Lecture 08

Data types (Dictionaries)

Python Dictionaries

Dictionary in Python is a collection of keys values, used to store data values like a map, which, unlike other data types which hold only a single value as an element. The dictionary is defined into element Keys and values.

- Keys must be a single element
- Value can be any type such as list, tuple, integer, etc.

A dictionary is an associative array (also known as hashes). Any key of the dictionary is associated (or mapped) to a value. The values of a dictionary can be any Python data type. So dictionaries are unordered key-value-pairs.

Properties

- **Accessed by key, not offset position**

Dictionaries are sometimes called associative arrays or hashes. They associate a set of values with keys, so an item can be fetched out of a dictionary using the key under which it is originally stored. The same indexing operation can be utilized to get components in a dictionary as in a list, but the index takes the form of a key, not a relative offset.

- **Unordered collections of arbitrary objects**

Unlike in a list, items stored in a dictionary aren't kept in any particular order. Keys provide the symbolic (not physical) locations of items in a dictionary.

Note – As of Python version 3.7, dictionaries are ordered. In Python 3.6 and earlier, dictionaries are unordered.

- **Variable-length, heterogeneous, and arbitrarily nestable**

Like lists, dictionaries can grow and shrink in place (without new copies being made), they can contain objects of any type, and they support nesting to any depth (they can contain lists, other dictionaries, and so on). Each key can have just one associated value, but that value can be a collection of multiple objects if needed, and a given value can be stored under any number of keys.

- **Of the category "mutable mapping"**

Dictionary allows in place changes by assigning to indexes (they are mutable), but they don't support the sequence operations that work on strings and lists. Because dictionaries are unordered collections, operations that depend on a fixed positional order (e.g., concatenation, slicing) don't make sense.

Instead, dictionaries are the only built-in, core type representatives of the mapping category— objects that map keys to values. Other mappings in Python are created by imported modules.

- **Tables of object references (hash tables)**

If lists are arrays of object references that support access by position, dictionaries are unordered tables of object references that support access by key. Internally, dictionaries are implemented as hash tables (data structures that support very fast retrieval), which start small and grow on demand. Moreover, Python employs optimized hashing algorithms to find keys, so retrieval is quick. Like lists, dictionaries store object references (not copies, unless explicitly asked).

Create a Dictionary

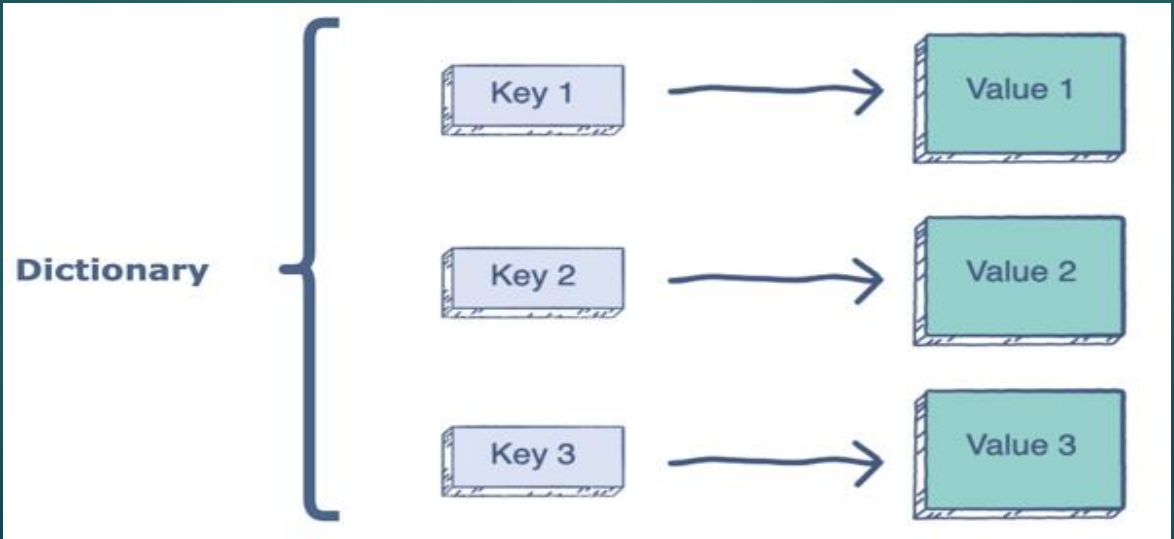
In Python, a dictionary can be created by placing a sequence of elements within curly {} braces, separated by 'comma'. Dictionary holds pairs of values, one being the Key and the other corresponding pair element being its Key value. Dictionary can also be created by the built-in function dict( ).

Note – Dictionary keys are case sensitive, the same name but different cases of Key will be treated distinctly.

```
>>> dict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
>>> dict
{'brand': 'Ford', 'model': 'Mustang', 'year': 1964}
>>>
```

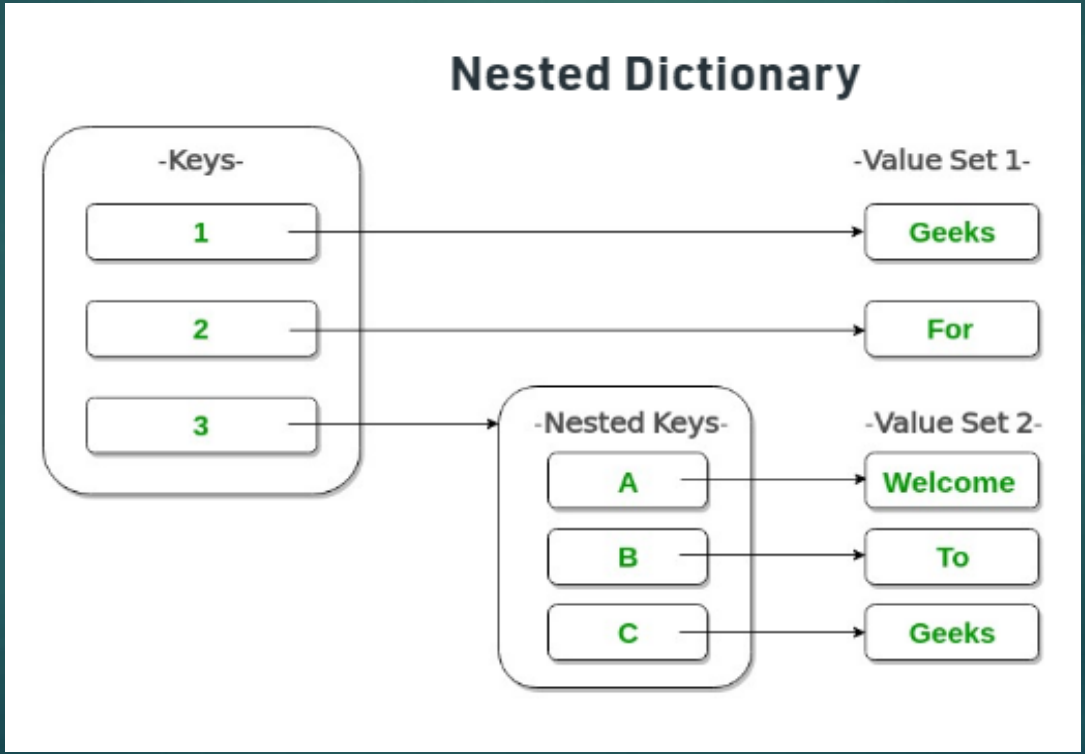
How does the dictionary works in Python?

Each key is associated with a single value. The association of a key and a value is called a key-value pair or an item. Keys are unique within a dictionary while values may not be. The values of a dictionary can be of any type, but the keys must be of an immutable data type (i.e., strings, numbers, or tuples).



Nested Dictionary

The dictionaries can be nested, like dictionary within dictionary. For this nested keys are defined.



Dictionary Item

The values in dictionary items can be of any data type. Dictionary items are presented in key:value pairs, and can be referred to by using the key name.

```
>>> dict = {
  "1": "apple",
  "2": False,
  "3": 1964
}
>>> dict
{'1': 'apple', '2': False, '3': 1964}
>>> print(dict['1'])
apple
>>>
```

When we say that dictionaries are ordered, it means that the items have a defined order, and that order will not change. Unordered means that the items does not have a defined order, you cannot refer to an item by using an index. **Dictionaries cannot have two items with the same key.**

Dictionary Length

To determine how many items a dictionary has, use the **len()** function:

```
>>> dict
{'1': 'apple', '2': 'mani', '3': 1964}
>>> print(dict['1'])
apple
>>> print(len(dict))
3
>>>
```

type()

From Python's perspective, dictionaries are defined as objects with the data type 'dict'.

```
>>> print(type(dict))
<class 'dict'>
>>>
```

Access

1. The items of a dictionary can be accessed by referring to its key name, inside square brackets.
2. Another method called to access is **get()** that will give you the same result.

```
>>> dict
{'1': 'corolla', '2': 'alto', '3': 'farrari', '4': 'altis', '5': 'cultus', '6':
'city', '7': '1993'}
>>> x=dict['5']
>>> x
'cultus'
>>> y=dict.get('4')
>>> y
'altis'
>>>
```

Get Keys

The **keys()** method will return a list of all the keys in the dictionary. The list of the keys is a view of the dictionary, meaning that any changes done to the dictionary will be reflected in the keys list.

Add a new item to the original dictionary, and see that the keys list gets updated as well.

```
>>> x=dict.keys()
>>> x
dict_keys(['1', '2', '3', '4', '5', '6', '7'])
>>> dict['8']='Alto VXL'
>>> dict
{'1': 'corolla', '2': 'alto', '3': 'farrari', '4': 'altis', '5': 'cultus', '6':
'city', '7': '1993', '8': 'Alto VXL'}
>>> x
dict_keys(['1', '2', '3', '4', '5', '6', '7', '8'])
>>>
```

Get Values

The **values()** method will return a list of all the values in the dictionary. The list of the values is a view of the dictionary, meaning that any changes done to the dictionary will be reflected in the values list. Make a change in the original dictionary, and see that the values list gets updated, or add a new item to the original dictionary, and see that the values list gets updated as well.

```
>>> a=dict.values()
>>> a
dict_values(['corolla', 'alto', 'farrari', 'altis', 'cultus', 'city', '1993', 'A
lto VXL'])
>>> dict['year']=2022
>>> a
dict_values(['corolla', 'alto', 'farrari', 'altis', 'cultus', 'city', '1993', 'A
lto VXL', 2022])
>>>
```

Get Items

The **items()** method will return each item in a dictionary, as tuples in a list.
The returned list is a view of the items of the dictionary, meaning that any changes done to the dictionary will be reflected in the items list. Make a change or add a new item in the original dictionary, and see that the items list gets updated as well.

```
>>> b=dict.items()
>>> b
dict_items([('1', 'corolla'), ('2', 'alto'), ('3', 'farrari'), ('4', 'altis'), (
'5', 'cultus'), ('6', 'city'), ('7', '1993'), ('8', 'Alto VXL'), ('year', 2022)]
)
>>> dict['year']=1947
>>> b
dict_items([('1', 'corolla'), ('2', 'alto'), ('3', 'farrari'), ('4', 'altis'), (
'5', 'cultus'), ('6', 'city'), ('7', '1993'), ('8', 'Alto VXL'), ('year', 1947)]
)
>>>
```

Check existence of Key

To determine if a specified key is present in a dictionary use the **in** keyword:

```
if "8" in dict:
  print("Yes, it is one of the keys in the dictionary")
else:
    print ('not in the list')
```

```
>>>
==== RESTART: C:\Users\Hera Noor\AppData\Local
Yes, it is one of the keys in the dictionary
>>>
```

Change and Update

You can change the value of a specific item by referring to its key name.

```
>>> dict
{'1': 'corolla', '2': 'alto', '3': 'farrari', '4': 'altis', '5': 'cultus', '6':
'city', '7': '1993', '8': 'Alto VXL', 'year': 1947}
>>> dict['4']='apple'
>>> dict
{'1': 'corolla', '2': 'alto', '3': 'farrari', '4': 'apple', '5': 'cultus', '6':
'city', '7': '1993', '8': 'Alto VXL', 'year': 1947}
```

The **update()** method will update the dictionary with the items from the given argument. The argument must be a dictionary, or an inerrable object with key:value pairs.

```
>>> dict.update({'4':'altis'})
>>> dict
{'1': 'corolla', '2': 'alto', '3': 'farrari', '4': 'altis', '5': 'cultus', '6':
'city', '7': '1993', '8': 'Alto VXL', 'year': 1947}
>>>
```

Remove Items

There are several methods to remove items from a dictionary.

- The **pop()** method removes the item with the specified key name
- The **popitem()** method removes the last inserted item (in versions before 3.7, a random item is removed instead).
- The **del** keyword removes the item with the specified key name. The del keyword can also delete the dictionary completely.
- The **clear()** method empties the dictionary.

```
>>> dict
{'1': 'corolla', '2': 'alto', '3': 'farrari', '4': 'altis', '5': 'cultus', '6':
'city', 'color': 'green'}
>>> dict.pop('color')
'green'
>>> dict.popitem()
('6', 'city')
>>> del dict['1']
>>> dict
{'2': 'alto', '3': 'farrari', '4': 'altis', '5': 'cultus'}
>>> dict.clear()
>>> dict
{}
```

Copy Dictionary

You cannot copy a dictionary simply by typing dict2 = dict1, because: dict2 will only be a reference to dict1, and changes made in dict1 will automatically also be made in dict2.

The way to make a copy is to use the built-in Dictionary method copy().

```
>>> dict
{'2': 'alto', '3': 'farrari', '4': 'altis', '5': 'cultus'}
>>> mydict=dict.copy()
>>> mydict
{'2': 'alto', '3': 'farrari', '4': 'altis', '5': 'cultus'}
>>>
```

Nested Dictionary

A dictionary can contain dictionaries, this is called nested dictionaries.

```
>>> myfamily = {
  "child1" : {
    "name" : "Zuha",
    "year" : 2009
  },
  "child2" : {
    "name" : "Mohammad Ahmed",
    "year" : 2014
  },
  "child3" : {
    "name" : "Hasan",
    "year" : 2017
  }
}
```

Nested Dictionary

Or, if you want to add three dictionaries into a new dictionary.

```
>>> child1 = {
  "name" : "Emil",
  "year" : 2004
}
>>> child1 = {
  "name" : "Zuha",
  "year" : 2009
}
>>> child2 = {
  "name" : "Mohammad Ahmed",
  "year" : 2014
}
>>> child3 = {
  "name" : "Hasan",
  "year" : 2019
}
>>> myfamily = {
  "child1" : child1,
  "child2" : child2,
  "child3" : child3
}
```

Built-in Functions

Function	Description
cmp(dict1, dict2)	It compares the items of both the dictionary and returns true if the first dictionary values are greater than the second dictionary, otherwise it returns false.
len(dict)	It is used to calculate the length of the dictionary.
str(dict)	It converts the dictionary into the printable string representation.
type(variable)	It is used to print the type of the passed variable.

Summary

- **clear()** – Remove all the elements from the dictionary
- **copy()** – Returns a copy of the dictionary
- **get()** – Returns the value of specified key
- **items()** – Returns a list containing a tuple for each key value pair
- **keys()** – Returns a list containing dictionary's keys
- **fromkeys()** – Returns a dictionary with the specified keys and value
- **pop()** – Remove the element with specified key
- **popitem()** – Removes the last inserted key-value pair
- **update()** – Updates dictionary with specified key-value pairs
- **values()** – Returns a list of all the values of dictionary
- **setdefault()** – Returns the value of the specified key. If the key does not exist: insert the key, with the specified value.
- The **del** keyword removes the item with the specified key name.

Task – 8

1. Write a script to develop the given dictionary:

   D={'CP':'COMPUTER PROGRAMMING', 'FCE':'FUNDAMENTALS OF COMPUTER ENGINEERING', 'PST':'PAKISTAN STUDIES', 'BEE':'BASICS OF ELECTRICAL ENGINEERING',}

2. Write a statement to add 'F.ENG': 'FUNCTIONAL ENGLISH' in the dictionary of Q1.
3. Write a statement to find out whether the given key (Chinese) is the part of dictionary in Q1.
4. Write a statement to print the value by providing the key (‘CP’) to the dictionary in Q1.
5. Write a program that takes a list of multiple choice responses. E.g. [a, b, c] and prints a dictionary of question-response pairs {‘Q1’: ‘a’, ‘Q2’:‘b’, ‘Q3’:‘c’}.

Thank you!