Material Imprimible

Curso de Python

Módulo 2

Booleanos de Python

Los booleanos representan uno de los dos valores: True o . False

Valores booleanos

En la programación a menudo es necesario saber si una expresión es True o False.

Puede evaluar cualquier expresión en Python y obtener una de las dos respuestas, True o . False

Cuando se comparan dos valores, se evalúa la expresión y Python devuelve la respuesta booleana:

```
print(10 > 9)
print(10 == 9)
print(10 < 9)
```

Cuando se ejecuta una condición en una instrucción if, Python devuelve o:True False

```
a = 200
b = 33

if b > a:
  print("b is greater than a")
else:
  print("b is not greater than a")
```

Evaluar valores y variables

La función bool()le permite evaluar cualquier valor, y darle o a cambio, True False

```
print(bool("Hello"))
print(bool(15))
x = "Hello"
```

```
y = 15
print(bool(x))
print(bool(y))
```

La mayoría de los valores son verdaderos

Casi cualquier valor se evalúa True si tiene algún tipo de contenido.

Cualquier cadena es True, excepto las cadenas vacías.

Cualquier número es True, excepto 0(cero) .

Cualquier lista, tupla, conjunto y diccionario son True, excepto los vacíos.

```
print(bool("abc"))
print(bool(123))
print(bool(["manzana", "frutilla", "banana"]))
```

Algunos valores son falsos

De hecho, no hay muchos valores que se evalúen como falso, excepto valores vacíos, como , , , el número y el valor . Y, por supuesto, el valor se evalúa como .False()[]{}""0 None False False

```
print(bool(False))
print(bool(None))
print(bool(0))
print(bool(""))
print(bool(()))
print(bool([]))
print(bool({}))
```

Un valor u objeto más, en este caso, se evalúa como , y es decir, si tiene un objeto que se realiza a partir de una clase con una función que devuelve: False__len__0False

```
class myclass():
  def __len__(self):
          return 0

myobj = myclass()
print(bool(myobj))
```

Las funciones pueden devolver un valor booleano

Puede crear funciones que devuelvan un valor booleano:

```
def myFunction() :
  return True

print(myFunction())
```

Puede ejecutar código basado en la respuesta booleana de una función:

```
def myFunction() :
  return True
if myFunction():
  print("YES!")
else:
  print("NO!")
```

Python también tiene muchas funciones integradas que devuelven un valor booleano, como la función `isinstance()`, que se puede utilizar para determinar si un objeto es de un tipo de datos determinado:

```
x = 200
print(isinstance(x, int))
```

Operadores de Python

Los operadores se utilizan para realizar operaciones en variables y valores.

Python divide los operadores en los siguientes grupos:

- Operadores aritméticos
- Operadores de asignación
- Operadores de comparación
- Operadores lógicos
- Operadores de identidad
- Operadores de membresía
- Operadores bit a bit

Operadores aritméticos de Python

Los operadores aritméticos se utilizan con valores numéricos para realizar operaciones matemáticas comunes:

Operador	Nombre	Ejemplo
+	Addition	x + y
-	Subtraction	x - y
*	Multiplication	x * y
/	Division	x / y
%	Modulus / Resto	x % y
**	Exponentiation	x ** y
//	Floor división/Redondea	x // y

Operadores de asignación

Los operadores de asignación se utilizan para asignar valores a variables:

Operador	Ejemplo	Igual que
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
//=	x //= 3	x = x // 3
**=	x **= 3	x = x ** 3
&=	x &= 3	x = x & 3
|=	x |= 3	x = x | 3
^=	x ^= 3	x = x ^ 3
>>=	x >>= 3	x = x >> 3
<<=	x <<= 3	x = x << 3

Operadores de comparación

Los operadores de comparación se utilizan para comparar dos valores

Operator	Name	Example
==	Igual	x == y
!=	No igual	x != y
>	Mayor que	x > y
<	Menor que	x < y
>=	Mayor igual que	x >= y
<=	Menor igual que	x <= y

Operadores lógicos

Los operadores lógicos se utilizan para combinar instrucciones condicionales

Operador	Descripción	Ejemplo
and	Devuelve True si ambas afirmaciones son verdaderas	x < 5 and x < 10
or	Devuelve True si una de las declaraciones es verdadera	x < 5 or x < 4
not	Invierte el resultado, devuelve False si el resultado es verdadero	not(x < 5 and x < 10)

Operadores de identidad

Los operadores de identidad se utilizan para comparar los objetos, no si son iguales, sino si en realidad son el mismo objeto, con la misma ubicación de memoria:

Operador	Descripcion	Ejemplo
is	Devuelve True si ambas variables son el mismo objeto	x is y
is not	Devuelve True si ambas variables no son el mismo objeto	x is not y

Operadores de membresía

Los operadores de pertenencia se utilizan para comprobar si se presenta una secuencia en un objeto

Operador	Descripcion	Ejemplo

in	Devuelve True si una secuencia con el valor especificado está presente en el objeto	x in y
not in	Devuelve True si una secuencia con el valor especificado no está presente en el objeto	x not in y

Operadores bit a bit

Los operadores bit a bit se utilizan para comparar números (binarios):

Operador	Nombre	Descripcion
&	AND	Establece cada bit en 1 si ambos bits son 1
|	OR	Establece cada bit en 1 si uno de los dos bits es 1
^	XOR	Establece cada bit en 1 si solo uno de dos bits es 1
~	NOT	Invierte todos los bits
<<	Zero	rellenar desplazamiento a la izquierda Desplazar a la izquierda presionando ceros desde la derecha y dejar caer los bits más a la izquierda
>>		Desplazamiento a la derecha firmado Desplace a la derecha empujando copias del bit más a la izquierda desde la izquierda, y deje que los bits más a la derecha se caigan

Colecciones de Python (Arrays)

Hay cuatro tipos de datos de colección en el lenguaje de programación Python:

- **List** es una colección que se ordena y se puede cambiar. Permite miembros duplicados.
- **Tuple** es una colección que se ordena e inmutable. Permite miembros duplicados.
- **Set** es una colección que no está ordenada y no está indizada. No hay miembros duplicados.
- **Dictionary** es una colección que es desordenada, cambiable e indizada. No hay miembros duplicados.

Al elegir un tipo de colección, es útil comprender las propiedades de ese tipo. Elegir el tipo adecuado para un conjunto de datos determinado podría significar la retención de significado y, podría significar un aumento en la eficiencia o la seguridad.

Lista

Una lista es una colección que se ordena y se puede cambiar. En Python las listas se escriben con corchetes.

```
thislist = ["manzana", "banana", "cherry"]
print(thislist)
```

Acceder a los elementos

Puede acceder a los elementos de la lista haciendo referencia al número de índice:

```
thislist = ["apple", "banana", "cherry"]
print(thislist[1])
```

Indexación negativa

La indexación negativa significa comenzar desde el final, `-1` se refiere al último elemento, `-2` se refiere al segundo último elemento, etc.

```
thislist = ["apple", "banana", "cherry"]
print(thislist[-1])
```

Rango de índices

Puede especificar un rango de índices especificando dónde empezar y dónde finalizar el intervalo.

Al especificar un intervalo, el valor devuelto será una nueva lista con los elementos especificados.

```
thislist = ["manzana", "banana", "frutilla", "Naranja", "kiwi", "melon", "mango"]
print(thislist[2:5])
# Esto devolverá los artículos de la posición 2 a la 5.
# Recuerde que el primer elemento es la posición 0,
#y tenga en cuenta que el elemento en la posición 5 NO está incluido
```

Nota: La búsqueda comenzará en el índice 2 (incluido) y finalizará en el índice 5 (no incluido). Recuerde que el primer elemento tiene el índice 0.

Al dejar fuera el valor inicial, el rango comenzará en el primer elemento

```
thislist = ["manzana", "banana", "frutilla", "Naranja", "kiwi", "melon", "mango"]
print(thislist[:4])
# Esto devolverá los elementos del índice 0 al índice 4.
# Recuerde que el índice 0 es el primer elemento, y el índice 4 es el quinto elemento
# Recuerde que el elemento en el índice 4 NO está incluido
```

Al dejar fuera el valor final, el rango pasará al final de la lista:

```
thislist = ["manzana", "banana", "frutilla", "Naranja", "kiwi", "melon", "mango"]
print(thislist[2:])
# Esto devolverá los elementos del índice 2 hasta el final.
# Recuerde que el índice 0 es el primer elemento, y el índice 2 es el tercero
```

Rango de índices negativos

Especifique los índices negativos si desea iniciar la búsqueda desde el final de la lista

Este ejemplo devuelve los elementos del índice -4 (incluido) al índice -1 (excluido)

```
thislist = ["manzana", "banana", "cereza", "naranja", "kiwi", "melon", "mango"]
print(thislist[-4:-1])
# La indexación negativa significa comenzar desde el final de la lista.
# Este ejemplo devuelve los elementos del índice -4 (incluido) al índice -1 (excluido)
# Recuerde que el último elemento tiene el índice -1,
```

Cambiar valor de elemento

Para cambiar el valor de un elemento específico, consulte el número de índice:

```
thislist = ["manzana", "banana", "cereza"]
thislist[1] = "coco"
print(thislist)
```

Bucle a través de una lista

Puede recorrer los elementos de la lista mediante un bucle:for

Imprima todos los elementos de la lista, uno por uno:

```
thislist = ["manzana", "banana", "cereza"]
for x in thislist:
  print(x)
```

Compruebe si el elemento existe

Para determinar si un elemento especificado está presente en una lista, utilice la palabra clave:in

Compruebe si "manzana" está presente en la lista:

```
thislist = ["manzana", "banana", "cereza"]
if "manzana" in thislist:
print("Si, 'manzana' esta en la lista de frutas")
```

Longitud de la lista

Para determinar cuántos elementos tiene una lista, utilice la función: len()

Imprima el número de elementos de la lista:

```
thislist = ["manzana", "banana", "cereza"]
print(len(thislist))
```

Añadir elementos

Para añadir un elemento al final de la lista, utilice el método append():

```
thislist = ["manzana", "banana", "cereza"]
thislist.append("naranja")
print(thislist)
```

Para añadir un elemento en el índice especificado, utilice el método insert():

```
thislist = ["manzana", "banana", "cereza"]
thislist.insert(1, "naranja")
print(thislist)
```

Eliminar elemento

Existen varios métodos para eliminar elementos de una lista:

El método remove() quita el elemento especificado:

```
thislist = ["manzana", "banana", "cereza"]
thislist.remove("banana")
print(thislist)
```

El método pop() quita el índice especificado (o el último elemento si no se especifica index):

```
thislist = ["manzana", "banana", "cereza"]
```

```
thislist.pop()
print(thislist)
```

La palabra clave del quita el índice especificado:

```
thislist = ["manzana", "banana", "cereza"]
del thislist[0]
print(thislist)
```

La palabra clave del también puede eliminar la lista por completo:

```
thislist = ["manzana", "banana", "cereza"]
del thislist
print(thislist) #this provocará un error porque ha eliminado correctamente "thislist".
```

NameError: el nombre 'thislist' no está definido

El método clear() vacía la lista:

```
thislist = ["manzana", "banana", "cereza"]
thislist.clear()
print(thislist)
```

Copiar una lista

No puede copiar una lista simplemente escribiendo , porque: sólo será una *referencia* a , y los cambios realizados en también se realizarán automáticamente en .list2 = list1list2list1list1list2

Hay maneras de hacer una copia, una forma es usar el método copy() List integrado.

```
thislist = ["manzana", "banana", "cereza"]
mylist = thislist.copy()
print(mylist)
```

Otra forma de hacer una copia es utilizar el método integrado.list()

```
thislist = ["manzana", "banana", "cereza"]
mylist = list(thislist)
print(mylist)
```

Unirse a dos listas

Hay varias maneras de unir, o concatenar, dos o más listas en Python.

Una de las formas más fáciles son mediante el uso del operador.+

```
list1 = ["a", "b" , "c"]
list2 = [1, 2, 3]
list3 = list1 + list2
print(list3)
```

Otra forma de unir dos listas es añadiendo todos los elementos de list2 a list1, uno por uno:

```
list1 = ["a", "b" , "c"]
list2 = [1, 2, 3]

for x in list2:
  list1.append(x)

print(list1)
```

O puede utilizar el método extend(), cuyo propósito es agregar elementos de una lista a otra lista:

```
list1 = ["a", "b" , "c"]
list2 = [1, 2, 3]

list1.extend(list2)
print(list1)
```

El constructor list()

También es posible utilizar el constructor list() para crear una nueva lista.

```
thislist = list(("manzana", "banana", "cereza")) # tener en cuenta los parentesis dobles
print(thislist)
```

Métodos de lista

Python tiene un conjunto de métodos integrados que puede usar en listas.

Método	Descripción
append()	Agrega un elemento al final de la lista
clear()	Elimina todos los elementos de la lista.
copy()	Devuelve una copia de la lista.
count()	Devuelve el número de elementos con el valor especificado.
extend()	Agregue los elementos de una lista (o cualquier iterable) al final de la lista actual
index()	Devuelve el índice del primer elemento con el valor especificado
insert()	Agrega un elemento en la posición especificada
pop()	Elimina el elemento en la posición especificada.
remove()	Elimina el elemento con el valor especificado.
reverse()	Invierte el orden de la lista
sort()	Ordena la lista

Tupla

Una tupla es una colección que se ordena e **inmutable.** En Python las tuplas se escriben con parentesis.

```
thistuple = ("manzana", "banana", "cereza")
print(thistuple)
```

Algunos de los métodos igual a List

Establecer (Set)

Un conjunto es una colección que no está ordenada y no está indizada. En Python, los conjuntos se escriben entre llaves.

```
thisset = {"manzana", "banana", "cereza"}
print(thisset)
# Nota: la lista establecida no está ordenada, lo que significa que los elementos aparecerán en un orden aleatorio.
# Actualice esta página para ver el cambio en el resultado.
```

Nota: Los conjuntos no están ordenados, por lo que no puede estar seguro en qué orden aparecerán los elementos.

Acceder a los elementos

No se puede tener acceso a los elementos de un conjunto haciendo referencia a un índice, ya que los conjuntos no están ordenados los elementos no tienen ningún índice.

Pero puede recorrer los elementos de conjunto mediante un bucle for o preguntar si un valor especificado está presente en un conjunto, mediante la palabra clave. in

```
thisset = {"manzana", "banana", "cereza"}
for x in thisset:
  print(x)
```

Compruebe si "banana" está presente en el set:

```
thisset = {"manzana", "banana", "cereza"}
print("banana" in thisset)
```

Cambiar elementos

Una vez creado un conjunto, no puede cambiar sus elementos, pero puede agregar nuevos elementos.

Añadir elementos

Para agregar un elemento a un conjunto, utilice el método.add()

Para agregar más de un elemento a un conjunto, utilice el método.update()

```
thisset = {"manzana", "banana", "cereza"}
thisset.add("naranja")
print(thisset)
```

Agregue varios elementos a un conjunto, utilizando el método:update()

```
thisset = {"manzana", "banana", "cereza"}
thisset.update(["naranja", "mango", "peras"])
print(thisset)
```

Obtener la longitud de un conjunto

Para determinar cuántos elementos tiene un conjunto, utilice el método.len()

```
thisset = {"apple", "banana", "cherry"}

print(len(thisset))
```

Eliminar elemento

Para quitar un elemento de un conjunto, utilice el método remove() , o el método. discard()

```
thisset = {"manzana", "banana", "cereza"}
thisset.remove("banana")
thisset.discard("cereza")
print(thisset)
```

Nota: Si el elemento que se va a quitar no existe, remove() generará un error.

Nota: Si el elemento que se va a quitar no existe, discard() **NO** generará un error.

También puede usar el método pop(), para quitar un elemento, pero este método quitará el *último* elemento. Recuerde que los conjuntos no están ordenados, por lo que no sabrá qué elemento se elimina.

El valor devuelto del método pop() es el elemento eliminado.

```
thisset = {"manzana", "banana", "cereza"}
```

```
x = thisset.pop()

print(x)

print(thisset)
```

El método clear() vacía el conjunto:

```
thisset = {"manzana", "banana", "cereza"}
thisset.clear()
print(thisset)
```

La palabra clave del eliminará el conjunto por completo:

```
thisset = {"apple", "banana", "cherry"}
del thisset
print(thisset) # esto generará un error porque el conjunto ya no existe
```

Unirse a dos conjuntos

Hay varias maneras de unir dos o más conjuntos en Python.

Puede utilizar el método union() que devuelve un nuevo conjunto que contiene todos los elementos de ambos conjuntos o el método update() que inserta todos los elementos de un conjunto en otro:

El método union() devuelve un nuevo conjunto con todos los elementos de ambos conjuntos:

```
set1 = {"a", "b" , "c"}
set2 = {1, 2, 3}
set3 = set1.union(set2)
print(set3)
```

El método update() inserta los elementos de set2 en set1:

```
set1 = {"a", "b" , "c"}
set2 = {1, 2, 3}
set1.update(set2)
print(set1)
```

Nota: Ambos y excluirá cualquier elemento duplicado.union()update()

El constructor set()

También es posible utilizar el constructor set() para crear un conjunto.

```
thisset = set(("manzana", "banana", "cereza"))
print(thisset)#Nota: la lista establecida no está ordenada, por lo que el resultado mostrará los elementos en un orden aleatorio.
```

Establecer métodos

Python tiene un conjunto de métodos integrados que puede usar en conjuntos.

Método	Descripción
add()	Agrega un elemento al conjunto
clear()	Elimina todos los elementos del conjunto.
copy()	Devuelve una copia del conjunto.
difference()	Devuelve un conjunto que contiene la diferencia entre dos o más conjuntos
difference_update()	Elimina los elementos de este conjunto que también se incluyen en otro conjunto especificado
discard()	Eliminar el elemento especificado
intersection()	Devuelve un conjunto, es decir, la intersección de otros dos conjuntos.
intersection_update()	Elimina los elementos de este conjunto que no están presentes en

	otros conjuntos especificados.
isdisjoint()	Devuelve si dos conjuntos tienen una intersección o no
issubset()	Devuelve si otro conjunto contiene este conjunto o no
issuperset()	Devuelve si este conjunto contiene otro conjunto o no
pop()	Elimina un elemento del conjunto.
remove()	Elimina el elemento especificado.
symmetric_difference()	Devuelve un conjunto con las diferencias simétricas de dos conjuntos.
symmetric_difference_update()	inserta las diferencias simétricas de este conjunto y otro
union()	Devuelve un conjunto que contiene la unión de conjuntos
update()	Actualice el conjunto con la unión de este conjunto y otros

Diccionario

Un diccionario es una colección que es desordenada, cambiable e indizada. En Python, los diccionarios se escriben entre llaves y tienen claves y valores.

```
thisdict =	{
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
```

```
}
print(thisdict)
```

Acceso a los elementos

Puede acceder a los elementos de un diccionario haciendo referencia a su nombre de clave, entre corchetes:

```
thisdict =	{
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
x = thisdict["model"]
print(x)
```

También hay un método llamado `get()` que le dará el mismo resultado:

```
thisdict =	{
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
x = thisdict.get("model")
print(x)
```

Cambiar valores

Puede cambiar el valor de un elemento específico haciendo referencia a su nombre de clave:

```
thisdict =	{
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}

thisdict["year"] = 2018
```

```
print(thisdict)
```

Bucle a través de un diccionario

Puede recorrer un bucle a través de un diccionario mediante un bucle.for

Al recorrer en bucle un diccionario, el valor devuelto son *las claves* del diccionario, pero también hay métodos para devolver los *valores*.

```
thisdict =	{
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
for x in thisdict:
  print(x)
```

Imprima todos los *valores* en el diccionario, uno por uno:

```
thisdict =	{
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
for x in thisdict:
  print(thisdict[x])
```

También puede utilizar el método values()para devolver valores de un diccionario:

```
thisdict =	{
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
for x in thisdict.values():
  print(x)
```

Recorra ambas *claves* y *valores*mediante el método:items()

```
thisdict =	{
```

```
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
for x, y in thisdict.items():
  print(x, y)
```

Compruebe si la clave existe

Para determinar si una clave especificada está presente en un diccionario, utilice la palabra clave:in

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
if "model" in thisdict:
  print("Yes, 'model' is one of the keys in the thisdict dictionary")
```

Adición de elementos

La adición de un elemento al diccionario se realiza mediante una nueva clave de índice y asignándole un valor:

```
thisdict =          {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict["color"] = "red"
print(thisdict)
```

Eliminación de elementos

Existen varios métodos para eliminar elementos de un diccionario:

El método pop() quita el elemento con el nombre de clave especificado:

```
thisdict =	{
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict.pop("model")
print(thisdict)
```

El método popitem() quita el último elemento insertado):

```
thisdict =	{
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict.popitem()
print(thisdict)
```

La palabra clave del quita el elemento con el nombre de clave especificado:

```
thisdict =	{
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
del thisdict["model"]
print(thisdict)
```

La palabra clave del también puede eliminar el diccionario por completo:

```
thisdict =	{
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
del thisdict
```

```
print(thisdict) # esto causará un error porque "thislist" ya no existe.
```

El método vacía el diccionario:clear()

```
thisdict =          {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict.clear()
print(thisdict)
```

Copiar un diccionario

No se puede copiar un diccionario simplemente escribiendo , porque: sólo será una *referencia* a , y los cambios realizados en también se realizarán automáticamente en .dict2 = dict1dict2dict1dict1dict2

Hay maneras de hacer una copia, una forma es usar el método Dictionary integrado. copy()

Haga una copia de un diccionario con el método:copy()

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
mydict = thisdict.copy()
print(mydict)
```

Otra forma de hacer una copia es utilizar la función dict() integrada .

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
mydict = dict(thisdict)
print(mydict)
```

Diccionarios anidados

Un diccionario también puede contener muchos diccionarios, esto se denomina diccionarios anidados.

```
myfamily = {
  "child1" : {
            "name" : "Emil",
            "year" : 2004
  },
  "child2" : {
            "name" : "Tobias",
            "year" : 2007
  },
  "child3" : {
            "name" : "Linus",
            "year" : 2011
  }
}

print(myfamily)
```

O bien, si desea anidar tres diccionarios que ya existen como diccionarios:

```
child1 = {
  "name" : "Emil",
  "year" : 2004
}
child2 = {
  "name" : "Tobias",
  "year" : 2007
}
child3 = {
  "name" : "Linus",
  "year" : 2011
```

```
}

myfamily = {
  "child1" : child1,
  "child2" : child2,
  "child3" : child3
}

print(myfamily)
```

El constructor dict()

También es posible utilizar el constructor dict() para crear un nuevo diccionario:

```
thisdict = dict(brand="Ford", model="Mustang", year=1964)
# tenga en cuenta que las palabras clave no son literales de cadena
# tenga en cuenta el uso de iguales en lugar de dos puntos para la asignación
print(thisdict)
```

Métodos de diccionario

Método	Descripción
clear()	Elimina todos los elementos del diccionario.
copy()	Devuelve una copia del diccionario.

fromkeys()	Devuelve un diccionario con las claves y el valor especificados
get()	Devuelve el valor de la clave especificada
items()	Devuelve una lista que contiene una tupla para cada par de valores clave
keys()	Devuelve una lista que contiene las claves del diccionario
pop()	Elimina el elemento con la clave especificada
popitem()	Elimina el último par clave-valor insertado
setdefault()	Devuelve el valor de la clave especificada. Si la clave no existe: inserte la clave, con el valor especificado

update()	Actualiza el diccionario con los pares clave-valor especificados.
values()	Devuelve una lista de todos los valores del diccionario.

Fuentes:

https://www.mclibre.org/consultar/python/lecciones/python-booleanos.html

https://es.khanacademy.org/computing/computer-programming/programming/logic-if-statements/a/review-logic-and-if-statements

https://plataforma.josedomingo.org/pledin/cursos/programacion_python3/curso/u31/

https://entrenamiento-python-basico.readthedocs.io/es/latest/leccion3/tipo_conjuntos.html