Dictionaries :

Dictionaries have **key-value pairs**, enabling **efficient lookup** and **retrieval** of values based on keys.

* dictionaries are **unordered** collections, meaning they don't maintain a specific order of elements.
* **Mutable in nature**, dictionaries allow for dynamic modifications, including adding, removing, or modifying key-value pairs as needed.
* **With keys required to be unique**, dictionaries ensure that each key maps to only one value, though the values themselves can be heterogeneous, accommodating various data types.
* This flexibility, combined with fast lookup times, makes dictionaries ideal for applications where **rapid access** to data is crucial.
* Additionally, Python supports **dictionary comprehension**,
* These features, coupled with dictionaries' **iterability and dynamic size**, establish them as fundamental tools in Python programming, widely utilized for storing and accessing data, **representing mappings**, **implementing lookup tables**, and various other tasks requiring associative arrays.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | Dictionary | | |
|  |  |  |  |  |
| Creation | Simple dictionary | thisdict = { "brand" : "Ford",  "model" : "Mustang",  "year" : 1964 } | | |
|
| # Nested dictionary | DictFirst = { 'Dict1': {'name': 'Ali', 'age': 19},  'Dict2': {'name': 'Bob', 'age': 25}} | | |
|
|
| by USING Constructor | thisdict = dict(brand="Ford", model="Mustang", year=1964) | | |
|
|
|  |  |  |  |  |
| Access / Slicing Items | |  |  |  |
| Get the value of the "model" key | | |
|  |  |  |
| x = thisdict["model"] ans= Mustang | | |
|
|
|  |  |  |
| Get the value of the "model" key by using get() | | |
|  |  |  |
| x = thisdict.get("model") ans= Mustang | | |
|
|
|  | | |
|  |  |  |  |  |
| Loop Through | | #To Print all key names: | #To print all value:- | #To print all Key & Value: |
| for x in thisdict.keys():  print(x) | for x in thisdict.values():  print(x) | for key, value in thisdict.items():  print(key, value) |
|
|
| brand model year | Ford Mustang 1964 | brand Ford model Mustang year 1964 |
|
|
|
| OR | OR | OR |
|
| for x in thisdict:  print(x) | for x in thisdict:  print(thisdict[x]) | for x in thisdict:  print(x,thisdict[x]) |
|
|
| brand model year | Ford Mustang 1964 | brand Ford model Mustang year 1964 |
|
|
|  |  |  |  |  |
| Length | | len(abc) | | |
|
|  |  |  |  |  |
| Check if Item Exists | | if "model" in thisdict:  print("Yes, model is one of the keys in the thisdict dictionary") | | |
|
|
|  |  |  |  |  |
| Changing the item | | thisdict["year"] = 2018 | | |
|
| thisdict.update({"brand": "tata"}) | | |
|
|  |  |  |  |  |
| Adding an Item | To add an item at the Last index | car.update({"color": "White"}) | | |
|
|
|
|  |  |  |  |
| To add an item at the specified index | ***------*** | | |
|
|
|  |  |  |  |  |
| Remove | To remove the specified item: | del thisdict["model"] | | |
|
| thisdict.pop("model") | | |
|
| To remove the last item: | thisdict.popitem() | | |
| To remove by Index | ***----*** | | |
|  |  |  |  |  |
| Clear | To empty the list: | thisdict.clear() | | |
| Delete | To delete the Data Completely | del thisdict | | |
|  |  |  |  |  |
| Copy | Method 01 | mydict = thisdict.**copy()** print(mydict) | | |
|
|
| Method 02 | mydict = dict(thisdict) print(mydict) | | |
|
|
|  |  |  |  |  |
| Join |  | thisdict = {"brand": "Ford", "model": "Mustang", "year": 1964} newDict = {"colour": "Red", "Country": "Germany"} | | |
|
|
| Method 01 | thisdict.update(newDict) print(thisdict) | | |
|
|
| Method 02 | merged\_dict = {\*\*thisdict, \*\*newDict} print(merged\_dict) | | |
|
|
| Method 03 | merged\_dict = dict(thisdict, \*\*newDict) print(merged\_dict) | | |
|
|
|  |  |  |  |  |
| Number of elements in dataset | | ***---*** | | |
|
|
|
| Index of the first element with the specified value | | ***---*** | | |
|
| Reverses the order | | ***---*** | | |
|
| Sorts the list | | ***---*** | | |
|

Dictionary comprehension is similar to list comprehension, but instead of creating lists, it creates dictionaries. It provides a **concise and efficient** way to generate dictionaries using an iterable and an expression to define key-value pairs. The syntax for dictionary comprehension is:

**{ key\_expression : value\_expression for x in iterable if condition }**

**{ x : x\*\*2 for x in Range(10) if x % 2 == 0}**