Python Para Neófitos

Breve Manual De Programación En Python 3.

🄯 Hashmania (2019)

Python Para Neófitos

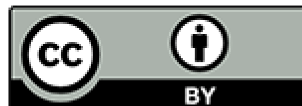
Breve Manual De Programación En Python 3.

hashmania@protonmail.com

CUENTA PARA DONAR AL AUTOR:
(https://www.blockchain.com/btc/address/15J3ZxLrhUTHMMJTnHVeyMraGXASJG3g9x)

bitcoin:15J3ZxLrhUTHMMJTnHVeyMraGXASJG3g9x

🄯 **Hashmania (2019)**



Python Para Neófitos by **Hashmania** is licensed under a
Creative Commons Attribution 4.0 Internacional License.

Python Para Neófitos *es un breve manual destinado a iniciar al programador novel en el lenguaje Python 3. Con tal fin se han seleccionado los ejemplos más simples para cada tema, no siendo necesario, además, seguir el orden estricto que aquí se expone. Para un estudio más exhaustivo o para la investigación de temas que aquí no se tratan se recomienda la consulta del MANUAL OFICIAL DE PYTHON 3 (https://docs.python.org/es/dev/tutorial/).*

ÍNDICE

1. INTRODUCCIÓN

1.1 - Instalación de Python 3
1.2 - Editor de código
1.3 - Ejecutar un .py

2. LOS FUNDAMENTOS DEL LENGUAJE

2.1 - Shell y operaciones aritméticas
2.2 - Comentarios
2.3 - Variables
2.4 - Tipos de datos
2.5 - Input y print
2.6 - Funciones integradas
2.7 - Caracteres de escape
2.8 - Condicionales if
2.9 - Bucle for
2.10 - Bucle while
2.11 - Break, continue y pass
2.12 - Excepciones (try y except)
2.13 - Definición de funciones
2.14 - Documentar funciones
2.15 - Expresión lambda
2.16 - Operadores filter y map
2.17 - Variables globales
2.18 - Clases
2.19 - Herencia
2.20 - Métodos para strings
2.21 - La función len()
2.22 - Encode y decode
2.23 - Lectura y escritura

3. GRUPOS DE DATOS

3.1 - Listas
3.2 - Métodos para listas
3.3 - El tipo range()
3.4 - Comprensión de listas
3.5 - Tuplas
3.6 - Diccionarios
3.7 - Métodos para diccionarios
3.8 - Sets o conjuntos
3.9 - Métodos para conjuntos
3.10 - Operaciones con conjuntos

4. MÓDULOS Y PAQUETES

4.1 - Importar módulo
4.2 - La carpeta __pycache__
4.3 - El atributo __name__
4.4 - Empaquetar módulos

5. LA BIBLIOTECA ESTÁNDAR

5.1 - Matemáticas - math
5.2 - Sistema operativo - os
5.3 - Funciones del intérprete - sys
5.4 - Comandos cmd - subprocess
5.5 - Números aleatorios - random
5.6 - Fecha y hora - datetime
5.7 - Tiempo - time
5.8 - Formato json
5.9 - Interfaz gráfica - tkinter
5.10 - Internet - urllib
5.11 - Subir archivos vía ftp - ftplib
5.12 - Socket
5.13 - Multihilos - threading
5.14 - Gestor de paquetes - pip

6. LIBRERÍAS DE TERCEROS

6.1 - Entorno virtual - virtualenv
6.2 - Ejecutables .exe - pyinstaller
6.3 - Solicitudes http - requests
6.4 - Automatizar navegador - selenium
6.5 - MongoDB - pymongo
6.6 - Matrices y gráficas - numpy y matplotlib
6.7 - Visión artificial - opencv
6.8 - Texto a voz
6.9 - Servidor - flask
6.10 - Otras librerías

1. INTRODUCCIÓN

Python es un lenguaje de programación creado por el holandés Guido Van Rossum en 1991, es utilizado en una gran variedad de ámbitos, desde el desarrollo web, al hacking, las comunicaciones, la administración de sistemas, etc. El aumento de su uso aplicado en el campo de la inteligencia artificial, machine learning y data science, está acelerando su crecimiento, situándose ya como el nº 3 en el ranking sobre los lenguajes de programación más utilizados en el mundo, el Tiobe Index, sólo por detrás de Java y C que llevan mucho tiempo estancados.

Algunos achacan a Python que al tratarse de un lenguaje interpretado es más lento en tiempo de ejecución que otros lenguajes compilados como Java o C. Y es verdad, Python no es lo más adecuado para escribir drivers o controladores de hardware. Sin embargo, esto no es un gran problema, las diferencias en velocidad son de muy pocos milisegundos y hoy en día el cuello de botella en los proyectos de desarrollo de software ya no está en la CPU. Por el contrario, Java o C no pueden compararse a Python en la sencillez y velocidad con que puede crearse un programa, aquí Python ha tenido un uso muy extendido como herramienta de scripting, sustituyendo paulatinamente incluso a scripts escritos en Bash de Linux. (Script: del inglés guión, es un conjunto de órdenes guardadas en un archivo de texto plano, generalmente muy ligero, que es ejecutado por lotes o línea a línea en tiempo real por un intérprete.) Otro campo en el que destaca Python es el web scraping y el crawling, técnicas para sustraer datos de sitios web.

Python es un lenguaje de código abierto, de alto nivel y multiparadigma, soporta orientación a objetos, programación imperativa y programación funcional. Al ser multiplataforma puede ser ejecutado en un windows, un mac, un linux, un teléfono móvil, una raspberry pi o controlar una placa de arduino.

Además de ser uno de los lenguajes más amados por los programadores es también uno de los más adecuados para iniciarse en la programación debido a su sencilla sintaxis. El código que sigue los principios de Python de legibilidad y transparencia se dice que es "pythonico". Los principios fueron descritos por el desarrollador Tim Peters en **El Zen de Python**:

- Bello es mejor que feo.
- Explícito es mejor que implícito.
- Simple es mejor que complejo.
- Complejo es mejor que complicado.
- Plano es mejor que anidado.
- Espaciado es mejor que denso.
- La legibilidad es importante.
- Los casos especiales no son lo suficientemente especiales como para romper las reglas.
- Lo práctico le gana a la pureza.
- Los errores nunca deberían pasar silenciosamente.
- A menos que se silencien explícitamente.
- Frente a la ambigüedad, evitar la tentación de adivinar.
- Debería haber una, y preferiblemente solo una, manera obvia de hacerlo.
- A pesar de que esa manera no sea obvia a menos que seas holandés.
- Ahora es mejor que nunca.
- Aunque "nunca" es a menudo mejor que "ahora mismo".
- Si la implementación es difícil de explicar, es una mala idea.
- Si la implementación es fácil de explicar, puede que sea una buena idea.
- Los espacios de nombres son una gran idea, ¡tengamos más de esos!

La comunidad de usuarios de Python ha adoptado una guía de estilo que facilita la lectura del código y la consistencia entre programas de distintos usuarios. Esta guía no es de seguimiento obligatorio, pero es altamente recomendable. El documento se denomina PEP 8 e incluye sugerencias relacionadas con las pautas de estilo. La indentación correcta, 4 espacios, nada de tabulaciones. La extensión de una línea de código, no más de 79 caracteres. La separación de fragmentos de código por medio de líneas en blanco. Las importaciones se colocan siempre al principio en este orden: módulos estándar, de terceros, y locales. Uso apropiado de los comentarios. Convenciones en los nombres utilizados. Sugerencias varias sobre funciones, clases, variables, etc.

Python es administrado por la Python Software Foundation. Posee una licencia de código abierto que es compatible con la Licencia pública general de GNU y garantiza, por tanto, a los usuarios finales (personas, organizaciones, compañías) la libertad de usar, estudiar, compartir (copiar) y modificar el software. Su propósito es doble: declarar que el software cubierto por esta licencia es libre, y protegerlo (mediante una práctica conocida como copyleft) de intentos de apropiación que restrinjan esas libertades a nuevos usuarios cada vez que la obra es distribuida, modificada o ampliada. Esta licencia fue creada originalmente por Richard Stallman fundador de la Free Software Foundation (FSF) para el proyecto GNU.

> *"Decidí escribir un intérprete para el nuevo lenguaje de scripting que había estado ideando recientemente, un descendiente de ABC que atraería a los hackers de Unix y C. Elegí el nombre de **Python** para el proyecto encontrándome en un estado de ánimo ligeramente irreverente y siendo un gran fan de Monty Python's Flying Circus". **Guido Van Rossum**, Benevolent Dictator for Life of Python.*

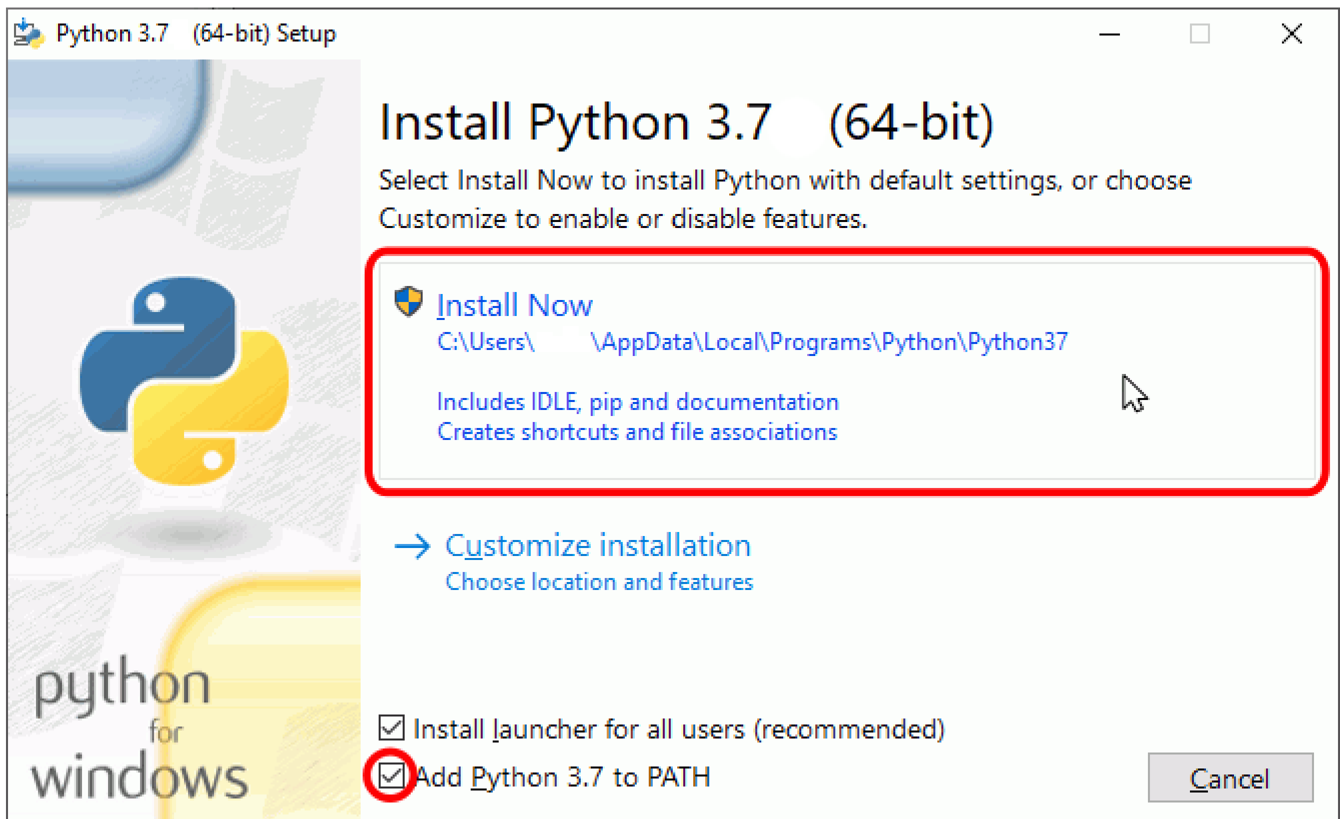
En julio de 2018 Guido Van Rossum declinó el honor de ser el Benevolente Dictador Vitalicio de Python, sin dejar un sucesor o sucesora y con una declaración altisonante:

> *Entonces, ¿qué van a hacer todos ustedes? ¿Crear una democracia? ¿Anarquía? ¿Una dictadura? ¿Una federación?*

1.1 Instalación de Python 3

En linux y mac Python ya viene preinstalado por defecto. En windows descargaremos el ejecutable desde la página oficial -> PYTHON.ORG (https://www.python.org/)

Aunque coexisten dos versiones de Python, la 2 y la 3, aquí nos centraremos en **Python 3**, actualmente la última versión es la 3.7.4. Una vez descargado el instalador (Windows x86-64 executable installer), haga doble clic en él para iniciar la instalación. Se aconseja marcar la casilla "Add Python to PATH" para poder ejecutar programas desde la línea de comandos.

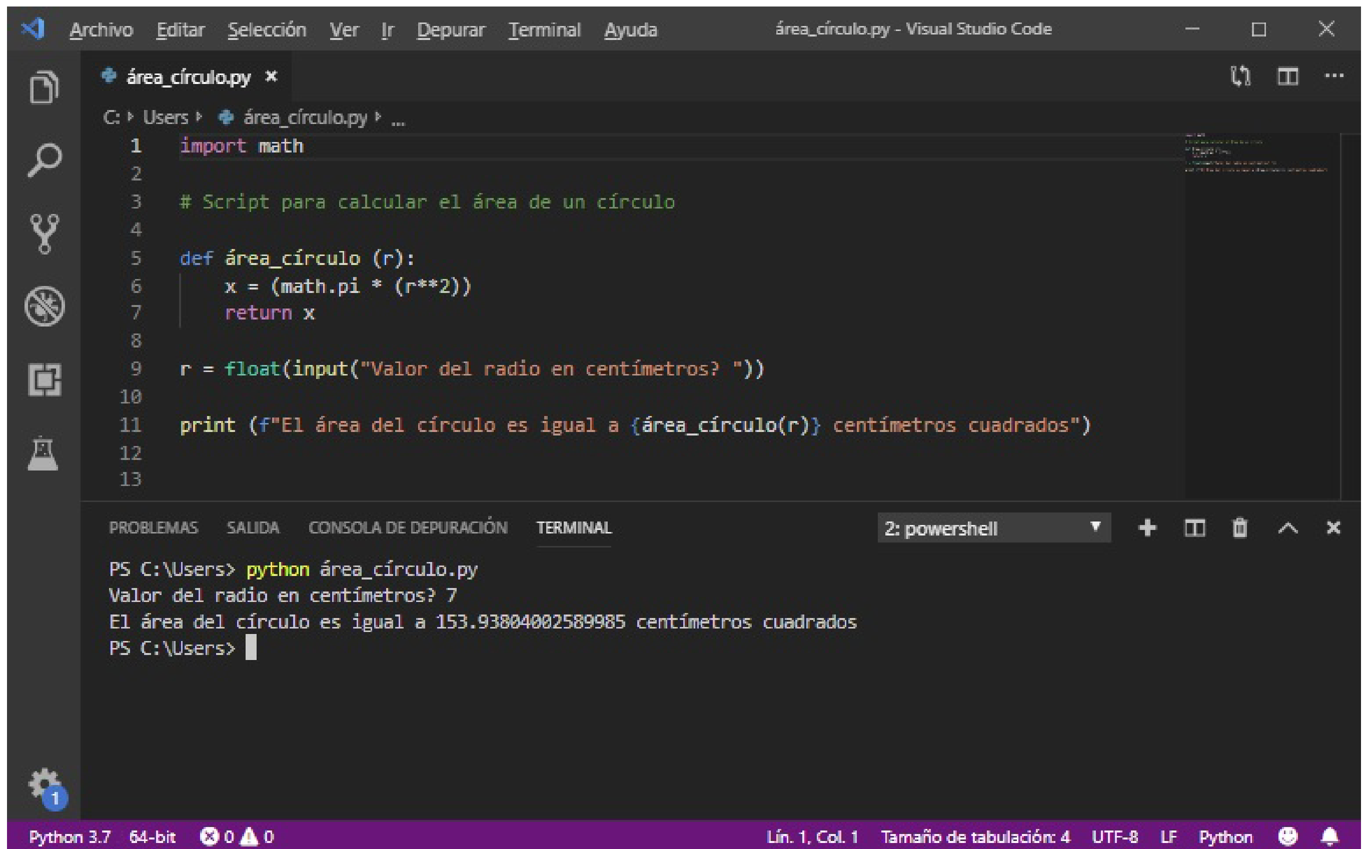


Para iniciar Python sólo hay que escribir en la consola de comandos: **python**, y **quit()** para salir.

```
C:\Windows\System32\cmd.exe - python
C:\Users>python
Python 3.7.4 (v3.7.4:ef4ec6ed12, Mar 25 2019, 22:22:05) [MSC v.1916 64 bit (AMD64)]
 on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> |
```

1.2 Editor de código

Es posible escribir código Python con el bloc de notas, al ser texto plano tan sólo hay que guardar el archivo con la extensión **.py**, pero es muy recomendable utilizar un editor que nos facilite la tarea. Aunque existen diversos editores de código: Atom, Sublime Text, VISUAL STUDIO CODE (https://code.visualstudio.com) es probablemente el más utilizado hoy en día.



1.3 Ejecutar un .py

Podemos ejecutar un archivo **.py** con un **doble click**, se abrirá la ventana de la consola que ejecutará el programa, al terminar automáticamente se cerrará. Para ver algún valor retornado en pantalla deberemos insertar algún input o contador de tiempo al final del script para que nos dé tiempo a ver la salida.

Otra opción es abrir el programa desde la consola **cmd** o desde **PowerShell**, así la ventana no se cerrará al terminar. Nos situaremos en el directorio del archivo y escribiremos: **python nombre_programa.py**. Evitaremos muchos cd.. (subir al directorio padre) para encontrar el archivo si en la carpeta que lo contiene escribimos cmd en la ruta, lo que nos abrirá la consola en el directorio deseado.

Puede sernos útil a la hora de depurar un programa abrirlo con el **IDLE** de Python que se instala junto con el lenguaje -> IDLE: File/Open/archivo.py y Run/Run Module

O podemos ejecutar un archivo.py desde **Visual Studio Code** pulsando en la pestaña Terminal y ejecutarlo en su consola.

Volver al índice

2. LOS FUNDAMENTOS DEL LENGUAJE

2.1 Shell y operaciones aritméticas

En informática, el shell o intérprete de comandos es el programa informático que provee una interfaz de usuario para acceder a los servicios del sistema operativo. El shell es la ventana y el símbolo >>> es el prompt. En Python podemos usar el shell como una calculadora pero en lugar de pulsar el símbolo = presionaremos ENTER. Así se realizan las operaciones básicas:

- **Suma** (20 + 5 = 25)
- **Resta** (20 - 5 = 15)
- **Multiplicación** (20 * 5 = 100)
- **Potencia** (3 ** 2 = 9)
- **División** (21 / 5 = 4.2)
- **División sin decimales** (21 // 5 = 4)
- **Módulo**, resto de la división (21 % 5 = 1)

La precedencia de operadores es un conjunto de reglas que especifica en qué orden deben ser evaluadas las operaciones de una expresión. En la siguiente lista los operadores han sido listados en orden de menor a mayor precedencia:

- or
- and
- not
- <, <=, >, >=, !=, ==
- suma y resta
- multiplicación, división y módulo
- +, - (positivo y negativo)
- potencia

2.2 Comentarios

Un comentario es un texto que se escribe en el programa para ayudarnos a entender el código. En Python los comentarios se escriben precedidos por una almohadilla (**#**) que le indica al programa que el texto que va detrás no tiene que ejecutarse. También se pueden ultilizar las triples comillas dobles (**"""**) al principio y al final de un comentario multilínea.

2.3 Variables

Python es un lenguaje interpretado de **tipado dinámico**, no se tiene que especificar cuál es el tipo de valor que se le asigna a una variable de antemano y puede tomar distintos valores en otro momento. Es **case sensitive** por lo que distingue entre mayúsculas y minúsculas.

El nombre de **una variable no puede comenzar nunca por un número ni por un símbolo** y para reducir el esfuerzo a la hora de leer el código se recomienda seguir la siguiente pauta de estilo para nombrar una variable:

CamelCase -> NombreDeAlumno = "Antonio"
snake_case -> nombre_de_alumno = "Salvador" -> Esta es la forma preferida.

En Python todo es un objeto y como tal puede ser asignado a una variable, puede ser un texto, una foto, un video, código html, etc. Python crea una dirección de memoria al valor de cada variable. Al cambiar el valor de una variable también cambia el enlace a la dirección de memoria.

```
a = "Hola Mundo"
a
```

```
'Hola Mundo'
```

2.4 Tipos de datos

int -> números enteros: 1, 4, -3, etc.
float -> decimales: 0.5, 3.14, -4.56, etc.
boolean -> Verdadero o Falso: True, False
strings -> Cadenas de texto: 'Hola', "hola mundo", "5", etc.

La función **type()** nos muestra el tipo de dato que le pasamos como argumento entre paréntesis:

```
a = 3.14
type(a)
```

```
float
```

También podemos hacer conversiones entre tipos de datos con:

int() -> convierte a entero un decimal o una cadena: int(3.14) -> 3
float() -> convierte a decimal un entero o una cadena: float("3.14") -> 3.14
str() -> convierte a cadena un decimal o un entero: str(3.14) -> "3.14"

```
# indicará un error si intentamos convertir un texto a decimal
float('hola mundo')
```

```
---------------------------------------------------------------------------
-
ValueError                                Traceback (most recent call las
t)
<ipython-input-2-647801b2df06> in <module>
      1 # indicará un error si intentamos convertir un texto a decimal
----> 2 float('hola mundo')

ValueError: could not convert string to float: 'hola mundo'
```

2.5 Input y print

La función **input()** permite obtener texto escrito por teclado. Al llegar a la función, el programa se detiene esperando que se escriba algo y se pulse la tecla ENTER.

```
x = input("Cómo te llamas? ")
```

```
Cómo te llamas? Leonardo
```

Lo que devuelve la función input() siempre es un string, aunque escribamos números lo considerará como texto. Deberemos introducir el input en int() o float() dependiendo de si queremos un número entero o un decimal.

```
y = int(input("Cuántos años tienes? "))
```

```
Cuántos años tienes? 75
```

```
# restamos 18 a la edad
z = y - 18
```

La función **print()** permite mostrar texto en pantalla. El texto a mostrar se indica como argumento de la función y se puede escribir indistintamente entre comillas simples (') o entre comillas dobles ("). También se pueden usar las triples comillas dobles (""") para textos multilínea.

```
# imprimir concatenando variables y cadenas
print(x + " dejó de ser menor de edad hace " + str(z) + " años.")
```

```
Leonardo dejó de ser menor de edad hace 57 años.
```

```
# imprimir con f formateando variables y cadenas
print(f"{x} dejó de ser menor de edad hace {z} años.")
```

```
Leonardo dejó de ser menor de edad hace 57 años.
```

2.6 Funciones integradas

El intérprete Python tiene un gran número de funciones preconstruidas integradas dentro del módulo __builtins__ que están siempre disponibles. Podemos listar todas estas funciones mediante la función **dir (__builtins__)** Por ejemplo, la función **max()** nos devuelve el valor máximo de una serie de números: max(3,7,5) -> 7

También podemos solicitar ayuda pasándole a **help()** la función como argumento:

```
help (max)
```

```
Help on built-in function max in module builtins:

max(...)
    max(iterable, *[, default=obj, key=func]) -> value
    max(arg1, arg2, *args, *[, key=func]) -> value

    With a single iterable argument, return its biggest item. The
    default keyword-only argument specifies an object to return if
    the provided iterable is empty.
    With two or more arguments, return the largest argument.
```

Incluso podemos pedir ayuda para los métodos de una función: **help(*función.método*)**

```
help(str.count)
```

```
Help on method_descriptor:

count(...)
    S.count(sub[, start[, end]]) -> int

    Return the number of non-overlapping occurrences of substring sub in
    string S[start:end].  Optional arguments start and end are
    interpreted as in slice notation.
```

Por ejemplo, la función **hex()** convierte un entero a hexadecimal:

```
hex(12)
```

```
'0xc'
```

La función **round()** redondea un número flotante a los decimales que le indiquemos:

```
pi = 3.14159265359
round(pi,3)
```

```
3.142
```

2.7 Caracteres de escape

Un carácter de escape es un carácter individual que suprime cualquier significado especial del carácter que le sigue. Para escapar caracteres dentro de cadenas de texto se usa el backslash \ seguido de cualquier carácter ASCII. Estos son los más utilizados:

Secuencia de escape	Significado
\\	Backslash
\'	Comillas simples
\"	Comillas dobles
\n	Nueva línea
\r	Retorno de carro
\t	Tabulador horizontal
\b	Elimina el carácter anterior
\w	Carácter minúscula
\W	Carácter mayúscula
\s	Engloba minúsculas y mayúsculas
\S	cualquier carácter que no es espacio en blanco
\d	número entre 0 - 9
\D	cualquier carácter que no es un numero
\N{name}	carácter Unicode

```
# imprimir comillas dobles dentro de comillas dobles
print("La \"boiserie\" es lo más bonito de la casa.")
```

```
La "boiserie" es lo más bonito de la casa.
```

```
# imprimir un salto de línea
print("hola \n mundo")
```

```
hola
 mundo
```

```
# imprime un tabulado
print("hola \t mundo")
```

```
hola 	 mundo
```

```
# Imprime un carácter Unicode
print("\N{DAGGER}")
```

```
†
```

2.8 Condicionales if

if, **elif**, **else** -> si se cumple a, si se cumple b, si no c

Una sentencia condicional **if** ejecuta su bloque de código interno sólo si se cumple cierta condición. Se define usando la palabra clave if seguida de la condición y del bloque de código.

Condiciones adicionales, si las hay, se introducen usando **elif** seguida de la condición y su bloque de código. Todas las condiciones se evalúan secuencialmente hasta encontrar la primera que sea verdadera, y su bloque de código asociado es el único que se ejecuta.

Opcionalmente, puede haber un bloque final con la palabra clave **else** seguida del bloque de código, se ejecuta sólo cuando todas las condiciones anteriores fueron falsas.

Aquí podemos observar otra de las propiedades de python: es un **lenguaje indentado**, es decir, se usa el espaciado (4 espacios) para delimitar los bloques de instrucciones en el lenguaje.

```python
# imprimirá ¡Pasa con cuidado! porque no se cumplen las dos primeras condiciones

color = "naranja"
if color == "rojo":
    print("¡Debes esperar!")
elif color == "verde":
    print("¡Puedes pasar!")
else:
    print("¡Pasa con cuidado!")

# observa como después de : va siempre una indentación de 4 espacios
```

```
¡Pasa con cuidado!
```

En los condicionales se utilizan mucho las expresiones boolenas y los operadores lógicos and, or y not:

Expresión	Significado
a == b	a es igual a b
a != b	a es distinto de b
a < b	a es menor que b
a <= b	a es menor o igual que b
a > b	a es mayor que b
a >= b	a es mayor o igual que b
a and b	a y b
a or b	a o b
not a	si no es a

2.9 Bucle for

Un **bucle for** repite unas instrucciones un número predeterminado de veces, cada repetición es una iteración. La variable de control no es necesario definirla antes del bucle. La sintaxis es la siguiente:

```
elemento = "gol"
for i in elemento:
    print (i)
```

```
g
o
l
```

En ocasiones se necesita una variable que cuente cuántas veces ha ocurrido algo (**contador**) o que acumule valores (**acumulador**).

```
# ejemplo de contador

cuenta = 0
for i in range(1, 6):
    if i % 2 == 0:
        cuenta = cuenta + 1
print(f"Desde 1 hasta 5 hay {cuenta} múltiplos de 2")
```

```
Desde 1 hasta 5 hay 2 múltiplos de 2
```

```
# ejemplo de acumulador

suma = 0
for i in [1, 2, 3, 4]:
    suma = suma + i
print(f"La suma de los números de 1 a 4 es {suma}")
```

```
La suma de los números de 1 a 4 es 10
```

2.10 Bucle while

Un **bucle while** permite repetir la ejecución de instrucciones mientras se cumpla una condición.

```
i = 1
while i <= 3:
    print(i)
    i += 1
print("Programa terminado")
```

```
1
2
3
Programa terminado
```

La ventaja del bucle while es que el número de iteraciones puede no estár definido antes de empezar el bucle.

```
password = input("Introduce tu clave: ")
while password != 'python':
    print("Es incorrecta, inténtalo de nuevo.")
    password = input("Introduce tu clave: ")
print("La clave es correcta.")
```

```
Introduce tu clave: admin
Es incorrecta, inténtalo de nuevo.
Introduce tu clave: python
La clave es correcta.
```

2.11 Break, continue y pass

Los bucles for y while pueden ser interrumpidos con la sentencia **break**, cuando se ejecuta el programa sale del bucle y continúa ejecutando el resto del código:

```
# cuando llega a la letra l se sale del bucle

palabra = "Australopithecus"
for n in palabra:
    print(n)
    if n == "l":
        break
print("terminado")
```

```
A
u
s
t
r
a
l
terminado
```

La sentencia **continue**, cuando se ejecuta, obliga a Python a dejar de ejecutar el código que haya dentro del bucle y a iniciar una nueva iteración (es decir, permite saltar una iteración):

```
# se salta la letra c y sigue el bucle

palabra = "bici"
for n in palabra:
    if n == "c":
        continue
    print(n)
print("terminado")
```

```
b
i
i
terminado
```

La sentencia **pass** se usa principalmente cuando el programa requiere una sentencia pero no quieres que pase nada porque todavía no la has pensado. Suele usarse a menudo mientras escribimos un código, a modo de boceto:

```
# hay una condición cuando llegue a la letra c, pero de momento no hace nada

palabra = "bitcoin"
for n in palabra:
    if n == "c":
        pass
    print(n)
```

```
b
i
t
c
o
i
n
```

Volver al índice

2.12 Excepciones (try y except)

Una de las características más poderosas del Python es la gestión de excepciones. Debido a lo dinámico que puede ser el lenguaje es muy común que el intérprete se tope con múltiples fuentes de error. Para evitar esta situación existen las sentencias **try** y **except**. Las mismas nos permitirán «atrapar» excepciones y como resultado, responder sin que el programa falle.

Un ejemplo típico y sencillo sería la división por 0, si indicamos qué hacer cuando se pueda producir el error evitaremos fallos no deseados, si no obtendremos el consecuente mensaje de error ya que, evidentemente, no puede dividir por 0.

```
a = 5
b = 0
print(a/b)
```

```
---------------------------------------------------------------------------
-
ZeroDivisionError                         Traceback (most recent call las
t)
<ipython-input-21-9eb79d2caede> in <module>
      1 a = 5
      2 b = 0
----> 3 print(a/b)

ZeroDivisionError: division by zero
```

Para evitar los errores le podemos decir a python que intente (try) hacer algo y si no puede que haga otra cosa (except):

```
a = 5
b = 0
try:
    print(a/b)
except:
    print("No se puede dividir por 0")
```

```
No se puede dividir por 0
```

Dado que dentro de un mismo bloque try pueden producirse excepciones de distinto tipo, es posible utilizar varios bloques except, cada uno para capturar un tipo distinto de excepción.

Esto se hace especificando a continuación de la sentencia except el nombre de la excepción que se pretende capturar. Un mismo bloque except puede atrapar varios tipos de excepciones, lo cual se hace especificando los nombres de las excepciones separados por comas a continuación de la palabra except. Es importante destacar que si bien luego de un bloque try puede haber varios bloques except, se ejecutará, a lo sumo, uno de ellos.

Finalmente, puede ubicarse un bloque **finally** donde se escriben las sentencias de finalización, que son típicamente acciones de limpieza. La particularidad del bloque finally es que se ejecuta siempre, haya surgido una excepción o no. Si hay un bloque except, no es necesario que esté presente el finally, y es posible tener un bloque try sólo con finally, sin except.

```
a = 5
b = 0

try:
    print(a/b)
    # Intentará dividir
except ZeroDivisionError:
    print ("No se puede dividir por 0")
    # si el error es al dividir por 0
except:
    print("Tiene algún error en su código")
    # si fuera otro tipo de error no especificado
finally:
    print("Debería estudiar más")
    # finalmente se ejecutará esta orden
```

```
No se puede dividir por 0
Debería estudiar más
```

2.13 Definición de funciones

Una función es una secuencia de instrucciones que tiene un nombre. Podemos ahorrarnos mucho tiempo al programar si a una porción o bloque de código le damos un nombre, así podremos recurrir a ella por su nombre en lugar de tener que volver a escribir todo el código de nuevo. Hay que tener en cuenta que no podemos llamar a nuestras funciones con el nombre de otras ya creadas anteriormente o con el nombre de las palabras reservadas por el sistema.

La sintaxis para una definición de función en Python es:

```
def nombre_función(parámetros):
    sentencias
```

```
def suma():
    print(a+b)

a = float(input("Introduce un número: "))
b = float(input("Otro número: "))
suma()
```

```
Introduce un número: 3.7
Otro número: 4.35
8.05
```

Las funciones en Python admiten argumentos y permiten devolver valores.

```
# cada vez que llamemos a las funciones devolverán el área y el volumen

def área_círculo(radio):
    pi = 3.1415
    x = (pi * (radio**2))
    return x

def volumen_esfera(radio):
    pi = 3.1415
    volumen=(4/3)*pi*radio**3
    return volumen

radio = float(input("Valor del radio en metros? "))
print (f"""El área de un círculo con radio de {radio} metros es igual a:
{área_círculo(radio)} metros cuadrados,
y el volumen de la esfera que lo contiene es igual a:
{volumen_esfera(radio)} metros cúbicos""")
```

```
Valor del radio en metros? 0.62
El área de un círculo con radio de 0.62 metros es igual a:
1.2075926000000001 metros cuadrados,
y el volumen de la esfera que lo contiene es igual a:
0.9982765493333332 metros cúbicos
```

2.14 Documentar funciones

Podemos documentar cada función para indicar qué hace, qué parámetros recibe y qué retorna. Colocaremos un docstring al principio de la función para describirla:

```
def saludo(nombre):
    """ La función saludo(nombre) imprime
    un saludo dirigido a (nombre)."""
    return (f"Hola {nombre}")

saludo("Miguel")
```

```
'Hola Miguel'
```

```
help(saludo)
```

```
Help on function saludo in module __main__:

saludo(nombre)
    La función saludo(nombre) imprime
    un saludo dirigido a (nombre).
```

2.15 Expresión lambda

lambda ejecuta una expresión, que puede darse con o sín parámetros, y retorna un resultado. Algunos también la denominan función anónima ya que no tiene nombre. Su sintaxis es: **lambda parámetros:expresión**

```
f = lambda x: x ** 2

print(f(3))
```

```
9
```

2.16 Operadores filter y map

filter() permite filtrar elementos de una lista o de cualquier objeto iterable que cumplan una condición.

```
# filtra los divisibles por 3

a = [1, 5, 6, 9, 13]

list( filter(lambda a: a%3 == 0, a) )
```

```
[6, 9]
```

map() toma una función y un iterable y retorna una lista con la función aplicada a cada argumento.

```
# suma 10 a cada elemento
nums = [13, 26, 34, 100, 23]
resultado = list(map(lambda x:x+10, nums))

resultado
```

```
[23, 36, 44, 110, 33]
```

2.17 Variables globales

En Python las variables dentro de las funciones se consideran locales, es decir, no podremos acceder a ellas fuera de la función. Si intentamos a acceder a la variable de una función desde fuera de ella nos indicará que esa variable no está definida:

```
def garito():
    local = "La Granja"
    print("Esto está dentro de la función")

garito()
print(local)
```

```
Esto está dentro de la función

---------------------------------------------------------------------------
-
NameError                                 Traceback (most recent call las
t)
<ipython-input-49-bc9fd68d93a7> in <module>
      4
      5 garito()
----> 6 print(local)

NameError: name 'local' is not defined
```

Con la palabra **global** podremos establecer variables dentro de una función a las que podamos acceder fuera de ella.

```
def pub():
    global word
    word = "La Granja"
    print("Esto está dentro de la función")

pub()
print (word)
```

```
Esto está dentro de la función
La Granja
```

2.18 Clases

TODO EN PYTHON ES UN OBJETO, y casi todo tiene atributos y métodos. Todo es un objeto en el sentido de que se puede asignar a una variable o ser pasado como argumento a una función.

¿Qué es un **atributo**? Los atributos o propiedades de los objetos son las características que puede tener un objeto: Si el objeto fuera Perro, los atributos podrían ser: tamaño, edad, color, raza, etc.

¿Qué es un **método**? Los métodos son la acción o función que realiza un objeto. Si nuestro objeto es Perro, los métodos pueden ser: caminar, ladrar, saltar, dormir, etc.

¿Qué es una **clase**? Se puede decir que una clase es una plantilla genérica de un objeto. La clase proporciona variables iniciales de estado (donde se guardan los atributos) e implementaciones de comportamiento (métodos). Podríamos decir que Perro pertenece a la clase Mascota.

¿Qué es una **instancia**? Una instancia es una copia específica de la clase con todo su contenido. Por ejemplo, Rocky sería una instancia de la clase Mascota.

Así pues, en nuestro ejemplo tendríamos: un objeto (perro), con unos atributos (raza, edad) y unos métodos (ladra, salta, corre) que pertenece a la clase (mascota). Podríamos hacer una instancia para Rocky (pastor alemán, 5 años) y otra instancia para Gala (bull terrier, 12 años).

El método **__init__** es un método especial que equivale más o menos a lo que se conoce como constructor en otros lenguajes. Su principal función es establecer un estado inicial en el objeto nada más instanciarlo, es decir, inicializar los atributos.

Y **self** es el primer parámetro que reciben los métodos de un objeto cuando lo invocas con la sintaxis: objeto.metodo(). El método recibirá como primer parámetro una referencia al objeto desde el que fue invocado y a través de él podrás acceder a atributos de ese objeto.

```
class coche:

    def __init__(self, matrícula, marca, modelo):
        self.matrícula = matrícula
        self.marca = marca
        self.modelo = modelo


v = coche("1234CBA", "Ford", "Fiesta")

print(v.matrícula)
print(v.marca)
print(v.modelo)
```

```
1234CBA
Ford
Fiesta
```

***Nota**: el ejemplo puede inducirnos a error si nos hace pensar que esto es demasiado complicado para introducir nuestro Ford Fiesta en el programa, evidentemente para un único coche no tiene mucho sentido pero si tuviesemos que instanciar cientos o miles de coches con decenas de características cada uno echaríamos en falta esta funcionalidad.*

2.19 Herencia

La **herencia** se utiliza para no tener que repetir atributos, ni métodos entre clases. Las clases que se obtienen aplicando el concepto de herencia, también son conocidas como subclases.

La **herencia simple** tiene lugar cuando una clase hija hereda los atributos y métodos de una única clase padre. Vamos a proceder a crear dos clases, una principal y una secundaria, en esta última vamos a agregar la función de presentarse accediendo a los atributos de la clase padre.

```
class Alumno(object):
    def __init__(self, edad, nombre):
        self.edad = edad
        self.nombre = nombre

class Informática (Alumno):
    def saludar(self):
       print (("Hola, soy"), (self.nombre + ","),
       ("tengo"), (self.edad), ("años"),("y estudio Informática."))

Antonio = Informática(20, "Antonio Fernández")

Antonio.saludar()
```

```
Hola, soy Antonio Fernández, tengo 20 años y estudio Informática.
```

Los casos de **herencia múltiple** en Python se dan cuando una clase secundaria o hija hereda atributos y métodos de más de una clase principal o padre. Basta con separar con una coma ambas principales a la hora de crear la clase secundaria:

```
class Alumno(object):
    def __init__(self, edad, nombre):
        self.edad = edad
        self.nombre = nombre

class Colegio(object):
    def presentar_colegio (self):
        print("Estudio en el Blas Infante.")

class Informática (Alumno, Colegio):
    def saludar(self):
       print (("Hola, soy"), (self.nombre + ","),
       ("tengo"), (self.edad), ("años"),("y estudio Informática."))

Antonio = Informática(20, "Antonio Fernández")

Antonio.saludar()
Antonio.presentar_colegio()
```

```
Hola, soy Antonio Fernández, tengo 20 años y estudio Informática.
Estudio en el Blas Infante.
```

***Nota**: la misma consideración que para las clases, esto tendría más sentido si tuvieramos muchos alumnos y asignaturas.*

2.20 Métodos para strings

Python dispone de un buen número de métodos para tratar con cadenas de texto, **dir(str)** para listar.

capitalize() retorna una copia de la cadena con la primera letra en mayúscula:

```
cadena = "bienvenido a python"
cadena.capitalize()
```

```
'Bienvenido a python'
```

lower() retorna una copia de la cadena en minúsculas:

```
cadena = "Hola Mundo"
cadena.lower()
```

```
'hola mundo'
```

upper() retorna una copia de la cadena en mayúsculas:

```
cadena.upper()
```

```
'HOLA MUNDO'
```

title() retorna una copia de la cadena con la primera letra de cada palabra en mayúscula:

```
cadena = "hola mundo"
cadena.title()
```

```
'Hola Mundo'
```

center() retorna una copia de la cadena centrada con un carácter de relleno:

```
cadena.center(50, "=")
```

```
'====================hola mundo===================='
```

ljust() retorna una copia de la cadena alineada a la izquierda con un carácter de relleno:

```
cadena.ljust(50, "=")
```

```
'hola mundo========================================'
```

rjust() retorna una copia de la cadena alineada a la derecha con un carácter de relleno:

```
cadena.rjust(50, "=")
```

```
'========================================hola mundo'
```

count() retorna un entero contando las apariciones de una subcadena dentro de la cadena:

```
cadena = "En un lugar de la Mancha"
cadena.count("a")
```

```
4
```

find() retorna un entero que representa la posición donde se inicia la subcadena dentro de cadena. Se empieza a contar desde 0, pero hay que tener en cuenta que si la subcadena no estuviese presente dentro de la cadena retornaría un -1:

```
cadena = "En un lugar de la Mancha"
cadena.find("un")
```

```
3
```

startswith() retorna un valor booleano, es decir, verdadero o falso dependiendo de si la cadena empieza con la subcadena determinada:

```
cadena = "En un lugar de la Mancha"
cadena.startswith("En")
```

```
True
```

replace() reemplaza una porción de la cadena por una subcadena:

```
cadena = "el coche es rojo"
cadena.replace("coche", "barco")
```

```
'el barco es rojo'
```

strip() elimina caracteres a la izquierda y derecha:

```
cadena = "wwwwwhola mundowwwww"
cadena.strip("w")
```

```
'hola mundo'
```

split() nos convierte un string a una lista y **join()** hace lo opuesto, convierte una lista en una cadena con los elementos separados por comas.

```
cadena = "verde, rojo, azul, amarillo"


a = cadena.split(",")

b = ','.join(a)

print(f"Lista = {a} y cadena = {b}")
```

```
Lista = ['verde', ' rojo', ' azul', ' amarillo'] y cadena = verde, rojo, a
zul, amarillo
```

splitlines() retorna una lista partiendo una cadena en líneas:

```
cadena = """Linea 1
Linea 2
Linea 3
Linea 4
"""

cadena.splitlines()
```

```
['Linea 1', 'Linea 2', 'Linea 3', 'Linea 4']
```

2.21 La función len()

La función **len()** devuelve la longitud de una cadena de caracteres o el número de elementos de una lista.

```
cadena = "verde, rojo, azul, amarillo"

lista = ['verde', ' rojo', ' azul', ' amarillo']

print (f"la cadena tiene {len(cadena)} caracteres y la lista {len(lista)} elementos")
```

```
la cadena tiene 27 caracteres y la lista 4 elementos
```

2.22 Encode y decode

En Python 3 las cadenas de caracteres pueden ser de tres tipos: Unicode, Byte y Bytearray.

El tipo Unicode permite caracteres de múltiples lenguajes y cada carácter en una cadena tendrá un valor inmutable, nosotros en concreto usamos la codificación **UTF-8**, es la que nos permite usar la Ñ y los acentos. El tipo Byte sólo permitirá caracteres ASCII y los caracteres son también inmutables. Y, finalmente, el tipo Bytearray es como el tipo Byte pero, en este caso, los caracteres de una cadena si son mutables.

Para hacernos una idea podemos decir que los textos suelen ir en Unicode mientras que los datos de una foto, por ejemplo, están codificados en Bytes.

Para declarar una cadena de texto **Unicode** basta utilizar las comillas para delimitarla:

```
lenguaje = "Python"
type(lenguaje)
```

```
str
```

Para declarar un texto **Byte** es necesario emplear las comillas y anteponer el carácter **b**:

```
lenguaje = b"Python"
type(lenguaje)
```

```
bytes
```

Sin embargo, declarar una cadena Byte en la que se incluya la ñ no es posible, en principio, porque este carácter no se encuentra en la tabla de ASCII estándar. Se producirá un error de sintaxis:

```
país = b"España"
```

```
  File "<ipython-input-39-c985e3dd6f7a>", line 1
    país = b"España"
          ^
SyntaxError: bytes can only contain ASCII literal characters.
```

Podemos convertir una cadena Unicode en Byte utilizando la función **bytes()** e indicando la codificación:

```
país = bytes("España", "UTF-8")
print(país)
```

```
b'Espa\xc3\xb1a'
```

Otra forma es mediante la función **encode()** pasándole la codificación como argumento. Al ser UTF-8 la codificación predeterminada no es necesario indicarlo, pero deberemos hacerlo para otras codificaciones.

```
país = "España"
print (país.encode()) # es lo mismo que encode("utf-8")
```

```
b'Espa\xc3\xb1a'
```

Para convertir una cadena Byte a Unicode utilizaremos la función **decode()**. Para decodificar es imprescindible especificar la codificación adecuada.

```
país = b'Espa\xc3\xb1a'
print (país.decode("utf-8"))
```

```
España
```

Por defecto, los archivos fuente de Python son tratados como codificados en UTF-8. Para especificar una codificación distinta un línea de comentario especial debe ser agregada como la primera línea del archivo. La sintaxis es: **# -*- coding: encoding -*-**

Por ejemplo, para el encoding Windows-1252 **# -*- coding: cp-1252 -*-**

Para **UNIX/LINUX**, como el código empieza con un “shebang line”, la declaración del encoding debe ser agregada como la segunda línea del archivo. Por ejemplo:

#!/usr/bin/env python3
 # -*- coding: cp-1252 -*-

2.23 Lectura y escritura

Python puede leer y escribir archivos usando la función predeterminada **open()** para obtener un objeto **file**. Los archivos en Python están categorizados en archivos de texto o archivos binarios, la diferencia entre estos dos tipos de archivos es de vital importancia al momento de manipularlos.

Un **archivo de texto** está formado por una secuencia de líneas, donde cada línea incluye una secuencia de carácteres. Esto es lo que se conoce como código o sintaxis. Un **archivo binario** es cualquier tipo de archivo que no es un archivo de texto, por ejemplo una foto.

Para leer o escribir un archivo siempre haremos este proceso: **ABRIR ARCHIVO -> LECTURA O ESCRITURA -> CERRAR ARCHIVO**

Para **ABRIR ARCHIVO** usaremos la siguiente sintaxis:

archivo = open(nombre_archivo, modo, buff, encoding="UTF-8")

Algunos de los parámetros:

nombre_archivo: este argumento es un objeto string que contiene el nombre del archivo al que queremos acceder. Si el archivo que queremos abrir no está en el mismo directorio que nuestro .py deberemos especificar la ruta (path) de acceso, teniendo en cuenta que a diferencia de Windows las rutas en Python se escriben con doble backslash \\ y no con backslash simple \.

Ruta (path) al escritorio en Windows -> "C:\Users\Nombre_Usuario\Desktop"

Ruta (path) al escritorio en Python -> "C:\\Users\\Nombre_Usuario\\Desktop"

modo: el modo de acceso determina el modo en el que el archivo ha sido abierto, por ejemplo, leer, escribir, anexar etc. Una lista completa de todos los valores posibles se especifica más adelante.

buff: si el valor de buff(buffering) es 0 no se realiza ningún buffering. Si el valor es 1, se ejecuta un buffer de línea.

encoding="UTF-8": establecer la codificación nos evitará problemas con la ñ y los acentos.

Algunos de los modos más usados con los que un archivo puede ser abierto en Python:

r -> abre un archivo solo para lectura. El puntero al archivo está localizado al comienzo del archivo. Este es el modo predeterminado de la función.

rb -> abre un archivo solo para lectura en formato binario. El puntero al archivo está localizado al comienzo del archivo. Este es el modo predeterminado de la función.

r+ -> abre un archivo para escritura y lectura. El puntero del archivo está localizado al comienzo del archivo.

w -> abre un archivo solo para escritura. Sobreescribe el archivo si este ya existe. Si el archivo no existe, crea un nuevo archivo para escritura.

wb -> abre un archivo solo para escritura en formato binario. Sobreescribe el archivo si este ya existe. Si el archivo no existe, crea un nuevo archivo para escritura.

w+ -> abre un archivo para escritura y lectura. Sobreescribe el archivo si este ya existe. Si el archivo no existe, crea un nuevo archivo para escritura.

a -> abre un archivo para anexo. El puntero del archivo esta al final del archivo si este existe. Es decir, el archivo está en modo anexo. Si el archivo no existe, crea un nuevo archivo para escritura.

Una vez que un archivo está abierto y tienes un objeto file puedes obtener mucha información relacionada con ese archivo.

Veamos una lista de los atributos relacionados con el objeto file:

nombre_archivo.**closed** : retorna True si el archivo está cerrado, si no, False.

nombre_archivo.**mode** : retorna el modo de acceso con el que el archivo ha sido abierto.

nombre_archivo.**name** : retorna el nombre del archivo.

LECTURA:

En la práctica existen muchos modos de leer un archivo en Python, no solo uno.

El método nombre_archivo.**read()** lee un string con todos los caracteres del archivo abierto.

Otra manera de leer un archivo es llamar a cierto número de carácteres. Por ejemplo, con el siguiente código el intérprete leerá los primeros cinco caracteres y los retornará como un string: **read(5)**

Si queremos leer un archivo línea por línea (en lugar de sacar todo el contenido del archivo de una sola vez) entonces podemos usar la función **readline()** Existen muchas razones por las cuales pudiéramos querer solo una línea de un archivo de texto, tal vez solo necesitamos la primera o la última línea o una a mitad del archivo.

Pero ¿y si quisiéramos todas las líneas que contiene el archivo pero separadas? Podemos usar la misma función solo que en una nueva forma. Esta es la función **readlines()**

Cuando queremos leer o retornar todas las línaes de un archivo en una forma más eficiente en cuestiones de memoria y rapidez podemos utilizar el **método de iterar** sobre un archivo. La ventaja de usar este método es que el código que se utiliza es simple y fácil de leer:

```
f = open('archivo_de_prueba.txt', 'r')
for line in f:
    print(line)
```

ESCRITURA:

Una de las cosas que se harán obvias a medida que vayamos trabajando con archivos en Python, es que el método **write()** solo requiere un parámetro, el cual es el string que queremos que sea escrito en el archivo.

También podremos utilizar **writelines()** si usamos el método de iterar.

Para **CERRAR ARCHIVO**:

Cuando hayamos terminado de trabajar con un archivo en Python debemos usar el método nombre_archivo.**close()** para cerrar el archivo, así liberamos recursos y memoria.

Es importante tener en cuenta que luego de que se llame al método nombre_archivo.close(), cualquier otro intento de usar el archivo resultará en un error.

Ejemplos de lectura y escritura:

```
## Lee el archivo.txt

fichero = "C:\\Users\\archivo.txt"
lectura = open(fichero, "r", encoding="UTF-8")
datos = lectura.read()
lectura.close()
print(datos)
```

```
En un lugar de la Mancha, de cuyo nombre no quiero acordarme, no ha mucho
tiempo que vivía un hidalgo de los de lanza en astillero, adarga antigua,
rocín flaco y galgo corredor. Una olla de algo más vaca que carnero, salpi
cón las más noches, duelos y quebrantos los sábados, lentejas los viernes,
algún palomino de añadidura los domingos, consumían las tres partes de su
hacienda.
```

```
# Escribe los datos en archivo2.txt

fichero = "archivo2.txt"
escritura = open(fichero, "w", encoding="UTF-8")
datos = "Hola Mundo"
escritura.writelines(datos)
escritura.close()
```

***Nota**: igual se puede escribir en html o cualquier otro formato simplemente cambiando la extensión.

Volver al índice

3. GRUPOS DE DATOS

En Python existen 4 tipos de estructuras de datos grupales:

- **Lista** -> [1, "item2", 3] Secuencia ordenada de ítems. Una lista de listas -> **matriz o array** -> [[1,2],[3,4]]
- **Tupla** -> (1, "item2", 3) Igual que la lista pero **inmutable**, una vez creada no se puede cambiar.
- **Diccionario** -> {'clave' : 'valor', 'clave2': [1,"item2",3]} Son parejas **clave:valor** con claves únicas.
- **Set** -> {True, "Hola mundo", (1, 2)} **Conjunto** no ordenado de objetos irrepetibles.

3.1 Listas

Las listas son secuencias ordenadas de elementos delimitadas por corchetes [ ], los ítems se separan por comas. Pueden contener números, strings, booleanos, variables, etc.

Podemos definir una lista vacía y cambiarla después:

```
lista = []
nombre = "Alberto"
edad = 30
lista = [nombre, edad]
lista
```

```
['Alberto', 30]
```

Si modificamos una variable después de declarar la lista nos devolverá el valor anterior:

```
nombre = "Andrés"
lista
```

```
['Alberto', 30]
```

Pero si se trata de objetos mutables y al modificar la variable se modifica el objeto, el resto de variables sí resultan afectadas:

```
nombres = ["Miguel", "Salvador"]
edades = [40, 41]
lista = [nombres, edades]
lista
```

```
[['Miguel', 'Salvador'], [40, 41]]
```

```
# añadimos Antonio a la lista de nombres
nombres += ["Antonio"]
lista
```

```
[['Miguel', 'Salvador', 'Antonio'], [40, 41]]
```

Las listas incluso pueden contener otras listas y tener muchos niveles de anidamiento.

```
# Ejemplo de lista anidada -> [músico, [disco, año]] -> el segundo elemento es también
una lista

discos = [
    ["Paco de Lucía", ["Zyryab", 1990]],
    ["Jimi Hendix", ["Axis: Bold as Love", 1967]],
    ["Hiromi Uehara", ["Voice", 2011]]
        ]
discos
```

```
[['Paco de Lucía', ['Zyryab', 1990]],
 ['Jimi Hendix', ['Axis: Bold as Love', 1967]],
 ['Hiromi Uehara', ['Voice', 2011]]]
```

```
# nº de elementos que componen la lista
len(discos)
```

```
3
```

Accedemos a los ítems mediante el índice entre corchetes []. El primer ítem de la lista es el 0 y el último elemento tiene índice -1, los elementos anteriores tienen valores negativos descendentes.

```
# el primer elemento de la lista será:
discos[0]
```

```
['Paco de Lucía', ['Zyryab', 1990]]
```

```
# al contener 3 elementos si contamos desde el 0 el último será [2] o [-1]
discos[-1]
```

```
['Hiromi Uehara', ['Voice', 2011]]
```

```
# si queremos el segundo elemento de la primera sublista (año del disco de Hiromi)
discos[0][1][1]
```

```
1990
```

Se pueden extraer sublistas: **nombre_lista [inicio:final]**

```
números = [0,1,2,3,4,5,6,7,8,9]

números[0:6]
# Nos devuelve del primero al 5º
```

```
[0, 1, 2, 3, 4, 5]
```

Podemos concatenar listas sumándolas:

```
números + nombres
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 'Miguel', 'Salvador', 'Antonio']
```

Se puede modificar cualquier elemento de una lista haciendo referencia a su posición:

```
números[3] = "elefante"
números
```

```
[0, 1, 2, 'elefante', 4, 5, 6, 7, 8, 9]
```

La palabra reservada **del** permite eliminar un elemento o varios elementos a la vez de una lista, e incluso la misma lista.

```
# borramos elefante de la lista
del números[3]
números
```

```
[0, 1, 2, 4, 5, 6, 7, 8, 9]
```

También podemos **iterar** los elementos de una lista:

```
for n in [1, 2, 3, 4]:
    print(n)
```

```
1
2
3
4
```

Incluso podemos **preguntar** si un elemento está en la lista:

```
if 5 in números:
    print("Está en la lista")
```

```
Está en la lista
```

O preguntar si no está:

```
if 3 not in lista:
    print("No está en la lista")
```

```
No está en la lista
```

****Nota**: para trabajar con **matrices o arrays** Python cuenta con una librería llamada NumPy.*

3.2 Métodos para listas

append() añade un ítem al final de la lista:

```
lista = [1,2,3,4,5]
lista.append(6)

lista
```

```
[1, 2, 3, 4, 5, 6]
```

clear() vacía todos los ítems de una lista:

```
lista.clear()
lista
```

```
[]
```

extend() une una lista a otra:

```
a1 = [1,2,3]
a2 = [4,5,6]
a1.extend(a2)

a1
```

```
[1, 2, 3, 4, 5, 6]
```

count() cuenta el número de veces que aparece un ítem:

```
h = ["Hola", "mundo", "mundo"]

h.count("mundo")
```

```
2
```

index() retorna el índice de un ítem (error si no aparece y si está repetido nos da el primero):

```
h.index("mundo")
```

```
1
```

insert() agrega un ítem a la lista en un índice específico, primera posición 0:

```
r = [5,10,15,25]
r.insert(-1,20)

r
```

```
[5, 10, 15, 20, 25]
```

pop() borra un ítem de la lista indicando el índice:

```
r.pop(2)
r
```

```
[5, 10, 20, 25]
```

remove() borra el primer ítem de la lista cuyo valor concuerde con el que indicamos:

```
r = [20,30,30,30,40]
r.remove(30)
r
```

```
[20, 30, 30, 40]
```

reverse() da la vuelta a la lista:

```
r.reverse()
r
```

```
[40, 30, 30, 20]
```

sort() ordena automáticamente los ítems de una lista por su valor de menor a mayor:

```
s = [5,-10,35,0,-65,100]
s.sort()
s
```

```
[-65, -10, 0, 5, 35, 100]
```

Podemos utilizar el argumento **reverse=True** para indicar que la ordene del revés:

```
s.sort(reverse=True)
s
```

```
[100, 35, 5, 0, -10, -65]
```

La función **join()** convierte una lista en una cadena formada por los elementos de la lista separados por comas. Y **split()** convierte una cadena de texto en una lista.

```
# join()
estilos = [ 'Jazz', 'Blues', 'Bossa', 'Rock']
estilos_string = ','.join(estilos)
estilos_string
```

```
'Jazz,Blues,Bossa,Rock'
```

```
# split()
string = 'Aurora es un incordio perpetuo ;-)'
lista = string.split()
lista
```

```
['Aurora', 'es', 'un', 'incordio', 'perpetuo', ';-)']
```

3.3 El tipo range()

El tipo **range()** es una lista inmutable de números enteros en sucesión aritmética.

**Nota: en Python 2, range() se consideraba una función, pero en Python 3 no se considera una función, sino un tipo de dato, aunque se utiliza como si fuera una función.*

El tipo range() con un único argumento se escribe range(n) y crea una lista inmutable de n números enteros consecutivos que empieza en 0 y acaba en n - 1. Para ver los valores del range(), es necesario convertirlo a lista mediante la función list().

```
n = range(10)
n
```

```
range(0, 10)
```

```
list (n)
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

El tipo range con dos argumentos se escribe range(m, n) y crea una lista inmutable de enteros consecutivos que empieza en m y acaba en n - 1.

```
list(range(4, 9))
```

```
[4, 5, 6, 7, 8]
```

```
list(range(-6, 2))
```

```
[-6, -5, -4, -3, -2, -1, 0, 1]
```

El tipo range con tres argumentos se escribe **range (m, n, p)** y crea una lista inmutable de enteros que empieza en m y acaba en n - 1, aumentando los valores de p en p. Si p es negativo los valores van disminuyendo de p en p. Evidentemente p no puede ser 0.

En resumen, los tres argumentos del tipo range(m, n, p) son:

m: el valor inicial

n: el valor final (que no se alcanza nunca, n - 1)

p: el paso (la cantidad que se avanza cada vez)

```
list(range(6, 25, 4))
```

```
[6, 10, 14, 18, 22]
```

```
list(range(15, 0, -3))
```

```
[15, 12, 9, 6, 3]
```

3.4 Comprensión de listas

La comprensión de listas en Python es un método sintáctico para crear listas (y por extensión también otras colecciones que veremos más abajo) a partir de los elementos de otras listas (o colecciones) de una forma rápida de escribir, muy legible y funcionalmente eficiente.

Hay dos maneras de definir los elementos que pertenecen a un conjunto: por extensión o por comprensión. Cuando se define un conjunto por extensión cada elemento se enumera de manera explícita. Por otro lado, cuando un conjunto se define por comprensión no se mencionan los elementos uno por uno sino que se indica una propiedad que todos estos cumplen.

```
# lista del 1 al 10

lista1 = [numero for numero in range (1,11)]

lista1
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

```
# lista del 1 al 50 todos los números divisibles por 5

divisibles5 = [numero for numero in range (1,51) if numero % 5 == 0]

divisibles5
```

```
[5, 10, 15, 20, 25, 30, 35, 40, 45, 50]
```

```
# números primos del 1 al 50

primos = [x for x in range(2,51) if(all(x % j for j in range(2, x)))]

primos
```

```
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47]
```

```
# lista los 15 primeros números de la sucesión de Fibonacci

fibo=[]
fibo.append(1)
fibo.append(1)
[fibo.append(fibo[k-1]+fibo[k-2]) for k in range(2,15)]

fibo
```

```
[1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610]
```

Volver al índice

3.5 Tuplas

Una tupla es un conjunto ordenado e inmutable de elementos del mismo o diferente tipo. Las tuplas se representan escribiendo los elementos entre paréntesis y separados por comas.

```
semáforo = ("verde", "rojo", "amarillo")
semáforo
```

```
('verde', 'rojo', 'amarillo')
```

Los métodos de las tuplas son muy similares a las listas y comparten varias de sus funciones pero como no pueden modificarse de ningún modo después de su creación podemos hacer muchas menos cosas con ellas.

```
# Cuántas veces aparece Alaska
valores = ("Texas", True, "Alaska", 5)
valores.count('Alaska')
```

```
1
```

```
# En qué posición está el 5 (contando desde 0)
valores.index(5)
```

```
3
```

3.6 Diccionarios

Un diccionario es una estructura de datos con características especiales que nos permite almacenar cualquier tipo de valor (value) como enteros, cadenas, listas e incluso otras funciones. Estos diccionarios nos permiten además identificar cada elemento por una clave (key), con el requerimiento de que las claves sean únicas.

Para crear un diccionario se encierra el listado de valores entre llaves, las parejas de clave y valor se separan con comas, y la clave y el valor se separan con dos puntos:

```
diccionario = {'local' : 'La Granja', 'actividades': ['skateboarding','conciertos','mer
cadillo'] }
diccionario
```

```
{'local': 'La Granja',
 'actividades': ['skateboarding', 'conciertos', 'mercadillo']}
```

Para acceder a un elemento lo podemos hacer mediante la clave:

```
diccionario['local']
```

```
'La Granja'
```

También podemos acceder a la lista que contiene el diccionario mediante el índice:

```
diccionario['actividades'][0]
```

```
'skateboarding'
```

Podemos recorrer todo el diccionario al iterar:

```
for key in diccionario:
  print (key, ":", diccionario[key])
```

```
local : La Granja
actividades : ['skateboarding', 'conciertos', 'mercadillo']
```

3.7 Métodos para diccionarios

dict() recibe como parámetro una representación de un diccionario y si es factible devuelve un diccionario de datos:

```
alumno =  dict(nombre='Antonio', apellido='Fernández', edad=19)
alumno
```

```
{'nombre': 'Antonio', 'apellido': 'Fernández', 'edad': 19}
```

zip() recibe como parámetro dos elementos iterables, ya sea una cadena, una lista o una tupla. Ambos parámetros deben tener el mismo número de elementos.

```
dic = dict(zip(['alfa','beta','gamma', 'delta'],[1,2,3,4]))

dic
```

```
{'alfa': 1, 'beta': 2, 'gamma': 3, 'delta': 4}
```

items() devuelve una lista de tuplas, cada tupla se compone de dos elementos: el primero será la clave y el segundo, su valor.

```
items = dic.items()

items
```

```
dict_items([('alfa', 1), ('beta', 2), ('gamma', 3), ('delta', 4)])
```

keys() retorna una lista de elementos, los cuales serán las claves de nuestro diccionario.

```
keys= dic.keys()

keys
```

```
dict_keys(['alfa', 'beta', 'gamma', 'delta'])
```

values() retorna una lista de elementos, que serán los valores de nuestro diccionario.

```
values= dic.values()
values
```

```
dict_values([1, 2, 3, 4])
```

clear() elimina todos los ítems del diccionario dejándolo vacío.

```
dic2 =  {'alfa': 1, 'beta': 2, 'gamma': 3, 'delta': 4}
dic2.clear()
dic2
```

```
{}
```

copy() retorna una copia del diccionario original.

```
dic1 = dic.copy()
dic1
```

```
{'alfa': 1, 'beta': 2, 'gamma': 3, 'delta': 4}
```

fromkeys() recibe un iterable y un valor y devuelve un diccionario que contiene como claves los elementos del iterable con el mismo valor ingresado. Si el valor no es ingresado devolverá none para todas las claves.

```
dic1 = dict.fromkeys(['alfa', 'beta', 'gamma', 'delta'],1)
dic1
```

```
{'alfa': 1, 'beta': 1, 'gamma': 1, 'delta': 1}
```

get() recibe como parámetro una clave y devuelve su valor. Si no lo encuentra devuelve none.

```
valor = dic.get('beta')
valor
```

```
2
```

pop() recibe como parámetro una clave, elimina esta y devuelve su valor. Si no lo encuentra, devuelve error.

```
valor = dic.pop('beta')
valor, dic
```

```
(2, {'alfa': 1, 'gamma': 3, 'delta': 4})
```

setdefault() funciona de dos formas. En la primera como get:

```
valor = dic.setdefault('alfa')

valor
```

```
1
```

Y en la segunda forma, nos sirve para agregar un nuevo elemento a nuestro diccionario.

```
dic1 = {'alfa': 1, 'beta': 2, 'gamma': 3, 'delta': 4}
valor = dic1.setdefault('epsilon',5)
dic1
```

```
{'alfa': 1, 'beta': 2, 'gamma': 3, 'delta': 4, 'epsilon': 5}
```

update() recibe como parámetro otro diccionario. Si se tienen claves iguales, actualiza el valor de la clave repetida; si no hay claves iguales, este par clave-valor es agregado al diccionario.

```
dic1 = {'alfa': 1, 'beta': 2, 'gamma': 3, 'delta': 4, 'epsilon': 5}
dic2 = {'dseta': 6, 'eta': 7, 'zeta': 8, 'iota': 9, 'kappa': 10}
dic1.update(dic2)
dic1
```

```
{'alfa': 1,
 'beta': 2,
 'gamma': 3,
 'delta': 4,
 'epsilon': 5,
 'dseta': 6,
 'eta': 7,
 'zeta': 8,
 'iota': 9,
 'kappa': 10}
```

3.8 Sets o conjuntos

Un set en Python es una estructura de datos, una colección no ordenada de objetos únicos, equivalente a los conjuntos en matemáticas. Los conjuntos son ampliamente utilizados en lógica y matemática, y desde el lenguaje podemos sacar provecho de sus propiedades (unión, intersección, diferencia, etc.) para crear código más eficiente y legible en menos tiempo.

Para crear un conjunto especificamos sus elementos entre llaves y separados por comas:

```
# Podemos generar un conjunto vacío y cambiarlo después
c = set()
c = {1, 2, 3, 4}

c
```

```
{1, 2, 3, 4}
```

Al igual que otras colecciones, sus miembros pueden ser de diversos tipos, pero no puede incluir objetos mutables como listas, diccionarios, e incluso otros conjuntos. pero sí pueden contener tuplas. Observamos como no guarda ningún orden:

```
s = {True, 3.14, None, False, "Hola mundo", (1, 2)}

s
```

```
{(1, 2), 3.14, False, 'Hola mundo', None, True}
```

También podemos obtener un conjunto a partir de cualquier objeto iterable:

```
s1 = set([1, 2, 3, 4])
s2 = set(range(10))

s1, s2
```

```
({1, 2, 3, 4}, {0, 1, 2, 3, 4, 5, 6, 7, 8, 9})
```

Un set puede ser convertido a una lista y viceversa. En este último caso, los elementos duplicados son unificados.

```
a = list({1, 2, 3, 4})
b = set([1, 2, 2, 3, 4])

a, b
```

```
([1, 2, 3, 4], {1, 2, 3, 4})
```

3.9 Métodos para conjuntos

add() añade un ítem a un conjunto, si ya existe no lo añade:

```
c = set()
c.add(1)
c.add(2)
c.add(3)
c.add(3)
c
```

```
{1, 2, 3}
```

discard() borra un ítem de un conjunto, si no está dentro del conjunto es simplemente ignorado:

```
c.discard(2)
c.discard(7)
c
```

```
{1, 3}
```

remove() opera de forma similar que discard() pero si no está dentro del conjunto lanza un KeyError:

```
c.remove(7)
```

```
---------------------------------------------------------------------------
-
KeyError                                  Traceback (most recent call las
t)
<ipython-input-10-204bd5865efd> in <module>
----> 1 c.remove(7)

KeyError: 7
```

copy() devuelve una copia de un conjunto:

```
c2 = c.copy()
c2
```

```
{1, 3}
```

clear() borra todos los ítems de un conjunto:

```
t = {1, 2, 3, 4}
t.clear()
t
```

```
set()
```

pop() retorna un elemento en forma aleatoria (no podría ser de otra manera ya que los elementos no están ordenados). Así, el siguiente bucle imprime y remueve uno por uno los miembros de un conjunto:

```
w = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
while s:
    print(w.pop())
```

```
1
2
3
4
5
6
7
8
9
10

--------------------------------------------------------------------------
-
KeyError                                  Traceback (most recent call las
t)
<ipython-input-20-2a0c921a22a8> in <module>
      1 w = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
      2 while s:
----> 3     print(w.pop())

KeyError: 'pop from an empty set'
```

remove() y pop() lanzan la excepción KeyError cuando un elemento no se encuentra en el conjunto o bien éste está vacío, respectivamente.

Para obtener el número de elementos aplicamos la ya conocida función **len()**:

```
len({1, 2, 3, 4})
```

```
4
```

3.10 Operaciones con conjuntos

<table>
<tr><th>Expresión</th><th>Significado</th><th>Ejemplo</th></tr>
<tr><td>A | B</td><td>Unión de los conjuntos A y B</td><td>{1, 2, 4, 5}.union({3, 4, 6}) = {1, 2, 3, 4, 5, 6}</td></tr>
<tr><td>A.union (B)</td><td></td><td>{1, 2, 3, 4, 5} | {3, 4, 5, 6} = {1, 2, 3, 4, 5, 6}</td></tr>
<tr><td>A |= B</td><td>Agrega B a A</td><td>{1,2,3}.update({3,4,5}) = {1, 2, 3, 4, 5}</td></tr>
<tr><td>A.update (B)</td><td></td><td></td></tr>
<tr><td>A & B</td><td>Los que comparten A y B</td><td>{1, 2, 3, 4, 5}.intersection({3, 4, 5, 6}) = {3, 4, 5}</td></tr>
<tr><td>A.intersection (B)</td><td></td><td>{1, 2, 3, 4, 5} & {3, 4, 5, 6} = {3, 4, 5}</td></tr>
<tr><td>A & = B</td><td>Elementos comunes entre A y B</td><td>{1,2,3}.intersection_update({3,4,5}) = {3}</td></tr>
<tr><td>A.intersection_update (B)</td><td></td><td></td></tr>
<tr><td>A - B</td><td>Los valores de A menos los de B</td><td>{1, 2, 3, 4}.difference({2, 3, 5}) = {1, 4}</td></tr>
<tr><td>A.difference (B)</td><td></td><td>{1, 2, 3, 4} - {2, 3, 5} = {1, 4}</td></tr>
<tr><td>A - = B</td><td>Elimina todos los elementos de B de A</td><td>{1,2,3}.difference_update({3,4,5}) = {1, 2}</td></tr>
<tr><td>A.difference_update (B)</td><td></td><td></td></tr>
<tr><td>A ^ B</td><td>No pertenecen a ambos a la vez</td><td>{1, 2, 3}.symmetric_difference({2, 3, 5}) = {1, 5}</td></tr>
<tr><td>A.symmetric_difference (B)</td><td></td><td>{1, 2, 3, 4} ^ {2, 3, 5} = {1, 4, 5}</td></tr>
<tr><td>A <= B</td><td>True si A es un subconjunto de B</td><td>{1, 2}.issubset({1, 2, 3}) = True</td></tr>
<tr><td>A.issubset (B)</td><td></td><td>{1, 2} <= {1, 2, 3} = True</td></tr>
<tr><td>A> = B</td><td>Devuelve true si B es un subconjunto de A</td><td>{1, 2, 3, 4, 5}.issuperset({1, 2, 3}) = True</td></tr>
<tr><td>A.issuperset (B)</td><td></td><td>{1, 2, 3, 4, 5} >= {1, 2, 3} = True</td></tr>
<tr><td>a.isdisjoint(b)</td><td>True si no comparten elementos entre sí</td><td>{1, 2, 3}.isdisjoint({5, 6, 7}) = True</td></tr>
<tr><td></td><td></td><td>{1, 2, 3}.isdisjoint({3, 4, 5}) = False</td></tr>
</table>

Volver al índice

4 MÓDULOS Y PAQUETES

Un módulo es cualquier archivo con la extensión .py, incluso estando vacío. También podemos encontrarnos con que un módulo forme parte de un paquete, librería o biblioteca. La diferencia entre un módulo y un paquete es que el paquete contiene un atributo path que indica la ruta donde está almacenado. Un paquete sería, por tanto, un directorio de módulos indexados. La función principal de los módulos es organizar el código. Cuando estamos escribiendo un programa el código puede hacerse muy largo, es una gran idea dividirlo en partes para estructurarlo mejor y que nos sea más fácil su comprensión.

En Python la utilización de módulos es tan habitual que es lo primero que nos encontraremos en cualquier programa. Es una convención de estilo declarar los módulos que vamos a utilizar en las primeras líneas del programa. Primero se importarán los módulos estándar, es decir, los que ya vienen integrados con la instalación original de Python, después los módulos de terceros, que son los que hemos instalado nosotros después, y por último los módulos locales, que serían los que hemos creado nosotros. Así pues, siempre seguiremos este orden:

- Módulos estándar
- Módulos de terceros
- Módulos locales

Para una mejor comprensión del funcionamiento de los módulos empezaremos por ver cómo se importan los **módulos locales**, es decir, los que hemos creado nosotros mismos, así entenderemos perfectamente cómo crearlos e importarlos en otros programas. Después veremos algunos de los módulos más utilizados de la **biblioteca estándar** y, por último, aprenderemos a instalar **módulos de terceros**. Debido a que existen miles de paquetes, librerías y frameworks realizados por la comunidad, desde los muy simples a los muy complejos, algunos necesitan un manual para ellos solos, sólo se hará una breve referencia a algunos de los más utilizados. Lo fundamental es comprender que podemos recurrir a herramientas extra para facilitarnos determinadas tareas.

4.1 Importar módulo

Primero vamos a crear un pequeño script para después poder acceder a él, lo vamos a llamar **script_1.py**:

```
saludo = "Hola, te saludo desde script_1.py"
canción = "Strawberry Fields Forever"
# guardamos con el nombre script_1.py
```

Hemos guardado 2 variables, ahora las importaremos.

***Nota**: deberemos guardarlo en el mismo directorio que el anterior para que funcione.

```
from script_1 import saludo

print(saludo)
```

```
Hola, te saludo desde script_1.py
```

Ahora veremos que si intentamos acceder a la otra variable nos indicará que no está definida porque no la hemos importado:

```
canción
```

```
---------------------------------------------------------------------------
-
NameError                                 Traceback (most recent call las
t)
<ipython-input-16-45bbfc94d305> in <module>
----> 1 canción

NameError: name 'canción' is not defined
```

Si quisieramos importar el módulo entero:

```
from script_1 import *

print(saludo,canción)
```

```
Hola, te saludo desde script_1.py Strawberry Fields Forever
```

También podemos importar funciones y otros objetos, veamos otro ejemplo, llamaremos a este script círculo.py:

```
# Script para calcular el área de un círculo

def área_círculo (radio):
    pi = 3.14159
    x = (pi * (radio**2))
    return x

# guardamos con el nombre círculo.py
```

Ahora creamos otro script en el que llamaremos a la función anterior:

```
from círculo import *

radio = float(input("Valor del radio en centímetros? "))
print (f"El área del círculo es igual a {área_círculo(radio)} centímetros cuadrados")
```

```
Valor del radio en centímetros? 2
El área del círculo es igual a 12.56636 centímetros cuadrados
```

4.2 La carpeta __pycache__

Para acelerar la carga de módulos en Python al ejecutar un programa, el intérprete compila en bytecode primero, grosso modo, y lo almacena en la carpeta __pycache__. En ocasiones nos pueden aparecer un montón de archivos que comparten los nombres de los archivos .py de la carpeta de tu proyecto, sólo que sus extensiones serán .pyc o .pyo., lo que hacen es tratar que el programa comience un poco más rápido. Cuando los scripts cambien, se volverán a compilar, y si eliminamos los archivos o toda la carpeta y ejecutamos el programa nuevamente volverán a aparecer. Puedes ignorarlos o borrarlos.

4.3 El atributo __name__

Cualquier módulo en python tiene un atributo especial llamado **__name__** que es usado para identificar de forma única el módulo en el sistema de importaciones. El intérprete pasa el valor del atributo a **__main__** si el módulo se está ejecutando como programa principal. Si el módulo no es llamado como programa principal, sino que es importado desde otro módulo, el atributo __name__ pasa a tener el nombre del archivo.

Para conseguir que la ejecución sea diferente al ejecutar el módulo directamente que al importarlo desde otro programa podemos usar **if __name__ == "__main__":**

Así si se ha ejecutado como programa principal ejecutará el código dentro del condicional, si se ha importado desde otro módulo no.

Volvamos a nuestro ejemplo anterior del área del círculo, observaremos que círculo.py al ejecutarse no devolvía nada (sólo definimos una función para ser llamada en otro script, incluso nos faltaba el input para introducir el valor del radio), estaba hecho sólo para ser importado desde otro programa. Ahora vamos a modificarlo para que sea operativo como programa principal y alternativamente podamos usarlo también importado en otro programa:

```
# círculo.py para calcular el área de un círculo

def área_círculo (radio):
    pi = 3.14159
    x = (pi * (radio**2))
    return x

if __name__ == "__main__":
    radio = float(input("Valor del radio en centímetros? "))
    print (f"El área del círculo es igual a {área_círculo(radio)} centímetros cuadrado
s")
```

```
Valor del radio en centímetros? 5
El área del círculo es igual a 78.53975 centímetros cuadrados
```

Si volvemos a importar el módulo desde otro script vemos que funciona exactamente igual:

```
from círculo import *

radio = float(input("Valor del radio en centímetros? "))
print (f"El área del círculo es igual a {área_círculo(radio)} centímetros cuadrados")
```

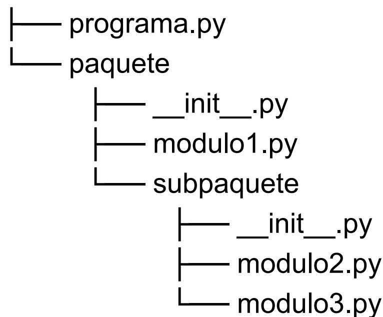
```
Valor del radio en centímetros? 3
El área del círculo es igual a 28.27431 centímetros cuadrados
```

Resumiendo, si ejecutamos círculo.py su nombre será main puesto que será nuestro programa principal, pero si lo ejecutamos importándolo desde otro programa su nombre será círculo.

4.4 Empaquetar módulos

Un paquete es un directorio que contiene archivos .py y un archivo de inicio llamado **__init__.py** que no necesita contener ninguna instrucción. Por lo general, __init__.py suele estar completamente vacío ya que sólo sirve para indicar al intérprete de Python que el directorio contiene un paquete, otros subpaquetes u otros módulos.

Un ejemplo sería la siguiente estructura:



Y aquí la forma de recurrir a ellos:

Expresión	Significado
import nombre_módulo	importa un módulo que no pertenece a un paquete
import paquete.nombre_módulo	importa un módulo que está dentro de un paquete
import paquete.subpaquete.nombre_módulo	importa un módulo que está dentro de un subpaquete

En Python, cada paquete, módulo, clase, función y método posee un **namespaces** donde los nombres de una variable se resuelven. Por ejemplo, para acceder a una variable de un módulo de un subpaquete: **paquete.subpaquete.nombre_módulo.nombre_variable**

Sin utilizar namespaces sería:

from paquete.subpaquete.nombre_módulo **import** nombre_variable1, nombre_variable2

Pero también podemos recurrir a las variables utilizando un **alias**:

import paquete.subpaquete.nombre_módulo **as manolo**
manolo.nombre_variable

Volver al índice

5. LA BIBLIOTECA ESTÁNDAR

5.1 Matemáticas - math

```
import math

a = math.pow(3, 3)    # Potencia con flotante
b = math.sqrt(25)     # Raíz cuadrada (square root)
c = math.pi           # pi
d = math.e            # e

print(a,b,c,d)
```

```
27.0 5.0 3.141592653589793 2.718281828459045
```

5.2 Sistema operativo - os

import os

- os.access(ruta, modo-acceso) # devuelve si se puede acceder a un archivo o directorio
- os.getcwd() # devuelve el directorio de trabajo
- os.chdir('/dir1/dir2') # cambia directorio trabajo
- os.chmod(path, mode) # cambia permisos a un archivo
- os.chown(path, uid, gid) # cambia propietario de un archivo
- os.chroot(path) # cambia al directorio de trabajo raíz
- os.cpu_count() # número de CPUs del sistema
- os.curdir # directorio actual
- os.pardir # directorio padre
- os.ctermid() # nombre del archivo del terminal
- os.devnull # ruta del dispositivo nulo
- os.environ.iteritems() # diccionario con variables de entorno y valor
- os.getuid() # devuelve id usuario real del proceso actual (Unix)
- os.getgid() # devuelve id grupo real del proceso actual (Unix)
- os.geteuid() # devuelve id usuario efectivo del proceso actual (Unix)
- os.getegid() # devuelve id grupo efectivo del proceso actual (Unix)
- os.getgroups() # devuelve lista de grupos suplementarios relacionados con el proceso actual (Unix)
- os.getpid() # devuelve id del proceso actual
- os.getenv(key, default=None) # obtiene valor de variable de entorno
- os.getlogin() # devuelve nombre usuario actual
- os.listdir('/home') # lista contenido de directorio
- os.mkdir(path [,mode=511]) # crea subdirectorio
- os.makedirs(path[, modo]) # crea directorios recursivamente
- os.path.abspath(path) # devuelve path absoluta de un archivo o directorio
- os.path.abspath(__file__) # devuelve ruta completa del fichero actual
- os.path.basename(path) # devuelve directorio base
- os.path.dirname(path) # devuelve directorio del archivo o directorio

- os.path.exists(path) # comprueba si existe fichero o directorio
- os.path.getatime(path) # devuelve fecha/hora del último acceso a un archivo o directorio
- os.path.getsize(path) # obtiene el tamaño de un archivo o directorio
- os.path.isabs(path) # devuelve si la ruta es absoluta
- os.path.isfile(path) # devuelve si la ruta es un archivo
- os.path.isdir(path) # devuelve si la ruta es un directorio
- os.path.islink(path) # devuelve si la ruta es un enlace simbólico
- os.path.ismount(path) # devuelve si la ruta es un punto de montaje
- os.putenv(key, value) # cambia/inserta variable de entorno
- os.remove(path) # borra un archivo
- os.removedirs(path) # elimina directorios recursivamente
- os.rename(old, new) # renombrar un archivo o directorio
- os.renames(old, new) # Renombrado recursivo.
- os.rmdir(path) # borrar un subdirectorio
- os.link(src, dst) # crea enlace duro
- os.symlink(path, nombre-destino) # crea enlace simbólico
- os.readlink(path) # cadena que representa ruta a la que apunta enlace simbólico
- os.stat(path) # Estado de archivo o de descriptor de archivo.
- os.sep # separador utilizado en una ruta (path)
- os.extsep # separador de extensión
- os.linesep # separador de líneas
- os.pathsep # separador usado para expresar varias rutas
- os.system('ls') # ejecuta un comando externo
- os.uname() # muestra Información del sistema. Tiene 5 atributos
- os.uname().sysname # muestra nombre del sistema. Otros atributos: nodename, release, version y machine
- os.unsetenv(key) # borra una variable de entorno
- os.urandom(n) # genera cadenas aleatorias de n bytes
- os.wait() # espera fin de un proceso hijo y devuelve tupla con estado pid y salida (unix)

```
# devuelve id del proceso actual
# y el número de CPUs del sistema

import os

os.getpid(), os.cpu_count()
```

```
(5064, 4)
```

```
# lista recursivamente los archivos del directorio en el que se ejecuta

import os

path=(os.getcwd().replace("/","\\"))

for dirName, subdirList, fileList in os.walk(path):
    print('[+] %s' % dirName.replace("\\","/"))
    for fname in fileList:
        print('\t%s' % fname)
```

5.3 Funciones del intérprete - sys

import sys

- sys.argv # devuelve la lista formada por programa y lista de argumentos agregados al ejecutar
- sys.executable # devuelve ruta del ejecutable del intérprete
- sys.exit() # fuerza salida del intérprete Python
- sys.getdefaultencoding() # devuelve codificación de caracteres por defecto
- sys.getfilesystemencoding() # devuelve codificación de caracteres que se utiliza para convertir los nombres de archivos unicode en nombres de archivos del sistema
- sys.path # devuelve paths de Python
- sys.path.append('ruta') # añade una nueva ruta al path
- sys.modules # muestra información de los módulos
- sys.version # obtiene versión de Python
- sys.copyright # obtiene información de copyright
- sys.platform # obtiene sistema operativo del sistema
- sys.version_info # obtiene información de versión

```
# devuelve la codificación por defecto
import sys

sys.getdefaultencoding()
```

```
'utf-8'
```

5.4 Comandos cmd - subprocess

```
import subprocess

# Comando que se ejecuta en la consola de comandos o cmd
# sustituir ipconfig por cualquier comando

comando = ('ipconfig')

m = subprocess.check_output(comando)

m = str(m)

# Eliminar caracteres especiales

datos = m.replace("\\xa0", "á").replace("\\x82", "é").replace("\\xa1", "í").replace
("\\xa2", "ó").replace("\\xa3", "ú").replace("\\xa4", "ñ").replace("\\xa5", "Ñ").replac
e("\\xb5", "Á").replace("\\x90", "É").replace("\\xd6", "Í").replace("\\xe0", "Ó").repla
ce("\\xe9", "Ú").replace("\\n", "\n").replace("\\r", "\r").replace("\\b", "\b")

# guardará los datos en archivo info.txt

nombre_fichero = ("info.txt")
fichero = open(nombre_fichero, "w")
fichero.writelines(datos[2 : -1])
fichero.close()
```

5.5 Números aleatorios - random

```
#Programa para adivinar un número aleatorio entre 0 y 100

#Importamos la librería random (número aleatorio)
import random

#Definimos la función adivinar_número
def adivinar_número():

    #Imprimimos el título + un salto de línea
    print("PROGRAMA PARA ADIVINAR UN NÚMERO ALEATORIO ENTRE 0 y 100" + "\n")

    #Asignamos a la variable número un entero al azar entre 0 y 100
    número = random.randint(0, 100)

    #Establecemos un contador = 0
    contador = 0

    #Mientras respondamos a la pregunta se ejecutará el siguiente ciclo
    while True:
        #Preguntamos por el número
        x = input("Introduzca un número: ")

        #Mientras introduzcamos un número
        try:
            x = int(x)

            #Si el número elegido es igual al de la máquina
            if x == número:
                print("Enhorabuena, acertaste en " + str(contador + 1) + " intentos")
                return 0

            #Si el número es menor
            elif x < número:
                print("El número que buscas es mayor.")
                contador = contador + 1

            #Si el número es mayor
            elif x > número:
                print("El número que buscas es menor.")
                contador = contador + 1

        #Si el valor introducido no es un número entero
        except ValueError:
            print("No has introducido un número válido")
            contador = contador + 1

#Ejecutamos la función adivinar_número
adivinar_número()
```

Volver al índice

5.6 Fecha y hora

```
from datetime import datetime
import locale

def dt():
    locale.setlocale(locale.LC_ALL, 'es')
    time = datetime.now()
    fecha = (time.strftime('%A %d/%m/%Y - %H:%M:%S'))
    return fecha.capitalize()

dt()
```

```
'Domingo 01/09/2019 - 19:27:44'
```

5.7 Tiempo - time

```
import time

start = time.time()
# proceso a cronometrar
# esperamos 3 segundos
time.sleep(3)
# hasta aquí cronometra

finish = (round(time.time() - start ,3))
print ("Programa ejecutado en " + str(finish) + " segundos.")
```

```
Programa ejecutado en 3.009 segundos.
```

5.8 Formato json

```
# leer json
import json

with open("lista.json") as f:
    lista = json.load(f)
    f.close

print(lista)
```

```
{'nombre': 'Mike', 'apellido1': 'Brown', 'apellido2': None, 'edad': 36, 'c
asado': True, 'altura': 1.75, 'cursos': ['Python', 'JS', 'Arduino']}
```

```
# escribir json
import json

lista2 = {'nombre': 'Mike', 'apellido1': 'Brown', 'apellido2': None, 'edad': 36,
          'casado': True, 'altura': 1.75, 'cursos': ['Python', 'JS', 'Arduino']}

with open('lista2.json', 'w') as fp:
    json.dump( lista2, fp , sort_keys=True, ensure_ascii=False)
    fp.close
```

5.9 Interfaz gráfica - tkinter

```
# CREACIÓN DE GRÁFICOS CON TKINTER
# Cada vez que pulsemos el botón nos saludará

from tkinter import*

# Crea nuestra primera ventana Tk()
tk = Tk()
def saludo():
    print('Hola a todos')

# creamos un botón en nuestra ventana
btn = Button(tk, text="púlsame", command=saludo)
# nos muestra el botón
btn.pack()
```

5.10 Internet - urllib

Un sencillo ejemplo: conectarnos a la web de python y descargar el logo.

```
# script de ejemplo para descargar una foto: el logo de python

import urllib.request

url = "https://www.python.org/static/img/python-logo.png"

# Por protocolo siempre que accedemos a un servidor hacemos una solicitud.
# Muchos sitios no aceptan peticiones mediante navegadores "no comunes",
# así que pondremos en nuestro encabezado que usamos, por ejemplo, un navegador
# Mozilla y un S.O. Linux, así evitaremos que nos baneen.

headers = {}
headers['User-Agent'] = 'Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, lik
e Gecko) Chrome/51.0.2704.103 Safari/537.36'

# Hacemos nuestra solicitud y tendremos una respuesta

solicitud = urllib.request.Request(url, headers=headers)
respuesta = urllib.request.urlopen(solicitud)

# Ahora escribimos los bytes de la foto en un archivo png

with open("foto.png", 'wb') as f:
        f.write(respuesta.read())
        f.close()
        print("Descarga finalizada!!")

# Ahora tendremos la foto en el mismo directorio que ejecutamos el script
```

```
Descarga finalizada!!
```

Ahora veremos un ejemplo de **web scraping**: ver el saldo de cualquier cuenta **bitcoin**.

```
# web scraping: script para ver el total recibido y el saldo de una cuenta bitcoin

import urllib.request

cuenta = input("Introduce el nº de cuenta Bitcoin: ")

if len(cuenta) != (34):
    print("Nº de cuenta no válido, el número de dígitos es incorrecto")

else:
    try:

        url = 'https://www.blockchain.com/btc/address/' + cuenta
        headers = {}
        headers['User-Agent'] = '''Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/51.0.2704.103 Safari/537.36'''

        solicitud = urllib.request.Request(url, headers=headers)
        respuesta = urllib.request.urlopen(solicitud)
        contador = 0
        extracto = []

        for línea in respuesta:
            a = str(línea)
            # buscamos el total recibido y el saldo en el html
            # (se muestran con la etiqueta <span data-c )
            if "<span data-c" in a:
                # eliminamos caracteres no deseados
                b = a.find("span data-c")
                c = a.find("BTC")
                d = a[b:c]
                e = d.find(">")
                resultado = d[e+1:-1]
                extracto.append(resultado)
                contador += 1

            if "Checksum does not validate" in a:
                print("Esa dirección no ha sido encontrada")

            if "<div" in a and contador == 2 :
                break

    except urllib.error.URLError as e:
        page=('Error code: ', e.code)
        print("No existe esa cuenta")


print(f"La cuenta {cuenta} ha recibido un total de {extracto[0]} BTC y su saldo actual
es de {extracto[1]} BTC.")
```

```
Introduce el nº de cuenta Bitcoin: 3FkenCiXpSLqD8L79intRNXUgjRoH9sjXa
La cuenta 3FkenCiXpSLqD8L79intRNXUgjRoH9sjXa ha recibido un total de 34.73
9066 BTC y su saldo actual es de 0.33589861 BTC.
```

5.11 Subir archivos vía ftp - ftplib

```
# Script para subir archivos por ftp a un servidor

import ftplib
import posixpath

nombre_fichero = "ejemplo.html"

def extraer_información():
    archi=open('ejemplo.html','r')
    linea=archi.readline()
    while linea!="":
        print (linea)
        linea=archi.readline()
        archi.close()
        return nombre_fichero

def hacer_conexión(servidor, user, password,carpeta, fichero, fichero_destino):
    try:
        s = ftplib.FTP(servidor, user, password)
        try:
            f = open(fichero, 'rb')
            s.cwd(carpeta)
            s.storbinary('STOR ' + fichero_destino, f)
            f.close()
            s.quit()
            return ("Fichero subido correctamente")
        except:
            return ("No fue posible subir el archivo")
    except:
        return("No es posible conectar al servidor")

def probar_conexión(servidor, user, password):
    try:
        s = ftplib.FTP(servidor, user, password)
        print ("Conexión satisfactoria")
    except:
        return("No es posible conectar al servidor")

## aquí ponemos los datos ftp de nuestro servidor
servidor_ftp = "______"
servidor_usuario = "______"
servidor_password = "______"
servidor_carpeta = "/public_html/"

fichero = extraer_información()
fichero_destino = posixpath.join(servidor_carpeta, fichero)
hacer_conexión (servidor_ftp, servidor_usuario, servidor_password, servidor_carpeta, fi
chero, fichero_destino)
```

Volver al índice

5.12 Socket

Dos programas, posiblemente situados en computadoras distintas, pueden intercambiar cualquier flujo de datos. Los sockets constituyen el mecanismo para la entrega de esos paquetes de datos provenientes de la tarjeta de red a los procesos o hilos apropiados, mediante protocolos de Internet TCP/IP.

Veamos un ejemplo muy simple, un **chat** entre un **servidor** y un **cliente**:

```
# Simple servidor para chat
import socket

def servidor():
    # Aquí ponemos nuestra ip o dominio
    # o localhost si queremos sólo para nuestra red
    host = "______"
    # Aquí un puerto que debe estar abierto en el router
    # si queremos entrar desde fuera de nuestra red
    port = 8088
    mySocket = socket.socket()
    mySocket.bind((host,port))
    mySocket.listen(5)
    conn, addr = mySocket.accept()
    print ("Conexión realizada con: " + str(addr))
    while True:
            data = conn.recv(1024).decode()
            if not data:
                    break
            print ("Cliente: " + str(data))
            data = str(data)
            ans = input(" -> ")
            print ("Espera...")
            conn.send(ans.encode())
    conn.close()
servidor()
```

```
# cliente chat
import socket

print("Introduce tu nombre y pulsa ENTER")
def cliente():
        host = '______' # el mismo dominio que para el servidor
        port = 8088   # el mismo puerto que para el servidor
        mySocket = socket.socket()
        mySocket.connect((host,port))
        message = input(" -> ")
        print ('Espera...')
        while message != 'q':
                mySocket.send(message.encode())
                data = mySocket.recv(1024).decode()

                print ('Servidor: ' + data)

                message = input(" -> ")
                print ('Espera...')
        mySocket.close()
cliente()
```

También podemos utilizar un socket para hacer un sencillo **escáner de puertos**:

```
import socket

sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
host = input ('Host? ')
puerto = int(input ('Puerto? '))

def scanner(puerto):
    if sock.connect_ex((host,puerto)):
        print (f'El puerto {puerto} está cerrado para {host}')
    else:
        print (f'El puerto {puerto} está abierto para {host}')

scanner(puerto)
```

```
Host? google.es
Puerto? 443
El puerto 443 está abierto para google.es
```

5.13 Multihilos - threading

En ocasiones podemos necesitar que un programa se bifurque para hacer dos o más tareas diferentes simultáneamente. En Python no existe el paralelismo propiamente dicho, pero no vamos a entrar en detalle a hablar sobre procesos, hilos, concurrencia o asincronismo, son conceptos muy complejos. Aquí lo que vamos a tratar es cómo podemos ejecutar 2 tareas a la vez, o casi a la vez.

Para hacernos una idea pensemos que normalmente vamos ejecutando línea por línea nuestro código pero ¿qué pasa si una tarea se demora demasiado? Pues que nos quedamos esperando y hasta que no termine no pasamos a la siguiente. Esto lo podemos resolver con los hilos, es decir, podemos crear dos subprocesos diferentes.

```
from threading import Thread

def cuenta_hasta_100_millones():
    a = range(0,100000001)
    for x in a:
        if x == 100000000:
            print("Hilo 1: he terminado de contar 100 millones.")
            break

hilo_1 = Thread(target = cuenta_hasta_100_millones)
hilo_2 = Thread(target = print("Hilo 2: Hola"))
hilo_1.start()
hilo_2.start()
```

```
Hilo 2: Hola
Hilo 1: he terminado de contar 100 millones.
```

Si hubieramos ejecutado esta secuencia sin hilos primero habría contado hasta 100 millones, esto apenas tarda un par de segundos, y después habría imprimido el hola. Con los hilos conseguimos que se ejecuten a la vez, o casi a la vez, y como tarda más en contar primero imprime el hola y después dice que ya ha terminado de contar.

5.14 Gestor de paquetes - pip

Desde Python 3.4 el gestor de paquetes **pip** ya viene integrado con la instalación original, es una herramienta muy útil que nos facilitará enormemente la instalación y desinstalación de librerías de terceros. Es superfácil de usar, tan sólo tendremos que escribir pip en la consola de comandos, instalar o desinstalar, y el nombre del paquete, automáticamente nos descargará e instalará la librería o nos la desinstalará si así se lo indicamos.

En la consola de comandos (cmd):

- **pip freeze**: nos lista todos los paquetes que tenemos instalados
- **pip install** nombre_paquete: nos descarga e instala el paquete
- **pip uninstall** nombre_paquete: nos desinstala el paquete

Si quisieramos hacer una desinstalación completa de todos los paquetes que tenemos, ojo, esto también nos desinstalaría pip, pero puede servirnos para eliminar todo si vamos a, por ejemplo, instalar Python de nuevo, podemos hacerlo con 2 simples líneas en la consola:

pip freeze > mis_paquetes.txt
pip uninstall -r mis_paquetes.txt -y

6. LIBRERÍAS DE TERCEROS

6.1 Entorno virtual - virtualenv

Cuando trabajamos con diferentes proyectos es posible crear un entorno distinto para cada unos de ellos, es como tenerlos aislados, cada uno con sus propias librerías. Para conseguir tal propósito desde la consola instalaremos el paquete **virtualenv**:

- **pip install virtualenv**
  - Creamos nuestro entorno virtual, también desde la consola:
  - mkdir mi_entorno -> creamos una carpeta
  - cd mi_entorno -> entramos en la carpeta
    - **virtualenv prueba1** -> creamos nuestro entorno virtual (tarda unos segundos)
    - cd prueba1 -> entramos en la carpeta prueba1
    - cd Scripts -> entramos en Scripts
      - **activate.bat** -> activamos nuestro entorno
        - Ahora ya podemos instalar aquí cualquier librería
        - **python** -> o podemos entrar en python, quit() para salir
        - **deactivate** -> desactivamos nuestro entorno

Así podemos trabajar en un entorno aislado, sin miedo a cargarnos nada, o trabajar con versiones diferentes del intérprete, o con versiones diferentes de una misma librería, etc. Si nos hemos hartado del proyecto pues lo borramos sin más, no influye para nada en nuestra instalación original.

6.2 Ejecutables .exe - pyinstaller

Podemos crear un ejecutable .exe para correr nuestro programa en otro windows sin python instalado. Para ello desde la consola instalaremos el paquete **pyinstaller:**

- **pip install pyinstaller**
  - Nos situamos en el directorio de nuestro programa.py
    - pyinstaller programa.py

Así de sencillo creamos nuestro ejecutable, miraremos en una carpeta que se nos crea llamada dist y allí estará. Hay que tener en cuenta las dependencias, si sólo trabajamos con la biblioteca estándar o con librerías soportadas muy comunes no tendremos problemas, pero no funcionará con algunas otras, a no ser que las incluyamos a mano, cosa bastante laboriosa. Otros parámetros que también podemos utilizar:

- --icon=imagen.ico -> para añadir un icono al ejecutable
- --onefile -> para que el ejecutable se genere en un único archivo
- --windowed -> evita que aparezca la consola, por ejemplo cuando usamos una interfaz gráfica
- pyinstaller programa.py --icon=imagen.ico --onefile --windowed

Para generar ejecutable con archivos externos, fotos, música, videos, etc.:

- pyi-makespec <nombre.py>
- datas = [('./resources/*.png','resources')],
  - y ejecutamos: pyinstaller nombre.spec

6.3 Solicitudes http - requests

```python
# instalar -> pip install requests
# script para entrar en una web mediante un proxy
# entraremos en http://httpbin.org/ip para ver nuestra ip
import requests
# proxy lista ->  https://hidemyna.me/en/proxy-list/
# en este caso he elegido un proxy de USA
http = "http://132.145.112.141:8080"
https = "https://132.145.112.141:8080"
headers = {}
headers['User-Agent'] = 'Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, lik
e Gecko) Chrome/51.0.2704.103 Safari/537.36'
proxy_dict = {
  "http": http,
  "https": https}
r = requests.get('http://httpbin.org/ip', headers=headers, proxies=proxy_dict)
r.encoding = r.apparent_encoding
print(r.text)
```

```
{
  "origin": "132.145.112.141, 132.145.112.141"
}
```

```python
# parámetros de la web de facebook
import requests

headers = {}
headers['User-Agent'] = 'Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, lik
e Gecko) Chrome/51.0.2704.103 Safari/537.36'

url = 'https://es-es.facebook.com'

r = requests.get(url, headers=headers)
r.encoding = r.apparent_encoding

# print (r.content) html en bytes, no lo muestro porque es muy largo
# print (r.text) el html, no lo muestro porque es muy largo
print (r.status_code)
print("\n")
print (r.headers) # respuesta del servidor
print("\n")
print (r.request.headers) # nuestra petición
print("\n")
print (r.cookies)
print("\n")

# cookies
session = requests.Session()
print(session.cookies.get_dict())
```

```
200


{'Cache-Control': 'private, no-cache, no-store, must-revalidate', 'Expire
s': 'Sat, 01 Jan 2000 00:00:00 GMT', 'Pragma': 'no-cache', 'Strict-Transpo
rt-Security': 'max-age=15552000; preload', 'Content-Encoding': 'gzip', 'co
ntent-security-policy': "default-src * data: blob: 'self';script-src *.fac
ebook.com *.fbcdn.net *.facebook.net *.google-analytics.com *.virtualeart
h.net *.google.com 127.0.0.1:* *.spotilocal.com:* 'unsafe-inline' 'unsafe-
eval' blob: data: 'self';style-src data: blob: 'unsafe-inline' *;connect-s
rc *.facebook.com facebook.com *.fbcdn.net *.facebook.net *.spotilocal.co
m:* wss://*.facebook.com:* https://fb.scanandcleanlocal.com:* attachment.f
bsbx.com ws://localhost:* blob: *.cdninstagram.com 'self' chrome-extensio
n://boadgeojelhgndaghljhdicfkmllpafd chrome-extension://dliochdbjfkdbacpmh
lcpmleaejidimm;", 'Vary': 'Accept-Encoding', 'X-Content-Type-Options': 'no
sniff', 'X-Frame-Options': 'DENY', 'X-XSS-Protection': '0', 'Content-Typ
e': 'text/html; charset="utf-8"', 'X-FB-Debug': '3RyyLABobqsDQaPKounMbHH/
7/vTLdPgKaqtJhoOvuNCrv85tOMXUKLratG2vPajJQKOXkiUS7etsjoNRzK1nQ==', 'Date':
'Mon, 02 Sep 2019 22:08:21 GMT', 'Transfer-Encoding': 'chunked', 'Connecti
on': 'keep-alive'}


{'User-Agent': 'Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/51.0.2704.103 Safari/537.36', 'Accept-Encoding': 'gzip,
deflate', 'Accept': '*/*', 'Connection': 'keep-alive'}


<RequestsCookieJar[]>


{}
```

6.4 Automatizar navegador - selenium

```
# script para enviar mensajes automáticos de whatsapp

# instalar selenium -> pip install selenium
# con selenium podemos logerarnos automáticamente a webs,
# hacer web scraping, crear bots, etc.

# descargar driver para nuestro navegador ->
# https://www.seleniumhq.org/download/
# para chrome -> https://sites.google.com/a/chromium.org/chromedriver/downloads

import time
from selenium import webdriver
from selenium.webdriver.chrome.options import Options
from selenium.webdriver.common.keys import Keys


chromedriver = 'chromedriver.exe'

options= Options()
options = webdriver.ChromeOptions()

# Guardar datos usuario para no escanear de nuevo
# nos crea una carpeta, después la podemos borrar
options.add_argument('--user-data-dir=./User_Data')

browser = webdriver.Chrome(executable_path=chromedriver, options=options)
browser.get('https://web.whatsapp.com/')

print('Cargando WHATSAPP - Espera 5 segundos!!!')
time.sleep(5)

# Aquí seleccionamos nuestro contacto
contacto = '______'
contact = browser.find_element_by_xpath('//*[@id="side"]/div[1]/div/label/input')
contact.send_keys(contacto, Keys.ENTER)
time.sleep(1)

# Enviar mensaje
mensaje = 'Hola, este es mensaje un mensaje automático.'
message = browser.find_element_by_xpath('//*[@id="main"]/footer/div[1]/div[2]/div/div[2]')
message.send_keys(mensaje, Keys.ENTER)

print('Mensaje enviado. El programa se cerrará en 5 segundos.')
time.sleep(5)

# Salir
browser.quit()
```

Volver al índice

6.5 MongoDB - pymongo

Python puede trabajar con una gran variedad de bases de datos, desde las relacionales como MySQL, SQLAlchemy o SQLite, esta última ya cuenta con soporte integrado en la biblioteca estándar, a las no sólo relacionales como Cassandra o MongoDB.

Las SQL trabajan con tablas, esto suponía un problema a la hora de introducir una nueva columna cuando ya teníamos creada la base de datos, por el contrario, MongoDB trabaja con bson, que no es más que la representación binaria del formato json, y nos permite una gran flexibilidad a la hora de introducir datos.

Una vez instalada la base de datos MongoDB en nuestro ordenador podemos editar la configuración en:

C:\Program Files\MongoDB\Server\4.0\bin\mongod.cfg

Y para instalar el paquete en Python:

pip install pymongo

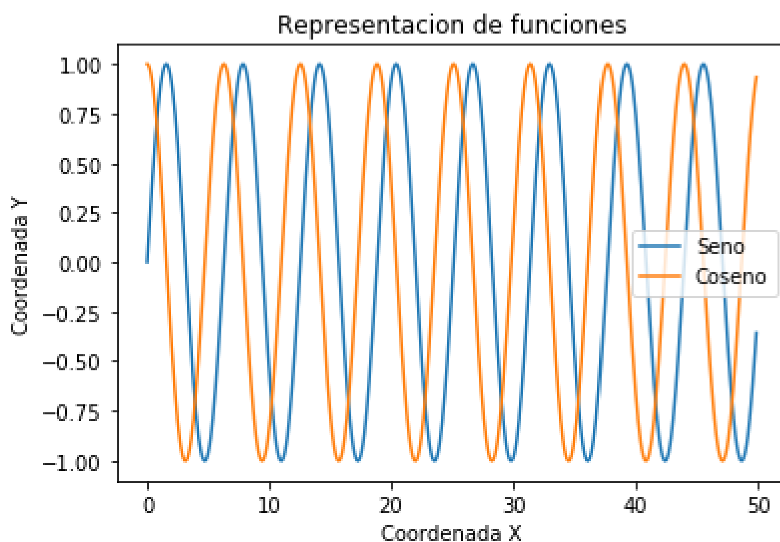
Veamos un sencillo ejemplo de CRUD (Create, read, update, delete) en MongoDB con Python:

```
# script para CRUD MongoDB

import pymongo

myclient = pymongo.MongoClient("mongodb://_USUARIO_:_CONTRASEÑA_@_IP_:_PUERTO_/")

mydb = myclient["mydatabase"]
mycol = mydb["users"]

# crear

mylist = [
  { "name": "Pedro", "password": "admin"},
  { "name": "Sandra", "password": "1234"}
]
c = mycol.insert_many(mylist)

# leer

lista = []
cursor = mycol.find({"name':'Pedro"})
for document in cursor:
    lista = (document['password'])

# actualizar

a = mycol.update_one({"password": "1234"}, {'$set': {"password": "4321"}})

# borrar

b = col.delete_many({"name": "Sandra"})
```

6.6 Matrices y gráficas - numpy y matplotlib

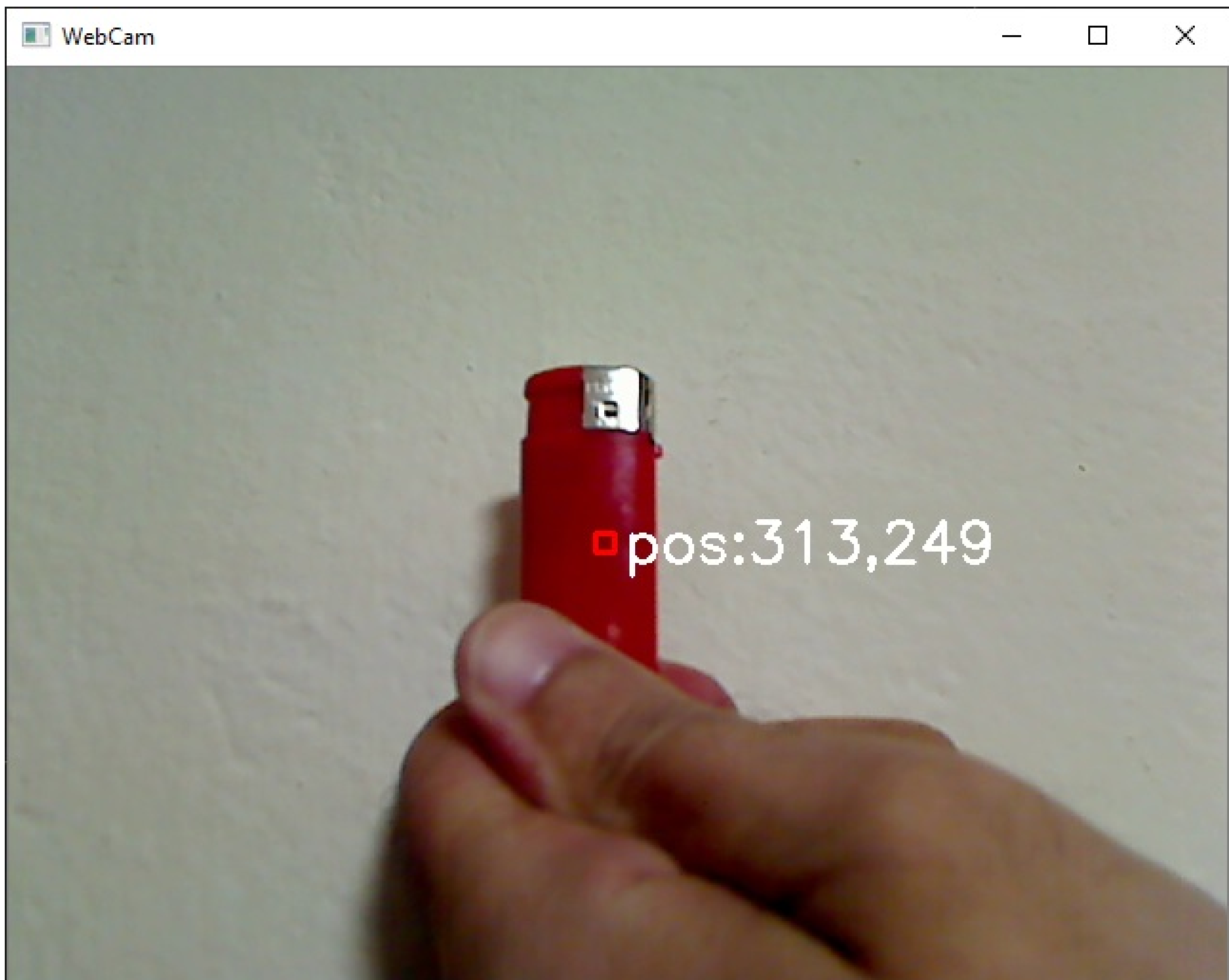
```
# pip install numpy        ->  cálculo de vectores y matrices
# pip install matplotlib   ->  generación de gráficos

import math
import numpy as np
from matplotlib import pyplot as plt

# Generamos los datos para el gráfico
x = np.array(range(500))*0.1
y = np.zeros(len(x))
for i in range(len(x)):
    y[i] = math.sin(x[i])

# Generamos un segundo conjunto de datos para el gráfico
x_2 = np.array(range(500))*0.1
y_2 = np.zeros(len(x_2))
for j in range(len(x_2)):
    y_2[j] = math.cos(x_2[j])

# Creamos el gráfico
plt.ion()
plt.plot(x,y,x_2,y_2)

# Colocamos las etiquetas de los ejes
plt.xlabel("Coordenada X")
plt.ylabel("Coordenada Y")

# Colocamos la leyenda
plt.legend(['Seno','Coseno'])

# Colocamos el título del gráfico
plt.title("Representacion de funciones")

# Guardamos la gráfica
# plt.savefig('gráfica.png')
```

```
Text(0.5, 1.0, 'Representacion de funciones')
```



6.7 Visión artificial - opencv

```
# script para seguimiento del color rojo con la cámara
# instalar opencv  ->  pip install opencv-contrib-python  -> cv2
import cv2
import numpy as np

# Iniciamos la cámara
captura = cv2.VideoCapture(0)

while(1):
    # Capturamos una imagen y la convertimos de RGB a HSV
    _, imagen = captura.read()
    hsv = cv2.cvtColor(imagen, cv2.COLOR_BGR2HSV)

    # Establecemos el rango de colores que vamos a detectar
    # En este caso de rojo claro a rojo oscuro
    rojo_bajos = np.array([162,100,100], dtype=np.uint8)
    rojo_altos = np.array([182,255,255], dtype=np.uint8)

    # Crear máscara con sólo los pixels dentro del rango de rojos
    mask = cv2.inRange(hsv, rojo_bajos, rojo_altos)

    #Encontrar el area de los objetos que detecta la camara
    moments = cv2.moments(mask)
    area = moments['m00']

    if(area > 20000):
        # Buscamos el centro (x, y) del objeto
        x = int(moments['m10']/moments['m00'])
        y = int(moments['m01']/moments['m00'])

        # Mostramos sus coordenadas por pantalla
        print ("x = " + str(x))
        print ("y = " + str(y))

        # Dibujamos una marca en el centro del objeto
        cv2.rectangle(imagen, (x-5, y-5), (x+5, y+5),(0,0,255), 2)
        cv2.putText(imagen, "pos:"+ str(x)+","+str(y), (x+10,y+10), cv2.FONT_HERSHEY_SI
MPLEX, 1, (255,255,255), 2)

    # Mostramos la imagen original con la marca del centro y la máscara
    cv2.imshow('mask', mask)
    cv2.imshow('WebCam', imagen)

    # Tecla q para salir
    if cv2.waitKey(1) & 0xFF == ord('q'):
        captura.release()
        break

cv2.destroyAllWindows()
```



En nuestro script la cámara detecta automáticamente el color rojo, lo sigue en su movimiento y nos indica constantemente sus coordenadas en la pantalla. Es sólo un pequeño ejemplo de lo que podemos llegar a hacer con esta librería.

OpenCV (Open Source Computer Vision) es una biblioteca libre de visión artificial originalmente desarrollada por Intel que analiza imágenes capturadas por una cámara y obtiene información de los objetos que se hallan presentes en la escena, constituye uno de los campos de la Inteligencia Artificial con un mayor ritmo de desarrollo y que más aplicaciones nuevas está presentando.

Opencv es utilizada en infinidad de aplicaciones, desde sistemas de seguridad con detección de movimiento, hasta aplicaciones de control de procesos donde se requiere el reconocimiento de objetos o personas, en la conducción autónoma de vehículos sirve para detectar las líneas de la carretera y a los otros vehículos, últimamente también se ha incorporado a la traducción automática de lenguas de signos, etc.

Para ello dispone de infinitud de algoritmos que permiten, con sólo escribir unas pocas líneas de código, identificar rostros, reconocer objetos, clasificarlos, detectar movimientos de manos…

OpenCV es una librería multiplataforma disponible para Windows, Mac, Linux y Android distribuida bajo licencia BSD. Puede programarse con C, C++, Python, Java y Matlab.

6.8 Texto a voz

Con un sencillo script podemos utilizar la api win32com.client para convertir cualquier texto en voz.

- **pip install pypiwin32** -> win32com.client -> manipular funciones de sistema operativo Windows
- **pip install comtypes** -> librería Python para objetos COM
- **pip install pydub** -> audio

***Nota**: si tienes problemas a la hora de convertir formatos de audio recomiendo FFmpeg (https://ffmpeg.org/download.html), es una colección de software libre que puede convertir gran variedad de archivos de audio y video. Tan sólo deberás añadirlo al path en tus variables de entorno para poder ejecutar desde la consola de comandos.

```
import os

import win32com.client as wincl
from comtypes.client import CreateObject
from comtypes.gen import SpeechLib
import pydub

text = input("Introduce un texto: ")
text = (text.replace("\n"," "))
print (text)
speak = wincl.Dispatch("SAPI.SpVoice")
speak.Speak(text)
engine = CreateObject("SAPI.SpVoice")
stream = CreateObject("SAPI.SpFileStream")
stream.Open('audio.wav', SpeechLib.SSFMCreateForWrite)
engine.AudioOutputStream = stream
engine.speak(text)
stream.Close()

sound = pydub.AudioSegment.from_wav("audio.wav")
sound.export("Voz.mp3", format="mp3")
os.remove("audio.wav")
```

6.9 Servidor - flask

Flask es un framework minimalista escrito en Python que permite crear aplicaciones web rápidamente y con un mínimo número de líneas de código.

Para instalar: **pip install flask**

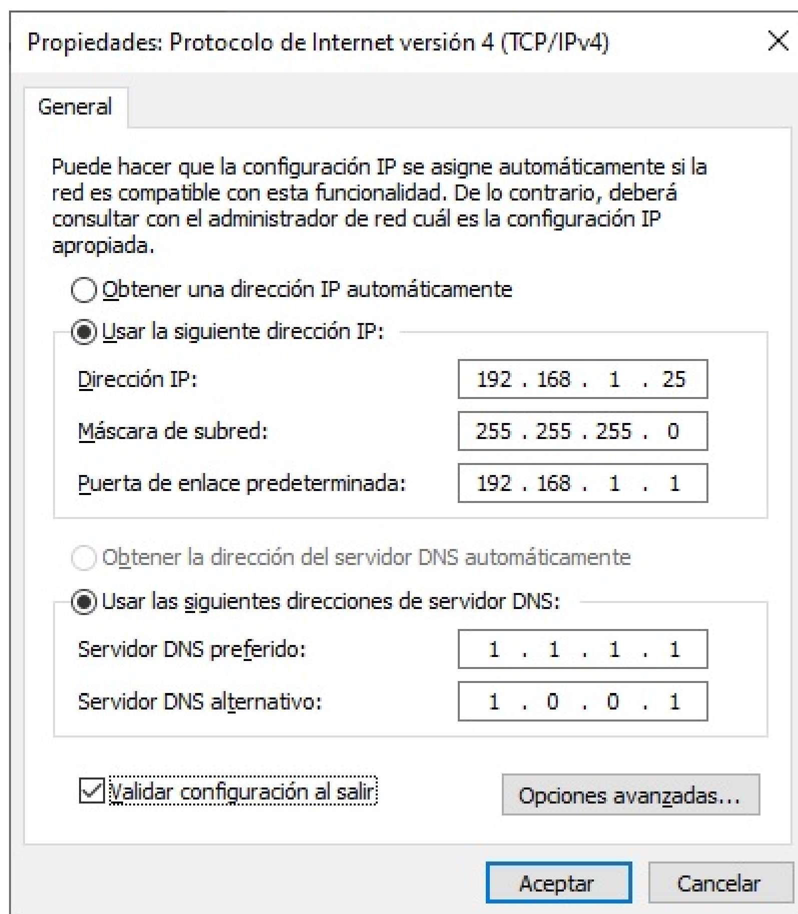
```
from flask import Flask
app = Flask(__name__)

@app.route('/')
def index():
    return 'Hola Mundo!'

if __name__ == '__main__':
    app.run(debug=True, host='localhost', port=80)
```

Como podemos observar con sólo 7 líneas de código ya podemos correr el servidor, para entrar pondremos en el navegador http://127.0.0.1/ (http://127.0.0.1/) o localhost, muy básico, sí, y sólo funcionará en nuestra red local, pero podemos ir agregándole múltiples funcionalidades.

Los que hemos sido fans de Apache Server durante años estamos de enhorabuena, al estar Flask basado en la especificación WSGI, podemos servir aplicaciones Flask con Apache a través de mod_wsgi, aunque este no es el objetivo de este manual es importante tenerlo en cuenta.

Si nunca has configurado un servidor vamos a ver algunas consideraciones que te ayudarán a hacerlo:

Configurar IP privada: lo primero que tenemos que hacer es configurar nuestro equipo para poder identificarlo dentro de nuestra red. Si quieres ver tu ip privada puedes hacer un ipconfing en la consola, si tienes tu adaptador de red en modo automático cada vez que te conectes tendrás una ip privada (dentro de tu red) diferente, esto no nos sirve. Normalmente la puerta de enlace siempre es 192.168.1.1 (tu router) y la de tu equipo será 192.168.1.XX. Para hacer que siempre tengamos la misma ip privada iremos a Configuración de Red / Ethernet o Wifi según sea tu conexión / y picaremos en Cambiar opciones del adaptador. En nuestro adaptador entraremos en Propiedades / Protocolo de Internet 4 (TCP/IPv4) y pulsamos en Propiedades, deberemos configurarlo así:



Puedes cambiar el 25 por cualquier otro, te aconsejo uno de rango bajo, pero en este ejemplo seguiré con el 25. Una vez que le des click a Aceptar tu ip privada será siempre **192.168.1.25**.

Configurar gestor DDNS: la mayoría de nosotros tenemos una ip pública (la que nos identifica en la red exterior) dinámica, es decir, nuestro proveedor de Internet nos asigna una IP pública diferente cada cierto tiempo. Esto es un gran inconveniente a la hora de acceder a nuestro equipo porque tenemos que estar pendientes de qué IP tenemos. Para solucionar este pequeño problema están los gestores de DDNS, existen múltiples, algunos gratuitos y otros de pago. Mi recomendación es DuckDNS (https://www.duckdns.org), es totalmente gratuito, cuida la privacidad de los usuarios y funciona con cualquier sistema operativo. Una vez que nos registremos tan sólo tendremos que elegir un subdominio,

vamos a suponer que elegimos **gazpacho.duckdns.org**, es que hace mucho calor hoy **;-)**. Una vez lo hayamos elegido nos darán un token (un código). Descargamos el instalador del Cliente DuckDNS (http://etx.ca/downloads/DuckDnsInst.exe), lo instalamos y nos aparecerá el icono de un patito al lado del reloj, con el botón derecho picamos en DuckDNS Settings y en Domain ponemos gazpacho y en Token el código que nos han dado. Para asegurarnos que el cliente se ejecuta siempre que encendemos el equipo vamos a la carpeta de instalación (C:\Program Files (x86)\DuckDNS) creamos un acceso directo a DuckDns.exe y lo pegamos en la carpeta de incio de Windows (C:\ProgramData\Microsoft\Windows\Start Menu\Programs\StartUp). Y ya tenemos nuestro DDNS listo.

Editar el host: para evitar problemas al acceder desde tu propia red, sólo tendremos que agregar al host situado en (C:\Windows\System32\drivers\etc) esta línea al final:

192.168.1.25 gazpacho.duckdns.org

Abrir puerto 80 en el router: por último, iremos a nuestro router y abriremos el puerto 80 TCP para 192.168.1.25, depende del modelo pero suele estar en NAT mapeo de puertos.

Cambiamos la última línea del script anterior y en lugar de localhost ponemos gazpacho.duckdns.org, y listo, ya podremos acceder a nuestro servidor desde cualquier parte del mundo.

Aunque que esto te haya parecido un poco largo deberemos hacerlo para configurar cualquier servidor, Apache, Flask, Django, NodeJS, etc.

6.10 Otras librerías

Para terminar citaré otras librerías muy usuales utilizadas en Python:

Librería	Campo
pygame	juegos
pyqt	interfaz gráfica
kivy	aplicaciones android
django	servidor
tornado	servidor
sympy	álgebra computacional
scipy	ciencia, ingeniería
pandas	análisis de datos
tensorflow	IA - machine learning
scikit-learn	IA - machine learning
keras	redes neuronales

En fin, seguro que me dejo bastantes temas por tocar, pero como podéis observar con Python se pueden hacer muuuuchas cosas.

Volver al índice