# SET:

Set is another data structure supported by python. Basically, sets are same as lists but

with a difference that **sets are with no duplicate entries**. Technically a set is a mutable and an unordered collection of items. This means that we can easily add or remove items from it. **Creating a Set:**

A set is created by placing all the elements inside curly brackets {}. Separated by comma or by using the built-in function set( ).

## Syntax:

Set\_variable\_name={var1, var2, var3, var4, …….}

**Example:**

s={1, 2.5, ‚abc‛ }

print(s) # { 1, 2.5, ‚abc‛ }

## Converting a list into set:

A set can have any number of items and they may be of different data types. *set()*

function is used to converting list into set.

s=set( [ 1, 2.5, ‚abc‛ ] )

print(s) # { 1, 2.5, ‚abc‛ }

We can also convert tuple or string into set.

tup=(1,2,3,3,4,4,4,4,5,5,)

print(set(tup))

str="MOTHILAL"

print(set(str))

# {1, 2, 3, 4, 5}

# {'T', 'I', 'H', 'L', 'A', 'M', 'O'}

## Updating a set:

Since sets are unordered, indexing has no meaning. Set operations do not allow users to access or change an element using indexing or slicing.

## Methods on set:

1. **len(S):** Returns the number of elements in set S.

|  |  |
| --- | --- |
| **Program** | **Output** |
| A={1,2,3,4,5,6,7}  print(len(A)) | 7 |

1. **S1.issubset(S2):** Test whether every element in S1 is in S2.

|  |  |
| --- | --- |
| **Program** | **Output** |
| A={1,2,3,4} B={1,2,3,4,5,6,7}  print(A.issubset(B)) | True |

1. **S1.issuperset(S2):** Test whether every element in S2 is in S1.

|  |  |
| --- | --- |
| **Program** | **Output** |
| A={1,2,3,4} B={1,2,3,4,5,6,7}  print(A.issuperset(B)) | False |

1. **S1 == S2:** Returns True if both sets are equal.

|  |  |
| --- | --- |
| **Program** | **Output** |
| A={1,2,3,4} B={1,2,3,4,5,6,7}  print(A==B) | False |

1. **S1 != S2:** Returns True if both sets are not equal.

|  |  |
| --- | --- |
| **Program** | **Output** |
| A={1,2,3,4} B={1,2,3,4,5,6,7}  print(A!=B) | True |

1. **S1.union(S2):** Generate new sets with all elements from both S1 and S2.

|  |  |
| --- | --- |
| **Program** | **Output** |
| A={1,2,3,4,5,6,7}  B={4,5,6,7,8,9,10}  C=A.union(B)  print(C) | {1, 2, 3, 4, 5, 6, 7, 8,  9, 10} |

1. **S1.intersection(S2):** Generate new sets with common elements from both S1 and S2.

|  |  |
| --- | --- |
| **Program** | **Output** |
| A={1,2,3,4,5,6,7}  B={4,5,6,7,8,9,10}  C=A.intersection(B)  print(C) | {4, 5, 6, 7} |

1. **S1.difference(S2):** Generate new set with elements in S1 not in S2.

|  |  |
| --- | --- |
| **Program** | **Output** |
| A={1,2,3,4,5,6,7}  B={4,5,6,7,8,9,10}  C=A.difference(B)  print(C) | {1, 2, 3} |

1. **S1.symmetric\_difference(S2):** Generate new set with elements in S1 not in S2 and elements in S2 not in S1.

|  |  |
| --- | --- |
| **Program** | **Output** |
| A={1,2,3,4,5,6,7}  B={4,5,6,7,8,9,10}  C=A.symmetric\_difference(B)  print(C) | {1, 2, 3, 8, 9, 10} |

1. **S.copy():** Generate new shallow copy of set S.

|  |  |
| --- | --- |
| **Program** | **Output** |
| A={1,2,3,4,5,6,7}  B=A.copy()  print(B) | {1, 2, 3, 4, 5, 6, 7} |

1. **S.add(X):** Add element X to set S.

|  |  |
| --- | --- |
| **Program** | **Output** |
| A={1,2,3,4,5,6,7}  A.add(10)  print(A) | {1, 2, 3, 4, 5, 6, 7, 10} |

1. **S.discard(X):** Removes X from set S if X is present.

|  |  |
| --- | --- |
| **Program** | **Output** |
| A={1,2,3,4,5,6,7}  A.discard(5)  print(A) | {1, 2, 3, 4, 6, 7} |

1. **S.remove(X):** Removes X from set S if X is present; raises KeyError if not present.

|  |  |
| --- | --- |
| **Program** | **Output** |
| A={1,2,3,4,5,6,7}  A.remove(5)  print(A) | {1, 2, 3, 4, 6, 7} |

1. **S.pop():** Removes and arbitrary element (Randomly element) from S.

|  |  |
| --- | --- |
| **Program** | **Output** |
| A={10,15,12,21,3,4,5,6,7}  A.pop()  print(A) | {4, 5, 6, 7, 10, 12, 15,  21} |

1. **S.clear():** Removes all elements from set S.

|  |  |
| --- | --- |
| **Program** | **Output** |
| A={10,15,12,21,3,4,5,6,7}  A.clear()  print(A) | set() |

1. **max(S):** Returns maximum element from S.

|  |  |
| --- | --- |
| **Program** | **Output** |
| A={10,15,12,21,4,5,6,7}  print(max(A)) | 21 |

1. **min(S):** Returns minimum element from S.

|  |  |
| --- | --- |
| **Program** | **Output** |
| A={10,15,12,21,4,5,6,7}  print(min(A)) | 4 |

1. **sorted(S):** Returns a new sorted list from the elements in the set S.

|  |  |
| --- | --- |
| **Program** | **Output** |
| A={10,15,12,21,4,5,6,7}  B=sorted(A)  print(B) | [4, 5, 6, 7, 10, 12, 15,  21] |

# Dictionary:

A dictionary represents a group of elements arranged in the form of key-value pairs. The first element is considered as „key‟ and the immediate next element is taken as its

„value‟. The key and its value are separated by a colon (:). All the key-value pairs in a dictionary are inserted in curly braces { }.

d= { ‘Regd.No’: 556, ‘Name’:’Mothi’, ‘Branch’: ‘CSE’ }

Here, the name of dictionary is „dict‟. The first element in the dictionary is a string

„Regd.No‟. So, this is called „key‟. The second element is 556 which is taken as its „value‟.

## Example:

d= { ‘Regd.No’: 556, ‘Name’:’Mothi’, ‘Branch’: ‘CSE’ } print(d[‘Regd.No’]) # 556

print(d[‘Name’]) # Mothi

print(d[‘Branch’]) # CSE

To access the elements of a dictionary, **we should not use indexing or slicing**. For example, **dict[0] or dict[1:3] etc. expressions will give error.** To access the value associated with a key, we can mention the key name inside the square braces, as: dict[„Name‟].

If we want to know how many key-value pairs are there in a dictionary, we can use the len( ) function, as shown

d= { ‘Regd.No’: 556, ‘Name’:’Mothi’, ‘Branch’: ‘CSE’ } print(len(d)) # 3

We can also insert a new key-value pair into an existing dictionary. This is done by mentioning the key and assigning a value to it.

d={'Regd.No':556,'Name':'Mothi','Branch':'CSE'}

print(d) # {'Regd.No':556,'Name':'Mothi','Branch':'CSE', 'Gender':'Male'}

d['Gender']="Male"

print(d) # {'Regd.No':556,'Name':'Mothi','Branch':'CSE', 'Gender':'Male'}

Suppose, we want to delete a key-value pair from the dictionary, we can use *del* statement as:

del dict[‘Regd.No’] #{'Gender': 'Male', 'Branch': 'CSE', 'Name': 'Mothi'}

We can use any datatypes for value. For example, a value can be a number, string, list, tuple or another dictionary. But keys should obey the rules:

* Keys should be unique. It means, duplicate keys are not allowd. If we enter same key again, the old key will be overwritten and only the new key will be available.

emp={'nag':10,'vishnu':20,'nag':20} print emp # {'nag': 20, 'vishnu': 20}

* Keys should be immutable type. For example, we can use a number, string or tuples as keys since they are immutable. We cannot use lists or dictionaries as keys. If they are used as keys, we will get „TypeError‟.

emp={**['nag']:10**,'vishnu':20,'nag':20} Traceback (most recent call last): File "<pyshell#2>", line 1, in <module>

emp={['nag']:10,'vishnu':20,'nag':20}

**TypeError: unhashable type: 'list'**

## Dictionary Methods:

1. **D.clear():** Removes all key-value pairs from dictionary D

|  |  |
| --- | --- |
| **Program** | **Output** |
| D={"Regdno":556,"Name":"Mothi","Branch":"CSE"} D.clear()  print(D) | {} |

1. **D2=D1.copy():** Copies all elements from D1 into a new dictionary D2.

|  |  |
| --- | --- |
| **Program** | **Output** |
| D1={"Regdno":556,"Name":"Mothi","Branch":"CSE"} D2=D1.copy()  print(D2) | {'Regdno': 556, 'Name':  'Mothi', 'Branch': 'CSE'} |

1. **D.keys():** Returns a sequence of keys from the dictionary D.

|  |  |
| --- | --- |
| **Program** | **Output** |
| D={"Regdno":556,"Name":"Mothi","Branch":"CSE"} print(D.keys()) | dict\_keys(['Regdno', 'Name', 'Branch']) |

1. **D.values():** Returns a sequence of values from the dictionary D.

|  |  |
| --- | --- |
| **Program** | **Output** |
| D={"Regdno":556,"Name":"Mothi","Branch":"CSE"} print(D.values()) | dict\_values([556, 'Mothi', 'CSE']) |

1. **D1.update(D2):** Add all elements from dictionary D2 to D1.

|  |  |
| --- | --- |
| **Program** | **Output** |
| D1={"Regdno":556,"Name":"Mothi","Branch":"CSE"}  D2={"Sub1":56,"Sub2":65,"Sub3":74}  D1.update(D2)  print(D1) | {'Regdno': 556, 'Name':  'Mothi', 'Branch': 'CSE',  'Sub1': 56, 'Sub2': 65,  'Sub3': 74} |

1. **D.pop(Key):** Removes the key and its value from D.

|  |  |
| --- | --- |
| **Program** | **Output** |
| D={"Regdno":556,"Name":"Mothi","Branch":"CSE"} D.pop("Branch")  print(D) | {'Regdno': 556, 'Name': 'Mothi'} |

1. **D.setdefault(K,V):** If key K is found, its value is returned. If key K is not found, then the K,V pair is stored into the dictionary D.

|  |  |
| --- | --- |
| **Program** | **Output** |
| D={"Regdno":556,"Name":"Mothi","Branch":"CSE"} D.setdefault("Gender","Male")  print(D) | {'Regdno': 556, 'Name':  'Mothi', 'Branch': 'CSE',  'Gender': 'Male'} |

## Using for loop with Dictionaries:

*for* loop is very convenient to retrieve the elements of a dictionary. Let‟s take a simple dictionary that contains color code and its name as:

colors = { 'r':"RED", 'g':"GREEN", 'b':"BLUE", 'w':"WHITE" }

Here, „r‟, „g‟, „b‟ represents keys and „RED‟, „GREEN‟, „BLUE‟ and „WHITE‟ indicate values.

colors = { 'r':"RED", 'g':"GREEN", 'b':"BLUE", 'w':"WHITE" }

for k in colors:

print(k)

for k in colors:

print(colors[k])

# displays only keys

# keys to to dictionary and display the values

## Converting Lists into Dictionary:

When we have two lists, it is possible to convert them into a dictionary. For example, we have two lists containing names of countries and names of their capital cities.

There are two steps involved to convert the lists into a dictionary. The first step is to create a „zip‟ class object by passing the two lists to zip( ) function. The zip( ) function is useful to convert the sequences into a zip class object. The second step is to convert the zip object into a dictionary by using dict( ) function.

## Example:

**countries = [ 'USA', 'INDIA', 'GERMANY', 'FRANCE' ]**

**cities = [ 'Washington', 'New Delhi', 'Berlin', 'Paris' ] z=zip(countries, cities)**

**d=dict(z) print(d)**

**Output:**

**{'GERMANY': 'Berlin', 'INDIA': 'New Delhi', 'USA': 'Washington', 'FRANCE': 'Paris'}**

**Recursion in Python:**

The term Recursion can be defined as the process of defining something in terms of itself. In simple words, it is a process inwhich a function calls itself directly or indirectly.

Advantages of using recursion

A complicated function can be split down into smaller sub-problems utilizing recursion.

Sequence creation is simpler through recursion than utilizing any nested iteration.

Recursive functions render the code look simple and effective.

Disadvantages of using recursion

A lot of memory and time is taken through recursive calls which makes it expensive for use.

Recursive functions are challenging to debug.

The reasoning behind recursion can sometimes be tough to think through.

Syntax:

def func(): <--

|

| (recursive call)

|

func() ----

Example 1: A Fibonacci sequence is the integer sequence of 0, 1, 1, 2, 3, 5, 8….

**PROGRAM:**

Program to print the fibonacci series upto n\_terms

# Recursive function

**def** recursive\_fibonacci(n):

**if** n <**=** 1:

**return** n

**else**:

**return**(recursive\_fibonacci(n**-**1) **+** recursive\_fibonacci(n**-**2))

n\_terms **=** 10

# check if the number of terms is valid

**if** n\_terms <**=** 0:

  print("Invalid input ! Please input a positive value")

**else**:

  print("Fibonacci series:")

**for** i **in** range(n\_terms):

    print(recursive\_fibonacci(i))

FF

13RRTRTTRR

# File Operations:

* Files are named locations on disk to store related information. They are used to permanently store data in a non-volatile memory (e.g. hard disk).
* Since Random Access Memory (RAM) is volatile (which loses its data when the computer is turned off), we use files for future use of the data by permanently storing them.
* When we want to read from or write to a file, we need to open it first. When we are done, it needs to be closed so that the resources that are tied with the file are freed.

Hence, in Python, a file operation takes place in the following order:

1. Open a file
2. Read or write (perform operation)
3. Close the file

### Opening Files in Python

Python has a built-in *open()* function to open a file. This function returns a file object, also called a handle, as it is used to read or modify the file accordingly.

*F = open("myfile.txt")*

We can specify the mode while opening a file. In mode, we specify whether we want to read **r**, write **w** or append **a** to the file.

The default is reading in text mode. In this mode, we get strings when reading from the file.

|  |  |
| --- | --- |
| **Mode** | **Description** |
| r | Opens a file for reading. (default) |
| w | Opens a file for writing. Creates a new file if it does not exist or  truncates the file if it exists. |
| x | Opens a file for exclusive creation. If the file already exists, the  operation fails. |
| a | Opens a file for appending at the end of the file without truncating it.  Creates a new file if it does not exist. |
| b | Opens in binary mode. |
| + | Opens a file for updating (reading and writing) |

*F = open("myfile.txt",'r') # read a file F = open("myfile.txt",'w') # write a file*

*F = open("myfile.txt",'a') # appends a file*

### Closing Files in Python

When we are done with performing operations on the file, we need to properly close the file. Closing a file will free up the resources that were tied with the file. It is done using the close() method available in Python.

*F = open("test.txt", 'r') # perform file operations F.close()*

### Reading Files in python:

Assume we have the following file, located in the same folder as Python:

*Demo.txt*

WELCOME TO CSE. WELCOME TO ALL.

Good Luck!

To read a file in Python, we must open the file in reading **r** mode.

There are various methods available for this purpose. We can use the *read(size)* method to read in the **size** number of data. If the *size* parameter is not specified, it reads and returns up to the end of the file.

### Example-1:

*F = open("Demo.txt",'r') print(F.read(5)) F.close()*

### Output-1:

*WELCO*

### Example-2:

*F = open("Demo.txt",'r') print(F.read()) F.close()*

### Output-2:

*WELCOME TO CSE. WELCOME TO ALL.*

*Good Luck!*

### Example-3:

*F = open("Demo.txt",'r') print(F.readlines()) F.close()*

### Output-3:

*['WELCOME TO CSE.\n', 'WELCOME TO ALL.\n', 'Good Luck!']*

Here, *readlines()* method read all lines in the form of list.

### Writing to Files in Python:

* In order to write into a file in Python, we need to open it in write w, append a mode.
* If we open the file in write mode, then new file created and writes data on it.
* If file is already exits then all the data in that file will be erased and new content will be stored.
* If we open the file in append mode, then that file content will not be erased and new content will be added at the end.

### Example:

*F = open("Demo.txt",'w') F.write("welcome to all") F.close()*

**OUTPUT:**

Here output will stored in a **Demo.txt** file with the content of **welcome to all.**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*END\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*