Notas de Python 3



Índice

[El Zen de python](#_1qherppr8w7y)

[La Biblia de python](#_7vylmqp0dyuc)

[Básicos del Lenguaje](#_onw8saw102mc)

[Operadores matemáticos](#_x0pr8o2d1eie)

[Conversiones](#_gypezs3xfyyk)

[Tipos de datos en Python](#_52xajtwytfqj)

[Variables y expresiones](#_hxqkrl1jf4y1)

[# -\*- coding: utf-8 -\*-](#_eg9xk7kz2ssy)

[¿Por qué poner if \_\_name\_\_ == '\_\_main\_\_': ?](#_v9pz7gshmiuo)

[Funciones en Python](#_l47j7lw72y5y)

[¿Qué es pass?](#_u4mcwc5a8r1h)

[Operadores Comunes (funciones)](#_qcgw2b2mdu14)

[Conocer los comandos que le puedes aplicar a x tipo de datos:](#_wrr2jnjjt03h)

[Clases](#_rgwlpnvflkci)

[Métodos especiales](#_m63exmab4yyx)

[Condicionales IF](#_c05sin3q4349)

[Bucle FOR](#_s4comxog7zwa)

[Bucle WHILE](#_pgtthivsmhcp)

[Uso de strings y ciclos](#_3wreso6z0lsa)

[Mutabilidad Inmutabilidad y Referencia](#_ts4exy9fd7m)

[Operaciones con strings](#_f6tv03ondrhe)

[Crear un help() para nuestras funciones](#_3hywdhabk6ud)

[Slices en python](#_pr0xuseprgdz)

[Métodos join() y split() en Python](#_d51kciph57k)

[For loops](#_gsp4hdbmky1x)

[While loops](#_pgf6ppm9bcga)

[Controlar los loop usando Break y Continue](#_ph7aftha0xfc)

[Recursividad](#_7fsg30owssjg)

[Iterators and generators](#_h0bal6f36a6b)

[Estructuras de Datos](#_ac2vgb8d3762)

[Las Listas, Tuplas y operaciones con Listas](#_fzppztyo5vqg)

[Tuplas](#_pt5zx7h0kzjw)

[Operaciones y métodos con listas:](#_6ft42xw8h699)

[Diccionarios en python](#_vvhw8c351ook)

[Uso de Set’s en python](#_q3i78nbdmhj)

[Métodos para sets](#_tsmclvgyirc3)

[Operaciones con sets](#_6jau4zbdl6ui)

[Dictionary comprehension - list comprehension](#_4jz7gv6kvbqx)

[Uso de objetos y módulos](#_rji0gaz8wuhj)

[Python en el mundo real](#_2y449jghng3d)

[Conclusiones finales](#_nsrofmt1nrmx)

[Clases bonus](#_g4fws4q23q2j)

Si crees que algo no está claro o que lo podrias explicar mejor

**no dudes en corregirlo.**

Si crees que falta algo o que algo se podría ver mejor

**no dudes en añadirlo.**

Somos una comunidad y

**estas notas son tuyas también.**

# El Zen de python

**Hermoso es mejor que feo.**

**Explícito es mejor que implícito.**

**Simple es mejor que complejo.**

**Complejo es mejor que complicado.**

**Plano 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 supera 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!**

## 

## La Biblia de python

Siempre que tengas una duda acerca del lenguaje puedes consultar esta ruta:

1. Ingresa a [python.org](https://www.python.org)
2. Haz click en la sección “[Documentation](https://www.python.org/doc/)”
3. Selecciona la versión de python que quieres explorar, [Python 3](https://docs.python.org/3/)

y como paso adicional, mejora tus notas.

Otras fuentes de información:

* <https://uniwebsidad.com/libros/python?from=librosweb>
* <3 tu corazón es la mejor fuente.

# Básicos del Lenguaje

## Operadores matemáticos

Operadores aritméticos:

* **+ :** Suma simple, ejemplo:
  + 10+5 = 15
* **-** **:** Resta simple, ejemplo:
  + 10-5 = 5
  + -10+5 = -5 (tambien puede haber resultados negativos)
* **\* :** Multiplicación simple, ejemplo:
  + 5\*2 = 10
  + -5\*2 = -10
  + 5\*0 = 0
* **\*\* :** Potencia, elevar a una potencia un número, ejemplo:
  + 5\*\*2 = 5\*5 = 25
  + 10\*\*3 = 10\*10\*10 = 1000
  + 3\*\*10 = 3\*3\*3\*3\*3\*3\*3\*3\*3\*3 = 59049
* **/** **:** División simple, ejemplo:
  + 10/5 = 2
  + 5/10 = 0.5
* **// :** División redondeada, ejemplo:
  + 10//5 = 2
  + 5//10 = 0 (El número se redondeo, 0.5 se redondea a 0)
* **% :** Es una división que devuelve el residuo de la operación, ejemplo:
  + 10%5 = 0 (porque la división es exacta y no hay residuo)
  + 5%10 = 5
  + 11%2 = 1
  + 19/8 = 3

Operadores de comparación(Relacionales)

* **>** : Mayor que, verdadero si el valor de la izquierda es mayor al de la derecha, ejemplo:
  + 1>0 Verdadero
  + -5>-2 Falso
* **<** : Menor que, verdadero si el valor de la izquierda es menor al de la derecha, ejemplo:
  + 1<0 Falso
  + -5<-2 Verdadero
* **==** : Igual a, verdadero si los valores son iguales, ejemplo:
  + 1==1 Verdadero
  + 3==-3 Falso
* **!=** : diferente de, verdadero si los valores son diferentes, ejemplo:
  + 1!=1 Falso
  + 3!=-3 Verdadero
* **>=** : Mayor o igual que, verdadero si el valor de la izquierda es mayor o igual al de la derecha, ejemplo:
  + 1>=0 Verdadero
  + 1>=1 Verdadero
  + 1>=2 Falso
* **<=** : Menor o igual que, verdadero si el valor de la izquierda es menor o igual al de la derecha, ejemplo:
  + 1>=2 Verdadero
  + 1>=1 Verdadero
  + 1>=0 Falso

Operadores de asignación

* **=** : asigna un valor a una variable, ejemplo:
  + a = 5
  + a+5 = 10
  + a+1 = 6
* **+=**
  + a += 5

Es lo mismo que decir:

* + a = a + 5
* **-=**
  + a -= 5

Es lo mismo que decir:

* + a = a - 5
* **\*=**
  + a \*= 5

Es lo mismo que decir:

* + a = a \* 5
* **\*\*=**
  + a \*\*= 5

Es lo mismo que decir:

* + a = a \*\* 5
* **/=**
  + a /= 5

Es lo mismo que decir:

* + a = a / 5
* **//=**
  + a //= 5

Es lo mismo que decir:

* + a = a // 5
* **%=**
  + a %= 5

Es lo mismo que decir:

* + a = a % 5

Más operadores en [AQUÍ](https://www.geeksforgeeks.org/basic-operators-python/)

* Operadores lógicos: and, or, not
* Operadores binarios: &, |, ^, ~, <<, >>
* Membership Operators: in, not in
* Operadores de identidad: is, not is

## Conversiones

De flotante a entero:

>>> int(4.3)

4

De entero a flotante:

>>> float(4)

4.0

De entero a string:

>>> str(4.3)

"4.3"

De tupla a lista:

>>> list((4, 5, 2))

[4, 5, 2]

## 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 estas?”

**Listas**: Son un grupo o array de datos, puede contener cualquiera de los datos anteriores, Las listas las declaras con corchetes. Estas pueden tener una lista dentro o cualquier tipo de dato.

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.

## Variables y expresiones

Una variable es un contenedor de un valor, es una forma el cual le decimos a la computadora

guardame este valor que luego quiero generar operaciones sobre este valor

Tenemos Variables:

- Públicas

- Privadas

- Super Privadas

- Constantes

El signo igual "=" significa asignación.

El signo de guión bajo "\_" en el nombre de una variable indica que es una variable privada.

Una variable escrita en mayúsculas "PI" es una variable constante.

El signo doble guión bajo "\_\_" indica que es una variable que no debe ser modificada porque puede romper el código,

es como un mensaje del programador.

Reglas de los nombres de variables:

- Las variables pueden contener números y letras

- Las variables no pueden con números ni con caracteres especiales

- Las variables con varias palabras debes unirlas con "\_"

- No se pueden utilizar palabras reservadas

Expresión: es una combinación de valores, variables y operadores: 2+2

Enunciado (statement): es una unidad de código que tiene un efecto : age=20

Orden de operaciones (PEMDAS)

- Paréntesis

- Exponente

- Multiplicación

- Division

- Adición

- Sustracción

## # -\*- coding: utf-8 -\*-

Esta línea de código solo se usa en la versión de python 2.7 y tiene una función muy importante cuando se van a usar caracteres que están fuera del lenguaje inglés, un ejemplo de estos caracteres serian:

* á, Á, ñ, ü, í.

“# -\*- coding: utf-8 -\*-” permite que el intérprete de python sepa que estos caracteres van a estar presentes y cómo representarlos.

## ¿Por qué poner if \_\_name\_\_ == '\_\_main\_\_': ?

Esto es porque hay dos formas de ejecutar un programa de python, ejecutándolo directamente y ejecutandolo desde otro programa usando "import"

entonces cuando tu pones en otro código

import Mi\_Programa.py

y tu no pusiste " if \_\_name \_\_ == ‘\_\_main \_\_’: " , se va a ejecutar el programa que hiciste en Mi\_Programa.py y después de que termine de ejecutarse se ejecutará el código en el que importaste el programa.

Sin embargo si sí colocaste " if \_\_name \_\_ == ‘\_\_main \_\_’: " entonces todo lo que esté dentro de ese bloque de código no se ejecutara y así podrás usar funciones específicas de ese programa sin que se ejecute el programa si no solo la función.

Para una explicación mas completa: [https://es.stackoverflow.com/questions/32165/qué-es-if-name-main](https://es.stackoverflow.com/questions/32165/qu%C3%A9-es-if-name-main)

## Funciones en Python

La manera de declarar una función en python es utilizando:

* **def nombre\_de\_la\_funcion(variable, variable2):**
  + código de la función.

Para llamar una función se utiliza el nombre de la función a utilizar y las variables que la función pida dentro del paréntesis, si la función no necesita ninguna variable se dejan los paréntesis vacíos pero aún se colocan. Ejemplo:

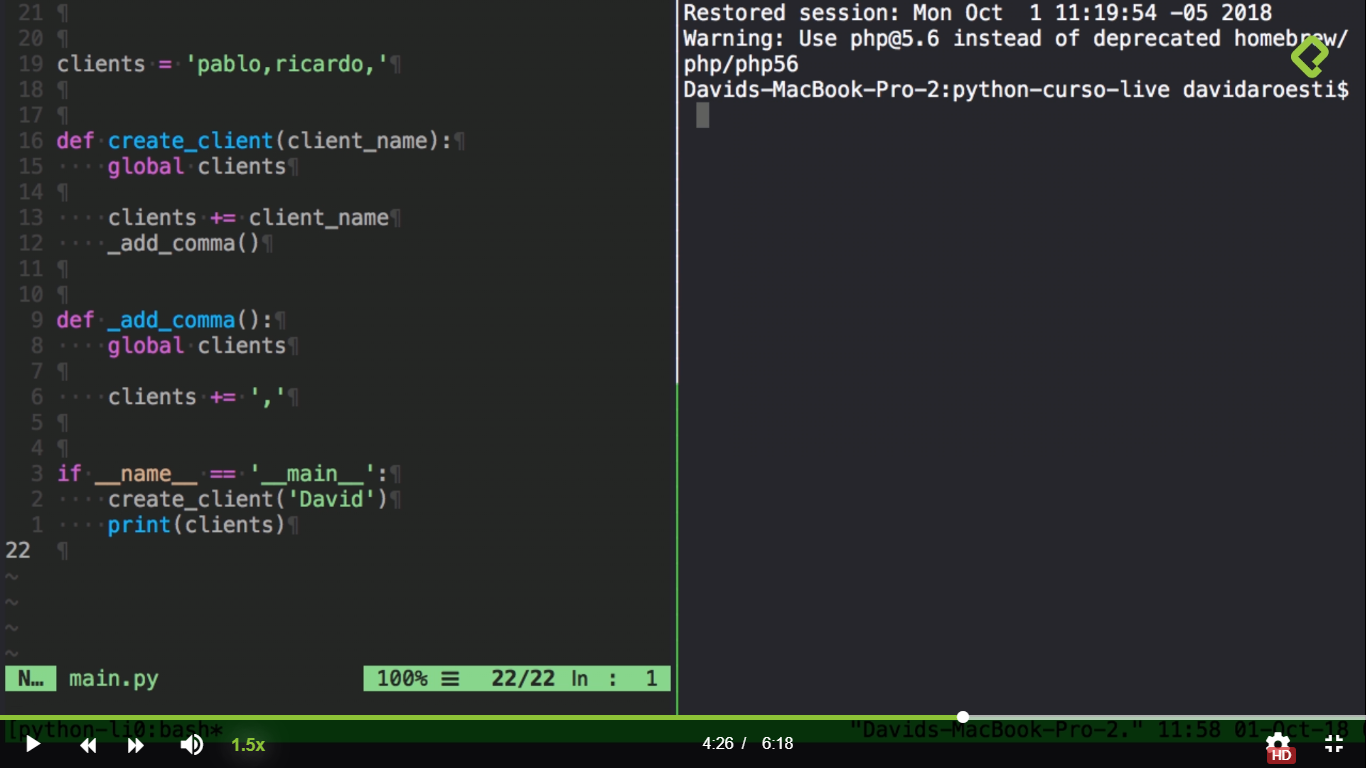
* **nombre\_de\_la\_funcion(1,2)**

Si la función no necesitará parámetros se colocaría:

* **nombre\_de\_la\_funcion()**

**Nota:**

Cuando utilizamos una variable declarada de manera global sin especificarle a la función que es una variable global la función supone que dentro de ella existe una variable local llamada como la variable global que tratamos de usar. Este error solo ocurre cuando tratamos de manipular la variable global, es decir asignar o reasignar; ya que si usamos “ print(clients) ” no se mostraría ningún error. Ejemplo de una variable global y cómo utilizarla dentro de una función:



Se utiliza el comando “global nombre\_de\_la\_variable\_global” para poder utilizarla, esto no se hace con la variable client\_name porque es una variable declarada dentro de la función.

## ¿Qué es pass?

Seguro has visto varias veces esto si estás empezando a aprender python, entonces ¿qué significa?

pass es una keyword muy interesante, es interesante porque no hace nada, así es, no hace nada, se coloca porque no se pueden dejar bloques de códigos vacíos, entonces si tú haces esto:

* if 1=1:
  + pass

No sucederá nada, es como si no hubiera nada dentro del if, si no lo pones entonces el editor de código que estés usando probablemente te de un error o el editor de código lo incluya por ti, por eso, una buena práctica en python es poner pass cuando dejas algo vacio.

## Operadores Comunes (funciones)

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

>>> **len("key")**

3

Tipo de dato:

>>> **type(4)**

< type int >

Aplicar una conversión a un conjunto como una lista:

>>> **map(str, [1, 2, 3, 4])**

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

Redondear un flotante con x número de decimales:

>>> **round(6.3243, 1)**

6.3

Generar un rango en una lista:

>>> **range(5)**

[0, 1, 2, 3, 4]

>>> **range(2,11,2)**

[2, 4, 6, 8, 10]

Enumerar una lista, string, tupla:

>>> **enumerate(‘pedro’)**

0, p

1, e

…

4, o

>>> **enumerate([1,2,3,4,5])**

0, 1

1, 2

…

4, 5

Sumar un conjunto:

>>> **sum([1, 2, 4])**

7

Organizar un conjunto:

>>> **sorted([5, 2, 1])**

[1, 2, 5]

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

>>>Li = [5, 2, 1]

>>>dir(Li)

>>>['append', 'count', 'extend', 'index', 'insert', 'pop', 'remove', 'reverse', 'sort']

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

Información sobre una función o librería:

>>> help(sorted)

(Aparecerá la documentación de la función sorted)

## 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:

>>> class Estudiante(object):

... def \_\_init\_\_(self,nombre\_r,edad\_r):

... self.nombre = nombre\_r

... self.edad = edad\_r

...

... def hola(self):

... return "Mi nombre es %s y tengo %i" % (self.nombre, self.edad)

...

>>> e = Estudiante(“Arturo”, 21)

>>> print e.hola()

Mi nombre es Arturo y tengo 21

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(a,b)

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

Returns:

-1 if a<b

0 if a=b

1 if a>b

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.

platzi length is 6

[1, 2, 3] length is 3

(1, 2, 3) length is 3

Length of range(1, 10) is 9

init(self,otro)

Es un constructor de nuestra clase, es decir, es un “método especial” que es llamado 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.

if a > b :

elementos

elif a == b :

elementos

else:

elementos

## 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:

for i in \_\_\_\_:

elementos

Ejemplo:

for i in range(10):

print i

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:

while (condición):

elementos

Ejemplo:

>>> x = 0

>>> while x < 10:

... print x

... x += 1

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.

# Uso de strings y ciclos

En python las cadenas(strings), son secuencias a las que se pueden

acceder a través de un índice

***cadena = "manzana"***

***cadena[0] = "m"***

El índice de la primera letra es 0

Para conocer la longitud de un string se usa la función len()

***longitud\_country = len('México')***

En este caso al ejecutar la función len("México") sabemos que la cadena tiene una longitud de 6 caracteres

Algo importante a tener en cuenta cuando hablamos de strings

es que estos son inmutables, esto significa que cada vez que modificamos uno tenemos que generar un nuevo objeto en memoria.

## Mutabilidad Inmutabilidad y Referencia

**Mutables** → Listas

**Inmutables** → Int - String - tuplas

Los valores enteros, strings y las tuplas utilizan un espacio de memoria específico.

*>>> x=34*

*>>> y=x*

*>>> print(x)*

*34*

*>>> print(y)*

*34*

*>>> x=10*

*>>> print(x)*

*10*

*>>> print(y)*

*34*

Las listas funcionan como un apuntador en memoria, entonces cuando se asigna una nueva variable a la lista, se referencia al mismo apuntador. Las dos variables tienen el mismo espacio de memoria - “Son la misma variable”. Termina siendo un Puntero de la variable.

*>>> x=[1,2,3,4]*

*>>> y=x*

*>>> print(y)*

*[1, 2, 3, 4]*

*>>> x[0]=5*

*>>> print(y)*

*[5, 2, 3, 4]*

*>>> y is x*

*True*

*>>> id(x)*

*2600461917256*

*>>> id(y)*

*2600461917256*

Si quieres cortar partes que no quieras del principio o del final de la cadena de caracteres, lo puedes hacer creando una sub-cadena. El mismo tipo de técnica te permite separar una cadena muy larga en componentes más manejables.

*mensaje9 = "Hola Mundo"*

*mensaje9a = mensaje9[1:8]*

*print(mensaje9a)*

*-> ola Mun*

## Operaciones con strings

En python las strings pueden ser manipuladas gracias a varios métodos que se explican a continuación:

* upper() : convierte todo el string a mayúsculas
  + string.upper()
* lower() : convierte todo el string a minúsculas
  + string.lower()
* find() : encuentra un carácter o una cadena de caracteres dentro de un string, si lo encuentra devuelve la posición donde la cadena comienza, si no la encuentra devuelve un -1
  + string.find(otra\_string)
* replace() : reemplaza un caracter o una cadena de caracteres por otra.
  + string.replace(cadena , reemplazoDeCadena)
* startswith() : verifica si un string comienza con un carácter o una lista de caracteres específica
  + string.startswith(otra\_string)
* endswith() : verifica si un string termina con un carácter o una lista de caracteres específica
  + string.endswith(otra\_string)
* capitalize() : convierte la letra inicial de un string a mayúsculas, así “Hola mundo”

Más información [aquí](https://www.w3schools.com/python/python_ref_string.asp)

## Crear un help() para nuestras funciones

Normalmente cuando se crea una función también se crea una guia de como utilizarla y en python podemos crear esto con las triples comillas dobles, ejemplo:

* def my\_function():
  + “””Esta es una guia de como utilizar la funcion my\_function()”””
  + pass

Ahora si alguien quiere saber cómo utilizar nuestra función sólo tendrá que teclear “**help(my\_function)**”

## Slices en python

Python tiene una de las sintaxis más poderosas para manipular secuencias

Esta sintaxis se llama slice (rebanada en español)

**secuencia[comienzo:final:pasos]**

Ej.

* ***my\_name = ‘David’***
* ***my\_name[0]***
  + ***D***
* ***my\_name[-1]***
  + ***d***
* ***my\_name[0:3]***
  + ***Dav***
* ***my\_name[::2]***
  + ***Dvd***
* ***my\_name[::-1]***
  + ***divaD***

En el último esto sucede porque al poner el número de pasos en 2 significa que va a ir de dos en dos recorriendo las letras así que solo toma las letras que están en la posición 0,2,4…

Ojo, si colocamos el índice así [1::2] tomaría la posición 1 como la posición 0.

Nota:

El carácter que está en la posición final no se cuenta, por eso a pesar de que en la 3ra línea del ejemplo va hasta la posición 3, no aparece la “i”.

## Métodos join() y split() en Python

**S.join(sequencee):**

Retorna una CADENA resultante de concatenar la “cadena de la secuencia”, separada por la cadena (S) sobre la que se llama al método.

*>>> '-'.join(['1','2','4','1'])*

*'1-2-4-1'*

*>>> '-'.join('asdfghjkl')*

*'a-s-d-f-g-h-j-k-l'*

*>>> ''.join('asdfghjkl')*

*'asdfghjkl'*

*>>> '\*poi\*'.join('asdfghjkl')*

*'a\*poi\*s\*poi\*d\*poi\*f\*poi\*g\*poi\*h\*poi\*j\*poi\*k\*poi\*l'*

*>>> '-'.join('s s d f g')*

*'s- -s- -d- -f- -g'*

*>>> '-'.join('ssdfg')*

*'s-s-d-f-g'*

*>>> a = ['a', 'b', 'c', 'd']*

*>>> "".join(a)*

*'abcd'*

*>>>*

Un buen truco con join() es dar formato a las salidas:

*>>> L = ['a', 'b', 'c', 'd']*

*>>> "\n".join(L)*

*'a\nb\nc\nd'*

*>>> print "\n".join(L)*

*a*

*b*

*c*

*d*

*>>>*

**S.split([sep, [, maxsplit]]):**

Retorna una LISTA a partir de la cadena “S”, la cual se delimita por los índices que indique “sep”. Si no se especifica “sep” se usan los espacios. Si se especifica “maxsplit”, éste indica el número máximo de índices a realizar.

*>>> 'sdfsdf'.split()*

*['sdfsdf']*

*>>> ' kkk '.split()*

*['kkk']*

*>>> '1-2-3-4-5-6-7-8'.split('-')*

*['1', '2', '3', '4', '5', '6', '7', '8']*

*>>> 'sdfsdf'.split('d')*

*['s', 'fs', 'f']*

*>>> a = str([1,2,3,4]).split('2')*

*>>> len(a)*

*2*

*>>> a*

*['[1, ', ', 3, 4]']*

*>>> '1-2-3-4-5'.split('-', 2)*

*['1', '2', '3-4-5']*

*>>>*

## For loops

Los **for loops** son unas de las formas que tiene python para iterar[[1]](#footnote-0). **for** nos permite iterar a través de una secuencia, por eso esto se usa cuando se quiere hacer algo un número específico de veces, la sintaxis para utilizarlo es:

* for i in range(10):
  + print(i)

>>>0

>>>1

>>>2

…

>>>9

Aquí lo que pasa es que acabas de iniciar la variable “i” (esta es una variable de ejemplo pero puedes ponerle el nombre que sea, también podría llamarse “iterador” o “pedro”) lo que hará la función range será hacer una lista de 10 números empezando por 0, luego de eso meterá el primer valor de la lista a la variable “i” y ejecutará lo que esté dentro del bloque de código, después meterá el segundo valor de la lista y hará lo mismo, esto hasta que termine toda la lista, después simplemente se acabara la iteración.

## While loops

Los while loops son otro tipo de iteradores, mientras que los for loops nos sirven para iterar a través de una secuencia, los while loops nos sirven para iterar mientras que una condición sea verdadera.

* while n != ‘Platzi’:
  + print(‘Escribe \‘Platzi\’ >:c \n’)
  + n=input()
* print(‘Gracias c:’)

>>>Escribe ‘Platzi’ >:c

no

>>>Escribe ‘Platzi’ >:c

jamas >:c

>>>Escribe ‘Platzi’ >:c

Platzi

>>>Gracias c:

## Controlar los loop usando Break y Continue

Puedes encontrar una explicación completa [aquí](https://www.pythonmania.net/es/2013/04/05/control-de-bucles-break-continue-y-pass/) recuerda que también son tus notas así que puedes hacer un resumen de lo que viene [aquí](https://www.pythonmania.net/es/2013/04/05/control-de-bucles-break-continue-y-pass/) para la comunidad.

## Recursividad

En todos los lenguajes de programación hay algo llamado recursividad, que consiste en que una función puede llamarse a sí misma.

Esto es representado normalmente con el ejemplo de sacar el factorial de un número y funciona de las siguientes maneras:

* **def factorial(n):**

**if n==1:**

**return 1**

**else:**

**return n \* factorial(n-1)**

* **def factorial(n):**

**while(n != 1):**

**return n \* factorial(n-1)**

**return 1**

De esta manera el proceso que sigue es este:

resultado = factorial(5)

return 5\*factorial(4)

return 4\*factorial(3)

return 3\*factorial(2)

return 2\*factorial(1)

1 → return 1

return 2\*1

return 3\*2

return 4\*6

return 5\*24

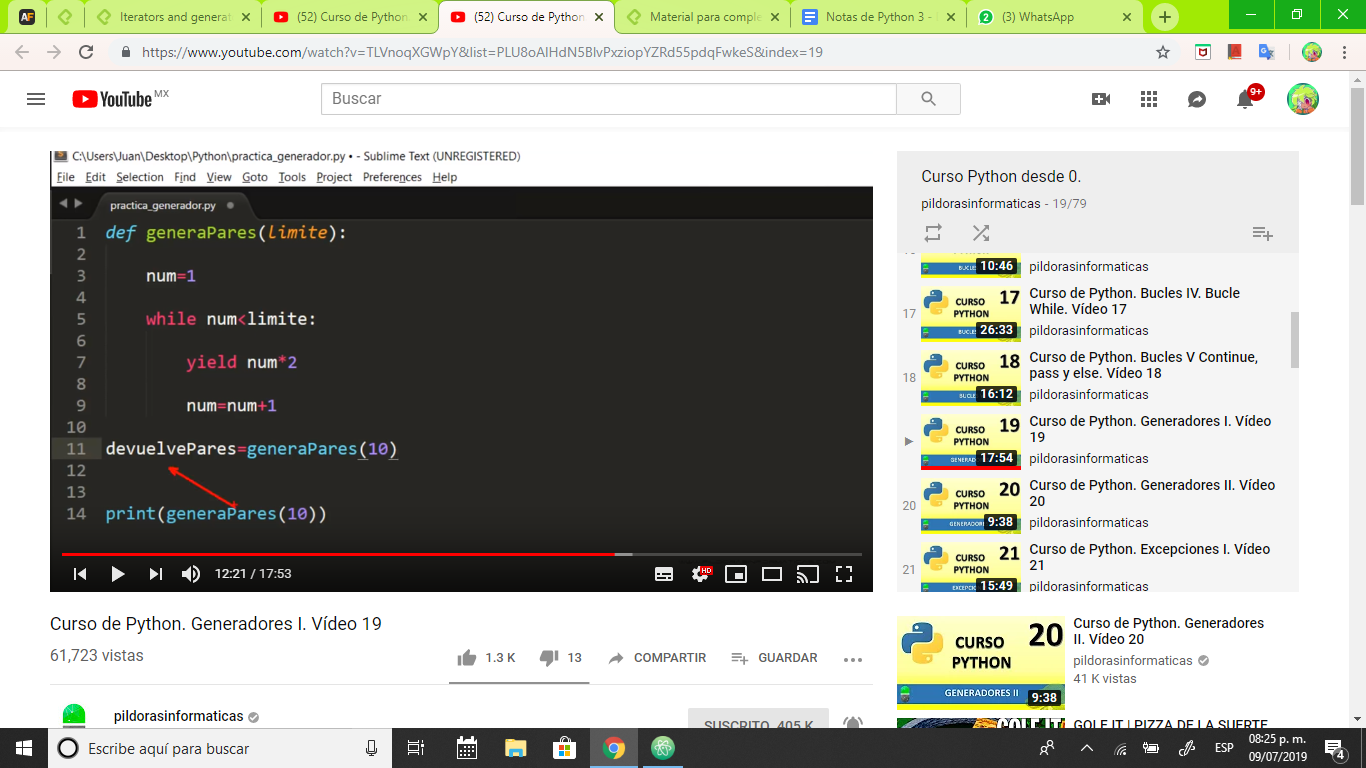
resultado = 120

Nota: Python tiene una protección contra los infinite loops recursivos, permitiendo solo 1000 llamadas recursivas, si permitiera más la memoria se llenaría y el pc se congelaría.

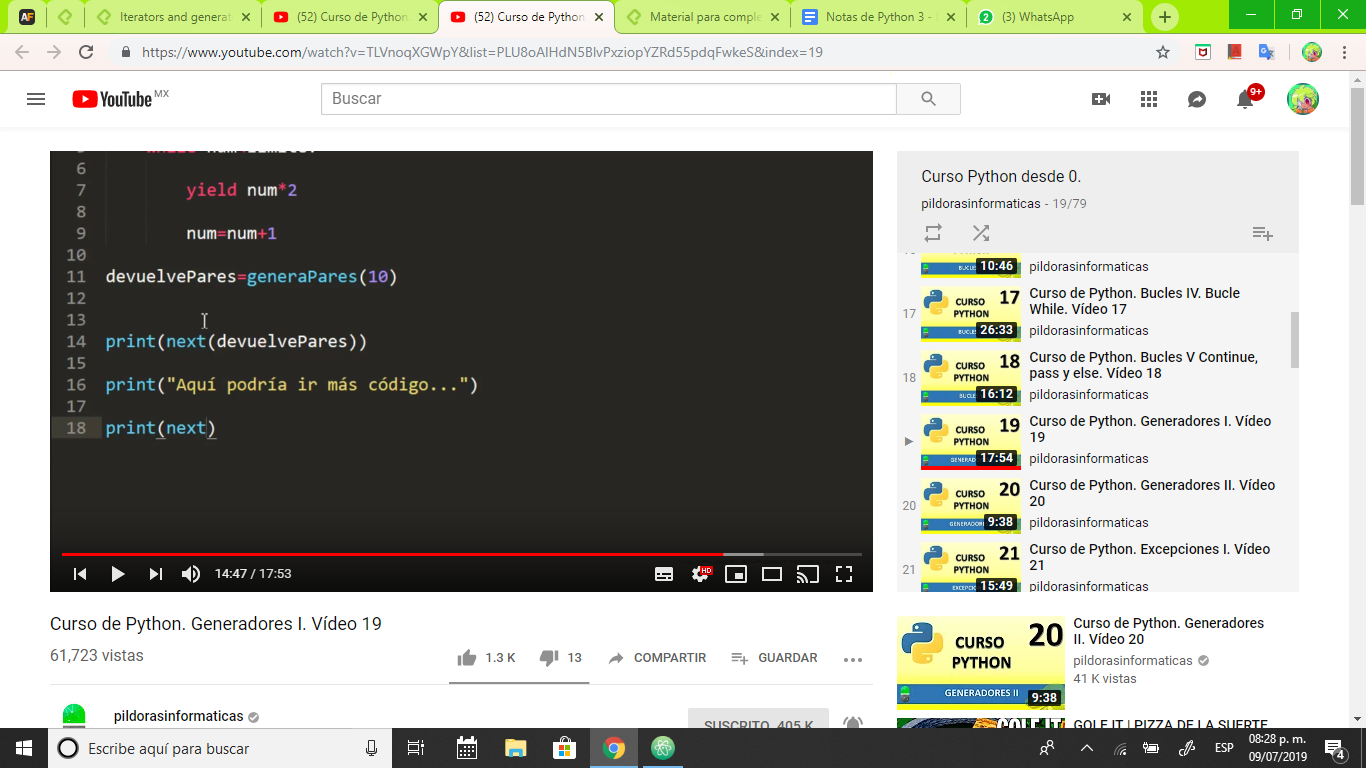
## Iterators and generators

Explicación: <https://youtu.be/TLVnoqXGWpY>

Básicamente, los generadores funciona al igual que la función range, que devuelve los valores de una lista, si producir la lista entera.



Esto devolverá el primer valor de la lista y hasta que sea llamada otra vez devolverá el segundo, al igual que range.



La función Next sirve para solicitar el siguiente valor sin necesidad de un iterador.

# Estructuras de Datos

## Las Listas, Tuplas y operaciones con Listas

Una lista es un tipo de dato que permite guardar una secuencia de datos, como los strings, sin embargo las listas pueden guardar varios tipos de datos dentro de si, como enteros, booleanos, strings e incluso otras listas, con la ventaja añadida de que las listas son mutables, osea que se pueden modificar los datos de una lista, cuando se asigna a una variable permite agrupar varios elementos en un solo lugar.

La forma de crear listas es usando corchetes(**[ ]**) o la keyword **list(),** así:

* ejemplo\_de\_lista = [ “nombre”, 25, 2.34, [1, 2, 3] ]
  + para acceder a una lista(lista2) dentro de otra lista(lista1) se ingresa la posicion en lista1 donde esta situada la lista2 y luego la posición de la lista2 a la que queremos acceder, ejemplo:
  + >>> ejemplo\_de\_lista [3][1]
  + >>> 2
* ejemplo\_de\_lista = list(“nombre”)
  + la keyword list solo acepta un parámetro y la lista se vería así:
  + [‘n’, ‘o’, ‘m’, ‘b’, ‘r’, ‘e’]

para declarar una lista vacía:

* ejemplo\_de\_lista = list()
* ejemplo\_de\_lista = []

Las listas son mutables y para modificarlas se usa la siguiente sintaxis:

>>>ejemplo\_de\_lista = [23]

>>> ejemplo\_de\_lista[0] = 1

>>> ejemplo\_de\_lista

>>> [1]

Esto no sirve para añadir valores, solo cambian valores ya existentes, si tratara de hacer la misma operación con una lista declarada vacía, saltará un error:

>>>ejemplo\_de\_lista = []

>>> ejemplo\_de\_lista[0] = 1

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

IndexError: **list assignment index out of range**

Los slices en python también funcionan con las listas:

**[comienzo:final:pasos]**

### Tuplas

Las tuplas son como las listas, funcionan de la misma forma, las tuplas se declaran con paréntesis **()** o con la keyword **tuple()**. En caso de que se quiera una tupla con un solo elemento se deberá de poner una coma:

>>> b = (5,) # Es una tupla

>>> type(b)

<type 'tuple'>

>>> c = (5) # Es un número

>>> type(c)

<type 'int'>

La diferencia es que las listas presentan una serie de funciones adicionales que permiten un amplio manejo de los valores que contienen. Basándonos en esta definición, puede decirse que las listas son dinámicas, mientras que las tuplas son estáticas.

### Operaciones y métodos con listas:

Operaciones:

**+** = Suma las listas, se usa de esta forma (list + list):

>>> lista1 = [1]

>>> lista2 = [2]

lista1 + lista2 = [1,2]

lista2 + lista1 = [2,1]

**\*** = multiplica los valores de una lista un número de veces. Así (list \* int) :

>>> lista1 = [1,2]

>>> lista1 \* 4

[1,2,1,2,1,2,1,2]

**Métodos:**

de agregado:

1. lista.append("nuevo elemento") #Añade el elemento al final
2. lista.extend(otra\_lista) #La añade al final
3. lista.insert(índice, "nuevo elemento") #Lo añade recorriendo la lista

de eliminación:

1. lista.pop() #Elimina elemento al final y lo retorna
2. lista.pop(índice) #Elimina el indicado y lo retorna
3. lista.remove("valor") #Solo lo elimina

de orden:

1. lista.reverse() #Invierte el orden
2. lista.sort() #La ordena de forma ascendente
3. lista.sort(reverse=True) #La ordena de forma descendente

de búsqueda:

1. lista.count(elemento) #Cuenta la cantidad de apariciones
2. lista.index(elemento) #Busca la primera posición donde el elemento aparezca
3. lista.index(elemento, indice\_inicio, indice\_fin) #Busca la primera posición en

#donde el elemento aparezca dentro de los índices

Obtener el valor máximo y mínimo de una tupla o lista:

1. min(tupla o lista)
2. max(lista o tupla)

Contar elementos:

1. len(lista o tupla)

## Diccionarios en python

Un diccionario es como una lista, la forma de declararlos es usando llaves o ‘curly braces’ (**{}**) o el keyword **dict()**, en las listas los valores están ligados a un índice que va de 0 a la longitud de la lista, pero en los diccionarios es un poco diferente, la primera diferencia es que los índices pasan a ser llaves y como llaves podemos hacer que cualquier tipo de valor sea una llave, un ejemplo:

>>>diccionario\_ejemplo = {}

>>> diccionario\_ejemplo[‘primer\_valor’] = 32

>>>diccionario\_ejemplo[2] = ‘pepe’

>>>lista = list()

>>>lista.append(‘pedro’)

>>>diccionario\_ejemplo[2.5] = lista

>>> diccionario\_ejemplo

{ 2: 'pepe', 'primer\_valor': 32, 2.5: ['pedro'] }

**Los valores no están necesariamente en el orden en el que los introducimos. También se puede declarar un diccionario con valores ya añadidos:**

>>>diccionario\_ejemplo = {32: ‘pedro’, ‘pepe’: 3.5, [1,2,3] : ‘lista’}

Iteración dentro de diccionarios

Hay varias formas de iterar sobre los valores que tiene un diccionario:

for llave in diccionario: # Si no colocamos nada, por defecto

… # nos dará las llaves del diccionario

for llave in diccionario.keys(): # Con el método keys(), nos dará

… # las llaves del diccionario

for valor in diccionario.values(): # Con el método values(), nos dará

… # los valores del diccionario

for llave, valor in diccionario.items(): # Con el método items(), nos dará

… # las llaves y los valores del

# diccionario en ese orden.

## Uso de Set’s en python

Los sets son muy similares a las listas, sin embargo, estos no permiten valores repetidos; set traducido al español sería el equivalente a conjuntos, y los sets funcionan con las reglas de la teoría de conjuntos que viste en preparatoria y se pueden ilustrar sus operaciones gracias a los diagramas de venn.

Para declarar un set se usa la función set() y se le da una lista, así que para iniciar un conjunto “s” con los calores [1, 2 y 3] tendríamos que teclear:

**>> s = set([1, 2, 3])**

Lo cual se vería así en un diagrama de venn

Los conjuntos tienen varias operaciones que los hacen convenientes a la hora de usarse en programación.

### Métodos para sets

Para obtener la longitud se usa “**len**”.

Para saber si un elemento existe en una lista se usa “**in**” y “**not in**”.

Dos métodos muy útiles son “**issubset**”e “**issuperset**”; el primero ayuda a probar si el primer conjunto es un subconjunto del segundo (o sea, que todos sus elementos estén en el otro) y el segundo prueba lo contrario.

Para copiar un conjunto se llama al método “**copy**”.

### Operaciones con sets

s = set([1, 2, 3])

t = set([3, 4, 5])

**Unión**

Para unir dos conjuntos se usa el método “**union**” que cada conjunto tiene o aplicamos el operador “**|**”. Como argumento pasamos uno o más conjuntos.

>> st = s.union(t) // es lo mismo que “st = s | t”

>> st

[1,2,3,4,5]

**Intersección**

La operación de intersección es cuando creamos un conjunto que se encuentra en ambos conjuntos. Para hacer la operación de intersección llamamos al método “**intersection**” o aplicamos el operador “**&**”.

>> e = s.intersection(t) // es lo mismo que “e = s & t”

>> e

[3]

**Diferencia**

La diferencia es eliminar del primer conjunto todo lo que no esté en el segundo. Para hacer esta operación se usa el método “**difference**” o el operador “**-**”, así como si estuviéramos restando números.

>> e = s.difference(t) // es lo mismo que “e = s - t”

>> i = t.difference(s) // es lo mismo que “i = t - s”

>> i

[4,5]

>> e

[1,2]

**Complemento**

Para calcular el complemento hacemos una simple **diferencia**, poniendo un conjunto como el conjunto universal.

**Diferencia simétrica**

Para obtener la diferencia simétrica de dos conjuntos con Python se usa el método “**symmetric\_difference**” o el operador “**^**”.

>> e = s.symmetric\_difference(t) // es lo mismo que “e = s ^ t”

>> e

[1,2,4,5]

## Dictionary comprehension - list comprehension

Estas son conceptos que nos permiten añadir syntactic sugar a nuestro código, esto se hace no solo para ahorrarse líneas de código sino porque también permiten que nuestro código sea más legible y por tanto mejor.

Estos son algunos ejemplos de cómo utilizar esta herramienta:

pares = []

>>> for num in range(1, 31):

... if num%2 == 0:

... pares.append(num)

...

>>> pares

[2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30]

>>> even = [num for num in range(1, 31) if num%2 == 0]

>>> even

[2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30]

>>> cuadrados = {}

>>> for num in range(1, 11):

... cuadrados[num] = num \*\* 2

...

>>> cuadrados

{1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49, 8: 64, 9: 81, 10: 100}

>>> squares = {num: num \*\* 2 for num in range(1, 11)}

>>> squares

{1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49, 8: 64, 9: 81, 10: 100}

Para entender cómo crear uno analicemos uno de los ejemplos:

[num for num in range(1, 31) if num%2 == 0]

Esta es la variable que se añadirá a la lista.

Aquí es donde la variable se declara.

Esta parte sirve para generar la lista de números que se quiere añadir.

Esta condicional decide cuál valor entrará y cuál no de la lista generada anteriormente.

# Uso de objetos y módulos

# Python en el mundo real

# Conclusiones finales

# Clases bonus

## Notas

Si tienes alguna duda en algún momento del curso puedes ingresar a este documento público de notas en comunidad.

## [Notas de la comunidad](https://docs.google.com/document/d/1evjmD6FMVRTM2-\_pQtsjQlBKDjhDHmZw7BYhLSpt8ew/edit?usp=sharing)

https://docs.google.com/document/d/1evjmD6FMVRTM2-\_pQtsjQlBKDjhDHmZw7BYhLSpt8ew/edit?usp=sharing

y puedes añadir información, aportar o simplemente corregir y así aprender todos juntos.

\*\*++Nunca paren de aprender.++\*\*

[**Un Repositorio con todos los ejercicios del curso.**](https://github.com/EliazBobadilla/Platzi-Curso-de-Python-2019)

**import time**

**from functools import lru\_cache**

**def factorial(n):**

**return n \* factorial(n-1) if n else 1**

**@lru\_cache**

**def factorial1(n):**

**return n \* factorial1(n-1) if n else 1**

1. "Iteración” - Repetir una secuencia varias veces [↑](#footnote-ref-0)