Volver

U3.3

Diccionarios en Python

Volver

Rafael Alberti
Instituto de Educación Secundaria

Volver

Indice

Volver

Diccionarios 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.

Volver

Nota sobre Hashable

- 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`).

Volver

Operaciones Principales

- Las principales operaciones que se suelen realizar con diccionarios son
  - almacenar un valor asociado a una clave
  - recuperar un valor a partir de una clave.
- El acceso a un elemento a partir de una clave es una operación realmente rápida, eficaz y que consume pocos recursos si lo comparamos con cómo lo haríamos con otros tipos de datos.

Volver

Características Adicionales

- Es un tipo mutable, es decir, su contenido se puede modificar después de haber sido creado.
- Es un tipo ordenado. Preserva el orden en que se insertan los pares clave: valor.
- Es un tipo eficiente. El acceso a un elemento a partir de una clave es muy rápido.

Volver

Características Adicionales

- Los diccionarios son tipos heterogéneos, es decir, pueden contener elementos de distinto tipo.
- Los diccionarios son tipos de datos flexibles, es decir, no es necesario que todos los elementos sean del mismo tipo.
- Los diccionarios son tipos de datos de alto nivel, es decir, permiten almacenar cualquier tipo de objeto.
- Los diccionarios son tipos de datos seguros, es decir, no se puede acceder a un elemento a partir de una clave que no existe.

Volver

Cómo crear un diccionario

- En Python hay varias formas de crear un diccionario.
- La más simple es encerrar una secuencia de pares clave: valor separados por comas entre llaves:

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

Volver

Cómo crear un diccionario

- Para crear un diccionario vacío, simplemente asigna a una variable el valor {}.
- También se puede usar el constructor de la clase dict() de varias maneras:
  - Sin parámetros (esto creará un diccionario vacío).
  - Con pares clave: valor encerrados entre llaves.
  - Con argumentos con nombre. El nombre del argumento será la clave en el diccionario.

Volver

Cómo crear un diccionario

- También se puede usar el constructor de la clase dict() de varias maneras:
  - Pasando un iterable. En este caso, cada elemento del iterable debe ser también un iterable con solo dos elementos.
    - El primero se toma como clave del diccionario y el segundo como valor.
    - Si la clave aparece varias veces, el valor que prevalece es el último.

Volver

Cómo acceder a los elementos de un diccionario en Python

- 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}
print(d['dos']) # Output: 2
```

Volver

El método get()

- 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}
print(d.get('uno')) # Output: 1
```

Volver

Cómo recorrer un diccionario - For dict Python

- Hay varias formas de recorrer los elementos de un diccionario:
  - recorrer solo las claves,
  - solo los valores
  - 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)
```

Volver

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 []
  - A la derecha el valor que se asocia a dicha clave.

```
d = {'uno': 1, 'dos': 2}
d['tres'] = 3
```

Volver

Añadir elementos mediante el método setdefault()

- También existe el método `setdefault(clave[, valor])`.
- Este método devuelve el valor de la clave si ya existe.
- 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}
print(d.setdefault('uno', 1.0)) # Output: 1
```

Volver

Modificar elementos de un diccionario

- 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'] = 1.0
```

Volver

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.
  - `popitem()`: Elimina el último par clave: valor del diccionario y lo devuelve.
  - `del d[clave]`: Elimina el par clave: valor.
  - `clear()`: Borra todos los pares clave: valor del diccionario.

Volver

Eliminar un elemento de un diccionario en Python

```
d = {'uno': 1, 'dos': 2, 'tres': 3, 'cuatro': 4, 'cinco': 5}
d.pop('uno')  # Elimina un elemento con pop()
d.pop(6)      # Trata de eliminar una clave con pop() que no existe
d.popitem()   # Elimina un elemento con popitem()
del d['tres'] # Elimina un elemento con del
del d['seis'] # Trata de eliminar una clave con del que no existe
d.clear()     # Borra todos los elementos del diccionario
```

Volver

Número de elementos (len) de un diccionario en Python

- Tambien se puede usar la función de Python `len()` para obtener el número de elementos de un diccionario.

```
d = {'uno': 1, 'dos': 2, 'tres': 3}
len(d) # Output: 3
```

Volver

Comprobar si un elemento está en un diccionario en Python

- Se puede usar el operador de pertenencia `in` para comprobar si una clave está contenida en un diccionario.
- Se puede usar el operador de pertenencia `not in` para comprobar si una clave no está contenida en un diccionario.
- Esto resulta útil, por ejemplo, para asegurarnos de que una clave existe antes de intentar eliminarla.

```
d = {'uno': 1, 'dos': 2, 'tres': 3}
'uno' in d          # Devuelve True
1 in d              # Devuelve False
'cuatro' not in d # Devuelve True
```

Volver

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.

Volver

Comparar si dos diccionarios son iguales

*Si se comparan dos diccionarios

- vacíos, el resultado es `True`.
- que contienen los mismos pares clave: valor, pero en distinto orden, el resultado es `True`.
- que contienen los mismos pares clave: valor, pero con distinto número de elementos, el resultado es `False`.

Volver

Comparar si dos diccionarios son iguales

```
d1 = {'uno': 1, 'dos': 2}
d2 = {'dos': 2, 'uno': 1}

d1 == d2          # Devuelve True
d1 == {'uno': 1} # Devuelve False
```

Volver

Diccionarios Anidados

- Un diccionario puede contener un valor de cualquier tipo, entre ellos, otro diccionario.
- 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']  # Devuelve 1
d['d2']['k1']  # Devuelve 3
```

Volver

Obtener una Lista de Claves

- A veces, es necesario tener almacenado en una lista las claves de un diccionario.
- Para ello, simplemente pasa el diccionario como argumento del constructor `list()`.

```
d = {'uno': 1, 'dos': 2, 'tres': 3}
list(d)  # Devuelve ['uno', 'dos', 'tres']
```

Volver

Objetos Vista de un Diccionario

- La clase dict implementa tres métodos muy particulares
  - `keys()`: Devuelve una vista de las claves del diccionario.
  - `values()`: Devuelve una vista de los valores del diccionario.
  - `items()`: Devuelve una vista de pares (clave, valor) del diccionario.

Volver

Objetos Vista de un Diccionario

- Los objeto vista `dict_values` o `dict_keys`, son objetos que permiten iterar a través de las claves y valores contenidos en el diccionario
- Si el diccionario se modifica, dichos objetos se actualizan al instante.
- Se puede pasar como argumento del constructor `list()`

Volver

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, o el valor por defecto si no existe.
- `items()`: Devuelve una vista de los pares clave-valor del diccionario.
- `keys()`: Devuelve una vista de las claves del diccionario.

Volver

Listado de Métodos de la Clase dict

- `pop(clave[, valor])`: Devuelve el valor del elemento y lo elimina del diccionario.
- `popitem()`: Devuelve un par (clave, valor) aleatorio del diccionario.
- `setdefault(clave[, valor])`: Si la clave está en el diccionario, devuelve su valor. Si no, la inserta y devuelve el valor (o None por defecto).
- `update(iterable)`: Actualiza el diccionario con los pares clave-valor del iterable.
- `values()`: Devuelve una vista de los valores del diccionario.

Volver

Depuración

- Conforme trabajes con conjuntos de datos más grandes puede ser complicado depurar imprimiendo y revisando los datos a mano.
- Aquí hay algunas sugerencias para depurar grandes conjuntos de datos:
  - Reducir la entrada
  - Revisar extractos y tipos
  - Escribir auto-verificaciones
  - Imprimir una salida ordenada

Volver

Resumen: Diccionarios en Python

Volver

Diccionarios

- Los diccionarios en Python son estructuras de datos que asocian claves con valores.
- Las claves deben ser inmutables y hashables.
- Las principales operaciones incluyen almacenar y recuperar valores por clave.

Volver

Creación y Manipulación

- Crear un diccionario: `{}` o `dict()`.
- Acceder a elementos: `d[clave]` o `d.get(clave, valor_por_defecto)`.
- Recorrer un diccionario con `for` y `keys()`, `values()`, `items()`.
- Añadir, modificar y eliminar elementos en un diccionario.

Volver

Conclusiones

- Los diccionarios son ideales para mapear claves a valores en Python.
- Permite un acceso rápido a los elementos.
- Se pueden crear de varias formas y manipular eficientemente.
- Utiliza objetos vista para claves, valores y pares clave-valor.

Volver

¡Gracias por su atención!