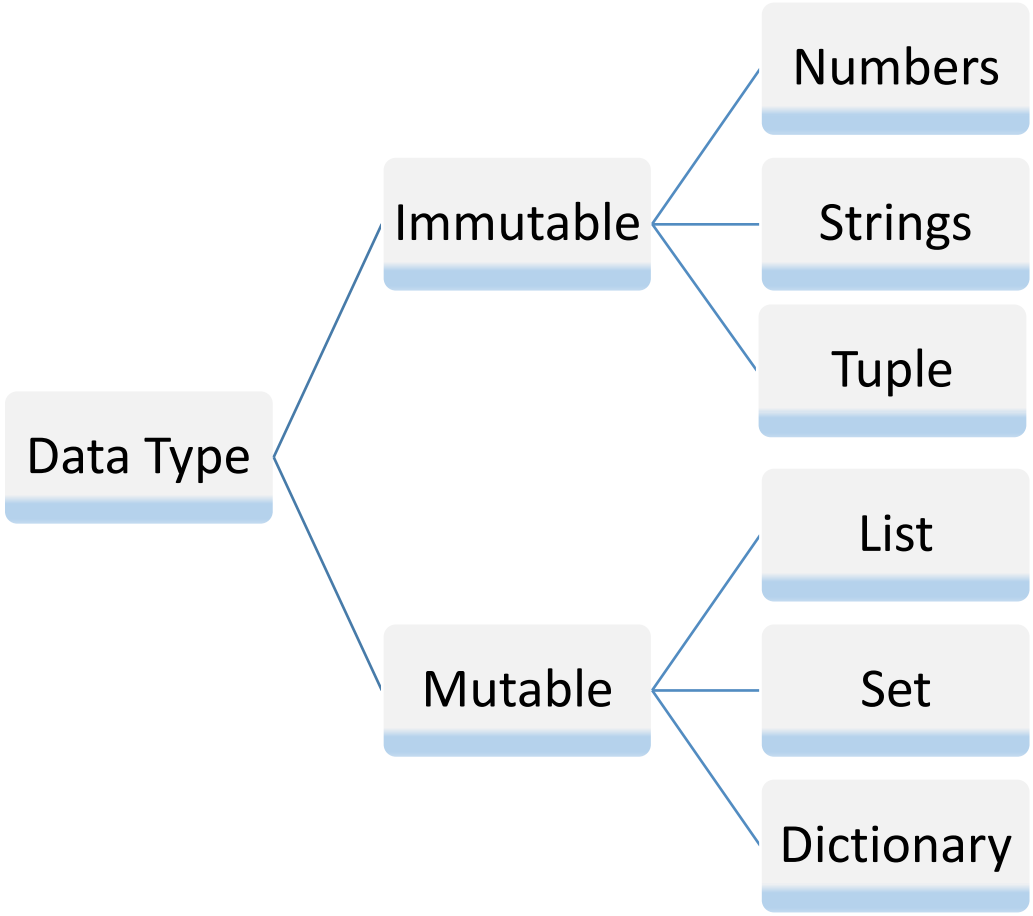
Data Types/Data Structures (Sets and Dictionaries)

Data Types/ Inbuilt Data Structures in Python



Sets

Sets

- A set is a collection of data types in Python, same as the list and tuple. However, it is not an ordered collection of objects. The set is a Python implementation of the set in Mathematics.
- A set object has suitable methods to perform mathematical set operations like union, intersection, difference, etc.
- A set object contains one or more items, not necessarily of the same type, which are separated by comma and enclosed in curly brackets {}.

Syntax:

set = {value1, value2, value3,...valueN}

The following defines a set object.

S1={1, "Bill", 75.50}

Sets(contd...)

A set doesn't store duplicate objects. Even if an object is added more than once inside the curly brackets, only one copy is held in the set object. Hence, indexing and slicing operations cannot be done on a set object.

S1={1, 2, 2, 3, 4, 4, 5, 5}

S1

{1, 2, 3, 4, 5}

Sets(contd...)

Python has an in-built function set(), using which a set object can be constructed out of any sequence such as a string, list or a tuple object.

```
s1=set("Python")
 s1
{'t', 'h', 'o', 'n', 'P', 'y'}
 s2=set([45,67,87,36, 55])
 s2
{67, 36, 45, 87, 55}
 s3=set((10,25,15))
 s3
{25, 10, 15}
```

The order of elements in the set is not necessarily the same as the order given at the time of assignment. Python optimizes the structure of a set for performing operations over it, as defined in mathematics.

Set Operations

- Various set operations can be performed. Operators |, &, - and ^ perform union, intersection, difference and symmetric difference operations, respectively.
- Each of these operators has a corresponding method associated with the built-in set class.

Set Operations (contd...)

Operation	Operator/Method	Example

	|	>>> s1={1,2,3,4,5}
		>>> s2={4,5,6,7,8}
		>>> s1|s2
		{1, 2, 3, 4, 5, 6, 7, 8}

The union of two sets is a set of all elements from both the collections.	union()	>>> s1={1,2,3,4,5}
		>>> s2={4,5,6,7,8}
		>>> s1.union(s2)
		{1, 2, 3, 4, 5, 6, 7, 8}
		>>> s2.union(s1)
		{1, 2, 3, 4, 5, 6, 7, 8}

Set Operations (contd...)

The intersection of two sets is a set containing elements common to both collections.

&

```
>>> s1={1,2,3,4,5}
>>> s2={4,5,6,7,8}
>>> s1&s2
{4, 5}
>>> s2&s1
{4, 5}
```

intersection()

```
>>> s1={1,2,3,4,5}
>>> s2={4,5,6,7,8}
>>> s1.intersection(s2)
{4, 5}
>>> s2.intersection(s1)
{4, 5}
```

Set Operations (contd...)

-

The difference of two sets results in a set containing elements only in the first set, but not in the second set.

```
>>> s1={1,2,3,4,5}
>>> s2={4,5,6,7,8}
>>> s1-s2
{1, 2, 3}
>>> s2-s1
{8, 6, 7}
```

difference()

```
>>> s1={1,2,3,4,5}
>>> s2={4,5,6,7,8}
>>> s1.difference(s2)
{1, 2, 3}
>>> s2.difference(s1)
{8, 6, 7}
```

Set Operations (contd...)

Symmetric Difference: the result of symmetric difference is a set consisting of elements in both sets, excluding the common elements.

^

Symmetric_difference()

```
>>> s1={1,2,3,4,5}
>>> s2={4,5,6,7,8}
>>> s1^s2
{1, 2, 3, 6, 7, 8}
>>> s2^s1
{1, 2, 3, 6, 7, 8}
>>> s1={1,2,3,4,5}
>>> s2={4,5,6,7,8}
>>>
s1.symmetric_differenc
e(s2)
{1, 2, 3, 6, 7, 8}
>>>
s2.symmetric_differenc
e(s1)
{1, 2, 3, 6, 7, 8}
```

Built-in Set Methods

add()

Adds a new element in the set object.

```
S1 ={'Java', 'Python', 'C++'}
 S1.add("Perl")
 S1
 {'Java', 'Python', 'Perl', 'C++'}
```

Built-in Set Methods(contd...)

update()

Adds multiple items from a list or a tuple.

```
S1={"Python", "Java", "C++"}
S1.update(["C", "Basic"])
S1
{'C++', 'Java', 'Python', 'Basic', 'C'}
S1.update(("Ruby", "PHP"))
S1
 {'C++', 'Ruby', 'Java', 'PHP', 'Python', 'Basic', 'C'}
```

Built-in Set Methods(contd...)

clear()

Removes the contents of set object and results in an empty set.

S1.clear()

S1

set()

Built-in Set Methods(contd...)

copy()

Creates a copy of the set object.

```
S1={"Python", "Java", "C++"}
S2=S1.copy()
S2
 {'Java', 'Python', 'C++'}
```

discard()

Returns a set after removing an item from it. No changes are done if the item is not present.

```
S1={"Python", "Java", "C++"}
S1.discard("Java")
S1
 {'Python', 'C++'}
```

Built-in Set Methods(contd...)

remove()

Returns a set after removing an item from it. Results in an error if the item is not present.

```
S1={"Python", "Java", "C++"}
S1.remove("C++")
S1
 {'Java', 'Python'}
S1={"Python", "Java", "C++"}
S1.remove("SQL")
 KeyError: 'SQL'
```

Python Set Methods	
Method	**Description**
add()	Adds an element to the set
clear()	Removes all elements from the set
copy()	Returns a copy of the set
difference()	Returns the difference of two or more sets as a new set
difference_update()	Removes all elements of another set from this set
discard()	Removes an element from the set if it is a member. (Do nothing if the element is not in set)
intersection()	Returns the intersection of two sets as a new set
intersection_update()	Updates the set with the intersection of itself and another
isdisjoint()	Returns True if two sets have a null intersection
issubset()	Returns True if another set contains this set
issuperset()	Returns True if this set contains another set
pop()	Removes and returns an arbitary set element. Raise KeyError if the set is empty
remove()	Removes an element from the set. If the element is not a member, raise a KeyError
symmetric_difference()	Returns the symmetric difference of two sets as a new set
symmetric_difference_update()	Updates a set with the symmetric difference of itself and another
union()	Returns the union of sets in a new set
update()	Updates the set with the union of itself and others

Built-in Functions with Set	
Function	**Description**
all()	Return True if all elements of the set are true (or if the set is empty).
any()	Return True if any element of the set is true. If the set is empty, return False.
enumerate()	Return an enumerate object. It contains the index and value of all the items of set as a pair.
len()	Return the length (the number of items) in the set.
max()	Return the largest item in the set.
min()	Return the smallest item in the set.
sorted()	Return a new sorted list from elements in the set(does not sort the set itself).
sum()	Retrun the sum of all elements in the set.

Frozenset

- Frozenset is a new class that has the characteristics of a set, but its elements cannot be changed once assigned.
- This datatype supports methods like copy(), difference(), intersection(), isdisjoint(), issubset(), issuperset(), symmetric_difference() and union().
- Being immutable it does not have method that add or remove elements

```
# initialize
A = frozenset([1, 2, 3, 4])
B = frozenset([3, 4, 5, 6])
```

Dictionaries

Dictionaries

- Like the list and the tuple, dictionary is also a collection type.
- However, it is not an ordered sequence, and it contains key-value pairs. One or more key:value pairs separated by commas are put inside curly brackets to form a dictionary object.

Syntax:

dict = { key1:value1, key2:value2,...keyN:valueN }

Dictionaries(contd...)

- The following declares a dictionary object.

capitals={"USA":"Washington, D.C.", "France":"Paris", "India":"New Delhi"}

- In the above example, capitals is a dictionary object.
- The left side of : is a key and right side of : is a value.
- The key should be an immutable object.
- A number, string or tuple can be used as key.

Dictionaries(contd…)

- The following definitions of dictionary are also valid:

numNames={1:"One", 2: "Two", 3:"Three“}.

items={("Parker","Reynolds","Camlin"):"pen",("LG","Whirlpool","Samsung"): "Refrigerator"}.

- However, a dictionary with a list as a key is not valid, as the list is mutable:

dict={["Mango","Banana"]:"Fruit", ["Blue", "Red"]:"Colour"}.
TypeError: unhashable type: 'list'

Dictionaries(contd…)

- But, a list can be used as a value.

dict={"Fruit":["Mango","Banana"], "Colour":["Blue", "Red"]}

- The same key cannot appear more than once in a collection. If the key appears more than once, only the last will be retained. The value can be of any data type. One value can be assigned to more than one key.

staff={"Krishna":"Officer", "Steve":"Manager",
"John":"officer", "Anil":"Clerk", "John":"Manager"}
staff
{"Krishna":"Officer", "Steve":"Manager", "Anil":"Clerk",
"John":"Manager"}

Accessing a Dictionary

- Dictionary is not an ordered collection, so a value cannot be accessed using an index in square brackets.
- A value in a dictionary can be fetched using the associated key, using the get() method. Specify the key in the get() method to retrieve its value.

```
capitals={"USA":"New York", "France":"Paris", "Japan":"Tokyo",
   "India":"New Delhi"}
capitals.get("France")
'Paris‘

points={"p1":(10,10), "p2":(20,20)}
points.get("p2")
(20,20)
```

Updating Dictionary

- The key cannot appear more than once. Use the same key and assign a new value to it to update the dictionary object.

```
captains={"England":"Root", "Australia":"Smith",
  "India":"Dhoni"}
   captains['India']='Virat'
   captains['Australia']='Paine'
   captains
  {'England': 'Root', 'Australia': 'Paine', 'India': 'Virat'}
```

- Use a new key and assign a value to it. The dictionary will show an additional key-value pair in it.

```
captains['SouthAfrica']='Plessis‘

captains
  {'England': 'Root', 'Australia': 'Paine', 'India': 'Virat',
  'SouthAfrica': 'Plessis'}
```

Deleting Values from a Dictionary

- Use the **del** keyword to delete a pair from a dictionary or the dictionary object itself. To delete a pair, use its key as parameter. To delete a dictionary object, use its name.

```
captains={'England': 'Root', 'Australia': Paine', 'India': 'Virat',
  'Srilanka': 'Jayasurya'}

 del captains['Srilanka']

captains
  {'England': 'Root', 'Australia': Paine', 'India': 'Virat‘

 del captains

captains
  NameError: name 'captains' is not defined NameError
  indicates that the dictionary object has been removed from
  memory.
```

View Keys and Values

- The keys() and values() methods of Python dictionary class return a view object consisting of keys and values respectively, used in the dictionary.

```
d1 = {'name': 'Steve', 'age': 21, 'marks': 60, 'course': 'Computer Engg'}
d1.keys()
dict_keys(['name', 'age', 'marks', 'course'])
```

- The result of the keys() method is a view which can be stored as a list object. If a new key-value pair is added, the view object is automatically updated.

```
keys=d1.keys()
keys
dict_keys(['name', 'age', 'marks', 'course'])
d1.update({"college":"IITB"})
keys
dict_keys(['name', 'age', 'marks', 'course', 'college'])
```

View Keys and Values (contd…)

- This is similar for the values() method.

```
d1= {'name': 'Steve', 'age': 21, 'marks': 60, 'course': 'Computer
  Engg'}
  values=d1.values()
  dict_values(['Steve', 21, 60, 'Computer Engg'])
```

- The result of the values() method is a view which can be stored as a list object. If a new key-value pair is added, the view is dynamically updated.

```
d1.update({"college":"IITB"})
 values
dict_values(['Steve', 21, 60, 'Computer Engg', 'IITB'])
```

Multi-dimensional Dictionary

- Let's assume there are three dictionary objects, as below:

```
d1={"name":"Steve","age":25, "marks":60}
 d2={"name":"Anil","age":23, "marks":75}
 d3={"name":"Asha", "age":20, "marks":70}
```

- Let us assign roll numbers to these students and create a multi-dimensional dictionary with roll number as key and the above dictionaries at their value.

```
students={1:d1,2:d2,3:d3}
 students
  {1: {'name': 'Steve', 'age': 25, 'marks': 60}, 2: {'name': 'Anil',
  'age': 23, 'marks': 75}, 3: {'name': 'Asha', 'age': 20, 'marks':
  70}}
```

Multi-dimensional Dictionary (contd...)

- The students object is a two-dimensional dictionary. Here d1, d2 and d3 are assigned as values to keys 1,2, and 3, respectively. students [1] returns d1.

```
students[1]
```

{'name': 'Steve', 'age': 25, 'marks': 60} The value of a key inside d1 can be obtained as below:

```
students[1]['age']
25
```

Built-in Dictionary Methods

- **len()**

Returns the number of key:value pairs in the dictionary.

```
lang={'A':('Ada','ActionScript'), 'P':("Python", "Perl","PHP")}
len(lang)
  2
```

- **max()**

If all keys in the dictionary are numbers, the heighest number will be returned. If all keys in the dictionary are strings, the one that comes last in alphabetical order will be returned.

```
lang={'J':'Java', 'A': 'ActionScript', 'P':'Python'}
max(lang)
'P‘
num={5:"five", 100:"hundred",3:"three"}
max(num)
100
```

Built-in Dictionary Methods (contd...)

- **min()**

If all keys in the dictionary are numbers, the lowest number will be returned. If all keys in the dictionary are strings, the one that comes first in alphabetical order will be returned.

```
lang={'J':'Java', 'A': 'ActionScript', 'P':'Python'} min(lang)
'A'
num={5:"five", 100:"hundred",3:"three"}
min(num)
3
```

Built-in Dictionary Methods (contd...)

- **pop()**

Returns the value associated with the key and the corresponding key-value pair is removed.

```
captains={'England': 'Root', 'Australia': 'Smith', 'India': 'Virat',
 'Pakistan': 'Sarfraz'}
captains.pop('India')
 'Virat‘
captains
 {'England': 'Root', 'Australia': 'Smith', 'Pakistan': 'Sarfraz'}
```

Built-in Dictionary Methods (contd...)

- **clear()**

Returns empty object by deleting all the key-value pairs.

```
captains
 {'England': 'Root', 'Australia': 'Smith', 'Pakistan': 'Sarfraz'}
 captains.clear()
 captains
 {}
```

Built-in Dictionary Methods (contd...)

- **items()**

Returns a list of tuples, each tuple containing the key and value of each pair.

```
captains={'England': 'Root', 'Australia': 'Smith', 'India': 'Virat', 'Pakistan': 'Sarfraz'}

captains.items()
```

```
dict_items([('England', 'Root'), ('Australia', 'Smith'), ('India', 'Virat'), ('Pakistan', 'Sarfraz')])
```

Built-in Dictionary Methods (contd...)

- **update()**

Adds key-value pairs from the second dictionary object to the first. If the second dictionary contains a key already used in first dictionary object, its value is updated.

```
mark1={"Sagar":67, "Amrita":88, "Bhaskar":91, "Kiran":49}
mark2={"Arun":55, "Bhaskar":95, "Geeta":78}
mark1.update(mark2)
mark1
{'Sagar': 67, 'Amrita': 88, 'Bhaskar': 95, 'Kiran': 49, 'Arun': 55, 'Geeta': 78}
```

mark1 dictionary now has new items from mark2, with the value of one key updated.

Python Dictionary Methods	
Method	**Description**
clear()	Remove all items form the dictionary.
copy()	Return a shallow copy of the dictionary.
fromkeys(seq[, v])	Return a new dictionary with keys from seq and value equal to v (defaults to None).
get(key[,d])	Return the value of key. If key doesnot exit, return d (defaults to None).
items()	Return a new view of the dictionary's items (key, value).
keys()	Return a new view of the dictionary's keys.
pop(key[,d])	Remove the item with key and return its value or d if key is not found. If d is not provided and key is not found, raises KeyError.
popitem()	Remove and return an arbitary item (key, value). Raises KeyError if the dictionary is empty.
setdefault(key[,d])	If key is in the dictionary, return its value. If not, insert key with a value of d and return d (defaults to None).
update([other])	Update the dictionary with the key/value pairs from other, overwriting existing keys.
values()	Return a new view of the dictionary's values

Built-in Functions with Dictionary	
Function	**Description**
all()	Return True if all keys of the dictionary are true (or if the dictionary is empty).
any()	Return True if any key of the dictionary is true. If the dictionary is empty, return False.
len()	Return the length (the number of items) in the dictionary.
sorted()	Return a new sorted list of keys in the dictionary.

Nested Dictionary

nested_dict = { 'dictA': {'key_1': 'value_1'}, 'dictB': {'key_2': 'value_2'}}

For Example: -

people = {1: {'name': 'John', 'age': '27', 'sex': 'Male'}, 2: {'name': 'Marie', 'age': '22', 'sex': 'Female'}}

print(people[1]['name'])

print(people[1]['age'])

print(people[1]['sex'])

Practice Programs

1. Write a Python Program to find Simple Interest. Use User Inputs without Hard Coding.

Hint: Simple Interest = (Principle Amount*Time*Rate)/100

2. Write a Python Program to check whether a number is Prime or Not.

Hint: Natural Number greater than 1 that has no positive divisor other than 1 and itself.

First Few Prime Numbers are: {2, 3, 5, 7, 11, 13, 17, 19,….}

3. Write a Python Function to rotate array of size ‘n’ by ‘d’ elements.

Hint: Input Array = [1, 2, 3, 4, 5, 6, 7]; Output Array = [3, 4, 5, 6, 7, 1, 2]

Practice Programs

4. Write a Python Program to find second largest number in a list.

Hint: Input: list1 = [10, 20, 4]; Output: 10

5. Write a Python Program to find Cumulative sum of a list

Hint: Input: list1 = [10, 20, 30, 40, 50]; Output: [0, 10, 30, 60, 100, 150]

6. Write a Python Program to accept the strings which contains all vowels.

Hint: Given a String, the task is to check and accept the given string if contains all vowels. i.e. ‘a’, ‘e’, ‘i’, ‘o’, ‘u’ or ‘A’, ‘E’, ‘I’, ‘O’, ‘U’.

Sample Input: ‘hello’; Output: Not Accepted

Sample Input: ‘ABeeIghiObhkUul’; Output: Not Accepted

Practice Programs

7. Write a Python Program to check if String contains any special character.

Hint: Sample Input: Hello$World; Output: Special Character Present; Sample Input: Hello World; Output: No Special Character Present.