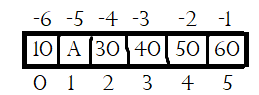
**LIST**

List is a data type in Python. A list is a collection of items which can be of any type. List is dynamic because based on requirement we can increase the size and decrease the size. In list elements are placed within square brackets and with comma separator. Duplicates values are allowed. We can differentiate duplicate elements by using index. Index plays important role. Python supports both positive and negative index. +ve index means from left to right, where –ve index means from right to left.

**Example:**

**X=[10,”A”,30,40,50,60]**

The X can be represented by following figure



X[0]=10 X[3]=40

X[1]=A X[4]=50

X[2]=30 X[5]=60

List objects are mutable.

**Creation of list objects:**

1. **We can create empty list object as follows:**

L=[]

1. **With list function:**

L=list(range(0,10))

print(L)

1. **With dynamic input:**

L=eval(input(“ Enter list “))

print(L)

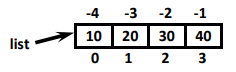
1. **If we know elements then we create list as follows**

L=[2,3,4,5,6,7]

**ACCESSING ELEMENTS OF LIST:**

We can access elements of the list either by using index or by using slice operator(:)

**By using index:**

List follows zero based index. ie index of first element is zero. List supports both +ve and -ve indexes. +ve index meant for Left to Right -ve index meant for Right to Left list=[10,20,30,40]

print(list[0]) ==>10

print(list[-1]) ==>40

print(list[10]) ==>IndexError: list index out of range

**By using slice operator:**

**Syntax**:

list2= list1[start:stop:step]

start ==>it indicates the index where slice has to start default value is 0

stop ===>It indicates the index where slice has to end. Default value is max allowed index of list i.e length of the list

step ==>increment value - default value is 1

**Example:**

**Output:**

[3, 5, 7]

[5, 7, 9]

[4, 5, 6, 7]

[9, 7, 5]

[5, 6, 7, 8, 9, 10]

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

print(n[2:7:2])

print(n[4::2])

print(n[3:7])

print(n[8:2:-2])

print(n[4:100])

**FUNCTIONS OR OPERATIONS OF LIST**

**len:** This function returns the length of a list.

**Example:** n=[1,3,4,5,6,8,9]

print(len(n)) 🡺7

**concatenation:** This function is used to join two list into one list using (+).

**Example**: n=[1,2,3,4,5]

n1=[6,7,8,9,10]

print(n+n1) 🡺[1,2,3,4,5,6,7,8,9,10]

**repetition:** This function is used to repeat the elements in the list using (\*)

**Example:** n=[1,2,4]

print(n\*3) 🡺[1,2,4,1,2,4,1,2,4]

**in:** It checks if the value is present in the list. If present it returns True else False.

**Example**: n=[1,2,3,4,5,6,7]

print(1 in n) 🡺True

print(10 in n) 🡺False

**Not in:** It checks if the value is not present in the list. If not present it returns True else false.

**Example**: n=[1,2,3,4,5,6,7]

print(1 not in n) 🡺False

print(10 not in n) 🡺True

**max:** Returns the maximum value from the list.

**Example:** L=[12,45,64,24,453,33]

print(max(L)) 🡺453

**Min:** Returns the minimum value from the list.

**Example:** L=[12,45,64,24,453,33]

print(max(L)) 🡺12

**Sum:** This function adds the values in the list that has numbers.

**Example:** n=[1,2,3,4,5,6,7]

print(sum(n)) 🡺28

**sorted:** Returns a new sorted list. The original list is not sorted.

**Example**: li=[3,23,5,4,1,45]

Lis=sorted(li)

print (Lis) 🡺[1,3,4,5,23,45]

**all:** This function returns True if all elements of the list are true (or if the list is empty).

**Example:** l=[6,3,7,0,1,2,3,9]

print(all(l)) 🡺False

**any:** This function returns True if any element of the list is true. If the list is empty, return false.

**Example:** l=[6,3,7,0,1,2,3,9]

print(any(l)) 🡺True

**LIST METHODS**

**count():** Returns the number of occurrences of a specified item in the list.

**Syntax:** list.count(obj)

**Example:** n=[2,2,2,3,3,3,44]

print(n.count(3)) 🡺3

**index():** Returns the index of first occurrence of the specified item. ValueError if no item in the list.

**Syntax:** list.index(obj)

**Example:**  n=[2,2,2,3,3,3,44]

print(n.index(2)) 🡺0

**append():** Adds an element at the end of the list.

**Syntax:** list.append(obj)

**Example:** list=[]

list.append(3)

list.append('a')

list.append(4)

print(list) 🡺[3,’a’,4]

**insert():** Inserts an element at the specified index position.

**Syntax:** list.insert(index,obj)

**Example:** n=[1,23,4,5,6,7,8]

n.insert(3,100)

print(n) 🡺 [1, 23, 4, 100, 5, 6, 7, 8]

**extend():** Add all items of one list to another list.

**Syntax:** list.extend(list2)

**Example:** l1=[1,2,3,4]

l2=[5,6,7,8]

l1.extend(l2)

print(l1) 🡺[1,2,3,4,5,6,7,8]

**remove():** Removes the specified item from the list. If the item present multiple times then only the first occurrence will be removed. ValueError is generated if the item is not present in the list. To work with this method first we have to check element is present or not using **in** operator.

**Syntax:** list.remove(obj)

**Example:** num=[32,35,52,2,3,6,88]

num.remove(3)

print(num) 🡺[32,35,52,2,6,88]

**pop():** It removes and returns the last element of the list. If the list is empty then pop() function raises IndexError. We can use to remove elements based on index.

**Syntax:** list.pop() or list.pop(index)

**Example:** num=[32,35,52,2,3,6,88]

print(num.pop()) 🡺88

**Example:**  num=[32,35,52,2,3,6,88]

print(num.pop(2)) 🡺52

**reverse():** Reverse the elements in the list.

**Syntax:** list.reverse()

**Example:** num=[32,35,52,2,3,6,88]

num.reverse()

print(num) 🡺[88,6,2,52,35,32]

**sort():** Sorts the elements in the list. To use this function, list should contain only homogeneous elements. Default ordering is Ascending order.

**Syntax:** list.sort()

**Example:** num=[32,35,52,2,3,6,88]

num.sort()

print(num) 🡺[2,3,6,32,35,52,88]

**DIFFERENCE BETWEEN REMOVE AND POP**

|  |  |
| --- | --- |
| **remove()** | **pop()** |
| We use to remove special element from the list. | Removes the last element from the list and removes the element based on the index position. |
| It can’t return any value | It returns the removed value. |
| ValueError if no element available | Error if list is empty |

**STRINGS**

Any sequence of characters within either single quotes or double quotes is considered as a String.

**Example:**

Str=’sbvr degree college’

Str1=”cnu reddy”

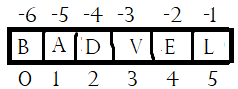
**ACCESSING CHARACTERS OF A STRING**

1. By using index
2. By using slice operator

**By using index:**

Python supports both +ve and –ve index.

+ve index means left to right (forward direction).

-ve index means right to left(Backward direction).

**Example:**

Str=’BADVEL’

print(Str[0]) 🡺B

print(Str[-3]) 🡺V

print(Str[10]) 🡺 IndexError: string index out of range

**By using slice operator:**

A substring of a string is called slicing. The slice operation is used to refer to sub-parts of sequences and strings. We can take a subset of string from the original string by using [] operator also known as slicing operator.

**Output:**

Learning Python is a very easy language

ning Python is a very easy language

Learning P

LrnPh vya na

eaga seye inhy nnaL

egaugnal ysae yrev a si nohtyP gninraeL

**Syntax:**

**S[beginindex:endindex:step]**

**Examples:**

str="Learning Python is a very easy language"

print(str[:])

print(str[4:])

print(str[:10])

print(str[::3])

print(str[::-2])

print(str[::-1])

**STRING ARE IMMUTABLE**

Python stings are immutable which means that once created they cannot be changed. Whenever we try to modify an existing string variable, a new string is created.

**Example:**

str="sreenu"

str1=" reddy "

print(id(str1))

str1=str+str1

print(str1)

print(id(str))

print(id(str1))

**STRING OPERATIONS IN PYTHON**

**Concatenating strings:**

The word concatenate mean to join together.

**Example:**

str="SBVR"

str1=" DEGREE COLLEGE"

str2=str+str1

**Output:**

SBVR DEGREE COLLEGE

print(str2)

**Appending strings:**

Appending means to add something at the end. In python, we can add one string at the end of another string using the += operators.

**Output:**

Enter the name sreenu reddy

Hello sreenu reddy.Welcome to python proramming

**Example:**

str="Hello "

name=input("Enter the name ")

str+= name

str +=".Welcome to python proramming"

print(str)

**Multiplying strings:**

We use \* operator to repeat a string n number of times.

**Example:**

**Output:**

cnu cnu cnu

str="cnu "

print(str \*3)

**Comparing strings:**

We can use comparison operators (<,<=,>,>=) and equality operators(==,!=) for strings. Comparison will be performed based on alphabetical order.

**Example:**

s1=input("Enter the first string: ")

**Output:**

Enter the first string: sreenu

Enter the second string: ashwika

frist string is greater than second string

s2=input("Enter the second string: ")

if(s1==s2):

print(" Both are equal ")

elif (s1<s2):

print(" first string is less than second string ")

else:

print(" frist string is greater than second string ")

**BUILT-IN FUNCTIONS IN STRINGS**

**Removing spaces from the string:**

**rstrip():** Removes the spaces at the right hand side

**lstip():** Removes the spaces at the left hand side

**strip():** Removes spaces at the both sides

**Example:**

s1=" badvel"

print(s1.lstrip())

s2="degree "

print(s2.rstrip())

s3=" sbvr degree college "

print(s3.strip())

**Changing case of a string:**

**upper():** Convert all characters to upper case

**lower():** Convert all characters to lower case

**swapcase():** Converts all lower case characters to upper case and all upper case to lower case.

**title():** Converts first character in every word to upper case

**capitalize():** Only first character will be converted to upper case and all remaining characters can be converted to lower case.

**Output:**

badvel

DEGREE

SBVR dEGREE college

I Am From Badvel

i am studying in bavel

**Example:**

s="BADVEL"

s1="degree"

s2=" sbvr Degree COLLEGE "

t=" i am from badvel "

c=" i am studying in bavel "

print(s.lower())

print(s1.upper())

print(s2.swapcase())

print(t.title())

print(c.capitalize())

**len():** This function returns the length of the string.

**Example:**

s=" i am from sbvr degree college "

print(len(s))

**replace()**: This function is used to replace all old substrings with new string.

**Example:**

**Output:**

i cm from bcdvel

s=" i am from badvel "

print(s.replace('a','c'))

**split():** We can split the given string according to specified separator by using split() method.

L=s.split(separator)

**Output:**

sbvr

degree

college

**Example:**

s="sbvr degree college"

l=s.split()

for x in l:

print(x)

**join():** We can join a group of strings(list or tuple) with respect to the given separator.

S=separator.join(group of strings)

**Output:**

sunny-bunny-chinny

**Example:**

t=['sunny','bunny','chinny']

s='-'.join(t)

print(s)

**count():** We can find the number of occurrences of substring present in the given string by using count().

s.count(substring)🡺 It will search throughout the string.

s.count(substring,begin,end)🡺 It will search from begin index to end -1 index

**Example:**

**Output:**

6

4

2

s="abcabcabcabcadda"

print(s.count('a'))

print(s.count('ab'))

print(s.count('a',3,7))

**Check Type Of Characters Present In The String**

**isalnum():** Returns true, if all characters are alphanumeric (a to z, A to Z, 0 to 9)

**isalpha():** Returns true, if all characters are only alphabet symbols ( a to z, A to Z)

**isdigit():** Returns true, if all characters are digits only (0 to 9)

**islower():** Returns true, if all characters are lower case alphabet symbols

**isupper():** Returns true, if all characters are upper case alphabet symbols

**istitle():** Returns true, if sting is title case

**isspace():** Returns true, if string contains only spaces.

**Example:**

**Output:**

True

True

False

True

False

True

False

s="2"

print(s.isalnum())

print(s.isdigit())

print(s.isalpha())

str='seenu'

print(str.islower())

print(str.isupper())

str1=" I Am From Badvel"

print(str1.istitle())

print(str1.isspace())

**FORMATTING THE STRINGS**

We can format the strings with variable values by using replacement operator {} and format() method.

**Example:**

**Output:**

nitiksha reddy age is 5

nitiksha reddy age is 5

nitiksha reddy age is 5

name='nitiksha reddy'

age=5

print("{} age is {}".format(name,age))

print("{0} age is {1}".format(name,age))

print("{x} age is {y}".format(x=name,y=age))

The % operator is used for string formatting in Python. The % operator takes a format string on the left (that has %d, %s, etc) and the corresponding values in tuple on the right. The format operator, % allows users to construct strings, replacing parts of the strings with the data stored in variables.

**Output:**

Name =nitiksha reddy age is =5

**Syntax:**

“<format>” % (<values>)

**Example:**

name='nitiksha reddy'

age=5

print("Name =%s age is =%d" %(name,age))

|  |  |
| --- | --- |
| **Format symbol** | **Purpose** |
| %c | Character |
| %d or %i | Signed decimal integer |
| %s | String |
| %u | Unsigned decimal integer |
| %o | Octal integer |
| %x or %X | Hexadecimal integer |
| %e or %E | Exponential notation |
| %f | Floating point number |
| %g or %G | Short number in floating point or exponential notation |

**TUPLE DATA STRUCTURE**

1. Tuple is exactly same as List except that it is immutable. i.e once we creates Tuple object, we cannot perform any changes in that object. Hence Tuple is read only version of List.
2. If our data is fixed and never changes then we should go for Tuple
3. Insertion order is preserved
4. Duplicates are allowed
5. Heterogeneous objects are allowed
6. We represent Tuple elements within parenthesis and with comma separator

**Example:**

t=('a',4,64,'j')

print(t)

print(type(t))

**TUPLE CREATION:**

**1.t=()**

Creation of empty tuple

**2.t=(23,)**

Creation of single valued tuple, parenthesis are optional, should ends with comma

**3.t=10,20,30**

Creation of multi values tuples & parenthesis are optional

**4.By using tuple() function**

list=[12,42,43,33]

t=tuple(list)

print(t)

**ACCESSING ELEMENTS OF TUPLE**

We can access either by index or by slice operator

1. **By using index:**

t=(10,20,30,40,50,60)

print(t[0])

print(t[-1])

print(t[20])🡺 Indexerror: tuple index out of range

1. **By using slice operator:**

t=(10,20,30,40,50,60)

print(t[2:6])

print(t[: :])

print(t[::3])

**MATHEMATICAL OPERATORS FOR TUPLE**

1. **Concatenation operator (+)**

**Output:**

(10, 20, 30, 40, 50, 60, 34, 534, 22)

t=(10,20,30,40,50,60)

t1=(34,534,22)

t2=t+t1

print(t2)

1. **Multiplication operator or repetition operator(\*)**

t=(10,20,30,40,50,60)

**Output:**

(10, 20, 30, 40, 50, 60, 10, 20, 30, 40, 50, 60)

t1=t\*2

print(t1)

**FUNCTIONS OF TUPLE**

**Len():**

Returns the number of elements present in the tuple

**Example:**

t=(10,20,30,40,50,60)

print(len(t))

**Count():**

Returns the number of occurrence of given element in the tuple

**Example:**

t=3,4,5,6,4,4,5,6,4,3,3,2,23,5,

print(t.count(4))

**Index():**

Return the index of first occurrence of the given element

**Example:**

t=(1,2,23,534,234,24,55)

print(t.index(534))

**Sorted():**

To sort elements based on default natural sorting order

**Example:**

t=(1,2,23,534,234,24,55)

print(sorted(t))

**Min() and max():**

Returns the minimum and maximum values.

**Example:**

t=(1,2,23,534,234,24,55)

print(min(t))

print(max(t))

|  |  |
| --- | --- |
| **List** | **Tuple** |
| Elements are placed in square brackets | Elements are placed in parenthesis |
| List objects are mutable | Tuple objects are immutable |
| If the content is not fixed and keep on changing then we should go for list | If the content is fixed and never changes then we should go for tuple |

**DICTIONARY DATA STRUCTURES**

If we want to represent a group of objects as key-value pairs then we should go for dictionary.

Duplicate keys are not allowed by duplicate values are allowed.

Heterogeneous objects are allowed for both key and values

Insertion order is preserved

Dictionaries are mutable and dynamic.

How to create dictionary?

d={} or d=dict()

We create empty dictionary. We can add entries as follows:

d[100]=”sreenu”

d[200]=”ashwika”

d[300]=”nitiksha”

print(d)

**output:**

{100: 'sreenu', 200: 'ashwika', 300: 'nitiksha'}

If we know data in advance then we can create dictionary as follows

d={100:'sreenu' ,200:'ashwika', 300:'nitiksha'}

d={key:value, key:value}

By using keys we can access the data in the dictionary.

d={100:'sreenu' ,200:'ashwika', 300:'nitiksha'}

print(d[100]) #sreenu

print(d[300]) #nitiksha

If the specified key is not available then we will get KeyError

print(d[400]) # KeyError: 400

We can prevent this by checking whether key is already available or not by using has\_key() function or by using in operator.

d.has\_key(400) ==> returns 1 if key is available otherwise returns 0

But has\_key() function is available only in Python 2 but not in Python 3.

Hence compulsory we have to use in operator.

if 400 in d:

print(d[400])

**Updating Dictionary**

If the key is not available then a new entry will be added to the dictionary with the specified key-value pair.

If the key is already available then old value will be replaced with new value.

d={100:"sreenu",200:"ashwika",300:"nitiksha"}

print(d)

d[400]="pavan"

print(d)

d[100]="sunny"

print(d)

**Output**

{100: 'sreenu', 200: 'ashwika', 300: 'nitiksha'}

{100: 'sreenu', 200: 'ashwika', 300: 'nitiksha', 400: 'pavan'}

{100: 'sunny', 200: 'ashwika', 300: 'nitiksha', 400: 'pavan'}

**Delete elements from dictionary**

**del d[key]:**

It deletes entry associated with the specified key. If the key is not available then we will get syntax error.

d={100:"durga",200:"ravi",300:"shiva"}

print(d)

del d[100]

print(d)

del d[400]

**Output**

{100: 'durga', 200: 'ravi', 300: 'shiva'}

{200: 'ravi', 300: 'shiva'}

KeyError: 400

**d.clear()**

To remove all entries from the dictionary.

**Eg:**

d={100:"durga",200:"ravi",300:"shiva"}

print(d)

d.clear()

print(d)

**Output**

{100: 'durga', 200: 'ravi', 300: 'shiva'}

{}

**del d**

Deletes complete dictionary. Now we cannot access d.

**Eg:**

d={100:"durga",200:"ravi",300:"shiva"}

print(d)

del d

print(d)

**Output**

{100: 'durga', 200: 'ravi', 300: 'shiva'}

NameError: name 'd' is not defined

**Important functions of dictionary:**

**1. dict():**

To create a dictionary

d=dict() ===>It creates empty dictionary

d=dict({100:"sreenu",200:"ashwika"}) ==>It creates dictionary with specified elements

**2. len()**

Returns the number of items in the dictionary

**3. clear():**

To remove all elements from the dictionary

**4. get():**

To get the value associated with the key

d.get(key)

If the key is available then returns the corresponding value otherwise returns None. It won’t raise any error.

d.get(key,defaultvalue)

If the key is available then returns the corresponding value otherwise returns default value.

**Eg:**

d={100:"durga",200:"ravi",300:"shiva"}

print(d[100]) ==>durga

print(d[400]) ==>KeyError:400

print(d.get(100)) ==durga

print(d.get(400)) ==>None

print(d.get(100,"Guest")) ==durga

print(d.get(400,"Guest")) ==>Guest

**5. pop():**

d.pop(key)

It removes the entry associated with the specified key and returns the corresponding value

If the specified key is not available then we will get KeyError

**Eg:**

d={100:"sreenu",200:"ashwika",300:"nitiksha"}

print(d.pop(100))

print(d)

print(d.pop(400))

**Output**

sreenu

200: 'ashwika', 300: 'nitiksha'}

KeyError: 400

**6. popitem():**

It removes an arbitrary item(key-value) from the dictionaty and returns it.

**Eg:**

d={100:"sreenu",200:"ashwika",300:"nitiksha"}

print(d)

print(d.popitem())

print(d)

**Output**

{100: 'sreenu', 200: ‘ashwika', 300: 'nitiksha'}

(300, 'nitiksha')

{100: 'sreenu', 200: 'ashwika'}

If the dictionary is empty then we will get KeyError

d={}

print(d.popitem()) ==>KeyError: 'popitem(): dictionary is empty'

**7. keys():**

It returns all keys associated with dictionary

**Eg:**

d={100:"sreenu",200:"ashwika",300:"nitiksha"}

print(d.keys())

for k in d.keys():

print(k)

**Output**

dict\_keys([100, 200, 300])

100

200

300

**8. values():**

It returns all values associated with the dictionary

d={100:"sreenu",200:"ashwika",300:"nitiksha"}

print(d.values())

for v in d.values():

print(v)

**Output**

dict\_values(['sreenu', 'ashwika', 'nitiksha'])

sreenu

ashwika

nitiksha

**9. items():**

It returns list of tuples representing key-value pairs.

[(k,v),(k,v),(k,v)]

**Eg:**

d={100:"sreenu",200:"ashwika",300:"nitiksha"}

for k,v in d.items():

print(k,"--",v)

**Output**

100 -- sreenu

200 -- ashwika

300 -- nitiksha

**10. copy():**

To create exactly duplicate dictionary(cloned copy)

d1=d.copy();

11. setdefault():

d.setdefault(k,v)

If the key is already available then this function returns the corresponding value.

If the key is not available then the specified key-value will be added as new item to the dictionary.

d={100:"sreenu",200:"ashwika",300:"nitiksha"}

print(d.setdefault(400,"pavan"))

print(d)

print(d.setdefault(100,"sachin"))

print(d)

**Output**

pavan

100: 'sreenu', 200: 'ashwika', 300: 'nitiksha', 400: 'pavan'}

sreenu

{100: 'sreenu', 200: 'ashwika', 300: 'nitiksha', 400: 'pavan'}

update():

d.update(x)

All items present in the dictionary x will be added to dictionary d