

## Sequence objects

- collection of elements - str, range(), zip(), enumerate()
- Container sequences/Objects - list, tuple, dict, set

### Operations on Generic Sequences

- Membership - in | not in
- Iteration - for-loop`

### Operations on Ordered/Indexed Sequences

- Indexing - obj[index\_pos]
- Slicing - obj[start : stop]
- Concatenation - `+` operator
- Repeatition - `\*` operator

### Functions on Generic Sequences

- len() - gives the number of elements in the sequence
- max() - gives the largest element in the sequence
- min() - gives the smallest element in the sequence
- sum() - applicable to numeric sequences, returns the sum of all elements in the sequence
- math.prod() - applicable to numeric sequences, returns the product of all elements in the sequence
- sorted() - sorts the elements in the sequence in ASC order and returns a list object

## Strings in Python

### Strings are -

- an ordered sequence of characters
- enclosed in a pair of single quotes or pair of double quotes
- immutable

### Empty string

```
In [1]: string = ''  
  
string = ""
```


**Note - bool() of empty str is always False**

```
In [3]: strg = input("Enter string - ")
        if strg :
            print(strg)
        else:
            print("String is empty")
```

String is empty

## Defining a string

```
In [4]: string = "aero-plane"
```

No description has been provided for this image

### Extract first element from the string


```
In [5]: string[0]
```

Out[5]: 'a'

### Extract 5th element from the string

```
In [6]: string[4]
```

Out[6]: 'e'

No description has been provided for this image

**Ex. Extract last element from the string**

```
In [7]: string[-1]
```

```
Out[7]: 'e'
```



No description has been provided for this image



No description has been provided for this image

**Ex. Extract first 3 characters from string**

```
In [8]: string[0 : 3]
```

```
Out[8]: 'aer'
```

**Ex. Extract all characters from index position 3**

```
In [9]: string[3 : ]
```

```
Out[9]: 'o-plane'
```

**Ex. Extract last 4 characters from the string**

```
In [11]: string[-4 :]
```

```
Out[11]: 'lane'
```

#### Reverse of string

```
In [12]: string[::-1]
```

```
Out[12]: 'enalp-orea'
```

**Ex. WAP to check if entered string is a palindrome or not.**

**Palindrome** - the string reads same characters left to right or right to left

Ex - madam

```
In [17]: strg = input("Enter a string - ")
print("Palindrome" if strg == strg[::-1] else "Not Palindrome") # if-else statement
```

Palindrome

```
In [20]: strg = input("Enter a string - ")
result = 1 if strg == strg[::-1] else 0
if result :
    print("success")
else:
    print("Empty")
```

Empty

**Ex. WAP to generate a new string by swapping first and last characters**

ex - abcde - ebcda

```
In [24]: strg = input("Enter a string - ")
strg[-1] + strg[1 : -1] + strg[0]
```

```
Out[24]: 'hbcdefga'
```

## Operations on Strings

- Concatenation
- Repetition
- Membership
- Iteration

```
In [ ]:
```

**Concatenation** - merging two strings into a single object using the + operator.

```
In [25]: "abc" + "pqr"
```

```
Out[25]: 'abcpqr'
```

**Repetition** - The repetition operator `*` will make multiple copies of that particular object and combines them together.

```
In [26]: "abc" * 3
```

```
Out[26]: 'abcabcabc'
```

**Strings are immutable and cannot be modified**

```
In [28]: strg = "abcdef"
strg[2] = "x"
```

```
-----
TypeError                                Traceback (most recent call last)
Cell In[28], line 2
      1 strg = "abcdef"
----> 2 strg[2] = "x"

TypeError: 'str' object does not support item assignment
```

```
In [31]: strg = "abcdef"
strg1 = strg[0:2] + "x" + strg[3:]
strg1
```

```
Out[31]: 'abxdef'
```

## Built-in Functions

- **len()** - returns length of the string
- **min(), max()** - returns minimum and maximum element from the string
- **sorted()** - sorts the characters of the string and returns a list

```
In [32]: string = "Mississippi"
len(string)
```

```
Out[32]: 11
```

```
In [33]: min(string)
```

```
Out[33]: 'M'
```

```
In [34]: max(string)
```

```
Out[34]: 's'
```

```
In [35]: sorted(string) # returns a list object
```

```
Out[35]: ['M', 'i', 'i', 'i', 'i', 'p', 'p', 's', 's', 's', 's']
```

## Strings Methods

- **str.index( obj )** - returns index of the first occurrence of the character
- **str.count( obj )** - returns count of number of occurrences of the character
- **str.upper()** - returns string of uppercase characters
- **str.lower()** - returns string of lowercase characters
- **str.title()** - returns string of sentence case characters
- **str.isupper()** - checks if all characters are uppercase
- **str.islower()** - checks if all characters are lowercase
- **str.isdigit()** - checks if all characters are digits
- **str.isalpha()** - checks if all characters are alphabets
- **str.isalnum()** - checks if all characters are either alphabets or digits
- **str.split( delimiter )** - splits the string on the mentioned delimiter and returns a list of obtained parts of the string
- **str.replace( str , str )** - replaces all the mentioned characters with the specified string and returns a new string
- **str.strip( delimiter )** - removes whitespace characters from start and end of the string (delimiter can also be specified)
- **delimiter.join( sequence )** - it is called on a string object which acts as a delimiter to join all string elements in the sequence passed as an argument to join()

```
In [37]: string
```

```
Out[37]: 'Mississippi'
```

```
In [38]: string.index("i")
```

```
Out[38]: 1
```

```
In [39]: string.count("i")
```

```
Out[39]: 4
```

```
In [40]: string.upper()
```

```
Out[40]: 'MISSISSIPPI'
```

```
In [41]: string.isalpha()
```

```
Out[41]: True
```

```
In [44]: strg = "have a nice day"  
strg.split()
```

```
Out[44]: ['have', 'a', 'nice', 'day']
```

```
In [45]: strg.replace("a", "*")
```

```
Out[45]: 'h*ve * nice d*y'
```

```
In [46]: strg.strip()
```

```
Out[46]: 'have a nice day'
```

## Examples

**Ex. WAP to convert the given string -**

**string = "I am in Python class"**

**o/p** - 'ssalc nohtyP ni ma I'

**o/p** - 'I Am In Python Class'

HINT - upper(), lower(), swapcase(), title(), capitalize()

```
In [47]: string = "I am in Python class"  
print(string[::-1])  
print(string.title())
```

```
ssalc nohtyP ni ma I
```

```
I Am In Python Class
```

**Ex. WAP to replace all vowels in a word with and asterisk.**

```
In [54]: strg = input("Enter a word - ").lower()  
for vowel in "aeiou":  
    strg = strg.replace(vowel, "*")  
strg
```

```
Out[54]: 's*ng*p*r*'
```

```
In [56]: trans_obj = str.maketrans("aeiou", "*****")  
strg.translate(trans_obj)
```

```
Out[56]: 's*ng*p*r*'
```

```
In [60]: profits = "$200"  
profits.replace("$", "")  
profits.strip("$")
```

Out[60]: '200'

```
In [64]: profits = "($1,200)"
trans_obj = str.maketrans("(", "-", "$,")
int(profits.translate(trans_obj))
```

Out[64]: -1200

**Ex. WAP to accept numbers from user in comma seperated format. Extract the integers and perform their summation**

```
In [67]: numbers = input("Enter numbers in comma seperated format - ")
num_list = numbers.split(",")
total = 0
for number in num_list :
    if number.isdigit() :
        total += int(number)
total
```

Out[67]: 100

**Ex. WAP to print foloowing pattern**

```
In [ ]: *
        **
        ***
        ****
        *****
```

**Ex. Write a Python program to get a string from a given string where all occurrences of its first char have been changed to '#', except the first char itself.**

Sample String : 'restart'

Expected Result : 'resta#t'

In [ ]:

---

---

## Python Sequences and Containers

Object	Container Object	Sequence Type	Element Type	Enclosed in	Immutability	Duplicates
str()	No	ordered/indexed	characters	"" or "	Yes	Yes
tuple()	Yes	ordered/indexed	mixed data (heterogeneous)	()	Yes	Yes
list()	Yes	ordered/indexed	mixed data (heterogeneous)	[]	No	Yes



Object	Container Object	Sequence Type	Element Type	Enclosed in	Immutability	Duplicates
set()	Yes	unordered	heterogeneous (immutable objects)	{}	No	No
dict()	Yes	unordered	Key - immutable Value - any type	{}	No	Key - No Value - Yes

## Tuples in Python

### Tuples -

- are Python containers
- are an ordered sequence of mixed data
- enclose elements in a pair of round brackets, separated by commas
- are immutable

### Defining a tuple

```
In [2]: tup = (1, 2, 3, 4)
tup
```

```
Out[2]: (1, 2, 3, 4)
```

### Empty Tuple

```
In [ ]: tup = ()
tup = tuple()
```

**Note - bool() of empty tuple is always False**

### Single value Tuple

```
In [4]: (10,)
```

```
Out[4]: (10,)
```

## Operations on Tuples

- Iteration
- Membership
- Indexing

- Slicing
- Concatenation
- Repetition

## Indexing and Slicing

```
In [5]: fam = ("Rosie", 34, "Sam", 36, "Jonah", 15, "Jessie", 12)
```

**Ex. Extract 1st element from tuple -**

```
In [6]: fam[0]
```

```
Out[6]: 'Rosie'
```

**Ex. Extract last element from tuple -**

```
In [7]: fam[-1]
```

```
Out[7]: 12
```

**Ex. Extract first 3 elements from tuple -**

```
In [8]: fam[0 : 3]
```

```
Out[8]: ('Rosie', 34, 'Sam')
```

**Ex. Extract last 2 element from tuple -**

```
In [9]: fam[-2 : ]
```

```
Out[9]: ('Jessie', 12)
```

**Ex. Extract "True" from the given Tuple**

```
In [10]: mix_tuple = (('a', 1, True), 234567, 'Science', -5)
mix_tuple[0][2]
```

```
Out[10]: True
```

```
In [11]: mix_tuple = (('a', 1, True), 234567, 'Science', -5)
mix_tuple[1][2]
```

```
-----
TypeError                                Traceback (most recent call last)
Cell In[11], line 2
      1 mix_tuple = (('a', 1, True), 234567, 'Science', -5)
----> 2 mix_tuple[1][2]

TypeError: 'int' object is not subscriptable
```

```
In [12]: mix_tuple = (('a', 1, True), 234567, 'Science', -5)
mix_tuple[2][2]
```

```
Out[12]: 'i'
```

```
In [ ]:
```

**Concatenation** - merging two tuples into a single tuple object using the **+** operator.

```
In [13]: (1, 2, 3) + (4, 5, 6)
```

```
Out[13]: (1, 2, 3, 4, 5, 6)
```

**Repetition** - The repetition operator **\*** will make multiple copies of that particular object and combines them together.

```
In [14]: (1, 2, 3) * 3
```

```
Out[14]: (1, 2, 3, 1, 2, 3, 1, 2, 3)
```

```
In [16]: tup = (1, 2, 3, 4)
tup[0] = 10
```

```
-----
TypeError                                Traceback (most recent call last)
Cell In[16], line 2
      1 tup = (1, 2, 3, 4)
----> 2 tup[0] = 10

TypeError: 'tuple' object does not support item assignment
```

**Tuples are immutable hence cannot be modified**

## Built-in functions on Tuples

- **len()** - returns length of the tuple
- **min(), max()** - returns minimum and maximum element from the tuple
- **sorted()** - sorts the elements of the tuple and returns a list
- **sum()** - applicable to only numeric tuples, returns summation of all the elements in the tuple

```
In [17]: tup = (1, 5, 3, 7, 2, 4)
```

```
In [18]: len(tup)
```

```
Out[18]: 6
```

```
In [19]: min(tup)
```

```
Out[19]: 1
```

```
In [20]: max(tup)
```

```
Out[20]: 7
```

```
In [21]: sorted(tup)
```

```
Out[21]: [1, 2, 3, 4, 5, 7]
```

```
In [22]: sum(tup)
```

```
Out[22]: 22
```

**Ex. WAP to reverse the tuple**

```
In [23]: tup = (1, 4, 3, 2, 6, 5, 8)
          tup[::-1]
```

```
Out[23]: (8, 5, 6, 2, 3, 4, 1)
```

## Tuple Methods

- **tup.index( object )** - returns the index position of first occurrence of the **object**
- **tup.count( object )** - returns the **count** of number of times the object is repeated in the tuple

```
In [24]: tup = ('car', 'bike', 'house', 'car', 'aeroplane', 'train', 'car')
```

```
In [25]: tup.index('car')
```

```
Out[25]: 0
```

```
In [26]: tup.count('car')
```

```
Out[26]: 3
```

## Unpacking Tuples

```
In [27]: tup = 1, 2, 3 # Packing of tuples
          tup
```

```
Out[27]: (1, 2, 3)
```

```
In [28]: a, b, c = tup # unpacking
          print(a, b, c)
```

```
1 2 3
```

**defining multiple variable in single line**

```
In [29]: name, age = "Jane", 25 # packing - unpacking of tuples

print(name, age)
```

Jane 25

## Examples

**Ex. WAP to print the word in the tuple and its len.**

```
In [30]: tup = ('car', 'bike', 'house', 'aeroplane')
for i in tup:
    print(i, " - ", len(i))
```

```
car - 3
bike - 4
house - 5
aeroplane - 9
```

**Ex. WAP to print the tuple and the sum of sub tuples**

```
In [31]: tup = ((3,9),(1,2,4),(1,4,8),(2,3))
for i in tup:
    print(i, " - ", sum(i))
```

```
(3, 9) - 12
(1, 2, 4) - 7
(1, 4, 8) - 13
(2, 3) - 5
```

**Ex. WAP to print sum of all the numbers in above tuple.**

```
In [ ]:
```

---

---

## Lists in Python

### Lists -

- are Python containers
- are an ordered sequence of mixed data
- enclose elements in a pair of square brackets, separated by commas
- mutable

### Empty List

```
In [ ]: lst = []

lst = list()
```

**Note - bool() of empty list is always False**

## Create a List

```
In [32]: friends = [ 'Ross', 'Monica', 'Joey', 'Chandler' ]
```

## Retrive elements from List

### Indexing and slicing

**Ex. Extract first element from the list**

```
In [33]: friends[0]
```

```
Out[33]: 'Ross'
```

**Ex. Extract second last element from the list**

```
In [34]: friends[-2]
```

```
Out[34]: 'Joey'
```

**Ex. Extract first 3 elements from the list**

```
In [35]: friends[0:3]
```

```
Out[35]: ['Ross', 'Monica', 'Joey']
```

## Add elements to List

**lst.append( object )** - appends element to the end of the list

**Ex. Append 'Phoebe' to the end of the list**

```
In [36]: friends.append("Phoebe")  
print(friends)
```

```
['Ross', 'Monica', 'Joey', 'Chandler', 'Phoebe']
```

**lst.insert( object )** - inserts element at the mentioned index location

**Ex. Insert 'Rachel' at index position 4**

```
In [37]: friends.insert(4, "Rachel")  
print(friends)
```

```
['Ross', 'Monica', 'Joey', 'Chandler', 'Rachel', 'Phoebe']
```

**lst.extend( seq-obj )** - always take a sequence as parameter, appends all the elements from sequence to end of the list.

**Ex. Extend tuple of new\_friends to the end of the friends list**

```
In [38]: new_friends = ("Jack", "George", "Rosie", "Jasmine")
friends.extend(new_friends)
print(friends)
```

```
['Ross', 'Monica', 'Joey', 'Chandler', 'Rachel', 'Phoebe', 'Jack', 'George', 'Rosie', 'Jasmine']
```

## Modify elements in List

**Ex. Replace 'Rosie' with 'Amy'**

```
In [39]: idx = friends.index("Rosie")
friends[idx] = "Amy"
print(friends)
```

```
['Ross', 'Monica', 'Joey', 'Chandler', 'Rachel', 'Phoebe', 'Jack', 'George', 'Amy', 'Jasmine']
```

## Remove elements from a list

**lst.pop()** - deletes the last element from the list

```
In [40]: friends.pop()
print(friends)
```

```
['Ross', 'Monica', 'Joey', 'Chandler', 'Rachel', 'Phoebe', 'Jack', 'George', 'Amy']
```

**lst.pop( index-value )** - deletes the element at the mentioned **index-value** from the list

```
In [41]: friends.pop(2)
print(friends)
```

```
['Ross', 'Monica', 'Chandler', 'Rachel', 'Phoebe', 'Jack', 'George', 'Amy']
```

**lst.remove( obj )** - removes the mention **obj** from the list

```
In [42]: friends.remove("Amy")
print(friends)
```

```
['Ross', 'Monica', 'Chandler', 'Rachel', 'Phoebe', 'Jack', 'George']
```

## Using slicing operator to remove multiple element from a list

```
In [43]: del friends[0:3]
print(friends)
```

```
['Rachel', 'Phoebe', 'Jack', 'George']
```

## Operations on Lists

- Iteration

- Membership
- Concatenation
- Repetition

```
In [46]: list_1 = [1, 2, 3, 4]
```

## Conacatenation

```
In [47]: list_1 + [5, 6, 7]
```

```
Out[47]: [1, 2, 3, 4, 5, 6, 7]
```

## Repetition

```
In [48]: list_1 * 3
```

```
Out[48]: [1, 2, 3, 4, 1, 2, 3, 4, 1, 2, 3, 4]
```

## Built-in functions on Lists

- **len()** - returns length of the list
- **min(), max()** - returns minimum and maximum element from the list
- **sorted()** - sorts the elements of the list and returns a list
- **sum()** - applicable to only numeric lists, returns summation of all the elements int the list

```
In [49]: numbers = [3, 4, 2, 6, 5, 1]
```

```
In [50]: len(numbers)
```

```
Out[50]: 6
```

```
In [51]: min(numbers)
```

```
Out[51]: 1
```

```
In [52]: max(numbers)
```

```
Out[52]: 6
```

```
In [53]: sorted(list_1)
```

```
Out[53]: [1, 2, 3, 4]
```

```
In [54]: sum(numbers)
```

```
Out[54]: 21
```

**Ex. WAP to create a list of sales of 5 products and calculate average sales**



```
In [56]: sales = []
        for i in range(5):
            sales.append(int(input("Enter Sales - ")))
        sum(sales)/len(sales)
```

Out[56]: 600.0

## List Methods

**lst.index( object )** - returns the index position of first occurrence of the **object**

```
In [57]: friends = ['Ross', 'Monica', 'Rachel', 'Joey', 'Phoebe']
        friends.index("Joey")
```

Out[57]: 3

**lst.count( object )** - returns the **count** of number of times the object is repeated in the list

```
In [58]: friends.count("Joey")
```

Out[58]: 1

**lst.sort()** - sorts the list object

```
In [59]: friends = ['Ross', 'Monica', 'Rachel', 'Joey', 'Phoebe']
        friends.sort()
        print(friends)
```

['Joey', 'Monica', 'Phoebe', 'Rachel', 'Ross']

**lst.reverse()** - reverses the sequence of elements in the list

```
In [60]: friends = ['Ross', 'Monica', 'Rachel', 'Joey', 'Phoebe']
        friends.reverse()
        print(friends)
```

['Phoebe', 'Joey', 'Rachel', 'Monica', 'Ross']

## Creating a copy of a list

### (Shallow copying and Deep Copying)

In Python, Assignment statements do not copy objects, they create bindings between a target and an object. When we use = operator user thinks that this creates a new object; well, it doesn't. It only creates a new variable that shares the reference of the original object. Sometimes a user wants to work with mutable objects, in order to do that user looks for a way to create "real copies" or "clones" of these objects. Or, sometimes a user wants copies

that user can modify without automatically modifying the original at the same time, in order to do that we create copies of objects.

A copy is sometimes needed so one can change one copy without changing the other. In Python, there are two ways to create copies :

- Deep copy
- Shallow copy

## Shallow Copy

```
In [63]: lst1 = [2, 8, ["Jane", "Thomas", "Jack"]]
         lst2 = lst1.copy()
         lst1[0] = 10
         print(lst1)
         print(lst2)
```

```
[10, 8, ['Jane', 'Thomas', 'Jack']]
[2, 8, ['Jane', 'Thomas', 'Jack']]
```

## Deep Copy

```
In [65]: import copy
         lst1 = [2, 8, ["Jane", "Thomas", "Jack"]]
         lst2 = copy.deepcopy(lst1)
         lst1[2][0] = "Amy"
         print(lst1)
         print(lst2)
```

```
[2, 8, ['Amy', 'Thomas', 'Jack']]
[2, 8, ['Jane', 'Thomas', 'Jack']]
```

## Conclusion -

1. In case of nested list - create a deep copy
2. Avoid use of mutable nested containers (nested list or nested dict) in case of taking backups creating copies
3. Deep copy should avoided for better space complicity
4. Use tuples inside list to avoid creating deep copy

```
In [ ]: conn.execute("Select * from table")
        resultset = sequence of tuples - each tuple represent a row in the table
```

```
In [ ]:
```

## Utility Functions

**zip( lst1, lst2, ... )** - returns a zip object, which is an sequence of tuples where the first item in each passed sequence is paired together, and then the second item in each passed

sequence are paired together etc. If the passed sequences have different lengths, the iterator with the least items decides the length of the new iterator.

**Ex. Write the program to multiply the two lists**

```
In [66]: num_list_1 = [1, 2, 3, 4]
        num_list_2 = [0, 5, 2, 1]
```

```
In [67]: zip(num_list_1, num_list_2)
```

```
Out[67]: <zip at 0x22d3aa053c0>
```

```
In [68]: list(zip(num_list_1, num_list_2)) # convert a non readable sequence to a list or tu
```

```
Out[68]: [(1, 0), (2, 5), (3, 2), (4, 1)]
```

```
In [69]: for i in zip(num_list_1, num_list_2) :
        print(i)
```

```
(1, 0)
(2, 5)
(3, 2)
(4, 1)
```

```
In [70]: for i in zip(num_list_1, num_list_2) :
        print(i[0] * i[1])
```

```
0
10
6
4
```

```
In [71]: for i, j in zip(num_list_1, num_list_2) :
        print(i * j)
```

```
0
10
6
4
```

**enumerate( seq-obj , start=0 )** - adds counter to an iterable and returns a sequence of tuples (the enumerate object).

```
In [72]: lst = [100, 200, 250, 400]
        enumerate(lst)
```

```
Out[72]: <enumerate at 0x22d3b37d350>
```

```
In [73]: lst = [100, 200, 250, 400]
        list(enumerate(lst))
```

```
Out[73]: [(0, 100), (1, 200), (2, 250), (3, 400)]
```

```
In [74]: lst = [100, 200, 250, 400]
list(enumerate(lst, start = 101))
```

```
Out[74]: [(101, 100), (102, 200), (103, 250), (104, 400)]
```

**Ex. WAP to calculate percentage of each student and print in tabular format assuming student ID as 101, 102, 103**

```
In [76]: marks = [(50, 60, 75), (80, 76, 98), (55, 74, 67), (74, 83, 92), (85, 45, 97)]
percentage = []

for i in marks :
    percentage.append(round(sum(i)/3, 2))
percentage
```

```
Out[76]: [61.67, 84.67, 65.33, 83.0, 75.67]
```

```
In [81]: print("-"*20)
print("SID\tPercentage")
print("-"*20)

for sid, percent in enumerate(percentage, start = 101) :
    print(f"{sid}\t{percent}%")
```

```
-----
SID      Percentage
-----
101      61.67%
102      84.67%
103      65.33%
104      83.0%
105      75.67%
```

---

## Sets in Python

### Sets -

- are Python containers
- are an unordered sequence of mixed data (immutable objects)
- encloses elements in a pair of curly brackets, separated by commas
- mutable
- do not allow duplicates
- allow set operations on data

### Creating a set

```
In [82]: s = {10, 20, 30, 40, 50}
print(s)
```

```
{50, 20, 40, 10, 30}
```

## Empty Set

```
In [83]: set()
```

```
Out[83]: set()
```

**Note - bool() of empty set is always False**

## Add elements to Set

### set.add( obj )

- adds a new element to the set

```
In [84]: s.add("abcd")  
print(s)
```

```
{50, 'abcd', 20, 40, 10, 30}
```

### set.update( seq )

- takes a sequence object as a parameter and adds all the elements from the sequence to the set

```
In [85]: s.update([1, 2, 3, 4])  
print(s)
```

```
{1, 2, 3, 4, 10, 'abcd', 20, 30, 40, 50}
```

## Remove element from sets

### pop()

- removes a random element from the set

```
In [86]: s.pop()  
print(s)
```

```
{2, 3, 4, 10, 'abcd', 20, 30, 40, 50}
```

### remove( obj )

- removes a specified element from the set, gives error if the element is not present in the set

```
In [87]: s.remove("abcd")  
print(s)
```

```
{2, 3, 4, 10, 20, 30, 40, 50}
```

```
In [88]: s.remove("abcd")
         print(s)
```

```
-----
KeyError                                Traceback (most recent call last)
Cell In[88], line 1
----> 1 s.remove("abcd")
      2 print(s)

KeyError: 'abcd'
```

### `discard( obj )`

- removes the specified element from the set, it will not give any error if element is not present.

```
In [90]: print(s.discard("abcd"))
         print(s)
```

None

```
{2, 3, 4, 10, 20, 30, 40, 50}
```

## Built-in functions on Sets

- **len()** - returns length of the sets
- **min(), max()** - returns minimum and maximum element from the set
- **sorted()** - sorts the elements of the set and returns a list
- **sum()** - applicable to only numeric sets, returns summation of all the elements in the set

```
In [91]: set_a = {10, 20, 30, 40, 50, 20}
```

```
In [92]: len(set_a)
```

Out[92]: 5

```
In [93]: min(set_a)
```

Out[93]: 10

```
In [94]: sum(set_a)
```

Out[94]: 150

```
In [95]: sorted(set_a)
```

Out[95]: [10, 20, 30, 40, 50]

## Operations on Sets

- Iteration
- Membership
- Set Operations
  - Union | Intersection | Difference | Symmetric Difference
  - Disjoint sets
  - Subsets and Supersets

## Union | Intersection | Difference | Symmetric Difference

```
In [96]: set1 = {1, 2, 3, 4, 5}
        set2 = {4, 5, 6, 7, 8}
```

```
In [97]: set1 | set2 # union
        set1.union(set2)
```

```
Out[97]: {1, 2, 3, 4, 5, 6, 7, 8}
```

```
In [98]: set1 & set2 # intersection
        set1.intersection(set2)
```

```
Out[98]: {4, 5}
```

```
In [99]: set1 ^ set2
        set1.symmetric_difference(set2)
```

```
Out[99]: {1, 2, 3, 6, 7, 8}
```

```
In [100]: set1 - set2
        set1.difference(set2)
```

```
Out[100]: {1, 2, 3}
```

## Disjoint set

- if the two sets have no common elements

```
In [101]: set1 = {1, 2, 3, 4, 5}
        set2 = {6, 7, 8, 9, 10}

        set1.isdisjoint(set2)
```

```
Out[101]: True
```

## Subset | Superset

- If all elements of set1 are present in set2 then,
  - set1 is subset of set2
  - set2 will be superset of set1

```
In [102... set1 = {1, 2, 3, 4, 5}
set2 = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}

set1.issubset(set2)
```

Out[102... True

```
In [103... set2.issuperset(set1)
```

Out[103... True

## Examples -

### Set of members drinking tea and coffee

```
In [104... drinks_coffee = {"Jane", "Jack", "Sam", "George", "Dori"}

drinks_tea = {"Jack", "Frank", "Cody", "Dori", "Bill"}
```

**Ex. Are there any members who drink tea and coffee both? (Yes/No). Display their names**

```
In [105... if drinks_coffee.isdisjoint(drinks_tea) :
    print("No common members")
else:
    print("Common members - ", drinks_coffee & drinks_tea)
```

Common members - {'Dori', 'Jack'}

**Ex. Does all the members who drink tea also drink coffee? (Yes/No). If No, then display names of members who drink only tea.**

```
In [106... if drinks_coffee.issubset(drinks_tea) :
    print("Yes")
else:
    print("Only Tea members - ", drinks_tea - drinks_coffee)
```

Only Tea members - {'Cody', 'Frank', 'Bill'}

**Ex. WAP to check if password entered by user is satisfying following condions or not -**

1. length of password must be  $\geq 8$
2. username and password must not be same
3. Password must contain atleast 1 digit, 1 capital alphabet, 1 small alphabet and 1 special character like "!@#\$\$%^&\*"

```
In [116... import string
username = "Jane"
password = "Jane@1234"

result = []
result.append(len(password) >= 8)
result.append(username != password)
result.append(set(password) & set(string.ascii_uppercase))
result.append(set(password) & set(string.ascii_lowercase))
```



```
result.append(set(password) & set(string.digits))
result.append(set(password) & set("!@#$%^&*"))
if all(result) :
    print("Valid password")
else:
    print("Invalid password")
```

Valid password

```
In [109... import string
string.ascii_lowercase
```

```
Out[109... 'abcdefghijklmnopqrstuvwxyz'
```

- **all()** - returns True if all elements in the container sequence are True else False.
- **any()** - returns True if any one elements in the container sequence is True else False.

---

---

## Dictionary in Python

Dictionaries are -

- are Python containers
- are sequence of mixed data (no indexing)
- Encloses elements in a pair of curly brackets
- elements are stored in the form of {key : value} pairs separated by commas
- keys are always unique and immutable
- values need not be unique and can be of any type
- mutable

### Empty Dictionary

```
In [117... d = {}

d = dict()
```

**Note - bool() of empty dict is always False**

## Creating a dictionary

**Ex. Create a dictionary consisting of country names and their currencies**

```
In [119... countries = {"India" : "INR", "USA": "USD"}
print(countries)
```

```
{'India': 'INR', 'USA': 'USD'}
```

## Retriving elements from a Dictionary

Ex. Print currency for "India"

```
In [120...] countries["India"]
```

```
Out[120...] 'INR'
```

Ex. Print currency for "Japan"

```
In [121...] countries["Japan"]
```

```
-----  
KeyError                                Traceback (most recent call last)  
Cell In[121], line 1  
----> 1 countries["Japan"]  
  
KeyError: 'Japan'
```

**dict.get()** - returns the value of the item with the specified key

dictionary.get(keyname, value)

- keyname - Required. The keyname of the item you want to return the value from
- value - Optional. A value to return if the specified key does not exist. Default value None

```
In [122...] countries.get("India")
```

```
Out[122...] 'INR'
```

```
In [123...] print(countries.get("Japan"))
```

None

```
In [124...] print(countries.get("Japan", "country not present"))
```

country not present

## Adding new element to dictionary

Ex. Add Japan and its currency to dictionary

```
In [125...] countries["Japan"] = "Yen"  
print(countries)
```

```
{'India': 'INR', 'USA': 'USD', 'Japan': 'Yen'}
```

## Modifying dictionary

Ex. Modify the currency for USA as "\$"

```
In [126... countries["USA"] = "$"  
print(countries)
```

```
{'India': 'INR', 'USA': '$', 'Japan': 'Yen'}
```

## Updating a dictionary

**dict.update( new\_dict )** - inserts the specified items to the dictionary

**Ex. Add contents from new\_country dictionary to countries**

```
In [128... new_dict = {"Indonesia" : "IDR", "Singapore" : "SGD", "Thailand" : "Bhat"}  
countries.update(new_dict)  
print(countries)
```

```
{'India': 'INR', 'USA': '$', 'Japan': 'Yen', 'Indonesia': 'IDR', 'Singapore': 'SGD',  
'Thailand': 'Bhat'}
```

## Remove element from dictionary

**dict.pop( key )**

- removes the specified key and its value from the dictionary

```
In [129... countries.pop("USA")  
print(countries)
```

```
{'India': 'INR', 'Japan': 'Yen', 'Indonesia': 'IDR', 'Singapore': 'SGD', 'Thailand':  
'Bhat'}
```

**dict.popitem()**

- randomly removes a key-value pair from dictionary

```
In [130... countries.popitem()  
print(countries)
```

```
{'India': 'INR', 'Japan': 'Yen', 'Indonesia': 'IDR', 'Singapore': 'SGD'}
```

**dict.clear()**

- removes all the pairs from the dictionary

```
In [ ]: countries.clear()
```

**Ex. Write a program to retrieve the capital of 'Germany' from a given dictionary**

```
In [132... europe = { 'Spain': { 'Capital': 'Madrid', 'Population': 4.77 },  
            'France': { 'Capital': 'Paris', 'Population': 6.7 },  
            'Germany': { 'Capital': 'Berlin', 'Population': 8.28 },
```

```
        'Norway': { 'Capital':'Oslo', 'Population':0.533 } }
europe["Germany"]["Capital"]
```

Out[132...] 'Berlin'

## Functions/Operations on Dictionary

- Iteration - for loop, dict.keys(), dict.values(), dict.items()
- Membership
- len(), sorted()

```
In [134...] employees = { 'Jane': 70000, 'Rosie': 90000, 'Mary': 40000, 'Sam': 55000, 'George':
```

```
In [135...] employees.keys()
```

Out[135...] dict\_keys(['Jane', 'Rosie', 'Mary', 'Sam', 'George'])

```
In [136...] employees.values()
```

Out[136...] dict\_values([70000, 90000, 40000, 55000, 76000])

```
In [138...] for i in employees :
              print(i, " - ", employees[i])
```

```
Jane - 70000
Rosie - 90000
Mary - 40000
Sam - 55000
George - 76000
```

```
In [139...] employees.items()
```

Out[139...] dict\_items([('Jane', 70000), ('Rosie', 90000), ('Mary', 40000), ('Sam', 55000), ('George', 76000)])

```
In [140...] for key, value in employees.items() :
              print(key, " - ", value)
```

```
Jane - 70000
Rosie - 90000
Mary - 40000
Sam - 55000
George - 76000
```

## Creating dictionaries using sequences

**Ex. WAP to create a dictionary where keys are employee codes starting from 101 and its values are the employee names**

```
In [143...] names = ['Jane', 'Rosie', 'Mary', 'Sam', 'George']
dict(enumerate(names, start = 101))
```

Out[143...] {101: 'Jane', 102: 'Rosie', 103: 'Mary', 104: 'Sam', 105: 'George'}

**Ex. WAP to create a dictionary combining the following two lists where name is key and marks as value**

```
In [144... names = ['Jane', 'Rosie', 'Mary', 'Sam', 'George']
salary = [70000, 90000, 40000, 55000, 76000]
dict(zip(names, salary))
```

```
Out[144... {'Jane': 70000, 'Rosie': 90000, 'Mary': 40000, 'Sam': 55000, 'George': 76000}
```

**Ex. WAP to create a dict of employes with emp\_id as keys starting from 101 and a tuple of names and their ages.**

```
In [145... names = ['Jane', 'Rosie', 'Mary', 'Sam', 'George']
ages = [40, 30, 44, 25, 56]
dict(enumerate(zip(names, ages), start = 101))
```

```
Out[145... {101: ('Jane', 40),
102: ('Rosie', 30),
103: ('Mary', 44),
104: ('Sam', 25),
105: ('George', 56)}
```

---

---

## Comprehensions in Python

**Ex. WAP to print product of squares of all even numbers in the given list**

```
In [146... numbers = [1, 2, 4, 3, 5, 7, 6]
product = 1

for i in numbers :
    if i % 2 == 0 :
        product *= (i**2)
product # o/p is a single int object (non-sequence)

# This is an example of reduce scenario
```

```
Out[146... 2304
```

**Ex. WAP to print create a list of squares of all even numbers in the given list**

```
In [147... numbers = [1, 2, 4, 3, 5, 7, 6]
squares = []

for i in numbers :
    if i % 2 == 0 :
        squares.append(i**2)
squares

# This is an example of comprehension scenario
```

```
Out[147... [4, 16, 36]
```

**Comprehensions** are an elegant way to define and create mutable data structures like lists, sets, dictionary based on existing sequences Syntax –

```
[<expression> for <var> in <sequence> if <condition>]
```

1. Identify the sequence
2. Identify condition if any
3. Expression
4. Mutable datastructure

**Ex. WAP to generate a list of squares of number in range of 1-10**

```
In [148... [i**2 for i in range(1, 11)]
```

```
Out[148... [1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

**Ex. WAP to create a list of squares of even number in range of 1-10**

```
In [149... [i**2 for i in range(1, 11) if i % 2 == 0 ]
```

```
Out[149... [4, 16, 36, 64, 100]
```

**Ex. WAP to create a dict of number from 1-10 as keys and their squares as values**

```
In [150... {i : i**2 for i in range(1, 11)}
```

```
Out[150... {1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49, 8: 64, 9: 81, 10: 100}
```

**Ex. WAP to generate a dictionary specifying the elements in lst as even or odd.**

```
In [151... lst = [2, 1, 4, 3, 5, 7, 9]
{ i : "even" if i % 2 == 0 else "odd" for i in lst}
```

```
Out[151... {2: 'even', 1: 'odd', 4: 'even', 3: 'odd', 5: 'odd', 7: 'odd', 9: 'odd'}
```

**Ex. WAP to add 7% service tax to all the values in the "sales" list**

```
In [153... sales = [290, 500, 800, 650]
[i*1.07 for i in sales]
```

```
Out[153... [310.3, 535.0, 856.0, 695.5]
```

**Ex. WAP to sum all the values in the "sales" tuple**

```
In [155... sales = ("290", "500", "800", "650")
sum([int(i.replace("$", "")) for i in sales])
```

```
Out[155... 2240
```

**Ex. WAP to extract all the digits from the string using list comprehension**

```
In [156... string = "Current assets is worth 2450000 and current liabilities stands at 1230000  
[int(ch) for ch in string if ch.isdigit()]
```

```
Out[156... [2, 4, 5, 0, 0, 0, 0, 1, 2, 3, 0, 0, 0, 0]
```

**Ex. Extract all the numbers from the string using list comprehension**

```
In [158... string = "Current assets is worth 2450000 and current liabilities stands at 1230000  
[int(ch) for ch in string.split() if ch.isdigit()]
```

```
Out[158... [2450000, 1230000]
```

```
In [160... print(string.split())
```

```
['Current', 'assets', 'is', 'worth', '2450000', 'and', 'current', 'liabilities', 'st  
ands', 'at', '1230000']
```

**Ex. WAP to create a dict of names and the total marks(percentage) of each student.**

```
In [163... names = ['Jane', 'Rosie', 'Mary', 'Sam', 'George']  
marks = ([70, 65, 32], [90, 76, 98], [40, 55, 78], [50, 87, 67], [76, 72, 89])  
  
dict(zip(names, [round(sum(i)/3, 2) for i in marks]))
```

```
Out[163... {'Jane': 55.67, 'Rosie': 88.0, 'Mary': 57.67, 'Sam': 68.0, 'George': 79.0}
```

---

## Functions in Python

A function is set of statements that take input in the form of parameters, performs computation on the input and returns a result in the form of return statement

### Syntax –

```
def function-name ( parameters if any ):
```

```
    # function code
```

```
    return statement
```

**Note : It is a best practice to avoid usage of input() and print() functions in a function definition**

**WAF to calculate factorial of a number**

```
In [5]: # Function Definition  
def factorial(num):  
    fact = 1  
    for i in range(num, 1, -1):
```

```
        fact *= i
    return fact
```

In [3]: `factorial(5)` *# Function call*

Out[3]: 120

```
In [9]: # Function Definition
def factorial(num):
    if type(num) == int :
        fact = 1
        for i in range(num, 1, -1):
            fact *= i
        return fact
```

In [10]: `print(factorial("abcd"))`

None

**Note - If a function has no return statement it will return None as default**

```
In [11]: # Function returning multiple values
def func(num) :
    return num**2, num**3 # returns multiple values in a single tuple object

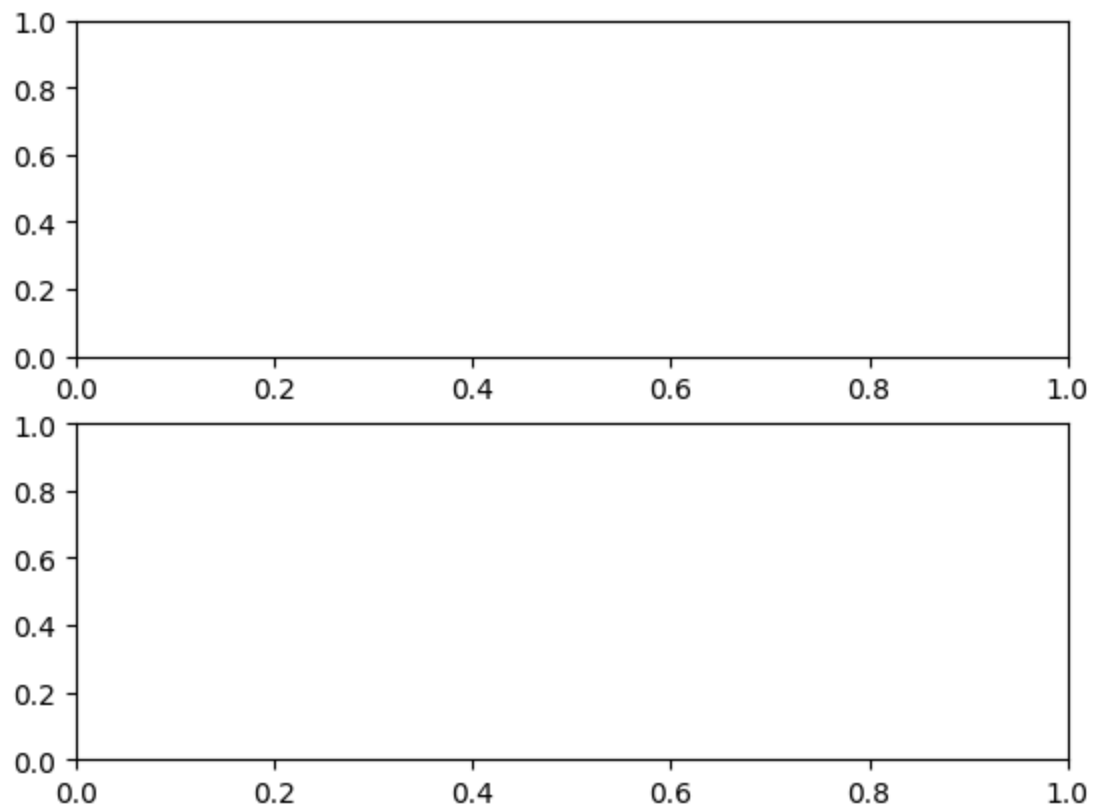
square, cube = func(5) # storing the returned values into variable - unpacking of
```

Out[11]: (25, 125)

```
In [13]: import matplotlib.pyplot as plt

fig, ax = plt.subplots(nrows=2) # examples of unpacking of tuples
```





## Function Arguments

- Required Positional Arguments
- Default Arguments
- Variable length Arguments
- Key-word Arguments
- Variable-length Keyword Arguments

### Required Positional Argument

In [ ]:

### Default Argument

In [ ]:

### Variable-length Argument

In [ ]:

### Key-word Argument

In [ ]:

### Variable-length Keyword Argument

In [ ]: