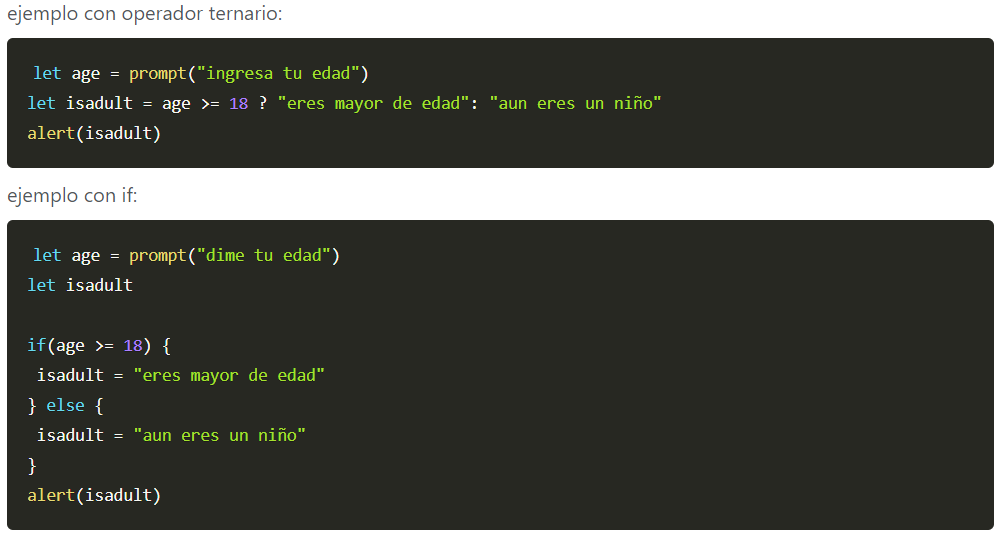
**¿Cuáles son las diferencias en la lista y la tupla?**

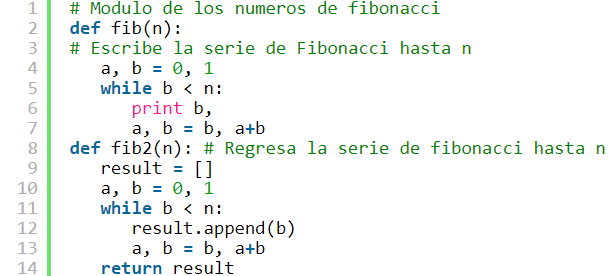
Una tupla es inmutable, así que una vez creada no puede ser modificada

**¿Cómo se puede usar expresiones if else en una sóloa línea, comúnmente llamadas operaraciones ternarias?**



**¿Para que sirve dir() y help()?**

Esta función devuelve una lista ordenada alfabéticamente de los métodos y propiedades de algún objeto



Help()

El método help () se utiliza para uso interactivo. Se recomienda intentarlo en su intérprete cuando necesite ayuda para escribir el programa Python y usar los módulos de Python

**Que son diccionarios?**

Los diccionarios en Python son un tipo de estructuras de datos que permite guardar un conjunto no ordenado de pares clave-valor, siendo las claves únicas dentro de un mismo diccionario (es decir que no pueden existir dos elementos con una misma clave).

**¿Que son \*args y \*\*kwargs ? ¿Como se usan?**

Esto es muy útil cuando estás creando funciones que pueden aceptar un número variable de parámetros. De ese modo, no es necesario darle un nombre a cada uno de ellos al definir la función, por que no sabes de antemano cuantas van a ser, por eso les pones un nombre genérico que es el nombre de todas ellas (o de la estructura que las contiene, un tuple o un dict).

1. **def** func(\*args, \*\*kwargs):
2. **for** a **in** args:
3. **print** a
4. **for** k,v **in** kwargs.items():
5. **print** k, ':', v

8. >>> func(1,2,6,3, uno='one', dos='two', tres='three')
9. 1
10. 2
11. 6
12. 3
13. dos : two
14. tres : three
15. uno : one
16. >>>

La función “func” acepta dos tipos de parámetros:

\*args: cuando ves un asterisco seguido por un nombre, significa que lo que sigue es un tuple. Un tuple, es como un array de objetos (numeros, strings, etc). O sea que args es un tuple, y en el ejemplo de arriba (donde llamo a la función) ese tuple es (1,2,6,3).

\*\*kwargs: cuando ves dos asteriscos, significa que lo que sigue es un diccionario (dict). O sea una serie de claves y valores. En este caso, kwargs es un dict.

Para verlo más claro, acá va otro ejemplo:

1. **def** func(\*args, \*\*kwargs):
2. **print** type(args)
3. **print** type(kwargs)

6. >>> func(1,2,6,3, uno='one', dos='two',tres='three')
7. <type 'tuple'>
8. <type 'dict'>
9. >>>

**¿Qué son índices negativos?**

Los índices pueden ser enteros negativos tambien. El concepto es sencillo:

lista[-1] hace mención al último elemento de la lista; lista[-2] al penúltimo, y así sucesivamente.

Como se puede order aleatoriamente -o desordenar- una lista?

Ya hemos visto las alternativas que tenemos para generar números aleatorios con [Python](http://www.manualweb.net/tutorial-python/). Ahora vamos a dar un paso más y generar una lista aleatoria con [Python](http://www.manualweb.net/tutorial-python/).

Lo primero de todo será tener una lista. Por ejemplo podemos crear una lista de números de una forma muy rápida mediante la función range().

1. lista = range(0,100)

En este caso hemos creado una lista de números de 0 al 100. Ahora lo que vamos a generar una lista aleatoria con [Python](http://www.manualweb.net/tutorial-python/) en la que solo aparezcan 10 números elegidos de estos 100. Para ello vamos a utilizar la función sample(). La función sample()tiene dos parámetros:

1. sample(lista,k=numero\_elementos)

El primer parámetro es la lista origen de la que vamos a generar la lista aleatoria, el segundo es el número de elementos que va a tener esta lista aleatoria con [Python](http://www.manualweb.net/tutorial-python/).

De esta forma si queremos 10 números de nuestra lista inicial codificaremos lo siguiente:

1. **print** sample(lista,k=10)

El resultado será, por ejemplo:

[74, 63, 30, 72, 35, 86, 87, 38, 0, 84]

Pero la lista aleatoria con [Python](http://www.manualweb.net/tutorial-python/) puede ser de cualquier tipo de elementos. Por ejemplo podríamos hacerla de cadenas de texto.

1. vehiculos = ['avión','bicicleta','coche','motocicleta','tractor','barco','transatlántico','camión']

Y la forma de generar una lista aleatoria a partir de estos elementos sería la misma:

1. **print** sample(vehiculos,k=4)

Generando en este caso salidas como:

['tractor', 'bicicleta', 'avión', 'motocicleta']

Como se puede ordenar una lista ?

Ordenar una lista en reversa (invertir orden)

**Método**: reverse()

>>> nombres\_masculinos.reverse()

>>> print nombres\_masculinos

['Jose', 'Jose', 'Ricky', 'Jacinto', 'David', 'Alvaro', 'Ricky']

Ordenar una lista en forma ascendente

**Método**: sort()

>>> nombres\_masculinos.sort()

>>> print nombres\_masculinos

['Alvaro', 'David', 'Jacinto', 'Jose', 'Jose', 'Ricky', 'Ricky']

Ordenar una lista en forma descendente

**Método**: sort(reverse=True)

>>> nombres\_masculinos.sort(reverse=True)

>>> print nombres\_masculinos

['Ricky', 'Ricky', 'Jose', 'Jose', 'Jacinto', 'David', 'Alvaro']

Explique o justifique los resultados de A0,A1,A2,A3,A4,A5,A6 ?

A0 = dict(zip(('a','b','c','d','e'),(1,2,3,4,5)))

Lo que hace la expresión zip es corresponde las dos listas dando como resultado el siguiente diccionario: {'a': 1, 'b': 2, 'c': 3, 'd': 4, 'e': 5}

A1 = range(10)

En este caso la función range genera automáticamente una secuencia con una progresión aritmética que puede utilizarse con uno, dos o tres parámetros, dando como resultado: range(0, 10)

A2 = sorted([i for i in A1 if i in A0])

El resultado generado es [] dado que el valor de A1 no se encuentra en A0

A3 = sorted([A0[s] for s in A0])

Se ordenan los valores de A0 en orden ascendente

A4 = [i for i in A1 if i in A3]

Muestra los valores que se encuentran en A1 y que están en A3

A5 = {i:i\*i for i in A1}

Devuelve el valor i en A1 y lo multiplica por i, dando como resultado un diccionario: {0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49, 8: 64, 9: 81}

A6 = [[i,i\*i] for i in A1]

Devuelve el valor i en A1 y lo multiplica por i, dando como resultado una lista de listas: [[0, 0], [1, 1], [2, 4], [3, 9], [4, 16], [5, 25], [6, 36], [7, 49], [8, 64], [9, 81]]

print(A0,A1,A2,A3,A4,A5,A6)

**¿Cómo se pueden generar números aleatorios? enteros y decimales**

**Enteros**

La función **randrange()** devuelve enteros que van desde un valor inicial a otro final separados entre sí un número de valores determinados. Esta separación (o paso) se utiliza en primer lugar con el valor inicial para calcular el siguiente valor y los sucesivos hasta llegar al valor final o al más cercano posible.

En el ejemplo que sigue se obtienen 25 números que van desde el 3 al 16. Como el paso es 3 nunca se obtendría el valor 16.

print('\nValores posibles: 3, 6, 9, 12, 15')

for i in range(25):

print(random.randrange(3, 16, 3), end=' ')

**Desimales**

**random()**

La función **random()** devuelve un número ***float*** entre 0 y 1    
En el ejemplo se obtienen tres números con muchos decimales entre 0 y 1.

for numero in range(3):

print(random.random(), end=' ')

**uniform()**

La función **uniform()** devuelve un número ***float*** incluido entre los valores indicados.

for numero in range(3):

print(random.uniform(100, 105), end=' ')

**¿Qué es pickling y unpickling?**

Los pickles de Python son la mar de útiles ya que representan un objeto Python como una cadena de bytes. Se pueden hacer multitud de cosas con dichos bytes, como por ejemplo, almacenarlos en un archivo o base de datos, o transferirlos a través de una red.

La representación “pickle” de un objeto Python, se denomina archivo pickle. El archivo pickle, por tanto, se puede utilizar para distintos propósitos, como el almacenamiento de resultados para que sea utilizado por otro programa Python, o bien para crear copias de seguridad. Para obtener el objeto original de Python, simplemente tienes que “unpicklear” esa cadena de bytes.

Para utilizar los pickles en Python, utilizaremos el módulo Pickle. Este nos permite almacenar casi cualquier objeto Python directamente en un archivo o cadena sin necesidad de realizar ninguna conversión. Lo que el módulo pickle realiza en realidad es lo que se llama serialización de objetos, es decir, convertir objetos a y de cadenas de bytes. El objeto que va a ser pickled se serializará en un flujo de bytes, los cuales se pueden escribir en un archivo, por ejemplo, y restaurar en un punto posterior.

### Instalación

El módulo Pickle viene ya incluido en la instalación de Python. Para obtener una lista de los módulos instalados, puede escribir el siguiente comando en el prompt de Python: help('modules').

Así que todo lo que tienes que hacer para trabajar con el módulo Pickle es: import pickle!

### Crear un archivo Pickle

A partir de esta sección, vamos a echar un vistazo a algunos ejemplos de pickling para entender mejor el concepto. Empecemos por crear un archivo pickle de un objeto. El objeto que utilizaremos será la lista de tareas que hicimos en el [tutorial de listas de Python](https://programacion.net/articulo/como_funcionan_las_listas_de_python_1496).

todo = ['write blog post', 'reply to email', 'read in a book']

Con el fin de convertir a pickle nuestro objeto de lista (todo), podemos hacer lo siguiente:

import pickle  
   
todo = ['write blog post', 'reply to email', 'read in a book']  
pickle\_file = file('todo.pickle', 'w')  
pickle.dump(todo, pickle\_file)

Ten en cuenta que hemos hecho un import pickle para poder utilizar el módulo Pickle. También hemos creado un archivo pickle para almacenar el objeto que vamos a hacer pickle, es decir todo.pickle. La función dump graba una representación pickled de “todo” en el objeto de fichero pickle\_file. En otras palabras, la función dump tiene dos argumentos: el objeto pickle, que es la lista de tareas y un objeto file donde queremos escribir el pickle, que es todo.pickle.

### Unpickling (restaurar) los datos pickled

Digamos que nos gustaría restaurar los datos pickled (en nuestro caso, la lista de tareas). Para ello, podemos utilizar el siguiente script:

import pickle  
   
pickle\_file = file('todo.pickle')  
todo = pickle.load(pickle\_file)  
print(todo)

El anterior script listará los elementos de “todo”.

['write blog post', 'reply to email', 'read in a book']

### Pickles como strings

En la sección anterior, hemos visto cómo podemos escribir/cargar pickles a/de un archivo. Sin embargo, esto no es necesario. Quiero decir que si queremos escribir/cargar pickles, no siempre tenemos que tratar con archivos, sino que podemos trabajar con pickles como cadenas. De este modo, podemos hacer lo siguiente:

import pickle  
   
todo = ['write blog post', 'reply to email', 'read in a book']  
pickled\_data = pickle.dumps(todo)  
print(pickled\_data)

Ojo, he utilizado la función dumps (con una “s” al final) que, de acuerdo a la documentación, devuelve la representación de un pickle como una cadena, en lugar de hacerlo en un archivo.

Con el fin de restaurar los datos pickled de antes, podemos utilizar la función load(string), tal que así:

restored\_data = pickle.loads(pickled\_data)

### ¿Y cómo hacerlo con más de un objeto?

En los ejemplos de antes, hemos creado un pickled y hemos hecho su respectiva restauración de un sólo objeto a la vez. En esta sección, os mostraré cómo proceder para hacer lo mismo en más de un objeto a la vez. Digamos que tenemos los siguientes objetos:

name = 'Jorge'  
website = 'http://programacion.net'  
english\_french = {'paper':'papier', 'pen':'stylo', 'car':'voiture'} # dictionary  
tup = (31,'abder',4.0) # tuple

Simplemente podemos escoger los objetos anteriores ejecutando una serie de funciones de volcado, de la siguiente manera:

import pickle  
   
name = 'Jorge'  
website = 'http://programacion.net'  
english\_french = {'paper':'papier', 'pen':'stylo', 'car':'voiture'} # dictionary  
tup = (31,'abder',4.0) # tuple  
   
pickled\_file = open('pickled\_file.pickle', 'w')  
pickle.dump(name, pickled\_file)  
pickle.dump(website, pickled\_file)  
pickle.dump(english\_french, pickled\_file)  
pickle.dump(tup, pickled\_file)

Esto creará un único fichero pickle de los cuatro objetos llamado pickled\_file.pickle.

Existe otra fabulosa manera de escribir el anterior script haciendo uso de la clase Pickler:

from pickle import Pickler  
   
name = 'Jorge'  
website = 'http://programacion.net'  
english\_french = {'paper':'papier', 'pen':'stylo', 'car':'voiture'} # dictionary  
tup = (31,'abder',4.0) # tuple  
   
pickled\_file = open('pickled\_file.pickle', 'w')  
p = Pickler(pickled\_file)  
p.dump(name); p.dump(website); p.dump(english\_french); p.dump(tup)

Para restaurar los datos originales, podemos utilizar más de una función load. Echa un vistazo:

import pickle  
   
pickled\_file = open('pickled\_file.pickle')  
   
name = pickle.load(pickled\_file)  
website = pickle.load(pickled\_file)  
english\_french = pickle.load(pickled\_file)  
tup = pickle.load(pickled\_file)  
   
print('Name: ')  
print(name)  
print('Website:')  
print(website)  
print('Englsh to French:')  
print(english\_french)  
print('Tuple data:')  
print(tup)

Para mostrar el anterior script sería:

Name:   
Abder  
Website:  
http://abder.io  
Englsh to French:  
{'car': 'voiture', 'pen': 'stylo', 'paper': 'papier'}  
Tuple data:  
(31, 'abder', 4.0)

Al igual que con la clase Pickler, podemos volver a escribir el script anterior usando la clase Unpickler, de la siguiente manera:

from pickle import Unpickler  
   
pickled\_file = open('pickled\_file.pickle')  
u = Unpickler(pickled\_file)  
name = u.load(); website = u.load(); english\_french = u.load(); tup = u.load()  
   
print('Name: ')  
print(name)  
print('Website:')  
print(website)  
print('English to French:')  
print(english\_french)  
print('Tuple data:')  
print(tup)

Ten en cuenta que las variables deben escribirse y leerse en el mismo orden para obtener la salida deseada. Para evitar cualquier problema, podemos utilizar un diccionario para administrar los datos, de la siguiente manera:

import pickle  
   
name = 'Jorge'  
website = 'http://programacion.net'  
english\_french = {'paper':'papier', 'pen':'stylo', 'car':'voiture'} # dictionary  
tup = (31,'abder',4.0) # tuple  
   
pickled\_file = open('pickled\_file.pickle', 'w')  
data = {'name':name, 'website':website,'english\_french\_dictionary':english\_french,'tuple':tup }  
pickle.dump(data, pickled\_file)

Para restaurar los datos pickled en el anterior script, podemos hacer lo siguiente:

import pickle  
   
pickled\_file = open('pickled\_file.pickle')  
data = pickle.load(pickled\_file)  
   
name = data['name']  
website = data['website']  
english\_french = data['english\_french\_dictionary']  
tup = data['tuple']  
   
print('Name: ')  
print(name)  
print('Website:')  
print(website)  
print('English to French:')  
print(english\_french)  
print('Tuple data:')  
print(tup)

Para que sirve la función map, lambda y filter?

Operador lambda:

El operador lambda o función lambda, es una forma de crear funciones anónimas, es decir, funciones sin nombre. Estas funciones son desechable, es decir, solo se necesitan donde se han creado. Las funciones lambda se utilizan principalmente en combinación con las funciones Map, Filter y Reduce.

La sintaxis general de una función lambda es bastante simple:  
*lambda argument\_list : expression*

El operador lambda o función lambda, es una forma de crear funciones anónimas, es decir, funciones sin nombre. Estas funciones son desechables, es decir, solo se nesecitan donde se han creado. Las funciones lambda se utilizan principalmente en combinatorio con las funciones Map, Filter y Reduce.

La sintaxis general de una funcion lambda es bastante simple:

*lambda argument\_list : expresison.*

Ejemplo: 

#Función lambda que devuelve la suma de sus dos argumentos:

f = lambda x, y : x + y

f(2 + 2)

#Resultado:

4

#Segundo ejemplo de lambda

#Función lambda que devuelve la raíz cuadrada de su argumento

a = lambda x : x\*\*1/2

a(233)

#Resultado:

15,2643

Como pueden ver, la función lambda nos devuelve valores que dependen de sus argumentos, esto significa que es una función pura.

Operador Map:

El operador Map, toma una función y un iterable como argumentos, y devuelve un nuevo iterablecon la función aplicada a cada argumento . **Ejemplo:**

#Ejemplo del operador Map

**def** add\_five(x):

**return** x + 5

nums = [11, 25, 34, 100, 23]

result = list(**map**(add\_five, nums))

**print**(result)

#Resultado:

[16, 30, 39, 105, 28]

Como pueden ver, "map" nos a devuelto una lista con todo los elementos de la lista "nums", vemos que a cada elemento le sumo 5.

Si usáramos el operador lambda en vez de declarar la función "add\_five", el resultado seria el mismo. Veamos un ejemplo:

#Ejemplo del operador Map y Lambda

nums = [11, 25, 34, 100, 23]

result = list(**map**(lambda x:x+5, nums))

**print**(result)

#Resultado:

[16, 30, 39, 105, 28]

Como pueden ver el resultado es el mismo, increíble, ¿Verdad?.  
Otra cosa a tener en claro es que en estos dos ejemplo, que para convertir el resultado en una lista, se hace uso del método "list".

    
Operador Filter:

El operado filter (función, lista) ofrece una forma elegante de filtrar todos los elementos de una lista, para los que la función de*función* devuelve True.    
El operador filter(f, l) necesita una función f como primer argumento. f devuelve un valor booleano, es decir, verdadero o falso. Esta función se aplicará a cada elemento de la lista. Solo si f devuelve True, el elemento de la lista se incluirá en la lista de resultados.

#Usando el operador Filter

nums = [0, 2, 5, 8, 10, 23, 31, 35, 36, 47, 50, 77, 93]

result = filter(lambda x: x % 2 == 0, nums)

**print**(result)

#Resultado:

[2, 8, 10, 36, 50]

Como pueden observar, el operador filter no incluyo a los elementos que no fueran divisibles entre 2, otra cosa es que hacemos uso de lambda, ya que es utilizado especialmente para este tipo de situaciones.

Qué es list comprehension, set comprehension y dict comprehension?

**Comprensión de la lista de Python**

Supongamos que tenemos dos listas, una que contiene números pares y otra que contiene números divisibles entre 3:

A = { a | una  es aún }UNA={una|una incluso}

B = { b | b  es divisible por 3 }segundo={segundo|segundo es divisible por 3}

Podemos crear otra lista que contenga números par y divisible por 3 fácilmente:

C = { c | c  es par y divisible por 3 } = { c | c ∈ A ∧ c ∈ B }do={do|do es par y divisible por 3}={do|do∈UNA∧do∈segundo}

En el código de Python, esto es lo mismo que lo siguiente:

# Create the lists

A = [a for a in range(20) if a % 2 == 0]

B = [b for b in range(20) if b % 3 == 0]

C = [c for c in range(20) if c in A and c in B]

# Print out the result

print('A: ', A)

print('B: ', B)

print('C: ', C)

Esto resulta en lo siguiente:

A: [0, 2, 4, 6, 8, 10, 12, 14, 16, 18]

B: [0, 3, 6, 9, 12, 15, 18]

C: [0, 6, 12, 18 ]

Python es incluso tan flexible, que permite anidar listas de comprensión en lista de comprensión:

# Create the lists

A = [a for a in range(5) if a % 2 == 0]

B = [b for b in range(5) if b % 3 == 0]

C = [a + b for a in A for b in B]

# Print out the result

print('A: ', A)

print('B: ', B)

print('C: ', C)

A: [0, 2, 4]

B: [0, 3]

C: [0, 3, 2, 5, 4, 7]

## **{} set comprehension**

Devuelve un conjunto basado en iterables existentes.

**Expression**

Opcional. Una expresión de salida que produce miembros del nuevo conjunto de miembros del conjunto de entrada que satisfacen la expresión de predicado.

**variable**

Necesario. Variable que representa a los miembros de un conjunto de entrada.

**input\_set**

Necesario. Representa el conjunto de entrada.

**predicate**

Opcional. Expresión que actúa como un filtro en los miembros del conjunto de entrada.

**[,…]]**

Opcional. Otra comprensión anidada.

## Valor de retorno

**conjunto**

## Complejidad del tiempo

## Ejemplo 1

>>> {s for s in [1, 2, 1, 0]}

set([0, 1, 2])

>>> {s\*\*2 for s in [1, 2, 1, 0]}

set([0, 1, 4])

>>> {s\*\*2 for s in range(10)}

set([0, 1, 4, 9, 16, 25, 36, 49, 64, 81])

## Ejemplo 2

>>> {s for s in [1, 2, 3] if s % 2}

set([1, 3])

## Ejemplo 3

>>> {(m, n) for n in range(2) for m in range(3, 5)}

set([(3, 0), (3, 1), (4, 0), (4, 1)])

Los diccionarios (o dicten Python) son una forma de almacenar elementos como lo haría en una lista de Python. Pero, en lugar de acceder a los elementos utilizando su índice, le asigna una clave fija y accede al elemento utilizando la clave. Con lo que ahora lidia es con un par "clave-valor", que a veces es una estructura de datos más apropiada para muchos problemas en lugar de una simple lista. A menudo, tendrá que lidiar con los diccionarios al hacer ciencia de datos, lo que hace que la comprensión del diccionario sea una habilidad que querrá dominar.