Diccionarios en Python – Parte 1

Materia: Redes Neuronales.

Profesora: Camacho Vázquez Vanessa Alejandra.

Un diccionario es muy similar a un archivo JSON, aunque son diferentes tipos de estructuras de datos. Sin embargo, si sabes manejar diccionarios en Python, ya tienes el 95% del conocimiento sobre archivos JSON.

Diccionarios

Los diccionarios se identifican por estar encerrados en llaves `{}` y funcionan a través del concepto de clave, valor y elemento (ítem).

Puntos clave:

A diferencia de las listas, los elementos (ítems) de un diccionario no se acceden mediante índices, sino utilizando las claves *keys* que se definen.

Aunque los diccionarios son iterables, no tienen un orden específico. Por ello, no se pueden acceder por índice como sucede con listas o tuplas.

Veamos un ejemplo:

```
In [1]: Rios = {
...     'Leticia' : 'Amazonas',
...     'Montería': 'Sinu',
...     'Bogotá'  : ['Bogotá', 'Tunjelito'],
...     'San Gil': 'Fonce',
...     'Honda'  : 'Magdalena'
... }

print(Rios)
```

```
{'Leticia': 'Amazonas', 'Montería': 'Sinu', 'Bogotá': ['Bogotá', 'Tunjelito'], 'San Gil': 'Fonce', 'Honda': 'Magdalena'}
```

El objeto *Rios* es un diccionario. Observe que la estructura del diccionario es

{clave:valor, clave:valor, ...}

En el ejemplo:

las claves son: 'Leticia', 'Montería', 'Bogotá', 'San Gil', 'Honda'.

Los respectivos valores son: 'Amazonas', 'Sinu', ['Bogotá', 'Tunjuelito'], 'Fonce', 'Magdalena'.

Note lo que ocurre cuando intentamos acceder por medio de índices (posiciones):

In [2]:
```
print(Rios[0])
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
Cell In [2], line 1
----> 1 print(Rios[0])

KeyError: 0
```

In [5]:
```
print('Los rios de Montería son:', Rios['Montería'])
print('Los rios de Bogotá son:', Rios['Bogotá'])
```

```
Los rios de Montería son: Sinu
Los rios de Bogotá son: ['Bogotá', 'Tunjelito']
```

También es posible crear un diccionario usando la función por defecto (built-in) `dict()`, usando una lista de tuplas:

In [ ]:
```
my_dic=dict([
            ('Colores', ['Negro','Rojo','Azul']),
            ('Animales', 'Gato'),
            ('Calzado', ['Botas','Botines','Deportivos','Sandalias'])
           ])

print(my_dic)
```

```
{'Colores': ['Negro', 'Rojo', 'Azul'], 'Animales': 'Gato', 'Calzado': ['Botas', 'Botines', 'Deportivos', 'Sandalias']}
```

Veamos el tipo de dato que es un diccionario.

In [ ]:
```
print(type(my_dic))
```

```
<class 'dict'>
```

Veamos por ejemplo, los ítems que pertenecen a la llave **Colores**

In [ ]:
```
print(my_dic['Colores'])
```

```
['Negro', 'Rojo', 'Azul']
```

Una vez ingresado a los ítem del diccionario, en caso de ser listas, podemos acceder a sus elementos tal cual lo hacemos con las listas (por sus índices)

In [ ]:
```
print(my_dic['Colores'][1])
```

```
Rojo
```

In [ ]:
```
print('Ví pasar un',my_dic['Animales'],'con',my_dic['Calzado'][0],'de color',my_dic['Colores'][0])
```

```
Ví pasar un Gato con Botas de color Negro
```

Nota: Si las llaves son strings sencillas (sin espacios), también es posible definir un diccionario de la siguiente manera:

In [13]:
```
my_dic=dict(
            Colores = ['Negro','Rojo','Azul'],
            Animales= 'Gato',
            Calzado =['Botas','Botines','Deportivos','Sandalias']
           )

print(my_dic)
```

```
{'Colores': ['Negro', 'Rojo', 'Azul'], 'Animales': 'Gato', 'Calzado': ['Botas', 'Botines', 'Deportivos', 'Sandalias']}
```

Claro está, si se intenta acceder a una llave incorrecta, obtendremos el siguiente error:

In [ ]:
```
print(my_dic['Valor'])
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-35-fe7f2260447b> in <module>
----> 1 print(my_dic['Valor'])

KeyError: 'Valor'
```

Podemos agregar, modificar y eliminar valores de un diccionario:

Agregar ítems al diccionario

In [14]:
```
my_dic['Valor']=['20','50','12']
print(my_dic)
```

```
{'Colores': ['Negro', 'Rojo', 'Azul'], 'Animales': 'Gato', 'Calzado': ['Botas', 'Botines', 'Deportivos', 'Sandalias'], 'Valor': ['20', '50', '12']}
```

Modificar valores:

In [15]:
```
my_dic['Colores']='Negro'
print(my_dic)
```

```
{'Colores': 'Negro', 'Animales': 'Gato', 'Calzado': ['Botas', 'Botines', 'Deportivos', 'Sandalias'], 'Valor': ['20', '50', '12']}
```

Eliminar valores:

In [16]:
```
del my_dic['Valor']
print(my_dic)
```

```
{'Colores': 'Negro', 'Animales': 'Gato', 'Calzado': ['Botas', 'Botines', 'Deportivos', 'Sandalias']}
```

La razón por la cuál no es permitido acceder a los ítems de los diccionarios con índices, es porque los diccionarios no tienen orden. Adicionalmente valores numéricos pueden ser claves en un diccionario.

Las `claves son inmutables` en los diccionarios.

In [6]:
```
d = {135: 'Coco', 1: 'Urban_Sound8k', 2: 'Mnist', 3: 'CheXpert'}
print(d)
```

```
{135: 'Coco', 1: 'Urban_Sound8k', 2: 'Mnist', 3: 'CheXpert'}
```

In [7]:
```
d[135]
```

Out[7]:
```
'Coco'
```

Puede ser confuso al principio, confundir estas llaves con índices. Incluso se podría pensar en tomar rebanadas de él sin éxito o agregar valores como se hace en las listas:

In [8]:
```
d[0:2]
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In [8], line 1
----> 1 d[0:2]

TypeError: unhashable type: 'slice'
```

Ejercicio

Investigue que significa unhashable. Busque la función `hash()` y úsela en este contexto.

n [9]:
```
d.append('Yolo')
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In [9], line 1
----> 1 d.append('Yolo')

AttributeError: 'dict' object has no attribute 'append'
```

Propiedades de los diccionarios

Dinámicos

Este concepto es muy importante, pues resalta la capacidad de un diccionario en incrementar su tamaño sin generar error:

[17]:
```
#Generar diccionario vacío
persona = {}
print(type(persona))

# Agregar llaves y sus definiciones (items)
persona['Nombre'] = 'Gengis'
persona['Apellido'] = 'Khan'
persona['Edad'] = 23
persona['Esposa'] = ['Börte Qatun','Yesugen','Qulan Qatun','Möge Qatun','Juerbiesu','Ibaqa Beki']
persona['Hijos'] = 'En estudio'
persona['Mascotas'] = {'Perro': 'Wahadi', 'Gato': 'Gotze','Leon':'Pichirilo'}

print(persona)
print('Hijos de',persona['Nombre'],':',persona['Hijos'])
```

```
<class 'dict'>
{'Nombre': 'Gengis', 'Apellido': 'Khan', 'Edad': 23, 'Esposa': ['Börte Qatun', 'Yesugen', 'Qulan Qatun', 'Möge Qatun', 'Juerbiesu', 'Ibaqa Beki'], 'Hijos': 'En estudio', 'Mascotas': {'Perro': 'Wahadi', 'Gato': 'Gotze', 'Leon': 'Pichirilo'}}
Hijos de Gengis : En estudio
```

{admonition}
:class: note
Del ejemplo anterior se puede observar que los diccionarios pueden contener diccionarios en su interior:

In [18]:
```
print(persona['Mascotas'])
```

```
{'Perro': 'Wahadi', 'Gato': 'Gotze', 'Leon': 'Pichirilo'}
```

In [19]:
```
print(persona['Mascotas']['Perro'])
```

```
Wahadi
```

Claves

No hay restricciones en la forma de definir las llaves:

In [ ]:
```
foo = {42: 'aaa', 2.78: 'bbb', False: 'cc'}
```

In [ ]:
```
foo[False]
```

Out[ ]:
```
'cc'
```

In [ ]:
```
d = {int: 1, float: 2, bool: 3}

d[int]
```

Out[ ]:
```
1
```

Sin embargo, las claves son únicases decir, no se pueden repetir:

In [11]:
```
foo = {42: 'aaa', 2.78: 'bbb', False: 'cc',False:'dodo'}
foo
```

Out[11]:
```
{42: 'aaa', 2.78: 'bbb', False: 'dodo'}
```

Ejercicio

Defina un diccionario de al menos 4 claves de tal manera que esas llaves sean tuplas. Acceda a cada elemento. ¿Puede hacer lo mismo para una clave que sea definida como lista o diccionario?

Operadores y Métodos:

Es posible utilizar algunos operadores sobre los diccionarios para verificar su estado (por ejemplo, si están o no están disponibles sin generar errores):

In [20]:
```
'Animales' in my_dic
```

Out[20]:
```
True
```

In [21]:
```
'Colores' not in my_dic
```

Out[21]:
```
False
```

También podemos usar lógica aristotélica (tablas de verdad) para chequear cosas sin tener errores:

In [ ]:
```
#my_dic['Valor']
```

In [ ]:
```
#'Valor' in my_dic and my_dic['Valor']
```

len() sobre diccionarios

In [22]:
```
print(my_dic)

print("\nEl diccionario tiene",len(my_dic),'ítems')
```

```
{'Colores': 'Negro', 'Animales': 'Gato', 'Calzado': ['Botas', 'Botines', 'Deportivos', 'Sandalias']}

El diccionario tiene 3 ítems
```

Métodos

d.clear()

In [ ]:
```
print(d)
d.clear()
print(d)
```

```
{<class 'int'>: 1, <class 'float'>: 2, <class 'bool'>: 3}
{}
```

Una vez ingresado a los ítem del diccionario, en caso de ser listas, podemos acceder a sus elementos tal cual lo hacemos con las listas (por sus índices)

d.get(<key>[, <default>])

In [23]:
```
print(my_dic.get('Colores'))
```

```
Negro
```

In [24]:
```
print(persona.get('Esposa'))
```

```
['Börte Qatun', 'Yesugen', 'Qulan Qatun', 'Möge Qatun', 'Juerbiesu', 'Ibaqa Beki']
```

items

In [25]:
```
print(my_dic.items())
```

```
dict_items([('Colores', 'Negro'), ('Animales', 'Gato'), ('Calzado', ['Botas', 'Botines', 'Deportivos', 'Sandalias'])])
```