Índice

[**¿Porque programar en Python?** 1](#_Toc32255378)

[**¿Qué es la programación?** 1](#_Toc32255379)

[**Python 2 vs Python 3** 1](#_Toc32255380)

[**¿Qué es Python?** 2](#_Toc32255381)

[**Tipos de datos en Python** 2](#_Toc32255382)

[**Funciones** 3](#_Toc32255383)

[**Variables** 3](#_Toc32255384)

[**Listas** 3](#_Toc32255385)

[**Tuplas** 4](#_Toc32255386)

[**Diccionarios** 4](#_Toc32255387)

[**Conversiones** 4](#_Toc32255388)

[**Operadores Comunes** 5](#_Toc32255389)

[**Clases** 6](#_Toc32255390)

[**Métodos especiales** 7](#_Toc32255391)

[**Condicionales IF** 7](#_Toc32255392)

[**Bucle FOR** 8](#_Toc32255393)

[**Bucle WHILE** 8](#_Toc32255394)

[**Operadores matemáticos en Python** 9](#_Toc32255395)

[**Tipos de datos en Python** 10](#_Toc32255396)

[**Valores y Tipos** 10](#_Toc32255397)

[**Declarar variables y expresiones** 11](#_Toc32255398)

[**Definir funciones con Python** 13](#_Toc32255399)

[**Funciones con parámetros** 14](#_Toc32255400)

[**Límites al declarar funciones** 14](#_Toc32255401)

[**Imprimir valor de variable** 14](#_Toc32255402)

[**Flujo de ejecución** 14](#_Toc32255403)

[**Declarar vs Ejecutar** 14](#_Toc32255404)

[**Donde se puede acceder a las variables** 14](#_Toc32255405)

[**Estructura de condicionales en Python** 15](#_Toc32255406)

[**Operadores relacionales** 15](#_Toc32255407)

[**Operadores lógicos** 15](#_Toc32255408)

[**Buenas prácticas del lenguaje** 15](#_Toc32255409)

[**Documentación de Python** 16](#_Toc32255410)

[**Comparación de strings y unicode** 17](#_Toc32255411)

[**Comparación de strings** 17](#_Toc32255412)

[**Diferencia entre ASCII y Unicode** 17](#_Toc32255413)

[**Factorial de un número con recursión** 17](#_Toc32255414)

[**Manejo de strings en Python** 18](#_Toc32255415)

[**Separar cadenas de texto en Python** 19](#_Toc32255416)

[**Ciclo for** 19](#_Toc32255417)

[**Función range** 19](#_Toc32255418)

[**Iteraciones con for** 20](#_Toc32255419)

[**Ciclo while** 20](#_Toc32255420)

[**Introducción a las listas en Python** 21](#_Toc32255421)

[**Operaciones con listas en Python** 22](#_Toc32255422)

[**Unir listas** 22](#_Toc32255423)

[**Multiplicar elementos** 22](#_Toc32255424)

[**Invertir listas** 22](#_Toc32255425)

[**Eliminar último elemento de la lista** 22](#_Toc32255426)

[**Ordenar la lista** 22](#_Toc32255427)

[**Eliminar un elemento** 23](#_Toc32255428)

[**Tienen las siguientes características** 23](#_Toc32255429)

[**Listas y Strings** 23](#_Toc32255430)

[**Qué es una búsqueda binaria** 24](#_Toc32255431)

[**Diccionarios en Python** 26](#_Toc32255432)

[**Declarar diccionarios y agregar datos** 26](#_Toc32255433)

[**Iterar en llaves** 26](#_Toc32255434)

[**Iterar en valores** 26](#_Toc32255435)

[**Iterar en ítems** 26](#_Toc32255436)

[**Tuplas en Python** 26](#_Toc32255437)

[**Definir una tupla** 27](#_Toc32255438)

[**Operadores en las tuplas** 27](#_Toc32255439)

[**Uso de sets en Python** 28](#_Toc32255440)

[**Declarar sets** 28](#_Toc32255441)

[**Operaciones con sets** 28](#_Toc32255442)

[**Dictionary comprehension - list comprehension** 29](#_Toc32255443)

[**Diferencias** 29](#_Toc32255444)

[**Manejo de errores en Python** 30](#_Toc32255445)

[**Jerarquía de errores en python** 30](#_Toc32255446)

[**Manejo de archivos en Python** 32](#_Toc32255447)

[**Lectura de archivo** 32](#_Toc32255448)

[**Escritura del archivo** 32](#_Toc32255449)

[**Programación orientada a objetos en Python** 33](#_Toc32255450)

[**Clases y objetos** 33](#_Toc32255451)

[**Clases y métodos** 33](#_Toc32255452)

[**Definición de clase** 34](#_Toc32255453)

[**Decoradores en Python** 34](#_Toc32255454)

[**Paquetes y módulos en python** 36](#_Toc32255455)

[**Entorno virtual en Python** 37](#_Toc32255456)

[**Instalación** 37](#_Toc32255457)

[**Crear un entorno virtual** 37](#_Toc32255458)

[**Encender un entorno virtual** 37](#_Toc32255459)

[**Ver las dependencias instaladas en el entorno virtual** 37](#_Toc32255460)

[**¿Qué es web scraping?** 38](#_Toc32255461)

[**Librerías que se utilizan** 38](#_Toc32255462)

[ **requests** 38](#_Toc32255463)

[ **BeautifulSoup** 38](#_Toc32255464)

[**Qué puedes construir con Python** 38](#_Toc32255465)

# **¿Porque programar en Python?**

Existen 3 razones:

1. Tiene una de las comunidades más grandes que cualquier lenguaje de programación junto a Java.
2. Tiene unas de las mejores sintaxis, sencillas y fácil de entender lo cual que hace que el código sea legible.
3. Es ampliamente utilizado en toda la industria.

# **¿Qué es la programación?**

Es dar instrucciones a la computadora de forma ordenada, estas instrucciones definen lo que se va a realizar, desde cosas sencillas hasta cosas avanzadas.

Todos los programas se componen de partes esenciales que son forma de darle datos de entrada a la computadora y formas estructuradas para procesar estos datos.

# **Python 2 vs Python 3**

Cuando hablamos de Python, tenemos varias versiones, actualmente compiten **python 2** y **python 3**.

Python 3 arregla ciertas inconsistencias en el lenguaje que hacía difícil programar, algunas de las diferencias más significativas son:

* **print** ahora es una función.
* **Unicode** está establecido por defecto en los strings.
* Strings.
* **Integer división**, en la versión de python 2.7.x sólo toma la parte entera de una división, en python 3 ya no sucede.
* Raising exceptions.
* Input vs raw\_input.

Recuerda tener en cuenta si las librerías que usas están escritas solo para Python 2.

# **¿Qué es Python?**

Python es un lenguaje de programación creado por Guido Van Rossum, con una sintaxis muy limpia, ideado para enseñar a la gente a programar bien. Se trata de un lenguaje interpretado o de script.

Ventajas

* **Legible**: Sintaxis intuitiva y estricta.
* **Productivo**: Ahorra mucho código.
* **Portable**: Para todo sistema operativo.
* **Recargado**: Viene con muchas librerías por defecto.

# **Tipos de datos en Python**

* **Enteros (int):** En este grupo están todos los números, enteros y long:

Ej: 1, 2.3, 2121, 2192, -123

* **Booleanos (bool):** Son los valores falso o verdadero, compatibles con todas las operaciones booleanas (and, not, or)

Ej: True, False

* **Cadenas (str):** Son una cadena de texto.

Ej: “Hola”, “¿Cómo estás?”

* **Listas:** Son un grupo o array de datos, puede contener cualquiera de los datos anteriores.

Ej: [1,2,3, ”hola” , [1,2,3] ], [1,“Hola”,True ]]

* **Diccionarios:** Son un grupo de datos que se acceden a partir de una clave

Ej: {“clave”:”valor”}, {“nombre”:”Fernando”}

* **Tuplas:** también son un grupo de datos igual que una lista con la diferencia que una tupla después de creada no se puede modificar.

Ej: (1,2,3, ”hola” , [1,2,3] ], [1,“Hola”,True ])

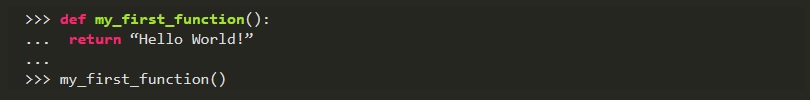
(**Pero jamás podremos cambiar los elementos dentro de esa Tupla**)

En Python trabajas con módulos y ficheros que usas para importar las librerías.

# **Funciones**

Las funciones se definen con la palabra reservada **def** junto a un nombre y unos paréntesis que reciben los parámetros a usar. Termina con dos puntos.

Ej:



Salida: Hello World!

# **Variables**

Las variables, a diferencia de los demás lenguajes de programación, no debes definirlas, ni tampoco su tipo de dato, ya que al momento de iterarlas se identificará su tipo. Recuerda que en Python todo es un objeto.

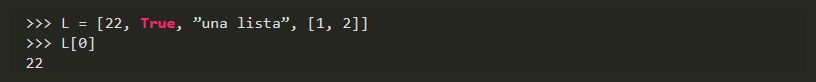
Ej:



# **Listas**

Las listas se declaran con corchetes. Estas pueden tener una lista dentro o **cualquier** tipo de dato.

Ej:



# **Tuplas**

Las tuplas se declaran con paréntesis, recuerda que no puedes editar los datos de una tupla después de que la has creado. Son inmutables.

Ej:



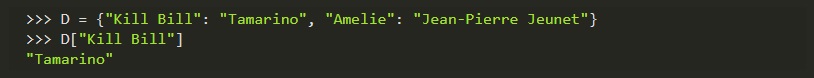
La diferencia entre tuplas y listas es que las tuplas por ser un tipo de dato inmutable consumen menos memoria y son más rápidas, las listas por otro lado, igual son rápidas, pero al poder cambiar sus valores consumen más memoria, las tuplas se deben usar para opciones que sabes que no cambiarán, las listas se usan para almacenar datos de iteraciones, por ejemplo: Ya que no sabes cuantos elementos vas a tener y todo esto cambia al momento de ejecutar.

# **Diccionarios**

En los diccionarios tienes un grupo de datos con un formato:

1. La primera cadena o número será la clave para acceder al segundo dato.
2. El segundo dato será el dato al cual accederás con la llave.

Recuerda que los diccionarios son listas de **llave:valor**.



# **Conversiones**

De flotante a entero.



De entero a flotante.



De entero a string.



De tupla a lista.



# **Operadores Comunes**

Longitud de una cadena, lista, tupla, etc.



Tipo de dato.



Aplicar una conversión a un conjunto como una lista.



Redondear un flotante con **x** número de decimales.



Generar un rango en una lista (esto es mágico.)



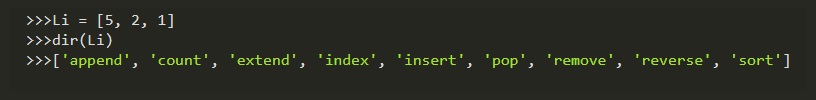
Sumar un conjunto.



Organizar un conjunto.



Conocer los comandos que le puedes aplicar a **x** tipo de datos.



* append
* count
* extend
* index
* insert
* pop
* remove
* reverse
* sort

Son posibles comandos que puedes aplicar a una lista.

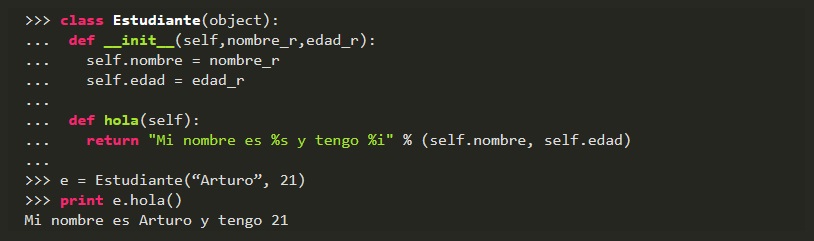
Información sobre una función o librería.



# **Clases**

Es uno de los conceptos con más definiciones en la programación, pero en resumen sólo son la representación de un objeto. Para definir la clase usas **class** y el nombre. En caso de tener parámetros los ponemos entre paréntesis.

Para crear un constructor hacemos una función dentro de la clase con el nombre **init** y de parámetros **self** (significa su clase misma), **nombre\_r** y **edad\_r**.



Lo que hicimos en las dos últimas líneas fue:

1. En la variable **e** llamamos la clase **Estudiante** y le pasamos la cadena “**Arturo**” y el entero **21**.
2. Imprimimos la función **hola()** dentro de la variable **e** (a la que anteriormente habíamos pasado la clase).

Y por eso se imprime la cadena “Mi nombre es Arturo y tengo 21”

# **Métodos especiales**

**cmp**(self,otro)

Método llamado cuando utilizamos los operadores de comparación para comprobar si tu objeto es menor, mayor o igual al objeto pasado como parámetro.

**len**(self)

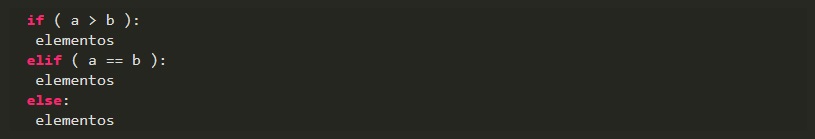
Método llamado para comprobar la longitud del objeto. Lo usas, por ejemplo, cuando llamas la función len(obj) sobre nuestro código. Como es de suponer el método te debe devolver la longitud del objeto.

**init**(self,otro)

Es un constructor de nuestra clase, es decir, es un “método especial” que es llamas automáticamente cuando creas un objeto.

# **Condicionales IF**

Los condicionales tienen la siguiente estructura. Lo que contiene los paréntesis es la comparación que debe cumplir para que los elementos se cumplan.



# **Bucle FOR**

El bucle de **for** lo puedes usar de la siguiente forma:

Recorres una cadena o lista a la cual va a tomar el elemento en cuestión con la siguiente estructura.



Ejemplo:



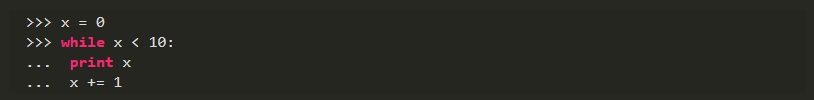
En este caso recorrerá una lista de diez elementos, es decir el print **i** se ejecutará diez veces. Ahora **i** va a tomar cada valor de la lista, entonces este for imprimirá los números del 0 al 9 (recordar que en un **range** vas hasta el número puesto -1).

# **Bucle WHILE**

En este caso **while** tiene una condición que determina hasta cuándo se ejecutará. O sea que dejará de ejecutarse en el momento en que la condición deje de ser verdadera. La estructura de un **while** es la siguiente:



Ejemplo:



En este ejemplo preguntará si **x** es menor que 10. Dado que es menor imprimirá **x** y luego sumará una unidad a **x** con la expresión **x += 1**. Luego **x** vale 1 y como sigue siendo menor a 10 se seguirá ejecutando, y así sucesivamente hasta que **x** llegue a ser mayor o igual a 10.

# **Operadores matemáticos en Python**

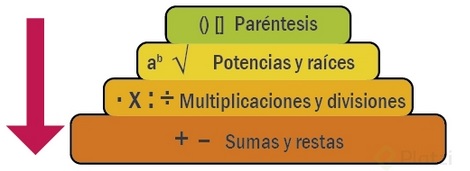
Los operadores nos permiten trabajar con valores y generar nuevos valores por medio de estas operaciones. Especifican la operación que debe realizar la computadora.

Existen muchos operadores básicos:

* (**+**) Suma.
* (**-**)Resta.
* (**\***) Multiplicación.
* (**/**) División.
* (**//**) División de enteros.
* (**%**) Operador de módulo.
* (**\*\***) Potencias.
* (**>**) Mayor que.
* (**<**)Menor que.
* (**==**)Igual.
* (**>=**) Mayor igual.
* (**<=**)Menor igual.

**ADVERTENCIA**: Los paréntesis () no son parte del lenguaje, su contenido sí.

# **Tipos de datos en Python**

¿En qué orden se evalúan las operaciones complejas en matemáticas?

* Paréntesis.
* Exponentes.
* Multiplicación / División.
* Adición / Sustracción.

Una forma fácil de recordar éste orden es usando el acrónimo PEMDAS (**P**arentesis**E**xponentes**M**ultiplicación**D**ivisión**A**dición**S**ustracción)

## **Valores y Tipos**

Los tipos le permiten a Python saber cuál es el resultado de aplicar determinada operación, los tipos básicos son:

* Integer <int>
* Float <float>
* String <str>
* Boolean <bool>

Para revisar de qué tipo es un dato puedes usar la función type.

Ejemplos:

**type**(2) - retorna **int**

**type**('Hola') - retorna **str**

Recuerda que tu código debe ser claro, para permitir que otros programadores lo puedan leer.

También puedes ver el tipo correspondiente al resultado de alguna operación:

**type**((2\*\*3)+53.4)

<class ‘float’>

**type**(‘Ernesto’ \* 2 )

<class ‘str’>

Los operadores dependen del contexto, por ej:

2 + 2 = 4

‘Hola’ + ‘Hola’ = HolaHola

# **Declarar variables y expresiones**

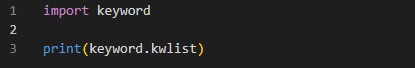
Las variables nos permiten guardar valores, permitiéndonos reutilizarlos en diferentes partes del código y haciendo nuestros programas más legibles.

El valor que contiene una variable puede ser reasignado, significa que podemos asignarle diferentes valores a una misma variable.

Las variables tienen algunas limitantes, por ejemplo:

* Tienen que tener un nombre significativo, es decir, que nos digan qué están haciendo.
* No podemos usar palabras reservadas del lenguaje como nombres para nuestras variables, por ejemplo:
  + **False**
  + **None**
  + **True**
  + **and**
  + **as**
  + **assert**
  + **break**
  + **class**
  + **continue**
  + **def**
  + **del**
  + **elif**
  + **else**
  + **except**
  + **finally**
  + **for**
  + **from**
  + **global**
  + **if**
  + **import**
  + **in**
  + **ls**
  + **lambda**
  + **nonlocal**
  + **not**
  + **or**
  + **pass**
  + **raise**
  + etc…

Para ver todas las palabras reservadas en Python se puede hacer:



Para definir una variable escribes el nombre que quieres asignarle y con = (igual) defines el valor que va a almacenar, por ejemplo:

pi **=** 3.14159

radio **=** 3

area **=** pi \* radio\*\*2

print(area)

Recuerda:

* Todos los programas de Python deben guardarse con extensión **.py**
* Para darle soporte a acentos en nuestros programas debemos usar la línea **# -\*- coding: utf-8 -\*-**
* Las variables globales, van a poder ser accedidas en cualquier parte del programa.
* Cuando en Python definimos una constante, por convención las definimos escribiendo el nombre con mayúscula.

# **Definir funciones con Python**

Las funciones son un concepto fundamental en programación, una función es una secuencia de comandos que realizan un cómputo.

En Python las funciones se definen usando la palabra reservada **def** (del inglés define, que significa definir una función) y luego el nombre de la función con paréntesis y **:** (dos puntos) que indican que lo que sigue son los comandos, una función debe retornar un valor, para esto se usa la palabra reservada **return**.



Para usar una función simplemente la llamamos por su nombre seguido por paréntesis con los parámetros que recibe.



También se puede definir una función vacía que no retorne ningún valor colocándole dentro de ella la palabra reservada **pass**, la cual es un placeholder, es una forma de decirle a Python que acabamos de definir una función, la cual no tiene nada adentro todavía, pero que en algún momento le vamos a poner algo, sino le ponemos palabra reservada **pass** Python no sabrá cuando empieza y cuando terminala función entonces lanzaría un error.



Recuerda:

* Si usas Python 3, debes usar la función input() para recibir datos del usuario.
* Para definir dónde comenzar el código usamos la línea



* Para definir un bloque dentro de la función debemos indentar con 4 espacios.
* Las funciones nos permiten ejecutar determinado código con diferentes valores.

# **Funciones con parámetros**

## **Límites al declarar funciones**

* Los nombres no pueden comenzar con dígitos.
* No pueden utilizar una palabra reservada.
* Las variables deben tener diferentes nombres que las funciones (sino la última definición es la que gana.)
* Los nombres de las funciones deben ser descriptivas de lo que hacen las funciones.

## **Imprimir valor de variable**

Para poder imprimir el valor de una variable dentro de un string podemos hacerlo así:



## **Flujo de ejecución**

* Comienza con la primera declaración de arriba hacia abajo.
* Cada declaración se ejecuta de izquierda a derecha, siguiendo el orden de operaciones.
* Cuando una declaración contiene una llamada a una función, el cuerpo de la función se ejecuta con las mismas reglas.

## **Declarar vs Ejecutar**

* Declarar una función es escribir su estructura.
* Ejecutar una función es llamar la función y ejecutar su código.

## **Donde se puede acceder a las variables**

*Escope* (ámbito) de las variables, cada vez que una función se ejecuta se genera un contenedor donde las variables de la función van a vivir, una vez se sale de la función estas variables no existirán más.

# **Estructura de condicionales en Python**

Uno de las cosas más poderosas en programación es controlar el flujo de ejecución, para esto utilizamos condicionales.

Las condicionales son sentencias que devuelven un valor booleano (True, False)

Podemos utilizar:

## **Operadores relacionales**

* **==** Es igual
* **!=** Es diferente
* **>** Es mayor
* **>=** Es mayor o igual
* **<** Es menor
* **<=** Es menor o igual

## **Operadores lógicos**

* **and**
* **or**
* **not**

# **Buenas prácticas del lenguaje**

El Zen de Python, es un apartado dentro del lenguaje donde se explican varios principios importantes para generar un gran desarrollo de software. Para verlo simplemente se debe hacer:



Está línea de código muestra los siguientes principios.

* Hermoso es mejor que feo.
* Explícito es mejor que implícito.
* Simple es mejor que complejo.
* Complejo es mejor que complicado.
* Sencillo es mejor que anidado.
* Escaso es mejor que denso.
* La legibilidad cuenta.
* Los casos especiales no son lo suficientemente especiales para romper las reglas.
* Lo práctico le gana a la pureza.
* Los errores no deben pasar en silencio, a menos que sean silenciados.
* En cara a la ambigüedad, rechazar la tentación de adivinar.
* Debe haber una - y preferiblemente sólo una - manera obvia de hacerlo, aunque esa manera puede no ser obvia en un primer momento a menos que seas holandés.
* Ahora es mejor que nunca, aunque “nunca” es a menudo mejor que “ahora mismo”.
* Si la aplicación es difícil de explicar, es una mala idea.
* Si la aplicación es fácil de explicar, puede ser una buena idea.
* Los espacios de nombres son una gran idea ¡hay que hacer más de eso!

# **Documentación de Python**

Para ver la documentación, la librería estándar que trae el lenguaje por defecto se puede ingresar al siguiente [enlace](https://docs.python.org/3/library/index.html) para verla.

# **Comparación de strings y unicode**

Los strings tiene una característica muy importante, **son inmutables**, esto quiere decir que no se pueden cambiar después de que se han declarado.

Si quieres modificar el texto de un string debes definir un nuevo string y modificarlo usando funciones como slice.

## **Comparación de strings**

Se pueden realizar operaciones con strings, por ejemplo, comparar si son iguales o mayores o menores. La manera en que Python hace esto es por orden lexografico, es decir, dándole un mayor peso a las letras finales y un menor a las iniciales, por ej: y < z = true

## **Diferencia entre ASCII y Unicode**

Los caracteres también son números, para esto existen estándares que asigna un número a cada carácter, para generar un estándar se creó el ASCII, pero esta sólo toma en cuenta los caracteres en inglés, para dar soporte a más lenguajes se crea UNICODE.

Python codifica en ASCII por default, para cambiarlo por UNICODE debemos colocar **u** antes del string.

# **Factorial de un número con recursión**

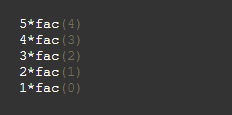
Una función es recursiva cuando dentro del bloque de instrucciones que la conforma se usa a sí misma.

El concepto puede sonar complicado, pero es muy común su uso, por ejemplo, cuando haces el cálculo del factorial de un número lo haces con una función recursiva:

El factorial de un número es el número multiplicado por los números antes de él, por ejemplo:

5! es 5\*4\*3\*2\*1

Esto se puede expresar como

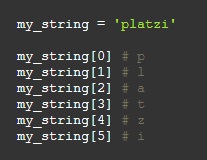


**Nota importante**

Cuándo estés trabajando con recursividad siempre debes pensar en el caso base, es decir, debes definir el momento en el que la función dejará de llamarse a sí misma, para que no hagas un loop infinito, por ejemplo, en el caso del factorial terminas la ejecución cuando llegas a cero.

# **Manejo de strings en Python**

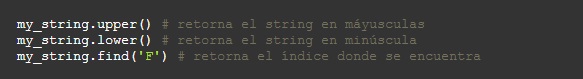
Un string es una secuencia de caracteres, donde cada carácter tiene un índice que inicia en cero, por ejemplo:



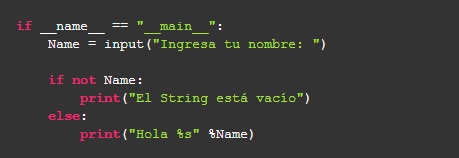
Para conocer la longitud de un string podemos usar la función **len()**



Los string tienen algunos métodos útiles cómo:

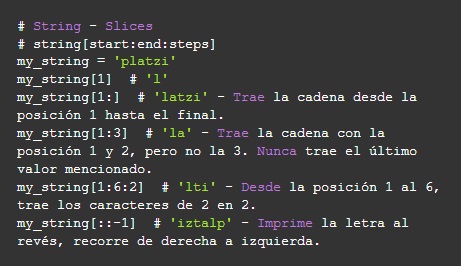


Para saber si un string está vacío podemos hacer.



# **Separar cadenas de texto en Python**

* La función **slice** (rebanadas en español) de Python nos permite separar los strings en substrings generando nuevas secuencias.
* Se definen los rangos que se quieren y los saltos necesarios



# **Ciclo for**

Cuando necesitamos realizar operaciones sobre una serie de datos podemos utilizar iteraciones.

## **Función range**

La función range nos permite generar un string a partir de un rango.



## **Iteraciones con for**

**for** nos permite recorrer un arreglo, asignando cada valor a una variable que decidas. Necesita la palabra reservada **in**.



Podemos operar los valores usando también condiciones, en este caso sólo queremos elevar al cuadrado, los valores que sean divisibles por 3.



* La palabra reservada **continue** permite saltar a la siguiente iteración del ciclo y **break** permite salirse del ciclo.
* Los ciclos nos permiten ejecutar una y otra vez una determinada cantidad de acciones mientras una condición se cumpla.
* Los ciclos pueden ser utilizados para recorrer un rango de números (desde:hasta) o para recorrer un string.

# **Ciclo while**

Otra forma de hacer iteraciones es con el **while** loop, éste ciclo se ejecuta **MIENTRAS** la condición se evalúe como verdadera, el ciclo termina cuando la evaluación resulta en falso.

En este tipo de ciclo es muy importante definir la condición de parada, si no el ciclo puede ejecutarse hasta el infinito y consumir todos los recursos de la máquina.

Recuerda:

Para detener la ejecución de un ciclo, puede utilizar **CTRL** + **C** en la consola.

# **Introducción a las listas en Python**

Ya vimos que los strings son una secuencia de caracteres, cuando hablamos de listas podemos hablar de secuencias de cualquier cosa, por ejemplo, listas de números, listas de strings, listas de objetos más complejos con tipos propios etc.

Una lista **es una secuencia de elementos**, para crearlas usamos corchetes [] o con la función **list()**, por ejemplo:



* Las listas se pueden acceder a través de índices, éstos índices inician en cero (0)
* Las listas **son mutables**, para añadir elementos a una lista podemos utilizar el método **append**.
* Las listas se pueden acceder con índices, que inician en cero.



Un ejemplo más claro.

mi\_lista = [‘Juan’, ‘Pedro’, ‘Pepe’]

|  |  |  |
| --- | --- | --- |
| Juan | Pedro | Pepe |
| 0 | 1 | 2 |

Cada elemento de la lista representa un índice distinto.

Podemos inicializar nuestras listas de dos formas:

* Utilizamos dos corchetes vacíos []
* Utilizamos la palabra reservada **list()**

Pero tiene mejor rendimiento el definir listas vacías usando [] y diccionarios usando {} que si se definen con **list()** y **dict()** respectivamente.

# **Operaciones con listas en Python**

Cuando trabajamos con listas podemos también hacer operaciones, por ejemplo.

## **Unir listas**

****

## **Multiplicar elementos**

****

## **Invertir listas**

****

## **Eliminar último elemento de la lista**

****

## **Ordenar la lista**

****

## **Eliminar un elemento**

****

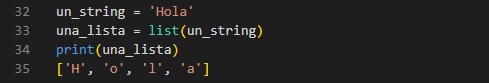
## **Tienen las siguientes características**

* Con las listas [] sólo se puede utilizar dos operadores y estos son:
  + La suma (+)
  + La multiplicación (\*).
* Con el signo **+** entre dos listas lo que hace es que concatena ambas listas formando una nueva que contempla los valores que componen ambas listas.
* Con el signo **\*** lo que conseguimos es multiplicar los elementos que están dentro de la lista por el número que deseemos. Si, por ejemplo, tengo una lista con un elemento y multiplico la lista por 5, ese elemento va multiplicarse por 5 veces.
* Las listas si son mutables y una de las maneras que podemos utilizar para cambiar uno de sus elementos sería indicando su índice y el operador de asignación seguido del elemento nuevo que queremos que reemplace al anterior, por ejemplo:

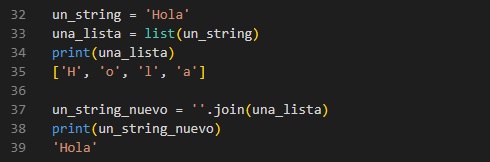
lista1[2] = “nuevoElem”

## **Listas y Strings**

Como un string es una secuencia de caracteres, se puede utilizar para inicializar una lista, por ej:



Si queremos volver a tener nuestro string podemos crear otra variable (porque los string son inmutables, no podemos asignarle a **un\_string** otro valor)



# **Qué es una búsqueda binaria**

Los algoritmos son una serie de instrucciones que nos lleva a un resultado, por ejemplo, que tal si queremos saber si un número se encuentra en una lista.

Una forma es buscar un ítem tras otro, pero sí la lista es muy larga esta forma no es la más eficiente, una forma más eficiente de solucionar éste problema es usar una búsqueda binaria.

Con el algoritmo de búsqueda binaria partimos de la lista ordenada, nosotros sabemos que hay números mayores y menores que el número que estamos buscando.

Seleccionamos un número aleatorio para dividir la lista, se puede escoger cualquier número, en éste caso sumamos el primer y el último índice de la lista, los sumamos y dividimos en dos (por esto se llama búsqueda binaria), luego comparamos el número que está en el índice, de esta manera ya eliminamos la mitad de las opciones. Podemos continuar dividiendo la lista y comparando hasta que lleguemos al resultado esperado. Por ejemplo:

* Si tenemos una lista de números

[2,5,3,8,19,7,8,**10**]

Si queremos buscar el número 10 con la forma tradicional iterando un ítem tras otro, nos tardaremos 8 pasos, porque la lista tiene 8 elementos y el número buscado está en última posición.

* Si tendríamos la lista ordenada, quedaría así:

[2, 3, 5, 7, 8, 8, 10, 19]

La búsqueda binaria consiste en saber que los números están ordenados y que, por ejemplo, el numero 10 es mayor o menor que un número aleatorio.

El ejemplo anterior, para hacerlo de forma ordenada tomamos el valor del primer índice que es 0 (cero) y el valor del último que es 7, lo sumamos y lo dividimos por dos.

0 + 7 = 7

7 / 2 = 3.5

Python sólo tomará la parte entera del resultado, es decir, el número 3 en este caso. Luego de esto, preguntamos si el valor que se encuentra en el índice 3 es mayor o menor al valor buscado, en este caso, el valor buscado es el número 10.

[2, 3, 5, **7**, 8, 8, 10, 19]

Tenemos que en el índice 3 se encuentra el valor **7**, preguntamos, ¿el valor 7 es mayor o menor que **10**? Como el valor buscado es mayor que 7, los valores de la lista anteriores al índice 3 se descartarán.

[2, 3, 5, **7**, 8, 8, 10, 19]

Quedando

[ 8, 8, 10, 19]

Volvemos hacer el mismo algoritmo anterior, sumamos el índice 4 + 7 = 11 y al resultado lo dividimos por 2.

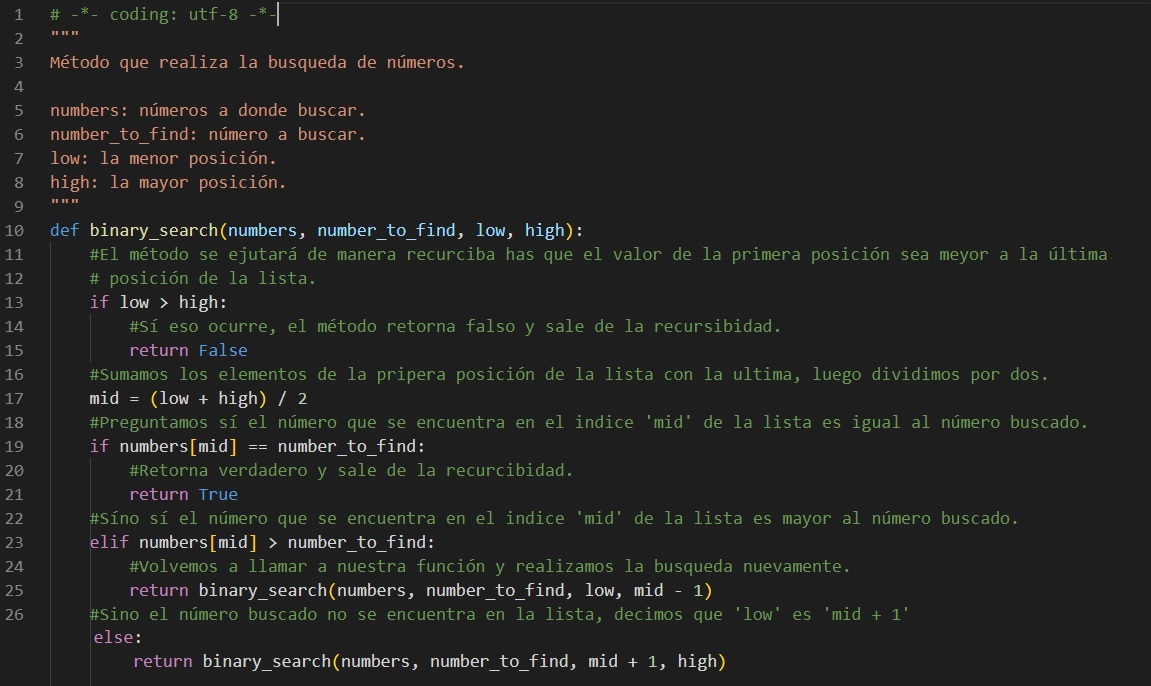
11 / 2 = **5**.5

Preguntamos si el valor del índice 5 es mayor o menor al valor buscado, en caso de ser mayor, descartamos los anteriores valores quedando la lista así.

[10, 19]

Así sucesivamente hasta encontrar el número buscado.

**Algoritmo en python**

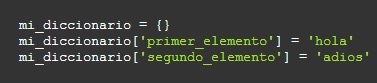


# **Diccionarios en Python**

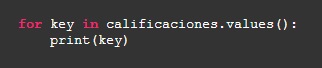
* Un diccionario es un mapa de valores.
* Los valores pueden ser de cualquier tipo.
* Los cuales deben tener una llave.
* Los diccionarios se declaran con (llaves) **{}** o con la función **dict()**.

Cuando iteramos en diccionarios podemos hacerlo a través de las llaves, valores o ítems.

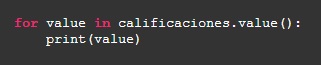
## **Declarar diccionarios y agregar datos**

****

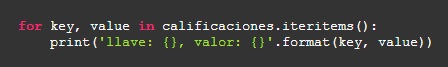
## **Iterar en llaves**



## **Iterar en valores**

****

## **Iterar en ítems**

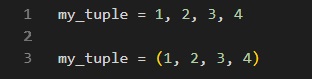
****

Los diccionarios al iterar no muestran sus valores de forma ordenada.

# **Tuplas en Python**

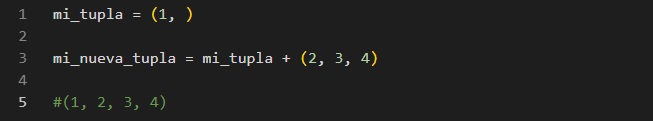
* Las tupas son secuencias de valores indexadas por enteros, lo que las hace diferente de las listas es que estas son inmutables.
* Las tuplas se crean separando los valores por comas
  + Es práctica común encerrar los valores con paréntesis.
* Se pueden utilizar para devolver más de un valor en una función o crear estructuras de datos ligeras.

## **Definir una tupla**



## **Operadores en las tuplas**

* Al igual que los strings si se quiere modificar una tupla se debe crear una nueva.



**Para tener en cuenta**

Si queremos declarar una tupla de la siguiente forma.

**mi\_tupla = (1)**

Cuando aplicamos la función type() para ver su tipo, nos arrojará.

**type(mi\_tupla)**

**<type ‘int’>**

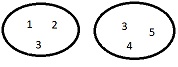
Esto significa que es un tipo **int** y no una **tupla**. Para declarar una tupla de forma correcta se debe de hacer de la siguiente manera:

**mi\_tupla = (1,)**

Si aplicamos nuestra función type() veremos que nos arroja **<type ‘tuple’>**

# **Uso de sets en Python**

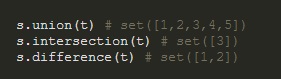
Los sets (conjuntos) son muy similares a las listas, pero estas no permiten elementos repetidos.

Cuando trabajamos con sets, podemos realizar las operaciones básicas de conjuntos, esto quiere decir que se puede calcular los valores de intercepción, diferencia, unión.

## **Declarar sets**



## **Operaciones con sets**

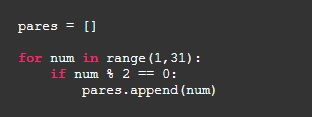


* Tienen 3 tipos de operaciones:
  + **Union**: Muestra los elementos que tienen en común dos sets y si tienen elementos repetidos los coloca una sola vez.
  + **Intersection**: Muestra aquellos elementos que tienen en común ambos sets.
  + **Difference**: Muestra los elementos únicos que tiene un set y que no se encuentra en el otro.

# **Dictionary comprehension - list comprehension**

* Dictionary comprehension y list comprehension nos permite escribir listas o diccionarios de forma más sencilla.
* Convierte en una secuencia en otra.
* Agrega azúcar sintáctica al código.

Encontrar números pares en la forma tradicional.



Esto mismo lo podemos expresar con una list comprehension.



## **Diferencias**

list comprehension: El resultado es una nueva lista.

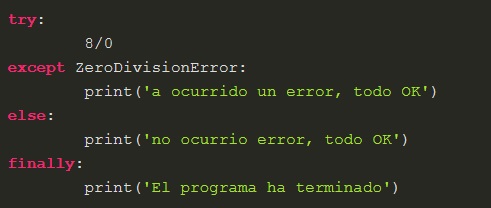
dictionary comprehension: El resultado es sólo un diccionario.

# **Manejo de errores en Python**

Cuando el programa no sabe cómo hacer algo o se traba en un proceso va a arrojar un error.

Es importante poder manejar estos errores y poder manejarlos, en python podemos capturar esos errores con **try**, **except**, **finally**.

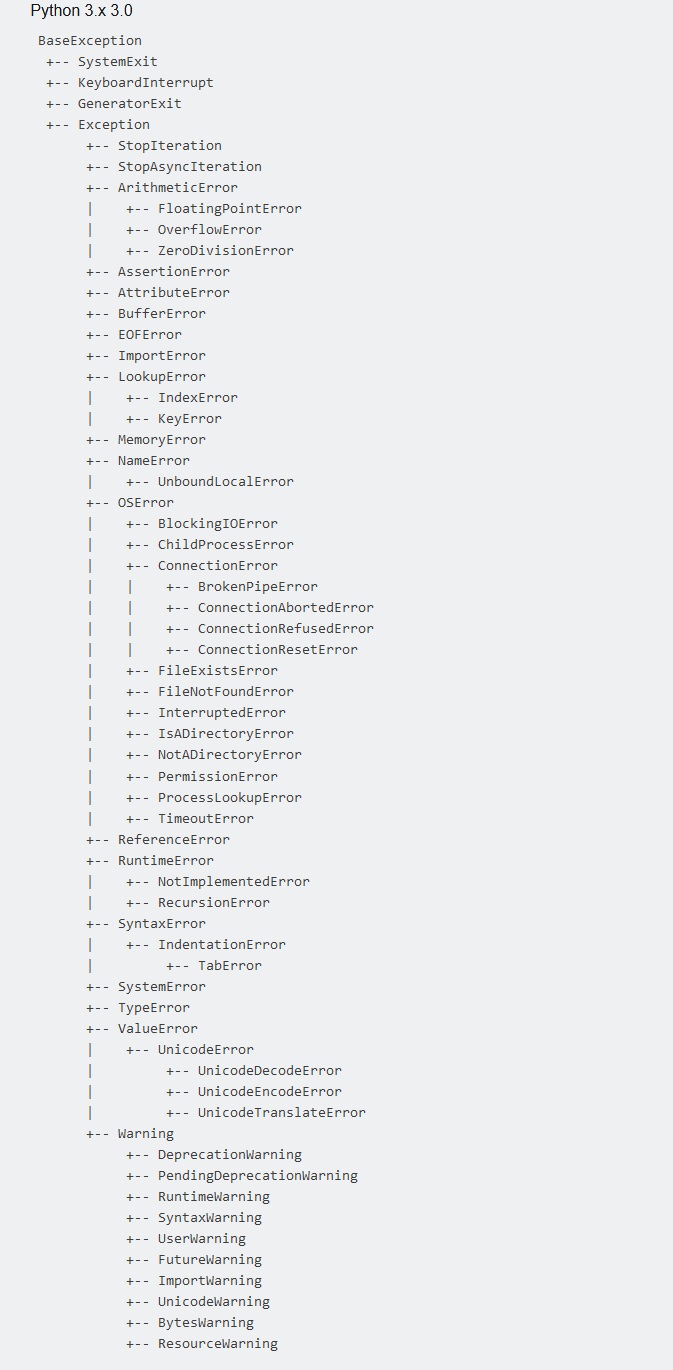
Ejemplo



* **try**: Bloque de código donde puede ocurrir un error.
* **except**: Bloque de código que se ejecuta cuando ocurre un error en el bloque **try**.
* **else**: Bloque de código que se ejecuta si no ocurrió ninguno en el **try**.
* **finally**: Bloque de código que se ejecuta sin importar que halla en los **except** o **else**.

# **Jerarquía de errores en python**

Existen las siguientes jerarquías.



# **Manejo de archivos en Python**

* Con Python también podemos leer y escribir archivos del sistema con la función **open**.
* La función **open** regresa un objeto de tipo **file**.
* Los archivos pueden ser de textos o binarios.

Existen varios modos en los que podemos manejar archivos.

* ‘**r**’ = leer
* ’**w**’ = escribir
* ’**a**’ = añadir
* ’**r+**’ = leer y escribir

Siempre recuerda cerrar el archivo con el método **close**.

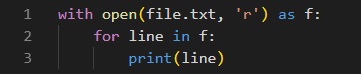
* Si se utiliza el administrador de contexto **with** no es necesario cerrar el archivo, ya que el administrador lo hace sólo.

El keyword **with** nos permite manejar el contexto al trabajar con archivos.

## **Lectura de archivo**

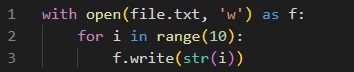
El objeto **file** tiene dos métodos de lecturas:

* read
* readline



## **Escritura del archivo**

Para escribir en un archivo, se utiliza el método **write**.



Si el archivo no existe, python lo creará.

# **Programación orientada a objetos en Python**

Los objetos nos permiten modelar objetos de la vida real, estos objetos van a tener un estado y métodos.

De esta forma vamos a poder definir cada parte de nuestro código modelándolas cómo objetos.

Encapsulación es una de las herramientas más poderosas que tienen los programadores, esta nos permite esconder la complejidad del software.

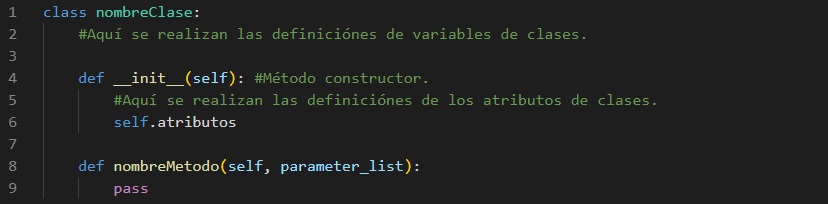
## **Clases y objetos**

* Los objetos tienen atributos que se pueden definir al momento de iniciar un objeto o directamente en la instancia.
* Las clases pueden tener variables de clase, variable de instancia y variables locales.
* Python no tiene un concepto de variables privadas integrado al lenguaje, por convención se definen con un (\_) guion bajo.
* **isinstance**: Método que ayuda a saber si objeto pertenece a una instancia de clase en especifica.
* **hasattr**: Método que ayuda a saber si un objeto contiene un atributo especifico.

## **Clases y métodos**

* Los métodos son como funciones que tienen sentido únicamente en el ámbito de la clase.
* Al igual que las variables, los métodos se definen con un guion bajo.
* La encapsulación es un concepto clave de la programación orientada a objetos.
  + La forma práctica de aplicar este principio, es declarar todas las variables y métodos como privados, a menos que sea necesario exponerlos a otros programadores.
* Un método clave que casi todas las clases deben tener es **\_\_init\_\_**
* Otro es el **\_\_str\_\_**
* Existen varios tipos de métodos:
  + Estáticos.
  + De clase.
  + De instancia.
  + Getters and setters.

## **Definición de clase**



Las **variables de clase** y **los atributos** se referencian con keyword **self** (hace referencia a su propia clase, **this** en otros lenguajes).

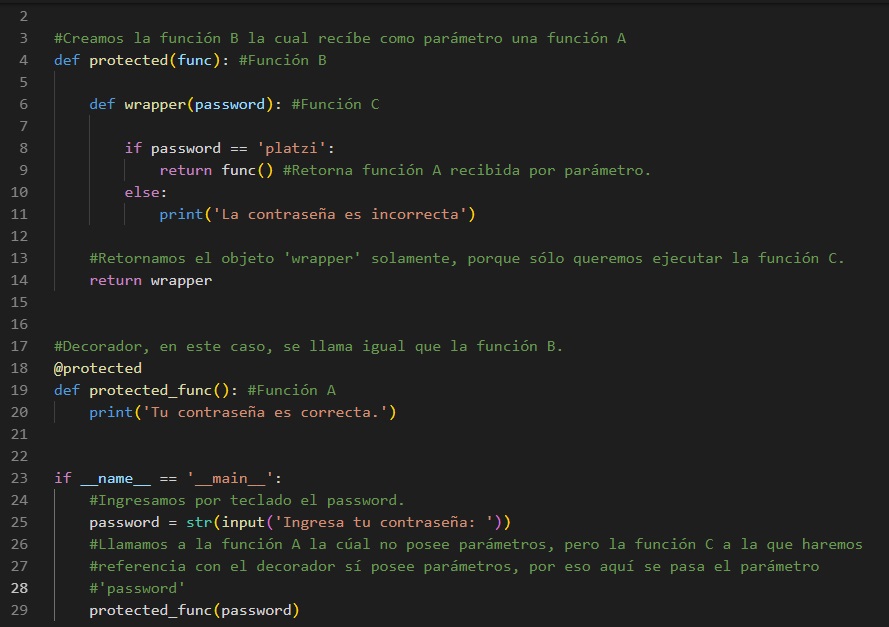
Ejemplo:

* self.nombre\_variable
* self.nombre\_atributo

# **Decoradores en Python**

* Es una función **A** que recibe otra función **B** y regresa una tercera función **C**.
* Para reconocer un decorador, puedes ver que tiene un **@** sobre la declaración de una función.
* Súper útil para definir si una función debe o no ejecutarse. Por ejemplo, en servidores web, existen ciertas funciones que nada más se deben ejecutar sólo si un usuario se encuentra autenticado.
* Un decorador agrega mayor funcionalidad a una función en concreta.
* **A**, **B**, **C** son funciones.
* **A** recibe como parámetro **B** para poder crear **C**.
  + Si envío como parámetro una función es porque necesito que esa función se le agreguen más cosas (ya sea antes o después de su ejecución).
  + Dentro de **nueva\_funcion()** se va a encargar de ejecutar a la función que recibe como parámetro.
  + En la **nueva\_funcion()** debe recibir los parámetros definidos de la función decorada.
  + El **\*args** indica que la función puede recibir **n** cantidad de argumentos (o parámetros) posicionales.
  + El **\*\*kwargs** indica que la función puede recibir **n** cantidad de argumentos (o parámetros) **no posicionales** sino para un diccionario de argumentos (llave-valor).
  + La palabra **args** y **kwargs** se utilizan por convención. Lo realmente importante es definir **\*** y **\*\***.
  + Un ejemplo de código a agregar antes es, abrir una conexión a la BD y el código después es, cerrar la conexión a la BD.
  + En un decorador se pueden enviar parámetros. Para esto, todo el código actual debe estar en una nueva función, cambiar el nombre y en vez de retornar la **nueva\_funcion()** se retornaría la función que tiene el código de todo el decorador.

Ejemplo:



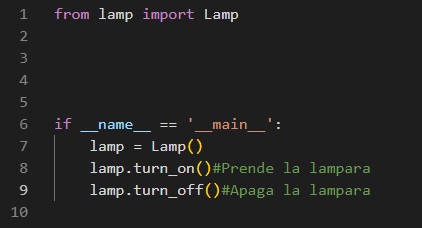
# **Paquetes y módulos en python**

* Un módulo permite agrupar funcionalidad en un solo archivo y ocultar la complejidad.
* Para poder requerir un módulo podemos utilizar **import**.
* Cuando varios módulos agrupan funcionalidades comunes, se pueden agrupar, a su vez, en paquetes.
  + Python reconoce que un directorio es un paquete porque contiene un archivo llamado **\_\_init\_\_.py** de lo contrario no será reconocido como tal.

Cuando python importa un módulo siempre lo importa con el nombre del archivo con excepción si el archivo que se está ejecutando es el punto de entrada, es decir, dentro de un proyecto debe de a ver un módulo llamado **main.py** que importe los módulos necesarios para su correcto funcionamiento seguido de la siguiente línea.



Ejemplo:



En la imagen anterior importamos desde el módulo **lamp** la clase **Lamp**. Debajo de la línea 6 instanciamos la clase **Lamp()** e hicimos usos de sus métodos.

# **Entorno virtual en Python**

* En Python la comunidad comparte su código usando **PyPi** (python package index), es un repositorio para instalar módulos de la comunidad.
* Debemos instalar la herramienta **pip** que nos permitirá poder descargar paquetes.
* Con **pip install [nombre]** se puede instalar el paquete que deseas.
* Podemos utilizar **requirements.txt** para ordenar los paquetes que requiere tu proyecto.
* Ambiente virtual.
* Nos permite encapsular un proyecto para poder instalar las versiones de los paquetes que se requieran sin tenerlos que instalar en todo el sistema operativo.

Ambientes virtuales

Es una buena práctica crear un ambiente virtual por cada proyecto de python en el que se trabaje. Esto evitará conflictos de paquetes en el intérprete principal.

## **Instalación**

* pip install virtalenv

## **Crear un entorno virtual**

Dentro de la carpeta de tu proyecto ejecutas



## **Encender un entorno virtual en Linux**

****

## **En windows**



## **Ver las dependencias instaladas en el entorno virtual**

****

**Instalar dependencias del archivo requirements**

# **¿Qué es web scraping?**

Es una herramienta que permite visitar sitios web, analizar su contenido, obtener y descargar información. Al software programado para scrapear se le suele llamar **bot**, **spider** o **crawler**.

## **Librerías que se utilizan**

Dentro de las librerías que se encuentran para este tipo de herramientas, las más utilizadas son:

* **requests**: Nos permite hacer solicitudes a internet y podemos traer el contenido HTML de diferentes sitios web, no sólo HTML sino **cualquier** contenido que se pueda transferir a través del protocolo http, por ejemplo, archivos con formatos .json, archivos con formatos .xml, etc.
* **BeautifulSoup**: Nos permite parsear ese texto de HTML, es decir, obtener el string de HTML de esta forma poderlo modificar y acceder a él como si fuera un árbol. Esto no va a permitir obtener una serie de herramientas para obtener información del sitio.

# **Qué puedes construir con Python**

* Desarrollo web.
* Ciencias de datos.
* Machine learning e Inteligencia artificial.
* Programas de escritorio.
* Líneas de comandos.
* Internet de las cosas.
* Web scripting y bots.
* Criptografía.
* Finanzas.
* Biología computacional.
* Ciencias exactas y academias.