Strings in Python

Strings: string slices, immutability, string functions and methods, string module

Strings

- String is a sequence of characters.
- String may contain alphabets, numbers and special characters.
- Usually strings are enclosed within a single quotes and double quotes.
- Strings is immutable in nature.
- **Example**:

  a='hello world'

  b="Python"

Inbuilt String functions

- Python mainly contains 3 inbuilt string functions.
- They are
  - len()
  - max()
  - min()
- len()- Find out the length of characters in string
- min()- Smallest value in a string based on ASCII values
- max()- Largest value in a string based on ASCII values

What is ASCII values

L.O :Explain the function of ASCII code.

ASCII

- HOW ASCII WORKS IN A COMPUTER SYSTEM?

Step 1.
The user presses the capital letter T (shift+T key) on the keyboard.

Step 2.
An electronic signal for the capital letter T is sent to the system unit.

Step 3.
The signal for the capital letter T is converted to its ASCII binary code (01010100) and is stored in memory for processing.

Step 4.
After processing, the binary code for the capital letter T is converted to an image, and displayed on the output device.

Decimal	Character
65	A
66	B
67	C
68	D
69	E
70	F
71	G
72	H
73	I
74	J
75	K
76	L
77	M
78	N
79	O
80	P
81	Q
82	R
83	S
84	T
85	U
86	V
87	W
88	X
89	Y
90	Z

Decimal	Character
97	a
98	b
99	c
100	d
101	e
102	f
103	g
104	h
105	i
106	j
107	k
108	l
109	m
110	n
111	o
112	p
113	q
114	r
115	s
116	t
117	u
118	v
119	w
120	x
121	y
122	z

Example for Inbuilt string functions

```
name=input("Enter Your name:")
print("Welcome",name)

print("Length of your name:",len(name))
print("Maximum value of chararacter in your name",
max(name))
print("Minimum value of character in your name",min(name))
```

OUTPUT

Enter Your name:PRABHAKARAN

Welcome PRABHAKARAN

Length of your name: 11

Maximum value of chararacter in your name R

Minimum value of character in your name A

Strings Concatenation

- The + operator used for string concatenation.

Example:

a="Hai"

b="how are you"

c=a+b

print(c)

```
Haihow are you
>>> a="Hai"
>>> b=" how are you"
>>> c=a+b
>>> print(c)
Hai how are you
```

```
>>> a="Hai"
>>> b=' how are you'
>>> c=a+b
>>> print(c)
Hai how are you
```

Operators on String

- The Concatenate strings with the “*” operator can create multiple concatenated copies.
- Example:

```
>>> print("Python"*10)
PythonPythonPythonPythonPythonPythonPythonPythonPythonPython
```

String Slicing

- Slicing operation is used to return/select/slice the particular substring based on user requirements.
- A segment of string is called slice.
- **Syntax**: string_variablename [start:end]

String Slice example

s="Hello"

H	e	l	l	o
0	1	2	3	4
-5	-4	-3	-2	-1

```
>>> s="hello"
>>> s[1:4]
'ell'
>>> s[1:]
'ello'
>>> s[:]
'hello'
>>> s[1:100]
'ello'
>>> s[-1]
'o'
>>> s[::]
'hello'
>>> s[:-3]
'he'
```

Strings are immutable

- Strings are immutable character sets.
- Once a string is generated, you cannot change any character within the string.

```
>>> a="python program"
>>> a[0]
'p'
>>> a[0]="b"
Traceback (most recent call last):
  File "<pyshell#16>", line 1, in <module>
    a[0]="b"
TypeError: 'str' object does not support item assignment
>>> a[0]
'p'
```

String Comparision

- We can compare two strings using comparision operators such as ==, !=, <,<=,>, >=
- Python compares strings based on their corresponding ASCII values.

Example of string comparision

```
str1="green"
str2="black"
print("Is both Equal:", str1==str2)
print("Is str1> str2:", str1>str2)
print("Is str1< str2:", str1<str2)
```

OUTPUT:

Is both Equal: False
Is str1> str2: True
Is str1< str2: False

String formatting operator

- String formatting operator % is unique to strings.
- **Example:**

```
print("My name is %s and i secured %d marks in python" % ("Arbaz",92))
```

- **Output:**

My name is Arbaz and i secured 92 marks in python

String functions and methods

len()	min()	max()	isalnum()	isalpha()
isdigit()	islower()	isuppe()	isspace()	isidentifier()
endswith()	startswith()	find()	count()	capitalize()
title()	lower()	upper()	swapcase()	replace()
center()	ljust()	rjust()	center()	isstrip()
rstrip()	strip()			

i) Converting string functions

captitalize()	Only First character capitalized
lower()	All character converted to lowercase
upper()	All character converted to uppercase
title()	First character capitalized in each word
swapcase()	Lower case letters are converted to Uppercase and Uppercase letters are converted to Lowercase
replace(old,new)	Replaces old string with nre string

Program:

```
str=input("Enter any string:")
print("String Capitalized:", str.capitalize())
print("String lower case:", str.lower())
print("String upper case:", str.upper())
print("String title case:", str.title())
print("String swap case:", str.swapcase())
print("String replace  case:",str.replace("python","python programming"))
```

Output:

```
Enter any string: Welcome to python
String Capitalized: Welcome to python
String lower case: welcome to python
String upper case: WELCOME TO PYTHON
String title case: Welcome To Python
String swap case: wELCOME TO PYTHON
String replace  case: Welcome to python programming
```

ii)Formatting String functions

center(width)	Returns a string centered in a field of given width
ljust(width)	Returns a string left justified in a field of given width
rjust(width)	Returns a string right justified in a field of given width
format(items)	Formats a string

Program:

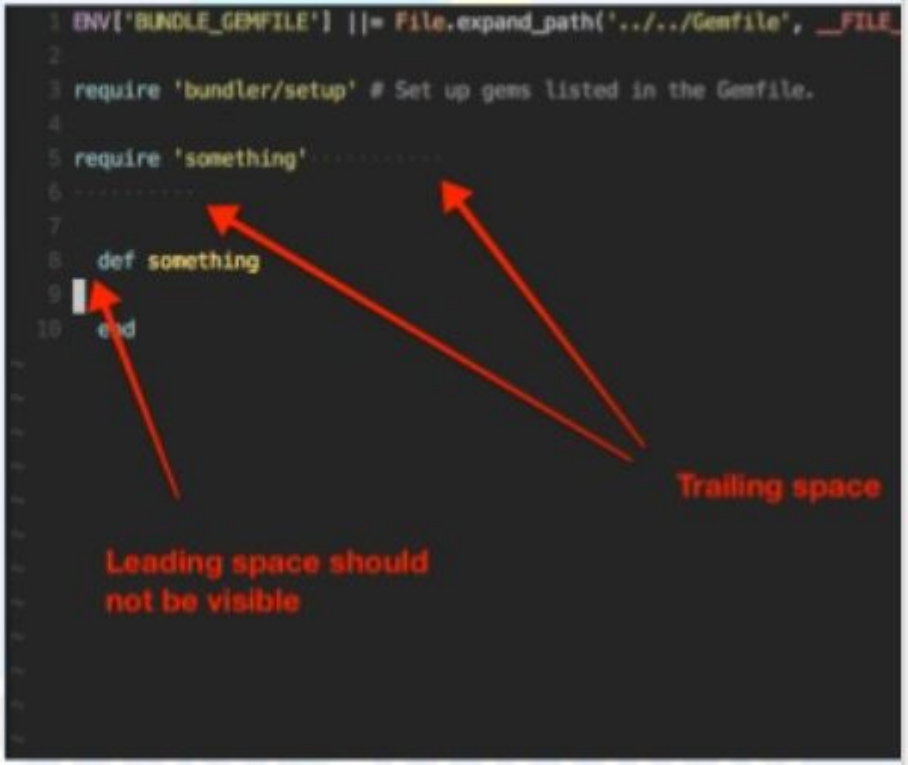
```
a=input("Enter any string:")
print("Center alignment:", a.center(20))
print("Left alignment:", a.ljust(20))
print("Right alignment:", a.rjust(20))
```

Output:

```
Enter any string:welcome
Center alignment:       welcome
Left alignment: welcome
Right alignment:                welcome
```

iii) Removing whitespace characters

lstrip()	Returns a string with leading whitespace characters removed
rstrip()	Returns a string with trailing whitespace characters removed
strip()	Returns a string with leading and trailing whitespace characters removed



Program

```
a=input("Enter any string:")
print("Left space trim:",a.lstrip())
print("Right space trim:",a.rstrip())
print("Left and right trim:",a.strip())
```

Output:

```
Enter any string:        welcome
Left space trim: welcome
Right space trim:        welcome
Left and right trim: welcome
```

iv) Testing String/Character

isalnum()	Returns true if all characters in string are alphanumeric and there is atleast one character
isalpha()	Returns true if all characters in string are alphabetic
isdigit()	Returns true if string contains only number character
islower()	Returns true if all characters in string are lowercase letters
isupper()	Returns true if all characters in string are uppercase letters
isspace()	Returns true if string contains only whitespace characters.

Program

```
a=input("Enter any string:")
print("Alphanumeric:",a.isalnum())
print("Alphabetic:",a.isalpha())
print("Digits:",a.isdigit())
print("Lowecase:",a.islower())
print("Upper:",a.isupper())
```

Output:

```
Enter any string:python
Alphanumeric: True
Alphabetic: True
Digits: False
Lowecase: True
Upper: False
```

v) Searching for substring

Endswith()	Returns true if the strings ends with the substring
Startswith()	Returns true if the strings starts with the substring
Find()	Returns the lowest index or -1 if substring not found
Count()	Returns the number of occurrences of substring

Program

```
a=input("Enter any string:")
print("Is string ends with thon:", a.endswith("thon"))
print("Is string starts with good:", a.startswith("good"))
print("Find:", a.find("ython"))
print("Count:", a.count("o"))
```

Output:

```
Enter any string : welcome to python
Is string ends with thon: True
Is string starts with good: False
Find: 12
Count: 3
```

String Modules

- String module contains a number of functions to process standard Python strings
- **Mostly used string modules:**

  string.upper()
  string.upper()
  string.split()
  string.join()
  string.replace()
  string.find()
  string.count()

Example

```
import string
text="Monty Python Flying Circus"
print("Upper:", string.upper(text))
print("Lower:", string.lower(text))
print("Split:", string.split(text))
print("Join:", string.join(string.split(test),"+"))
print("Replace:", string.replace(text,"Python", "Java"))
print("Find:", string.find(text,"Python"))
print("Count", string.count(text,"n"))
```

Output

Upper: “MONTY PYTHON FLYING CIRCUS”

Lower: “monty python flying circus”

Split: [‘Monty’, ‘Python’, ‘Flying’, ‘Circus’]

Join : Monty+Python+Flying+Circus

Replace: Monty Java Flying Circus

Find: 7

Count: 3

Python Tuples

- **Tuples** are very similar to lists, except that they are immutable (they cannot be changed).
- They are created using **parentheses**, rather than square brackets.

Advantages of Tuple over List

- We generally use tuple for heterogeneous (different) datatypes and list for homogeneous (similar) datatypes.
- Since tuple are immutable, iterating through tuple is faster than with list. So there is a slight performance boost.
- Tuples that contain immutable elements can be used as key for a dictionary. With list, this is not possible.
- If you have data that doesn't change, implementing it as tuple will guarantee that it remains write-protected.

Creating a Tuple

- A tuple is created by placing all the items (elements) inside a parentheses (), separated by comma.
- The parentheses are optional but is a good practice to write it.
- A tuple can have any number of items and they may be of different types (integer, float, list, string etc.).

```
# empty tuple
my_tuple = ()
print(my_tuple)

# tuple having integers
my_tuple = (1, 2, 3)
print(my_tuple)

# tuple with mixed datatypes
my_tuple = (1, "Hello", 3.4)
print(my_tuple)

# nested tuple
my_tuple = ("mouse", [8, 4, 6], (1, 2, 3))
print(my_tuple)

# tuple can be created without parentheses
# also called tuple packing
my_tuple = 3, 4.6, "dog"
print(my_tuple)
```

```
()
(1, 2, 3)
(1, 'Hello', 3.4)
('mouse', [8, 4, 6], (1, 2, 3))
(3, 4.6, 'dog')
```

- Creating a tuple with one element is a bit tricky.
- Having one element within parentheses is not enough. We will need a trailing comma to indicate that it is in fact a tuple.

```
# only parentheses is not enough
my_tuple = ("hello")
print(type(my_tuple))

# need a comma at the end
my_tuple = ("hello",)
print(type(my_tuple))

# parentheses is optional
my_tuple = "hello",
print(type(my_tuple))
```

```
<class 'str'>
<class 'tuple'>
<class 'tuple'>
```

Accessing Elements in a Tuple

- You can access the values in the tuple with their index, just as you did with lists:
- nested tuple are accessed using nested indexing
- Negative indexing can be applied to tuples similar to lists.
- We can access a range of items in a tuple by using the slicing operator
- Trying to reassign a value in a tuple causes a TypeError.

```
marks = (23,45,32)
print(marks[0])
print(marks[2])
```

```
23
32
```

```
# nested tuple
n_tuple = ("SIKANDER", [8, 4, 6], (1, 2, 3))

print(n_tuple[0])
print(n_tuple[1])

print(n_tuple[0][0])
print(n_tuple[1][0])
```

```
SIKANDER
[8, 4, 6]
S
8
```

Changing a Tuple

- Unlike lists, tuples are immutable.
- This means that elements of a tuple cannot be changed once it has been assigned. But, if the element is itself a mutable datatype like list, its nested items can be changed.

```
n_tuple = ("SIKANDER", [8, 4, 6], (1, 2, 3))
print(n_tuple)

n_tuple[1][1] = 23
print(n_tuple)
```

```
('SIKANDER', [8, 4, 6], (1, 2, 3))
('SIKANDER', [8, 23, 6], (1, 2, 3))
```

Similar to List,

- We can use + operator to combine two tuples. This is also called **concatenation**.
- We can also **repeat** the elements in a tuple for a given number of times using the * operator.
- Both + and * operations result into a new tuple.

```
# Concatenation
print((1, 2, 3) + (4, 5, 6))

# Repeat
print(("Repeat",) * 3)
```

```
(1, 2, 3, 4, 5, 6)
('Repeat', 'Repeat', 'Repeat')
```

Deleting a Tuple

- We cannot change the elements in a tuple. That also means we cannot delete or remove items from a tuple.
- But deleting a tuple entirely is possible using the keyword del.

```
my_tuple = ('p','r','o','g','r','a','m','i','z')

del my_tuple[3]
# TypeError: 'tuple' object doesn't support item deletion

# can delete entire tuple
del my_tuple

# NameError: name 'my_tuple' is not defined
my_tuple
```

Python Tuple Methods

- Methods that add items or remove items are not available with tuple. Only the following two methods are available.

Method	Description
count(x)	Return the number of items that is equal to x
index(x)	Return index of first item that is equal to x

```
my_tuple = ('a','p','p','l','e',)

print('Total count of element p is ' , my_tuple.count('p'))

print('Index of l is' , my_tuple.index('l'))
```

```
Total count of element p is  2
Index of l is 3
```

Tuple Membership Test

- We can test if an item exists in a tuple or not, using the keyword in.

```
my_tuple = ('a','p','p','l','e',)

print('a' in my_tuple)

print('b' in my_tuple)

print('g' not in my_tuple)
```

```
True
False
True
```

Iterating Through a Tuple

- Using a for loop we can iterate though each item in a tuple.

```
names = ('Sikander', 'Sharath','John','Kate')
for name in names:
     print('Hello ',name)
```

```
Hello  Sikander
Hello  Sharath
Hello  John
Hello  Kate
```

Built-in Functions with Tuple

Function	Description
all()	Return True if all elements of the tuple are true (or if the tuple is empty).
any()	Return True if any element of the tuple is true. If the tuple is empty, return False.
enumerate()	Return an enumerate object. It contains the index and value of all the items of tuple as pairs.
len()	Return the length (the number of items) in the tuple.
max()	Return the largest item in the tuple.
min()	Return the smallest item in the tuple
sorted()	Take elements in the tuple and return a new sorted list (does not sort the tuple itself).
sum()	Retrun the sum of all elements in the tuple.
tuple()	Convert an iterable (list, string, set, dictionary) to a tuple.

INTRODUCTION

- List is a sequence of values called items or elements.
- The elements can be of any data type.
- The list is a most versatile data type available in Python which can be written as a list of comma-separated values (items) between square brackets.
- List are mutable, meaning, their elements can be changed.

CREATING A LIST?

- In Python programming, a list is created by placing all the items (elements) inside a square bracket [ ], separated by commas.
- It can have any number of items and they may be of different types (integer, float, string etc.).

Method -1 without constructor

```
# empty list
my_list = []
# list of integers
my_list = [1, 2, 3]
# list with mixed datatypes
my_list = [1, "Hello", 3.4]
# nested list
my_list = [“welcome", [8, 4, 6]]
```

Method-2 using list constructor

```
# empty list
my_list = list()
# list of integers
my_list = list([1, 2, 3])
```

ACCESS ITEMS FROM A LIST

- It can be access in several ways
- Use the index operator [] to access an item in a list. Index starts from 0. So, a list having 5 elements will have index from 0 to 4.
- Example:

```
list = ['p','r','o','b','e']

print(list[2]) #Positive Indexing

print(list[-2]) #Negative Indexing
```

Output

```
o
b
```

ACCESSING THE ELEMENTS OF A LIST

- Index operator [] is used to access an item in a list. Index starts from 0
- marks=[90,80,50,70,60]
- print(marks[0])

  Output: 90
- Nested list:
- my_list = [“welcome", [8, 4, 6]]
- Print(marks[1][0])

  Output: 8

ACCESSING - NEGATIVE INDEXING

- Python allows negative indexing for its sequences. The index of -1 refers to the last item, -2 to the second last item and so on

```
my_list = ['p','r','o','b','e']

# Output: e
print(my_list[-1])

# Output: p
print(my_list[-5])
```

HOW TO CHANGE OR ADD ELEMENTS TO A LIST?

#Change Elements

= operator with index

```
>>> marks=[90,60,80]
>>> print(marks)
[90, 60, 80]
>>> marks[1]=100
>>> print(marks)
[90, 100, 80]
```

#Add Elements

- add one item to a list using append() method
- add several items using extend()
- insert one item at a desired location by using the method insert()

```
>>> marks.append(50)
>>> print(marks)
[90, 100, 80, 50]

>>> marks.extend([60,80,70])
>>> print(marks)
[90, 100, 80, 50, 60, 80, 70]

>>> marks.insert(3,40)
>>> print(marks)
[90, 100, 80, 40, 50, 60, 80, 70]
```

HOW TO DELETE OR REMOVE ELEMENTS FROM A LIST?

- delete one or more items from a list using the keyword del.
- It can even delete the list entirely.

```
>>> print(marks)
[90, 100, 80, 40, 50, 60, 80, 70]

>>> del marks[6]
>>> print(marks)
[90, 100, 80, 40, 50, 60, 70]
>>> del marks
>>> print(marks)
Name Error: name 'marks' is not defined
```

- clear() method to empty a list.

```
>>> marks.clear()
>>> print(marks)
[]
```

- remove() method to remove the given item

```
>>> marks=[90,60,80]
>>> marks.remove(80)
>>> print(marks)
[90, 60]
>>> marks.remove(100)
Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    marks.remove(100)
ValueError: list.remove(x): x not in list
```

- pop() method to remove an item at the given index.

```
>>> marks=[100,20,30]
>>> marks.pop()
30
>>> print(marks)
[100, 20]
>>> >>> marks.pop(0)
100
>>> print(marks)
[20]
```

delete items in a list by assigning an empty list to a slice of elements.

```
marks=[100,20,30]
>>> marks[1:2]=[]
>>> print(marks)
[100, 30]
```

- >>> marks.extend([40,50,60,70])
- >>> marks
- [90, 40, 50, 60, 70]
- >>> marks.pop()
- 70
- >>> marks.pop()
- 60

LIST OPERATORS

- Slicing [::] (i.e) list[start:stop:step]
- Concatenation = +
- Repetition= *
- Membership = in
- Identity = is

SLICE LISTS

- Accessing a range of items in a list by using the slicing operator [ ] using (colon :).
- Slicing can be best visualized by considering the index to be between the elements.
- ✓ Example:

```
list = ['p','r','o','b','e']

print(list[0:4]) #Positive

print(list[-2:-1]) # Negative
```

Output

```
['p', 'r', 'o', 'b']
['b']
```

PYTHON LIST METHODS

append() - Add an element to the end of the list
extend() - Add all elements of a list to the another list
insert() - Insert an item at the defined index
remove() - Removes an item from the list
pop() - Removes and returns an element at the given index
clear() - Removes all items from the list
index() - Returns the index of the first matched item
count() - Returns the count of number of items passed as an argument
sort() - Sort items in a list in ascending order
reverse() - Reverse the order of items in the list
copy() - Returns a shallow copy of the list

LIST METHODS

- append() - Add an element to the end of the list.
- count() - Returns the count of number of items passed as an argument.
- extend() - Add all elements of a list to the another list.
- index() - Returns the index of the first matched item.
- insert() - Insert an item at the defined index.
- pop() - Removes and returns an element at the given index.
- copy() - Returns a shallow copy of the list
- remove() - Removes an item from the list.
- reverse() - Reverse the order of items in the list.
- sort() - Sort items in a list in ascending order.

APPEND() METHODS

- The append() method adds a single item to the existing list.
- The append() method only modifies the original list. It doesn't return any value.
- The append() method takes a single *item* and adds it to the end of the list.
- The *item* can be numbers, strings, another list, dictionary etc.
- Syntax

```
list.append(item)
```

APPEND() METHODS - PROGRAMS

Example 1: Adding Element to a List

```
list = ['hi', 'hello']
print('old list: ', list)
list.append('welcome!')
print('Updated list: ', list)
```

Output

```
old list: ['hi', 'hello']
Updated list: ['hi', 'hello','welcome!']
```

Example 2: Adding List to a List

```
list = ['hi', 'hello']
print('old list: ', list)
list1=['welcome']
list.append(list1)
print('Updated list: ', list)
```

Output

```
old list: ['hi', 'hello']
Updated list: ['hi', 'hello', ['welcome']]
```

COUNT() METHODS

- Count() method counts how many times an element has occurred in a list and returns it.
- The count() method takes a single argument:
  - **element** - element whose count is to be found.
- The count() method returns the number of occurrences of an element in a list.
- Syntax

```
list.count(element)
```

COUNT() METHODS - PROGRAMS

Example: Count the occurrence of an element in the list

```
list = ['h','i','h','o','w','y','o','u','!','!']

print('The count of i is:',list.count('i'))

print('The count of o is:',list.count('o'))
```

Output

```
The count of i is: 1
The count of o is: 2
```

Example 2: Count the occurrence of tuple and list inside the list

```
list = ['a', ('h', 'i'), ('a', 'b'), [8, 4]]

count = list.count(('a', 'b'))

print("The count of ('a', 'b') is:", count)

count = list.count([3, 4])

print("The count of [3, 4] is:", count)
```

Output

```
The count of ('a', 'b') is: 1
The count of [3, 4] is: 0
```

EXTEND() METHODS

- The extend() extends the list by adding all items of a list to the end.
- Extend() method takes a single argument (a list) and adds it to the end.
- This method also add elements of other native data_types ,like tuple and set to the list,
- Syntax

```
list1.extend(list2)
```

Note : The elements of *list2* are added to the end of *list1*.

```
list.extend(list(tuple_type))
```

Note : The elements of *tuple* are added to the end of *list1*.

EXTEND() METHODS - PROGRAMS

Example 1: Using extend() Method

```
language = ['C', 'C++', 'Python']

language1 = ['JAVA', 'COBOL']

language.extend(language1)

print('Language List: ', language)
```

Output

```
Language List: ['C', 'C++', 'Python', 'JAVA', 'COBOL']
```

EXTEND() METHODS - PROGRAMS

Example 2: Add Elements of Tuple and Set to List

```
language = ['C', 'C++', 'Python']
language_tuple = ['JAVA', 'COBOL']
language_set = {'.Net', 'C##'}
language.extend(language_tuple)
print('New Language List: ', language)
language.extend(language_set)
print('Newest Language List: ', language)
```

Output

```
New Language List: ['C', 'C++', 'Python', 'JAVA', 'COBOL'] Newest
Language List: ['C', 'C++', 'Python', 'JAVA', 'COBOL',
'.Net', 'C##']
```

INDEX( ) METHODS

- Index() method search and find given element in the list and returns its position.
- However, if the same element is present more than once, index() method returns its smallest/first position.
- Index in Python starts from 0 not 1.
- The index() method returns the index of the element in the list.
- The index method takes a single argument:
  - **element** - element that is to be searched.
- Syntax

```
list.index(element)
```

INDEX( ) METHODS - PROGRAMS

Example 1: Find position of element present in the list and not present in the list

```
list = ['h','i','h','o','w','y','o','u','!','!']

print('The count of i is:',list.index('i'))

print('The count of o is:',list.index('o'))

print('The count of o is:',list.index('z'))
```

Output

```
The count of i is: 1

The count of o is: 3

ValueError: 'z' is not in list
```

INSERT( ) METHODS

- The insert() method inserts the element to the list at the given index.
- The insert() function takes two parameters:
  - **index** - position where element needs to be inserted
  - **element** - this is the element to be inserted in the list
- The insert() method only inserts the element to the list. It doesn't return any value.
- Syntax

```
list.insert(index, element)
```

INSERT( ) METHODS - PROGRAMS

Example 1: Inserting Element to List

```
vowels=['a','i','o','u']

print("old list =",vowels)

vowels.insert(1,'e')

print("new list =",vowels)
```

Output

```
old list = ['a', 'i', 'o', 'u']
new list = ['a', 'e', 'i', 'o', 'u']
```

INSERT( ) METHODS - PROGRAMS

Example 2: **Inserting a Tuple (as an Element) to the List**

```
list = [{1, 2}, [5, 6, 7]]

print('old List: ', list)

number_tuple = (3, 4)

list.insert(1, number_tuple)

print('Updated List: ', list)
```

Output

```
old List: [{1, 2}, [5, 6, 7]]
Updated List: [{1, 2}, (3, 4), [5, 6, 7]]
```

POP( ) METHODS

- The pop() method takes a single argument (index) and removes the element present at that index from the list.
- If the index passed to the pop() method is not in the range, it throws **IndexError: pop index out of range** exception.
- The parameter passed to the pop() method is optional. If no parameter is passed, the default index **-1** is passed as an argument.
- Also, the pop() method removes and returns the element at the given index and updates the list.
- Syntax

```
list.pop(index)
```

POP( ) METHODS - PROGRAMS

Example 1: Pop Element at the Given Index from the List

```
list = ['Python', 'Java', 'C++', 'French', 'C']

return_value = language.pop(3)

print('Return Value: ', return_value)

print('Updated List: ', language)
```

Output

```
old list = ['Python', 'Java', 'C++', 'French', 'C']
Return Value: French
Updated List: ['Python', 'Java', 'C++', 'C']
```

POP( ) METHODS - PROGRAMS

Example 2: pop() when index is not passed

```
language = ['Python', 'Java', 'C++', 'C']
print('Return Value: ', language.pop())
print('Updated List: ', language)
print('Return Value: ', language.pop(-1))
print('Updated List: ', language)
```

Output

```
Return Value: C
Updated List: ['Python', 'Java', 'C++']
Return Value: C++ Updated List:
['Python', 'Java']
```

COPY( ) / ALIASING METHODS

- The copy() method returns a shallow copy of the list.
- A list can be copied with = operator.

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

new_list = old_list
```

- The problem with copying the list in this way is that if you modify the *new_list*, the *old_list* is also modified.
- Syntax

```
new_list = old_list
```

COPY( ) / ALIASING METHODS - PROGRAMS

Example 1: Copying a List

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

new_list = old_list

new_list.append('a')

print('New List:', new_list )

print('Old List:', old_list )
```

Output

```
New List: [1, 2, 3, 'a']
Old List: [1, 2, 3, 'a']
```

SHALLOW COPY( ) / CLONING METHODS

- We need the original list unchanged when the new list is modified, you can use copy() method.
- This is called **shallow copy**.
- The copy() method doesn't take any parameters.
- The copy() function returns a list. It doesn't modify the original list.
- Syntax

```
new_list = list.copy()
```

SHALLOW COPY( ) / CLONING METHODS - PROGRAMS

Example 2: Shallow Copy of a List Using Slicing

```
list = [1,2,4]
new_list = list[:]
new_list.append(5)
print('Old List: ', list)
print('New List: ', new_list)
```

Output

```
Old List: [1, 2, 4]
New List: [1, 2, 4, 5]
```

REMOVE( ) METHODS

- The remove() method searches for the given element in the list and removes the first matching element.
- The remove() method takes a single element as an argument and removes it from the list.
- If the element(argument) passed to the remove() method doesn't exist, valueError exception is thrown.
- Syntax

```
list.remove(element)
```

REMOVE( ) METHODS - PROGRAMS

Example 1 :Remove Element From The List

```
list = [5,78,12,26,50]

print('Original list: ', list)

list.remove(12)

print('Updated list elements: ', list)
```

Output

```
Original list:  [5, 78, 12, 26,
50]Updated list:  [5, 78, 26, 50]
```

REVERSE( ) METHODS

- The reverse() method reverses the elements of a given list.
- The reverse() function doesn't return any value. It only reverses the elements and updates the list.
- Syntax

```
list.reverse()
```

REVERSE( ) METHODS - PROGRAM

Example 1 :Reverse a List Using Slicing Operator

```
list = [1,5,8,6,11,55]

print('Original List:', list)

reversed_list = list[::-1]

print('Reversed List:', reversed_list)
```

Output

Original List: [1, 5, 8, 6, 11, 55]

Reversed List: [55, 11, 6, 8, 5, 1]

Example 2: Reverse a List

```
list = [1,5,8,6,11,55]
print('Original List:', list)
list.reverse()
print('Reversed List:', list)
```

Output

Original List: [1, 5, 8, 6, 11, 55]

Reversed List: [55, 11, 6, 8, 5, 1]

SORT( ) METHODS

- The sort() method sorts the elements of a given list.
- The sort() method sorts the elements of a given list in a specific order - Ascending or Descending.
- Syntax

```
list.sort()
```

REVERSE( ) METHODS - PROGRAM

Example 1 :Sort a given List in ascending order

```
list = [1,15,88,6,51,55]

print('Original List:', list)

list.sort()

print('sorted List:', list)
```

Output

```
Original List: [1, 15, 88, 6, 51, 55]
sorted List: [1, 6, 15, 51, 55, 88]
```

Example 2 : Sort the list in Descending order

```
list = [1,15,88,6,51,55]

print('Original List:', list)

list.sort(reverse=True)

print('sorted List:', list)
```

Output

```
Original List: [1, 15, 88, 6, 51, 55]
sorted List: [88, 55, 51, 15, 6, 1]
```

BUILT-IN FUNCTIONS WITH LIST

Function	Description
all()	Return True if all elements of the list are true (or if the list is empty).
any()	Return True if any element of the list is true. If the list is empty, return False.
enumerate()	Return an enumerate object. It contains the index and value of all the items of list as a tuple.
len()	Return the length (the number of items) in the list.
list()	Convert an iterable (tuple, string, set, dictionary) to a list.
max()	Return the largest item in the list.
min()	Return the smallest item in the list
sorted()	Return a new sorted list (does not sort the list itself).
sum()	Return the sum of all elements in the list.
copy()	
reversed()	

WHAT IS DICTIONARY?

A dictionary is like a list, but more in general. In a list, index value is an integer, while in a dictionary index value can be any other data type and are called keys. The key will be used as a string as it is easy to recall. A dictionary is an extremely useful data storage construct for storing and retrieving all key value pairs, where each element is accessed (or indexed) by a unique key. However, dictionary keys are not in sequences and hence maintain no left-to right order.

KEY VALYE PAIR

We can refer to a dictionary as a mapping between a set of indices (which are called keys) and a set of values. Each key maps a value. The association of a key and a value is called a key-value pair.

Syntax:

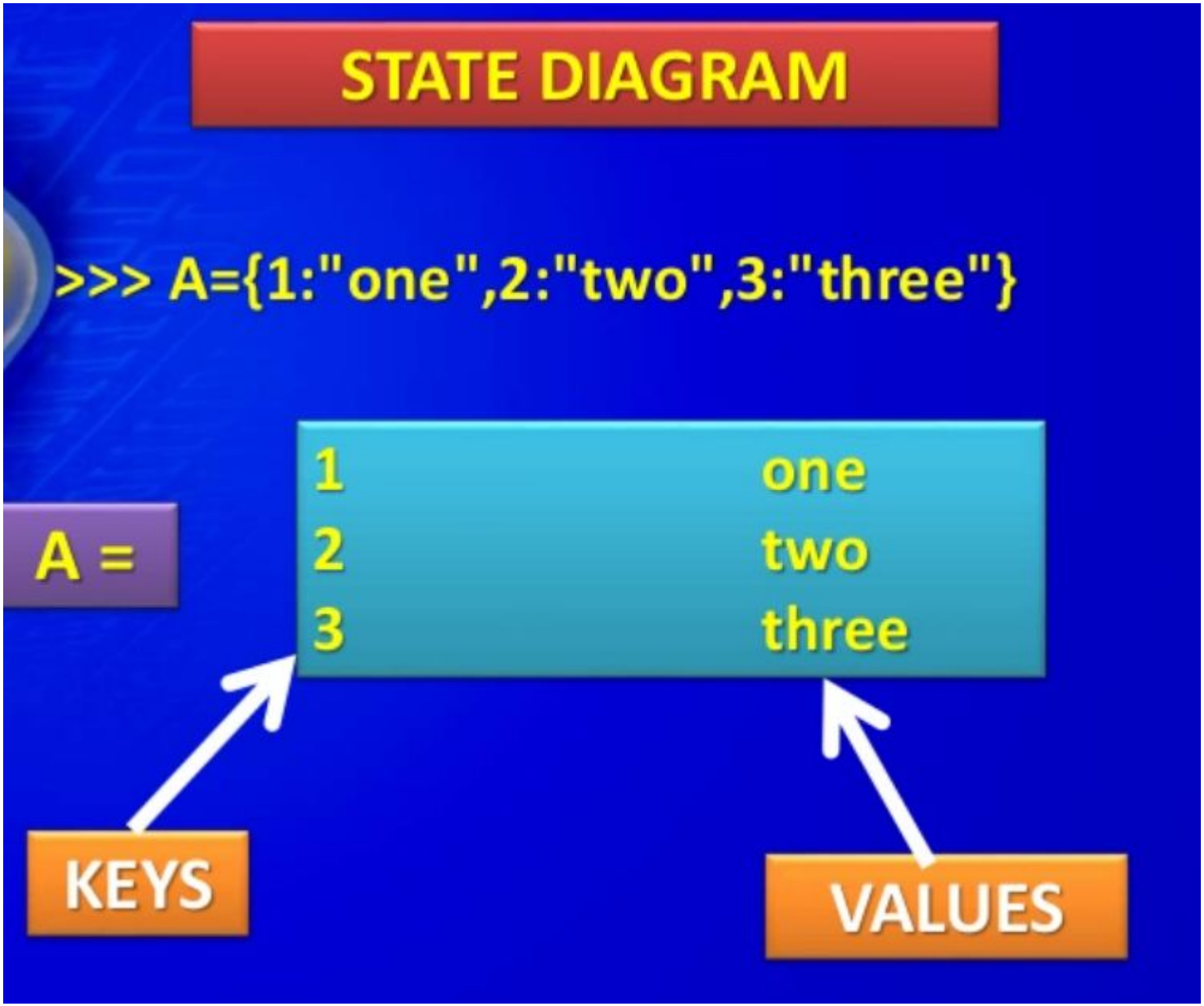
```
my_dict = {'key1': 'value1','key2': 'value2','key3': 'value3'...'keyn': 'valuen'}
```

DICTIONARIES

- ✓Curley brackets are used to represent a dictionary.
- ✓Each pair in the dictionary is represented by a key and value separated by a colon.
- ✓Multiple pairs are separated by vcomas

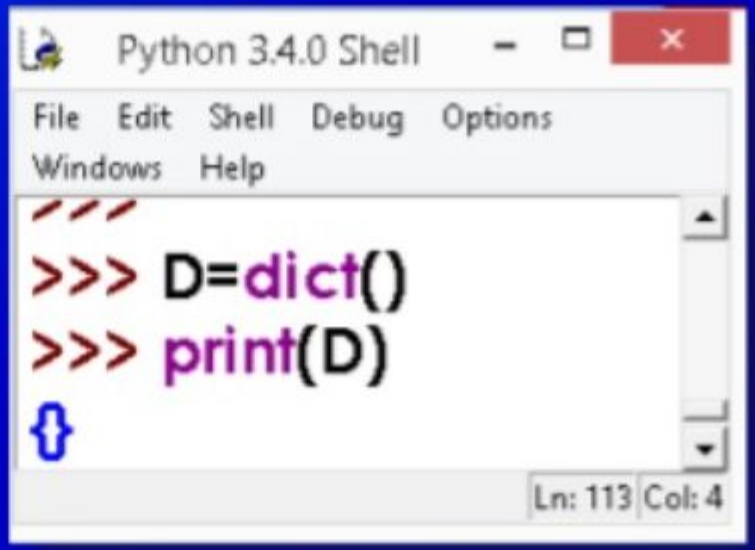
DICTIONARIES

- ✓A dictionary is an unordered collection of key-value pairs.
- ✓A dictionary has a length, specifically the numb er of keyvalue pairs.
- ✓A dictionary provides fast look up by key.
- ✓The keys must be immutable object types.


STATE DIAGRAM
>>> A={1:"one",2:"two",3:"three"}
A =
1 one
2 two
3 three
KEYS
VALUES

CREATING DICTIONARAY – dict()

The function dict ( ) is used to create a new dictionary with no items. This function is called built-in function. We can also create dictionary using {}.



CREATING AND TRAVERSING DICTIONARAY

Python 3.4.0: dictex.py - C:/Python34/dictex.py

File Edit Format Run Options Windows Help

```
def Creating_Dictionary():
    device = {'Four': 'scanner', 'three': 'printer', 'two': 'Mouse', 'one': 'keyboard'}
    for i in device:
        print(device[i])
Creating_Dictionary()
```

Ln: 7 Col: 0

OUT PUT

Python 3.4...

File Edit Shell Debug
Options Windows Help

```
scanner
keyboard
printer
Mouse
>>>
```

Ln: 129 Col: 4

DICTIONARAY – BUILT IN METHODS

Dictionary Method	Meaning
dict.clear()	Removes all the elements of the dictionary
dict.copy()	Returns (shallow)copy of dictionary.
dict.get(key, default=None)	for key key, returns value or default if key not in dictionary (note that default's default is None)
dict.items()	returns a list of dict's (key, value) tuple pairs

DICTIONARAY – BUILT IN METHODS

Dictionary Method	Meaning
dict.keys()	returns list of dictionary dict's keys
dict.setdefault key, default=None	similar to get(), but will set dict[key]=default if key is not already in dict
dict.update(dict2)	adds dictionary dict2's key-values pairs to dict
dict.values()	returns list of dictionary dict's values

DICTIONARAY – BUILT IN METHODS

Dictionary Method	Meaning
dict.pop()	returns list of dictionary dict's keys
dict.popitem()	similar to get(), but will set dict[key]=default if key is not already in dict

M.Tech - IoT IoT Arch | Google | (51) WhatsApp | Meet - oxw-qhh | MSN India | Breaking

https://meet.google.com/oxw-qhhm-jqb

Meenakshi Srinivasan is presenting

Krishnaraj N and 2 more

17

10:4

class.py - Python session - Visual Studio Code

list.PY | self.py | init.py | inheritance.py

Employee

```
ass Employee():
  def say_hi(self):
      print("hi")

o1 = Employee()
o1.say_hi()
```

TERMINAL 2: Python

```
Microsoft Windows [Version 6.2.9200]
(c) 2012 Microsoft Corporation. All
(c) 2012 Microsoft Corporation. All

C:\Users\meena\Desktop\Python session
on/Python37-32/python.exe "c:/Users/
hi

C:\Users\meena\Desktop\Python sessio
n>
```

Ln 1, Col 18 Spaces: 4 UTF-8 CRLF Python

Class- Collections of objects/Design

Objects-Entities of class

Real world example:

meet.google.com is sharing your screen. Stop sharing Hide

Before creating an object we need to create class.

A Y L M K J J

Raise hand

Class List CC

Turn on captions

Meena is

Enter any notes specific to this class

to search

The _init_ method

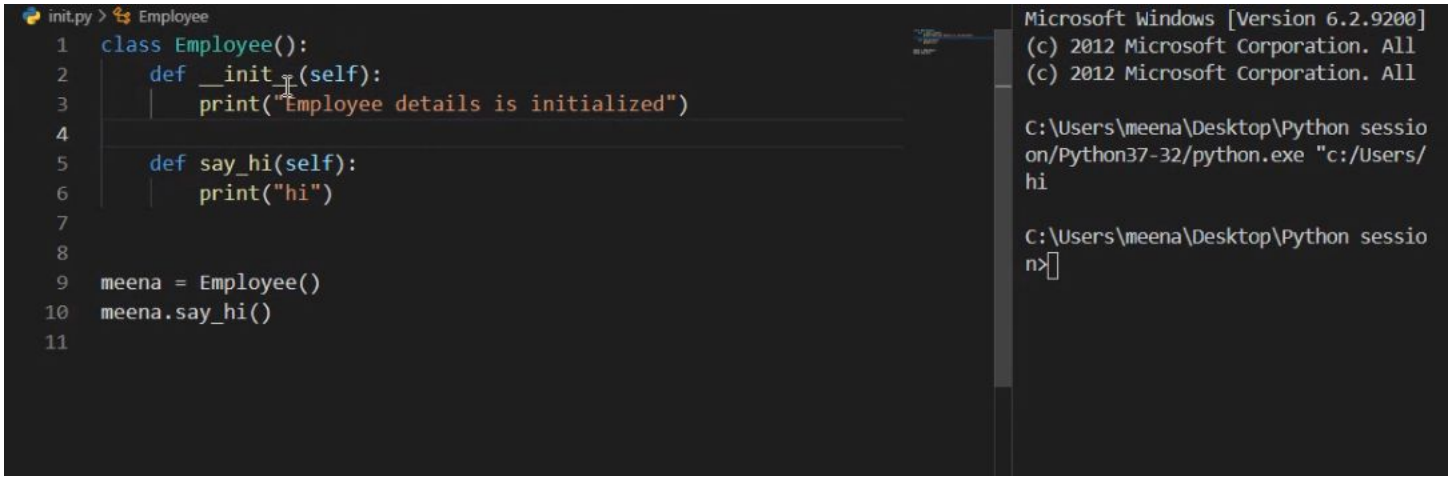
-> It is a special method (underscores)

->To initialize the variables

->The idea behind _init_ is it will be getting called automatically.

In previous example, we have created a method called “config”, but we called this method using object

But here it is called automatically for each object


init.py > Employee
class Employee():
def __init__(self):
print("Employee details is initialized")
def say_hi(self):
print("hi")
meena = Employee()
meena.say_hi()
Microsoft Windows [Version 6.2.9200]
(c) 2012 Microsoft Corporation. All
(c) 2012 Microsoft Corporation. All
C:\Users\meena\Desktop\Python sessio
on/Python37-32/python.exe "c:/Users/
hi
C:\Users\meena\Desktop\Python sessio
n>