REPASO 2. MÉTODOS PRINCIPALES DE CADENAS

GOBIERNO DE ESPAÑA
MINISTERIO DE EDUCACIÓN Y FORMACIÓN PROFESIONAL
Región de Murcia
Sef Servicio Regional DE EMPLEO y FORMACIÓN
AVANZA

Índice:

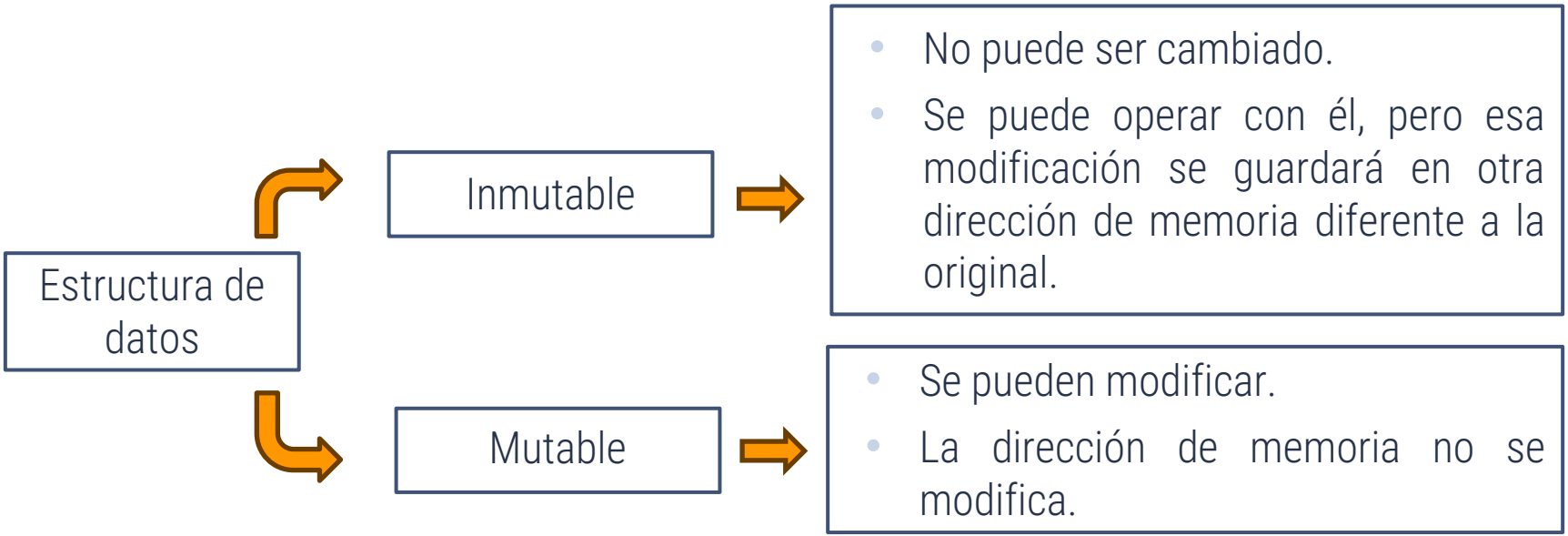
1

Introducción

1. Introducción

- Cuando necesitamos escribir un programa, raramente es suficiente con los tipos de datos básicos para almacenar y manejar la información.
- Por ejemplo necesitamos almacenar un conjunto de temperaturas, esta tarea se volverá muy tediosa usando solo tipos de datos enteros almacenados en diferentes variables.
- Para este tipo de propósitos nace el concento de:
  - **Estructura de datos:** Conjunto de datos compuesto que engloba conjuntos de datos más básicos, a los cuales se puede acceder de manera independiente.

1. Introducción



1. Introducción

Estructura	Características	Principales Métodos
Cadenas	• Ordenadas • Inmutables	.lower(), .upper(), count(), .strip(), .capitalize(), .replace(), .split() y .find()
Listas	• Ordenadas • Mutables	.append(), .pop(), .count(), .sort(), .remove(), .extend(), .insert()
Tuplas	• Ordenadas • Inmutables	.index(), .count()
Diccionarios	• No ordenadas • Mutables	.values, .keys(), .ítems(), .clear(), .get(), .pop() y .update()
Conjuntos	• No ordenadas • Inmutables • Los elementos no se pueden repetir	.add(), .update(), .clear() y .remove()

2

Estructuras de datos

2. Estructuras de datos

- Concepto:
  - Id(): Función para conocer la posición en memoria de un objeto. Al recibir el objeto como argumento devuelve un numero entero que indica la posición de memoria donde se encuentra almacenado.

2.1. Cadenas

- Son un tipo de dato inmutable que contiene una secuencia de caracteres almacenados en la memoria.

```
#Estas 3 variables son cadenas
nombre = "Alejandro"
apellido1 = "Perez"
apellido2 = "Alvarez"
```

2.1. Cadenas

- **Repetir:** Con el operador * se puede repetir la cadena.

```
saludo ="Buenas tardes "
print(saludo * 4)
```

- **Slicing o rebanado:** Obtener una subcadena de la original.

  [comienzo:fin:salto en la secuencia]

  - El primer carácter de una cadena empieza en la posición 0.
  - El final de la cadena lo marcamos como -1.
  - cadena[: :-1] -> Cadena a la inversa

2.1. Cadenas

Ejemplo:

```
texto = "Bienvenidos al curso de Python"
texto[:] #Parametros por defecto
"Bienvenidos al curso de Python" #Se muestra la cadena entera

texto[::] #Tambien mostrará la cadena completa
"Bienvenidos al curso de Python"

texto[0] #Primer caracter
"B"

texto[-1] #Ultimo caracter
"n"

print(texto[:6]) #Desde la posicion 0 a la 5
"Bienve"

print(texto[1:8]) #Desde la posicion 1 a la 7
"ienveni"
```

2.1. Cadenas

```
texto[3:1] #Daria error porque no se puede ir de la posicion 3 a la 1

texto[-2::3] #ten en cuenta que el ultimo caracter de la cadena es -1
"o"

"""-10: Indica que empezaremos a contar desde el décimo carácter contando desde el final de la cadena.
-2: Indica que terminaremos justo antes del segundo carácter contando desde el final.
:2: Este número indica el "paso" o incremento que se dará al recorrer la cadena. En este caso, se tomará un carácter sí y otro no"""
texto[-10:-2:2]
"ePt"
```

2.1. Cadenas

- Errores:
  - Las cadenas son inmutables, no se puede asignar por indexación un nuevo carácter a esta.
  - No se puede introducir por parámetro una posición fuera del índice de la cadena.

2.1. Cadenas

- **Literales de cadenas o secuencias de escape de caracteres:** Introducir operaciones especiales dentro de las cadenas.

Literales	Descripción
\’	Introducir comillas simples
\”	Introducir comillas dobles
\n	Introducir un salto de línea
\t	Introducir una tabulación horizontal
\\	Barra invertida

2.1. Cadenas

```
texto1= "Lista de la compra: \n *Leche \n *Azucar \n *Huevos \n *Harina"
print(texto1)

texto2="Esto es \t una tabulación"
print(texto2)
```

2.1. Cadenas

Cadena cruda o raw: Para manejar caracteres especiales dentro de las cadenas. Se pone r delante de la declaración de la cadena.

```
ruta = r"C:\Proyectos\Carpeta\Unidad2\Python"
print(ruta)
```

2.1. Cadenas

- **Concatenar:** Cuando queremos concatenar un texto y variables se pueden hacer de diferentes maneras:
  - **Función str():** Convierte un tipo de dato diferente en cadena de caracteres.

```
texto = "El numero ganador del sorteo es:"
numero = 345982
print(texto+numero) #Esto producirá un error, porque no son del mismo tipo

print(texto+str(numero))
```

2.1. Cadenas

- Unir 2 o mas cadenas con el operador **+**.

```
2   #Estas 3 variables son cadenas
3   nombre = "Alejandro " #Dejamos un espacio al final
4   apellido1 = "Perez"
5   apellido2 = "Alvarez"
6
7   nombreCompleto = nombre + apellido1 + apellido2
8
9   print(nombreCompleto)
10  #Al utilizar concatenar las cadenas se unen tal se escriben
11  #Al no haber espacios los apellidos se concatenan juntos
12
```

```
PROBLEMAS   SALIDA   CONSOLA DE DEPURACIÓN   TERMINAL   PUERTOS

PS C:\Users\Loreto\Desktop\Avanza\Python\1.Programación Python - IFCD32\U
python.exe "c:/Users/Loreto/Desktop/Avanza/Python/1.Programación Python -
Alejandro PerezAlvarez
```

2.1. Cadenas

- Formateo de cadenas:
  - **f'**: delante de la cadena e introduciendo las variables entre llaves.
  - (Comilla simple o doble)

```
numero = 345982
print(f'El numero ganador del sorteo es: {numero}')
```

  - **.format**: Poniendo llaves donde debería ir la variable, y al final, usar el método con la variable entre paréntesis.

```
numero = 345982
print("El numero ganador del sorteo es: {}".format(numero))

#Tambien se pueden declarar las variables dentro del .format()
print("Mi hijo tiene {edad} años".format(edad=12))
```

2.1. Cadenas

Métodos	Descripción
.lower()	Transforma la cadena en minúsculas.
.upper()	Transforma la cadena en mayúsculas.
.title()	Convierte la primera letra de cada palabra de una cadena a mayúsculas.
.count()	Indica cuantas veces aparece un carácter o una subcadena dentro de la cadena a la cual se le aplica el método.
.strip()	Elimina los espacios sobrantes al principio y al final de la cadena.

2.1. Cadenas

Métodos	Descripción
.capitalize()	Convierte en mayúscula el primer carácter de la cadena
.replace()	Remplaza los caracteres que se le indica por la cadena que se quiera.
.split()	Divide la cadena en el carácter que se le indique y elimina el dicha carácter. Crea una lista con las palabras resultantes.
.find()	Busca el carácter que indiquemos y devuelve la posición en la que aparece dentro de la cadena.

2.1. Cadenas

```
cadena = "HOLA"
print(cadena.lower()) #hola

cadena1 = "hola"
print(cadena1.upper()) #HOLA

texto = "El sofa es rojo"
print(texto.count("o")) #3

texto1 = " Ya estamos aquí "
print(texto1.strip()) #Ya estamos aquí

texto2 = "en un lugar de la mancha"
print(texto2.capitalize()) #En un lugar de la mancha

texto3 = "Tengo una camiseta rosa y unos vaqueros negros"
print(texto3.split("o")) #['Teng', ' una camiseta r', 'sa y un', 's vaquer', 's negr', 's']

print(texto3.replace("a","u")) #Tengo unu cumisetu rosu y unos vuqueros negros
```

2.1. Cadenas

- Ejemplos con find():

```
#Ejemplo 1 - Encontrando la primera ocurrencia de una subcadena
texto = "El perro marrón salta sobre la valla"
indice = texto.find("marrón")
print(indice)  # Output: 9 (el índice donde comienza la palabra "marrón")

#Ejemplo 2 - Buscando una subcadena que no existe
texto = "Hola mundo"
indice = texto.find("Python")
print(indice)  # Output: -1 (la subcadena no se encontró)

#Ejemplo 3 - Especificando un rango de búsqueda
texto = "El perro marrón salta sobre la valla"
indice = texto.find("la", 10)  # Buscar "la" a partir del índice 10
print(indice)  # Output: 28 (encuentra la segunda "la")
```

2.1. Cadenas

```
#Ejemplo 4 - Uso de find() para validar datos
email = "micorreo@ejemplo.com"
arroba = email.find("@")
if arroba != -1:
    print("El correo electrónico es válido.")
else:
    print("El correo electrónico no es válido.")

#Ejemplo 5 - Combinando find() con slicing para extraer subcadenas
texto = "Hola, mundo. Esto es un ejemplo."
indice_coma = texto.find(",")
saludo = texto[:indice_coma]
print(saludo)  # Output: Hola
```

2.1. Cadenas

- **Función len():** Devuelve el número de caracteres que tiene la cadena.

```
texto = "Acabo de llegar a clase"
print(len(texto)) #23
```

- Se puede utilizar el **bucle for** para recorrer una cadena.

```
texto = "El barco es rosa"
for letra in texto:
    print(letra)
```

2.2. Listas

- Son las estructuras son las mas utilizadas y las mas versátiles.
- Son estructuras ordenadas y mutables.
- Dentro de ellas puede haber cualquier tipo de dato básico, otras listas, tuplas, diccionarios o conjuntos.
- Para crear una lista:
  - Se usan **[ ]**
  - Con la función **list()**

2.2. Listas

```
#Se introducen entre corchete los datos porque son varios
datos = list(["Andres Rubio",12,True, 6.22]) #['Andres Rubio', 12, True, 6.22]
print(datos)

print(type(datos)) #<class 'list'>

"""Al ser una cadena que es una estructura iterable,
Python los separa automaticamente en elementos individuales
y crea la lista"""
vocales = list("aeiou")
print(vocales) #['a', 'e', 'i', 'o', 'u']
```

2.2. Listas

```
"""Por el contrario si se declara con numeros enteros
se produce un error, ya que los enteros no son iterables"""
numeros = list(12345)
print(numeros)
```

```
Traceback (most recent call last):
  File "c:\Users\Loreto\Desktop\Avanza\Python\1.Programación Python
    numeros = list(12345)
              ^^^^^^^^^^
TypeError: 'int' object is not iterable
```

2.2. Listas

```
print(compra)
#['huevos', 'tomates', 'leche']

print(compra[2])
#leche

"""crear sublistas más pequeñas de una más grande. Para ello debemos de usar : entre corchetes,
indicando a la izquierda el valor de inicio, y a la izquierda el valor final que no está incluido."""
print(compra[1:2])
#['tomates']

print(id(compra)) #Se ve la posicion del objeto en memoria
#2079295150336

compra[0] = "Espinacas" #Modificamos datos en la posicion 0
print(id(compra)) #Aunque se ha modificado el id de memoria es el mismo ya que es mutable el objeto
#2079295150336

print(compra) #Se ha modificado huevos por espinacas
#['Espinacas', 'tomates', 'leche']

print(len(compra))
#3
```

2.2. Listas

- Las listas al igual que las cadenas se pueden concatenar y repetir:

```
print(compra *3)
#['Espinacas', 'tomates', 'leche', 'Espinacas', 'tomates', 'leche', 'Espinacas', 'tomates', 'leche']

print(datos + compra)
#['Andres Rubio', 12, True, 6.22, 'Espinacas', 'tomates', 'leche']
```

2.2. Listas

Métodos	Descripción
.append()	Agrega un elemento al final de la lista
.pop()	Muestra el ultimo elemento de la lista y lo borra.
.count()	Cuenta el número de veces que aparece en la lista el elemento que pasamos como parámetro.
.short()	Ordena los elementos de la lista.
.remove()	Recibe como argumento un elemento y borra su primera aparición de la lista.

2.2. Listas

Métodos	Descripción
.insert()	Inserta un dato en la lista. Tiene 2 parámetros, el primero muestra la posición y el segundo el dato que introducir.
.extend()	Inserta al final de la lista el objeto iterable que pasamos como parámetro.
.reverse()	Invierte el orden de la lista.

2.2. Listas

```
coches = ["Mercedes","Audi","BMW","Seat","Opel"] #Creamos la lista

coches.append("Toyota") #Añadimos Toyota a la lista
print(coches)
#['Mercedes', 'Audi', 'BMW', 'Seat', 'Opel', 'Toyota']

coches.pop() #Se limina el ultimo elemento de la lista (Toyota)
print(coches)
#['Mercedes', 'Audi', 'BMW', 'Seat', 'Opel']

#Sort Ordena por orden alfabetico y de menor a mayor
coches.sort()
print(coches)
#['Audi', 'BMW', 'Mercedes', 'Opel', 'Seat']

numeros = [1,6,23,88,20,14,7,23, 35]
numeros.sort()
print(numeros)
#[1, 6, 7, 14, 20, 23, 23, 35, 88]
```

2.2. Listas

```
numeros.reverse()
print(numeros)
#[88, 35, 23, 23, 20, 14, 7, 6, 1]

numeros.count(23)
print(numeros)
#2

numeros.remove(23)
print(numeros)
[88, 35, 23, 20, 14, 7, 6, 1]

numeros.extend(range(8))
print(numeros)
#[88, 35, 23, 20, 14, 7, 6, 1, 0, 1, 2, 3, 4, 5, 6, 7]

numeros.insert(4,15)
print(numeros)
#[88, 35, 23, 20, 15, 14, 7, 6, 1, 0, 1, 2, 3, 4, 5, 6, 7]
```

2.2. Listas

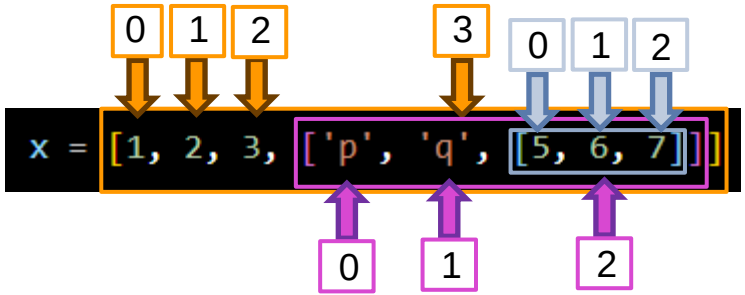
- Como las listas son iterables se pueden recorrer en bucle y mostrar u operar con los elementos:

```
numeros = [2,5,7,8,15]

for i in numeros:
    print(i + 2)
"""4
7
9
10
17"""
```

2.2.1. Listas anidadas

- Una lista dentro de otra. Incluso podemos tener una lista dentro de otra lista y a su vez dentro de otra lista. Para acceder a sus elementos sólo tenemos que usar [ ] tantas veces como niveles de anidado tengamos.




```
print(x[3][0])    #p
print(x[3][2][0]) #5
print(x[3][2][2]) #7
```

2.2.2. Listas anidadas

- Si se quiere iterar en una lista anidada, se debe acceder posicionalmente a ella en un bucle de iteración.

```
for i in x:
    print(i)
"""
1
2
3
['p', 'q', [5, 6, 7]]
"""
```

```
for i in x[3]:
    print(i)
"""
p
q
[5, 6, 7]
"""
```

```
for i in x [3][2]:
    print(i)
"""
5
6
7
"""
```

2.2.3. List comprehension

- Listas de comprensión. Crear una secuencia a partir de otra. Se crea:

  nueva_lista[<acción> for <item> in <iterador> if condición]

```
#Crear una lista de letras con la palabra hola
saludo = [letra for letra in "hola"]
print(saludo) #['h', 'o', 'l', 'a']

#Crear una lista de los cuadrados de los pares con rango comprendido entre 1 y 19
numeros_pares =[i*2 for i in range(0,20) if i%2 ==0]
print(numeros_pares) #[0, 4, 8, 12, 16, 20, 24, 28, 32, 36]
```

2.3. Tuplas

- Las **tuplas** en Python o tuples son muy similares a las listas, pero con dos diferencias. Son inmutables, lo que significa que no pueden ser modificadas una vez declaradas, y en vez de inicializarse con corchetes se hace con **()**. Dependiendo de lo que queramos hacer, las tuplas pueden ser más rápidas.

```
tupla = (1, 2, 3)
print(tupla) #(1, 2, 3)
```

- También pueden declararse sin (), separando por , todos sus elementos.

```
tupla = 1, 2, 3
print(type(tupla)) #<class 'tuple'>
print(tupla)       #(1, 2, 3)
```

2.3. Tuplas

- Como hemos comentado, las tuplas son tipos **inmutables**, lo que significa que una vez asignado su valor, no puede ser modificado. Si se intenta, tendremos un *TypeError*.

```
tupla = (1, 2, 3)
#tupla[0] = 5 # Error! TypeError
```

2.3 Tuplas

- Al igual que las listas, las tuplas también pueden ser anidadas.

```
tupla = 1, 2, ('a', 'b'), 3
print(tupla)       #(1, 2, ('a', 'b'), 3)
print(tupla[2][0]) #a
```

2.3 Tuplas

- Y también es posible convertir una lista en tupla haciendo uso de al función tuple().

```
lista = [1, 2, 3]
tupla = tuple(lista)
print(type(tupla)) #<class 'tuple'>
print(tupla)       #(1, 2, 3)
```

2.3 Tuplas

- Y se puede también asignar el valor de una tupla con n elementos a n variables.

```
l = (1, 2, 3)
x, y, z = l
print(x, y, z) #1 2 3
```

2.3 Tuplas

Métodos	Descripción
.index()	Busca un elemento dentro de la tupla y muestra su posición. En el caso de no encontrarla producirá un error.
.count()	Cuenta el número de veces que aparece en la lista el elemento que pasamos como parámetro.

2.4. Diccionarios

- Un diccionario en Python es una colección de elementos, donde cada uno tiene una llave **key** y un valor **value**. Los diccionarios se pueden crear con paréntesis { } separando con una coma cada par key: value

```
#Tenemos tres keys que son el nombre, la edad y el documento.
d1 = {
  "Nombre": "Sara",
  "Edad": 27,
  "Documento": 1003882
}
print(d1)
#{'Nombre': 'Sara', 'Edad': 27, 'Documento': 1003882}
```

2.4. Diccionarios

- Otra forma equivalente de crear un diccionario en Python es usando **dict()** e introduciendo los pares key: value entre paréntesis.

```
d2 = dict([
    ('Nombre', 'Sara'),
    ('Edad', 27),
    ('Documento', 1003882),
])
print(d2)
#{'Nombre': 'Sara', 'Edad': '27', 'Documento': '1003882'}
```

2.4. Diccionarios

- También es posible usar el constructor dict() para crear un diccionario.

```
d3 = dict(Nombre='Sara',
          Edad=27,
          Documento=1003882)
print(d3)
#{'Nombre': 'Sara', 'Edad': 27, 'Documento': 1003882}
```

2.4. Diccionarios

- Los diccionario en Python tienen las siguientes caracteristicas:

- Son dinámicos, pueden crecer o decrecer, se pueden añadir o eliminar elementos.
- Son indexados, los elementos del diccionario son accesibles a través del key.
- Y son anidados, un diccionario puede contener a otro diccionario en su campo value.

2.4. Diccionarios

- Se puede acceder a sus elementos con [ ] o también con la función get().

```
print(d1['Nombre'])     #Sara
print(d1.get('Nombre')) #Sara
```

- Para modificar un elemento basta con usar [ ] con el nombre del key y asignar el valor que queremos.

```
d1['Nombre'] = "Laura"
print(d1)
#{'Nombre': Laura', 'Edad': 27, 'Documento': 1003882}
```

2.4. Diccionarios

- Si el key al que accedemos no existe, se añade automáticamente.

```
d1['Direccion'] = "Calle 123"
print(d1)
#{'Nombre': 'Laura', 'Edad': 27, 'Documento': 1003882, 'Direccion': 'Calle 123'}
```

2.4. Diccionarios

- Los diccionarios se pueden iterar de manera muy similar a las listas u otras estructuras de datos. Para imprimir los key.

```
# Imprime los key del diccionario
for x in d1:
    print(x)
#Nombre
#Edad
#Documento
#Direccion
```

2.4. Diccionarios

- También se puede imprimir solo el value.

```
# Impre los value del diccionario
for x in d1:
    print(d1[x])
#Laura
#27
#1003882
#Calle 123
```

2.4. Diccionarios

- O si queremos imprimir el key y el value a la vez.

```
# Imprime los key y value del diccionario
for x, y in d1.items():
    print(x, y)
#Nombre Laura
#Edad 27
#Documento 1003882
#Direccion Calle 123
```

2.4. Diccionarios

- Los diccionarios en Python pueden contener uno dentro de otro. Podemos ver como los valores anidado uno y dos del diccionario d contienen a su vez otro diccionario.

```
anidado1 = {"a": 1, "b": 2}
anidado2 = {"a": 1, "b": 2}
d = {
  "anidado1" : anidado1,
  "anidado2" : anidado2
}
print(d)
#{'anidado1': {'a': 1, 'b': 2}, 'anidado2': {'a': 1, 'b': 2}}
```

2.4. Diccionarios

Métodos	Descripción
.values()	Devuelve los valores del diccionario.
.keys()	Devuelve todas las claves del diccionario.
.items()	Obtenemos los elementos del diccionario en forma tupla (clave, valor)
.clear()	Vacía el diccionario
.pop()	Recibe una clave como parámetro, devuelve el valor asociado a esta y lo borra.

2.4. Diccionarios

Métodos	Descripción
.get()	Recibe una clave como parámetro y devuelve un valor. La diferente con introducir la clave directamente es que si no existe, no va a devolver un error, si no un tipo de dato None
.update()	Recibe otro diccionario como parámetro y los une. Si existe alguna clave igual, actualiza el valor de esta.

2.4. Diccionarios

```
d = {'a': 1, 'b': 2}
print(list(d.values())) #[1, 2]

d = {'a': 1, 'b': 2}
k = d.keys()
print(k)       #dict_keys(['a', 'b'])
print(list(k)) #['a', 'b']

"""El método items() devuelve una lista con los keys y values del diccionario.
Si se convierte en list se puede indexar como si de una lista normal se tratase,
siendo los primeros elementos las key y los segundos los value."""
d = {'a': 1, 'b': 2}
it = d.items()
print(it)             #dict_items([('a', 1), ('b', 2)])
print(list(it))       #[('a', 1), ('b', 2)]
print(list(it)[0][0]) #a
```

2.4. Diccionarios

```
d = {'a': 1, 'b': 2}
d.clear()
print(d) #{}

d = {'a': 1, 'b': 2}
print(d.get('a')) #1
print(d.get('z', 'No encontrado')) #No encontrado

d = {'a': 1, 'b': 2}
d.pop('a')
print(d) #{'b': 2}

d1 = {'a': 1, 'b': 2}
d2 = {'a': 0, 'd': 400}
d1.update(d2)
print(d1)
#{'a': 0, 'b': 2, 'd': 400}
```

2.5. enumerate

- Es un método muy útil cuando necesitas iterar sobre una secuencia (como una lista, tupla o cadena) y al mismo tiempo obtener el índice de cada elemento.

```
lista = ["Gafas","Lentillas","Suero","Funda"]

for indice,elemento in enumerate(lista):
    print(indice,elemento)
"""0 Gafas
1 Lentillas
2 Suero
3 Funda"""
```

2.6. Conversión de tipos

- Del mismo modo que se pueden hacer castings entre datos básicos, se puede hacer lo mismo con algunas de la estructuras de datos.
- Tiene varios propósitos:
  - Realizar operaciones no permitidas en una estructura de datos.

```
#Filtrar una lista
lista_num=[1,1,1,1,3,3,4,5,2,2,2,2,6,6,6]
print(lista_num) #[1, 1, 1, 1, 3, 3, 4, 5, 2, 2, 2, 2, 6, 6, 6]

#Convertimos la lista en un conjunto
con = set(lista_num)
print(con) #[1, 2, 3, 4, 5, 6]

#Volvemos a convertir en lista
lista_sin_repetir = list(con)
print(lista_sin_repetir) #[1, 2, 3, 4, 5, 6]
```

2.6. Conversión de tipos

- Ejemplo de como cambiar de tupla a lista, para modificar su valor.

```
#Las tuplas no se pueden modificar
tupla = (8,9)
#pero las listas sí
#Conversimos la tuplas en lista
lista_t = list(tupla)
print(lista_t) #[8,9]
lista_t[0]=2
print(lista_t) #[2,9]
#Convertimos otra vez la lista en tupla
tupla1 = tuple(lista_t)
print(tupla1) #(2,9)
```

¡Gracias!