Clase 4. Diccionarios y Sets

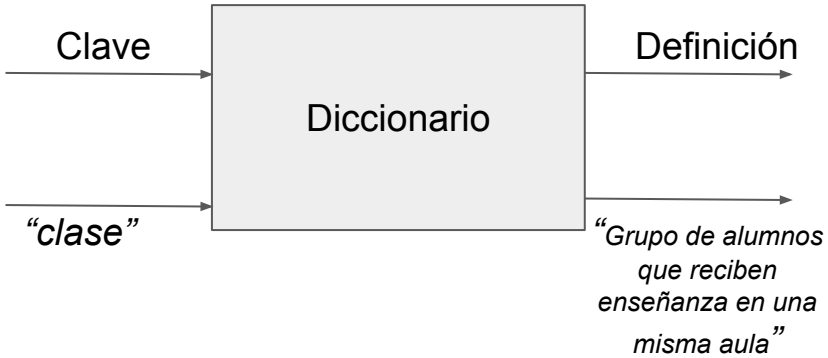
¿Que es un

diccionario?

Clave	**Valor (definición)**
Diccionario	Repertorio en forma de libro o en soporte electrónico en el que se recogen, según un orden determinado, las palabras o expresiones de una o más lenguas, o de una materia concreta, acompañadas de su definición, equivalencia o explicación.
Clase	Grupo de alumnos que reciben enseñanza en una misma aula
Cebra	Animal solípedo del África austral, parecido al asno, de pelo blanco amarillento, con listas transversalespardas o negras. Hay varias especies, y alguna del tamaño del caballo.
Clave1	Cualquier tipo de dato

¿Que es un diccionario?

Un diccionario sirve para guardar pares clave-valor.
Por ejemplo si fuese un diccionario de palabras, las **claves** serían las palabras y las definiciones serían los **valores**.



¿Que es un diccionario?

Las claves y valores pueden ser de cualquier tipo.

Ejemplo: Cantidad de goles en la selección

Clave	**Valor**
Messi	73
Aguero	42
Batistuta	54

Ejemplo: Numeros a letras

Clave	**Valor**
10	diez
11	once
3	tres

Aspectos importantes de un diccionario

- Las claves son ***únicas***.
- Los valores pueden repetirse. Por ejemplo dos palabras que tengan la misma definición
- Los pares clave-valor **no** tienen orden. No existe (como sí existe para vectores por ejemplo) un orden. No hay un “primer” par y un “último”. Por lo que al recorrerlo no hay garantía de que siempre nos den el mismo orden

Aspectos importantes de un diccionario

- El diccionario está **indexado** por las claves
- Las claves **no** pueden cambiar, si pueden cambiar los valores.

DICCIONARIOS

Creación

Se puede crear con pares iniciales o vacío

Python

```
>>> dic_vacio = {}
>>> dic_vacio
{}
```

Python

```
>>> dic_con_valores = {“clave1”: 1,
“clave2”: 4, “clave3”: 5}
>>> dic_con_valores
{“clave1”: 1, “clave2”: 4, “clave3”: 5}
```

DICCIONARIOS

Agregar pares

Python

```
>>> dic = {}
>>> dic[“clave”] = 2
>>> dic
{“clave”: 2}
>>> dic[90] = “algo”
>>> dic
{“clave: 2, 90: “algo”}

# Se pueden modificar valores
>>> dic[“clave”] = “valor”
>>> dic
{“clave: “valor”, 90: “algo”}
```

DICCIONARIOS

Obtener valores

Python

```
>>> dic = {“clave1”: 4, 10: 78}
>>> dic[“clave1”]
4
>>> dic[10]
78

# Es un error consultar por claves que no existe!
>>>dic[“está clave no existe”]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 8

# Se puede consultar si una clave es parte del diccionario
# Sintaxis: clave in dic
>>> “clave1” in dic
True
>>> “Girasol” in dic
False
```

DICCIONARIOS

Obtener valores

Python

```
>>> dic = {“clave1”: 4, 10: 78}

# Tambien se puede usar la funcion get
para obtener un valor
# Sintaxis: diccionatio.get(clave,
valor_default)
>>> dic.get(10)
78
>>> dic.get(“inventado”)

>>> dic.get(“inventado”, 5)
5
```

DICCIONARIOS

Eliminar valores

Python

```
>>> dic = {“clave1”: 4, 10: 78}

# Se usa la función pop
# Sintaxis: dic.pop(clave_a_eliminar) =>
Retona el valor de la clave eliminada.

>>> dic
{“clave1”: 4, 10: 78}
>>> dic.pop(“clave1”)
4
>>> dic
{10: 78}
```

DICCIONARIOS

Cómo recorrerlos

Se pueden utilizar estas 3 funciones:

- keys
- values
- items

U otras que se les ocurran!

Python

```
>>> dic = {"clave1": 1, "clave7": 7}

# .keys retorna un <iterable> de las
claves
>>> for clave in dic.keys():
      print(clave)
clave1
clave7

# .values retorna un <iterable> de los
valores
>>> for valor in dic.values():
      print(valor)
1
7

# .items retorna un <iterable> de los
pares (clave, valor)
>>> for clave, valor in dic.items():
      print(clave, valor)
clave1 1
clave7 7
```

Por qué los utilizamos?

Por la complejidad algorítmica de sus operaciones!

Si todas las operaciones se realizan en tiempo constante… por qué utilizar otras estructuras y no usar siempre diccionarios?

- Agregar un par: O(1)
- Eliminar un par: O(1)
- Obtener un valor a partir de su clave: ***O(1)***

Dónde está el secreto?

De una forma simple, se puede decir que internamente tienen un vector y una **función de hashing.**

La función de hashing lo que hace es convertir la clave en una posición del vector. Por ejemplo:
f(“clave”) = 45.

Así sabe que el valor de “clave” está en la posición 45

Es importante que la función de hashing se ejecute en O(1) (tiempo constante).

Entonces escribir/leer significa:

- obtener la posición del vector usando la función de hashing
- Leer/Guardar el valor en esta posición.

Y así quedan las operaciones en O(1).

Qué problemas le ven a este funcionamiento?

Dónde está el secreto?

Los problemas:

- Hay que tener una array lo suficientemente grande (infinito quizás) por que no sabemos qué claves van a usar.
- Necesitamos una función de hashing **perfecta**. Esto es, que nunca asigne la misma posición a dos claves distintas. Por ejemplo: f(1) = 10 y f(7) = 10. En ese caso se dice que hay ***colisión***.

Independientemente de estos problemas, si la función de hashing es buena (tiene pocas colisiones) con un vector suficientemente grande, podemos asumir en general las operaciones son en O(1) (tiempo constante, no depende de la cantidad de datos).

DICCIONARIOS

Aplicaciones útiles: Mapas de funciones

```
# Sin diccionarios
if (entrada == “sumar”):
  return sumar(a, b)
elif (entrada == “restar”):
  return restar(a, b)
elif (entrada == “dividir”):
  return dividir(a, b)

# El tiempo de ejecución termina siendo
O(n) siendo n la cantidad de operaciones
```

Python

```
# Con diccionarios
operaciones = {
  “sumar”: sumar,
  “restar”: restar,
  “dividir”: dividir
}

return operaciones[entrada](a, b)

# El tiempo de ejecución queda en O(1)
```

Aplicaciones útiles: Acceso a datos

```
# Con diccionarios
personas = {
  “fede”: {
    “color”: “amarillo”
  },
  “jose”: {
    “color”: “azul”
  },
  ...
}

# Para buscar el color favorito de una
#persona
print(personas[“fede”][“color”]
#Buscar la persona es O(1)!
```

Python

```
# Sin diccionarios
NOMBRE = 0
COLOR_FAVORIO = 1
personas = [
  {“fede”, “amarillo”},
  {“jose”, “azul”},
  {“samuel”, “gris”},
  {“nicolas”, “rojo”}
]

# Para buscar el color favorito de una
persona
 for persona in personas:
   if (persona[NOMBRE] == “samuel”):
    print(persona[COLOR_FAVORITO])

#Buscar la persona es O(n) siendo n la
#cantidad de personas
```

Ejercicio

Tiempo ⏱️

- Escribir una función que reciba una lista de tuplas y retorne un diccionario en donde las claves sean los primeros elementos de las tuplas y los valores los segundos.

Tiempo ⏱️

Ejercicio

- Escribir una función que reciba una frase y devuelva un diccionario con la cantidad de apariciones de cada letra.

```
# Por ejemplo:
Entrada: “Hoy es Lunes”
Respuesta:
{
 ‘h’: 1,
 ‘o’: 1,
 ‘y’: 1,
 ‘e’: 2
  ...
}

# Aclaración: Los espacios no son
caracteres. Además, “E” == “e”.
```

Ejercicio de entrevista de trabajo

Tiempo ⏱️

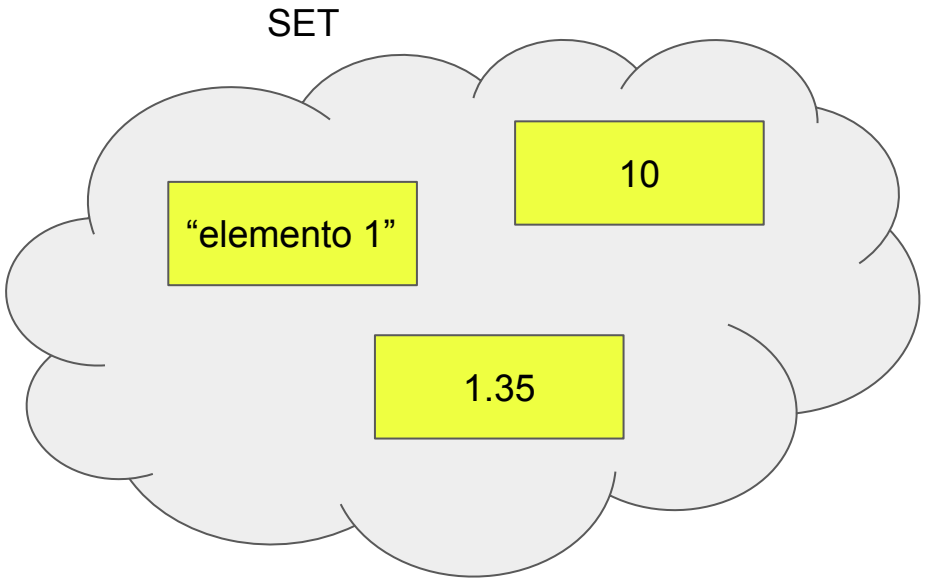
- Escribir una función que reciba dos frases y retorne si con las letras de la primera se puede formar la segunda (usando todas las letras)

```
# Por ejemplo:
Entrada: “Irónicamente” y
“renacimiento”.
Salida: True

Entrada: “Sol” y “palabra”.
Salida: False
```

¿Que es un set?

Un set es una **conjunto** de elementos sin ningun orden especifico ni repeticiones.



Creación

Ojo con la palabra “set” es la función para crear set! No se puede usar como nombre de variable

Python

```
>>> s = {'hola', 12, 3.8}
>>> s
{3.8, 12, 'hola'}
```

Python

```
# No es tan simple crearlo vacio
>>> s = {}
>>> type(s)
<class 'dict'>
# Si se crean con {} se crea un
# diccionario!

# Para crearlo vacio (o no) se puede
# usar la clase set
>>> s = set()
>>> type(s)
<class 'set'>
```

Agregar valores

Dos posibles funciones:

- add: agrega un elemento
- update: agrega un conjunto de elementos

Ambas funciones trabajan en O(1)

Python

```
>>> s = set()
>>> s
set()
>>> s.add(1)
>>> s
{1}

>>> s.update([1, 2, 3])
>>> s
{1, 2, 3}

# NO agrego dos veces el 1, por qué?
```

Eliminar valores

Dos posibles funciones:

- discard: elimina el elemento, si no está presente no hace nada.
- remove: elimina el elemento, si no está presente lanza una excepción.

Ambas funciones trabajan en O(1)

Python

```
>>> s = {1, 2, 3, 4, 5}
>>> s.discard(1)
>>> s
{2, 3, 4, 5}
>>> s.discard(1)
>>> s
{2, 3, 4, 5}

>>> s.remove(2)
>>> s
{3, 4, 5}
>>> s.remove(2)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 2
```

Recorrerlos

- Preguntar si un elemento es parte del set es O(1)

Python

```
>>> s = {1, 2, 3, 4, 5}
>>> 1 in s
True
>>> 90 in s
False
>>> for i in s:
      print(i)
1
2
3
4
5
```

Tiempo ⏱️

Ejercicio

Escribir una función que a partir de un texto y un diccionario de palabras prohibidas, reemplace todas las palabras prohibidas del texto por un “****”.

```
# Por ejemplo:
Entrada: “Hoy es Lunes”, {“es”, “azul”}
Salida: “Hoy **** Lunes”
```