# **Guía de instalación y conceptos básicos**

## **¿Qué es Python?**

Python es un lenguaje de programación creado por [Guido Van Rossum](https://en.wikipedia.org/wiki/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.

## **Instalación**

Existen dos versiones de Python que tienen gran uso actualmente, Python 2.x y Python 3.x, para este curso necesitas usar una versión 3.x

Para instalar Python solo debes seguir los pasos dependiendo del sistema operativo que tengas instalado.

### **Windows**

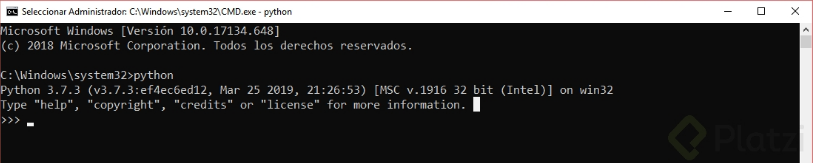
Para instalar Python en Windows ve al [sitio oficial](https://www.python.org/downloads/) y presiona sobre el botón **Download** Python 3.X

Se descargará un archivo de instalación con el nombre python-3.X.exe , ejecútalo. Y sigue los pasos de instalación.

Al finalizar la instalación haz lo siguiente para corroborar una instalación correcta

1. Presiona las teclas **Windows + R** para abrir la ventana de **Ejecutar**.
2. Una vez abierta la ventana **Ejecutar** escribe el comando **cmd** y presiona **ctrl+shift+enter** para ejecutar una línea de comandos con permisos de administrador.
3. Windows te preguntará si quieres abrir el Procesador de comandos de Windows con permisos de administrador, presiona **sí**.
4. En la línea de comandos escribe **python**.

Tu consola se mostrará así.



### **MacOS**

La forma sencilla es tener instalado [homebrew](https://brew.sh/) y usar el comando:

**Para instalar la Versión 2.7**



Para instalar la Versión 3.x



### **Linux**

Generalmente Linux ya lo trae instalado, para comprobarlo puedes ejecutar en la terminal el comando

**Versión 2.7**



Versión 3.x



Si el comando arroja un error quiere decir que no lo tienes instalado, en ese caso los pasos para instalarlo cambian un poco de acuerdo con la distribución de linux que estés usando. Generalmente el gestor de paquetes de la distribución de Linux tiene el paquete de Python

**Si eres usuario de Ubuntu o Debian por ejemplo puedes usar este comando para instalar la versión 3.1**:



**Si eres usuario de Red Hat o Centos por ejemplo puedes usar este comando para instalar python**



## **Tipos de datos en Python**

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

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

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

ejemplo: True, False

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

ejemplos: “Hola”, “¿Cómo estás?”

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

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

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

ejemplo: {“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.

ejemplos: (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 las defines con def junto a un nombre y unos paréntesis que reciben los parámetros a usar. Terminas con dos puntos.

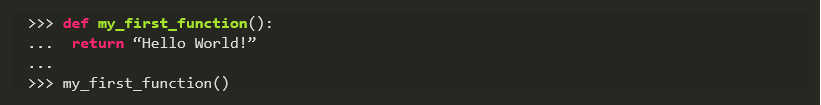
def nombre\_de\_la\_función(parametros):

Después por indentación colocas los datos que se ejecutarán desde la función:

Las funciones las defines con def junto a un nombre y unos paréntesis que reciben los parámetros a usar. Terminas con dos puntos.

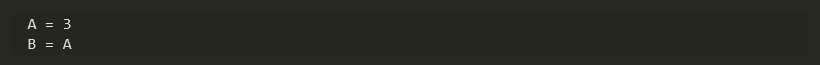
def nombre\_de\_la\_función(**parametros**):

Después por indentación colocas los datos que se ejecutarán desde la función:



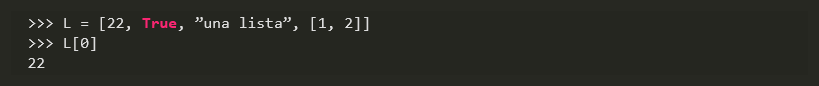
## **Variables**

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.



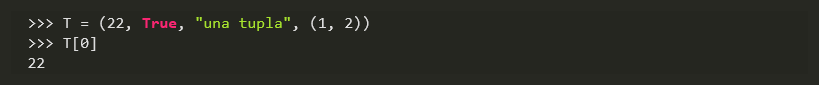
## **Listas**

Las listas las declaras con corchetes. Estas pueden tener una lista dentro o cualquier tipo de dato.



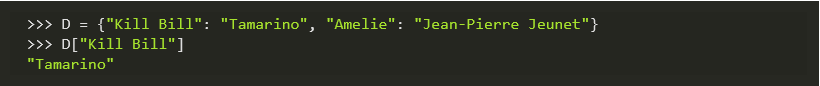
## **Tuplas**

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



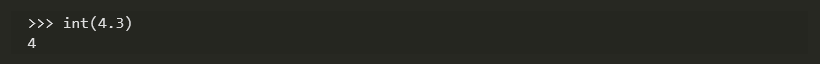
## **Diccionarios**

En los diccionarios tienes un grupo de datos con un formato: la primera cadena o número será la clave para acceder al segundo dato, el segundo dato será el dato al cual accederás con la llave. Recuerda que los diccionarios son listas de **llave**:**valor**.

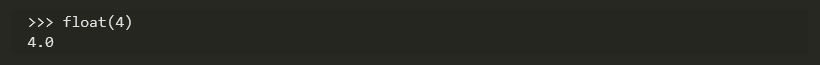


## **Conversiones**

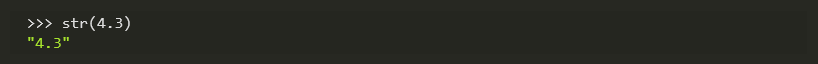
De flotante a entero:



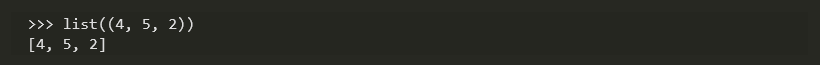
De entero a flotante:



De entero a string:

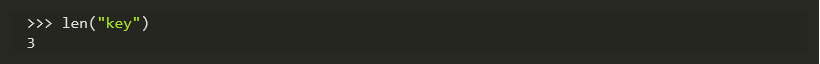


De tupla a lista:

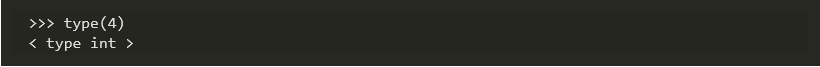


## **Operadores Comunes**

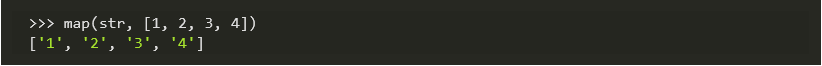
Longitud de una cadena, lista, tupla, etc.:



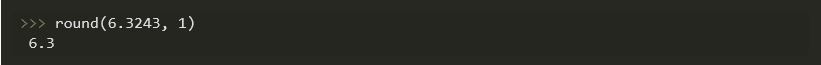
Tipo de dato:



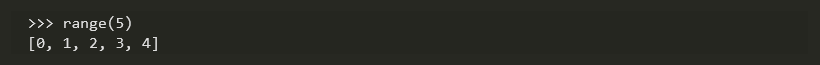
Aplicar una conversión a un conjunto como una lista:



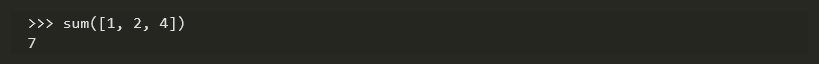
Redondear un flotante con x número de decimales:



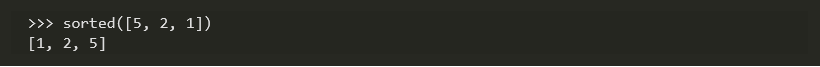
Generar un rango en una lista (esto es mágico):



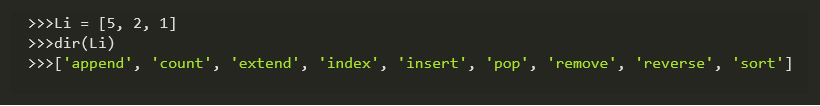
Sumar un conjunto:



Organizar un conjunto:

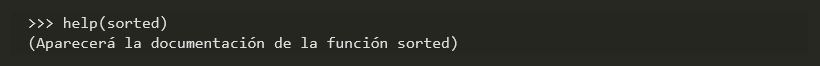


Conocer los comandos que le puedes aplicar a x tipo de datos:



‘**append’**, ‘**count’**, ‘**extend’**, ‘**index’**, ‘**insert’**, ‘**pop’**, ‘**remove’**, ‘**reverse’**, ‘**sort’** son posibles comandos que puedes aplicar a una lista.

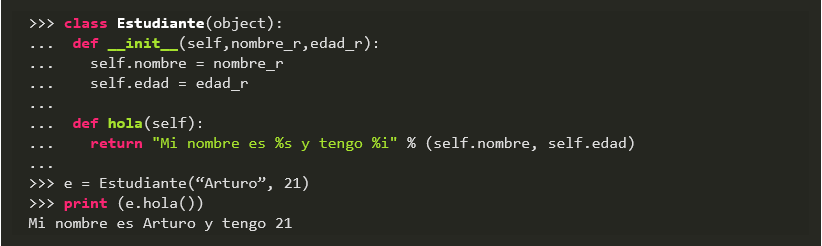
Información sobre una función o librería:



## **Clases**

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 pones entre paréntesis.

Para crear un constructor haces 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 utilizas los operadores de comparación para comprobar si tu objeto es menor, mayor o igual al objeto pasado como parámetro.

**len**(self)

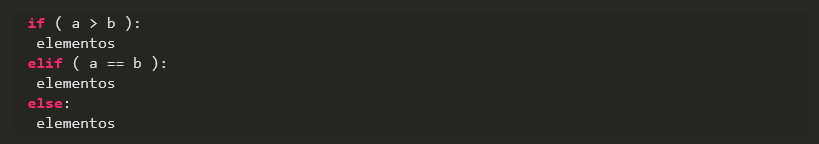
Método llamado para comprobar la longitud del objeto. Lo usas, por ejemplo, cuando llamas la función len(obj) sobre nuestro código. Como es de suponer el método te debe devolver la longitud del objeto.

**init**(self,otro)

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

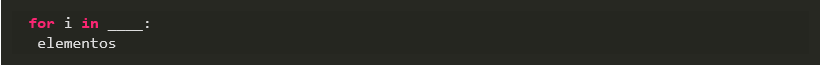
## **Condicionales IF**

Los condicionales tienen la siguiente estructura. Ten en cuenta que lo que contiene los paréntesis es la comparación que debe cumplir para que los elementos se cumplan.

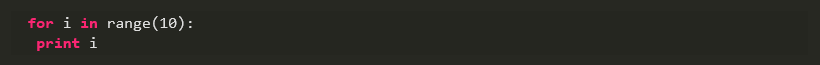


## **Bucle FOR**

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:



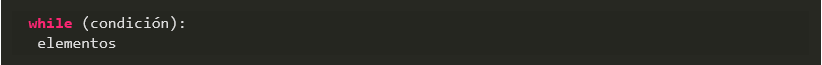
Ejemplo:



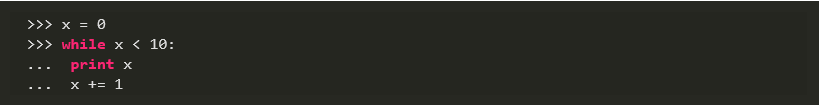
En este caso recorrerá una lista de diez elementos, es decir el **print i** de ejecutar diez veces. Ahora **i** va a tomar cada valor de la lista, entonces este **for** imprimirá los números del 0 al 9 (recordar que en un **range** vas hasta el número puesto **-1**).

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 cierta. La estructura de un while es la siguiente:



Ejemplo:



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

# **¿Qué es la programación?**

Python es uno de los lenguajes más emocionantes de la actualidad y puedes lograr muchas cosas con él. Este curso te va a servir como una introducción al lenguaje.

## **¿Qué es la programación?**

Es una disciplina que combina parte de otras disciplinas como las Matemáticas, Ingeniería y la Ciencia. Sin embargo, la habilidad más importante es resolver problemas. Es lo que harás todos los días como programador o programadora.

La programación es una secuencia de instrucciones que le damos a la computadora para que haga lo que nosotros deseamos. Podemos construir una aplicación web, móvil, un programa que lleve cohetes a la luna o marte, resolver problemas de finanzas.

La estructura de un programa. Casi todos los programas tienen un input, output, operaciones matemáticas, ejecución condicional y repeticiones.

# **¿Por qué programar con Python?**

Python es uno de los mejores lenguajes para principiantes porque tiene una sintaxis clara, una gran comunidad y esto hace que el lenguaje sea muy amigable para los que están iniciando.

Python está diseñado para ser fácil de usar, a diferencia de otros lenguajes donde la prioridad es ser rápido y eficiente. Python no es de los lenguajes más rápidos, pero casi nunca importa.

Es el tercer lenguaje, según Github, entre los más populares. En StackOverflow se comenta que es uno de los lenguajes que mayor popularidad está obteniendo.

““**Python cuando podamos, C++ cuando necesitemos**””

* **python --version** para conocer la versión que tenemos instalada.
* **python [nombre del archivo]** para ejecutar nuestro programa.

# **Operadores matemáticos**

En programación estos operadores son muy similares a nuestras clases básicas de matemáticas.

* **//**: Es división de entero, básicamente tiramos la parte decimal.
* **%**: Es el residuo de la división, lo que te sobra.
* **\*\***: Exponente.

Los operadores son contextuales, dependen del tipo de valor. Un valor es la representación de una entidad que puede ser manipulada por un programa.

Podemos conocer el tipo del valor con type() y nos devolverá algo similar a **<class 'init'>**, **<class 'float'>**, **<class 'str'>**. Dependiendo del tipo los operadores van a funcionar de manera diferente.

# **Variables y expresiones**

Una variable **es simplemente el contenedor de un valor**. Es una forma de decirle a la computadora de que nos guarde un valor para luego usarlo.

Python es un lenguaje **dinámico**, este concepto de privado y público se genera por convenciones del lenguaje. En programación el signo **=** significa **asignación**.

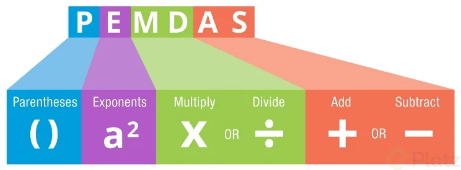
Si una variable está en mayúscula, usualmente se refiere a una **constante**, no debería reasignarse. Es una convención.

Reglas de Variables:

* Pueden contener números y letras.
* No deben comenzar con número.
* Múltiples palabras se unen con \_
* No se pueden utilizar palabras reservadas.

Expresiones son instrucciones para el intérprete para evaluar la expresión. Los enunciados tienen efectos dentro del programa, como print que genera un **output**.

**PEMDAS** = Paréntesis, Exponente, Multiplicación-División, Adición-Sustracción



# **Funciones**

En el contexto de la programación las funciones son simplemente una agrupación de enunciados(statments) que tienen un nombre. Una función tiene un nombre, debe ser descriptivo, puede tener parámetros y puede regresar un valor después que se generó el cómputo.

Python es un lenguaje que se conoce como batteries include (baterías incluidas) esto significa que tiene una librería estándar con muchas funciones y librerías.

Para declarar funciones que no son las globales, las built-in functions, necesitamos importar un módulo.

Con el keyword def declaramos una función.

# **Operadores lógicos**

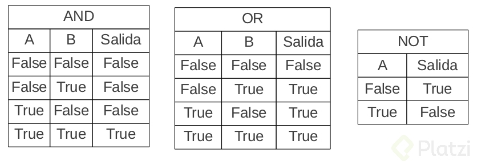
Para comprender el flujo de nuestro programa debemos entender un poco sobre estructuras y expresiones booleanas

* **==** se refiere a igualdad.
* **!=** no hay igualdad.
* **>** mayor que.
* **<** menor que.
* **>=** mayor o igual.
* **<=** menor o igual.

**and** únicamente es verdadero cuando **ambos** valores **son verdaderos**.

**or** es verdadero cuando **uno** de **los dos valores** es **verdadero**.

**not** es lo **contrario** al valor. Falso es Verdadero. Verdadero es Falso.



# **Estructuras condicionales**

En Python **es importante la indentación**, de esa manera identifica donde empieza y termina un bloque de código sin necesidad de llaves **{}** como en otros lenguajes.

# **Strings en Python**

Los **strings** o **cadenas de textos** tienen un comportamiento distinto a otros tipos como los booleanos, enteros, floats. Las cadenas son secuencias de caracteres, todas se pueden acceder a través de un índice.

Podemos saber la longitud de un string, cuántos caracteres se encuentran en esa secuencia. Lo podemos saber con la **built-in function** global llamada **len**.

Algo importante a tener en cuenta cuando hablamos de strings es que estos son inmutables, esto significa que cada vez que modificamos uno estamos generando un nuevo objeto en memoria.

El índice de la primera letra es 0, en la programación se empieza a contar desde 0

|  |  |
| --- | --- |
| Método | Descripción |
| [capitalize()](https://www.w3schools.com/python/ref_string_capitalize.asp) | Convierte el primer carácter en mayúsculas |
| [casefold()](https://www.w3schools.com/python/ref_string_casefold.asp) | Convierte una cadena en minúsculas |
| [center()](https://www.w3schools.com/python/ref_string_center.asp) | Devuelve una cadena centrada |
| [count()](https://www.w3schools.com/python/ref_string_count.asp) | Devuelve el número de veces que un valor especificado se produce en una cadena |
| [encode()](https://www.w3schools.com/python/ref_string_encode.asp) | Devuelve una versión codificada de la cadena |
| [endswith()](https://www.w3schools.com/python/ref_string_endswith.asp) | Devuelve true si los extremos de cadena con el valor especificado |
| [expandtabs()](https://www.w3schools.com/python/ref_string_expandtabs.asp) | Establece el tamaño de la pestaña de la cadena |
| [find()](https://www.w3schools.com/python/ref_string_find.asp) | Busca la cadena de un valor especificado y devuelve la posición de donde fue encontrado |
| [format()](https://www.w3schools.com/python/ref_string_format.asp) | Formatos especifican los valores de una serie |
| [format\_map()](https://www.programiz.com/python-programming/methods/string/format_map) | Formatos especifican los valores de una serie |
| [index()](https://www.w3schools.com/python/ref_string_index.asp) | Busca la cadena de un valor especificado y devuelve la posición de donde fue encontrado |
| [isalnum()](https://www.w3schools.com/python/ref_string_isalnum.asp) | Devuelve verdadero si todos los caracteres de la cadena son alfanuméricos |
| [isalpha()](https://www.w3schools.com/python/ref_string_isalpha.asp) | Devuelve True si todos los caracteres de la cadena están en el alfabeto |
| [isdecimal()](https://www.w3schools.com/python/ref_string_isdecimal.asp) | Devuelve True si todos los caracteres de la cadena son decimales |
| [isdigit()](https://www.w3schools.com/python/ref_string_isdigit.asp) | Devuelve verdadero si todos los caracteres de la cadena son dígitos |
| [isidentifier()](https://www.w3schools.com/python/ref_string_isidentifier.asp) | Devuelve True si la cadena es un identificador |
| [islower()](https://www.w3schools.com/python/ref_string_islower.asp) | Devuelve verdadero si todos los caracteres de la cadena son minúsculas |
| [isnumeric()](https://www.w3schools.com/python/ref_string_isnumeric.asp) | Devuelve verdadero si todos los caracteres de la cadena son numéricos |
| [isprintable()](https://www.w3schools.com/python/ref_string_isprintable.asp) | Devuelve verdadero si todos los caracteres de la cadena son imprimibles |
| [isspace()](https://www.w3schools.com/python/ref_string_isspace.asp) | Devuelve True si todos los caracteres de la cadena son espacios en blanco |
| [istitle()](https://www.w3schools.com/python/ref_string_istitle.asp) | Devuelve True si la cadena sigue las reglas de un título |
| [isupper()](https://www.w3schools.com/python/ref_string_isupper.asp) | Devuelve True si todos los caracteres de la cadena son mayúsculas |
| [join()](https://www.w3schools.com/python/ref_string_join.asp) | Se une a los elementos de un iterable al final de la cadena |
| [ljust()](https://www.w3schools.com/python/ref_string_ljust.asp) | Devuelve una versión justificada izquierda de la cadena |
| [lower()](https://www.w3schools.com/python/ref_string_lower.asp) | Convierte una cadena en minúsculas |
| [lstrip()](https://www.w3schools.com/python/ref_string_lstrip.asp) | Devuelve una versión de ajuste izquierdo de la cuerda |
| [maketrans()](https://www.programiz.com/python-programming/methods/string/maketrans) | Devuelve una tabla de traducción para ser utilizado en traducciones |
| [partition()](https://www.w3schools.com/python/ref_string_partition.asp) | Devuelve una tupla donde la cadena se separó en tres partes |
| [replace()](https://www.w3schools.com/python/ref_string_replace.asp) | Devuelve una serie en un valor especificado es reemplazado con un valor especificado |
| [rfind()](https://www.w3schools.com/python/ref_string_rfind.asp) | Busca la cadena de un valor especificado y devuelve la última posición de donde fue encontrado |
| [rindex()](https://www.w3schools.com/python/ref_string_rindex.asp) | Busca la cadena de un valor especificado y devuelve la última posición de donde fue encontrado |
| [rjust()](https://www.w3schools.com/python/ref_string_rjust.asp) | Devuelve una versión justificada derecha de la cadena |
| [rpartition()](https://www.w3schools.com/python/ref_string_rpartition.asp) | Devuelve una tupla donde la cadena se separó en tres partes |
| [rsplit()](https://www.w3schools.com/python/ref_string_rsplit.asp) | Divide la cadena en el separador especificado y devuelve una lista |
| [rstrip()](https://www.w3schools.com/python/ref_string_rstrip.asp) | Devuelve una versión ajuste correcto de la cadena |
| [split()](https://www.w3schools.com/python/ref_string_split.asp) | Divide la cadena en el separador especificado y devuelve una lista |
| [splitlines()](https://www.w3schools.com/python/ref_string_splitlines.asp) | Divide la cadena en los saltos de línea y devuelve una lista |
| [startswith()](https://www.w3schools.com/python/ref_string_startswith.asp) | Devuelve true si la cadena comienza con el valor especificado |
| [strip()](https://www.w3schools.com/python/ref_string_strip.asp) | Devuelve una versión recortada de la cadena |
| [swapcase()](https://www.w3schools.com/python/ref_string_swapcase.asp) | Permutas de los casos, se convierte en minúsculas mayúsculas y viceversa |
| [title()](https://www.w3schools.com/python/ref_string_title.asp) | Convierte el primer carácter de cada palabra en mayúsculas |
| [translate()](https://www.tutorialspoint.com/python/string_translate.htm) | Devuelve una cadena traducida |
| [upper()](https://www.w3schools.com/python/ref_string_upper.asp) | Convierte una cadena en mayúsculas |
| [zfill()](https://www.w3schools.com/python/ref_string_zfill.asp) | Rellena la cadena con un número determinado de valores de 0 a principios |

# **Operaciones con Strings en Python**

Los strings tienen varios métodos que nosotros podemos utilizar.

* **upper**: convierte todo el string a mayúsculas.
* **lower**: convierte todo el string a minúsculas.
* **find**: encuentra el índice en donde existe un patrón que nosotros definimos.
* **startswith**: significa que empieza con algún patrón.
* **endswith**: significa que termina con algún patrón.
* **capitalize**: coloca la primera letra en mayúscula y el resto en minúscula.

**in** y **not in** nos permite saber con cualquier secuencia sin una subsecuencia o substrings se encuentra adentro de la secuencia mayor.

**dir**: Nos dice todos los métodos que podemos utilizar dentro de un objeto.

**help**: Nos imprime en pantalla el docstrings o comentario de ayuda o instrucciones que posee la función. Casi todas las funciones en Python las tienen.

# **Operaciones con strings: Slices en python**

Los slices en Python nos permiten manejar secuencias de una manera poderosa.

Slices en español significa ““rebanada””, si tenemos una secuencia de elementos y queremos una rebanada tenemos una sintaxis para definir qué pedazos queremos de esa secuencia.



# **For loops**

Las iteraciones es uno de los conceptos más importantes en la programación. En Python existen muchas maneras de iterar, pero las dos principales son los for loops y while loops.

Los **for loops** nos permiten iterar a través de una secuencia y los **while loops** nos permiten iterara hasta cuando una condición se vuelva falsa.

for loops:

* Tienen dos keywords **break** y **continue** que nos permiten salir anticipadamente de la iteración
* Se usan cuando se quiere ejecutar varias veces una o varias instrucciones.
* for [**variable**] in [**secuencia**]:

Es una convención usar la letra i como variable en nuestro for, pero podemos colocar la que queramos.

**range**: Nos da un objeto rango, es un iterador sobre el cual podemos generar secuencias.

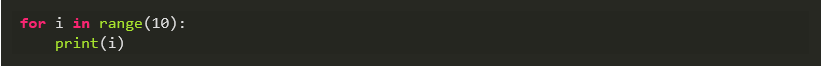
# **While loops**

Al igual que las **for loops**, las **while loop**s nos sirve para iterar, pero las for loops nos sirve para iterar a lo largo de una secuencia mientras que las while loops nos sirve para iterar mientras una condición sea verdadera.

Si no tenemos un mecanismo para convertir el mecanismo en falsedad, entonces nuestro while loops se ira al infinito (infinite loop).

# **Iterators and generators**

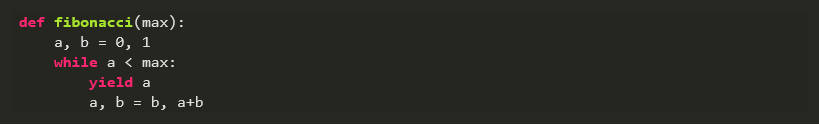
Aunque no lo sepas, probablemente ya utilices iterators en tu vida diaria como programador de Python. Un iterator es simplemente un objeto que cumple con los requisitos del Iteration Protocol (protocolo de iteración) y por lo tanto puede ser utilizado en ciclos. Por ejemplo.



En este caso, la función **range** es un iterable que regresa un nuevo valor en cada ciclo. Para crear un objeto que sea un iterable, y por lo tanto, implemente el protocolo de iteración, debemos hacer tres cosas:

* Crear una clase que implemente los métodos **iter** y **next**
* **iter** debe regresar el objeto sobre el cual se iterará.
* **next** debe regresar el siguiente valor y aventar la excepción StopIteration cuando ya no haya elementos sobre los cual iterar.

Por su parte, los generators son simplemente una forma rápida de crear iterables sin la necesidad de declarar una clase que implemente el protocolo de iteración. Para crear un generator simplemente declaramos una función y utilizamos el keyword **yield** en vez de **return** para regresar el siguiente valor en una iteración. Por ejemplo:



Es importante recalcar que una vez que se ha agotado un generator ya no podemos utilizarlo y debemos crear una nueva instancia. Por ejemplo:

