

[Latin: related to dictate]
dictatorial /ˈdɪktəˈtɔːriəl/ *n.*
like a dictator. 2 overbearing
dictatorially *adv.* [Latin: related
TATOR]

diction /ˈdɪkʃ(ə)n/ *n.* manne
ciation in speaking or sing
dictio from *dico dict-* say]

dictionary /ˈdɪkʃənəri/ *n.*
book listing (usu. alphabet
explaining the words of a l
giving corresponding words
language. 2 reference book
the terms of a particul

¿Cómo funcionan los Diccionarios en Python?

PyBirras • Tenerife • 26/jul/2019

Juan Ignacio Rodríguez de León @jileon

Diccionarios

Lo mejor desde el pan de molde

```
numbers = ['cero', 'uno', 'dos', 'tres']  
assert numbers[1] == 'uno'
```

```
numbers = {'cero': 0, 'uno': 1, 'dos': 2, 'tres': 3}  
assert numbers['dos'] == 2  
assert numbers['uno'] + numbers['dos'] == numbers['tres']
```

Con las listas es fácil

- Con las listas (o tuplas) accedemos a los valores por su **índice o posición**
- Esto es fácil de implementar: a partir del índice se consigue la **dirección de memoria** donde está el dato

Con las listas es fácil

- Con las listas (o tuplas) accedemos a los valores por su **índice o posición**
- Esto es fácil de implementar: a partir del índice se consigue la **dirección de memoria** donde está el dato

Con las listas es fácil

- Con las listas (o tuplas) accedemos a los valores por su **índice o posición**
- Esto es fácil de implementar: a partir del índice se consigue la **dirección de memoria** donde está el dato

Con las listas es fácil

Cómo acceder al
tercer elemento
(Índice 2)

$$\underbrace{1024}_{\text{Offset}} + \underbrace{(2)}_{\text{Index}} \times \underbrace{8}_{\text{Width}} = 1040$$

Offset →

64 bits							
1024	1025	1026	1027	1028	1029	1030	1031
C	E	R	0	0			
1032	1033	1034	1035	1036	1037	1038	1039
U	N	0	0				
1040	1041	1042	1043	1044	1045	1046	1047
D	0	S	0				
1048	1049	1050	1051	1052	1053	1054	1055
T	R	E	S	0			

Con las listas es fácil



¿Cómo lo hacen los diccionarios?

- Se puede usar (casi) cualquier cosa como índice
- No hay una función matemática que a partir de la clave pueda indicarnos la posición en memoria del contenido

¿Cómo lo hacen los diccionarios?

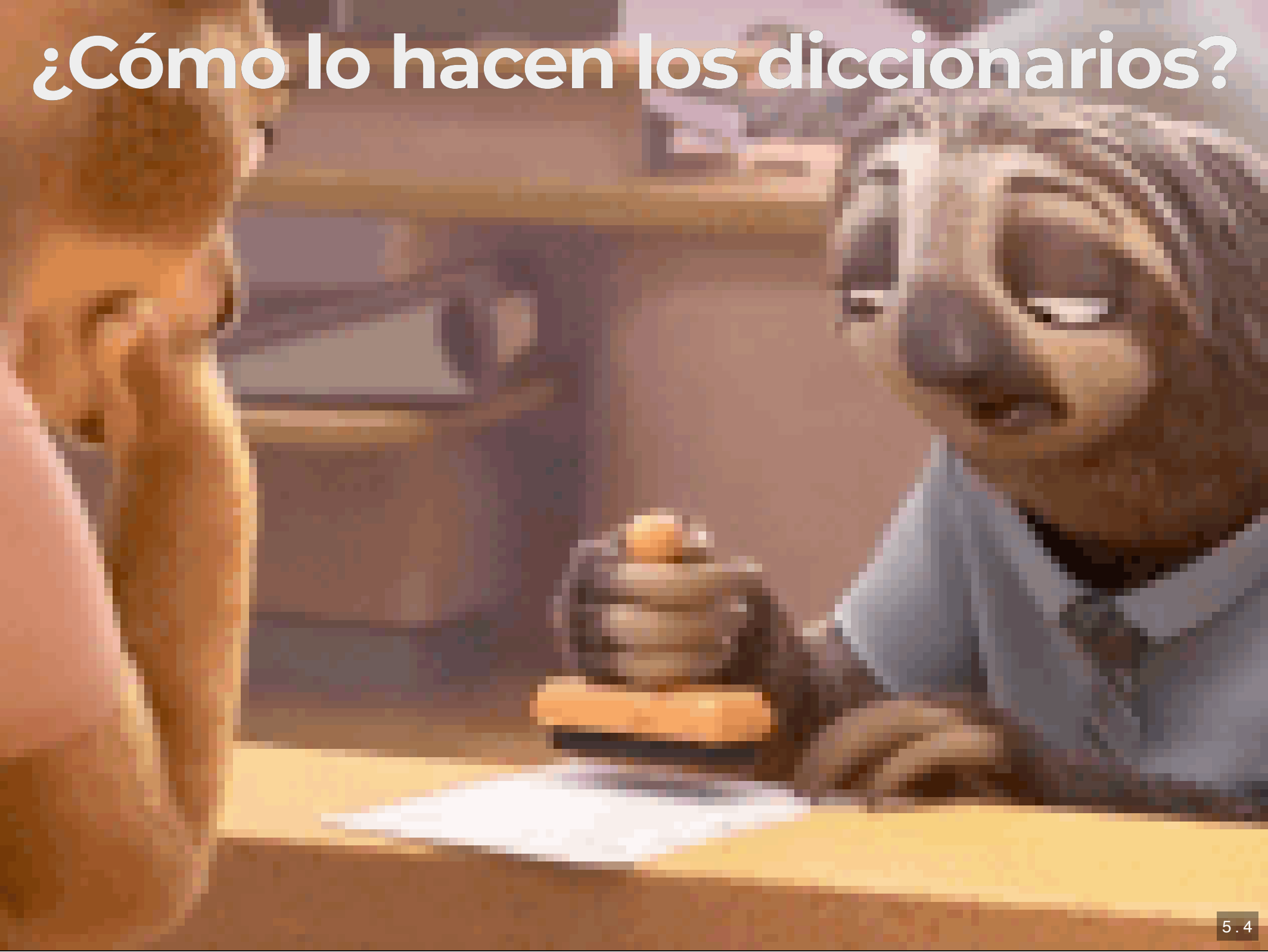
Una aproximación ingenua

- Podríamos guardar una lista de tuplas
- Cada tupla constaría de dos elementos, la clave y el valor
- Para acceder, buscando la tupla cuya clave sea igual a la indicada
- Devolvemos el valor si lo encontramos, o elevamos una excepción `KeyError` si no

¿Cómo lo hacen los diccionarios?

No escala. Según crece el diccionario, más tardará, de media, en localizar un valor

¿Cómo lo hacen los diccionarios?



notación de Landau

El acceso a la lista es independiente del tamaño de la misma, porque solo tiene que hacer una multiplicación y una suma, no importa donde esté ubicado el valor

El acceso a la lista es **$O(1)$** u **Orden constante**

notación de Landau

Nuestro "Diccionario" tiene orden **$O(n)$** , u **Orden lineal** o **de primer orden** porque lo rápido que sea dependerá del número de valores almacenados

$O(1)$ es mucho, mucho mejor que $O(n)$

notación de Landau

Nuestro "Diccionario" tiene orden **$O(n)$** , u **Orden lineal** o **de primer orden** porque lo rápido que sea dependerá del número de valores almacenados

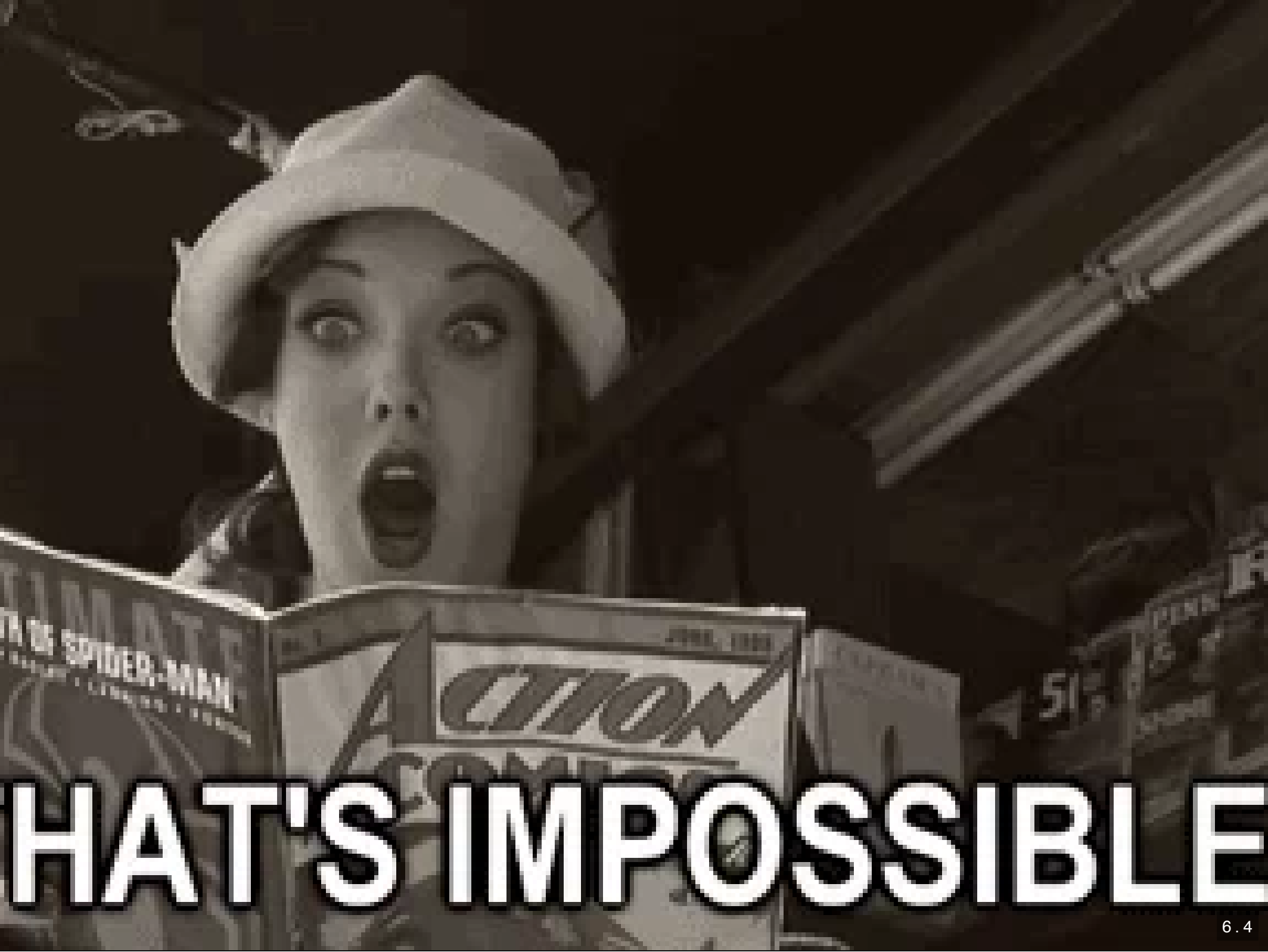
$O(1)$ es mucho, mucho mejor que $O(n)$

El mérito del diccionario

es tener orden $O(1)$, es decir, que devuelva en un **tiempo constante** el resultado independientemente del tamaño del diccionario

El mérito del diccionario

es tener orden $O(1)$, es decir, que devuelva en un **tiempo constante** el resultado independientemente del tamaño del diccionario



WHAT'S IMPOSSIBLE

Funciones Hash

- Transforman un **dato o conjunto de datos** en un **número dentro un rango limitado**
- Si **dos datos son iguales**, producen el **mismo valor de hash**
- Si **dos datos son diferentes**, aun así **podrían producir el mismo valor de hash**. Esto se conoce como **colisiones**
- Ante un **pequeño** cambio en los datos de entrada, se produce un **número muy diferente**

Speaker notes

La última propiedad es deseable, pero no estrictamente necesaria para ser una función hash

Funciones Hash

- Transforman un **dato o conjunto de datos** en un **número dentro un rango limitado**
- Si **dos datos son iguales**, producen el **mismo valor de hash**
- Si **dos datos son diferentes**, aun así **podrían producir el mismo valor de hash**. Esto se conoce como **colisiones**
- Ante un **pequeño** cambio en los datos de entrada, se produce un **número muy diferente**

Speaker notes

La última propiedad es deseable, pero no estrictamente necesaria para ser una función hash

Funciones Hash

- Transforman un **dato o conjunto de datos** en un **número dentro un rango limitado**
- Si **dos datos son iguales**, producen el **mismo valor de hash**
- Si **dos datos son diferentes**, aun así **podrían producir el mismo valor de hash**. Esto se conoce como **colisiones**
- Ante un **pequeño** cambio en los datos de entrada, se produce un **número muy diferente**

Speaker notes

La última propiedad es deseable, pero no estrictamente necesaria para ser una función hash

Funciones Hash

- Transforman un **dato o conjunto de datos** en un **número dentro un rango limitado**
- Si **dos datos son iguales**, producen el **mismo valor de hash**
- Si **dos datos son diferentes**, aun así **podrían producir el mismo valor de hash**. Esto se conoce como **colisiones**
- Ante un **pequeño** cambio en los datos de entrada, se produce un **número muy diferente**

Speaker notes

La última propiedad es deseable, pero no estrictamente necesaria para ser una función hash

Funciones Hash

- Transforman un **dato o conjunto de datos** en un **número dentro un rango limitado**
- Si **dos datos son iguales**, producen el **mismo valor de hash**
- Si **dos datos son diferentes**, aun así **podrían producir el mismo valor de hash**. Esto se conoce como **colisiones**
- Ante un **pequeño** cambio en los datos de entrada, se produce un **número muy diferente**

Speaker notes

La última propiedad es deseable, pero no estrictamente necesaria para ser una función hash

Funciones Hash

- Transforman un **dato o conjunto de datos** en un **número dentro un rango limitado**
- Si **dos datos son iguales**, producen el **mismo valor de hash**
- Si **dos datos son diferentes**, aun así **podrían producir el mismo valor de hash**. Esto se conoce como **colisiones**
- Ante un **pequeño** cambio en los datos de entrada, se produce un **número muy diferente**

Speaker notes

La última propiedad es deseable, pero no estrictamente necesaria para ser una función hash

Funciones Hash

- Transforman un **dato o conjunto de datos** en un **número dentro un rango limitado**
- Si **dos datos son iguales**, producen el **mismo valor de hash**
- Si **dos datos son diferentes**, aun así **podrían producir el mismo valor de hash**. Esto se conoce como **colisiones**
- Ante un **pequeño** cambio en los datos de entrada, se produce un **número muy diferente**

Speaker notes

La última propiedad es deseable, pero no estrictamente necesaria para ser una función hash

Funciones Hash

- Transforman un **dato o conjunto de datos** en un **número dentro un rango limitado**
- Si **dos datos son iguales**, producen el **mismo valor de hash**
- Si **dos datos son diferentes**, aun así **podrían producir el mismo valor de hash**. Esto se conoce como **colisiones**
- Ante un **pequeño** cambio en los datos de entrada, se produce un **número muy diferente**

Speaker notes

La última propiedad es deseable, pero no estrictamente necesaria para ser una función hash

Funciones Hash

- Transforman un **dato o conjunto de datos** en un **número dentro un rango limitado**
- Si **dos datos son iguales**, producen el **mismo valor de hash**
- Si **dos datos son diferentes**, aun así **podrían producir el mismo valor de hash**. Esto se conoce como **colisiones**
- Ante un **pequeño** cambio en los datos de entrada, se produce un **número muy diferente**

Speaker notes

La última propiedad es deseable, pero no estrictamente necesaria para ser una función hash

Matemáticamente

$$\forall a, b | a = b \Rightarrow \text{hash}(a) = \text{hash}(b)$$

Pero Lo contrario no tiene que ser cierto, si dos valores tienen el mismo valor de hash, no implica que sean iguales

$$\forall a, b | \text{hash}(a) = \text{hash}(b) \nRightarrow a = b$$

Diferentes funciones Hash

- Existen muchas funciones hash
- Una de las más conocidas son la familia de funciones SHA-A (Secure Hash Algorithm)
- Existen 4 variedades: SHA-224, SHA-256, SHA-384 y SHA-512, que producen resultados de 224, 256, 384 y 512 bits respectivamente
- Otras funciones hash conocidas son MD5, BLAKE, Tiger, Whirlpool...

Qué función hash usa Python?

- Python utiliza distintas funciones hash
- Dependiendo de varios factores, entre ellos, el tipo de dato
- Para los enteros, por ejemplo, es simplemente el mismo número
- Se puede controlar el valor de hash para las instancias de nuestras clases, definiendo un método `__hash__`
- Los detalles están explicadas en el [PEP-0456 Secure and interchangeable hash algorithm](#)

Speaker notes

Se puede abrir la consola y hacer varias pruebas

Como curiosidad, el valor `-1` está prohibido (Se usa como valor especial de error en C), así que `hash(1) == 1` pero `hash(-1) == 2`

"hola"

3488318975358358463

3.1416

326507370104658947

4135423

4135423

"En un lugar
de la Mancha

han de caer del
todo sin duda
alguna. Vale."

4132349824358270397

-266660762471380438



Funciones Hash

Al crear un diccionario

- Internamente se crea un array en C de 8 elementos
- Cada entrada en la tabla almacena:
 - Un indicador de **En uso/Libre**
 - El **hash** de la clave
 - Un puntero hacia el valor de la **clave**
 - Un puntero hacia el **valor** almacenado

Speaker notes

Es un array en C. No puede crecer, y todos los elementos del mismo deben ser del mismo tipo, en este caso, estructuras con los cuatro valores anteriores.

Al crear un diccionario

- Internamente se crea un array en C de 8 elementos
- Cada entrada en la tabla almacena:
 - Un indicador de **En uso/Libre**
 - El **hash** de la clave
 - Un puntero hacia el valor de la **clave**
 - Un puntero hacia el **valor** almacenado

Speaker notes

Es un array en C. No puede crecer, y todos los elementos del mismo deben ser del mismo tipo, en este caso, estructuras con los cuatro valores anteriores.

Al crear un diccionario

- Internamente se crea un array en C de 8 elementos
- Cada entrada en la tabla almacena:
 - Un indicador de **En uso/Libre**
 - El **hash** de la clave
 - Un puntero hacia el valor de la **clave**
 - Un puntero hacia el **valor** almacenado

Speaker notes

Es un array en C. No puede crecer, y todos los elementos del mismo deben ser del mismo tipo, en este caso, estructuras con los cuatro valores anteriores.

Al crear un diccionario

- Internamente se crea un array en C de 8 elementos
- Cada entrada en la tabla almacena:
 - Un indicador de **En uso/Libre**
 - El **hash** de la clave
 - Un puntero hacia el valor de la **clave**
 - Un puntero hacia el **valor** almacenado

Speaker notes

Es un array en C. No puede crecer, y todos los elementos del mismo deben ser del mismo tipo, en este caso, estructuras con los cuatro valores anteriores.

Al crear un diccionario

- Internamente se crea un array en C de 8 elementos
- Cada entrada en la tabla almacena:
 - Un indicador de **En uso/Libre**
 - El **hash** de la clave
 - Un puntero hacia el valor de la **clave**
 - Un puntero hacia el **valor** almacenado

Speaker notes

Es un array en C. No puede crecer, y todos los elementos del mismo deben ser del mismo tipo, en este caso, estructuras con los cuatro valores anteriores.

Al crear un diccionario, además

- Se almacena el **tamaño actual** de la tabla (inicialmente $8 = 2^3$)
- Y tambien se lleva la cuenta de **cuantos slots hay usados**

Speaker notes

El tamaño de la tabla es un múltiplo de 2. Esto es importante y se explicara más adelante.

Se usa la proporcion entre los slots usados y el tamaño para determinar si el diccionario necesita ser ampliado.

Al crear un diccionario, además

- Se almacena el **tamaño actual** de la tabla (inicialmente $8 = 2^3$)
- Y tambien se lleva la cuenta de **cuantos slots hay usados**

Speaker notes

El tamaño de la tabla es un múltiplo de 2. Esto es importante y se explicara más adelante.

Se usa la proporción entre los slots usados y el tamaño para determinar si el diccionario necesita ser ampliado.









Al crear un diccionario, además

- Se almacena el **tamaño actual** de la tabla (inicialmente $8 = 2^3$)
- Y tambien se lleva la cuenta de **cuantos slots hay usados**

Speaker notes

El tamaño de la tabla es un múltiplo de 2. Esto es importante y se explicara más adelante.

Se usa la proporcion entre los slots usados y el tamaño para determinar si el diccionario necesita ser ampliado.

	Libre	Hash	clave	valor
0 = 000				
1 = 001				
2 = 010				
3 = 011				
4 = 100				
5 = 101				
6 = 110				
7 = 111				

Tamaño: 8

En uso: 0

Diccionario
Vacio

Speaker notes

Los números a la izquierda son las posiciones en el array. Se añade el valor en binario porque se usara en la explicación que sigue.

Vamos a insertar un valor

Supongamos que ejecutamos:

```
d = {}  
d['tres'] = 3
```

Vamos a insertar un valor

Calculamos el hash de "tres"

Speaker notes

Pregunta al público por qué solo tres bits

Pista: $8 = 2^3$

Vamos a insertar un valor

Calculamos el hash de "tres"

2524995206407244382

Speaker notes

Pregunta al público por qué solo tres bits

Pista: $8 = 2^3$

Vamos a insertar un valor

Calculamos el hash de "tres"

2524995206407244382

Cuyo valor en binario es:

Speaker notes

Pregunta al público por qué solo tres bits

Pista: $8 = 2^3$

Vamos a insertar un valor

Calculamos el hash de "tres"

2524995206407244382

Cuyo valor en binario es:

1000110000101010010101110000110010000101110111100101001011110

Speaker notes

Pregunta al público por qué solo tres bits

Pista: $8 = 2^3$

Vamos a insertar un valor

Calculamos el hash de "tres"

2524995206407244382

Cuyo valor en binario es:

1000110000101010010101110000110010000101110111100101001011110

Como la tabla es de tamaño 8, solo interesan los tres últimos bits:

Speaker notes

Pregunta al público por qué solo tres bits

Pista: $8 = 2^3$

Vamos a insertar un valor

Calculamos el hash de "tres"

2524995206407244382

Cuyo valor en binario es:

1000110000101010010101110000110010000101110111100101001011110

Como la tabla es de tamaño 8, solo interesan los tres últimos bits:

110 = 6

Speaker notes

Pregunta al público por qué solo tres bits









Pista: $8 = 2^3$

Insertamos "tres"

Vamos al array, previamente vacío, y añadimos los valores correspondientes en la entrada 6

Ahora tenemos un slot en uso

Insertamos "tres"

	Libre	Hash	clave	valor
0 = 000				
1 = 001				
2 = 010				
3 = 011				
4 = 100				
5 = 101				
6 = 110		2524995206407244382	"tres"	3
7 = 111				

Tamaño: 8

En uso: 1

Diccionario
Una entrada

Vamos a recuperar el valor

Supongamos que ejecutamos:

```
x = d['tres']
```









Vamos a recuperar el valor

- Calculamos el hash de la clave
- Nos quedamos con los tres bits menos significativos
- Vamos a la entrada indicada por esos tres bits (En este caso, 6)
- Analizamos la entrada

Vamos a recuperar el valor

- Si no esta ocupada, elevamos la excepcion `KeyError`
- Si está, comparamos la clave indicada con la almacenada:
 - si son iguales, hemos encontrado la entrada. Devolvemos el valor de la tabla
 - Si el valor es diferente, es una colisión. Veremos como se trata más adelante

Vamos a recuperar el valor

	Libre	Hash	clave	valor
0 = 000				
1 = 001				
2 = 010				
3 = 011				
4 = 100				
5 = 101				
6 = 110		2524995206407244382	"tres"	3
7 = 111				

Tamaño: 8
En uso: 1

Diccionario
Una entrada

Insertamos "uno" => 1

```
d[ 'uno' ] = 1
```

Insertamos "uno" => 1

Calculamos el hash de "uno"

Insertamos "uno" => 1

Calculamos el hash de "uno"

-1091141910288860634

Insertamos "uno" => 1

Calculamos el hash de "uno"

-1091141910288860634

En binario es:

Insertamos "uno" => 1

Calculamos el hash de "uno"

-1091141910288860634

En binario es:

111100100100100000111100100101001010010011110000010111011010

Insertamos "uno" => 1

Calculamos el hash de "uno"

-1091141910288860634

En binario es:

111100100100100000111100100101001010010011110000010111011010

Los tres últimos bits:

Insertamos "uno" => 1

Calculamos el hash de "uno"

-1091141910288860634

En binario es:

111100100100100000111100100101001010010011110000010111011010









Los tres últimos bits:

010 = 2

Insertamos "uno" => 1

- Vamos al array. La entrada 2 esta disponible
- !Qué suerte!
- Añadimos los valores correspondientes en la entrada 2
- Ahora tenemos **dos** slots en uso

Insertamos "uno" => 1

	Libre	Hash	clave	valor
0 = 000				
1 = 001				
2 = 010		-1091141910288860634	"uno"	1
3 = 011				
4 = 100				
5 = 101				
6 = 110		2524995206407244382	"tres"	3
7 = 111				

Tamaño: 8

En uso: 2

Diccionario
Una entrada

Colisiones

- Más tarde o más temprano, habrá una **colisión**
- Es decir dos valores de **claves diferentes** nos darán los mismos tres bits finales
- En nuestro caso es con la clave "**cero**"

Colisiones

- Más tarde o más temprano, habrá una **colisión**
- Es decir dos valores de **claves diferentes** nos darán los mismos tres bits finales
- En nuestro caso es con la clave "**cero**"

Colisiones

- Más tarde o más temprano, habrá una **colisión**
- Es decir dos valores de **claves diferentes** nos darán los mismos tres bits finales
- En nuestro caso es con la clave "**cero**"

Colisiones

- Más tarde o más temprano, habrá una **colisión**
- Es decir dos valores de **claves diferentes** nos darán los mismos tres bits finales
- En nuestro caso es con la clave **"cero"**

Colisiones

Calculamos el hash de "cero"

-5141992977061496694

En binario es:

100011101011100000001110101111010000011010011011100011101110110

Los tres últimos bits:

110 = 6 ⚠

Colisiones

- La entrada 6 ya está ocupada
- Pero quedan todavía 6 slots libres!
- Para resolver esto, python genera una secuencia de valores a partir del 6









Colisiones


- Esta lista siempre es igual para cada valor inicial
- Por ejemplo, podemos pensar que a partir del 6, Python genera [7,0,1,2,3,4,5] (No es el caso, usa un sistema más elaborado)
- Siguiendo esa lista, se busca la primera celda que este libre. En este caso, la 7
- Su guardan los datos en esa celda, y se marcan como ocupada

Speaker notes

```
1 j = (5*j) + 1 + perturb; 2 perturb  $\gg$  PERTURB_SHIFT; 3 use j % 2**i as the next table index;
```

Colisiones

	Libre	Hash	clave	valor
0 = 000				
1 = 001				
2 = 010		-1091141910288860634	"uno"	1
3 = 011				
4 = 100				
5 = 101				
6 = 110		2524995206407244382	"tres"	3
7 = 111		-5141992977061496694	"cero"	0



Tamaño: 8

En uso: 3

Diccionario
Tres entradas

Ampliación de la tabla

- El límite de densidad es $\frac{2}{3}$ del tamaño
- Para el tamaño de 8, tenemos:

$$5 < \frac{2 \times 8}{3} < 6$$

- Así que tenemos que ampliar la tabla al añadir la sexta entrada

Speaker notes

El valor de $\frac{2}{3}$ es empírico, no se trata de ningún valor universal

Ampliación de la tabla

- La ampliación se hace multiplicando por cuatro el numero de slots usados
- Luego se busca el siguiente múltiplo de 2 que puede acomodar esa cantidad
- En nuestro caso, $6 \times 4 = 24$ así que se amplía a 32 (2^5)

Speaker notes

Si el número de entradas en el diccionario crece mucho (más de 50000), se multiplica por dos en vez de por cuatro

Ampliación de la tabla

- Internamente, Python solicita una nueva tabla de 32 entradas
- para cada entrada en la tabla original, calcula la nueva posición...
- ... usando el valor del hash calculado, pero ahora con los 5 últimos bits, en vez de tres
- Las entradas son copiadas a la nueva tabla, en las nuevas posiciones

Ampliación de la tabla

- Por ejemplo, los últimos bits de hash de "tres" eran
... 01001011110
- En una tabla de 8 entradas, ocupaba la posición 6
(110)
- En la nueva tabla de 32 entradas, ocupa la posición
30 (11110)

Speaker notes

Esto resuelve muchas de las colisiones que ocurrían antes, que ahora ya pueden no serlo.

Por ejemplo, la colisión entre "cero" y "tres" se producía porque usábamos solo tres bits. Usando los cinco últimos, "cero" acaba en la posición 22 (10110) mientras que "tres", como hemos visto, acaba en la posición 30 (11110)

Cosas que hemos aprendido

- Qué es una función hash
- Cómo funciona una tabla hash
- Por qué las claves de un diccionario tiene que ser inmutables
- Por qué es obligatorio que si dos variables a y b , son iguales, entonces sus valores de hash también deben ser iguales

Cosas que no hemos visto

- Como y cuando la tabla decrece
- Como se borran entradas en el diccionario (Pista: se usa el campo de En uso)
- La secuencia usada en caso de colisión
- Algunos detalles se han simplificado (Pero no muchos)

GRACIAS POR SU ATENCIÓN

Espero no haberles aburrido demasiado

¿Preguntas?