Day 09: Dictionaries & Nesting.

- Dictionary is a built in python data type.

→ It is simply a type of collection addition to index, it has $\boxed{\text{Key}}$ and $\boxed{\text{a value of that Key.}}$

That' is Python dictionaries are a collection of some Key - value pairs.

Syntax

```
Dictionary = {
        "Key" :  "Value to a Key",
        "Key1" :  "Value 1" ,
        " ___ " : " ______ ",
   }
```

→ Dictionaries are mutable unordered collection with elements in the form of a Key: Value Pairs that associate Key to values.

So, we can access the Values with its Keys.

Lets see How we can do that:

#

$$\begin{tikzcd} & \boxed{\text{Dictionary["Key"]}} \arrow[dl] \arrow[dr] & \\ \text{Name of Dictionary} & & \boxed{\text{Key name}} \end{tikzcd}$$

#

<dictionary-name> = {<Key> : <value>, <Key>: <value>...}

→ The curly brackets marks the begining and end of the dictionary.

→ each entry (Key: value) consists of a pair separated by a colon →
The Key and corres-ponding values is given by writing colon (:) between them.

→ The Key-value pairs are separated by commas (,).

The Key of a dictionary must be of immutable types.
↳ A python string
↳ a tuple.
↳ number

eg → dict = {[2,3]: "abc"}
↓
will give Type Error.
As the Key a list
↓
which is mutable

£ Dictionaries do not allow mutable datatype as its Key data.

Dictionaries are also called associative arrays or mappings or hashes.

The key order is not maintained in dictionaries. This is because the elements (key : value pairs) are unordered, and ~~thee~~ one ~~or~~ cannot access element as per specific order.

$\boxed{\text{The only way to access a value is through key.}}$

→ Now, lets we need to add new items to dictionary.

```
Dictionary["Key3"] = "value 3"

print (Dictionary)
```

└──→ Gonna print all key & value pairs.

↪ key1, key2, $\boxed{\text{key3}}$ ← new one.

we can also new empty dictionary.

$\boxed{\text{empty\_dictionary = \{\}}}$

Using this statement we can also wipe out existing dictionary.

You can't change the Key but only can change the value associated to a Key.

You can add new ~~ideo~~ items in dictionary

#

Tranversing a Dictionary.

Traversal of a collection means accessing and processing each elements of it. Thus, transvering a dictionary also means the same.

The for loop make it easy to traverse or loop over the items in a dictionary, as per following syntax:

```
for <items> in Dictionary:
#
```

Dictionary name.

You can name it as per your choice.

this items are the Key (Not the whole Key and Value)

using the statement you only get the Keys and loop through each.

To access Key and values we can do as:

```
d1 = { 5: "Hello" , "b": "alphabet" }
    for Key in d1:
        print( Key, ":", d1[Key] )
```

This gonna print the Key

This gonna access the Key's values from the dictionary d1.

```
5 : Hello
b : alphabet
```

output.

Nesting.

```
{
    Key : Value,
    Key2: value,
    Key3: {Dict2},
}
```

With nesting we can create or represent data in more complex and nesested format.

Keys & Values → Accessing.

<dictionary>. Keys() ⟶ To see all the keys

<dictionary>. Values() ⟶ To see all the values.

Keys() → all the keys defined in a dictionary in form a sequence

Similiarly,

Values( ) ⟶ all the values defined in a dictionary.

d. Keys() ⟶ dict-keys([. . . . .])

d. Values() ⟶ dict-values([. . . . . .])

list( ) ⟶ List() is useful for converting the sequence returned by Keys() & values() into list.

list(d. Keys()) ⟶ [. . . . .]

list(d. Values()) ⟶ [. . . . .]

Characteristics of a Dictionary.

1. Un Ordered Set.

2. Not a sequence.

3. Indexed by Keys, Not numbers.

4. Keys must be Unique.

5. You cannot change Existing Key, rather it will create a new Key.

6. Mutable

↳ Like list, dictionaries are also mutable we can change the value of a certain key "in place" using the assignment statement

⇓

$\boxed{\texttt{<dictionary>[Key] = <value>}}$

We can add a new Key: value pair to a dictionary But the key being added should be unique. If the key already Exists, the value is simply changed as in the assignment statement above.

7. Internally stored as Mapping.

Multiple ways of creating Dictionaries.

1. Initializing a Dictionary.

```
Employee = {'name': 'Abhishek', 'role': "Developer"}
```

2. Adding Key: Value Pairs to an Empty Dictionary.

→ To create an empty dictionary.

① Employees = { }

② Employee = $\boxed{\text{dict ( )}}$ ⟶ Constructor.

~~ed~~

→ To add new value pairs, one at a time.

<dictionary>[<key>] = <value>

⟹ Employees['age'] = 19

3. Creating a Dictionary from names and value Pair.

(i) $\boxed{\text{Using specific key}}$: Value Pairs as keyword arguments to dict( ) function.

Employee = dict(name = 'XYZ', salary = 1000, age = 24)

⟶ python automatically convert argument names to string type key.

(ii) $\boxed{\text{Specific comma - separated Key : value Pairs.}}$

employee = dict({'name' : 'ABC', 'salary' : 10, 'age' : 20})

The key value pairs are enclose in curly braces as show.

(iii) Specify Keys separately and corresponding values separately.

In this method, the keys and values are enclosed in parenthesis (i.e Tuples) and are given as arguments to the Zip() function, which is then given as argument of dict().

Employee = dict(Zip(('name','salary'), ('Abhi', 2400)))

The zip function club first value from first set with the first value of second set, second value from first set with the second value of second set and so, on.

First group contains all the keys and second group contains all values in same order as that of corresponding keys.

(iv) Specific Key: Value pairs separately in form of sequence.

In this method, one list or tuple argument is passed to dict(). This argument (passed list or tuple) contains list/tuple of individual key: value pairs.

Employee = dict([['name', 'John'], ['salary', 10000]])

Adding Elements to Dictionary.

<dictionary>[<Key>] = <value>

Deleting Elements from a Dictionary.

(i) del ()

→ To delete a dictionary element or a dictionary entry.

del <dictionary>[Key]

If the Key is not present in the dictionary. python raises Exception (KeyError).

(ii) pop() method.

To delete a dictionary element we have another method, named pop()

<dictionary>.pop(<Key>)

→ The pop() method will not only delete the Key: Value pair but also return the corresponding value.

• If you try to delete a Key which doesnot exist, the python returns errors.

However you can display the specify error statement

<dictionary>. pop(<Key>, <in-case-of error>)

employee. pop ('new', "Not found")

→ If the new Key is not found then the error Not found is raised.

Checking For Existence of a key.

Using membership operator in and not in work with dictionaries as well.

They check for the Existence of Keys only.

<Key> $\boxed{\text{in}}$ <dictionary>

<Key> $\boxed{\text{not in}}$ <dictionary>

- The In operator will return True if the given Key is present in the dictionary, otherwise False.
- The not in operator will return True if the given Key is not present in the dictionary. otherwise False.

The $\boxed{\text{in}}$ & $\boxed{\text{not in}}$ operator check for membership only in Keys of the dictionary and not in Values.

Pretty Printing a Dictionary. dump()

printing a large dictionary could be hard to read and therefore we have a way of making it more readable and presentable

→ we need to import json module.

```
import json

json.dumps(<dictionary name>, inent=<n>)
```

↑ This is a number(n)

Split()

The split() works on string type data and breaks up a string into words and creates a list out of it

```
text = " This is sample text"

words = text.split().
```

→ words : ['This', 'is', 'sample', 'text']

Also, we can pass something in split().

```
text = " one, Two, three"

words = text.split(,)
```

→ words : ['one', 'Two', 'three']

default split() breaks up text based on white spaces. but we can give a separate character ~~also~~, Eg ~~&~~ like , ! ; : . . . .

Dictionary Functions & Methods

1. The len() function/method

   This method returns length of the dictionary i.e count the elements.

   len(<dictionary>)

2. The clear() ~~func~~ method

   This method removes all items from the dictionary and the dicitionary becomes empty dictionary.

   <dictionary>.clear()

3. The get () method.

   This method gives you the item with the given key, similar to dictionary[key].

   → If the key is not present, error is raise which you can customise.

   $<dictionary>.get(key, \underbrace{[default]}_{})$

   → You can pass your personalized message.

4. The items() method.

   This method returns all the items in the dicitionary as a sequence of (key, value) tuples.

   <dictionary>.items()

5. The Keys() method

→ Returns all of the Keys in the dictionary.

<dictionary>.Keys()

6. The Values() Method.

→ Returns all the values from the dicitionary

<dictionary>.values()

7. The update() method

→ This method merges Key: value Pairs from the new dictionary into the original dictionary adding or replacing as needed.

<dictionary>.update(<other.dictionary>)

Dictionary to be updated

This dictionary's items will be taken for updating Other dictionary.