What are Lists

List is a data type where you can store multiple items under 1 name. More technically, lists act like dynamic arrays which means you can add more items on the fly.

Array Vs Lists

1.Fixed Vs Dynamic Size

2.Convenience -> Hetrogeneous

3.Speed of Execution

4.Memory

Characterstics of a List

1.Ordered

2.Changeble/Mutable

3.Hetrogeneous

4.Can have duplicates

5.are dynamic

6.can be nested

7.items can be accessed

8.can contain any kind of objects in python

Empty list

In [1]:

```
print([])
```

```
[]
```

1D -> Homo list

In [2]:

```
print([1,2,3,4,5])
```

```
[1, 2, 3, 4, 5]
```

2D list

In [3]:

```
print([1,2,3,[4,5]])
```

```
[1, 2, 3, [4, 5]]
```

3D list

In [4]:

```
print([[[1,2],[3,4]],[[5,6],[7,8]]])
```

```
[[[1, 2], [3, 4]], [[5, 6], [7, 8]]]
```

Hetrogenous list

In [5]:

```
print([1,True,5.6,5+6j,'Hello'])
```

```
[1, True, 5.6, (5+6j), 'Hello']
```

Using Type conversion

In [6]:

```
print(list('hello'))
```

```
['h', 'e', 'l', 'l', 'o']
```

Accessing Items from a List

Indexing

In [7]:

```
L = [[[1,2],[3,4]],[[5,6],[7,8]]]
```

In [8]:

```
print(L[0][0][1])#positive
```

```
2
```

Slicing

In [9]:

```
L = [1,2,3,4,5,6,7]
print(L[::-1])
```

```
[7, 6, 5, 4, 3, 2, 1]
```

Adding Items to a List

append():In this methord append an element to the end of the list

In [10]:

```
L = [1,2,3,4,5]
L.append(100)
print(L)
```

```
[1, 2, 3, 4, 5, 100]
```

In [11]:

```
L = [1,2,3,4,5]
L.append([6,7,8])
print(L)
```

```
[1, 2, 3, 4, 5, [6, 7, 8]]
```

extend():Add the specified list to the end of the current list

In [12]:

```
L = [1,2,3,4,5]
L.extend([6,7,8])
print(L)
```

```
[1, 2, 3, 4, 5, 6, 7, 8]
```

In [13]:

```
L = [1,2,3,4,5]
L.extend('delhi')
print(L)
```

```
[1, 2, 3, 4, 5, 'd', 'e', 'l', 'h', 'i']
```

insert():it is a methord insert the specified value at the specified position.

In [14]:

```
# insert():
L = [1,2,3,4,5]
L.insert(1,100)
print(L)
```

```
[1, 100, 2, 3, 4, 5]
```

Editing items in a List

editing with indexing

In [15]:

```
L = [1,2,3,4,5]
L[-1] = 500
print(L)
```

```
[1, 2, 3, 4, 500]
```

editing with slicing

In [16]:

```
L = [1,2,3,4,5]
L[1:4] = [200,300,400]
print(L)
```

```
[1, 200, 300, 400, 5]
```

Deleting items from a List

del: delete the items based on index position.

del with indexing

In [17]:

```
L = [1,2,3,4,5]
del L[-1]
print(L)
```

```
[1, 2, 3, 4]
```

del with slicing

In [18]:

```
L = [1,2,3,4,5]
del L[1:3]
print(L)
```

```
[1, 4, 5]
```

remove: it remove first occurance of a given lisst.

In [19]:

```
L = [1,2,3,4,5]
L.remove(5)
print(L)
```

```
[1, 2, 3, 4]
```

pop:it always delete last items of a list

In [20]:

```
L = [1,2,3,4,5]
L.pop()
print(L)
```

```
[1, 2, 3, 4]
```

clear:it always produce emptyy list

In [21]:

```
L = [1,2,3,4,5]
L.clear()
print(L)
```

```
[]
```

Operations on Lists

Arithmetic (+ ,*)

In [22]:

```
L1 = [1,2,3,4]
L2 = [5,6,7,8]
print(L1 + L2)# Concatenation/Merge
```

```
[1, 2, 3, 4, 5, 6, 7, 8]
```

In [23]:

```
print(L1*3)
```

```
[1, 2, 3, 4, 1, 2, 3, 4, 1, 2, 3, 4]
```

In [24]:

```
L1 = [1,2,3,4,5]
L2 = [1,2,3,4,[5,6]]

print(5 not in L1)
print([5,6] in L2)
```

```
False
True
```

Loops

In [25]:

```
L1 = [1,2,3,4,5]
L2 = [1,2,3,4,[5,6]]
L3 = [[[1,2],[3,4]],[[5,6],[7,8]]]
for i in L3:
  print(i)
```

```
[[1, 2], [3, 4]]
[[5, 6], [7, 8]]
```

List Functions

1.len/min/max/sorted

In [26]:

```
# len/min/max/sorted
L = [2,1,5,7,0]
print(len(L))
print(min(L))
print(max(L))
print(sorted(L,reverse=True))
```

```
5
0
7
[7, 5, 2, 1, 0]
```

2.count

In [27]:

```
L = [1,2,1,3,4,1,5]
L.count(5)
```

Out[27]:

```
1
```

3.index

In [28]:

```
L = [1,2,1,3,4,1,5]
L.index(1)
```

Out[28]:

```
0
```

4.reverse:permanently reverses the list

In [29]:

```
L = [2,1,5,7,0]
L.reverse()
print(L)
```

```
[0, 7, 5, 1, 2]
```

5.sort vs sorted

In [30]:

```
L = [2,1,5,7,0]
print(L)
print(sorted(L))
print(L)
L.sort()
print(L)
```

```
[2, 1, 5, 7, 0]
[0, 1, 2, 5, 7]
[2, 1, 5, 7, 0]
[0, 1, 2, 5, 7]
```

6.copy

In [31]:

```
L = [2,1,5,7,0]
print(L)
print(id(L))
```

```
[2, 1, 5, 7, 0]
1670538875072
```

In [32]:

```
L1 = L.copy()
print(L1)
print(id(L1))
```

```
[2, 1, 5, 7, 0]
1670538873600
```

List Comprehension provides a concise way of creating lists.

newlist = [expression for item in iterable if condition == True]

Advantages of List Comprehension

1.More time-efficient and space-efficient than loops.

2.Require fewer lines of code.

3.Transforms iterative statement into a formula.

In [ ]:

```
```

Add 1 to 10 numbers to a list

In [33]:

```
L = [i for i in range(1,11)]
print(L)
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

scalar multiplication on a vector

In [34]:

```
v = [2,3,4]
s = -3
# [-6,-9,-12]
[s*i for i in v]
```

Out[34]:

```
[-6, -9, -12]
```

Add squares

In [35]:

```
L = [1,2,3,4,5]
[i**2 for i in L]
```

Out[35]:

```
[1, 4, 9, 16, 25]
```

Print all numbers divisible by 5 in the range of 1 to 50

In [36]:

```
[i for i in range(1,51) if i%5 == 0]
```

Out[36]:

```
[5, 10, 15, 20, 25, 30, 35, 40, 45, 50]
```

find languages which start with letter p

In [37]:

```
languages = ['java','python','php','c','javascript']
[language for language in languages if language.startswith('p')]
```

Out[37]:

```
['python', 'php']
```

Nested if with List Comprehension

In [38]:

```
basket = ['apple','guava','cherry','banana']
my_fruits = ['apple','kiwi','grapes','banana']
```

add new list from my_fruits and items if the fruit exists in basket and also starts with 'a'

In [39]:

```
[fruit for fruit in my_fruits if fruit in basket if fruit.startswith('a')]
```

Out[39]:

```
['apple']
```

Print a (3,3) matrix using list comprehension -> Nested List comprehension

In [40]:

```
[[i*j for i in range(1,4)] for j in range(1,4)]
```

Out[40]:

```
[[1, 2, 3], [2, 4, 6], [3, 6, 9]]
```

cartesian products -> List comprehension on 2 lists together

In [41]:

```
L1 = [1,2,3,4]
L2 = [5,6,7,8]
[i*j for i in L1 for j in L2]
```

Out[41]:

```
[5, 6, 7, 8, 10, 12, 14, 16, 15, 18, 21, 24, 20, 24, 28, 32]
```

Zip: The zip() function returns a zip object, which is an iterator of tuples where the first item in each passed iteratoris paired together, and then the second item in each passed iterator are paired together.If the passed iterators have different lengths, the iterator with the least items decides the length of the new iterator.

Write a program to add items of 2 lists indexwise

In [42]:

```
L1 = [1,2,3,4]
L2 = [-1,-2,-3,-4]
list(zip(L1,L2))
[i+j for i,j in zip(L1,L2)]
```

Out[42]:

```
[0, 0, 0, 0]
```

In [43]:

```
L = [1,2,print,type,input]
print(L)
```

```
[1, 2, <built-in function print>, <class 'type'>, <bound method Kernel.raw_input of <ipyk
ernel.ipkernel.IPythonKernel object at 0x00000184F3DBC3A0>>]
```

Disadvantages of Python Lists

Slow

Risky usage

eats up more memory

In [44]:

```
a = [1,2,3]
b = a.copy()

print(a)
print(b)

a.append(4)
print(a)
print(b)
```

```
[1, 2, 3]
[1, 2, 3]
[1, 2, 3, 4]
[1, 2, 3]
```

lists are mutable data type

In [ ]:

```

```

TASK AND SOLUTTION (DAY-04)

1: Combine two lists index-wise(columns wise) Write a program to add two lists index-wise. Create a new list that contains the 0th index item from both the list, then the 1st index item, and so on till the last element. any leftover items will get added at the end of the new list. Given List: list1 = ["M", "na", "i", "Kh"] list2 = ["y", "me", "s", "an"] Output:[['M','y'], ['na', me'], ['i', 's'], ['Kh', 'an']]

In [1]:

```
list1 = ["M", "na", "i", "Kh"]
list2 = ["y", "me", "s", "an"]
[[i,j] for (i,j) in zip(list1,list2)]
```

Out[1]:

```
[['M', 'y'], ['na', 'me'], ['i', 's'], ['Kh', 'an']]
```

2 Add new item to list after a specified item Write a program to add item 7000 after 6000 in the following Python List ist1 = [10, 20, [300, 400, [5000, 6000], 500], 30, 40] Output:[10, 20, [300, 400, [5000, 6000, 7000], 500], 30, 40]

In [2]:

```
list1 = [10, 20, [300, 400, [5000, 6000], 500], 30, 40]
list1[2][2].append(7000)
list1
```

Out[2]:

```
[10, 20, [300, 400, [5000, 6000, 7000], 500], 30, 40]
```

3.Update no of items available Suppose you are given a list of candy and another list of same size representing no of items of respective candy.

i.e -candy_list = ['Jelly Belly','Kit Kat','Double Bubble','Milky Way','Three Musketeers']

no_of_items = [10,20,34,74,32] Write a program to show no. of items of each candy type.

In [3]:

```
candy_list = ['Jelly Belly','Kit Kat','Double Bubble','Milky Way','Three Musketeers']
no_of_items = [10,20,34,74,32]
for (i,j) in zip(candy_list,no_of_items):
  print(i,'-',j)
```

```
Jelly Belly - 10
Kit Kat - 20
Double Bubble - 34
Milky Way - 74
Three Musketeers - 32
```

4.Running Sum on list Write a program to print a list after performing running sum on it.i.e:Input:list1 = [1,2,3,4,5,6]Output:[1,3,6,10,15,21]

In [4]:

```
list1 = [1,2,3,4,5,6]
result = []
sum = 0
for i in list1:
  sum = sum + i
  result.append(sum)
print(result)
```

```
[1, 3, 6, 10, 15, 21]
```

5: You are given a list of integers. You are asked to make a list by running through elements of the list by adding all elements greater and itself.

For 1st element 2 ->>these are greater (4+6+10) values and 2 itself so on adding becomes 22.

For 2nd element 4 ->> greater elements are (6, 10) and 4 itself, so on adding 20

like wise for all other elememts.[2,4,6,10,1]-->[22,20,16,10,23]

In [5]:

```
L = [2,4,6,10,1]
result = []
for i in L:
  sum = 0
  for j in L:
    if i <= j:
      sum = sum + j

  result.append(sum)
print(result)
```

```
[22, 20, 16, 10, 23]
```

6: Find list of common unique items from two list. and show in increasing order Input:num1 = [23,45,67,78,89,34] num2 = [34,89,55,56,39,67]Output: [34, 67, 89]

In [6]:

```
num1 = [23,45,67,78,89,34,67]
num2 = [34,89,55,56,39,67,67]
common = []
for i in num1:
  if i in num2:
    if i not in common:
      common.append(i)
print(common)
```

```
[67, 89, 34]
```

7: Sort a list of alphanumeric strings based on product value of numeric

character in it. If in any string there is no numeric character take it's product value as 1. Input:['1ac21', '23fg', '456', '098d','1','kls'] Output: ['456', '23fg', '1ac21', '1', 'kls', '098d']

In [7]:

```
L = ['1ac21', '23fg', '456', '098d', '1', 'kls']
prod_value = []

for item in L:
    product = 1
    for char in item:
        if char.isdigit():
            product *= int(char)
    prod_value.append(product)

sorted_list = [i[1] for i in sorted(list(zip(prod_value, L)), reverse=True)]

print(sorted_list)
```

```
['456', '23fg', '1ac21', 'kls', '1', '098d']
```

8: Split String of list on K character. Eg:Input:['CampusX is a channel', 'for data-science', 'aspirants.']Output:['CampusX', 'is', 'a', 'channel', 'for', 'data-science', 'aspirants.']

In [8]:

```
L = ['CampusX is a channel', 'for data-science', 'aspirants.']
inp = 'a'
result = []
for i in L:
  result.extend(i.split(inp))
print(result)
```

```
['C', 'mpusX is ', ' ch', 'nnel', 'for d', 't', '-science', '', 'spir', 'nts.']
```

9: Convert Character Matrix to single String using string comprehension. Example 1:Input:[['c', 'a', 'm', 'p', 'u', 'x'], ['i', 's'], ['b', 'e', 's', 't'], ['c', 'h', 'a', 'n', 'n', 'e', 'l']] Output:campux is best channel

In [9]:

```
L = [['c', 'a', 'm', 'p', 'u', 'x'], ['i', 's'], ['b', 'e', 's', 't'], ['c', 'h', 'a', '
n', 'n', 'e', 'l']]
print(" ".join(["".join(i) for i in L]))
```

```
campux is best channel
```

10: Add Space between Potential Words.Example:Input:['campusxIs', 'bestFor', 'dataScientist'] Output: ['campusx Is', 'best For', 'data Scientist']

In [10]:

```
test_list = ["campusxIs", "bestFor", "dataScientist"]
res = []
for i in test_list:
    temp = [[]]
    for char in i:

        if char.isupper():
            temp.append([])
```

```
        temp[-1].append(char)

    temp_string = ""
    for item in temp:
        temp_string = temp_string + "".join(item) + " "
    res.append(temp_string[0:-1])
print(res)
```

```
['campusx Is', 'best For', 'data Scientist']
```

11: Write a program that can perform union operation on 2 lists.

Input:

[1,2,3,4,5,1]

[2,3,5,7,8]

Output:[1,2,3,4,5,7,8]

In [11]:

```
L1 = [1,2,3,4,5,1]
L2 = [2,3,5,7,8]
union = []
for i in L1:
  if i not in union:
    union.append(i)
for j in L2:
  if j not in union:
    union.append(j)
print(union)
```

```
[1, 2, 3, 4, 5, 7, 8]
```

12: Write a program that can find the max number of each row of a matrix

Example: Input:[[1,2,3],[4,5,6],[7,8,9]] Output:[3,6,9]

In [12]:

```
L = [[1,2,3],[4,5,6],[7,8,9]]
result = []
for i in L:
  result.append(max(i))
print(result)
```

```
[3, 6, 9]
```

13: Write a list comprehension to print the following matrix [[0, 1, 2], [3, 4, 5], [6, 7, 8]]

In [13]:

```
[[j + 3*i for j in range(0,3)] for i in range(0,3)]
```

Out[13]:

```
[[0, 1, 2], [3, 4, 5], [6, 7, 8]]
```

14: Write a list comprehension that can transpose a given matrix matrix = [[1,2,3], [4,5,6], [7,8,9]] [1, 4, 7] [2, 5, 8] [3, 6, 9]

In [14]:

```
matrix = [
[1,2,3],
[4,5,6],
[7,8,9]
]
[[row[i] for row in matrix]for i in range(len(matrix))]
```

Out[14]:

```
[[1, 4, 7], [2, 5, 8], [3, 6, 9]]
```

15: Write a list comprehension that can flatten a nested list. Input:matrix = [[1,2,3],[4,5,6],[7,8,9]] Output:[1, 2, 3, 4, 5, 6, 7, 8, 9]

In [15]:

```
matrix = [
[1,2,3],
[4,5,6],
[7,8,9]
]
[item for row in matrix for item in row]
```

Out[15]:

```
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

In [ ]:

```

```

TUPLES

->A tuple in Python is similar to a list. The difference between the two is that we cannot change the elements of a tuple once it is assigned whereas we can change the elements of a list.

->In short, a tuple is an immutable list. A tuple can not be changed in any way once it is created.

*Characterstics

->Ordered

->Unchangeble

->Allows duplicate

create a tuple with a single item

In [1]:

```
t2 = ('hello',)
print(t2)
print(type(t2))
```

```
('hello',)
<class 'tuple'>
```

homo

In [2]:

```
t3 = (1,2,3,4)
print(t3)
```

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

hetro

In [3]:

```
t4 = (1,2.5,True,[1,2,3])
print(t4)
```

```
(1, 2.5, True, [1, 2, 3])
```

tuple

In [4]:

```
t5 = (1,2,3,(4,5))
print(t5)
```

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

using type conversion

In [5]:

```
t6 = tuple('hello')
print(t6)
```

```
('h', 'e', 'l', 'l', 'o')
```

Accessing Items

1.Indexing

2.Slicing

In [6]:

```
t3 = (1,2,3,4)
```

In [7]:

```
print(t3)
```

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

In [8]:

```
print(t3[0])
```

```
1
```

In [9]:

```
print(t3[-1])
```

```
4
```

In [10]:

```
t5[-1][0]
```

Out[10]:

```
4
```

Editing items

In [11]:

```
print(t3)
t3[0] = 100
# immutable just like strings
```

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

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_19760\2742057923.py in <module>
      1 print(t3)
----> 2 t3[0] = 100
      3 # immutable just like strings

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

Adding items

In [12]:

```
print(t3) # not possible
```

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

Deleting items

In [13]:

```
print(t3)
del t3
print(t3)
```

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

---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_19760\1737609542.py in <module>
      1 print(t3)
      2 del t3
----> 3 print(t3)

NameError: name 't3' is not defined
```

In [14]:

```
t = (1,2,3,4,5)

t[-1:-4:-1]
```

Out[14]:

```
(5, 4, 3)
```

In [15]:

```
print(t5)
del t5[-1]
```

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

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_19760\3182731913.py in <module>
      1 print(t5)
----> 2 del t5[-1]

TypeError: 'tuple' object doesn't support item deletion
```

operation in tuple

In [16]:

```
t1 = (1,2,3,4)
t2 = (5,6,7,8)
```

addtion

In [17]:

```
print(t1 + t2)
```

```
(1, 2, 3, 4, 5, 6, 7, 8)
```

Multiplication

In [18]:

```
print(t1*3)
```

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

membership

In [19]:

```
1 in t1
```

Out[19]:

```
True
```

iteration

In [20]:

```
for i in t1:
  print(i)
```

```
1
2
3
4
```

len/sum/min/max/sorted

In [21]:

```
t = (1,2,3,4)
```

In [22]:

```
len(t)
```

Out[22]:

```
4
```

In [23]:

```
sum(t)
```

Out[23]:

```
10
```

In [24]:

```
min(t)
```

Out[24]:

```
1
```

In [25]:

```
max(t)
```

Out[25]:

```
4
```

In [26]:

```
sorted(t,reverse=True)
```

Out[26]:

```
[4, 3, 2, 1]
```

count

In [27]:

```
t = (1,2,3,4,5)
t.count(50)
```

Out[27]:

```
0
```

In [28]:

```
# index
t.index(50)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_19760\3136301198.py in <module>
      1 # index
----> 2 t.index(50)

ValueError: tuple.index(x): x not in tuple
```

Difference between Lists and Tuples

Syntax, Mutability, Speed, Memory, Built in functionality, Error prone, Usability,

In [29]:

```
import time
L = list(range(100000000))
T = tuple(range(100000000))
start = time.time()
for i in L:
  i*5
print('List time',time.time()-start)
start = time.time()
for i in T:
  i*5
print('Tuple time',time.time()-start)
```

```
List time 20.871058702468872
Tuple time 30.75348401069641
```

In [30]:

```
import sys
L = list(range(1000))
T = tuple(range(1000))
print('List size',sys.getsizeof(L))
print('Tuple size',sys.getsizeof(T))
```

```
List size 8056
Tuple size 8040
```

In [31]:

```
a = [1,2,3]
```

```
b = a
a.append(4)
print(a)
print(b)
```

```
[1, 2, 3, 4]
[1, 2, 3, 4]
```

In [32]:

```
a = (1,2,3)
b = a
a = a + (4,)
print(a)
print(b)
```

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

Special Syntax

tuple unpacking

In [33]:

```
a,b,c = (1,2,3)
print(a,b,c)
```

```
1 2 3
```

In [34]:

```
a,b = (1,2,3)
print(a,b)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_19760\164063033.py in <module>
----> 1 a,b = (1,2,3)
      2 print(a,b)

ValueError: too many values to unpack (expected 2)
```

In [35]:

```
a = 1
b = 2
a,b = b,a
print(a,b)
```

```
2 1
```

In [36]:

```
a,b,*others = (1,2,3,4)
print(a,b)
print(others)
```

```
1 2
[3, 4]
```

zipping tuples

In [37]:

```
a = (1,2,3,4)
```

```
b = (5,6,7,8)

tuple(zip(a,b))
```

Out[37]:

```
((1, 5), (2, 6), (3, 7), (4, 8))
```

SET

*A set is an unordered collection of items. Every set element is unique (no duplicates) and must be immutable

(cannot be changed).However, a set itself is mutable. We can add or remove items from it. ***Sets can also be used to perform mathematical set operations like union, intersection, symmetric difference, etc.***
Characterstics:**Unordered** ***Mutable*****No Duplicates *Can't contain mutable data types**

Creating Sets

empty

In [38]:

```
s = set()
print(s)
print(type(s))
```

```
set()
<class 'set'>
```

1D and 2D

In [39]:

```
s1 = {1,2,3}
print(s1)
```

```
{1, 2, 3}
```

In [40]:

```
s2 = {1,2,3,{4,5}}
print(s2)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_19760\3523522171.py in <module>
----> 1 s2 = {1,2,3,{4,5}}
      2 print(s2)

TypeError: unhashable type: 'set'
```

homo and hetro

In [41]:

```
s3 = {1,'hello',4.5,(1,2,3)}
print(s3)
```

```
{1, (1, 2, 3), 4.5, 'hello'}
```

using type conversion

In [42]:

```
s4 = set([1,2,3])
print(s4)
```

```
{1, 2, 3}
```

duplicates not allowed

In [43]:

```
s5 = {1,1,2,2,3,3}
print(s5)
```

```
{1, 2, 3}
```

set can't have mutable items

In [44]:

```
s6 = {1,2,[3,4]}
print(s6)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_19760\1011244668.py in <module>
----> 1 s6 = {1,2,[3,4]}
      2 print(s6)

TypeError: unhashable type: 'list'
```

In [45]:

```
s1 = {1,2,3}
s2 = {3,2,1}
print(s1 == s2)
```

```
True
```

Accessing Items

In [46]:

```
s1 = {1,2,3,4}
s1[0:3]
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_19760\1772707306.py in <module>
      1 s1 = {1,2,3,4}
----> 2 s1[0:3]

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

Editing Items

In [47]:

```
s1 = {1,2,3,4}
s1[0] = 100
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
```

```
~\AppData\Local\Temp\ipykernel_19760\1929688710.py in <module>
      1 s1 = {1,2,3,4}
----> 2 s1[0] = 100

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

Adding Items

In [48]:

```
S = {1,2,3,4}
```

In [49]:

```
S.add(5)
```

In [50]:

```
print(S)
```

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

In [51]:

```
S.update([5,6,7])
```

In [52]:

```
print(S)
```

```
{1, 2, 3, 4, 5, 6, 7}
```

Deleting Items

del

In [53]:

```
s = {1,2,3,4,5}
print(s)
del s[0]
print(s)
```

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

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_19760\3862068075.py in <module>
      1 s = {1,2,3,4,5}
      2 print(s)
----> 3 del s[0]
      4 print(s)

TypeError: 'set' object doesn't support item deletion
```

discard

In [54]:

```
s.discard(50)
print(s)
```

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

remove

In [55]:

```
s.remove(50)
print(s)
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_19760\3169689303.py in <module>
----> 1 s.remove(50)
      2 print(s)

KeyError: 50
```

pop

In [56]:

```
s.pop()
```

Out[56]:

```
1
```

clear

In [57]:

```
s.clear()
print(s)
```

```
set()
```

Set Operation

In [58]:

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

Union(|)

In [59]:

```
s1 | s2
```

Out[59]:

```
{1, 2, 3, 4, 5, 6, 7, 8}
```

Intersection(&)

In [60]:

```
s1 & s2
```

Out[60]:

```
{4, 5}
```

Symmetric Difference(^)

In [61]:

```
s1 - s2
```

Out[61]:

```
{1, 2, 3}
```

In [62]:

```
s2 - s1
```

Out[62]:

```
{6, 7, 8}
```

In [63]:

```
s1 ^ s2
```

Out[63]:

```
{1, 2, 3, 6, 7, 8}
```

Membership Test

In [64]:

```
1 not in s1
```

Out[64]:

```
False
```

Iteration

In [65]:

```
for i in s1:
  print(i)
```

```
1
2
3
4
5
```

Set Functions

In [66]:

```
s = {3,1,4,5,2,7}
```

len

In [67]:

```
len(s)
```

Out[67]:

```
6
```

sum

In [68]:

```
sum(s)
```

Out[68]:

```
22
```

min

In [69]:

```
min(s)
```

Out[69]:

```
1
```

max

In [70]:

```
max(s)
```

Out[70]:

```
7
```

sorted

In [71]:

```
sorted(s,reverse=True)
```

Out[71]:

```
[7, 5, 4, 3, 2, 1]
```

union/update

In [72]:

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

In [73]:

```
s1 | s2
s1.union(s1)
```

Out[73]:

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

In [74]:

```
s1.update(s2)
print(s1)
print(s2)
```

```
{1, 2, 3, 4, 5, 6, 7, 8}
{4, 5, 6, 7, 8}
```

intersection/intersection_update

In [75]:

In [75]:

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

s1.intersection(s2)

s1.intersection_update(s2)
print(s1)
print(s2)
```

```
{4, 5}
{4, 5, 6, 7, 8}
```

difference/difference_update

In [76]:

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

s1.difference(s2)

s1.difference_update(s2)
print(s1)
print(s2)
```

```
{1, 2, 3}
{4, 5, 6, 7, 8}
```

symmetric_difference/symmetric_difference_update

In [77]:

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

s1.symmetric_difference(s2)

s1.symmetric_difference_update(s2)
print(s1)
print(s2)
```

```
{1, 2, 3, 6, 7, 8}
{4, 5, 6, 7, 8}
```

isdisjoint/issubset/issuperset

In [78]:

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

s1.isdisjoint(s2)
```

Out[78]:

```
True
```

In [79]:

```
s1 = {1,2,3,4,5}
s2 = {3,4,5}
s1.issuperset(s2)
```

Out[79]:

```
True
```

copy

In [80]:

```
s1 = {1,2,3}
s2 = s1.copy()

print(s1)
print(s2)
```

```
{1, 2, 3}
{1, 2, 3}
```

Frozenset:Frozen set is just an immutable version of a Python set object

create frozenset

In [81]:

```
fs1 = frozenset([1,2,3])
fs2 = frozenset([3,4,5])
fs1 | fs2
```

Out[81]:

```
frozenset({1, 2, 3, 4, 5})
```

what works and what does not

works -> all read functions

does't work -> write operations

In [82]:

```
fs = frozenset([1,2,frozenset([3,4])])
fs
```

Out[82]:

```
frozenset({1, 2, frozenset({3, 4})})
```

Set Comprehension

In [83]:

```
{i**2 for i in range(1,11) if i>5}   # examples
```

Out[83]:

```
{36, 49, 64, 81, 100}
```

Dictionary

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.In some languages it is known as map or assosiative arrays.

dict = { 'name' : 'nitish' , 'age' : 33 , 'gender' : 'male' }

Characterstics:

1.Mutable

2.Indexing has no meaning

3.keys can't be duplicated

4.keys can't be mutable items

Create Dictionary

empty dictionary

In [84]:

```
d = {}
d
```

Out[84]:

```
{}
```

1D dictionary

In [85]:

```
d1 = { 'name' : 'nitish' ,'gender' : 'male' }
d1
```

Out[85]:

```
{'name': 'nitish', 'gender': 'male'}
```

with mixed keys

In [86]:

```
d2 = {(1,2,3):1,'hello':'world'}
d2
```

Out[86]:

```
{(1, 2, 3): 1, 'hello': 'world'}
```

2D dictionary -> JSON

In [87]:

```
s = {
    'name':'nitish',
     'college':'bit',
     'sem':4,
     'subjects':{
         'dsa':50,
```

```
        'maths':67,
        'english':34
    }
}
s
```

Out[87]:

```
{'name': 'nitish',
 'college': 'bit',
 'sem': 4,
 'subjects': {'dsa': 50, 'maths': 67, 'english': 34}}
```

using sequence and dict function

In [88]:

```
d4 = dict([('name','nitish'),('age',32),(3,3)])
d4
```

Out[88]:

```
{'name': 'nitish', 'age': 32, 3: 3}
```

duplicate keys

In [89]:

```
d5 = {'name':'nitish','name':'rahul'}
d5
```

Out[89]:

```
{'name': 'rahul'}
```

mutable items as keys

In [90]:

```
d6 = {'name':'nitish',(1,2,3):2}
print(d6)
```

```
{'name': 'nitish', (1, 2, 3): 2}
```

Accessing items

In [91]:

```
my_dict = {'name': 'Jack', 'age': 26}
# []
my_dict['age']
# get
my_dict.get('age')

s['subjects']['maths']
```

Out[91]:

```
67
```

Adding key-value pair

In [93]:

```
d4['gender'] = 'male'
```

```
d4
d4['weight'] = 72
d4

s['subjects']['ds'] = 75
s
```

Out[93]:

```
{'name': 'nitish',
 'college': 'bit',
 'sem': 4,
 'subjects': {'dsa': 50, 'maths': 67, 'english': 34, 'ds': 75}}
```

Remove key-value pair

In [94]:

```
d = {'name': 'nitish', 'age': 32, 3: 3, 'gender': 'male', 'weight': 72}
```

In [95]:

```
# pop
d.pop(3)
print(d)
```

In [96]:

```
# popitem
d.popitem()
print(d)
```

```
{'name': 'nitish', 'age': 32, 3: 3, 'gender': 'male'}
```

In [97]:

```
# del
del d['name']
print(d)
```

```
{'age': 32, 3: 3, 'gender': 'male'}
```

In [98]:

```
# clear
d.clear()
print(d)
```

```
{}
```

In [99]:

```
del s['subjects']['maths']
s
```

Out[99]:

```
{'name': 'nitish',
 'college': 'bit',
 'sem': 4,
 'subjects': {'dsa': 50, 'english': 34, 'ds': 75}}
```

Editing key-value pair

In [100]:

```
s['subjects']['dsa'] = 80
s
```

Out[100]:

```
{'name': 'nitish',
 'college': 'bit',
 'sem': 4,
 'subjects': {'dsa': 80, 'english': 34, 'ds': 75}}
```

Dictionary Operations

Membership

In [101]:

```
print(s)
'name' in s
```

```
{'name': 'nitish', 'college': 'bit', 'sem': 4, 'subjects': {'dsa': 80, 'english': 34, 'ds
': 75}}
```

Out[101]:

```
True
```

Iteration

In [102]:

```
d = {'name':'nitish','gender':'male','age':33}
for i in d:
  print(i,d[i])
```

```
name nitish
gender male
age 33
```

len/sorted

In [103]:

```
len(d)
print(d)
```

```
{'name': 'nitish', 'gender': 'male', 'age': 33}
```

In [104]:

```
sorted(d,reverse=True)
```

Out[104]:

```
['name', 'gender', 'age']
```

In [105]:

```
max(d)
```

Out[105]:

```
'name'
```

update

In [106]:

```
d1 = {1:2,3:4,4:5}
```

```
d2 = {4:7,6:8}
d1.update(d2)
print(d1)
```

```
{1: 2, 3: 4, 4: 7, 6: 8}
```

Dictionary Comprehension

```
{ key: value for vars in iterable }
```

print 1st 10 numbers and their squares

In [107]:

```
{i:i**2 for i in range(1,11)}
```

Out[107]:

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

In [108]:

```
distances = {'delhi':1000,'mumbai':2000,'bangalore':3000}
print(distances.items())
```

```
dict_items([('delhi', 1000), ('mumbai', 2000), ('bangalore', 3000)])
```

using existing dict

In [109]:

```
distances = {'delhi':1000,'mumbai':2000,'bangalore':3000}
{key:value*0.62 for (key,value) in distances.items()}
```

Out[109]:

```
{'delhi': 620.0, 'mumbai': 1240.0, 'bangalore': 1860.0}
```

using zip

In [110]:

```
days = ["Sunday", "Monday","Tuesday","Wednesday","Thursday","Friday","Saturday"]
temp_C = [30.5,32.6,31.8,33.4,29.8,30.2,29.9]
{i:j for (i,j) in zip(days,temp_C)}
```

Out[110]:

```
{'Sunday': 30.5,
 'Monday': 32.6,
 'Tuesday': 31.8,
 'Wednesday': 33.4,
 'Thursday': 29.8,
 'Friday': 30.2,
 'Saturday': 29.9}
```

using if condition

In [111]:

```
products = {'phone':10,'laptop':0,'charger':32,'tablet':0}
{key:value for (key,value) in products.items() if value>0}
```

Out[111]:

```
{'phone': 10, 'charger': 32}
```

Nested Comprehension

print tables of number from 2 to 4

In [112]:

```
{i:{j:i*j for j in range(1,11)} for i in range(2,5)}
```

Out[112]:

```
{2: {1: 2, 2: 4, 3: 6, 4: 8, 5: 10, 6: 12, 7: 14, 8: 16, 9: 18, 10: 20},
 3: {1: 3, 2: 6, 3: 9, 4: 12, 5: 15, 6: 18, 7: 21, 8: 24, 9: 27, 10: 30},
 4: {1: 4, 2: 8, 3: 12, 4: 16, 5: 20, 6: 24, 7: 28, 8: 32, 9: 36, 10: 40}}
```

In [ ]:

```

```