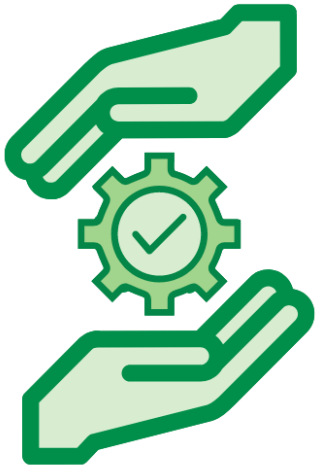

Bootcamp de Desarrollo Web

Clase 14

Python – Estructuras de flujo

if – Sentencia básica

- En Python, la sentencia if se utiliza para ejecutar un bloque de código si, y solo si, se cumple una determinada condición. Por tanto, if es usado para la toma de decisiones.

- La estructura básica de esta sentencia if es la siguiente:

```
x = 17
if x < 20:
    print('x es menor que 20')
```

Sentencia if … else

- Hay ocasiones en que la sentencia if básica no es suficiente y es necesario ejecutar un conjunto de instrucciones o sentencias cuando la condición se evalúa a False.

- Para ello se utiliza la estructura if ... else... Esta es estructura es como sigue:

```
1.  if condición:
2.      bloque de código (cuando condición se evalúa a True)
3.  else:
4.      bloque de código 2 (cuando condición se evalúa a False)
```

if … elif … else

También es posible que te encuentres situaciones en que una decisión dependa de más de una condición.

En estos casos se usa una sentencia if compuesta, cuya estructura es como se indica a continuación:

```
1.  if cond1:
2.      bloque cond1 (sentencias si se evalúa la cond1 a True)
3.  elif cond2:
4.      bloque cond2 (sentencias si cond1 es False pero cond2 es True)
5.  ...
6.  else:
7.      bloque else (sentencias si todas las condiciones se evalúan a False)
```

Sentencias if anidadas

Cualquiera de los bloques de sentencias anteriores se puede volver a incluir una sentencia if, o if … else … o if … elif … else

```
1.  x = 28
2.  if x < 0:
3.      print(f'{x} es menor que 0')
4.  else:
5.      if x > 0:
6.          print(f'{x} es mayor que 0')
7.      else:
8.          print('x es 0')
```

Tipos de datos complejos

¿Qué es una lista?

Las listas en Python son un tipo contenedor, compuesto, que se usan para almacenar conjuntos de elementos relacionados del mismo tipo o de tipos distintos.

Junto a las clases **tuple, range y str**, son uno de los tipos de secuencia en Python, con la particularidad de que son mutables. Esto último quiere decir que su contenido se puede modificar después de haber sido creada.

Para crear una lista en Python, simplemente hay que encerrar una secuencia de elementos separados por comas entre paréntesis cuadrados [].

¿Qué es una lista?

Por ejemplo, para crear una lista con los números del 1 al 10 se haría del siguiente modo:

```
>>> numeros = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

```
>>> elementos = [3, 'a', 8, 7.2, 'hola']
```

¿Qué es una lista?

Las listas también se pueden crear usando el constructor de la clase, list(iterable). En este caso, el constructor crea una lista cuyos elementos son los mismos y están en el mismo orden que los ítems del iterable. El objeto iterable puede ser o una secuencia, un contenedor que soporte la iteración o un objeto iterador.

Por ejemplo, el tipo str también es un tipo secuencia. Si pasamos un string al constructor list() creará una lista cuyos elementos son cada uno de los caracteres de la cadena:

¿Qué es una lista?

Las listas también se pueden crear usando el constructor de la clase, list(iterable). En este caso, el constructor crea una lista cuyos elementos son los mismos y están en el mismo orden que los ítems del iterable. El objeto iterable puede ser o una secuencia, un contenedor que soporte la iteración o un objeto iterador.

Por ejemplo, el tipo str también es un tipo secuencia. Si pasamos un string al constructor list() creará una lista cuyos elementos son cada uno de los caracteres de la cadena:

```
1.  >>> vocales = list('aeiou')
2.  >>> vocales
3.  ['a', 'e', 'i', 'o', 'u']
```

Cómo acceder a los elementos de una lista en Python

Para acceder a un elemento de una lista se utilizan los índices. Un índice es un número entero que indica la posición de un elemento en una lista. El primer elemento de una lista siempre comienza en el índice 0.

Por ejemplo, en una lista con 4 elementos, los índices de cada uno de los ítems serían 0, 1, 2 y 3.

```
>>> lista = ['a', 'b', 'd', 'i', 'j']
>>> lista[0]  # Primer elemento de la lista. Índice 0
'a'
>>> lista[3]  # Cuarto elemento de la lista. Índice 3
'i'
```

Cómo acceder a los elementos de una lista en Python

Si se intenta acceder a un índice que está fuera del rango de la lista, el intérprete lanzará la excepción IndexError. De igual modo, si se utiliza un índice que no es un número entero, se lanzará la excepción TypeError:

```
>>> lista = [1, 2, 3]  # Los índices válidos son 0, 1 y 2
>>> lista[8]
Traceback (most recent call last):
  File "<input>", line 1, in <module>
IndexError: list index out of range

>>> lista[1.0]
Traceback (most recent call last):
  File "<input>", line 1, in <module>
TypeError: list indices must be integers or slices, not float
```

Acceso a los elementos usando un índice negativo

En Python está permitido usar índices negativos para acceder a los elementos de una secuencia. En este caso, el índice -1 hace referencia al último elemento de la secuencia, el -2 al penúltimo y así, sucesivamente:

```
1.  >>> vocales = ['a', 'e', 'i', 'o', 'u']
2.  >>> vocales[-1]
3.  'u'
4.  >>> vocales[-4]
5.  'e'
```

Acceso a un subconjunto de elementos

También es posible acceder a un subconjunto de elementos de una lista utilizando rangos en los índices. Esto es usando el operador [:]:

```
>>> vocales = ['a', 'e', 'i', 'o', 'u']
>>> vocales[2:3]  # Elementos desde el índice 2 hasta el índice 3-1
['i']
>>> vocales[2:4]  # Elementos desde el 2 hasta el índice 4-1
['i', 'o']
>>> vocales[:]  # Todos los elementos
['a', 'e', 'i', 'o', 'u']
>>> vocales[1:]  # Elementos desde el índice 1
['e', 'i', 'o', 'u']
>>> vocales[:3]  # Elementos hasta el índice 3-1
['a', 'e', 'i']
```

También es posible acceder a los elementos de una lista indicando un paso con el operador [::]:

También es posible acceder a un subconjunto de elementos de una lista utilizando rangos en los índices. Esto es usando el operador [:]:

```
>>> letras = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k']
>>> letras[::2]  # Acceso a los elementos de 2 en 2
['a', 'c', 'e', 'g', 'i', 'k']
>>> letras[1:5:2]  # Elementos del índice 1 al 4 de 2 en 2
['b', 'd']
>>> letras[1:6:3]  # Elementos del índice 1 al 5 de 3 en 3
['b', 'e']
```

for list Python – Recorrer una lista

Para recorrer una lista en Python utilizaríamos la siguiente estructura:

```
>>> colores = ['azul', 'blanco', 'negro']
>>> for color in colores:
        print(color)

azul
blanco
negro
```

Añadir elementos a una lista en Python

Las listas son secuencias mutables, es decir, sus elementos pueden ser modificados (se pueden añadir nuevos ítems, actualizar o eliminar).

Para añadir un nuevo elemento a una lista se utiliza el método append() y para añadir varios elementos, el método extend():

```
>>> vocales = ['a']
>>> vocales.append('e')  # Añade un elemento
>>> vocales
['a', 'e']

>>> vocales.extend(['i', 'o', 'u'])  # Añade un grupo de elementos
>>> vocales
['a', 'e', 'i', 'o', 'u']
```

Añadir elementos a una lista en Python

También es posible utilizar el operador de concatenación + para unir dos listas en una sola. El resultado es una nueva lista con los elementos de ambas:

```
>>> lista_1 = [1, 2, 3]
>>> lista_2 = [4, 5, 6]
>>> nueva_lista = lista_1 + lista_2
>>> nueva_lista
[1, 2, 3, 4, 5, 6]
```

Por otro lado, el operador * repite el contenido de una lista n veces:

```
>>> numeros = [1, 2, 3]
>>> numeros *= 3
>>> numeros
[1, 2, 3, 1, 2, 3, 1, 2, 3]
```

Añadir elementos a una lista en Python

También es posible añadir un elemento en una posición concreta de una lista con el método **insert(índice, elemento).** Los elementos cuyo índice sea mayor a índice se desplazan una posición a la derecha:

```
>>> vocales = ['a', 'e', 'u']
>>> vocales.insert(2, 'i')
>>> vocales
['a', 'e', 'i', 'u']
```

Modificar elementos de una lista

Es posible modificar un elemento de una lista en Python con el operador de asignación =. Para ello, lo único que necesitas conocer es el índice del elemento que quieres modificar o el rango de índices:

```
>>> vocales = ['o', 'o', 'o', 'o', 'u']

# Actualiza el elemento del índice 0
>>> vocales[0] = 'a'
>>> vocales
['a', 'o', 'o', 'o', 'u']

# Actualiza los elementos entre las posiciones 1 y 2
>>> vocales[1:3] = ['e', 'i']
>>> vocales
['a', 'e', 'i', 'o', 'u']
```

Eliminar un elemento de una lista en Python

En Python se puede eliminar un elemento de una lista de varias formas.

Con la sentencia del se puede eliminar un elemento a partir de su índice:

```
# Elimina el elemento del índice 1
>>> vocales = ['a', 'e', 'i', 'o', 'u']
>>> del vocales[1]
>>> vocales
['a', 'i', 'o', 'u']

# Elimina los elementos con índices 2 y 3
>>> vocales = ['a', 'e', 'i', 'o', 'u']
>>> del vocales[2:4]
>>> vocales
['a', 'e', 'u']

# Elimina todos los elementos
>>> del vocales[:]
>>> vocales
[]
```

Eliminar un elemento de una lista en Python

Además de la sentencia del, podemos usar los métodos remove() y pop([i]). remove() elimina la primera ocurrencia que se encuentre del elemento en una lista. Por su parte, pop([i]) obtiene el elemento cuyo índice sea igual a i y lo elimina de la lista. Si no se especifica ningún índice, recupera y elimina el último elemento.

```
>>> letras = ['a', 'b', 'k', 'a', 'v']

# Elimina la primera ocurrencia del carácter a
>>> letras.remove('a')
>>> letras
['b', 'k', 'a', 'v']

# Obtiene y elimina el último elemento
>>> letras.pop()
'v'
>>> letras
['b', 'k', 'a']
```

Eliminar un elemento de una lista en Python

Finalmente, es posible eliminar todos los elementos de una lista a través del método clear():

```
1.  >>> letras = ['a', 'b', 'c']
2.  >>> letras.clear()
3.  >>> letras
4.  []
```

Longitud (len) de una lista en Python

Como cualquier tipo secuencia, para conocer la longitud de una lista en Python se hace uso de la función len(). Esta función devuelve el número de elementos de una lista:

```
1.  >>> vocales = ['a', 'e', 'i', 'o', 'u']
2.  >>> len(vocales)
3.  5
```

Cómo saber si un elemento está en una lista en Python

Para saber si un elemento está contenido en una lista, se utiliza el operador de pertenencia in:

```
>>> vocales = ['a', 'e', 'i', 'o', 'u']
>>> if 'a' in vocales:
...     print('Sí')
...
Sí
>>> if 'b' not in vocales:
...     print('No')
...
No
```

sort list Python – Ordenar una lista en Python

Las listas son secuencias ordenadas. Esto quiere decir que sus elementos siempre se devuelven en el mismo orden en que fueron añadidos.

No obstante, es posible ordenar los elementos de una lista con el método sort(). El método sort() ordena los elementos de la lista utilizando únicamente el operador < y modifica la lista actual (no se obtiene una nueva lista):

sort list Python – Ordenar una lista en Python

```
1.  # Lista desordenada de números enteros
2.  >>> numeros = [3, 2, 6, 1, 7, 4]
3.
4.  # Identidad del objeto numeros
5.  >>> id(numeros)
6.  4475439216
7.
8.  # Se llama al método sort() para ordenar los elementos de la lista
9.  >>> numeros.sort()
10. >>> numeros
11. [1, 2, 3, 4, 6, 7]
12.
13. # Se comprueba que la identidad del objeto numeros es la misma
14. >>> id(numeros)
15. 4475439216
```

Listado de métodos de la clase list

- **append():**Añade un nuevo elemento al final de la lista.
- **extend():**Añade un grupo de elementos (iterables) al final de la lista.
- **insert(indice, elemento):**Inserta un elemento en una posición concreta de la lista.
- **remove(elemento):**Elimina la primera ocurrencia del elemento en la lista.

Listado de métodos de la clase list

- **pop([i]):** Obtiene y elimina el elemento de la lista en la posición i. Si no se especifica, obtiene y elimina el último elemento.

- **clear():** Borra todos los elementos de la lista.

- **index(elemento):**Obtiene el índice de la primera ocurrencia del elemento en la lista. Si el elemento no se encuentra, se lanza la excepción ValueError.

- **count(elemento):**Devuelve el número de ocurrencias del elemento en la lista.

Listado de métodos de la clase list

- **sort():**Ordena los elementos de la lista utilizando el operador <.
- **reverse():**Obtiene los elementos de la lista en orden inverso.
- **copy():**Devuelve una copia poco profunda de la lista.

Tuplas

Qué es una tupla

La clase tuple en Python es un tipo contenedor, compuesto, que en un principio se pensó para almacenar grupos de elementos heterogéneos, aunque también puede contener elementos homogéneos.

Junto a las clases list y range, es uno de los tipos de secuencia en Python, con la particularidad de que son inmutables. Esto último quiere decir que su contenido NO se puede modificar después de haber sido creada.

En general, para crear una tupla en Python simplemente hay que definir una secuencia de elementos separados por comas.

Qué es una tupla

Ejemplos:

```
1.  >>> numeros = 1, 2, 3, 4, 5
```

```
1.  >>> elementos = 3, 'a', 8, 7.2, 'hola'
```

```
>>> tup = 1, ['a', 'e', 'i', 'o', 'u'], 8.9, 'hola'
```

Crear una tupla

A continuación te indico las diferentes formas que existen de crear una tupla en Python:

- Para crear una tupla vacía, usa paréntesis () o el constructor de la clase tuple() sin parámetros.

- Para crear una tupla con un único elemento: elem, o (elem,). Observa que siempre se añade una coma.

- Para crear una tupla de varios elementos, sepáralos con comas: a, b, c o (a, b, c).

Crear una tupla

Las tuplas también se pueden crear usando el constructor de la clase, tuple(iterable). En este caso, el constructor crea una tupla cuyos elementos son los mismos y están en el mismo orden que los ítems del iterable. El objeto iterable puede ser una secuencia, un contenedor que soporte la iteración o un objeto iterador.

```
# Aquí, a, b y c no son una tupla, sino tres argumentos con
# los que se llama a la función "una_funcion"
>>> una_funcion(a, b, c)

# Aquí, a, b y c son tres elementos de una tupla. Esta tupla,
# es el único argumento con el que se invoca a la
# función "una_funcion"
>>> una_funcion((a, b, c))
```

Cómo acceder a los elementos de una tupla en Python

Para acceder a un elemento de una tupla se utilizan los índices. Un índice es un número entero que indica la posición de un elemento en una tupla. El primer elemento de una tupla siempre comienza en el índice 0.

Por ejemplo, en una tupla con 3 elementos, los índices de cada uno de los ítems serían 0, 1 y 2.

```
>>> tupla = ('a', 'b', 'd')
>>> tupla[0]  # Primer elemento de la tupla. Índice 0
'a'
>>> tupla[1]  # Segundo elemento de la tupla. Índice 1
'b'
```

Cómo acceder a los elementos de una tupla en Python

Si se intenta acceder a un índice que está fuera del rango de la tupla, el intérprete lanzará la excepción IndexError. De igual modo, si se utiliza un índice que no es un número entero, se lanzará la excepción TypeError:

```
>>> tupla = 1, 2, 3  # Los índices válidos son 0, 1 y 2
>>> tupla[8]
Traceback (most recent call last):
  File "<input>", line 1, in <module>
IndexError: tuple index out of range

>>> tupla[1.0]
Traceback (most recent call last):
  File "<input>", line 1, in <module>
TypeError: tuple indices must be integers or slices, not float
```

tuple unpacking

En este último apartado de esta sección vamos a ver el concepto conocido como tuple unpacking (desempaquetado de una tupla). Realmente el unpacking se puede aplicar sobre cualquier objeto de tipo secuencia, aunque se usa mayoritariamente con las tuplas, y consiste en lo siguiente:

```
1.  >>> bebidas = 'agua', 'café', 'batido'
2.  >>> a, b, c = bebidas
3.  >>> a
4.  'agua'
5.  >>> b
6.  'café'
7.  >>> c
8.  'batido'
```

for tuple Python – Recorrer una tupla

El bucle for en Python es una de las estructuras ideales para iterar sobre los elementos de una secuencia. Para recorrer una tupla en Python utiliza la siguiente estructura:

```
>>> colores = 'azul', 'blanco', 'negro'
>>> for color in colores:
        print(color)

azul
blanco
negro
```

Listado de métodos de la clase tuple en Python

Para terminar, se muestran los métodos de la clase tuple en Python, que son los métodos definidos para cualquier tipo secuencial:

index(elemento): Obtiene el índice de la primera ocurrencia del elemento en la tupla. Si el elemento no se encuentra, se lanza la excepción ValueError.

count(elemento): Devuelve el número de ocurrencias del elemento en la tupla.

Diccionario

Qué es el tipo dict en Python

La clase dict de Python es un tipo mapa que asocia claves a valores. A diferencia de los tipos secuenciales (list, tuple, range o str), que son indexados por un índice numérico, los diccionarios son indexados por claves. Estas claves siempre deben ser de un tipo inmutable, concretamente un tipo hashable.

> 🎯 **NOTA:** Un objeto es *hashable* si tiene un valor de *hash* que no cambia durante todo su ciclo de vida. En principio, los objetos que son instancias de clases definidas por el usuario son *hashables*. También lo son la mayoría de tipos inmutables definidos por Python (*int*, *float* o *str*).

Qué es el tipo dict en Python

Piensa siempre en un diccionario como un contenedor de pares clave: valor, en el que la clave puede ser de cualquier tipo hashable y es única en el diccionario que la contiene. Generalmente, se suelen usar como claves los tipos int y str aunque, como te he dicho, cualquier tipo hashable puede ser una clave.

Las principales operaciones que se suelen realizar con diccionarios son almacenar un valor asociado a una clave y recuperar un valor a partir de una clave. Esta es la esencia de los diccionarios y es aquí donde son realmente importantes.

Cómo crear un diccionario

En Python hay varias formas de crear un diccionario. Las veremos todas a continuación.

La más simple es encerrar una secuencia de pares clave: valor separados por comas entre llaves {}

```
1.  >>> d = {1: 'hola', 89: 'Pythonista', 'a': 'b', 'c': 27}
```

- En el diccionario anterior, los enteros 1 y 89 y las cadenas 'a' y 'c' son las claves. Como ves, se pueden mezclar claves y valores de distinto tipo sin problema.
- Para crear un diccionario vacío, simplemente asigna a una variable el valor {}.

Cómo crear un diccionario

```
# 1. Pares clave: valor encerrados entre llaves
>>> d = {'uno': 1, 'dos': 2, 'tres': 3}
>>> d
{'uno': 1, 'dos': 2, 'tres': 3}

# 2. Argumentos con nombre
>>> d2 = dict(uno=1, dos=2, tres=3)
>>> d2
{'uno': 1, 'dos': 2, 'tres': 3}

# 3. Pares clave: valor encerrados entre llaves
>>> d3 = dict({'uno': 1, 'dos': 2, 'tres': 3})
>>> d3
{'uno': 1, 'dos': 2, 'tres': 3}

# 4. Iterable que contiene iterables con dos elementos
>>> d4 = dict([('uno', 1), ('dos', 2), ('tres', 3)])
>>> d4
{'uno': 1, 'dos': 2, 'tres': 3}

# 5. Diccionario vacío
>>> d5 = {}
>>> d5
{}

# 6. Diccionario vacío usando el constructor
>>> d6 = dict()
>>> d6
{}
```

Cómo acceder a los elementos de un diccionario en Python

Acceder a un elemento de un diccionario es una de las principales operaciones por las que existe este tipo de dato. El acceso a un valor se realiza mediante indexación de la clave. Para ello, simplemente encierra entre corchetes la clave del elemento d[clave]. En caso de que la clave no exista, se lanzará la excepción KeyError.

```
>>> d = {'uno': 1, 'dos': 2, 'tres': 3}
>>> d['dos']
2
>>> d[4]
Traceback (most recent call last):
  File "<input>", line 1, in <module>
KeyError: 4
```

Cómo acceder a los elementos de un diccionario en Python

La clase dict también ofrece el método get(clave[, valor por defecto]). Este método devuelve el valor correspondiente a la clave clave. En caso de que la clave no exista no lanza ningún error, sino que devuelve el segundo argumento valor por defecto. Si no se proporciona este argumento, se devuelve el valor None.

```
>>> d = {'uno': 1, 'dos': 2, 'tres': 3}
>>> d.get('uno')
1

# Devuelve 4 como valor por defecto si no encuentra la clave
>>> d.get('cuatro', 4)
4

# Devuelve None como valor por defecto si no encuentra la clave
>>> a = d.get('cuatro')
>>> a
>>> type(a)
<class 'NoneType'>
```

for dict Python – Recorrer un diccionario

Hay varias formas de recorrer los elementos de un diccionario: recorrer solo las claves, solo los valores o recorrer a la vez las claves y los valores. Puedes ver aquí cómo usar el bucle for para recorrer un diccionario.

```
>>> d = {'uno': 1, 'dos': 2, 'tres': 3}
>>> for e in d:
...     print(e)
...
uno
dos
tres

# Recorrer las claves del diccionario
>>> for k in d.keys():
...     print(k)
...
uno
dos
tres

# Recorrer los valores del diccionario
>>> for v in d.values():
...     print(v)
...
1
2
3

# Recorrer los pares clave valor
>>> for i in d.items():
...     print(i)
...
('uno', 1)
('dos', 2)
('tres', 3)
```

Añadir elementos a un diccionario en Python

La clase **dict** es mutable, por lo que se pueden añadir, modificar y/o eliminar elementos después de haber creado un objeto de este tipo.

Para añadir un nuevo elemento a un diccionario existente, se usa el operador de asignación =. A la izquierda del operador aparece el objeto diccionario con la nueva clave entre corchetes [] y a la derecha el valor que se asocia a dicha clave.

```
>>> d = {'uno': 1, 'dos': 2}
>>> d
{'uno': 1, 'dos': 2}

# Añade un nuevo elemento al diccionario
>>> d['tres'] = 3
>>> d
{'uno': 1, 'dos': 2, 'tres': 3}
```

Añadir elementos a un diccionario en Python

También existe el método setdefault(clave[, valor]). Este método devuelve el valor de la clave si ya existe y, en caso contrario, le asigna el valor que se pasa como segundo argumento. Si no se especifica este segundo argumento, por defecto es None.

```
>>> d = {'uno': 1, 'dos': 2}
>>> d.setdefault('uno', 1.0)
1
>>> d.setdefault('tres', 3)
3
>>> d.setdefault('cuatro')
>>> d
{'uno': 1, 'dos': 2, 'tres': 3, 'cuatro': None}
```

Modificar elementos de un diccionario

En el apartado anterior hemos visto que para actualizar el valor asociado a una clave, simplemente se asigna un nuevo valor a dicha clave del diccionario.

```
>>> d = {'uno': 1, 'dos': 2}
>>> d
{'uno': 1, 'dos': 2}
>>> d['uno'] = 1.0
>>> d
{'uno': 1.0, 'dos': 2}
```

Eliminar un elemento de un diccionario en Python

En Python existen diversos modos de eliminar un elemento de un diccionario. Son los siguientes:

pop(clave [, valor por defecto]): Si la clave está en el diccionario, elimina el elemento y devuelve su valor; si no, devuelve el valor por defecto. Si no se proporciona el valor por defecto y la clave no está en el diccionario, se lanza la excepción KeyError.

popitem(): Elimina el último par clave: valor del diccionario y lo devuelve. Si el diccionario está vacío se lanza la excepción KeyError. (NOTA: En versiones anteriores a Python 3.7, se elimina/devuelve un par aleatorio, no se garantiza que sea el último).

Eliminar un elemento de un diccionario en Python

del d[clave]: Elimina el par clave: valor. Si no existe la clave, se lanza la excepción KeyError.

clear(): Borra todos los pares clave: valor del diccionario.

```
>>> d = {'uno': 1, 'dos': 2, 'tres': 3, 'cuatro': 4, 'cinco': 5}

# Elimina un elemento con pop()
>>> d.pop('uno')
1
>>> d
{'dos': 2, 'tres': 3, 'cuatro': 4, 'cinco': 5}

# Trata de eliminar una clave con pop() que no existe
>>> d.pop(6)
Traceback (most recent call last):
  File "<input>", line 1, in <module>
KeyError: 6

# Elimina un elemento con popitem()
>>> d.popitem()
('cinco', 5)
>>> d
{'dos': 2, 'tres': 3, 'cuatro': 4}

# Elimina un elemento con del
>>> del d['tres']
>>> d
{'dos': 2, 'cuatro': 4}

# Trata de eliminar una clave con del que no existe
>>> del d['seis']
Traceback (most recent call last):
  File "<input>", line 1, in <module>
KeyError: 'seis'

# Borra todos los elementos del diccionario
>>> d.clear()
>>> d
{}
```

Comparar si dos diccionarios son iguales

En Python se puede utilizar el operador de igualdad == para comparar si dos diccionarios son iguales. Dos diccionarios son iguales si contienen el mismo conjunto de pares clave: valor, independientemente del orden que tengan.

Otro tipo de comparaciones entre diccionarios no están permitidas. Si se intenta, el intérprete lanzará la excepción TypeError.

```
>>> d1 = {'uno': 1, 'dos': 2}
>>> d2 = {'dos': 2, 'uno': 1}
>>> d3 = {'uno': 1}
>>> print(d1 == d2)
True
>>> print(d1 == d3)
False
>>> print(d1 > d2)
Traceback (most recent call last):
  File "<input>", line 1, in <module>
TypeError: '>' not supported between instances of 'dict' and 'dict'
```

Diccionarios anidados en Python

Un diccionario puede contener un valor de cualquier tipo, entre ellos, otro diccionario. Este hecho se conoce como diccionarios anidados.

Para acceder al valor de una de las claves de un diccionario interno, se usa el operador de indexación anidada [clave1][clave2]...

```
>>> d = {'d1': {'k1': 1, 'k2': 2}, 'd2': {'k1': 3, 'k4': 4}}
>>> d['d1']['k1']
1
>>> d['d2']['k1']
3
>>> d['d2']['k4']
4
>>> d['d3']['k4']
Traceback (most recent call last):
  File "<input>", line 1, in <module>
KeyError: 'd3'
```

Listado de métodos de la clase dict

- **clear():**Elimina todos los elementos del diccionario.
- **copy():**Devuelve una copia poco profunda del diccionario.
- **get(clave[, valor]):**Devuelve el valor de la clave. Si no existe, devuelve el valor valor si se indica y si no, None.
- **items():**Devuelve una vista de los pares clave: valor del diccionario.
- **keys():**Devuelve una vista de las claves del diccionario.

Listado de métodos de la clase dict

- **pop(clave[, valor]):** Devuelve el valor del elemento cuya clave es clave y elimina el elemento del diccionario. Si la clave no se encuentra, devuelve valor si se proporciona. Si la clave no se encuentra y no se indica valor, lanza la excepción KeyError.

- **popitem():** Devuelve un par (clave, valor) aleatorio del diccionario. Si el diccionario está vacío, lanza la excepción KeyError.

Listado de métodos de la clase dict

- **setdefault(clave[, valor]):**Si la clave está en el diccionario, devuelve su valor. Si no lo está, inserta la clave con el valor valor y lo devuelve (si no se especifica valor, por defecto es None).

- **update(iterable):**Actualiza el diccionario con los pares clave: valor del iterable.

- **values() :**Devuelve una vista de los valores del diccionario.