, para una clase podemos definir propiedades y métodos.

A la hora de definir una clase podemos también hacerlo a partir de una clase. De esta manera, además de sus propios métodos y propiedades la nueva clase “hereda” los métodos y propiedades de la clase desde la que la hemos definido.

La herencia sirve para reutilizar código que, de otra forma, debería existir por separado para padres e hijos de una relación de herencia.

Como los hijos podrán, a su vez, tener también hijos esa cadena de reutilización no tiene límites.

Para señalar que estamos heredando la clase padre desde la clase hijo debemos primero definir la clase padre y al definir la clase hijo ponemos:

Class Hijo(Padre):

Y a continuación continuamos creando métodos y propiedades que sean propios del hijo.

Hagamos un ejemplo para fijar ideas.

Clase Padre:

- Vehículos ○
 - Marca ○ Modelo
 - EnMarcha ○
 - Frenado ○
 - Cambio

Clase Hijo:

- Motos ○
 - PieEstacionamiento
nto
- Automovil ○
 - PuertaDD ○
 - PuertaDI ○
 - PuertaTD ○
 - PuertaTI
- Camión ○
 - PuertaD ○
 - PuertaL ○ Carga

Por ejemplo:

```

1  class Vehiculo():
2      def __init__(self,marca,modelo):
3          self.__marca = marca
4          self.__modelo = modelo
5      def marca(self):
6          return self.__marca
7      def modelo(self):
8          return self.__modelo
9
10     class Coche(Vehiculo):
11
12         def __init__(self,Marca,Modelo):
13             self.__estadomotor = False
14             self.__estadoluces = False
15             super().__init__(Marca,Modelo)
16         def encenderMotor(self):
17             self.__estadomotor = True
18         def apagarMotor(self):
19             self.__estadomotor = False
20             self.__estadoluces = False
21         def encenderLuces(self):
22             if self.__estadomotor:
23                 self.__estadoluces = True
24             else:
25                 print("No se encenderán las luces sin encender primero el motor")
26         def apagarLuces(self):
27             self.__estadoluces = False
28         def mostrarEstado(self):
29             print("Marca: ",self.marca()," Modelo: ",self.modelo()," Motor: ",self.__estadomotor," Luces: ",self.__estadoluces)
30
31     class Patineta(Vehiculo):
32
33         def __init__(self,Marca,Modelo):
34             self.__estadorulemanes = True
35             super().__init__(Marca,Modelo)
36         def romperRulemanes(self):
37             self.__estadorulemanes = False
38         def repararRulemanes(self):
39             self.__estadorulemanes = True
40         def mostrarEstado(self):
41             print("Marca: ",self.marca()," Modelo: ",self.modelo()," Estado Rulemanes: ",self.__estadorulemanes)
42

```

Supercargar métodos

Cuando definimos un método para un hijo que se llama igual que el método de la clase padre entonces, al invocar ese método en el hijo se invoca al propio método del hijo y el del padre queda oculto.

La función `super()` la utilizamos para llamar a los métodos de la clase padre en vez de los métodos del mismo nombre de la clase hijo.

Dentro del código de la clase del hijo, para invocar a un método del padre en vez de a un método del hijo con el mismo nombre se escribe:

`super().metodoPadre(parámetros del método del padre)`

Herencia múltiple:

Nos puede interesar que una clase herede las características de varias clases preexistentes. Entonces, al definir el nuevo hijo indicamos la lista de sus padres:

```
class Hijo(Padre1,Padre2,...,PadreN)
```

A la hora de invocar un método que exista en varias de las clases se da preferencia a la clase que se menciona primero.

Esto no debería ser un problema con los métodos que nosotros vayamos inventando ya que si las clases hablan de objetos distintos no tienen por qué llamarse igual sus métodos. Sin embargo esto es una dificultad para el método constructor que se llama, por necesidad, igual en todas las clases.

Función `isinstance()`

Nos informa si un objeto es una instancia de una clase:

Su uso es:

```
isinstance(Objeto, Clase)
```

Devuelve `True` si el objeto es una implementación directa de la clase o un descendiente de la clase y `False` si no lo es.

Realicemos un ejemplo concreto de los conceptos que nos ayudará a comprender lo que hemos explicado:



The screenshot shows a Python code editor window with the following details:

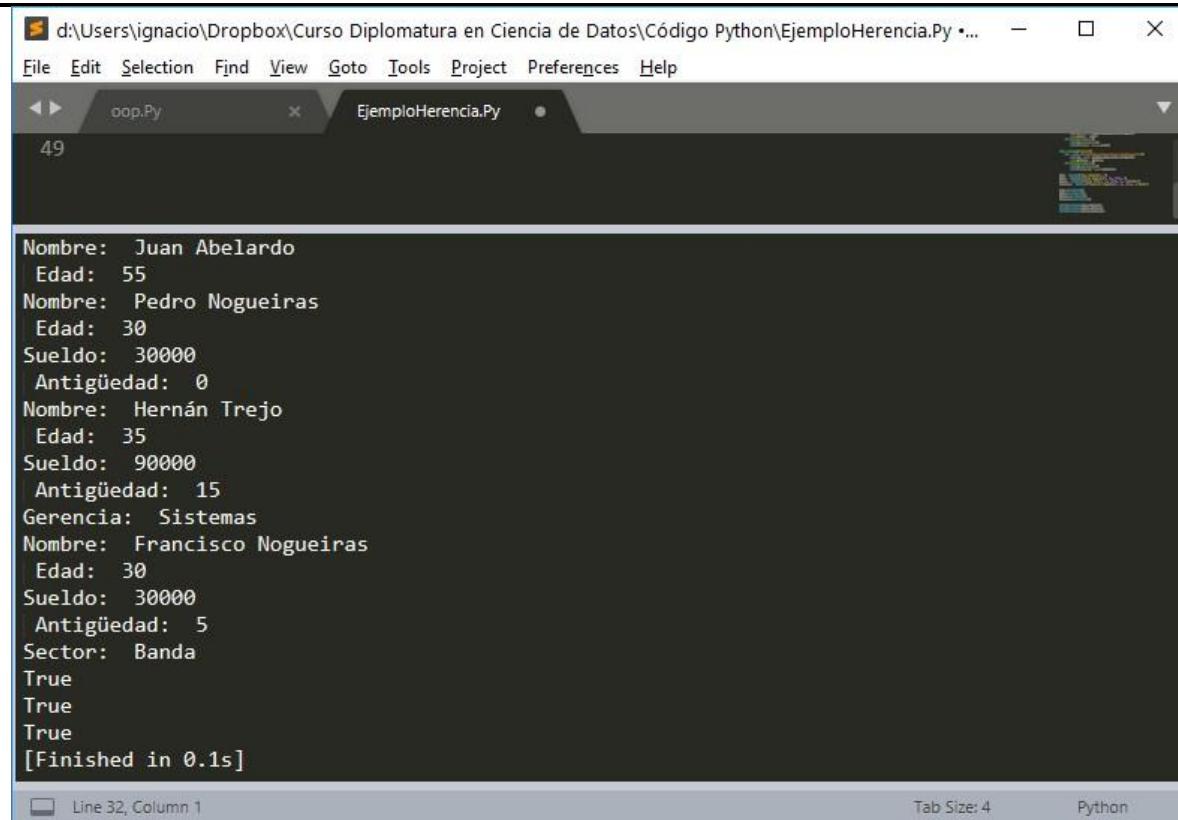
- Title Bar:** d:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python\EjemploHerencia.Py
- Menu Bar:** File Edit Selection Find View Goto Tools Project Preferences Help
- Tab Bar:** oop.py (active tab) and EjemploHerencia.py
- Code Area:**

```

1 class Persona():
2     def __init__(self,nombre,edad):
3         self.nombre = nombre
4         self.edad = edad
5     def descripcion(self):
6         print("Nombre: ",self.nombre,"\n","Edad: ",self.edad)
7
8 class Empleado(Persona):
9     def __init__(self,nombre,edad,sueldo,antiguedad):
10        super().__init__(nombre,edad)
11        self.sueldo = sueldo
12        self.antiguedad = antiguedad
13    def descripcion(self):
14        super().descripcion()
15        print("Sueldo: ",self.sueldo,"\\n","Antigüedad: ",self.antiguedad)
16
17 class Operario(Empleado):
18     def __init__(self,nombre,edad,sueldo,antiguedad,sector):
19        super().__init__(nombre,edad,sueldo,antiguedad)
20        self.sector = sector
21    def descripcion(self):
22        super().descripcion()
23        print("Sector: ",self.sector)
24
25 class Gerente(Empleado):
26     def __init__(self,nombre,edad,sueldo,antiguedad,gerencia):
27        super().__init__(nombre,edad,sueldo,antiguedad)
28        self.gerencia = gerencia
29    def descripcion(self):
30        super().descripcion()
31        print("Gerencia: ",self.gerencia)
32
33 juan = Persona("Juan Abelardo", 55)
34 pedro = Empleado("Pedro Nogueiras", 30, 30000, 0)
35 hernan = Gerente("Hernán Trejo", 35, 90000, 15, "Sistemas")
36 francisco = Operario("Francisco Nogueiras", 30, 30000, 5,"Banda")
37
38 juan.descripcion()
39 pedro.descripcion()
40 hernan.descripcion()
41 francisco.descripcion()
42
43 print(isinstance(juan,Persona))
44 print(isinstance(pedro,Persona))
45 print(isinstance(hernan,Persona))
46
47

```
- Status Bar:** Line 32, Column 1 | Tab Size: 4 | Python

Y al ejecutar obtenemos:



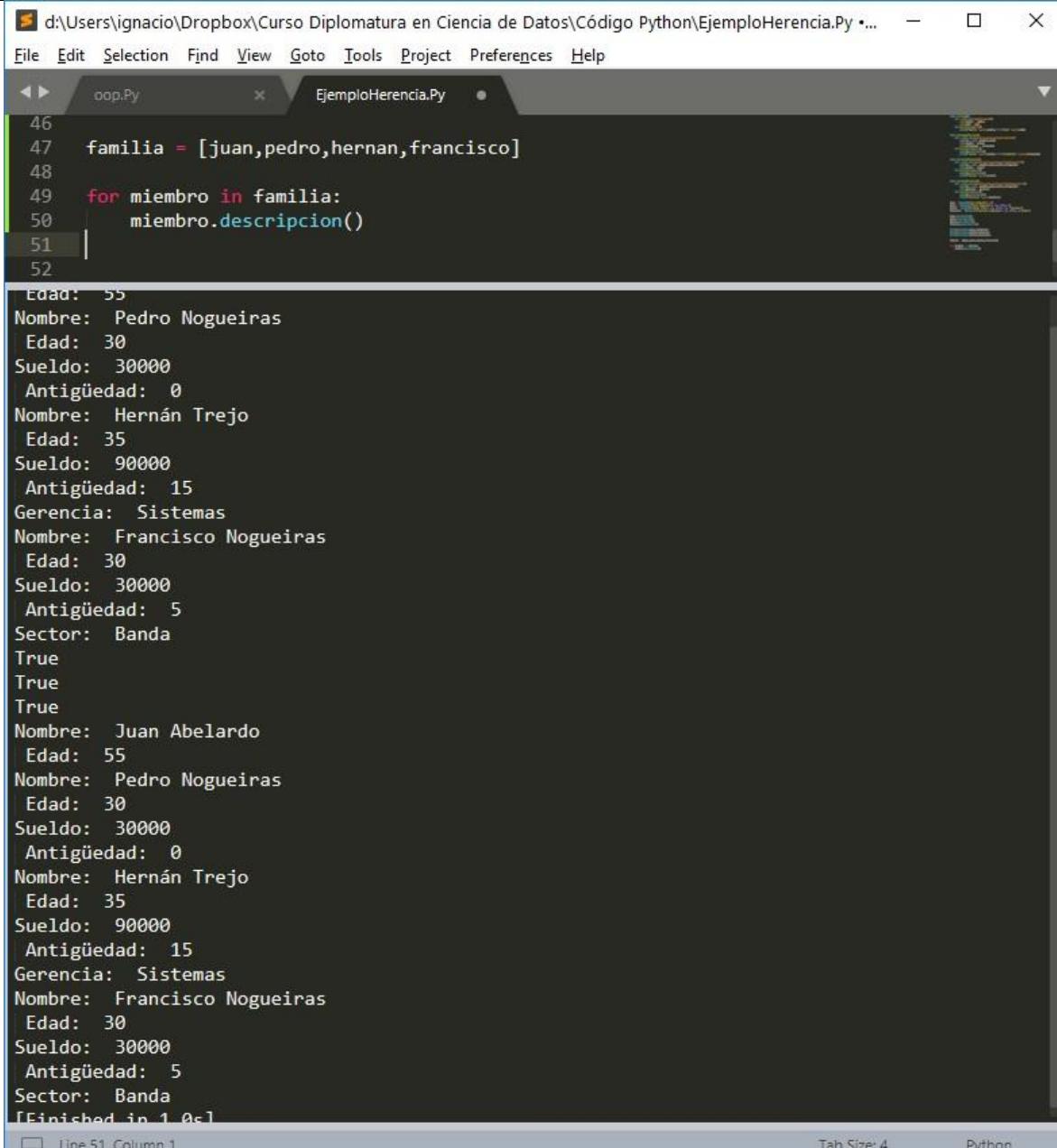
A screenshot of a Python code editor window titled "EjemploHerencia.Py". The window shows two tabs: "oop.Py" and "EjemploHerencia.Py", with "EjemploHerencia.Py" active. The code in "EjemploHerencia.Py" demonstrates inheritance from the "Employee" class to the "Banda" class. The output pane displays the results of running the code, showing instances of the classes with their respective attributes: Nombre, Edad, Sueldo, Antigüedad, Gerencia, and Sector. The output ends with "[Finished in 0.1s]". The status bar at the bottom indicates "Line 32, Column 1" and "Tab Size: 4".

Polimorfismo:

Usamos polimorfismo cuando tenemos clases diferentes que tienen todos un método que se llama igual.

Entonces, cuando para cada objeto invocamos ese método se comporta cada uno de la manera prevista para esa clase.

Por ejemplo, si agregamos a nuestro ejemplo anterior:



The screenshot shows a Python code editor with two tabs: 'oop.py' and 'EjemploHerencia.py'. The 'EjemploHerencia.py' tab is active, displaying the following code:

```
46 familia = [juan,pedro,hernan,francisco]
47 for miembro in familia:
48     miembro.descripcion()
49
50
51
52 Edad: 55
Nombre: Pedro Nogueiras
Edad: 30
Sueldo: 30000
Antigüedad: 0
Nombre: Hernán Trejo
Edad: 35
Sueldo: 90000
Antigüedad: 15
Gerencia: Sistemas
Nombre: Francisco Nogueiras
Edad: 30
Sueldo: 30000
Antigüedad: 5
Sector: Banda
True
True
True
Nombre: Juan Abelardo
Edad: 55
Nombre: Pedro Nogueiras
Edad: 30
Sueldo: 30000
Antigüedad: 0
Nombre: Hernán Trejo
Edad: 35
Sueldo: 90000
Antigüedad: 15
Gerencia: Sistemas
Nombre: Francisco Nogueiras
Edad: 30
Sueldo: 30000
Antigüedad: 5
Sector: Banda
[Finished in 1.0s]
```

The output window shows the execution results for each member of the 'familia' list, demonstrating the execution of the 'descripcion()' method for each object.

Como cada objeto de la colección es de una clase distinta pero todas las clases tienen su método descripción siempre se ejecuta en la forma prevista.

Conviene aclarar que no es necesario que las distintas clases estén relacionadas por herencia. Alcanza con que comparten simplemente el nombre del método.

Manejo de archivos separados por comas:

Ya conseguimos manejar archivos en texto plano.

Con lo que sabemos de expresiones regulares podríamos tratar de identificar las columnas, separarlas, investigar los tipos, convertirlos, pero...

Para que pelarnos la cabeza probando y, lo que es peor, ajustando, nuestro software cuando podemos usar algo mucho más probado y disponible sin pagar: el paquete csv.

Los comandos principales son:

```
open csv.reader  
csv.writer  
csv.DictReader  
csv.DictWriter  
close
```

Empecemos por leer un archivo de prueba que hemos armado especialmente para la ocasión:

Entero	Flotante	Nombre	Fecha
1	1,5	XQMSA	16/10/2019
2	2,5	AWILJ	15/10/2019
3	3,5	RGWAF	14/10/2019
4	4,5	BANUR	13/10/2019
5	5,5	QEUNE	12/10/2019
6	6,5	HCHXL	11/10/2019
7	7,5	CMZTB	10/10/2019
8	8,5	OQOVI	9/10/2019
9	9,5	REYAZ	8/10/2019
10	10,5	CUSFB	7/10/2019

El código Python para leerlo es:

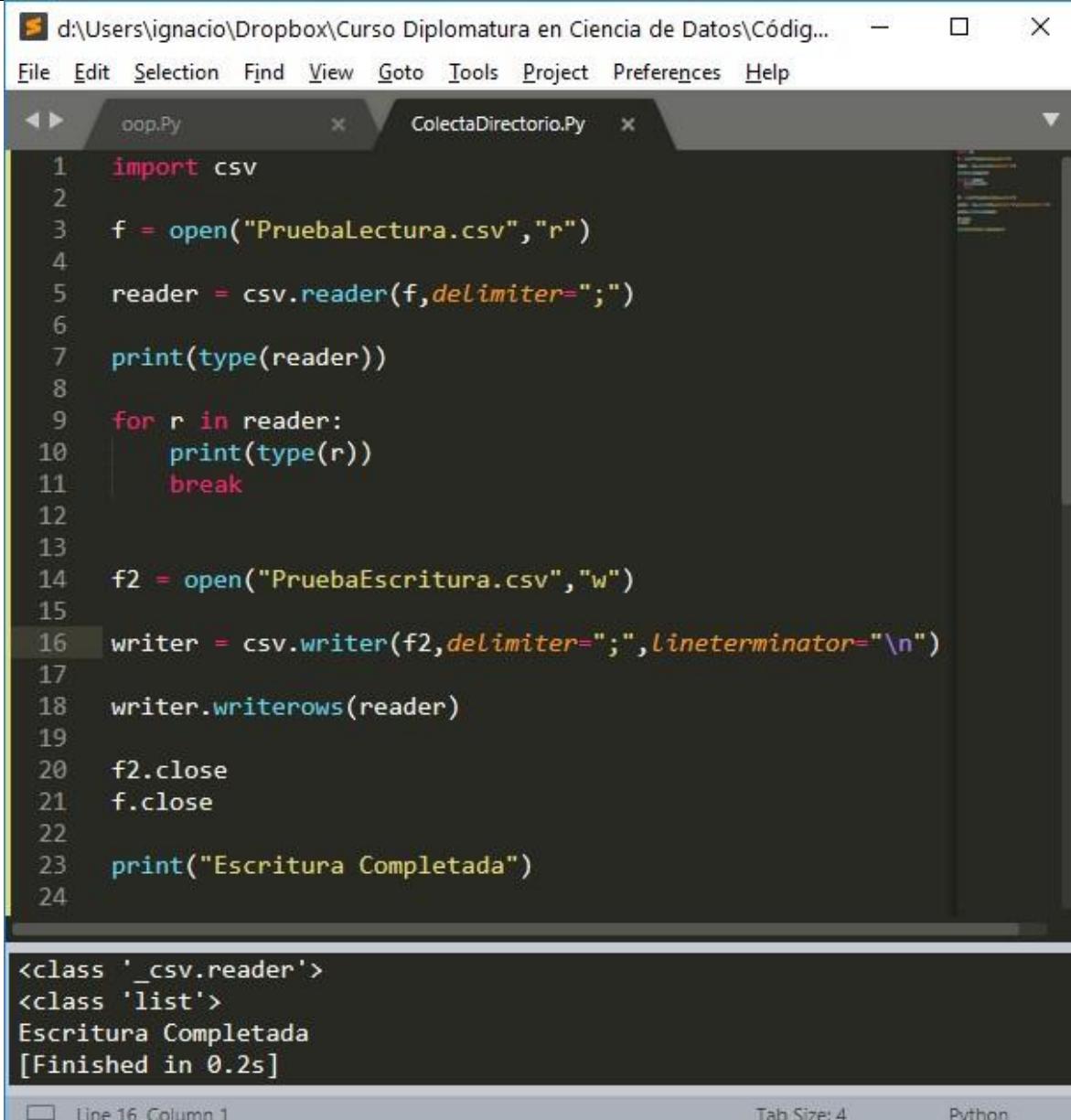
```
d:\Users\ignacio\Dropbox\Curso Diplomatura ... ━ ━ ━ X
File Edit Selection Find View Goto Tools Project Preferences Help
← → oop.Py × ColectaDirectorio.Py
1 import csv
2
3 f = open("PruebaLectura.csv","r")
4
5 reader = csv.reader(f,delimiter=";")
6
7 for row in reader:
8     print(row)
9
10 f.close()
11
12
['Entero', 'Flotante', 'Nombre', 'Fecha']
[ '1', '1,5', 'XQMSA', '16/10/2019' ]
[ '2', '2,5', 'AWILJ', '15/10/2019' ]
[ '3', '3,5', 'RGWAF', '14/10/2019' ]
[ '4', '4,5', 'BANUR', '13/10/2019' ]
[ '5', '5,5', 'QEUNE', '12/10/2019' ]
[ '6', '6,5', 'HCHXL', '11/10/2019' ]
[ '7', '7,5', 'CMZTB', '10/10/2019' ]
[ '8', '8,5', 'OQOVI', '9/10/2019' ]
[ '9', '9,5', 'REYAZ', '8/10/2019' ]
[ '10', '10,5', 'CUSFB', '7/10/2019' ]
Line 2, Column 1 Tab Size: 4 F
```

Cada renglón del archivo es leído en un objeto de la clase `_csv.reader`. Cada renglón es también una colección de los valores que van tomando los campos.

Notamos que:

- Todos los campos los reconoce inicialmente como strings
- Los puntos decimales los refleja como comas. Si necesitamos convertir ese número a flotante vamos a precisar reemplazarlo por un punto.

Vamos ahora a escribir un archivo plano con la misma información:



```

d:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código...
File Edit Selection Find View Goto Tools Project Preferences Help
oop.py ColectaDirectorio.py
1 import csv
2
3 f = open("PruebaLectura.csv","r")
4
5 reader = csv.reader(f,delimiter=";")
6
7 print(type(reader))
8
9 for r in reader:
10     print(type(r))
11     break
12
13
14 f2 = open("PruebaEscritura.csv","w")
15
16 writer = csv.writer(f2,delimiter=";",lineterminator="\n")
17
18 writer.writerows(reader)
19
20 f2.close
21 f.close
22
23 print("Escritura Completada")
24
<class '_csv.reader'>
<class 'list'>
Escritura Completada
[Finished in 0.2s]

```

Line 16, Column 1 Tab Size: 4 Python

Debemos resaltar que para obtener lo que teníamos tuvimos que hacer dos aclaraciones:

- delimiter = ";"
- lineterminator="\n"

Si omitimos el delimiter el Excel nos toma todos los campos en la misma celda.

Si omitimos el lineterminator entonces va dejando un renglón libre intermedio pues el terminador por defecto es \r\n

Otra cosa que notamos es que necesitamos mantener f abierto para que reader contenta los datos. Si no hemos sacado los datos de un archivo necesitamos tenerlos en una estructura que es una lista de listas. En particular una lista de renglones y cada renglón una lista de campos.

Tratemos de generar y grabar un archivo a partir de una estructura que hayamos generado desde los tipos básicos.

```
d:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código... - □ ×
File Edit Selection Find View Goto Tools Project Preferences Help
oop.py < ColectaDirectorio.Py >
1 import csv
2
3 encabezado = ["Entero","Flotante","Texto","Fecha"]
4
5 l = []
6
7 l.append(encabezado)
8
9 for i in range(100):
10     r = []
11     r.append(i)
12     r.append(float(i)+0.5)
13     r.append("abcde")
14     r.append("10/10/2019")
15     l.append(r)
16
17
18 f2 = open("PruebaEscritura2.csv","w")
19
20 writer = csv.writer(f2,delimiter=";",lineterminator="\n")
21
22 writer.writerows(l)
23
24 f2.close

Escritura Completada
[Finished in 0.3s]

Line 1, Column 1 Tab Size: 4 Python
```

Vamos ahora a utilizar DictReader y DictWriter sobre los mismos conjuntos que hemos estado trabajando:

The screenshot shows a code editor window with a dark theme. At the top, there's a menu bar with options like File, Edit, Selection, Find, View, Goto, Tools, Project, Preferences, and Help. Below the menu, there are two tabs: 'oop.py' and 'ColectaDirectorio.py'. The 'oop.py' tab is active, displaying the following Python code:

```
1 import csv
2
3 f = open("PruebaEscritura2.csv","r")
4
5 reader = csv.DictReader(f, delimiter = ";")
6 print(reader.fieldnames)
7
8 for r in reader:
9     print(r["Entero"],r["Flotante"],r["Texto"],r["Fecha"])
10
11 f.close
12
```

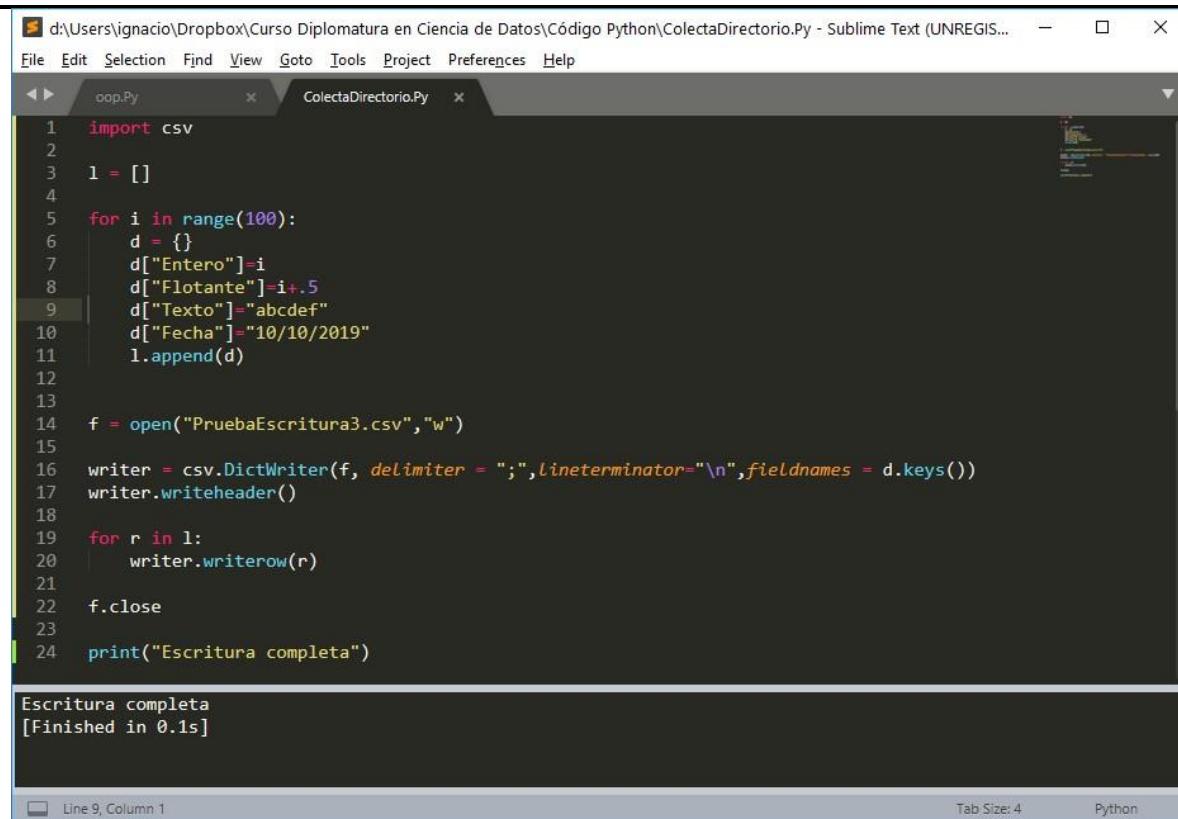
Below the code, the output of the script is shown in the terminal window:

```
['Entero', 'Flotante', 'Texto', 'Fecha']
0 0.5 abcde 10/10/2019
1 1.5 abcde 10/10/2019
2 2.5 abcde 10/10/2019
3 3.5 abcde 10/10/2019
4 4.5 abcde 10/10/2019
5 5.5 abcde 10/10/2019
6 6.5 abcde 10/10/2019
7 7.5 abcde 10/10/2019
8 8.5 abcde 10/10/2019
9 9.5 abcde 10/10/2019
10 10.5 abcde 10/10/2019
11 11.5 abcde 10/10/2019
12 12.5 abcde 10/10/2019
13 13.5 abcde 10/10/2019
14 14.5 abcde 10/10/2019
```

At the bottom of the terminal window, there are status indicators: 'Line 11, Column 1', 'Tab Size: 4', and 'Python'.

Ya aprendimos a leer directamente en un objeto que podemos recorrer como si fuera una lista de diccionarios. Es lo más parecido al concepto de recordset o de dataframe.

Vamos ahora a aprender a escribir.



The screenshot shows a Sublime Text window with two tabs: 'oop.py' and 'ColectaDirectorio.py'. The 'oop.py' tab contains a short Python script that imports the 'csv' module, creates a list 'l', and appends 100 dictionaries 'd' to it. Each dictionary has keys 'Entero', 'Flotante', 'Texto', and 'Fecha' with values ranging from 0 to 10. The 'ColectaDirectorio.py' tab contains a script that opens a CSV file named 'PruebaEscritura3.csv' in write mode, creates a CSV writer object, and writes each dictionary from 'l' as a row. Finally, it prints 'Escritura completa'.

```

1 import csv
2
3 l = []
4
5 for i in range(100):
6     d = {}
7     d["Entero"] = i
8     d["Flotante"] = i+.5
9     d["Texto"] = "abcdef"
10    d["Fecha"] = "10/10/2019"
11    l.append(d)
12
13
14 f = open("PruebaEscritura3.csv", "w")
15
16 writer = csv.DictWriter(f, delimiter = ";", lineterminator = "\n", fieldnames = d.keys())
17 writer.writeheader()
18
19 for r in l:
20     writer.writerow(r)
21
22 f.close()
23
24 print("Escritura completa")

```

Escritura completa
[Finished in 0.1s]

Y, por supuesto, el resultado es:

Entero	Flotante	Texto	Fecha
0	0.5	abcdef	10/10/2019
1	1.5	abcdef	10/10/2019
2	2.5	abcdef	10/10/2019
3	3.5	abcdef	10/10/2019
4	4.5	abcdef	10/10/2019
5	5.5	abcdef	10/10/2019
6	6.5	abcdef	10/10/2019
7	7.5	abcdef	10/10/2019
8	8.5	abcdef	10/10/2019
9	9.5	abcdef	10/10/2019
10	10.5	abcdef	10/10/2019

Archivos de Excel

Vamos a empezar por leer archivos de Microsoft Excel desde Python.

Microsoft solo proporciona APIs para sus plataformas de desarrollo. Esto resulta complicado desde la perspectiva de Python que está concebido como una plataforma libre y queda, por lo tanto, sin soporte oficial ante un formato de archivo privativo.

Sin embargo, gracias a la comunidad de desarrollo se ha creado una librería (no oficial) para la manipulación casi total de estos archivos que se llama OpenPyxl.

Lo primero que debemos hacer es instalar la librería openpyxl para que nuestro interprete Python lo encuentre.

```
$pip install openpyxl
```

Una vez instalada la librería podemos comenzar a escribir código, recomiendo descargar el siguiente documento de Excel para seguir el tutorial (puedes usar uno propio, si lo deseas).

El modelo de objetos de Excel:

- Workbook ○ Tupla de worksheets ■ Tupla de filas
 - Tupla de columnas ○ Cada columna tiene un valor

Vamos a practicar con la lectura del archivo de Excel que usé para construir el csv que veníamos trabajando:

The screenshot shows a Python code editor with two tabs: 'oop.py' and 'ColectaDirectorio.py'. The 'oop.py' tab contains the following code:

```
1 import openpyxl
2
3 doc = openpyxl.load_workbook('PruebaLecturaExcel.xlsx')
4
5 for s in doc:
6     try:
7         for r in s:
8             try:
9                 for c in r:
10                     print(c.value)
11             except TypeError:
12                 break
13     except TypeError:
14         break
15
16 doc.close()
17 |
18
19
20
21
```

The output window below the code editor displays the printed values:

```
Entero
Flotante
Nombre
Fecha
1
1.5
HHBDS
2019-10-16 00:00:00
2
2.5
ZACNT
2019-10-15 00:00:00
3
3.5
AZPFC
2019-10-14 00:00:00
4
4.5
PCLRE
2019-10-13 00:00:00
5
```

At the bottom of the interface, there is a status bar with the text "Line 17, Column 1", "Tab Size: 4", and "Python".

Ahora vamos a tratar de guardar los resultados obtenidos en un diccionario:

```
import openpyxl
doc = openpyxl.load_workbook('PruebaLecturaExcel.xlsx')
ls = []
for s in doc:
    lr = []
    try:
        for r in s:
            lc = []
            try:
                for c in r:
                    lc.append(c.value)
            except TypeError:
                break
            lr.append(lc)
    except TypeError:
        break
    ls.append(lr)
doc.close()
ns = 1
for s in ls:
    print("Hoja: ", ns)
    for r in s:
        print(r)
    ns = ns+1
```

Hoja: 1
['Entero', 'Flotante', 'Nombre', 'Fecha']
[1, 1.5, 'HHBDS', datetime.datetime(2019, 10, 16, 0, 0)]
[2, 2.5, 'ZACNT', datetime.datetime(2019, 10, 15, 0, 0)]
[3, 3.5, 'AZPFC', datetime.datetime(2019, 10, 14, 0, 0)]
[4, 4.5, 'PCLRE', datetime.datetime(2019, 10, 13, 0, 0)]
[5, 5.5, 'LQSQP', datetime.datetime(2019, 10, 12, 0, 0)]
[6, 6.5, 'RBYLG', datetime.datetime(2019, 10, 11, 0, 0)]
[7, 7.5, 'KXMWY', datetime.datetime(2019, 10, 10, 0, 0)]
[8, 8.5, 'SNQLN', datetime.datetime(2019, 10, 9, 0, 0)]
[9, 9.5, 'ZMMBR', datetime.datetime(2019, 10, 8, 0, 0)]
[10, 10.5, 'XTLSE', datetime.datetime(2019, 10, 7, 0, 0)]
[Finished in 1.3s]

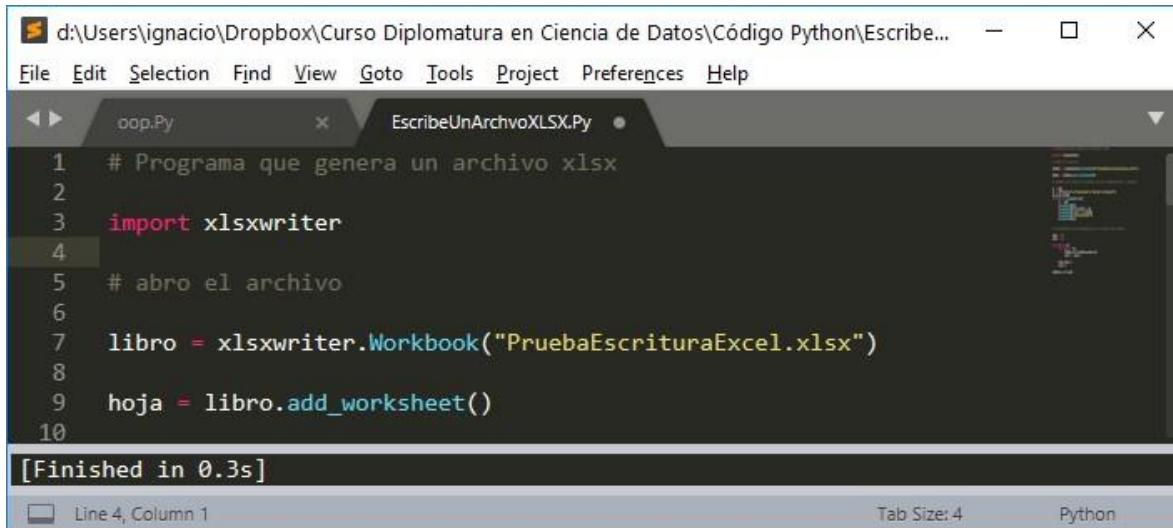
Nuestro siguiente paso es escribir desde cero un archivo Excel. Para eso vamos a seguir los pasos detallados en: <https://donnierock.com/2018/12/09/escribir-un-fichero-excel-desde-python/>

Para instalarlo hacemos:

```
pip install XlsxWriter
```

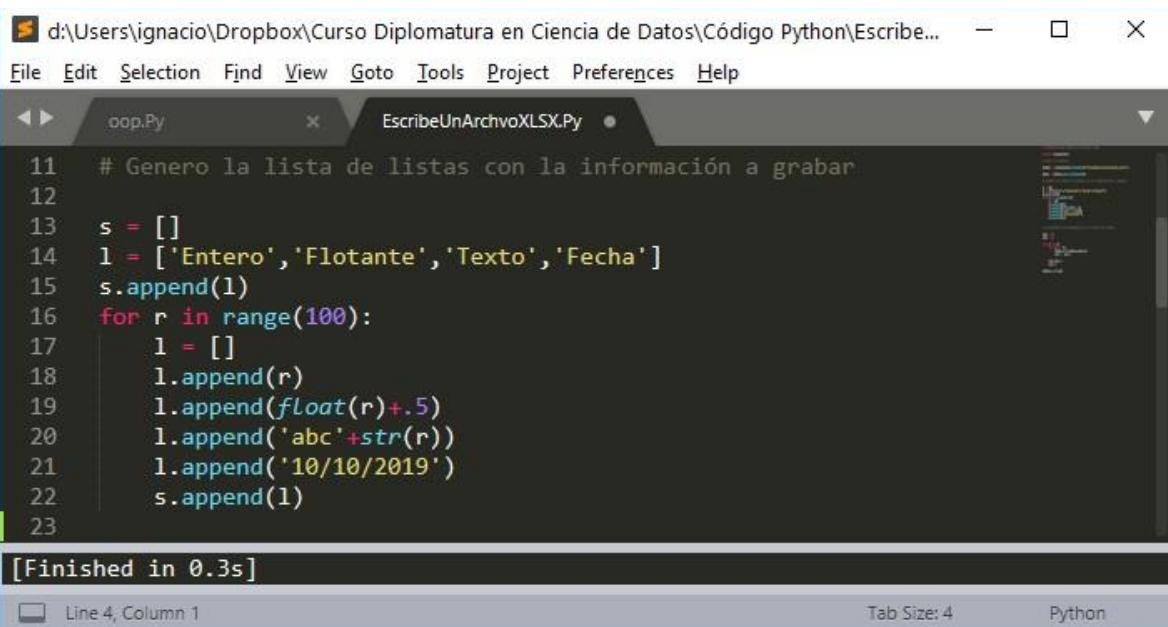
Partimos con la ventaja de que ya conocemos el modelo de objetos de Excel.

El primer paso es importar la librería y crear el libro y agregarle una hoja:



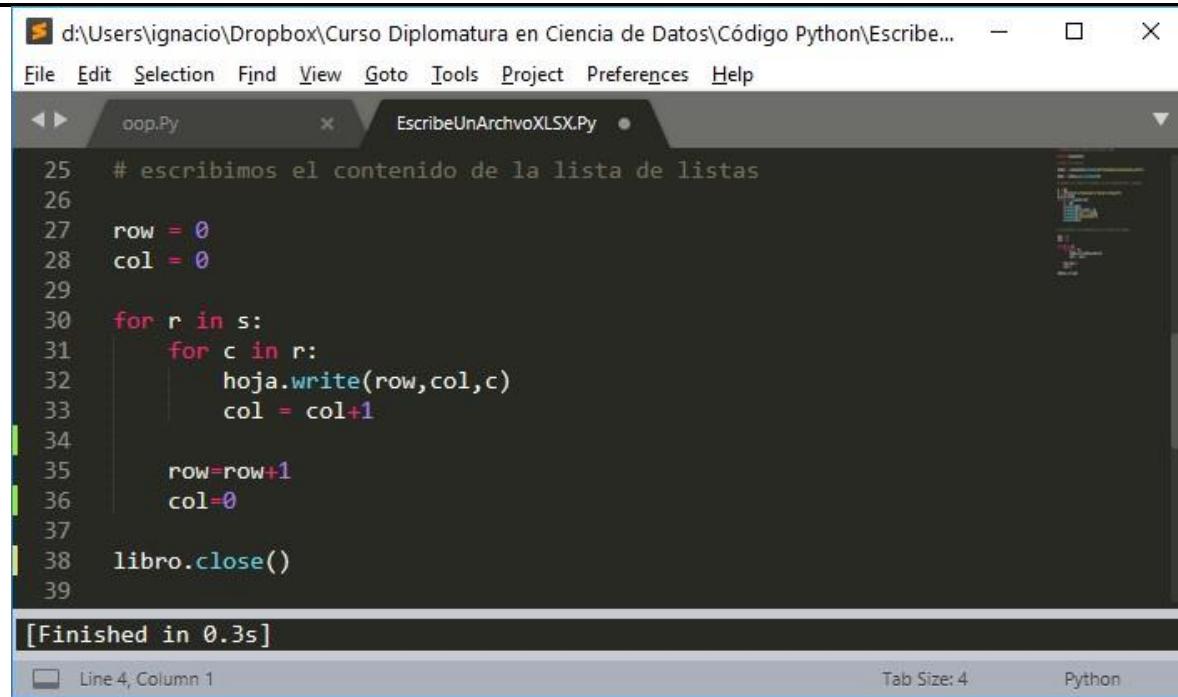
```
File Edit Selection Find View Goto Tools Project Preferences Help
oop.py EscribeUnArchivoXLSX.Py
1 # Programa que genera un archivo xlsx
2
3 import xlsxwriter
4
5 # abro el archivo
6
7 libro = xlsxwriter.Workbook("PruebaEscrituraExcel.xlsx")
8
9 hoja = libro.add_worksheet()
10
[Finished in 0.3s]
Line 4, Column 1 Tab Size: 4 Python
```

Luego generamos la lista de listas con los datos que queremos guardar:



```
File Edit Selection Find View Goto Tools Project Preferences Help
oop.py EscribeUnArchivoXLSX.Py
11 # Genero la lista de listas con la información a grabar
12
13 s = []
14 t = ['Entero', 'Flotante', 'Texto', 'Fecha']
15 s.append(t)
16 for r in range(100):
17     l = []
18     l.append(r)
19     l.append(float(r)+.5)
20     l.append('abc'+str(r))
21     l.append('10/10/2019')
22     s.append(l)
23
[Finished in 0.3s]
Line 4, Column 1 Tab Size: 4 Python
```

Y, finalmente, nos toca grabar el texto:



The screenshot shows a code editor window with the following details:

- Title Bar:** d:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python\Escribe... - EscribeUnArchivoXLSX.Py
- Menu Bar:** File Edit Selection Find View Goto Tools Project Preferences Help
- Code Area:** The code is written in Python and performs the following steps:
 - Imports the `openpyxl` module.
 - Creates a new workbook and selects the first sheet.
 - Initializes variables `row` and `col` to 0.
 - Iterates over each row `r` in the list `s`. For each row, it iterates over each column `c` and writes the value `c` to the cell at position `(row, col)`.
 - After processing all columns in a row, increments the `row` index by 1.
 - After processing all rows, closes the workbook.
- Status Bar:** [Finished in 0.3s], Line 4, Column 1, Tab Size: 4, Python

Nos queda ahora abrir el archivo y ver si tuvimos éxito:

The screenshot shows a Microsoft Excel spreadsheet titled "PruebaEscrituraExcel - Micros...". The ribbon menu is visible at the top, showing tabs like Inicio, Insertar, Diseño de página, Fórmulas, Datos, Revisar, and Vista. The formula bar at the top has "A1" selected and displays "fx Entero". The main area contains a table with 12 rows and 5 columns. The columns are labeled A, B, C, D, and E. The first row (header) contains the values "Entero", "Flotante", "Texto", "Fecha", and an empty cell. Rows 2 through 12 contain numerical values from 0 to 10 in column A, and corresponding floating-point numbers, text strings, and dates in columns B, C, and D respectively. The date format is "10/10/2019". The table is styled with a light blue header row and standard white rows for data. The bottom of the screen shows the Excel ribbon, a status bar with "Sheet1", and a zoom level of "140%".

	A	B	C	D	E
1	Entero	Flotante	Texto	Fecha	
2	0	0,5	abc0	10/10/2019	
3	1	1,5	abc1	10/10/2019	
4	2	2,5	abc2	10/10/2019	
5	3	3,5	abc3	10/10/2019	
6	4	4,5	abc4	10/10/2019	
7	5	5,5	abc5	10/10/2019	
8	6	6,5	abc6	10/10/2019	
9	7	7,5	abc7	10/10/2019	
10	8	8,5	abc8	10/10/2019	
11	9	9,5	abc9	10/10/2019	
12	10	10,5	abc10	10/10/2019	

Lo hemos conseguido!

Normas sugeridas para generar código Python:

Una guía de estilo tiene que ver con la consistencia. La consistencia es importante entre distintos proyectos desarrollados en una organización, más importante dentro de cada proyecto, más importante aún dentro de cada módulo y terriblemente importante dentro de cada función.

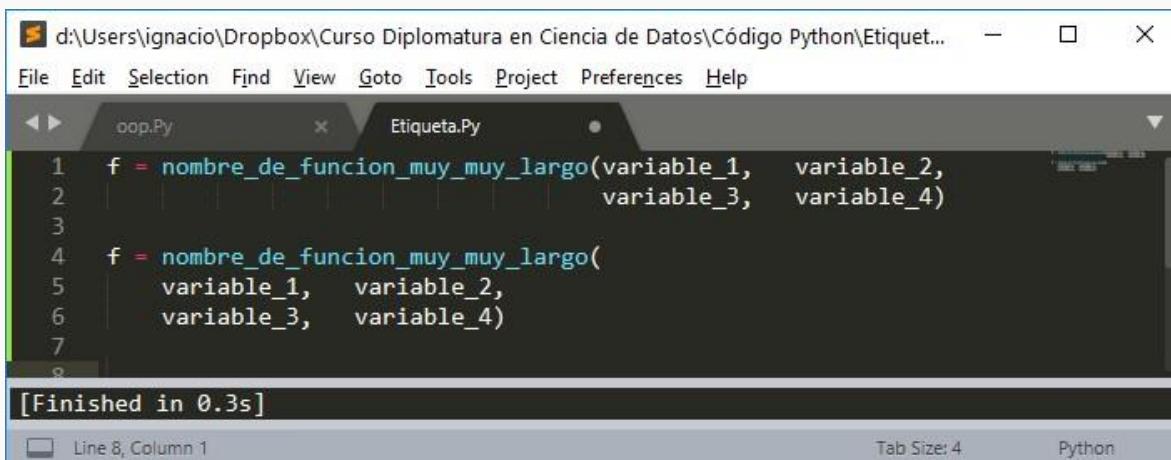
En cualquier caso no conviene romper la consistencia hacia atrás de un desarrollo sólo para cumplir con estas recomendaciones. Por ese motivo, si se está por comenzar a desarrollar es particularmente oportuno fijar las normas que proveerán consistencia a futuro.

Distribución del código en la hoja:

Indentación:

Usar 4 espacios para la indentación.

Cuando necesitamos continuar una línea larga en la siguiente debemos hacer que quede claro mediante la indentación cual es la situación:



```
oop.py
1 f = nombre_de_funcion_muy_muy_largo(variable_1,    variable_2,
2                                variable_3,    variable_4)
3
4 f = nombre_de_funcion_muy_muy_largo(
5     variable_1,    variable_2,
6     variable_3,    variable_4)
7
8
[Finished in 0.3s]
```

Lo que no deberíamos hacer es poner argumentos en la línea que vamos a interrumpir si no vamos a usar la alineación vertical.

```
foo = long_function_name(var_one, var_two,
var_three, var_four)
```

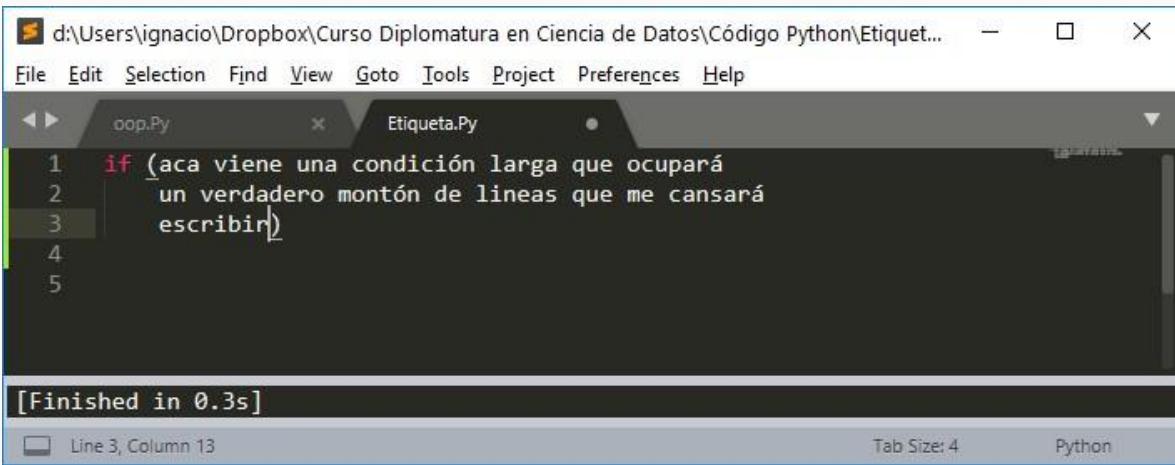
Tampoco vale usar una indentación que parezca normal y por lo tanto confunda:

```
def long_function_name(
```

```
var_one, var_two, var_three,  
var_four):  
  
    print(var_one)
```

Por lo tanto la regla de los cuatro espacios es opcional para las líneas de continuación y podría aportar más claridad otro criterio.

En el caso de que la condición de un if sea muy larga conviene aprovechar una circunstancia afortunada: el if más un espacio más un paréntesis genera 4 espacios. Usamos esos cuatro espacios para alinear todas las líneas de continuación que lleve la condición:



The screenshot shows a code editor window with two tabs: 'oop.py' and 'Etiqueta.py'. The 'oop.py' tab is active and contains the following code:

```
1 if (aca viene una condición larga que ocupará  
2     un verdadero montón de líneas que me cansará  
3     escribir)  
4  
5
```

The status bar at the bottom indicates '[Finished in 0.3s]'. The bottom right corner shows 'Tab Size: 4' and 'Python'.

Un comentario alentador, sublime texto lo organizó de manera automática!

Después de la condición no hay que usar indentación adicional:

```

1 if (aca viene una condición larga que ocupará
2     un verdadero montón de lineas que me cansará
3     escribir):
4
5     Hacer.Algo()
6
7

```

[Finished in 0.3s]

Line 5, Column 17 Tab Size: 4 Python

El paréntesis, corchete o llave que se cierra puede estar alineado con la última línea o bien con el nombre del objeto:

```

1 mi_lista = [1, 2, 3,
2     4, 5, 6
3 ]
4
5 mi_lista = [1, 2, 3,
6     4, 5, 6
7 ]
8

```

[Finished in 0.3s]

Line 7, Column 1 Tab Size: 4 Python

Tabuladores versus espacios:

Se recomienda el uso de espacios en blanco.

Máximo largo de línea:

Se sugiere reducir el ancho a 72 caracteres.

Para continuar en la siguiente línea se sugiere usar la continuación implícita de Python que se da cada vez que queda abierto un paréntesis, corchete o llave.

Como segunda alternativa menos preferible es usar la continuación explícita que se consigue poniendo una barra invertida como último carácter de la línea que se quiere continuar abajo.

```
File Edit Selection Find View Goto Tools Project Preferences Help  
oop.py x Etiqueta.py x  
1 mi_lista = [1, 2, 3,  
2     4, 5, 6  
3     ]  
4  
5 mi_string = "aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa\"  
6             aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa"  
7  
8  
[Finished in 0.1s]  
Line 7, Column 1 Tab Size: 4 Python
```

Cuando en una formula vamos a continuar en el siguiente renglón se aceptan las siguientes dos versiones con tal de que sea localmente consistente:

```
d:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python\Etiquet...
File Edit Selection Find View Goto Tools Project Preferences Help
oop.py Etiqueta.py

1 mi_resultado = (a +
2         b +
3         c +
4         d)
5
6 mi_resultado = (a
7     +b
8     +c
9     -d)

[Finished in 0.1s]
Line 9, Column 20
Tab Size: 4
Python
```

Para grupos que empiezan se sugiere usar la segunda opción.

Líneas en blanco:

Se recomienda rodear a las definiciones de alto nivel y a las clases con dos líneas en blanco.

Para las definiciones de métodos dentro de una clase se sugiere una línea en blanco.

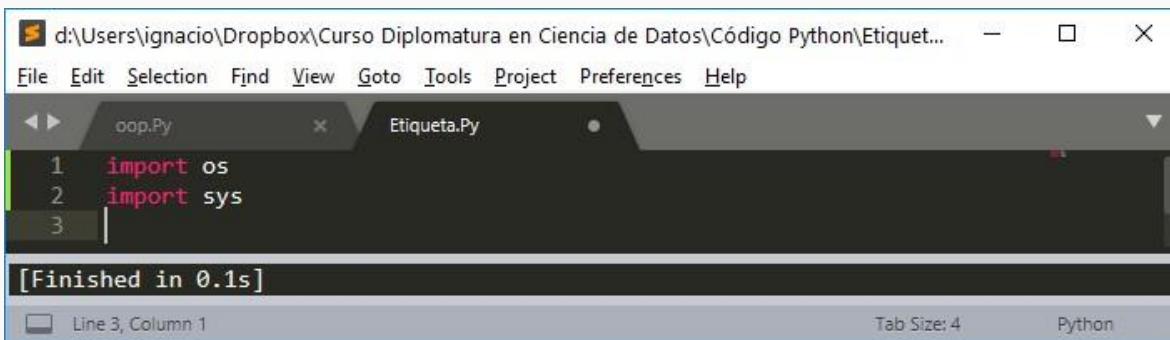
Para indicar divisiones lógicas de alto nivel dentro del código se pueden usar líneas adicionales.

Codificación de los archivos fuentes:

Se debe usar UTF-8

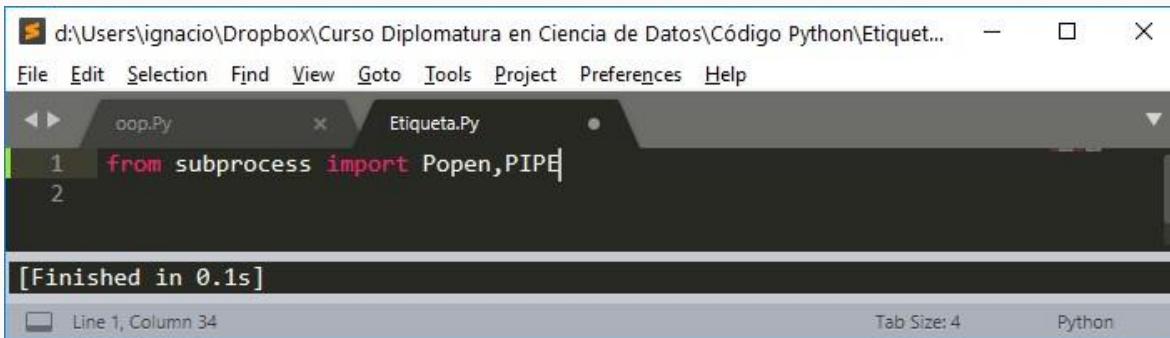
Importación de paquetes:

Los paquetes deben importarse en líneas separadas, al principio del archivo, sólo después de los comentarios del módulo y antes de las constantes y globales del módulo:



```
import os
import sys
```

Sin embargo se considera aceptable usar:



```
from subprocess import Popen,PIPE
```

Los imports se deben separar en tres grupos:

1. Librerías estándar
2. Librerías de terceras partes
3. Librerías locales

Cada uno de estos grupos debe estar separado de los otros por una línea en blanco.

Las importaciones absolutas son preferibles.

Solo es aceptable utilizar importaciones relativas cuando estamos en un entorno complejo y nos llevaría a perder legibilidad.

Se debe evitar el uso de * (comodines) en las sentencias import.

Comillas:

En Python las comillas dobles “ y las simples ‘ son equivalentes.

Se sugiere utilizar sólo un tipo de comillas.

Espacios en blanco:

Se sugiere evitar espacios en blanco en sitios extraños:

```
spam( ham[ 1 ], { eggs: 2 } )
```

No: bar = (0,)

No: if x == 4 : print x , y ; x , y = y , x

La forma correcta sería:

The screenshot shows a Python code editor window with three examples of bad whitespace usage:

```
spam( ham[ 1 ], { eggs: 2 } )
bar = (0, )
if x == 4 : print x , y ; x , y = y , x
```

[Finished in 0.1s]

Line 3, Column 22

Tab Size: 4

Python

Tampoco se pueden dejar espacios en blanco antes de un paréntesis, corchete o llave:

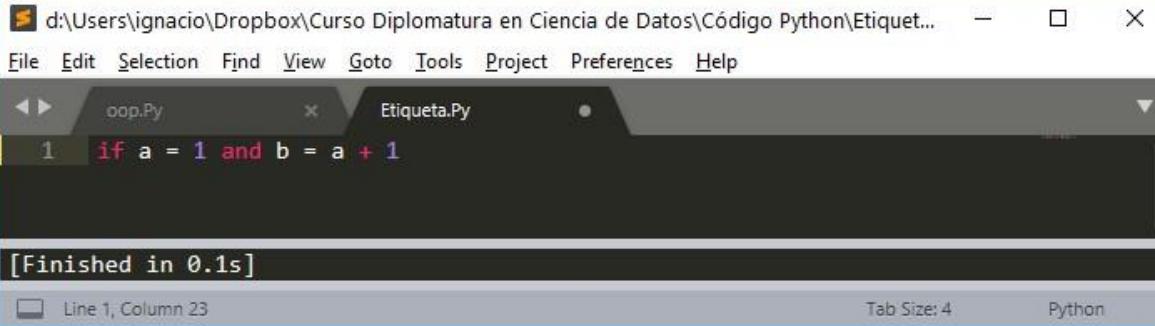
```
spam (1) dct ['key']
= lst [index]
```

Ni tampoco más de un espacio en blanco consecutivo.

```
x      =
1 y
= 2
```

Está prohibido dejar espacios en blanco al final de las líneas pues son invisibles.

Siempre rodear es espacios en blanco los operadores binarios.

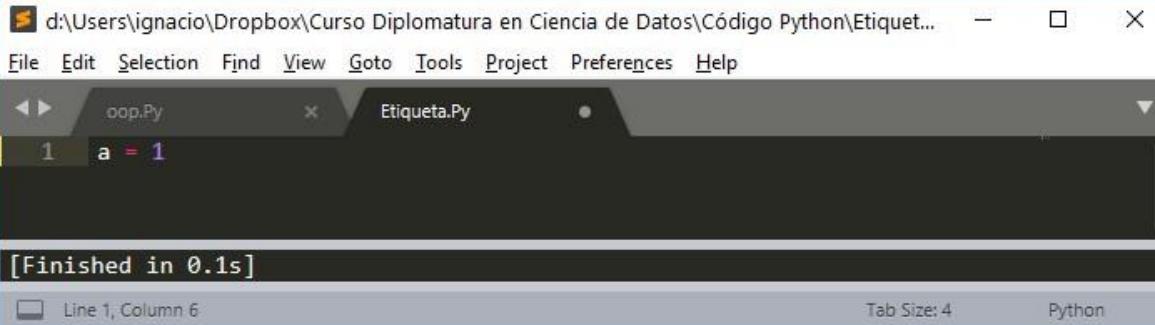


A screenshot of a Python code editor window titled "d:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python\Etiquet...". The tabs show "oop.py" and "Etiqueta.py". The code in "oop.py" is:

```
1 if a = 1 and b = a + 1
```

The status bar at the bottom shows "[Finished in 0.1s]". The bottom right corner indicates "Python".

Dejar un espacio entre los operadores de asignación y las expresiones que los rodeen:



A screenshot of a Python code editor window titled "d:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python\Etiquet...". The tabs show "oop.py" and "Etiqueta.py". The code in "oop.py" is:

```
1 a = 1
```

The status bar at the bottom shows "[Finished in 0.1s]". The bottom right corner indicates "Python".

Puede ser conveniente agregar espacios en blanco para separar visualmente los operadores con más alta prioridad.

En la definición de funciones dejamos un espacio en blanco después de:

```
d:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python\Etiquet...
File Edit Selection Find View Goto Tools Project Preferences Help
oop.py x Etiqueta.py
1 def f(input: str)
2 def f() -> str;

[Finished in 0.1s]
Line 1, Column 8 Tab Size: 4 Python
```

Cuando el signo igual se usa para asignar argumentos en una llamada a función no deberá estar separado por espacios en blanco.

Múltiples sentencias en una misma línea están des-recomendados.

Cuando se termina el renglón con una coma se sugiere rodearla de paréntesis:

```
d:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python\Etiquet...
File Edit Selection Find View Goto Tools Project Preferences Help
oop.py x Etiqueta.py
1 files = ('a.txt',)

[Finished in 0.1s]
Line 1, Column 18 Tab Size: 4 Python
```

Comentarios:

Los comentarios deben actualizarse junto con el código. Un comentario que contradice el código confunde más que ayuda.

Los comentarios deben estar redactados como oraciones completas.

Los comentarios que aplican a bloques de código deben estar precedidos por un renglón de comentario en blanco.

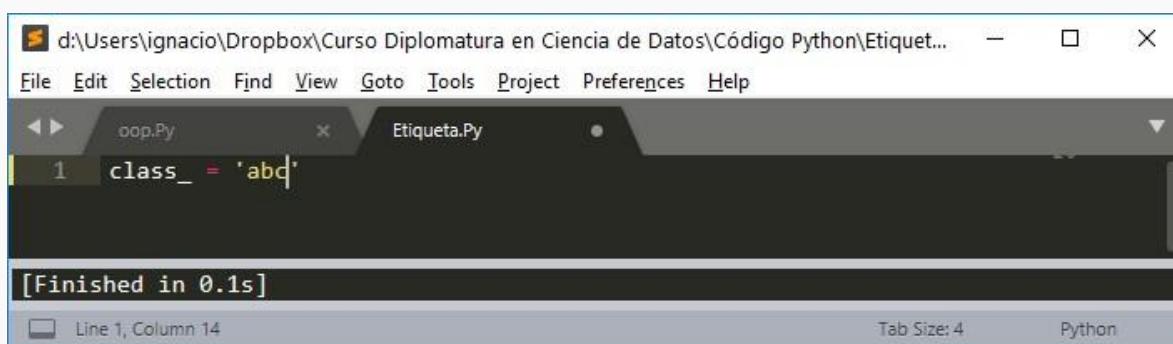
Los comentarios dentro de la propia línea deben separarse por al menos dos espacios en blanco del código que comentan.

Convenciones de nombres:

Hay múltiples convenciones de nombres:

- b (sólo minúsculas en la primer letra)
- B (siempre comenzando con mayúscula)
- Sólo minúsculas
- Sólo minúsculas con _
- Sólo mayúsculas
- Sólo mayúsculas con _
- Al iniciar una palabra utilizar una mayúscula.
- Lo mismo que la anterior pero comenzando en minúscula
- También existe la costumbre de comenzar el nombre de todas las variables por un prefijo que indique la parte del programa a la que pertenecen. No se usa mucho en Python. Por ejemplo la librería X11 le agrega a todos sus métodos públicos una X al principio. Eso se juzga innecesario en Python pues en cada caso está el nombre del objeto antes de la invocación del método.

Cuando queremos usar como nombre de variable una palabra reservada se acostumbra agregarle al final un guión bajo:



The screenshot shows a code editor window titled 'Etiquet...'. The menu bar includes File, Edit, Selection, Find, View, Goto, Tools, Project, Preferences, and Help. There are two tabs open: 'oop.Py' and 'Etiqueta.Py'. The 'Etiqueta.Py' tab contains the following code:
1 class_ = 'abc'
A status bar at the bottom indicates '[Finished in 0.1s]'. Other details include 'Line 1, Column 14', 'Tab Size: 4', and 'Python'.

El uso de un doble guión bajo antes de cada variable se encarga de que quede encapsulada en la clase.

El uso de doble guión bajo antes y después de un nombre sólo debe usarse para los casos documentados como `__init__`, `__import__` o `__file__`

Normas de nomenclatura prescriptivas:

No usar l (se puede confundir con un 1) O (se puede confundir con un 0) ni I como nombres de variables.

Si necesita usar l use L

En algunas fuentes estos caracteres son indistinguibles.

Los identificadores usados deben ser compatibles con ASCII.

The screenshot shows a code editor interface with two tabs: 'oop.py' and 'Etiqueta.py'. The 'oop.py' tab contains the following code:

```
1 # Cosas que andan pero NO son una buena idea.
2 l = 1
3 O = 2
4 I = 3
5 print(l,O,I)
6 ñ = 1
7 á = 2
8 é = 3
9 í = 4
10 ó = 5
11 ú = 6
12 print(ñ,á,é,í,ó,ú)
13
```

The 'Etiqueta.py' tab is currently active. Below the code editor, the terminal output window shows the execution results:

```
1 2 3
1 2 3 4 5 6
[Finished in 0.4s]
```

At the bottom of the editor, status bars indicate 'Line 1, Column 46', 'Tab Size: 4', and 'Python'.

Nombres de módulos y paquetes:

Los nombres de los módulos y paquetes deberían ser cortos y en minúscula.

Diplomatura en Python aplicado a la ciencia de datos

Se pueden usar guiones bajos en los nombres de los módulos pero se desalienta su uso en los nombres de los paquetes.

Nombres de las clases:

Las clases deben nombrarse con sustantivos y deben comenzar en mayúscula.

Nombres de funciones y variables:

Deben ser en minúscula separando las palabras con _ para mejorar la legibilidad.

Siempre usar self por el primer argumento para instanciar métodos.

Si el nombre de un argumento colisiona con una palabra reservada se recomienda usar un guión bajo agregado al final.

Los métodos y las variables de instancia siguen la misma convención que funciones y variables.

Las constantes se definen normalmente a nivel de módulo en mayúsculas y separando las palabras con guión bajo.

Diseñando para la herencia:

Ante la duda de si una propiedad o método debe ser público o privado es mejor empezar haciéndolo privado. Convertir un atributo o un método en público solo requiere tocar en un solo lugar. Convertir una propiedad o método en privado requiere tocar en la definición de la clase y en cada lugar en el que se haya usado la propiedad o método.

- Los métodos y propiedades públicos no deben llevar guiones bajos en el comienzo.
- Si colisionan con una palabra reservada se les agrega un guión bajo al final.
- Para los atributos simples se recomienda usar propiedades y no crear métodos para alterarlos. Este segundo método lo reservamos cuando necesitamos hacer

verificaciones o modificaciones adicionales con cada cambio. Para cosas que sean difíciles o largas de calcular se recomienda usar métodos y no propiedades.

-
- Si la clase que estamos construyendo la estamos pensando como clase que será padre de otras clases en vez de ser instanciada directamente. (Esto también se conoce como clase abstracta, porque no se instanciará nunca) y tiene propiedades que no queremos que sean usados por las clases hijas entonces deberíamos hacerlas empezar por dos guiones bajos.

Recomendaciones para la programación:

Lo veremos más adelante pues se trata de recomendaciones para usar muchas cosas que todavía no hemos visto.

Manejo de archivos y directorios

Manejo de archivos separados por comas:

Ya conseguimos manejar archivos en texto plano.

Con lo que sabemos de expresiones regulares podríamos tratar de identificar las columnas, separarlas, investigar los tipos, convertirlos, pero...

Para que pelarnos la cabeza probando y, lo que es peor, ajustando, nuestro software cuando podemos usar algo mucho más probado y disponible sin pagar: el paquete csv.

Los comandos principales son:

```
open csv.reader  
csv.writer  
csv.DictReader  
csv.DictWriter  
close
```

Empecemos por leer un archivo de prueba que hemos armado especialmente para la ocasión:

Entero	Flotante	Nombre	Fecha
1	1,5	XQMSA	16/10/2019
2	2,5	AWILJ	15/10/2019
3	3,5	RGWAF	14/10/2019
4	4,5	BANUR	13/10/2019
5	5,5	QEUNE	12/10/2019
6	6,5	HCHXL	11/10/2019
7	7,5	CMZTB	10/10/2019
8	8,5	OQOVI	9/10/2019
9	9,5	REYAZ	8/10/2019
10	10,5	CUSFB	7/10/2019

El código Python para leerlo es:

```

d:\Users\ignacio\Dropbox\Curso Diplomatura ...
File Edit Selection Find View Goto Tools Project Preferences Help
oop.py ColectaDirectorio.Py
1 import csv
2
3 f = open("PruebaLectura.csv","r")
4
5 reader = csv.reader(f,delimiter=";")
6
7 for row in reader:
8     print(row)
9
10 f.close()
11
12
['Entero', 'Flotante', 'Nombre', 'Fecha']
['1', '1,5', 'XQMSA', '16/10/2019']
['2', '2,5', 'AWILJ', '15/10/2019']
['3', '3,5', 'RGWAF', '14/10/2019']
['4', '4,5', 'BANUR', '13/10/2019']
['5', '5,5', 'QEUNE', '12/10/2019']
['6', '6,5', 'HCHXL', '11/10/2019']
['7', '7,5', 'CMZTB', '10/10/2019']
['8', '8,5', 'OQOVI', '9/10/2019']
['9', '9,5', 'REYAZ', '8/10/2019']
['10', '10,5', 'CUSFB', '7/10/2019']

Line 2, Column 1 Tab Size: 4 F

```

Cada renglón del archivo es leído en un objeto de la clase `_csv.reader`. Cada renglón es también una colección de los valores que van tomando los campos.

Notamos que:

- Todos los campos los reconoce inicialmente como strings
- Los puntos decimales los refleja como comas. Si necesitamos convertir ese número a flotante vamos a precisar reemplazarlo por un punto.

Vamos ahora a escribir un archivo plano con la misma información:

```

File Edit Selection Find View Goto Tools Project Preferences Help
oop.py ColectaDirectorio.py
1 import csv
2
3 f = open("PruebaLectura.csv","r")
4
5 reader = csv.reader(f,delimiter=";")
6
7 print(type(reader))
8
9 for r in reader:
10     print(type(r))
11     break
12
13
14 f2 = open("PruebaEscritura.csv","w")
15
16 writer = csv.writer(f2,delimiter=";",lineterminator="\n")
17
18 writer.writerows(reader)
19
20 f2.close()
21 f.close()
22
23 print("Escritura Completada")
24

```

```

<class '_csv.reader'>
<class 'list'>
Escritura Completada
[Finished in 0.2s]

```

Line 16, Column 1 Tab Size: 4 Python

Debemos resaltar que para obtener lo que teníamos tuvimos que hacer dos aclaraciones:

- delimiter = ";"
- lineterminator="\n"

Si omitimos el delimiter el Excel nos toma todos los campos en la misma celda.

Si omitimos el lineterminator entonces va dejando un renglón libre intermedio pues el terminador por defecto es \r\n

Otra cosa que notamos es que necesitamos mantener f abierto para que reader contenta los datos. Si no hemos sacado los datos de un archivo necesitamos tenerlos en una estructura

que es una lista de listas. En particular una lista de renglones y cada renglón una lista de campos.

Tratemos de generar y grabar un archivo a partir de una estructura que hayamos generado desde los tipos básicos.

```
d:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código... ━ ━ X
File Edit Selection Find View Goto Tools Project Preferences Help
ColectaDirectorio.py x
oop.py x
1 import csv
2
3 encabezado = ["Entero","Flotante","Texto","Fecha"]
4
5 l = []
6
7 l.append(encabezado)
8
9 for i in range(100):
10     r = []
11     r.append(i)
12     r.append(float(i)+0.5)
13     r.append("abcde")
14     r.append("10/10/2019")
15     l.append(r)
16
17
18 f2 = open("PruebaEscritura2.csv","w")
19
20 writer = csv.writer(f2,delimiter=";",lineterminator="\n")
21
22 writer.writerows(l)
23
24 f2.close

Escrutura Completada
[Finished in 0.3s]

Line 1, Column 1 Tab Size: 4 Python
```

Vamos ahora a utilizar DictReader y DictWriter sobre los mismos conjuntos que hemos estado trabajando:

The screenshot shows a code editor window with a dark theme. At the top, there's a menu bar with options like File, Edit, Selection, Find, View, Goto, Tools, Project, Preferences, and Help. Below the menu, there are two tabs: 'oop.py' and 'ColectaDirectorio.py'. The 'oop.py' tab is active, displaying the following Python code:

```
1 import csv
2
3 f = open("PruebaEscritura2.csv","r")
4
5 reader = csv.DictReader(f, delimiter = ";")
6 print(reader.fieldnames)
7
8 for r in reader:
9     print(r["Entero"],r["Flotante"],r["Texto"],r["Fecha"])
10
11 f.close
12
13
```

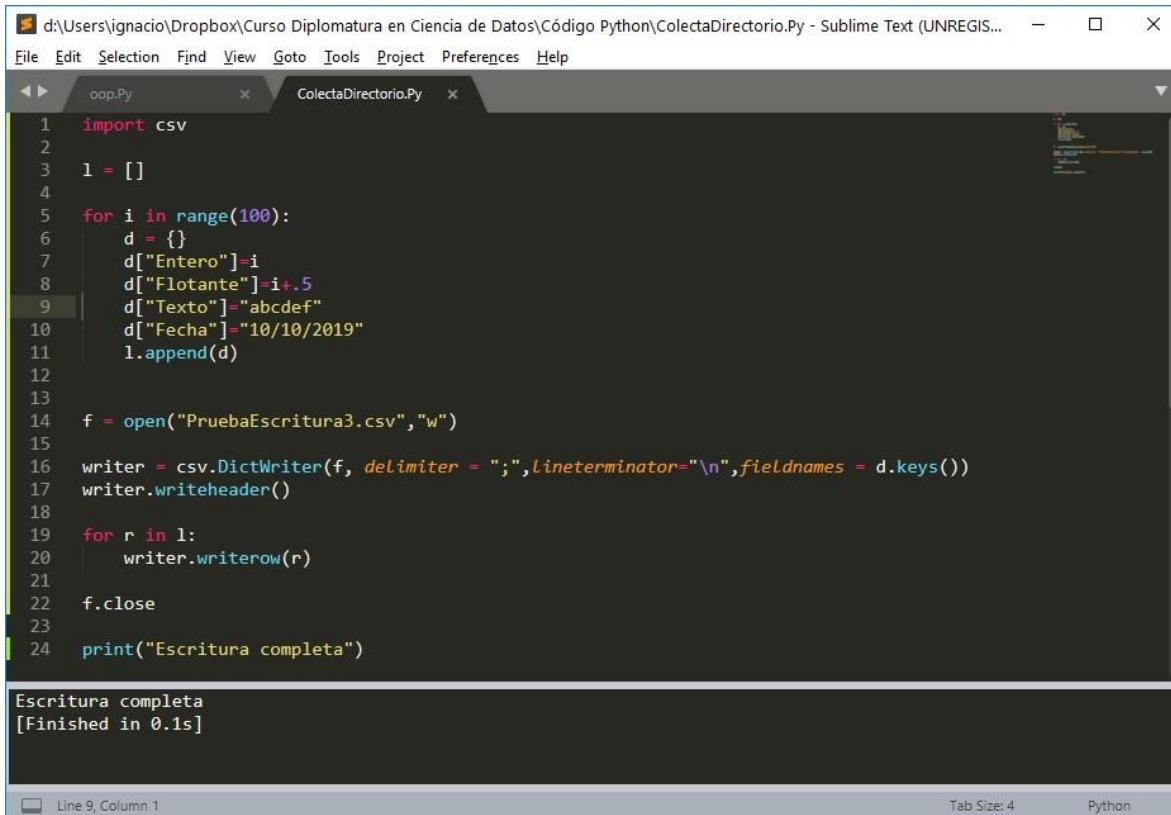
Below the code, the output of the script is shown in a large text area:

```
['Entero', 'Flotante', 'Texto', 'Fecha']
0 0.5 abcde 10/10/2019
1 1.5 abcde 10/10/2019
2 2.5 abcde 10/10/2019
3 3.5 abcde 10/10/2019
4 4.5 abcde 10/10/2019
5 5.5 abcde 10/10/2019
6 6.5 abcde 10/10/2019
7 7.5 abcde 10/10/2019
8 8.5 abcde 10/10/2019
9 9.5 abcde 10/10/2019
10 10.5 abcde 10/10/2019
11 11.5 abcde 10/10/2019
12 12.5 abcde 10/10/2019
13 13.5 abcde 10/10/2019
14 14.5 abcde 10/10/2019
```

At the bottom of the editor window, there are status indicators: 'Line 11, Column 1', 'Tab Size: 4', and 'Python'.

Ya aprendimos a leer directamente en un objeto que podemos recorrer como si fuera una lista de diccionarios. Es lo más parecido al concepto de recordset o de dataframe.

Vamos ahora a aprender a escribir.



```

d:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python\ColectaDirectorio.Py - Sublime Text (UNREGIS...
File Edit Selection Find View Goto Tools Project Preferences Help
oop.py ColectaDirectorio.py
1 import csv
2
3 l = []
4
5 for i in range(100):
6     d = {}
7     d["Entero"] = i
8     d["Flotante"] = i + .5
9     d["Texto"] = "abcdef"
10    d["Fecha"] = "10/10/2019"
11    l.append(d)
12
13
14 f = open("PruebaEscritura3.csv", "w")
15
16 writer = csv.DictWriter(f, delimiter = ";", lineterminator = "\n", fieldnames = d.keys())
17 writer.writeheader()
18
19 for r in l:
20     writer.writerow(r)
21
22 f.close()
23
24 print("Escritura completa")

```

Escritura completa
[Finished in 0.1s]

Line 9, Column 1 Tab Size: 4 Python

Y, por supuesto, el resultado es:

Entero	Flotante	Texto	Fecha
0	0.5	abcdef	10/10/2019
1	1.5	abcdef	10/10/2019
2	2.5	abcdef	10/10/2019
3	3.5	abcdef	10/10/2019
4	4.5	abcdef	10/10/2019
5	5.5	abcdef	10/10/2019
6	6.5	abcdef	10/10/2019
7	7.5	abcdef	10/10/2019
8	8.5	abcdef	10/10/2019
9	9.5	abcdef	10/10/2019

Conexión a Excel

Vamos a empezar por leer archivos de Microsoft Excel desde Python.

Microsoft solo proporciona APIs para sus plataformas de desarrollo. Esto resulta complicado desde la perspectiva de Python que está concebido como una plataforma libre y queda, por lo tanto, sin soporte oficial ante un formato de archivo privativo.

Sin embargo, gracias a la comunidad de desarrollo se ha creado una librería (no oficial) para la manipulación casi total de estos archivos que se llama OpenPyxl.

Lo primero que debemos hacer es instalar la librería openpyxl para que nuestro interprete Python lo encuentre.

```
$pip install openpyxl
```

Una vez instalada la librería podemos comenzar a escribir código, recomiendo descargar el siguiente documento de Excel para seguir el tutorial (puedes usar uno propio, si lo deseas).

El modelo de objetos de Excel:

- Workbook ○ Tupla de worksheets ■ Tupla de filas
 - Tupla de columnas ○ Cada columna tiene un valor

Vamos a practicar con la lectura del archivo de Excel que usé para construir el csv que veníamos trabajando:

The screenshot shows a Python IDE interface with two tabs: 'oop.py' and 'ColectaDirectorio.py'. The 'oop.py' tab contains the following code:

```
1 import openpyxl
2
3 doc = openpyxl.load_workbook('PruebaLecturaExcel.xlsx')
4
5 for s in doc:
6     try:
7         for r in s:
8             try:
9                 for c in r:
10                     print(c.value)
11             except TypeError:
12                 break
13         except TypeError:
14             break
15
16 doc.close()
17 |
```

The 'ColectaDirectorio.py' tab is visible in the background. Below the code editor, the terminal window displays the output of the script, which is the content of the Excel file 'PruebaLecturaExcel.xlsx'. The output is as follows:

```
Entero
Flotante
Nombre
Fecha
1
1.5
HHBDS
2019-10-16 00:00:00
2
2.5
ZACNT
2019-10-15 00:00:00
3
3.5
AZPFC
2019-10-14 00:00:00
4
4.5
PCLRE
2019-10-13 00:00:00
5
```

At the bottom of the terminal window, it says 'Line 17, Column 1'.

Ahora vamos a tratar de guardar los resultados obtenidos en un diccionario:

```

d:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python\Colecta...
File Edit Selection Find View Goto Tools Project Preferences Help
oop.py ColectaDirectorio.py
1 import openpyxl
2
3 doc = openpyxl.load_workbook('PruebaLecturaExcel.xlsx')
4
5 ls = []
6
7 for s in doc:
8     lr = []
9     try:
10         for r in s:
11             lc = []
12             try:
13                 for c in r:
14                     lc.append(c.value)
15             except TypeError:
16                 break
17             lr.append(lc)
18     except TypeError:
19         break
20     ls.append(lr)
21 doc.close()
22
23 ns = 1
24
25 for s in ls:
26     print("Hoja: ", ns)
27     for r in s:
28         print(r)
29     ns = ns+1
30
Hoja: 1
['Entero', 'Flotante', 'Nombre', 'Fecha']
[1, 1.5, 'HHBDS', datetime.datetime(2019, 10, 16, 0, 0)]
[2, 2.5, 'ZACNT', datetime.datetime(2019, 10, 15, 0, 0)]
[3, 3.5, 'AZPFC', datetime.datetime(2019, 10, 14, 0, 0)]
[4, 4.5, 'PCLRE', datetime.datetime(2019, 10, 13, 0, 0)]
[5, 5.5, 'LQSQP', datetime.datetime(2019, 10, 12, 0, 0)]
[6, 6.5, 'RBYLG', datetime.datetime(2019, 10, 11, 0, 0)]
[7, 7.5, 'KXMWY', datetime.datetime(2019, 10, 10, 0, 0)]
[8, 8.5, 'SNQLN', datetime.datetime(2019, 10, 9, 0, 0)]
[9, 9.5, 'ZMMBR', datetime.datetime(2019, 10, 8, 0, 0)]
[10, 10.5, 'XTLSE', datetime.datetime(2019, 10, 7, 0, 0)]
[Finished in 1.3s]

```

Line 31, Column 1 Tab Size: 4 Python

Nuestro siguiente paso es escribir desde cero un archivo Excel. Para eso vamos a seguir los pasos detallados en: <https://donnierock.com/2018/12/09/escribir-un-fichero-excel-desde-python/>

Para instalarlo hacemos:

```
pip install XlsxWriter
```

Partimos con la ventaja de que ya conocemos el modelo de objetos de Excel.

El primer paso es importar la librería y crear el libro y agregarle una hoja:

A screenshot of a Python code editor window titled "oop.py". The code is as follows:

```
d:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python\Escribe... - EscribeUnArchivoXLSXPy

File Edit Selection Find View Goto Tools Project Preferences Help

1 # Programa que genera un archivo xlsx
2
3 import xlsxwriter
4
5 # abro el archivo
6
7 libro = xlsxwriter.Workbook("PruebaEscrituraExcel.xlsx")
8
9 hoja = libro.add_worksheet()
10

[Finished in 0.3s]
```

The status bar at the bottom shows "Line 4, Column 1" and "Tab Size: 4". The language is identified as "Python".

Luego generamos la lista de listas con los datos que queremos guardar:

A screenshot of a Python code editor window titled "oop.py". The code is as follows:

```
d:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python\Escribe... - EscribeUnArchivoXLSXPy

File Edit Selection Find View Goto Tools Project Preferences Help

11 # Genero la lista de listas con la información a grabar
12
13 s = []
14 l = ['Entero', 'Flotante', 'Texto', 'Fecha']
15 s.append(l)
16 for r in range(100):
17     l = []
18     l.append(r)
19     l.append(float(r)+.5)
20     l.append('abc'+str(r))
21     l.append('10/10/2019')
22     s.append(l)
23

[Finished in 0.3s]
```

The status bar at the bottom shows "Line 4, Column 1" and "Tab Size: 4". The language is identified as "Python".

Y, finalmente, nos toca grabar el texto:

The screenshot shows a Python code editor window titled 'oop.py'. The code is a script named 'EscribeUnArchivoXLSX.py' which writes the contents of a list of lists to an Excel file. The code uses the 'openpyxl' library. The terminal output at the bottom of the editor says '[Finished in 0.3s]'. The status bar at the bottom indicates 'Line 4, Column 1', 'Tab Size: 4', and 'Python'.

```
25 # escribimos el contenido de la lista de listas
26
27 row = 0
28 col = 0
29
30 for r in s:
31     for c in r:
32         hoja.write(row,col,c)
33         col = col+1
34
35     row=row+1
36     col=0
37
38 libro.close()
39
```

[Finished in 0.3s]

Line 4, Column 1 Tab Size: 4 Python

Nos queda ahora abrir el archivo y ver si tuvimos éxito:

The screenshot shows a Microsoft Excel spreadsheet titled "PruebaEscrituraExcel - Micros...". The ribbon menu is visible at the top, showing tabs like Inicio, Insertar, Diseño de página, Fórmulas, Datos, Revisar, and Vista. The "Fuente" (Font) section of the ribbon is selected, showing Calibri 11pt, bold, italic, and underline options. The main area displays a table with 12 rows and 5 columns. Column A is labeled "A" and contains the number 1. Column B is labeled "B" and contains the text "Flotante". Column C is labeled "C" and contains the text "Texto". Column D is labeled "D" and contains the text "Fecha". Column E is labeled "E". The first row has a yellow background, and the second row has a light blue background. The data from row 2 to row 12 follows a pattern where column B contains a value from 0 to 10, column C contains a value from 0,5 to 10,5, column D contains the text "abc" followed by a number from 0 to 10, and column E contains the date "10/10/2019". The status bar at the bottom shows "Listo" and a zoom level of 140%.

	A	B	C	D	E
1	Entero	Flotante	Texto	Fecha	
2	0	0,5	abc0	10/10/2019	
3	1	1,5	abc1	10/10/2019	
4	2	2,5	abc2	10/10/2019	
5	3	3,5	abc3	10/10/2019	
6	4	4,5	abc4	10/10/2019	
7	5	5,5	abc5	10/10/2019	
8	6	6,5	abc6	10/10/2019	
9	7	7,5	abc7	10/10/2019	
10	8	8,5	abc8	10/10/2019	
11	9	9,5	abc9	10/10/2019	
12	10	10,5	abc10	10/10/2019	

Lo hemos conseguido!

Normas sugeridas para generar código Python:

Una guía de estilo tiene que ver con la consistencia. La consistencia es importante entre distintos proyectos desarrollados en una organización, más importante dentro de cada proyecto, más importante aún dentro de cada módulo y terriblemente importante dentro de cada función.

En cualquier caso no conviene romper la consistencia hacia atrás de un desarrollo sólo para cumplir con estas recomendaciones. Por ese motivo, si se está por comenzar a desarrollar es particularmente oportuno fijar las normas que proveerán consistencia a futuro.

Distribución del código en la hoja:

Indentación:

Usar 4 espacios para la indentación.

Cuando necesitamos continuar una línea larga en la siguiente debemos hacer que quede claro mediante la indentación cual es la situación:

The screenshot shows a code editor window with two tabs: 'oop.py' and 'Etiqueta.py'. The 'oop.py' tab contains the following code:

```
1 f = nombre_de_funcion_muy_muy_largo(variable_1, variable_2,
2                                         variable_3, variable_4)
3
4 f = nombre_de_funcion_muy_muy_largo(
5     variable_1, variable_2,
6     variable_3, variable_4)
```

The 'Etiqueta.py' tab is currently active. Below the tabs, there is a status bar with the message '[Finished in 0.3s]'. At the bottom of the editor, it says 'Line 8, Column 1' and has tabs for 'Python' and 'Tab Size: 4'.

Lo que no deberíamos hacer es poner argumentos en la línea que vamos a interrumpir si no vamos a usar la alineación vertical.

```
foo = long_function_name(var_one, var_two,
var_three, var_four)
```

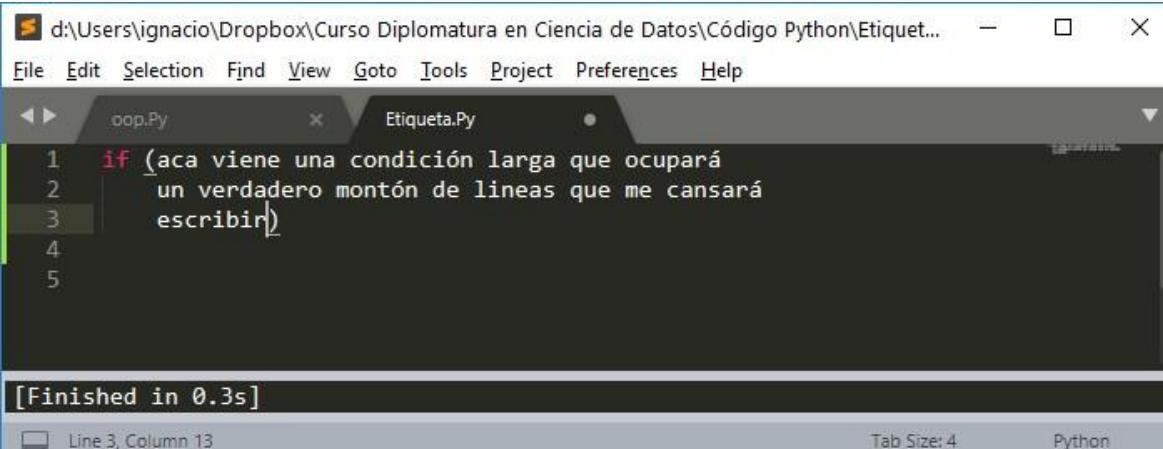
Tampoco vale usar una indentación que parezca normal y por lo tanto confunda:

```
def long_function_name(
```

```
var_one, var_two, var_three,  
  
var_four):  
  
    print(var_one)
```

Por lo tanto la regla de los cuatro espacios es opcional para las líneas de continuación y podría aportar más claridad otro criterio.

En el caso de que la condición de un if sea muy larga conviene aprovechar una circunstancia afortunada: el if más un espacio más un paréntesis genera 4 espacios. Usamos esos cuatro espacios para alinear todas las líneas de continuación que lleve la condición:



The screenshot shows a Python code editor window. The title bar reads "d:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python\Etiquet...". The menu bar includes File, Edit, Selection, Find, View, Goto, Tools, Project, Preferences, and Help. The main editor area has tabs for "oop.Py" and "Etiqueta.Py". The code in "Etiqueta.Py" is:

```
1 if (aca viene una condición larga que ocupará  
2     un verdadero montón de lineas que me cansará  
3     escribir)  
4  
5
```

The status bar at the bottom shows "[Finished in 0.3s]", "Line 3, Column 13", "Tab Size: 4", and "Python".

Un comentario alentador, sublime texto lo organizó de manera automática!

Después de la condición no hay que usar indentación adicional:

The screenshot shows a Python code editor window. The title bar says "d:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python\Etiqueta.py". The menu bar includes File, Edit, Selection, Find, View, Goto, Tools, Project, Preferences, and Help. There are two tabs open: "oop.py" and "Etiqueta.py". The "Etiqueta.py" tab contains the following code:

```
1 if (aca viene una condición larga que ocupará
2     un verdadero montón de líneas que me cansará
3     escribir):
4
5     Hacer.Algo()
6
7
```

The status bar at the bottom shows "[Finished in 0.3s]", "Line 5, Column 17", "Tab Size: 4", and "Python".

El paréntesis, corchete o llave que se cierra puede estar alineado con la última línea o bien con el nombre del objeto:

The screenshot shows a Python code editor window with the same file structure as the previous one. The "Etiqueta.py" tab contains two examples of list assignment:

```
1 mi_lista = [1, 2, 3,
2     4, 5, 6
3 ]
4
5 mi_lista = [1, 2, 3,
6     4, 5, 6
7 ]
```

The status bar at the bottom shows "[Finished in 0.3s]", "Line 7, Column 1", "Tab Size: 4", and "Python".

Tabuladores versus espacios:

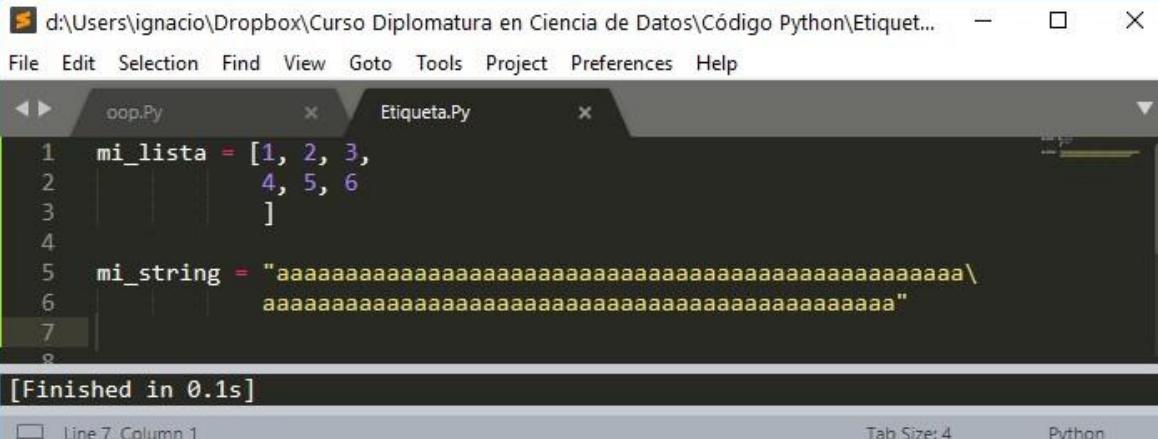
Se recomienda el uso de espacios en blanco.

Máximo largo de línea:

Se sugiere reducir el ancho a 72 caracteres.

Para continuar en la siguiente línea se sugiere usar la continuación implícita de Python que se da cada vez que queda abierto un paréntesis, corchete o llave.

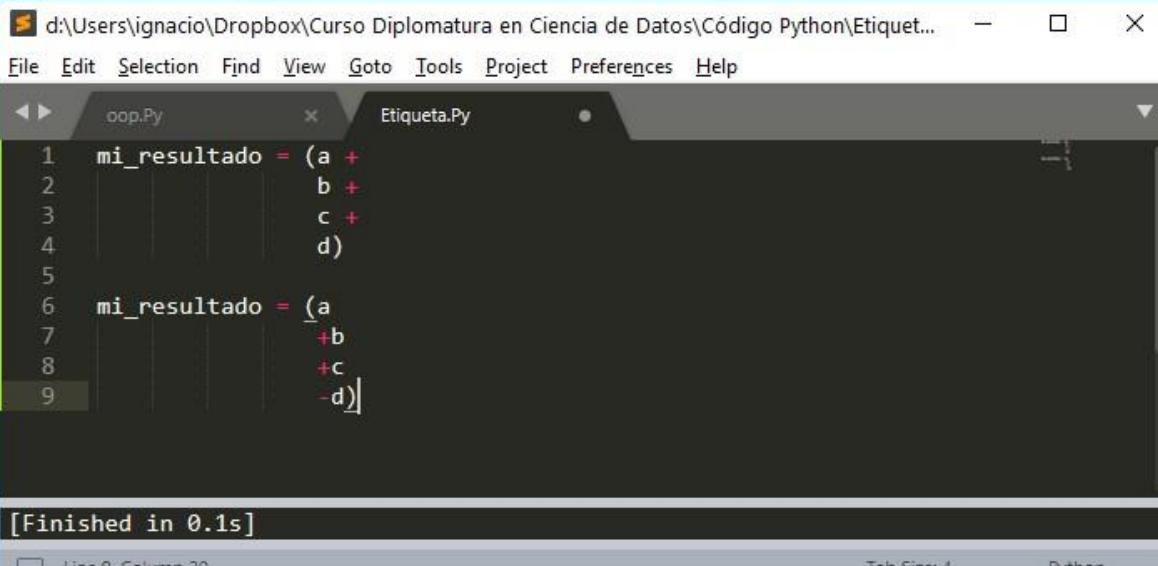
Como segunda alternativa menos preferible es usar la continuación explícita que se consigue poniendo una barra invertida como último carácter de la línea que se quiere continuar abajo.



```
oop.py
1 mi_lista = [1, 2, 3,
2     4, 5, 6
3 ]
4
5 mi_string = "aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa\
6             aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa"
7
8
[Finished in 0.1s]
```

Line 7, Column 1 Tab Size: 4 Python

Cuando en una formula vamos a continuar en el siguiente renglón no se aceptan las siguientes dos versiones con tal de que sea localmente consistente:



```
oop.py
1 mi_resultado = (a +
2     b +
3     c +
4     d)
5
6 mi_resultado = (a
7     +b
8     +c
9     -d)]
```

[Finished in 0.1s]

Line 9, Column 20 Tab Size: 4 Python

Para grupos que empiezan se sugiere usar la segunda opción.

Líneas en blanco:

Se recomienda rodear a las definiciones de alto nivel y a las clases con dos líneas en blanco.

Para las definiciones de métodos dentro de una clase se sugiere una línea en blanco.

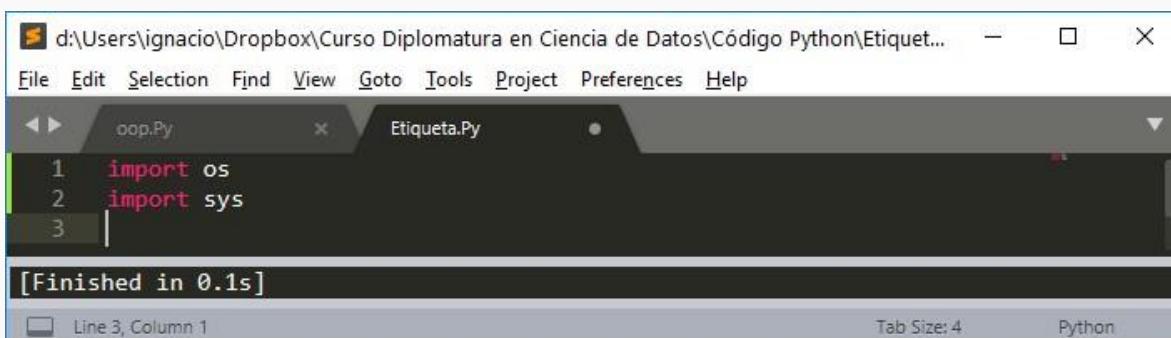
Para indicar divisiones lógicas de alto nivel dentro del código se pueden usar líneas adicionales.

Codificación de los archivos fuentes:

Se debe usar UTF-8

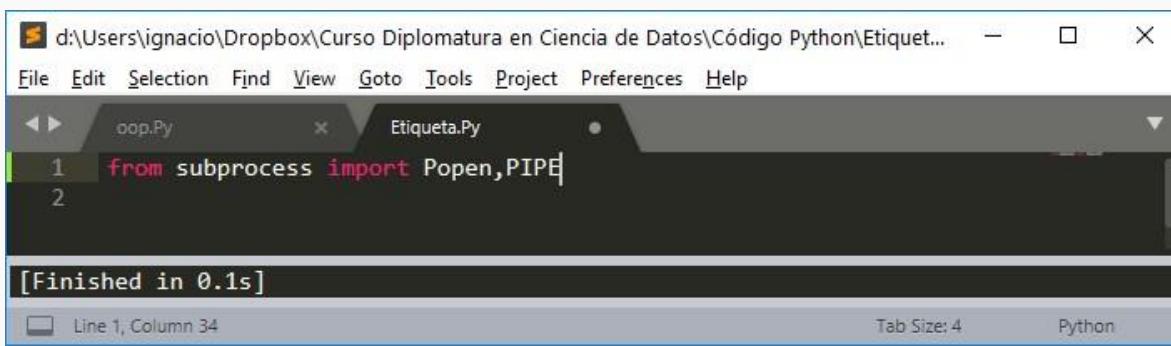
Importación de paquetes:

Los paquetes deben importarse en líneas separadas, al principio del archivo, sólo después de los comentarios del módulo y antes de las constantes y globales del módulo:



```
File Edit Selection Find View Goto Tools Project Preferences Help  
oop.py Etiqueta.py  
1 import os  
2 import sys  
3  
[Finished in 0.1s]  
Line 3, Column 1 Tab Size: 4 Python
```

Sin embargo se considera aceptable usar:



```
File Edit Selection Find View Goto Tools Project Preferences Help  
oop.py Etiqueta.py  
1 from subprocess import Popen, PIPE  
2  
[Finished in 0.1s]  
Line 1, Column 34 Tab Size: 4 Python
```

Los imports se deben separar en tres grupos:

4. Librerías estándar

Cada uno de estos grupos debe estar separado de los otros por una línea en blanco.

Las importaciones absolutas son preferibles.

Solo es aceptable utilizar importaciones relativas cuando estamos en un entorno complejo y nos llevaría a perder legibilidad.

Se debe evitar el uso de * (comodines) en las sentencias import.

Comillas:

En Python las comillas dobles “ y las simples ‘ son equivalentes.

Se sugiere utilizar sólo un tipo de comillas.

Espacios en blanco:

Se sugiere evitar espacios en blanco en sitios extraños:

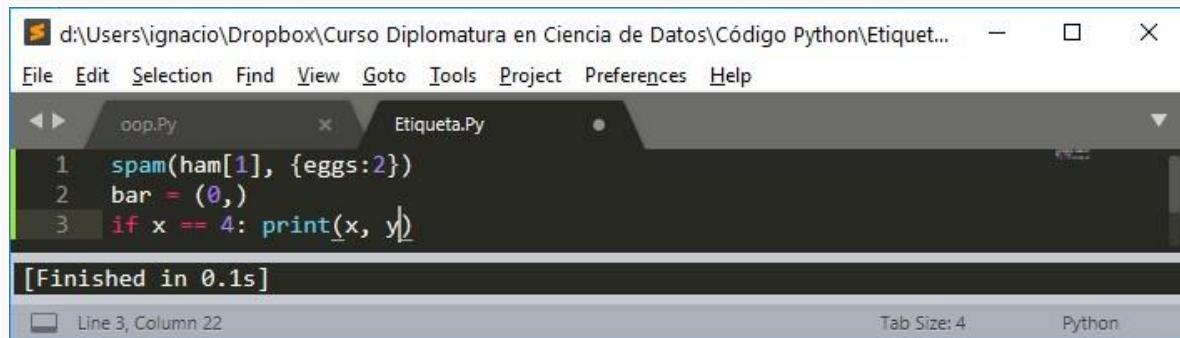
5. Librerías de terceras partes
6. Librerías locales

```
spam( ham[ 1 ], { eggs: 2 } )
```

```
No: bar = (0, )
```

```
No: if x == 4 : print x , y ; x , y = y , x
```

La forma correcta sería:



A screenshot of a code editor window titled "Etiqueta.Py". The code in the editor is:

```
1 spam(ham[1], {eggs:2})
2 bar = (0,)
3 if x == 4: print(x, y)
```

The status bar at the bottom shows "[Finished in 0.1s]". Other details include "Line 3, Column 22", "Tab Size: 4", and "Python".

Tampoco se pueden dejar espacios en blanco antes de un paréntesis, corchete o llave:

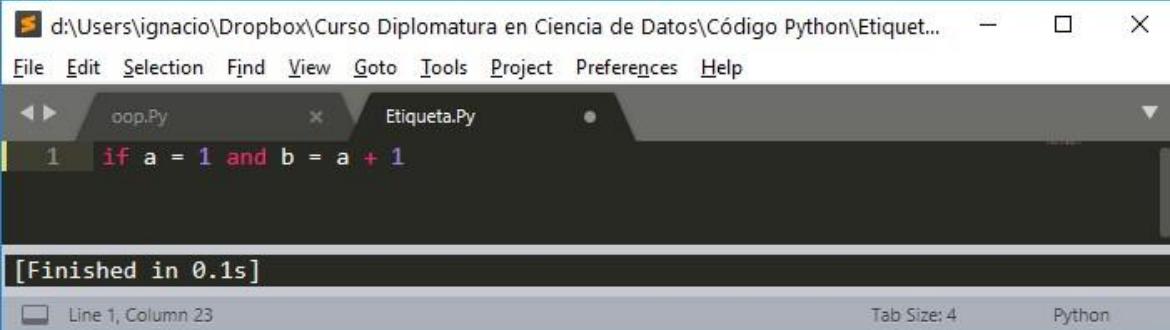
```
spam (1) dct ['key']
= lst [index]
```

Ni tampoco más de un espacio en blanco consecutivo.

```
x      =
1 y
= 2
```

Está prohibido dejar espacios en blanco al final de las líneas pues son invisibles.

Siempre rodear es espacios en blanco los operadores binarios.

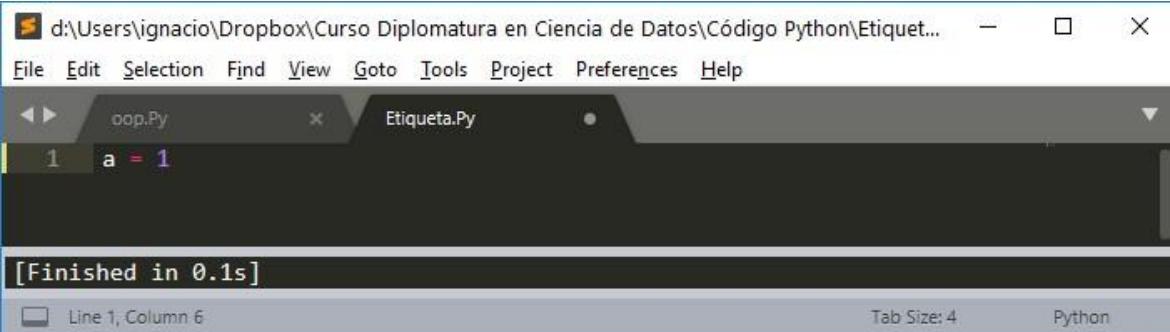


A screenshot of a Python code editor window titled "Etiqueta.Py". The code in the editor is:

```
1 if a = 1 and b = a + 1
```

The status bar at the bottom shows "[Finished in 0.1s]". The bottom right corner indicates "Python".

Dejar un espacio entre los operadores de asignación y las expresiones que los rodeen:



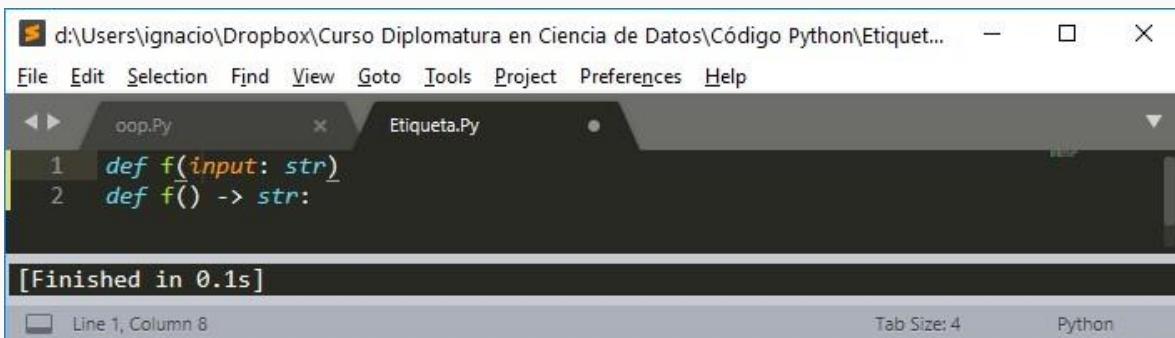
A screenshot of a Python code editor window titled "Etiqueta.Py". The code in the editor is:

```
1 a = 1
```

The status bar at the bottom shows "[Finished in 0.1s]". The bottom right corner indicates "Python".

Puede ser conveniente agregar espacios en blanco para separar visualmente los operadores con más alta prioridad.

En la definición de funciones dejamos un espacio en blanco después de:

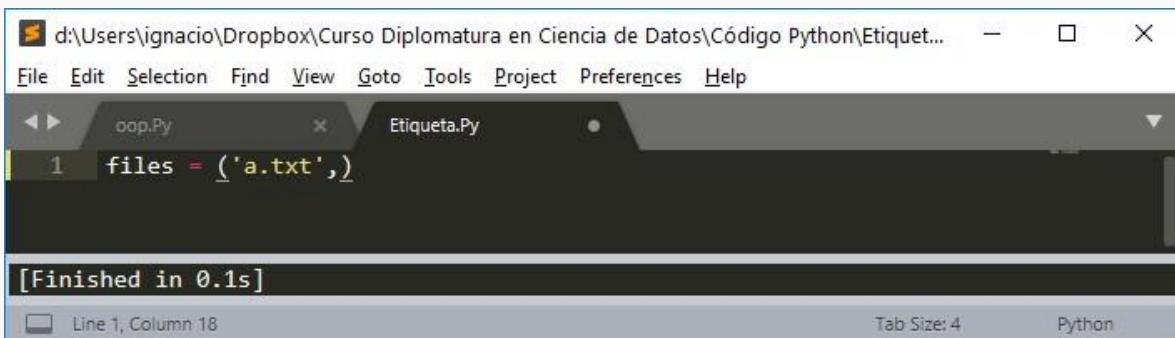


```
def f(input: str)
def f() -> str:
```

Cuando el signo igual se usa para asignar argumentos en una llamada a función no deberá estar separado por espacios en blanco.

Múltiples sentencias en una misma línea están des-recomendados.

Cuando se termina el renglón con una coma se sugiere rodearla de paréntesis:



```
files = ('a.txt',)
```

Comentarios:

Los comentarios deben actualizarse junto con el código. Un comentario que contradice el código confunde más que ayuda.

Los comentarios deben estar redactados como oraciones completas.

Los comentarios que aplican a bloques de código deben estar precedidos por un renglón de comentario en blanco.

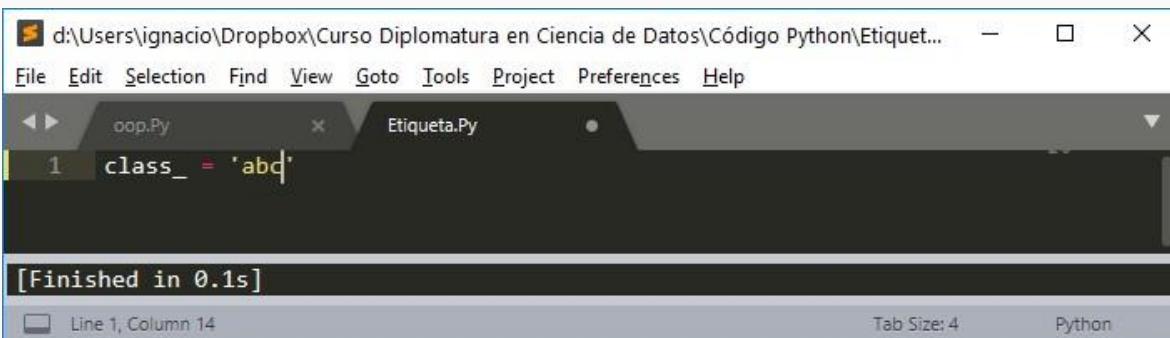
Los comentarios dentro de la propia línea deben separarse por al menos dos espacios en blanco del código que comentan.

Convenciones de nombres:

Hay múltiples convenciones de nombres:

- b (sólo minúsculas en la primer letra)
- B (siempre comenzando con mayúscula)
- Sólo minúsculas
- Sólo minúsculas con _
- Sólo mayúsculas
- Sólo mayúsculas con _
- Al iniciar una palabra utilizar una mayúscula.
- Lo mismo que la anterior pero comenzando en minúscula
- También existe la costumbre de comenzar el nombre de todas las variables por un prefijo que indique la parte del programa a la que pertenecen. No se usa mucho en Python. Por ejemplo la librería X11 le agrega a todos sus métodos públicos una X al principio. Eso se juzga innecesario en Python pues en cada caso está el nombre del objeto antes de la invocación del método.

Cuando queremos usar como nombre de variable una palabra reservada se acostumbra agregarle al final un guión bajo:



```
d:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python\Etiquet...
File Edit Selection Find View Goto Tools Project Preferences Help
oop.Py Etiqueta.Py
1 class_ = 'abc'

[Finished in 0.1s]
Line 1, Column 14 Tab Size: 4 Python
```

El uso de un doble guión bajo antes de cada variable se encarga de que quede encapsulada en la clase.

El uso de doble guión bajo antes y después de un nombre sólo debe usarse para los casos documentados como `__init__`, `__import__` o `__file__`

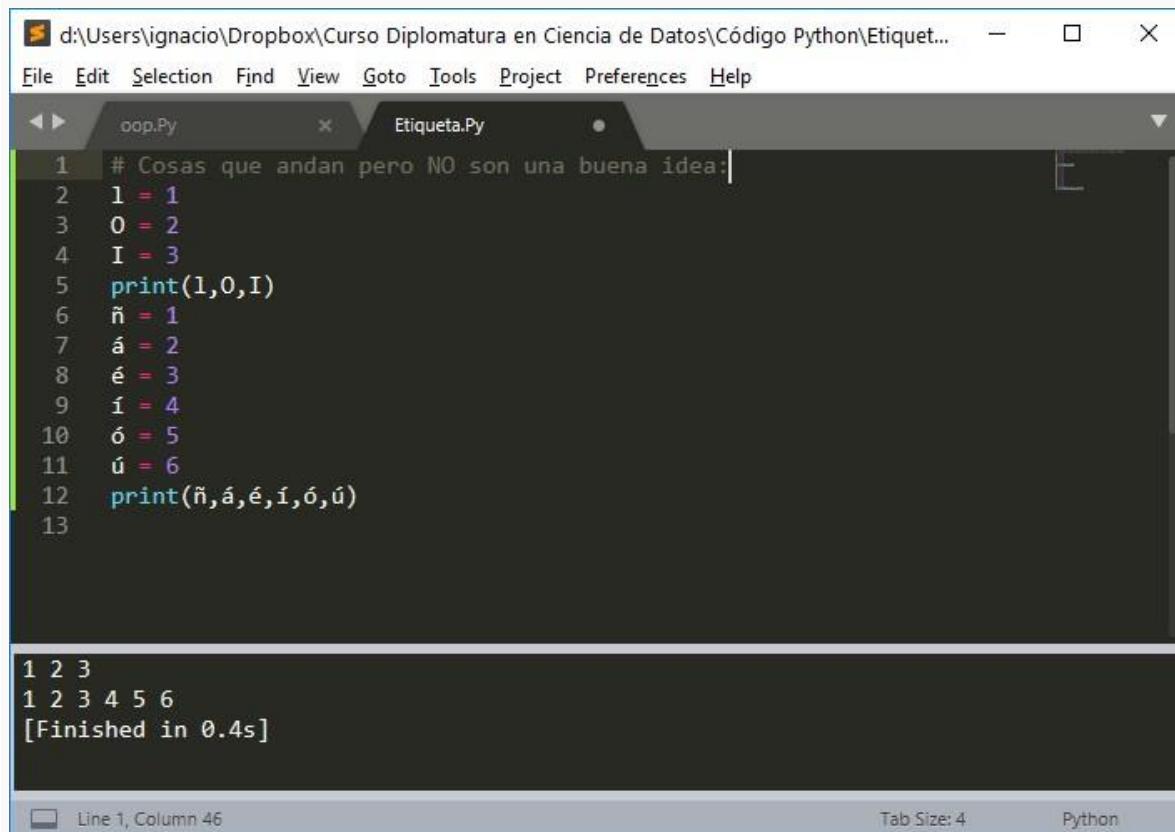
Normas de nomenclatura prescriptivas:

No usar l (se puede confundir con un 1) O (se puede confundir con un 0) ni l como nombres de variables.

Si necesita usar l use L

En algunas fuentes estos caracteres son indistinguibles.

Los identificadores usados deben ser compatibles con ASCII.



```
# Cosas que andan pero NO son una buena idea:  
1 = 1  
O = 2  
I = 3  
print(1,O,I)  
ñ = 1  
á = 2  
é = 3  
í = 4  
ó = 5  
ú = 6  
print(ñ,á,é,í,ó,ú)
```

```
1 2 3  
1 2 3 4 5 6  
[Finished in 0.4s]
```

Line 1, Column 46 Tab Size: 4 Python

Nombres de módulos y paquetes:

Los nombres de los módulos y paquetes deberían ser cortos y en minúscula.

Diplomatura en Python aplicado a la ciencia de datos

Se pueden usar guiones bajos en los nombres de los módulos pero se desalienta su uso en los nombres de los paquetes.

Nombres de las clases:

Las clases deben nombrarse con sustantivos y deben comenzar en mayúscula.

Nombres de funciones y variables:

Deben ser en minúscula separando las palabras con _ para mejorar la legibilidad.

Siempre usar self por el primer argumento para instanciar métodos.

Si el nombre de un argumento colisiona con una palabra reservada se recomienda usar un guión bajo agregado al final.

Los métodos y las variables de instancia siguen la misma convención que funciones y variables.

Las constantes se definen normalmente a nivel de módulo en mayúsculas y separando las palabras con guión bajo.

Diseñando para la herencia:

Ante la duda de si una propiedad o método debe ser público o privado es mejor empezar haciéndolo privado. Convertir un atributo o un método en público solo requiere tocar en un solo lugar. Convertir una propiedad o método en privado requiere tocar en la definición de la clase y en cada lugar en el que se haya usado la propiedad o método.

- Los métodos y propiedades públicos no deben llevar guiones bajos en el comienzo.
- Si colisionan con una palabra reservada se les agrega un guión bajo al final.
- Para los atributos simples se recomienda usar propiedades y no crear métodos para alterarlos. Este segundo método lo reservamos cuando necesitamos hacer

verificaciones o modificaciones adicionales con cada cambio. Para cosas que sean difíciles o largas de calcular se recomienda usar métodos y no propiedades.

-
- Si la clase que estamos construyendo la estamos pensando como clase que será padre de otras clases en vez de ser instanciada directamente. (Esto también se conoce como clase abstracta, porque no se instanciará nunca) y tiene propiedades que no queremos que sean usados por las clases hijas entonces deberíamos hacerlas empezar por dos guiones bajos.

Recomendaciones para la programación:

Lo veremos más adelante pues se trata de recomendaciones para usar muchas cosas que todavía no hemos visto.

Conexión a Bases de Datos

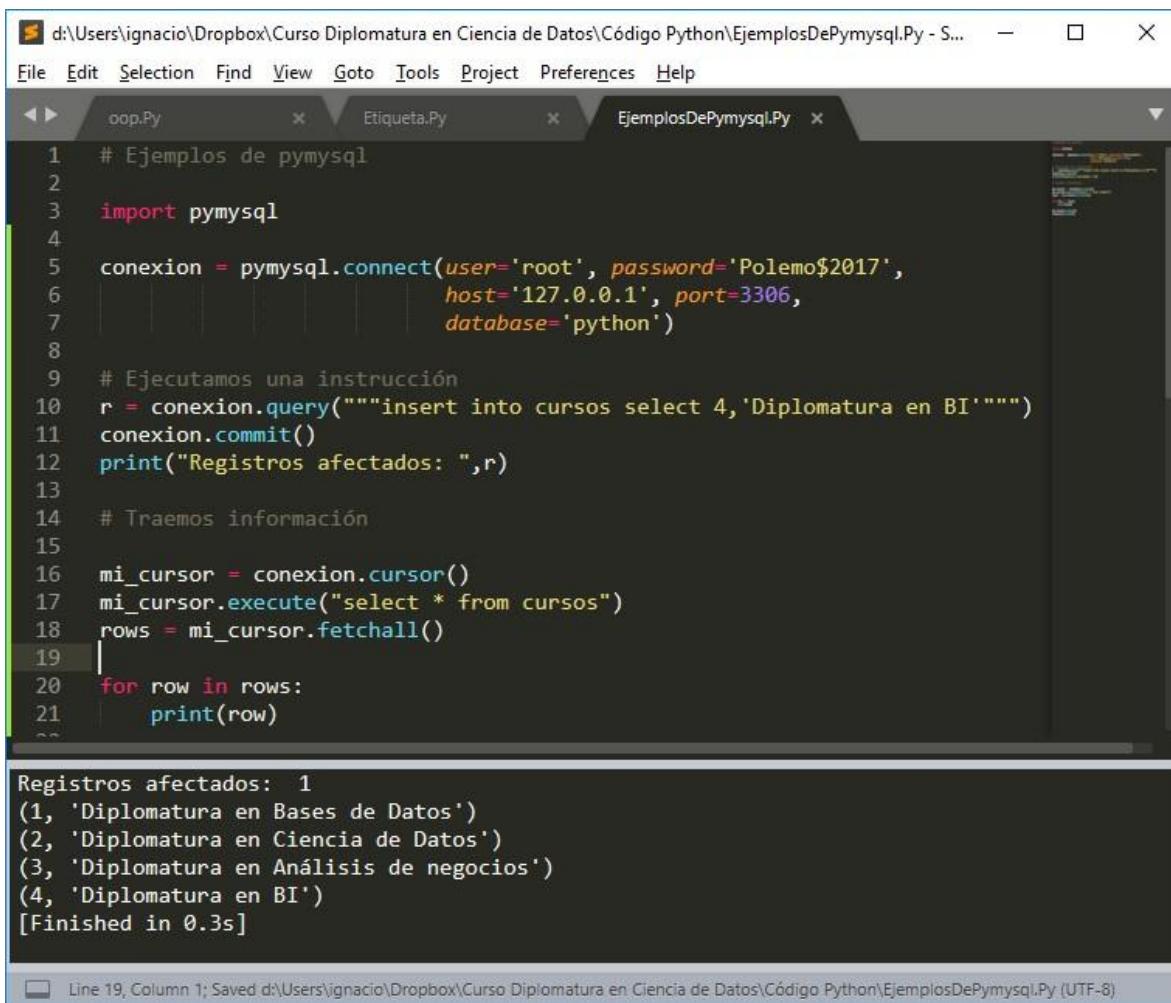
Instalación de pymysql

Para instalar pymysql usamos desde una ventana de comandos:

```
pip install pymysql  
pip install cryptography
```

Utilización de pymysql

Comencemos por el siguiente ejemplo:



```
File Edit Selection Find View Goto Tools Project Preferences Help  
oop.Py x Etiqueta.Py x EjemplosDePymysql.Py x  
1 # Ejemplos de pymysql  
2  
3 import pymysql  
4  
5 conexion = pymysql.connect(user='root', password='Polemo$2017',  
6                             host='127.0.0.1', port=3306,  
7                             database='python')  
8  
9 # Ejecutamos una instrucción  
10 r = conexion.query("""insert into cursos select 4,'Diplomatura en BI'""")  
11 conexion.commit()  
12 print("Registros afectados: ",r)  
13  
14 # Traemos información  
15  
16 mi_cursor = conexion.cursor()  
17 mi_cursor.execute("select * from cursos")  
18 rows = mi_cursor.fetchall()  
19  
20 for row in rows:  
21     print(row)  
  
Registros afectados: 1  
(1, 'Diplomatura en Bases de Datos')  
(2, 'Diplomatura en Ciencia de Datos')  
(3, 'Diplomatura en Análisis de negocios')  
(4, 'Diplomatura en BI')  
[Finished in 0.3s]  
Line 19, Column 1; Saved d:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python\EjemplosDePymysql.py (UTF-8)
```

Vamos ahora paso por paso:

Empezamos por cargar la librería pymysql con el import

Luego creamos el objeto conexión que nos permitirá acceder con los privilegios del usuario a la base de datos que se especifica con su URL y puerto.

El método query nos permite ejecutar una consulta pero no devolver una respuesta.

La variable que nos devuelve query es el número de registros que se verán afectados.

Para que los cambios generados por query queden permanentes necesitamos aplicar el método query.commit

Cuando queremos traernos información recurrimos a los cursores:

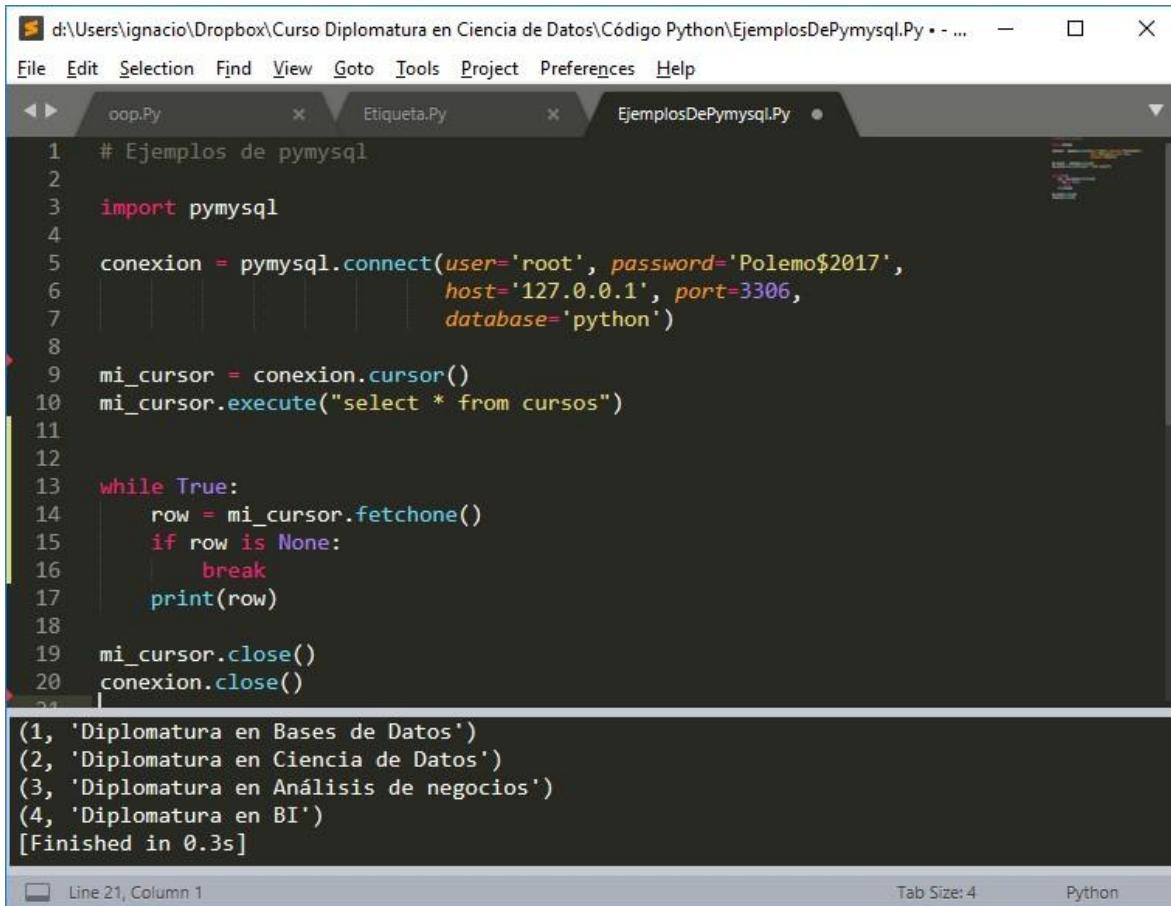
El primer paso es declarar el cursor con cursor()

Luego le indicamos con execute cual es la consulta que nos devolverá los datos que vamos a recorrer.

Para obtener los resultados tenemos dos formas de trabajar:

- Para traernos los resultados de a uno usamos fetchone()
- Para traernos los resultados de a varios usamos fetchmany(n)
- Para traernos todos a memoria y luego recorrerlos dentro de Python usamos fechall()

Fecthall() ya lo usamos en el primer ejemplo. Nos toca ahora usar fetchone()



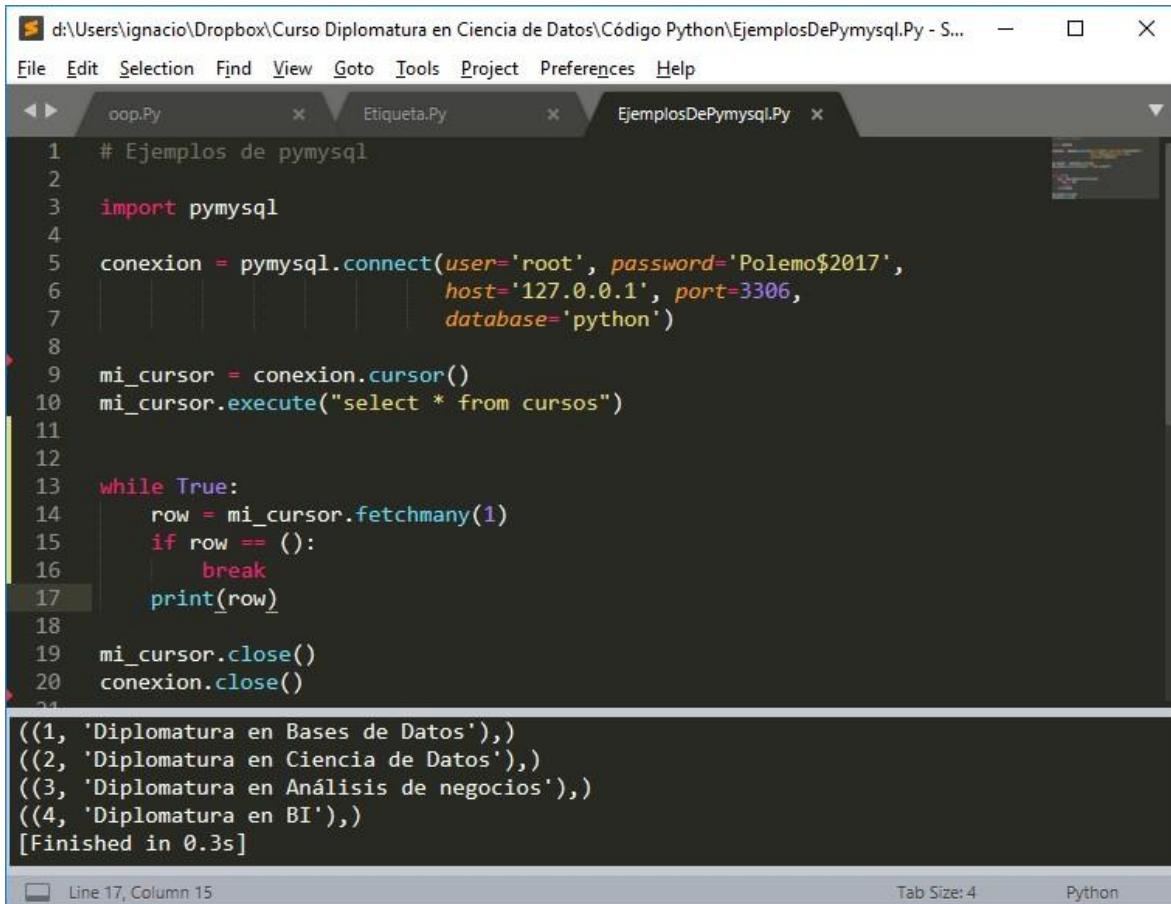
```
d:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python\EjemplosDePymysql.Py - ... ━ ━ ━ X
File Edit Selection Find View Goto Tools Project Preferences Help
◀ ▶ oop.Py × Etiqueta.Py × EjemplosDePymysql.Py ●
```

```
1 # Ejemplos de pymysql
2
3 import pymysql
4
5 conexion = pymysql.connect(user='root', password='Polemo$2017',
6                             host='127.0.0.1', port=3306,
7                             database='python')
8
9 mi_cursor = conexion.cursor()
10 mi_cursor.execute("select * from cursos")
11
12
13 while True:
14     row = mi_cursor.fetchone()
15     if row is None:
16         break
17     print(row)
18
19 mi_cursor.close()
20 conexion.close()
```

```
(1, 'Diplomatura en Bases de Datos')
(2, 'Diplomatura en Ciencia de Datos')
(3, 'Diplomatura en Análisis de negocios')
(4, 'Diplomatura en BI')
[Finished in 0.3s]
```

Line 21, Column 1 Tab Size: 4 Python

Y, por supuesto, nos debemos un ejemplo para fetchmany:



```
d:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python\EjemplosDePymysql.Py - S... — □ X
File Edit Selection Find View Goto Tools Project Preferences Help
◀ ▶ oop.py × Etiqueta.py × EjemplosDePymysql.py ×
1 # Ejemplos de pymysql
2
3 import pymysql
4
5 conexion = pymysql.connect(user='root', password='Polemo$2017',
6                             host='127.0.0.1', port=3306,
7                             database='python')
8
9 mi_cursor = conexion.cursor()
10 mi_cursor.execute("select * from cursos")
11
12
13 while True:
14     row = mi_cursor.fetchmany(1)
15     if row == ():
16         break
17     print(row)
18
19 mi_cursor.close()
20 conexion.close()
21
((1, 'Diplomatura en Bases de Datos'),)
((2, 'Diplomatura en Ciencia de Datos'),)
((3, 'Diplomatura en Análisis de negocios'),)
((4, 'Diplomatura en BI'),)
[Finished in 0.3s]
```

Line 17, Column 15 Tab Size: 4 Python

Notamos que con `fetchmany` no podemos contar con un `None` como respuesta si se acaban los datos.

Y ahora los recorremos de a dos:

```

d:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python\EjemplosDePymysql.Py - S...
File Edit Selection Find View Goto Tools Project Preferences Help
◀ ▶ oop.Py × Etiqueta.Py × EjemplosDePymysql.Py ×
2
3 import pymysql
4
5 conexion = pymysql.connect(user='root', password='Polemo$2017',
6                             host='127.0.0.1', port=3306,
7                             database='python')
8
9 mi_cursor = conexion.cursor()
10 mi_cursor.execute("select * from cursos")
11
12
13 while True:
14     rows = mi_cursor.fetchmany(2)
15     if rows == () or rows is None:
16         break
17     print("Avanzo página")
18     for row in rows:
19         print(row)
20
21 mi_cursor.close()
22 conexion.close()
23
Avanzo página
(1, 'Diplomatura en Bases de Datos')
(2, 'Diplomatura en Ciencia de Datos')
Avanzo página
(3, 'Diplomatura en Análisis de negocios')
(4, 'Diplomatura en BI')
[Finished in 0.3s]

```

Line 18, Column 21 Tab Size: 4 Python

Adaptación de impedancias:

Hemos visto que gracias a la herencia guardamos todos los datos de un objeto concreto en diferentes niveles de objetos padres, abuelos, etc.

En el mundo de las bases de datos relacionales esto no se refleja con facilidad ya que cada objeto queda partido en una multiplicidad de registros de tablas separadas.

Este problema se conoce como “Adaptación de impedancia”

Vamos a abordarlo intentando que:

- No existan tablas para las clases abstractas
- Que la información de cómo se lee y graba cada objeto quede en un solo lugar para facilitar el mantenimiento.
- Podría ser una buena práctica que del lado de la base de datos se utilice un procedimiento almacenado.
- Que sólo un método utilice el procedimiento para grabar

- Que sólo un método utilice el procedimiento que me devuelve un objeto.

f-Strings: una forma mejorada de formatear strings

Vamos a cubrir:

- Técnicas tradicionales de formateo de string:
 - Uso de % ◦ Uso de str.format()
- f-Strings ◦ Sintaxis básica ◦ Expresiones arbitrarias ◦ f-Strings en múltiples líneas ◦ Velocidad
- Particularidades de f-strings en Python
 - Comillas ◦ Diccionarios ◦ Llaves ◦ Barras invertidas ◦ Comentarios dentro de la línea

Desde la versión 3.6 de Python contamos con una forma más simple, clara y rápida de formatear strings: f-strings.

Empecemos por ver como eran las cosas antes:

Técnicas tradicionales:

Antes de Python 3.6 teníamos dos caminos para formatear strings el uso de % y la función str.format()

Repasemos sus usos y limitaciones:

Formateo con %

En primer lugar una nota de avvertencia. El uso de % puede llevar a casos que generan errores en tiempo de ejecución.

El operador % sirve para insertar los datos de una variable string dentro de una cadena de caracteres:

```

d:\Users\ignacio\Dropbox\Curso Diplomatura en Cienci...
File Edit Selection Find View Goto Tools Project Preferences Help
oop.py Etiqueta.py PyMySQLLeerCursos.py
1 # prueba de %
2
3 a = "mundo"
4 b = "Adiós %s cruel!"
5 c = b % a
6 print(c)
7
Adiós mundo cruel!
[Finished in 0.2s]

```

Line 7, Column 1 Tab Size: 4 Python

Guardamos en a los datos que queremos insertar.

Guardamos en b los datos dentro de los cuales vamos a hacer la inserción. El lugar en el cual vamos a hacer la inserción está marcado con %s

Luego guardamos en c el resultado de b % a

Si queremos insertar de una vez varias variables debemos marcar cada lugar donde se producirá una inserción con %s y en vez de una variable indicar una tupla con las variables:

```

d:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python\prueba_de_porcentaje...
File Edit Selection Find View Goto Tools Project Preferences Help
oop.py Etiqueta.py PyMySQLLeerCursos.py prueba_de_porcentaje.py
1 # prueba de %
2
3 a1 = "Buenos Aires"
4 a2 = "Mar del Plata"
5 a3 = "Pinamar"
6 a4 = "Miramar"
7 a5 = "Mar de Ajó"
8 b = "Los que viven en %s suelen veranear en %s, %s, %s o %s."
9 c = b % (a1,a2,a3,a4,a5)
10 print(c)
11
Los que viven en Buenos Aires suelen veranear en Mar del Plata, Pinamar, Miramar o Mar de Ajó.
[Finished in 0.1s]

```

Line 7, Column 18 Tab Size: 4 Python

Limitaciones de %

En cuanto el string crece en tamaño y complejidad el código se vuelve poco legible.

Por ese motivo se inventó: str.format()

Se introdujo en Python 2.6 para mejorar el uso de %:

A screenshot of a Python code editor window titled "prueba_de_porcentaje.py". The code defines five variables (a1-a5) and a string b containing a placeholder {} for each variable. It then uses c = b.format(a1,a2,a3,a4,a5) to replace the placeholders with the values of the variables. Finally, it prints c. The output window shows the resulting string: "Los que viven en Buenos Aires suelen veranear en Mar del Plata, Pinamar, Miramar o Mar de Ajó." [Finished in 0.2s]

```
# prueba de str.format
a1 = "Buenos Aires"
a2 = "Mar del Plata"
a3 = "Pinamar"
a4 = "Miramar"
a5 = "Mar de Ajó"
b = "Los que viven en {} suelen veranear en {}, {}, {} o {}."
c = b.format(a1,a2,a3,a4,a5)
print(c)
```

Puede parecer un poco inútil. Sin embargo podemos referenciar las variables simplemente poniendo el número en acción:

A screenshot of a Python code editor window titled "prueba_de_porcentaje.py". The code is identical to the previous one, but the string b uses numbered placeholders {1}, {0}, {2}, {3}, and {4} instead of {} to refer to the variables a1 through a5. The output window shows the resulting string: "Los que viven en Mar del Plata suelen veranear en Buenos Aires, Pinamar, Miramar o Mar de Ajó." [Finished in 0.1s]

```
# prueba de str.format
a1 = "Buenos Aires"
a2 = "Mar del Plata"
a3 = "Pinamar"
a4 = "Miramar"
a5 = "Mar de Ajó"
b = "Los que viven en {1} suelen veranear en {0}, {2}, {3} o {4}."
c = b.format(a1,a2,a3,a4,a5)
print(c)
```

Podemos ahora guardar los datos que queremos reemplazar en un diccionario y manejarnos dentro del string que vamos a usar como base del reemplazo con las claves del diccionario entre llaves:

The screenshot shows a Sublime Text window with several tabs: oop.py, Etiqueta.py, PyMySQLLeerCursos.py, and prueba_de_porcentaje.py. The code in prueba_de_porcentaje.py is:

```

1 # prueba de str.format
2
3 d = {"c0": "Buenos Aires", "c1": "Mar del Plata"}
4 b = "Los que viven en {c0} suelen veranear en {c1} y los que viven en {c1} veranean en {c0}."
5 c = b.format(**d)
6 print(c)
7

```

The output in the terminal below the code is:

```

Los que viven en Buenos Aires suelen veranear en Mar del Plata y los que viven en Mar
del Plata veranean en Buenos Aires.
[Finished in 0.1s]

```

At the bottom, it says Line 6, Column 9, Tab Size: 4, and Python.

Limitaciones de str.format()

Si bien representa una mejoría respecto a % todavía puede ser muy largo y difícil de seguir si el string a formatear es muy grande.

f-Strings:

Son los llamados “formatted string literals”

The screenshot shows a Sublime Text window with several tabs: oop.py, Etiqueta.py, PyMySQLLeerCursos.py, and prueba_de_porcentaje.py. The code in prueba_de_porcentaje.py is:

```

1 # prueba de str.format
2
3 c0="Buenos Aires"
4 c1="Mar del Plata"
5 b = f"Los que viven en {c0} suelen veranear en {c1} y los que viven en {c1} veranean en {c0}."
6 print(b)
7

```

The output in the terminal below the code is:

```

Los que viven en Buenos Aires suelen veranear en Mar del Plata y los que viven en Mar del Plata veranean en
Buenos Aires.
[Finished in 0.2s]

```

At the bottom, it says Line 7, Column 1, Tab Size: 4, and Python.

El primer paso es guardar en variables lo que vamos a insertar.

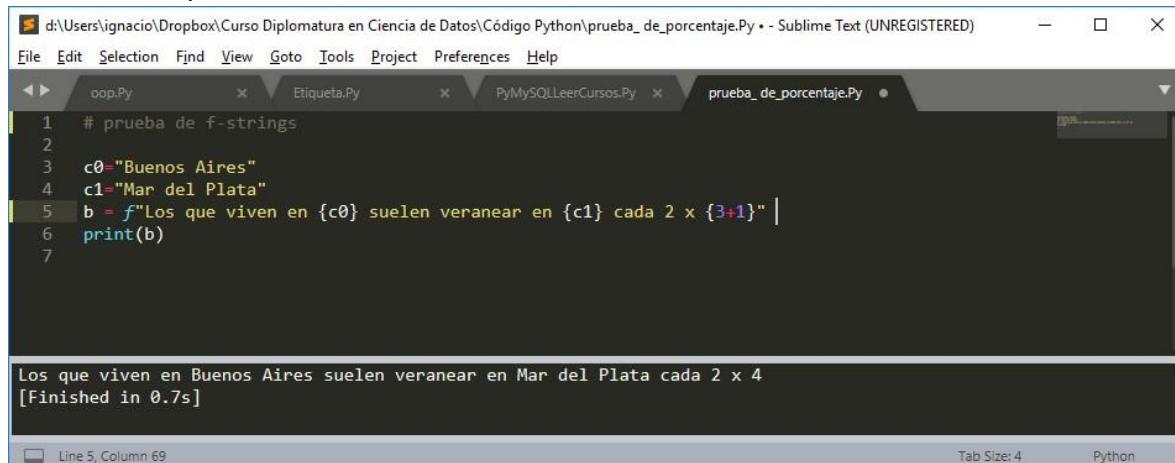
En los lugares donde vamos a insertar esos valores ponemos entre llaves los nombres de las variables que contienen los datos.

Y, a la hora de hacer la inserción simplemente precedemos al string con una f que bien podría estar en mayúscula.

Expresiones arbitrarias.

Los f-strings se evalúan en tiempo de ejecución. Por lo tanto podemos poner cualquier expresión Python válida dentro de ellos:

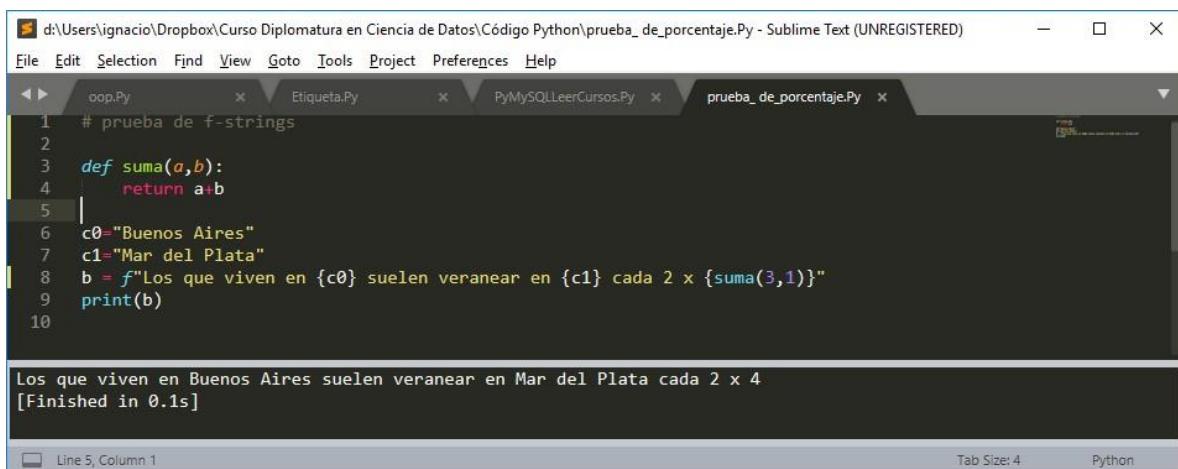
- Conteniendo operaciones aritméticas



A screenshot of the Sublime Text editor showing a Python script named 'prueba_de_porcentaje.py'. The code contains an f-string with a multiplication operation: `b = f"Los que viven en {c0} suelen veranear en {c1} cada 2 x {3*1}"`. The output in the console below shows the result of the multiplication: `Los que viven en Buenos Aires suelen veranear en Mar del Plata cada 2 x 4`.

```
d:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python\prueba_de_porcentaje.py - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
oop.py Etiqueta.py PyMySQLLeerCursos.py prueba_de_porcentaje.py
1 # prueba de f-strings
2
3 c0="Buenos Aires"
4 c1="Mar del Plata"
5 b = f"Los que viven en {c0} suelen veranear en {c1} cada 2 x {3*1}"
6 print(b)
7
Los que viven en Buenos Aires suelen veranear en Mar del Plata cada 2 x 4
[Finished in 0.7s]
Line 5, Column 69
Tab Size: 4 Python
```

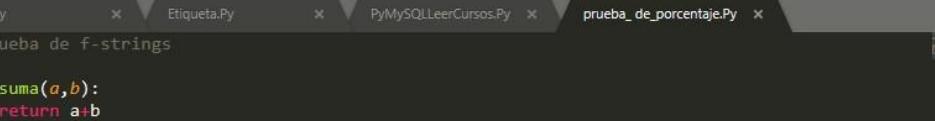
- Conteniendo llamadas a funciones



A screenshot of the Sublime Text editor showing a Python script named 'prueba_de_porcentaje.py'. The code includes a user-defined function 'suma' and uses it within an f-string: `b = f"Los que viven en {c0} suelen veranear en {c1} cada 2 x {suma(3,1)}"`. The output in the console shows the result of the function call: `Los que viven en Buenos Aires suelen veranear en Mar del Plata cada 2 x 4`.

```
d:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python\prueba_de_porcentaje.py - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
oop.py Etiqueta.py PyMySQLLeerCursos.py prueba_de_porcentaje.py
1 # prueba de f-strings
2
3 def suma(a,b):
4     return a+b
5
6 c0="Buenos Aires"
7 c1="Mar del Plata"
8 b = f"Los que viven en {c0} suelen veranear en {c1} cada 2 x {suma(3,1)}"
9 print(b)
10
Los que viven en Buenos Aires suelen veranear en Mar del Plata cada 2 x 4
[Finished in 0.1s]
Line 5, Column 1
Tab Size: 4 Python
```

- Llamando un método en forma directa



The screenshot shows a Sublime Text window with the following details:

- Title Bar:** d:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python\prueba_de_porcentaje.Py - Sublime Text (UNREGISTERED)
- Menu Bar:** File Edit Selection Find View Goto Tools Project Preferences Help
- Tab Bar:** oop.Py, Etiqueta.Py, PyMySQLLeerCursos.Py, prueba_de_porcentaje.Py
- Code Area:** Python code demonstrating f-strings and function calls.
- Status Bar:** Line 8, Column 62, Tab Size: 4, Python

```
1 # prueba de f-strings
2
3 def suma(a,b):
4     return a+b
5
6 c0="Buenos Aires"
7 c1="Mar del Plata"
8 b = f"Los que viven en {c0.upper()} suelen veranear en {c1.upper()} cada 2 x {suma(3,1)}"
9 print(b)
10
```

Output in the console:

```
Los que viven en BUENOS AIRES suelen veranear en MAR DEL PLATA cada 2 x 4
[Finished in 0.1s]
```

f-Strings en múltiples líneas:

Podemos usar f-strings en múltiples líneas, lo que nos permite optimizar la legibilidad de nuestro código:

The screenshot shows a Sublime Text window with the following details:

- File Path:** d:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python\prueba_de_porcentaje.Py
- Editors:** oop.Py, Etiqueta.Py, PyMySQLLeerCursos.Py, prueba_de_porcentaje.Py (active tab)
- Code Content (Line 12 selected):**

```
1 # prueba de f-strings
2
3 c0="Buenos Aires"
4 c1="Mar del Plata"
5
6 b = (
7     f"Los que viven en {c0.upper()}"
8     f" suelen veranear en {c1.upper()}"
9     f" cada 2 x 4"
10    )
11 print(b)
12
```
- Output Panel:** Shows the execution result:

```
Los que viven en BUENOS AIRES suelen veranear en MAR DEL PLATA cada 2 x 4
[Finished in 0.1s]
```
- Status Bar:** Line 12, Column 1 | Tab Size: 4 | Python

Hay que recordar que la f debe estar al comienzo de cada línea y que cada línea debe tener sus comillas de apertura y de cierre.

Para repartir un string entre múltiples líneas nos queda la opción de utilizar un escape con una \

```

d:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python\prueba_de_porcentaje.Py - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
oop.py Etiqueta.py PyMySQLLeerCursos.py prueba_de_porcentaje.py
1 # prueba de f-strings
2
3 c0="Buenos Aires"
4 c1="Mar del Plata"
5
6 b = f"Los que viven en {c0.upper()}" \
7     f" suelen veranear en {c1.upper()}" \
8     f" cada 2 x 4"
9
10 print(b)
11

```

Los que viven en BUENOS AIRES suelen veranear en MAR DEL PLATA cada 2 x 4
[Finished in 0.1s]

Line 9, Column 5 Tab Size: 4 Python

Y también podemos usar comillas triples:

```

d:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python\prueba_de_porcentaje.Py - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
oop.py Etiqueta.py PyMySQLLeerCursos.py prueba_de_porcentaje.py
1 # prueba de f-strings
2
3 c0="Buenos Aires"
4 c1="Mar del Plata"
5
6 b = f"""
7     Los que viven en {c0.upper()}
8     suelen veranear en {c1.upper()}
9     cada 2 x 4
10 """
11
12 print(b)
13

```

Los que viven en BUENOS AIRES
suelen veranear en MAR DEL PLATA
cada 2 x 4
[Finished in 0.1s]

Line 11, Column 1 Tab Size: 4 Python

De paso nos beneficiamos de insertar dentro del string caracteres de retorno de carro que nos ayuden a obtener en pantalla algo similar a lo que vemos en el editor de texto.

Velocidad:

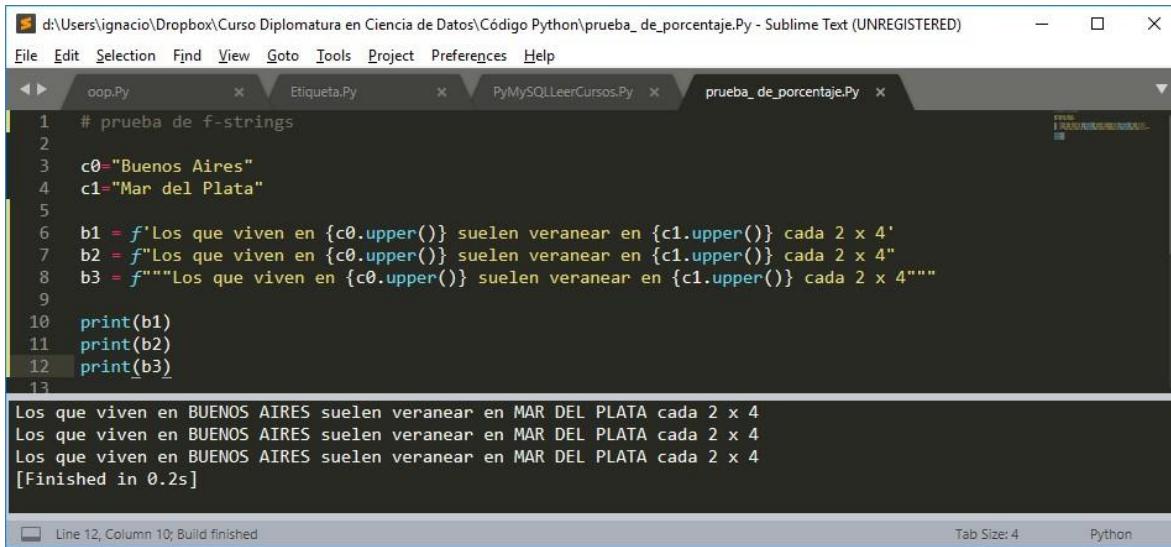
Podemos contar con que los f-strings son entre un 30% y un 50% más rápidos de evaluar que las opciones previas que hemos visto.

Particularidades de los f-Strings:

Ahora que hemos aprendido a usar los f-strings necesitamos tomar nota de algunos detalles de implementación.

Comillas:

Se pueden usar diferentes tipos de comillas:



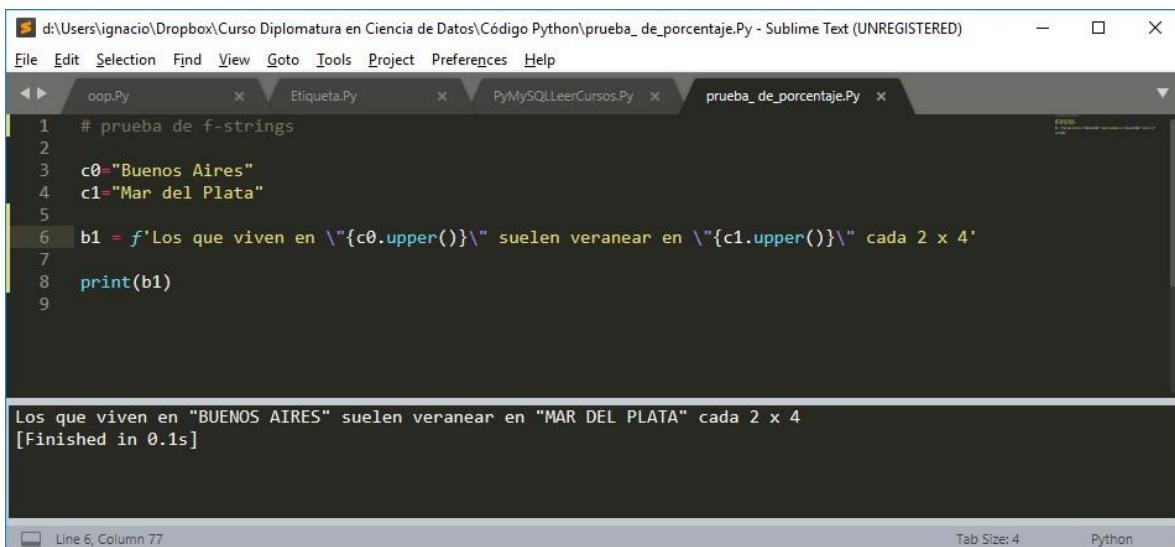
```

d:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python\prueba_de_porcentaje.Py - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
oop.py Etiqueta.py PyMySQLLeerCursos.py prueba_de_porcentaje.py
1 # prueba de f-strings
2
3 c0="Buenos Aires"
4 c1="Mar del Plata"
5
6 b1 = f'Los que viven en {c0.upper()} suelen veranear en {c1.upper()} cada 2 x 4'
7 b2 = f"Los que viven en {c0.upper()} suelen veranear en {c1.upper()} cada 2 x 4"
8 b3 = f"""Los que viven en {c0.upper()} suelen veranear en {c1.upper()} cada 2 x 4"""
9
10 print(b1)
11 print(b2)
12 print(b3)
13
Los que viven en BUENOS AIRES suelen veranear en MAR DEL PLATA cada 2 x 4
Los que viven en BUENOS AIRES suelen veranear en MAR DEL PLATA cada 2 x 4
Los que viven en BUENOS AIRES suelen veranear en MAR DEL PLATA cada 2 x 4
[Finished in 0.2s]

```

Line 12, Column 10; Build finished Tab Size: 4 Python

La cosa se complica si queremos mostrar comillas como parte del texto que nuestro usuario final debe ver. Para eso recurrimos a las sentencias de escape:



```

d:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python\prueba_de_porcentaje.Py - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
oop.py Etiqueta.py PyMySQLLeerCursos.py prueba_de_porcentaje.py
1 # prueba de f-strings
2
3 c0="Buenos Aires"
4 c1="Mar del Plata"
5
6 b1 = f'Los que viven en \"{c0.upper()}\" suelen veranear en \"{c1.upper()}\" cada 2 x 4'
7
8 print(b1)
9

```

```

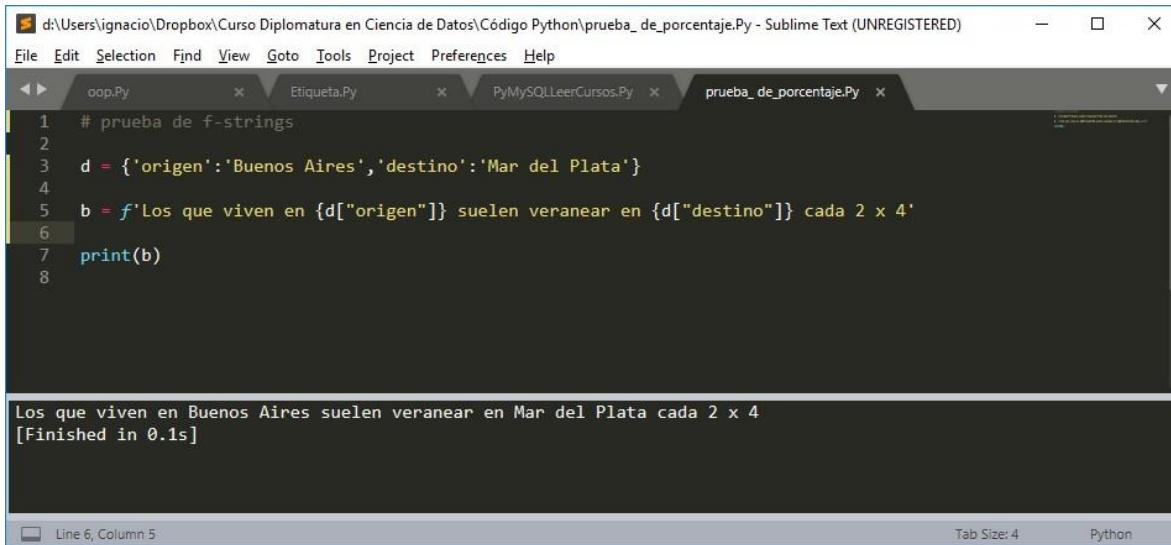
Los que viven en "BUENOS AIRES" suelen veranear en "MAR DEL PLATA" cada 2 x 4
[Finished in 0.1s]

```

Line 6, Column 77 Tab Size: 4 Python

Uso de diccionarios:

Si vamos a usar un diccionario y usamos comillas simples en la definición de las claves deberemos igualmente asegurarnos de usar comillas dobles dentro de los f-strings:



```

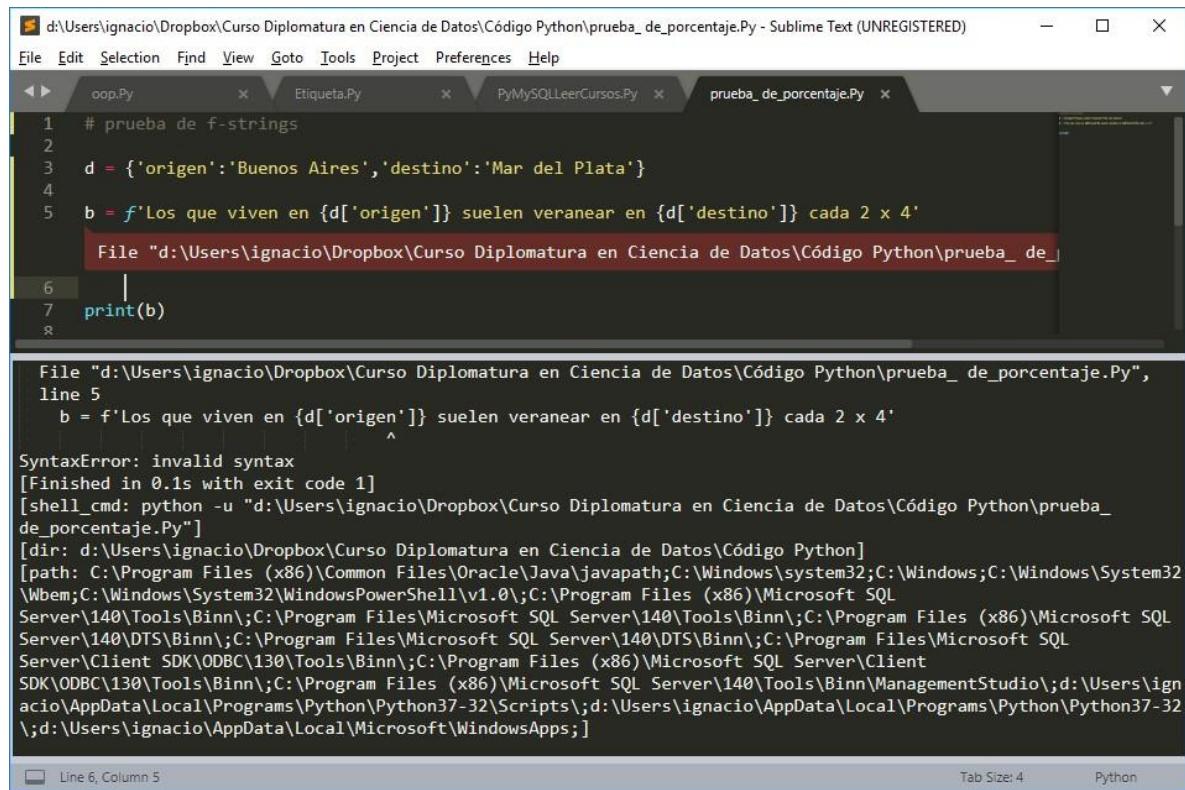
d:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python\prueba_de_porcentaje.py - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
oop.py Etiqueta.py PyMySQLLeerCursos.py prueba_de_porcentaje.py
1 # prueba de f-strings
2
3 d = {'origen':'Buenos Aires','destino':'Mar del Plata'}
4
5 b = f'Los que viven en {d["origen"]} suelen veranear en {d["destino"]} cada 2 x 4'
6
7 print(b)
8

```

Los que viven en Buenos Aires suelen veranear en Mar del Plata cada 2 x 4
[Finished in 0.1s]

Line 6, Column 5 Tab Size: 4 Python

Pero si tratamos de usar comillas simples puede surgir un error de sintaxis:



```

d:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python\prueba_de_porcentaje.py - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
oop.py Etiqueta.py PyMySQLLeerCursos.py prueba_de_porcentaje.py
1 # prueba de f-strings
2
3 d = {'origen':'Buenos Aires','destino':'Mar del Plata'}
4
5 b = f'Los que viven en {d['origen']} suelen veranear en {d['destino']} cada 2 x 4'
6
7 print(b)
8

```

File "d:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python\prueba_de_porcentaje.py", line 5
 b = f'Los que viven en {d['origen']} suelen veranear en {d['destino']} cada 2 x 4'
 ^
SyntaxError: invalid syntax
[Finished in 0.1s with exit code 1]
[shell_cmd: python -u "d:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python\prueba_de_porcentaje.py"]
[dir: d:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python]
[path: C:\Program Files (x86)\Common Files\Oracle\Java\javapath;C:\Windows\system32;C:\Windows;C:\Windows\System32\WBem;C:\Windows\System32\WindowsPowerShell\v1.0;C:\Program Files (x86)\Microsoft SQL Server\140\Tools\Binn\;C:\Program Files\Microsoft SQL Server\140\DT\Binn\;C:\Program Files\Microsoft SQL Server\140\Tools\Binn\;C:\Program Files (x86)\Microsoft SQL Server\140\Tools\Binn\ManagementStudio\;d:\Users\ignacio\AppData\Local\Programs\Python\Python37-32\Scripts\;d:\Users\ignacio\AppData\Local\Programs\Python\Python37-32\;d:\Users\ignacio\AppData\Local\Microsoft\WindowsApps;]

Line 6, Column 5 Tab Size: 4 Python

Llaves

Para que aparezcan las llaves {} nos alcanza con triplicarlas:



```

d:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python\prueba_de_porcentaje.Py - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
oop.py Etiqueta.py PyMySQLLeerCursos.py prueba_de_porcentaje.py
1 # prueba de f-strings
2
3 d = {'origen':'Buenos Aires','destino':'Mar del Plata'}
4
5 b = f'Los que viven en {{d["origen"]}} suelen veranear en {{d["destino"]}} cada 2 x 4'
6
7 print(b)
8
Los que viven en {Buenos Aires} suelen veranear en {Mar del Plata} cada 2 x 4
[finished in 0.1s]
Line 5, Column 79 Tab Size: 4 Python

```

Barras invertidas:

Como vimos hace un rato para hacer aparecer comillas usamos la secuencia de escape.

El mismo truco nos puede ayudar ahora para hacer aparecer la propia barra si tenemos necesidad de que un usuario la vea:



```

d:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python\prueba_de_porcentaje.Py - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
oop.py Etiqueta.py PyMySQLLeerCursos.py prueba_de_porcentaje.py
1 # prueba de f-strings
2
3 d = {'origen':'Buenos Aires','destino':'Mar del Plata'}
4
5 b = f'Los que viven en \\{d["origen"]}\\\\ suelen veranear en {d["destino"]} cada 2 x 4'
6
7 print(b)
8
Los que viven en \Buenos Aires\ suelen veranear en Mar del Plata cada 2 x 4
[finished in 0.1s]
Line 5, Column 42; Build finished Tab Size: 4 Python

```

Comentarios en línea

Las expresiones con f-strings no deben tener comentarios en línea con # dentro de las llaves.

Fuera de la zona entre llaves no hay problemas:



```

d:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python\prueba_de_porcentaje.Py - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
oop.py Etiqueta.py PyMySQLLeerCursos.py prueba_de_porcentaje.py
1 # prueba de f-strings
2
3 d = {'origen':'Buenos Aires','destino':'Mar del Plata'}
4
5 b = f'Los que viven en #{d["origen"]}# suelen veranear en {d["destino"]} cada 2 x 4'
6
7 print(b)
8
Los que viven en #Buenos Aires# suelen veranear en Mar del Plata cada 2 x 4
[finished in 0.1s]
Line 5, Column 24 Tab Size: 4 Python

```

Funciones Lambda

Una función lambda es una pequeña función anónima que puede tomar cualquier cantidad de argumentos pero que puede consistir sólo en una única expresión:

Ejemplo:

A screenshot of the Sublime Text code editor. The window title is "d:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python\lambda.Py - Sublime Text". The menu bar includes File, Edit, Selection, Find, View, Goto, Tools, Project, Preferences, and Help. There are five tabs at the top: oop.py, Etiqueta.py, PyMySQLLeerCursos.py, lambda.py (which is the active tab), and another unnamed tab. The code in lambda.py is:

```
1 # prueba de funciones lambda
2
3 x = Lambda a: a+10
4
5 print(x(5))
```

The output panel shows:

```
15
[Finished in 0.1s]
```

At the bottom, status bars indicate "Line 4, Column 1; Build finished", "Tab Size: 4", and "Python".

Si tenemos más argumentos se parece a:

A screenshot of the Sublime Text code editor. The window title is "d:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python\lambda.Py - Sublime Text". The menu bar includes File, Edit, Selection, Find, View, Goto, Tools, Project, Preferences, and Help. There are five tabs at the top: oop.py, Etiqueta.py, PyMySQLLeerCursos.py, lambda.py (which is the active tab), and another unnamed tab. The code in lambda.py is:

```
1 # prueba de funciones lambda
2
3 x = Lambda a,b,c: a+b+c
4
5 print(x(5,4,3))
```

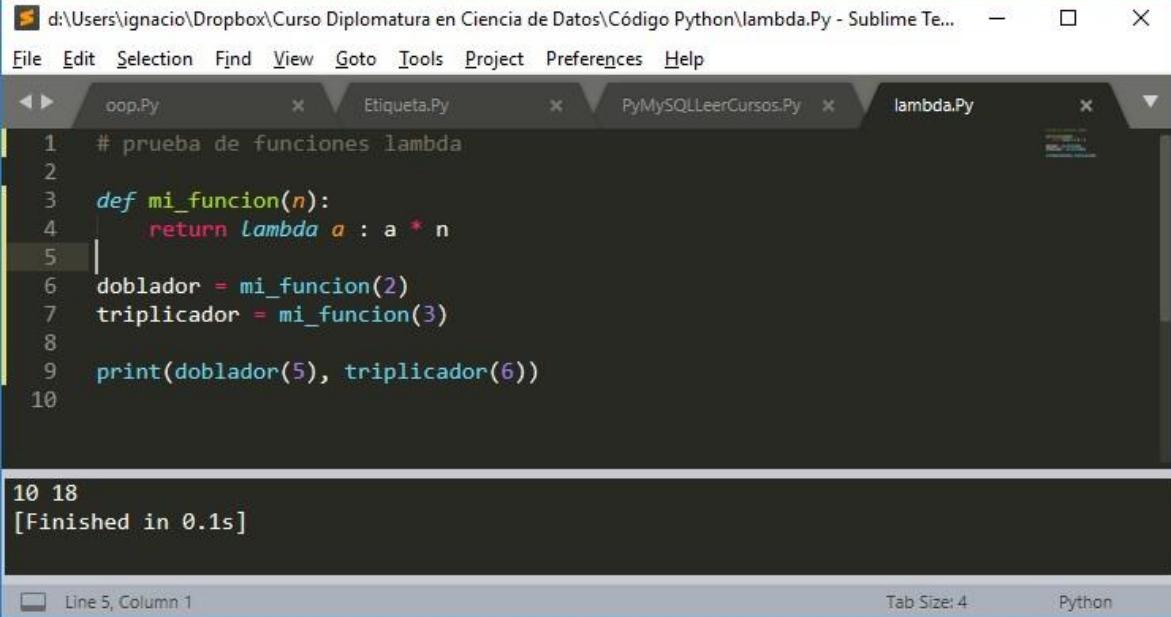
The output panel shows:

```
12
[Finished in 0.1s]
```

At the bottom, status bars indicate "Line 5, Column 14", "Tab Size: 4", and "Python".

Muy interesante, pero, para que sirven?

La utilidad de las funciones lambda resulta más clara cuando las usamos como funciones anónimas dentro de otra función:



The screenshot shows a Sublime Text window with several tabs open. The active tab contains the following Python code:

```

1 # prueba de funciones lambda
2
3 def mi_funcion(n):
4     return lambda a : a * n
5
6 doblador = mi_funcion(2)
7 triplicador = mi_funcion(3)
8
9 print(doblador(5), triplicador(6))
10

```

Below the code, the output is displayed:

```

10 18
[Finished in 0.1s]

```

At the bottom of the interface, status bars show "Line 5, Column 1", "Tab Size: 4", and "Python".

Matplotlib

Vamos a empezar a descubrir cómo obtener salidas gráficas. Las librerías que necesitamos para empezar son:

- matplotlib.pyplot
- numpy

La primera tiene específicamente que ver con los gráficos mientras que la segunda se trata de dotar a Python de un objeto, común en otros lenguajes de programación que queda ausente de la configuración básica de Python: vectores.

Tenemos que empezar, como de costumbre, por instalar estas librerías, y desde una ventana de comando ejecutamos:

```
pip install matplotlib
```

Notamos que nos trae, de regalo:

- Matplotlib

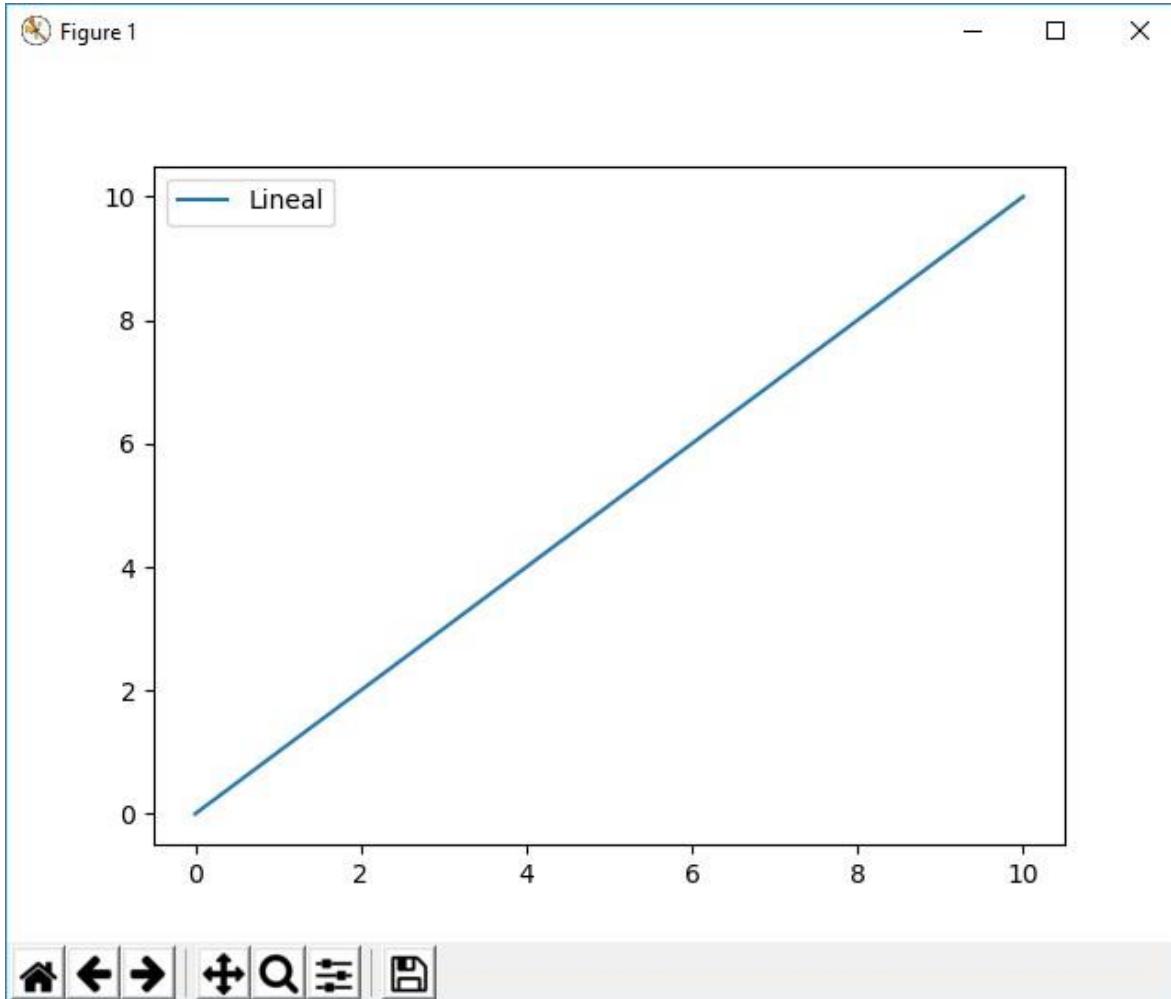
- Dateutil
- Pyparsing
- Numpy
- Cycler
- Kiwisolver

Intentemos ahora un primer ejemplo:

```
d:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python\prueba de matplotlib.Py... ━ ━ X
File Edit Selection Find View Goto Tools Project Preferences Help
◀ ▶ oop.Py x Etiqueta.Py x PyMySQLLeerCursos.Py x prueba de matplotlib.Py x
1 # Ejemplo de matplotlib
2
3 import matplotlib.pyplot as plt
4 import numpy as np
5
6 # preparamos los datos
7
8 x = np.linspace(0,10,100)
9
10 # cargo los datos en el plot_
11
12 plt.plot(x,x,label="Lineal")
13
14 # agrego una leyenda
15
16 plt.legend()
17
18 # muestro el plot
19
20 plt.show()
21
```

Line 10, Column 29 Tab Size: 4 Python

Y, tras una leve espera aparece:



Hemos usado unas cuantas configuraciones por defecto. Ahora el desafío pasa por aprender a tomar control de cada detalle de leyenda, títulos y ejes.

En principio tenemos dos componentes principales:

- Figura
- Ejes

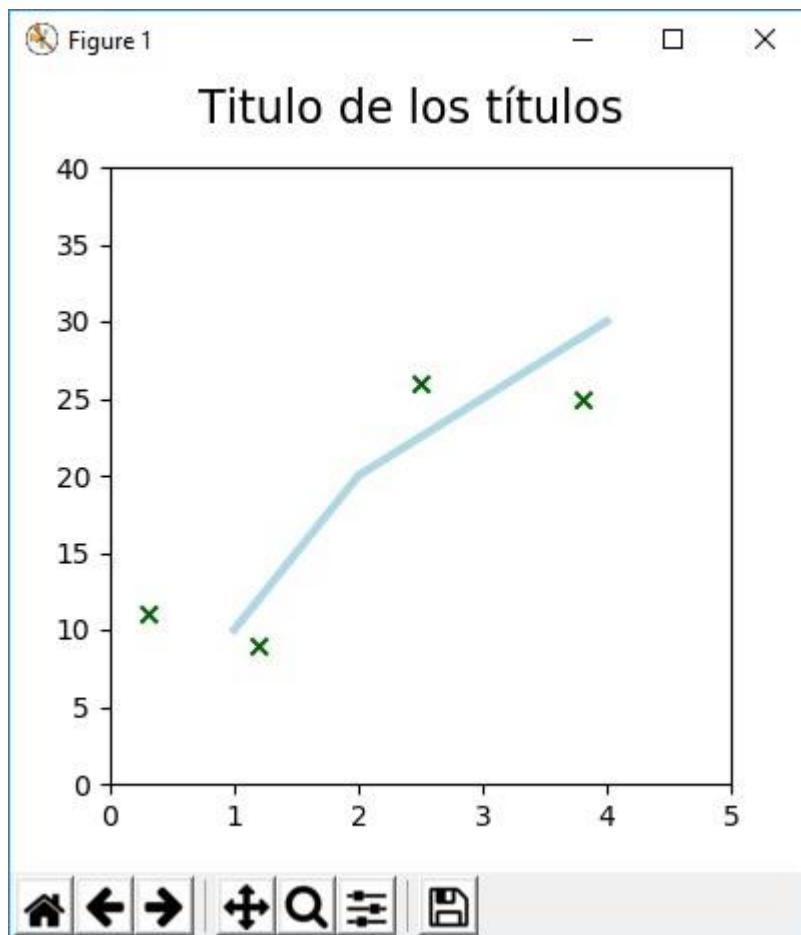
La figura es el elemento central. Podemos tener múltiples figuras independientes.

Dentro de la figura vamos a poder incorporar otros elementos
Con `figure()` creamos un objeto de la clase figura dentro del cual podemos trabajar.

Dentro de `figure()` podemos usar el parámetro `figsize` para administrar el tamaño de la figura.

El método `subplot` nos permite crear diferentes plots dentro de la figura. Sus parámetros son el número de filas, el número de columnas y el índice del que vamos a trabajar.

El método `suptitle` nos sirve para agregar un título general:

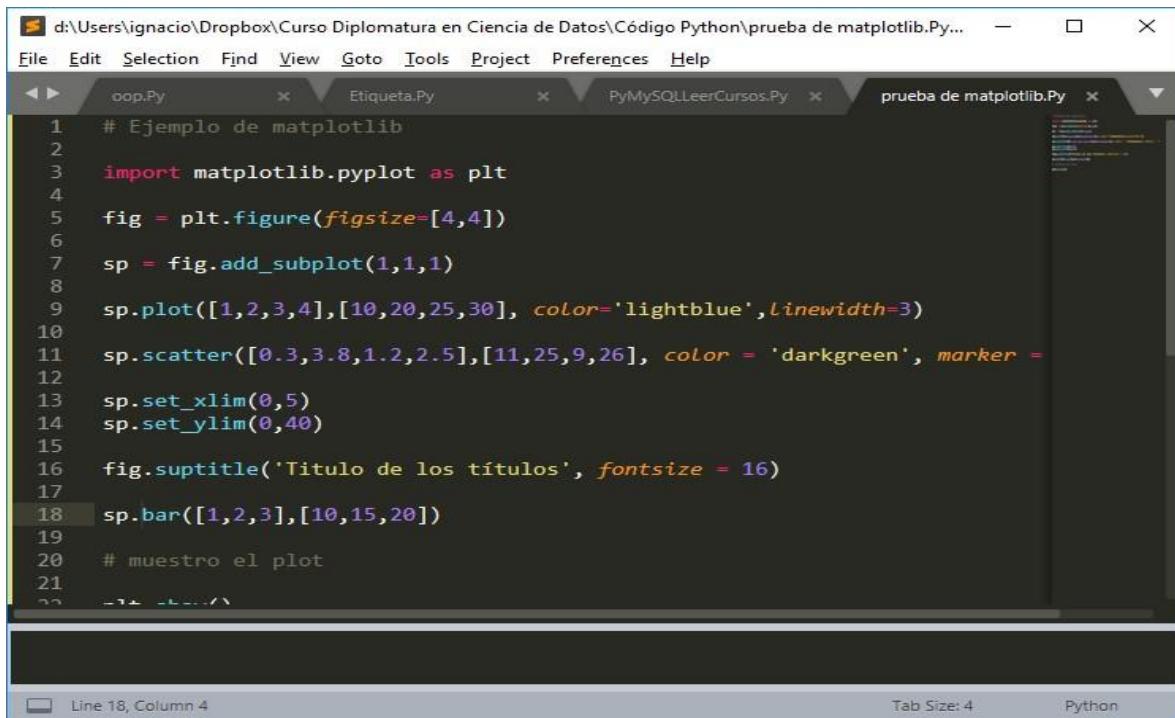


Tenemos distintos tipos de gráficos que podemos realizar:

<code>ax.bar()</code>	Rectángulos verticales
<code>ax.barh()</code>	Rectángulos horizontales

ax.axhline()	Línea horizontal alrededor de los ejes
ax.vline()	Línea vertical alrededor de los ejes
ax.fill()	Polígonos llenos
ax.fill_between()	Relleno entre 0 y un valor de y
ax.stackplot()	Plots superpuestos

Por ejemplo:



```

d:\Users\ignacio\Dropbox\Curso Diplomatura en Ciencia de Datos\Código Python\prueba de matplotlib.Py...
File Edit Selection Find View Goto Tools Project Preferences Help
oop.Py x Etiqueta.Py x PyMySQLLeerCursos.Py x prueba de matplotlib.Py x
1 # Ejemplo de matplotlib
2
3 import matplotlib.pyplot as plt
4
5 fig = plt.figure(figsize=[4,4])
6
7 sp = fig.add_subplot(1,1,1)
8
9 sp.plot([1,2,3,4],[10,20,25,30], color='lightblue', linewidth=3)
10
11 sp.scatter([0.3,3.8,1.2,2.5],[11,25,9,26], color = 'darkgreen', marker =
12
13 sp.set_xlim(0,5)
14 sp.set_ylim(0,40)
15
16 fig.suptitle('Título de los títulos', fontsize = 16)
17
18 sp.bar([1,2,3],[10,15,20])
19
20 # muestro el plot
21
22 plt.show()

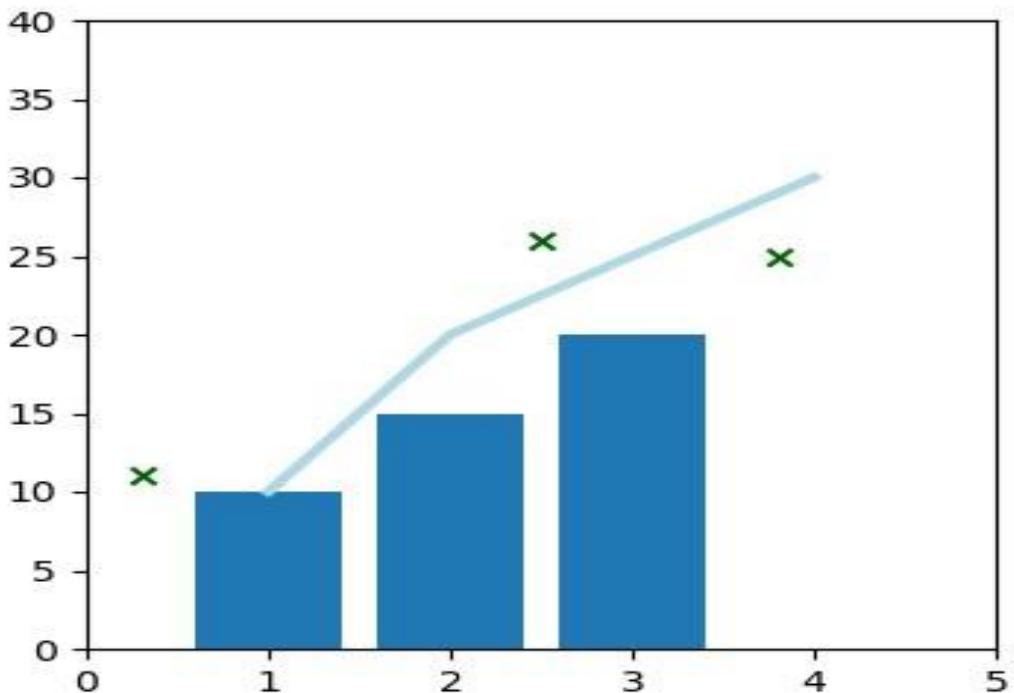
```

Line 18, Column 4 Tab Size: 4 Python

Nos deja:

 Figure 1

Titulo de los títulos



Tenemos también métodos más que útiles para los gráficos científicos en general y, en particular para la ciencia de datos:

ax.arrow()	Arrow
ax.quiver()	2D field of arrows
ax.streamplot()	2D vector fields

ax.hist()	Histogram
ax.boxplot()	Boxplot
ax.violinplot()	Violinplot

Los nombres de la mayoría de las funciones son autodescriptivos. Usaremos los ejercicios para ilustrar el uso de cada una.

Una sugerencia útil pasa por el uso de `fill_between()` para generar gráficos de áreas.

Otras opciones a considerar son:

ax.pcolor()	Pseudocolor plot
ax.pcolormesh()	Pseudocolor plot
ax.contour()	Contour plot
ax.contourf()	Filled contour plot
ax.clabel()	Labeled contour plot

Las variantes son muchas. Lo más difícil, es, sin duda, tener la habilidad para concebir un gráfico que comunique lo que necesitamos comunicar. Esto queda fuera del alcance del curso pero, dejemos algunas sugerencias:

- Lo importante va en el centro.
- La relación de aspecto del elemento central deberá ser 3x2 o 2x3
- La información innecesaria distrae, mejor dejarla fuera.