Study Material

KVS RO Jaipur

Computer Science

Python Revision: Part 1

Python Fundamentals

1. What is Python:-

·Python is a general-purpose, high level programming language. It was created by Guido van Rossum, and released in 1991.

·In order to communicate with a computer system, we need a computer programming language. This language may be C, C++, Java, Python or any other computer language. Here we will discuss Python.

·Python is inspired by two languages – (i) ABC language which was an optional language of BASIC language. (ii)Modula-3

2. Why Python:-

- Platform Independent (Cross-Platform)
- Free and Open Source
- simple syntax similar to the English language.
- fewer lines than some other programming languages. interpreted Language.
- Python can be treated in a procedural way, an object-orientated way. (OOL and POL.)

3. What can Python do?:-

It is possible to develop various Apps/ Software's with Python like–

- Web development
- Machine learning
- Data Analysis
- Scripting
- Game development.
- Embedded applications
- Desktop applications

4. How To Use Python?:-

4.1 Python can be downloaded from www.python.org. (Standard Installation)

4.2. It is available in two versions-

• Python 2.x • Python 3.x (It is in Syllabus)

OR

Apart from above standard Python. We have various Python IDEs and Code Editors. Some of them are as under:-

(i) Anaconda Distribution:- free and open-source distribution of the Python. having various inbuilt libraries and tools like jupyter Notebook, Spyder etc

(ii) PyCharm	(iii) Canopy
(iv) Thonny	(v) Visual Studio Code
(vi) Eclipse + PyDev	(vii) Sublime Text
(viii)Atom	(ix) GNU Emacs
(x) Vim	(xi) Spyder and many more...

5. Python Interpreter - Interactive And Script Mode :-

We can work in Python in Two Ways:-

(i) Interactive Mode
(ii) Script Mode

(i) Interactive Mode:- it works like a command interpreter as shell prompt works in DOS Prompt or Linux. On each (>>>) symbol we can execute one by one command.

(ii) Script Mode:- it used to execute the multiple instruction (complete program) at once.

6. Python Character Set:-

Character Set is a group of letters or signs which are specific to a language. Character set includes letter, sign, number and symbol.

- Letters: A-Z, a-z
- Digits: 0-9
- Special Symbols: _, +, -, *, /, (, #,@, {, } etc.
- White Spaces: blank space, tab, carriage return, newline, formfeed etc.
- Other characters: Python can process all characters of ASCII and UNICODE.

7. Python Tokens:-

Token is the smallest unit of any programming language. It is also known as Lexical Unit.

Types of token are

i. Keywords
ii. Identifiers (Names)
iii. Literals
iv. Operators
v. Punctuators

7.1 Keywords:-

Keywords are reserved words. Each keyword has a specific meaning to the Python interpreter. As Python is case sensitive, these cannot be used as identifiers, variable name or any other purpose keywords must be written exactly as given below:-

7.2 Identifiers:-

In programming languages, identifiers are names used to identify a variable, function, or other entities in a program.

· The rules for naming an identifier in Python are as follows:

· The name should begin with an uppercase or a lowercase alphabet or an underscore sign (_).

· This may be followed by any combination of characters a-z, A-Z, 0-9 or underscore (_). Thus, an identifier cannot start with a digit.

· It can be of any length. (However, it is preferred to keep it short and meaningful).

· It should not be a keyword or reserved word.

· We cannot use special symbols like !, @, #, $, %, etc. in identifiers.

Legal Identifer Names Example:

myvar my_var _my_var myVar myvar2

Illegal Identifier Names Example:-

2myvar my-var my var=

7.3 Literals/ Values:-

Literals are often called Constant Values. Python permits following types of literals -

a.–String literals - “Rishaan”

b.–Numeric literals – 10, 13.5, 3+5i

c. –Boolean literals – True or False

d.–Special Literal None

e. –Literal collections

7.4 Operators:-

An operator is used to perform specific mathematical or logical operation on values. The values that the operator works on are called operands

Python supports following types of operators -

Unary Operator

- •Unary plus (+)
- •Unary Minus (-)
- •Bitwise complement (~)
- •Logical Negation (not)

Binary Operator

- •Arithmetic operator (+, -, *, /, %, **, //)
- •Relational Operator(<, >, <=, >=, ==,!=)
- •Logical Operator (and, or)
- •Assignment Operator (=, /=, +=, -=, *=, %=, **=, //=)
- •Bitwise Operator (& bitwise and, ^ bitwise xor,|bitwise or)
- •Shift operator (<< shift left, >> shift right)
- •Identity Operator (is, is not)
- •Membership Operator (in, not in)

7.5 Punctuators:-

Punctuators are symbols that are used in programming languages to organize sentence structure, and indicate the rhythm and emphasis of expressions, statements, and program structure.

Common punctuators are: „ “ # $ @ []{}=:;(),

8. Variable:-

· Variables are containers for storing data values.

- Unlike other programming languages, Python has no command for declaring a variable.
- A variable is created the moment you first assign a value to it.

Example:-

```
x = 5
y = "John"
print(x)
print(y)
```

- Variables do not need to be declared with any particular type and can even change type after they have been set.

```
x = 4 # x is of type int
x = "Sally" # x is now of type str
print(x)
```

- String variables can be declared either by using single or double quotes:

```
x = "John"
# is the same as
x = 'John'
```

Variable Names

A variable can have a short name (like x and y) or a more descriptive name (age, carname, total_volume).

Rules for Python variables:

- A variable name must start with a letter or the
- underscore character
- A variable name cannot start with a number
- A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _)
- Variable names are case-sensitive (age, Age and AGE are three different variables)

Output Variable:-

The Python print statement is often used to output variables. To combine both text and a variable, Python uses the , character:

```
x = "awesome"
print("Python is " , x)
```

Display Multiple variable:-

```
x = "awesome“
y=56
print(“Value of x and y is=" ,x,y)
```

9. Data Type:-

Every value belongs to a specific data type in Python. Data type identifies the type of data which a variable can hold and the operations that can be performed on those data.

Data Types in Python:-

- Numbers :- int, float and complex.
- List :- [5, 3.4, "New Delhi", "20C", 45]
- Tuple :- (10, 20, "Apple", 3.4, 'a')
- Strings :- str = 'Hello Friend'
- Set:- {2,6,9}
- Dictionary:- {'Fruit':'Apple', 'Climate':'Cold', 'Price(kg)':120}

Mutable and Immutable data type

Mutable:- The values stored can be changes, Size of object can be changed, e.g. List, Dictionary

Immutable:- The value stored can not be changed, Size can not be changed, e.g. Tuple, String

10. Expression in python:- An expression is a combination of values, i.e., constant , variable and operator.

e.g. Expression 6-3*2+7-1 evaluated as 6

11. Precedence of Arithmetic Operators:- Precedence helps to evaluate an expression.

12. Comments in python:- Comments are non-executable statement of python. It increase the readability and understandability of code.

Types of comment –

i. Single line comment (#) – comments only single line.

e.g. *a=7 # 7 is assigned to variable ‘a’*
print(a) # displaying the value stored in ‘a’

ii. Multi-line comment (“‘…….’”) – Comments multiple line.

e.g. *“‘Program -1*
A program in python to store a value in variable ‘a’ and display the value stored in it.”’
a=7
print(a)

13. Input and output in python:-

input ( ) method - It is used to take input from user.

print ( ) method – It is used to display message or result in output.

Example:-

```
Name = input ("Enter your name : ")
Marks = int(input ("Enter your marks "))
print("Your name is ", Name)
print("You got ", Marks, " marks")
```

14. Debugging:- Debugging is a process of locating and removing errors from program.

Python Control Statements

In any programming language a program may execute sequentially, selectively or iteratively. Every programming language provides constructs to support Sequence, Selection and Iteration. In Python all these construct can broadly categorized in 2 categories.

Conditional Control Construct

(Selection, teration)

Un-Conditional Control Construct

(pass, break,continue, exit(), quit())

Python have following types of control statements

1. Selection (branching) Statement
2. Iteration (looping) Statement
3. Jumping (break / continue)Statement

Python Selection Statements

Python have following types of selection statements

1. if statement
2. if else statement
3. Ladder if else statement (if-elif-else)
4. Nested if statement

If statements

This construct of python program consist of one if condition with one block of statements. When condition becomes true then executes the block given below it.

Syntax:

```
if ( condition):
        Statement(s)
```

Example:

```
age=int(input("Enter Age: "))
if ( age>=18):
  print("You are eligible for vote")
if(age<0):
        print("You entered Negative Number")
```

if - else statements

This construct of python program consist of one if condition with two blocks. When condition becomes true then executes the block given below it. If condition evaluates result as false, it will executes the block given below else.

Syntax:

```
if ( condition):
        Statement(s)
else:
        Statement(s)
```

Example

```
age=int(input("Enter Age: "))
if ( age>=18):
        print("You are eligible for vote")
 else:
        print("You are not eligible for vote")
```

Ladder if else statements (if-elif-else)

This construct of python program consist of more

than one if condition. When first condition evaluates result as true then executes the block given below it. If condition evaluates result as false, it transfer the control at else part to test another condition. So, it is multi-decision making construct.

Syntax:

```
if ( condition-1):
        Statement(s)
elif (condition-2):
        Statement(s)
 elif (condition-3):
        Statement(s)
else:
        Statement(s)
```

Example:

```
num=int(input("Enter Number: "))
 If ( num>=0):
        print("You entered positive number")
elif ( num<0):
        print("You entered Negative number")
else:
        print("You entered Zero ")
```

Nested if statements

It is the construct where one if condition take part inside of other if condition. This construct consist of more than one if condition. Block executes when condition becomes false and next condition evaluates when first condition became true.

So, it is also multi-decision making construct.

Syntax:

```
 if ( condition-1):
        if (condition-2):
            Statement(s)
        else:
        Statement(s)
```

Example:

```
num=int(input("Enter Number: "))
if ( num<=0):
        if ( num<0):
            print("You entered Negative number")
        else:
            print("You entered Zero ")
else:
        print("You entered Positive number")
```

Python Iteration Statements

The iteration (Looping) constructs mean to execute the block of statements again and again depending upon the result of condition. This repetition of statements continues till condition meets True result. As soon as condition meets false result, the iteration stops.

Python supports following types of iteration statements

1. while
2. for

Four Essential parts of Looping:

i. Initialization of control variable

ii. Condition testing with control variable

iii. Body of loop Construct

iv. Increment / decrement in control variable

Python while loop

The while loop is conditional construct that executes a block of statements again and again till given condition remains true. Whenever condition meets result false then loop will terminate.

Syntax:

```
Initialization of control variable
while (condition):
……..
Updation in control variable
```

……

Example: Sum of 1 to 10 numbers.

```
num=1
sum=0
while(num<=10):
        sum + = num
        num + = 1
print("The Sum of 1- 10 numbers: ",sum)
```

Python range( ) Function

The range() function returns a sequence of numbers, starting from 0 by default, and increments by 1 (by default), and ends at a specified number.

Syntax:

range(**start** value, **stop** value, **step** value)

Where all 3 parameters are of integer type

- Start value is Lower Limit
- Stop value is Upper Limit
- Step value is Increment / Decrement

Note: The Lower Limit is included but Upper Limit is not included in result.

Example

range(5) => sequence of 0,1,2,3,4
range(2,5) => sequence of 2,3,4
range(1,10,2) => sequence of 1,3,5,7,9
range(5,0,-1) => sequence of 5,4,3,2,1
range(0,-5) => sequence of [ ] blank list
(default Step is +1)
range(0,-5,-1) => sequence of 0, -1, -2, -3, -4
range(-5,0,1) => sequence of -5, -4, -3, -2, -1
range(-5,1,1) => sequence of -5, -4, -3, -2, -1, 0

Python for loop

A for loop is used for iterating over a sequence (that is either a list, a tuple, a string etc.) With for loop we can execute a set of statements, and for loop can also execute once for each element in a list, tuple, set etc.

Example: print 1 to 10 numbers

```
for num in range(1,11,1):
        print(num, end=" ")
```

Output: 1 2 3 4 5 6 7 8 9 10

Nested Loops--????

Un- Conditional Control Construct

(pass, break, continue, exit(), quit())

pass Statement (Empty Statement)

The pass statement do nothing, but it used to complete the syntax of programming concept. Pass is useful in the situation where user does not requires any action but syntax requires a statement. The Python compiler encounters pass statement then it do nothing but transfer the control in flow of execution.

Example

```
a=int(input("Enter first Number: "))
b=int(input("Enter Second Number: "))
 if(b==0):
        pass
else:
  print("a/b=",a/b)
```

Jumping Statements

- break
- continue

break Statement

The jump- break statement enables to skip over a part of code that used in loop even if the loop condition remains true. It terminates to that loop in which it lies. The execution continues from the statement which find out of loop terminated by break.

Continue Statement

Continue statement is also a jump statement. With the help of continue statement, some of statements

in loop, skipped over and starts the next iteration. It forcefully stop the current iteration and transfer the flow of control at the loop controlling condition.

+ive index	0	1	2	3	4
String	**H**	**e**	**l**	**l**	**o**
-ive index	-5	-4	-3	-2	-1

String in Python: **String is a sequence which is made up of one or more UNICODE characters.**

A string can be created by enclosing one or more characters in single, double or triple quote.

For Example:

str = "Manoj" # str is a variable storing a string. .

str = "M" #String can also be enclosed in single quotes

Initializing Strings in Python: To initialize string enclose the character or sequence of characters in Single or Double Quotes as follows:

str = 'A' # String enclosed in Single Quotes

str= "Manoj" #String enclosed in double quotes

str = """This course will introduce the learner to text mining and text manipulation basics.

The course begins with an understanding of how text is handled by python"""

We can also use str() function to create string:

N = str () # This function will create an empty string

name = str(12345) # This will store string "12345"

Table 8.1 Indexing of characters in string 'Hello World!'

An inbuilt function len() in Python returns the length of the string that is passed as parameter.

For example:

```
The length of string str1 = 'Hello World!' is 12.
#gives the length of the string str1
>>> len(str1)
12
>>> n = len(str1)
      #length of the string is assigned to n

>>> print(n)
12
```

>>> str1[n-1] #gives the last character of the string
'!'
>>> str1[-n] #gives the first character of the string
'H'

String is Immutable: A string is an immutable data type, that means the contents of the string cannot be changed after it has been created. If you try to change, it would lead to an error.

For Example:

str1 = "Hello World!"
str1[1] = 'a' #if we try to replace character 'e' with 'a'

TypeError: 'str' object does not support item assignment

String Operations with different Operators:

Concatenation (+)

To concatenate means to join. Python allows us to join two strings using concatenation operator plus which is denoted by symbol (+).

str1 = 'Hello' #First string

>>> str2 = 'World!' #Second string

>>> str1 + str2 #Concatenated strings

Output: 'HelloWorld!'

Repetition (*)

Python allows us to repeat the given string using repetition operator which is denoted by symbol *.

For example:

>>> str1 = 'Hello'

>>> str1 * 2 #repeat the value of str1, 2 times

'HelloHello'

>>> str1 * 5 #repeat the value of str1 5 times

'HelloHelloHelloHelloHello'

Membership Operator ('in' and 'not in')

Python has two membership operators 'in' and 'not in'. The 'in' operator takes two strings and returns

True if the first string appears as a substring in the second string, otherwise it returns False.

>>> str1 = 'Hello World!'

>>> 'W' in str1

True

>>> 'Wor' in str1

True

>>> 'My' in str1

False

The 'not in' operator also takes two strings and

returns True if the first string does not appear as a

substring in the second string, otherwise returns False.

For Example:

>>> str1 = 'Hello World!'

>>> 'My' not in str1

True

Accessing String with slicing operator "[]"

e.g.

str='Computer Sciene'

OUTPUT

print('str-', str)	**('str-', 'Computer Sciene')**
print('str[0]-', str[0])	**('str[0]-', 'C')**
print('str[1:4]-', str[1:4])	**('str[1:4]-', 'omp')**
print('str[2:]-', str[2:])	**('str[2:]-', 'mputer Sciene')**
print('str *2-', str *2)	**('str *2-', 'Computer ScieneComputer Sciene')**
print("str +'yes'-", str +'yes')	**("str +'yes'-", 'Computer Scieneyes')**

String Comparison by using Operator ("==" & "!=")

The string comparison operator in python is used to compare two strings.

"==" operator returns Boolean True if two strings are the same and return Boolean False if two strings are not the same.

"!=" operator returns Boolean True if two strings are not the same and return Boolean False if two strings are the same.

These operators are mainly used along with if condition to compare two strings where the decision is to be taken based on string comparison.

Code:

```
string1 = "hello"

string2 = "hello, world"

string3 = "hello, world"

string4 = "world"

print(string1==string4)

print(string2==string3)

print(string1!=string4)

print(string2!=string3)
```

Output:

Escape Sequence Operator "\."

To insert a non-allowed character in the given input string, an escape character is used. An escape character is a "\" or "backslash" operator followed by a non-allowed character. An example of a non-allowed character in python string is inserting double quotes in the string surrounded by double-

quotes.

1. Example of non-allowed double quotes in python string:

Code:

Operator	Description
%d	Signed decimal integer
%u	unsigned decimal integer
%c	Character
%s	String
%f	Floating-point real number

```
string = "Hello world I am from "India""

print(string)
```

Output:

2. Example of allowed double quotes with escape sequence operator:

Code:

```
string = "Hello world I am from \"India\""

print(string)
```

Output:

String Formatting Operator "%."

String formatting operator is used to format a string as per requirement. To insert another type of variable along with string, the "%" operator is used along with python string. "%" is prefixed to another character indicating the type of value we want to insert along with the python string. Please refer to the below table for some of the commonly used different string formatting specifiers:

Code:

```
name = "india"

age = 19

marks = 20.56

string1 = 'Hey %s' % (name)

print(string1)

string2 = 'my age is %d' % (age)

print(string2)

string3= 'Hey %s, my age is %d' % (name, age)

print(string3)

string3= 'Hey %s, my subject mark is %f' % (name, marks)

print(string3)
```

Output:

Traversing a String:

We can access each character of a string or

traverse a string using for loop and while loop.

(A) String Traversal Using for Loop:

```
>>> str1 = 'Hello World!'

>>> for ch in str1:

 print(ch,end = '')

Hello World! #output of for loop
```

In the above code, the loop starts from the first character of the string str1 and automatically ends when the last character is accessed.

(B) String Traversal Using while Loop:

```
>>> str1 = 'Hello World!'

>>> index = 0

#len(): a function to get length of string

>>> while index < len(str1):

 print(str1[index],end = '')

 index += 1

Hello World! #output of while loop
```

Here while loop runs till the condition index < len(str) is True, where index varies from 0 to len(str1) -1

String Methods and Built-in Functions:

Python has several built-in functions that allow us to work with strings. Some of the commonly used built-in functions for string manipulation.

Built-in Functions Description

String.len() Returns the length of the string.

String.endswith() Returns True if a string ends with the given suffix otherwise returns False

String.startswith() Returns True if a string starts with the given prefix otherwise returns False

String.isdigit() Returns "True" if all characters in the string are digits, Otherwise, It returns "False".

String.isalpha() Returns "True" if all characters in the string are alphabets, Otherwise, It returns "False".

string.isdecimal() Returns true if all characters in a string are decimal.

str.format() It allows multiple substitutions and value formatting.

String.index() Returns the position of the first occurrence of substring in a string

string.uppercase() A string must contain uppercase letters.

string.whitespace() A string containing all characters that are considered whitespace.

string.swapcase() Method converts all uppercase characters to lowercase and vice versa.

string.Isdecimal() Returns true if all characters in a string are decimal

String.Isalnum() Returns true if all the characters in a given string are alphanumeric.

string.Istitle() Returns True if the string is a

titlecased string

String.partition() splits the string at the first occurrence of the separator and returns a tuple.

String.Isidentifier() Check whether a string is a valid identifier or not.

String.rindex() Returns the highest index of the substring inside the string if substring is found.

String.Max() Returns the highest alphabetical character in a string.

String.min() Returns the minimum alphabetical character in a string.

String.splitlines() Returns a list of lines in the string.

string.capitalize() Return a word with its first character capitalized.

string.expandtabs() Expand tabs in a string replacing them by one or more spaces

string.find() Return the lowest index in a sub string.

string.rfind() find the highest index.

string.count() Return the number of (non-overlapping) occurrences of substring sub in string

string.lower() Return a copy of String, but with upper case letters converted to lower case.

string.split() Return a list of the words of the string,If the optional second argument sep is absent or None

string.rsplit() Return a list of the words of the string s, scanning s from the end.

rpartition() Method splits the given string into three parts

string.splitfields() Return a list of the words of the string when only used with two arguments.

string.join() Concatenate a list or tuple of words with intervening occurrences of sep.

string.strip() It return a copy of the string with both leading and trailing characters removed

string.lstrip() Return a copy of the string with leading characters removed.

string.rstrip() Return a copy of the string with trailing characters removed.

string.swapcase() Converts lower case letters to upper case and vice versa.

string.translate() Translate the characters using table

string.upper() lower case letters converted to upper case.

string.ljust() left-justify in a field of given width.

string.rjust() Right-justify in a field of given width.

string.casefold() Returns the string in lowercase which can be used for caseless comparisons.

Python Revision: Part 2

LIST

Definition :

A List is a collection of comma separated values within a square bracket. Items in a list need not to be the same.

Features of list

- A List is used to store sequence of values / items.
- All Elements of a list are enclosed in [ ] square bracket.
- Items / values are separated by comma.
- Lists are mutable (changeable) in python.

Different type of lists

```
# List of text
Fruit=['Mango','Orange','Apple']
#List of Characters
Grade=['A','B','C']
# List of integers
Marks=[75,63,95]
# List of floats
Amount=[101.11,125.81,99.99]
# List with different data types
Record=[75,'Mango',101.11]
# List within a List
list1=[2,3,[2,3]]
# List with text splitted into chars
list2=list('IP&CS')
#Empty list
list3=list()
```

Creation of list from user input

eval()– this method can be used to accept tuple from user. Which can be converted to list using list() method.

E.g.

```
T=eval(input("Enter a set of numbers "))
print("Tuple is = ",T)
L=list(T)
print("List is = ",L)
```

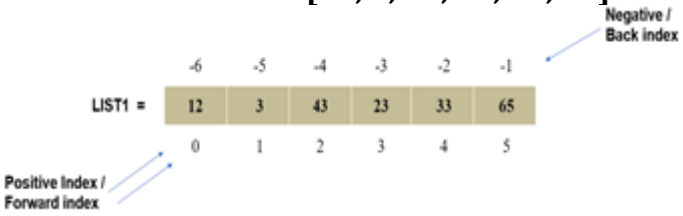
Output

```
Enter a set of numbers 4,7,5,6,9,2
Tuple is= (4,7,5,6,9,2)
List is=[4,7,5,6,9,2]
```

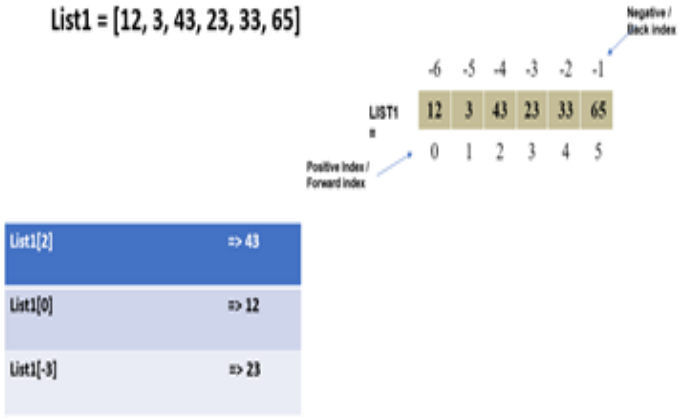
List Indexing

Here,

List1 = [12, 3, 43, 23, 33, 65]



Accessing elements from a list



Traversing of list elements (Iteration/Looping over a list)

List1=[12,3,43,23,33,65]

Method-1

```
List1 = [12,3,43,23,33,65]
for i in range(0,len(List1)):
        print(List[i])
```

Method-2

```
List1 = [12,3,43,23,33,65]
for x in List1:
        print(x)
```

both result in same O/P

```
12
3
43
23
33
65
```

SLICING : *Accessing a part of a list is known as slicing.*

-9	-8	-7	-6	-5	-4	-3	-2	-1
A	M	I	T		J	A	I	N
0	1	2	3	4	5	6	7	8

List1 = ['A', 'M', 'I', 'T', ' ','J','A', 'I', 'N']

Statement	Output
print(List1[:])	['A', 'M', 'I', 'T', ' ', 'J', 'A', 'I', 'N']
print(List1[2:])	['I', 'T', ' ', 'J', 'A', 'I', 'N']
print(List1[:6])	['A', 'M', 'I', 'T', ' ', 'J']
print(List1[0:4])	['A', 'M', 'I', 'T']
print(List1[3:6])	['T', ' ', 'J']
print(List1[-2:-5])	[]
print(List1[-5:-2])	[' ', 'J', 'A']
print(List1[1:7:2])	['M', 'T', 'J']

[Start : End : Step]
(Select from **start** index to **end-1** index number with jump of **step**)

ADDITION OF ELEMENTS IN LIST:

Two ways to add an item in the list

1. Append command	
append( ) is used to add only one element at a time.	
ListF = [44, 55]	
ListF	# [44, 55]
ListF.append(77)	
ListF	# [44, 55, 77]
ListF.append(88,88)	# Error
ListF.append([88, 99])	# [44, 55, 77, [88, 99]]

2. Extend command	
extend( ) is used to add one or more elements	
ListF = [44, 55]	
List F	# [44, 55]
ListF.extend(77)	# Error
ListF.extend([77])	# [44, 55, 77]
ListF.extend([88, 99])	# [44, 55, 77, 88, 99]

MODIFICATION /UPDATION OF ITEM

ListD= [36, 46, 56, 66, 76]	Output
ListD[2]	# 56
ListD[2] = 1000	# 56
ListD[2]	# 1000
ListD	# [36, 46, 1000, 66, 76]
ListD[2:3]= 400,500	# [36, 46, 400, 500, 66, 76]

DELETION OF ELEMENTS FROM LIST

Using del command **O/P**

List1=[44,55,66,77,88]
del List1[2]
print(List1) *[44,55,77,88]*
del List1[1:3]
print(List1) *[44,88]*

Using pop(index) method **O/P**

L1=['K', 'V', '4']
print(L1.pop( )) *4*
print(L1.pop(1)) *V*
print(L1) *['K', '4']*
print(L1.pop(-1)) *4*

OPERATORS

E.g. L1=[23,12,43,22,5,34]
L2=[4,3,5]

Sl. No	Operator	For List L1 & L2	Output
1	Concatenation'+'	L1+L2	[23, 12, 43, 22, 5, 34, 4, 3, 5]

2	Replication/ Repetition '*'	L2*3	[4,3,5,4,3,5,4,3,5]	3	Membership ' in ' and ' not in' (Check presence of element)	43 in L1 3 not in L2	True False

METHODS / FUNCTIONS

E.g. L1=[23,12,43,23,5,34]
L2=['K', 'V', 'S']

Sl. No.	Method / Function	Example For given list L1 & L2	Output
1	**index(item)** To display index of an item	L1.index(43)	2
2	**insert(index, item)** To add an item at given index	L2.insert(1, 'A')	['K','A','V','S']
3	**reverse()** Reverse the order of elements	L2.reverse()	['S','V','A','K']
4	**len(List)** Returns length of list	len(L1)	6
5	**sort()** Arrange the elements in ascending or descending Order	**Ascending order** L2.sort() **Descending order** L2.sort(**reverse =True**)	['A','K','S','V'] ['V','S','K','A']
6	**count(item)** Counts frequency of an item	L1.count(23)	2
7	**remove(item)** Remove the item	L2.remove('A')	['K','V','S']
8	**max(list)** Returns maximum value	max(L1)	43
9	**min(list)** Returns minimum value	min(L1)	5

Reference Videos:-
List (Part-1) ***https://youtu.be/J8NQuV4CEfw***
List (Part-2) ***https://youtu.be/pheM8nHPNG4***

List (Part-3) ***https://youtu.be/GYptLReCqaw***

TUPLE

Definition : Python Tuple is used to store the sequence of immutable Python objects. The tuple is similar to lists since the value of the items stored in the list can be changed, whereas the tuple is immutable, and the value of the items stored in the tuple cannot be changed.

Creating Tuple

Empty tuple: An empty tuple can be created as follows.

T1 = ()

Initialing Tuple with values:

Example

rgb = ('red', 'green', 'blue')
numbers = (3,)
'''a tuple with one element, you need to include a trailing comma after the first element'''
tuple1 = (10, 20, 30, 40, 50, 60)

Understanding Index numbers

Z=(3,7,4,2)

z =	(3,	7,	4,	2)
index	0	1	2	3
negative index	-4	-3	-2	-1

```
z = (3, 7, 4, 2)          # Access the first item of a tuple at index 0
print(z[0])               #Output 3
z = (3, 7, 4, 2)          # Accessing the last item of a tuple
print(z[3])               #Output 2
        OR
print(z[-1])              #Output 2
```

Tuple slices

```
print(z[0:2])       #output would be  3,7
print(z[:3])        #Output 3,7,4
print(z[-4:-1])     #Output 3,7,4
print(z[-1:-3:-1])  #Output 2,4
```

Basic Tuple operations

The operators like concatenation (+), repetition (*), Membership (in) works in the same way as they work with the list. Consider the following table for more detail.

Let's say Tuple t = (1, 2, 3, 4, 5) and Tuple t1 = (6, 7, 8, 9) are declared.

Operator	Description	Example
Repetition	enables the tuple elements to be repeated multiple times.	T1*2 = (1, 2, 3, 4, 5, 1, 2, 3, 4, 5)
Concatenation	It concatenates the tuple on either side of the operator.	T1+T2 = (1, 2, 3, 4, 5, 6, 7, 8, 9)
Membership	It returns true if a particular item exists in the tuple else false	print (2 in T1) prints True.
Iteration	The for loop is used to iterate over the tuple elements.	for i in T1: print(i) **Output** 1 2 3

		4 5

Toppers = (("Vinodini" , "XII-F", 98.7), ("Soundarya" , "XII-H" , 97.5), ("Tharani" , "XII- F", 95.3), ("Saisri" , "XII-G" , 93.8))

List vs. Tuple

SN	List	Tuple
1	The literal syntax of list is shown by the [].	The literal syntax of the tuple is shown by the ().
2	The List is mutable.	The tuple is immutable.
3	The List has a variable length.	The tuple has the fixed length.
4	The list provides more functionality than a tuple.	The tuple provides less functionality than the list.
5	The list is used in the scenario in which we need to store the simple collections with no constraints where the value of the items can be changed.	The tuple is used in the cases where we need to store the read-only collections i.e., the value of the items cannot be changed. It can be used as the key inside the dictionary.

Nested Tuple

A tuple can be defined inside another tuple; called Nested tuple. In a nested tuple, each tuple is considered as an element.

The for loop will be useful to access all the elements in a nested tuple.Read more on Sarthaks.com - https://www.sarthaks.com/1017899/what-is-nested-tuple-explain-with-an-example

Example 1:

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

Output:

('Vinodini' , 'XII-F', 98.7)

('Soundarya' , 'XII-H' , 97.5)

('Tharani' , 'XII-F', 95.3)

('Saisri' , 'XII-G' , 93.8)

Example 2:

```
tup = (50,60,70,80,90, (200, 201))
```

The tuple (200, 201) is called a nested tuple as it is inside another tuple.

The nested tuple with the elements (200, 201) is treated as an element along with other elements in the tuple 'tup'. To retrieve the nested tuple, we can access it as an ordinary element as tup[5] as its index is 5.

Built-in Tuple Methods

Method	Description
count()	Returns the number of times a specified value occurs in a tuple
index()	Searches the tuple for a specified value and returns the position of where it was found

Some function that are useful to work with tuple

1. len((1, 2, 3, [6, 5]))

Output: 4 # It returned 4, not 5, because the list counts as 1.

2. max()

It returns the item from the tuple with the highest value.

```
a=(3,1,2,5,4,6)
max(a)
```

Output : 6

3. min()

Like the max() function, the min() returns the item with the lowest values.

```
a=(3,1,2,5,4,6)
min(a)
```

Output: 1

4. sum()

This function returns the arithmetic sum of all the items in the tuple.

```
a=(3,1,2,5,4,6)
sum(a)
```

Output: 21

5. tuple()

```
# Creating an  empty tuple
        t1 = tuple()
# creating a tuple from a list
        t2 = tuple([1, 4, 6])
# creating a tuple from a string
        t1 = tuple('Python')
# creating a tuple from a dictionary
        t1 = tuple({1: 'one', 2: 'two'})
```

6. count()

Return the number of times the value X appears in the tuple:

```
thistuple = (1, 3, 7, 8, 7, 5, 4, 6, 8, 5)
x = thistuple.count(5)
print(x)
```

Output: 2

7. index()

The index() method returns the index of the specified element in the tuple.

Example 1: Find the index of the element

```
vowels = ('a', 'e', 'i', 'o', 'i', 'u')
index = vowels.index('e')
print('The index of e:', index)
index = vowels.index('i')
print('The index of i:', index)
```

Output

```
The index of e: 1
The index of i: 2
```

Programs on Tuples

1. Write a Python program to test if a variable is a tuple or not.

Ans.

```
x = ('tuple', False, 3.2, 1)
if type(x) is tuple:
        print('x is a tuple')
else:
        print('Not a tuple.')
```

2. Write a Python program which accepts a sequence of comma-separated numbers from user and generate a list and a tuple with those numbers.

Ans.

```
values = input("Enter some no. separated by , ")
list = values.split(",")
tuple = tuple(list)
print('List : ',list)
print('Tuple : ',tuple)
```

3. Write a python program to print sum of tuple elements.

Ans.

```
test_tup = (7, 5, 9, 1, 10, 3)
print("The original tuple is : " + str(test_tup))
res = sum(list(test_tup))
 # printing result
print("The sum of all tuple elements are : " + str(res))
```

4. Write a python program to Check if the given element is present in tuple or not.

Ans.

```
test_tup = (10, 4, 5, 6, 8)
# printing original tuple
print("The original tuple : " + str(test_tup))
N = int(input("value to be checked:"))
res = False
for ele in test_tup :
        if N == ele :
                res = True
                break
print("Does contain required value ? : " + str(res))
```

5. Write a Python program to find the length of a tuple.

Ans

```
#create a tuple
tuplex = tuple("KVS RO JAIPUR")
print(tuplex)
#use the len() to find the length of tuple
print(len(tuplex))
```

6. Write a Python program calculate the product, multiplying all the numbers of a given tuple.

Ans

```
nums = (4, 3, 2, 2, -1, 18)
print ("Original Tuple: ")
print(nums)
temp = list(nums)
product = 1
for x in temp:
        product *= x
print(product)
```

7.Write a Python program to calculate the average value of the numbers in a given tuple of tuples.

Ans

```
nums = ((10, 10, 10, 12), (30, 45, 56, 45),
          (81, 80, 39, 32), (1, 2, 3, 4))
print ("Original Tuple: ")
print(nums)
result = [sum(x) / len(x) for x in zip(*nums)]
print("\nAverage of numbers:\n",result)
```

8. Write a Python program to check if a specified element presents in a tuple of tuples.

Ans:

```
colors = (
   ('Red', 'White', 'Blue'),
   ('Green', 'Pink', 'Purple'),
   ('Orange', 'Yellow', 'Lime'),
)
print("Original list:")
print(colors)
c1 = 'White'
result = any(c in tu for tu in colors)
print(result)
```

DICTIONARY

Key points: -

- Python Dictionaries are a collection of some **key:value** pairs.
- Python Dictionaries are **unordered** collection
- Dictionaries are **mutable** means values in a dictionary can be changed using its key
- Keys are **unique**.
- Enclosed within brace **{ }**

Working with dictionaries	
Creating Python Dictionary dict( ) method	**# empty dictionary** my_dict = {} **# dictionary with integer keys** my_dict = {1: 'apple', 2: 'ball'} **# dictionary with mixed keys** my_dict = {'name': 'John', 1: [2, 4, 3]} **# using dict()** my_dict = dict({1:'apple', 2:'ball'})
Accessing Elements from Dictionary get( ) method	my_dict = {'name': 'Jack', 'age': 26} print(my_dict['name']) print(my_dict.get('age')) **# Trying to access keys which doesn't exist returns None** print(my_dict.get('address')) **# KeyError** print(my_dict['address']) **Output: -** Jack 26 None Traceback (most recent call last): File "<string>", line 15, in <module> print(my_dict['address']) KeyError: 'address'
Changing and Adding Dictionary elements	**# Changing and adding Dictionary Elements** my_dict = {'name': 'Jack', 'age': 26} **# update value** my_dict['age'] = 27 print(my_dict) **# add item** my_dict['address'] = 'Downtown' print(my_dict) **Output: -** {'name': 'Jack', 'age': 27} {'name': 'Jack', 'age': 27, 'address': 'Downtown'}
Removing elements from Dictionary pop( ) method popitem method( ) clear( ) method del function	**# Removing elements from a dictionary** squares = {1: 1, 2: 4, 3: 9, 4: 16, 5: 25} **# remove a particular item, returns its value** print(squares.pop(4)) print(squares) **# remove an arbitrary item, return (key,value)** print(squares.popitem()) print(squares) **# remove all items** squares.clear() print(squares) **# delete the dictionary itself** del squares **Output: -** 16 {1: 1, 2: 4, 3: 9, 5: 25} (5, 25) {1: 1, 2: 4, 3: 9} {}
len() – Finding number of items in dictionary	dict={'Name':'Aman','Age':37 } print("Length: - ", len(dict)) **OUTPUT** Length:- 2
keys() -	x=dict(name="Aman",age=37,country="India") print(x.keys())

returns all the available keys	**OUTPUT** dict_keys(['country','age','name'])
values()- returns all the available values	x=dict(name="Aman",age=37,country="India") print(x.values()) **OUTPUT** dict_values(['India',37,'Aman'])
items()- return the list with all dictionary keys with values.	x=dict(name="Aman",age=37,country="India") print(x.items()) **OUTPUT-** dict_items([('country','India'),('age',37),('name','Aman')])
update()- used to change the values of a key and add new keys	x=dict(name="Aman",age=37,country="India") d1=dict(age=39) x.update(d1,state="Rajasthan") print(x) **OUTPUT-** {'country':'India','age':39,'name':'Aman','state':'Rajasthan'}
fromkeys() i- is used to create dictionary from keys	keys={'a','e','i','o','u'} value="Vowel" vowels=dict.fromkeys(keys,value) print(vowels) **OUTPUT-** {'i':'Vowel','u':'Vowel','e':'Vowel','a':'Vowel','o':'Vowel'}
copy()- returns a shallow copy of the dictionary.	x=dict(name="Aman",age=37,country="India") y=x.copy() print(y) OUTPUT->{'country':'India','age':37,'name':'Aman'}

Dictionary Membership Test in not in	**#Membership Test for Dictionary Keys** squares = {1: 1, 3: 9, 5: 25, 7: 49, 9: 81} print(1 in squares) print(2 not in squares) print(49 in squares) **Output -** True True False
Iterating Through a Dictionary	**# Iterating through a Dictionary** squares = {1: 1, 3: 9, 5: 25, 7: 49, 9: 81} for i in squares: print(squares[i])

Multiple choice questions based on Dictionary	
1	Which of the statement(s) is/are correct. i. Python dictionary is an ordered collection of items. ii. Python dictionary is a mapping of unique keys to values iii. Dictionary is mutable. iv. All of these.
2	Which function is used to return a value for the given key. i. len( ) ii. get( ) iii. keys( ) iv. None of these
3	Keys of the dictionary must be i. similar ii. unique iii. can be similar or unique iv. All of these
4	Which of the following will delete key-value pair for key='red' form a dictionary D1 i. Delete D1("red") ii. del. D1("red") iii. del D1["red"] iv. del D1
5	Which function is used to remove all items form a particular dictionary. i. clear( ) ii. pop( ) iii. delete iv. rem( )
6	In dictionary the elements are accessed through i. key ii. value iii. index iv. None of these
7	Which function will return key-value pairs of the dictionary i. key( ) ii. values( ) iii. items( ) iv. get( )
8	To create a dictionary , key-value pairs are separated by……. i. (;) ii. (,) iii. (:) iv. (/)
9	Which of the following statements are not correct: a. An element in a dictionary is a combination of key-value pair b. A tuple is a mutable data type c. We can repeat a key in dictionary d. clear( ) function is used to deleted the dictionary. i. a,b,c ii. b,c,d iii. b,c,a iv. a,b,c,d
10	Like lists, dictionaries are which mean they can be changed. i. Mutable ii. immutable iii. variable iv. None of these
11	To create an empty dictionary , we use i. d=[ ] ii. d =( ) iii. d = {} iv. d= < >
12	To create dictionary with no items , we use ii. Dict ii. dict( ) iii. d = [ ] iv. None of these
13	What will be the output >>>d1={'rohit':56,"Raina":99} >>>print("Raina" in d1) i. True ii. False iii. No output iv. Error

FUNCTIONS

<Place Holder>

Definition :

<Two Columns from this point onwards>

Data File Handling

Text Files

Definition :

BINARY FILE HANDLING IN PYTHON

Binary File

* In binary file data is in unreadable format and to work on a binary file we have to convert the data into readable form for read as well as write operation.
* The binary file stores some objects which have some structure associated with them. Like list, nested list, tuple, dictionary etc.
* These objects are first serialized and then stored in a binary file.
* If a binary file (existing) open for reading purpose, when the last record is reached (EOF reached), it may raise an EOFError exception, if not handled properly.
* Thus a binary file should open for reading purposes either in "try and except" blocks or using "with statement".

Serialization or Pickling:

It is the process of transforming a python object to a stream of bytes called byte streams. These byte streams can then be stored in binary files. Serialization process is also called pickling.

De-serialization or un-pickling:

It is the inverse of the pickling process where a byte stream is converted back to a Python object.

The pickle module implements the algorithm for serializing and de-sterilizing the python objects and deals with binary files. The Pickle Module must be imported to read and write objects in binary files.

MODES OF BINARY FILES

b = Open the binary file

rb = Open binary file in read only mode (file must be existed)

wb = Open binary file in write only mode (new file created, overwritten if existed)

ab = Open binary file in append (write only) mode. (open if exists otherwise create new file)

rb+ = Open binary file in read & write mode. (file must be existed)

wb+ = Open binary file in write & read mode. (new file created, overwritten if existed)

ab+ = Open binary file in append (write & read) mode. (open if exists otherwise create new file)

BASIC OPERATIONS IN BINARY FILE

- Creating a new file
- Reading from file
- Writing into file
- Appending the file
- Searing in File
- Deleting data from file
- Creating a copy of file

OPEN AND CLOSE BINARY FILE

open( ) function:

Syntax:

File_handler/File_Object=open(file_name, access mode)

Example:

file_obj= open("bin_file.dat",'wb')

This statement opens bin_file.dat binary file in write mode

Note : if file mode is not mentioned in open function then default file mode i.e 'rb' is used,

close() function:

The close( ) method of a file object flushes any unwritten information and close the file

object after which no more writing can be done.

Example

```
file_obj.close( )
```

WRITING INTO FILE- PICKLING

To write an object into file, the file should be opened in write mode. The dump( ) function of the pickle module is used to write objects into a file.

Syntax:

```
import pickle
File_Handler=open("Bin_file.dat",'wb')

pickle.dump(python_Object_to_be_Written, File_Handler)
```

Expamle:

```
# Write Dictionary Object in Binary file

import pickle

stud={}

fwb=open("student.dat","wb")

choice='y'

while choice.lower()=='y':

    rno=int(input("Enter Roll No: "))

    name=input("Enter Name: ")

    marks=float(input("Enter Marks(out of 500): "))

    per=marks/5

    if(per>=33):

    res="Pass"

    else:

    res="Fail"

    stud['rollno']=rno

    stud['name']=name

    stud['Marks']=marks

    stud['percent']=per

    stud['result']=res

    pickle.dump(stud, fwb)

    print("Record Saved in File")

    choice=input("More Record(Y/N)? ")

fwb.close()
```

APPENDING RECORD IN BINARY FILE

Binary file must open in append mode (i.e., "ab'") to append the records in file. A file opened in append mode will retain the previous records and append the new records written in the file.

Syntax:

```
import pickle
File_Handler=open("Bin_file.dat",'ab')
pickle.dump(python_Object_to_be_Written, File_Handler)
```

Expamle:

```
# Append the Binary file
```

```
import pickle

stud={}

fwb=open("student.dat","ab")

choice='y'

while choice.lower()=='y':

        rno=int(input("Enter Roll No: "))

        name=input("Enter Name: ")

        marks=float(input("Enter Marks(out of 500): "))

        per=marks/5

        if(per>=33):

        res="Pass"

        else:

        res="Fail"

        stud['rollno']=rno

        stud['name']=name

        stud['Marks']=marks

        stud['percent']=per

        stud['result']=res

        pickle.dump(stud, fwb)

        print("Record Saved in File")

        choice=input("More Record(Y/N)? ")

fwb.close()
```

READING FROM BINARY FILE: UN-PICKLING

While working with a binary file for reading purposes the runtime exception EOFError raised when it encountered the EOF position. This EOFError exception can be handled with two ways.

1. **Use of try and except block**

The try and except statements together, can handle runtime exceptions. In the try block, i.e., between the try and except keywords, write the code that can generate an exception and in the except block, ie., below the except keyword, write what to do when the exception (EOF - end of file) has occurred

```
File_object=("binary File Name",'access mode')
try:
        ……..
        Write actual code, work in binary file
        ……..
except EOFError:
        ……..
        Write Code here that can handle the error raised in try block.
        ……..
```

2. **Use of with statement**

The with statement is a compact statement which combines the opening of file, processing of file along with inbuilt exception handling and also closes the file automatically after with block is over.
Explicitly, we need not to mention any exception for the "with statement".
Syntax:

```
with open("File_Name",'mode') as File_Handler:
```

```
# Read Records from Binary file
import pickle
stud={}
frb=open("student.dat","rb")
try:
    while True:
    stud=pickle.load(frb)
```

```
    print(stud)
except EOFError:
    frb.close()
```

SEARCHING RECORD FROM FILE

```
import pickle
stud={}
found=0
print("Searching in file student.dat...")
try:
    frb=open("student.dat", "rb")
    while True:
    stud=pickle.load(frb)
    if stud['percent']>51.0:
            print(stud)
            found+=1
except EOFError:
    if found==0:
            print("No Record found with
marks>51")
    else:
            print(found," Record(s)
Searched")
    frb.close()
```

COPY OF FILE IN ANOTHER FILE

```
#CREATIN A COPY OF EXISTING FILE
PROGRAM
import pickle
def fileCopy():
    ifile = open("student.dat","rb")
    ofile = open("newfile.dat","wb")
    try:
    while True:
            rec=pickle.load(ifile)
            pickle.dump(rec,ofile)
    except EOFError:
    ifile.close()
    ofile.close()
    print("Copied successfully")

def display1():
    ifile = open("student.dat","rb")
    print("----Records of Student file---")
    try:
    while True:
            rec=pickle.load(ifile)
            print(rec)
    except EOFError:
    ifile.close()

def display2():
    ofile = open("newfile.dat","rb")
    print("----Records of Copy file---")
    try:
    while True:
            rec=pickle.load(ofile)
            print(rec)
    except EOFError:
    ofile.close()
fileCopy()
display1()
display2()
```

RANDOM ACCESS & UPDATING BINARY FILE

Python provides two functions that help to manipulate the position of the file pointer and we can read and write from the desired position in the file. The Two functions of Python are: tell() and seek()

(i) The tell() Function

tell() method can be used to get the current position of File Handle in the file. This method takes no parameters and returns an integer value. Initially the file pointer points to the beginning of the file(if not opened in append mode).

Syntax

```
fileObject.tell( )
```

This method returns the current position of the file read/write pointer within the file.

(ii) The seek( ) Fuction

In Python, seek() function is used to change the position of the File Handle to a given specific position in the file.

Syntax

fileObject.seek(offset, mode)

where

offset is a number-of-bytes

mode is a number from 0 or 1 or 2

0: sets the reference point at the beginning of the file.

1: sets the reference point at the current file position.

2: sets the reference point at the end of the file.

Example:

```
f=open("chapter.dat,'rb')

f.seek(20)

# will place the file pointer at 20th byte from the beginning of the file (default)

f.seek(20,1) # will place the file pointer at 20th byte ahead of current file-pointer position (mode = 1)

f.seek(-20,2)

# will place file pointer at 20 bytes behind (backward direction) from end-of file (mode = 2)

f.seek(-10,1) # will place file pointer at 10 bytes behind from current file-pointer position (mode = 1)
```

Note:

- **Backward movement of file-pointer is not possible from the beginning of the file (BOF).**
- **Forward movement of file-pointer is not possible from the end of file (EOF).**

PROGRAM OF tell() and seek()

```
import pickle

stu={ }

found =false

# open binary file in read and write mode

rfile=open("stu.dat", "rb+")

# Read from the file

try:

    while True:

        pos=rfile.tell()

        stu=pickle.load(rfile)

        if stu["marks"] > 81:

            stu["Marks"] +=2

            rfile.seek(pos)

            pickle.dump(stu, rfile)

            found=True

except EOFError:

    if found=False:
```

```
            print("Sorry, No Record found")
        else:
            Print("Record(s) updated")
        rfile.close()
```

DELETE FILE

To delete a file, import the OS module, and run its os.remove( ) function:

Remove the file "demofile.txt"

```
import os
os.remove("demofile.txt")
```

Check if File exist:

To avoid getting an error, we might want to check if the file exists before try to delete it:

```
import os
if os.path.exists("demofile.txt"):
    os.remove("demofile.txt")
else:
    print("The file does not exist")
```

Delete Folder

To delete an entire folder, use the os.rmdir() method:

```
import os
os.rmdir("myfolder")
```

Note: Only empty folders can be removed.

#PROGRAM

```
import pickle
import os
def delete_rec():
    f1 = open("student.dat","rb")
    f2 = open("temp.dat","wb")
    rn=int(input("Enter rollno to delete:"))
    try:
    while True:
    d = pickle.load(f1)
    if d["rollno"]!=rn:
    pickle.dump(d,f2)
    except EOFError:
    print("Record Deleted.")
    f1.close()
    f2.close()
    os.remove("student.dat")
  os.rename("temp.dat","student.dat")

delete_rec()
```

CSV Files

CSV (Comma Separated Values)

is a file format for data storage which looks like a text file. The information is organized with one record on each line and each field is separated by comma.

- **A Comma Separated Values (CSV) file is a plain text file that contains the comma-separated data.**
- **These files are often used for exchanging data between different applications.**
- **CSV files are usually created by programs that handle huge amounts of data. They are used to export data from spreadsheets (ex:- excel file) and databases (Ex:- Oracle, MySQL). It can be used to import data into a spreadsheet or a database.**

CSV File Characteristics :

- **One line for each record**
- **Comma separated fields**
- **Space-characters adjacent to commas are ignored**
- **Fields with in-built commas are separated by double quote characters.**

When Use CSV?

- **When data has a strict tabular structure**
- **To transfer large database between programs**
- **To import and export data to office applications.**
- **To store, manage and modify shopping cart catalogue**

Why Use CSV / Advantages

- **CSV is faster to handle**
- **CSV is easy to generate**
- **CSV is human readable and easy to edit manually**
- **CSV is simple to implement and parse**
- **CSV is processed by almost all existing applications**

CSV Disadvantages

- **No standard way to represent binary data**
- **There is no distinction between text and numeric values**
- **Poor support of special characters and control characters**
- **CSVallows to move most basic data only. Complex configurations cannot be imported and exported this way.**
- **Problems with importing CSV into SQL (no distinction between NULL and quotes)**

CSV File Structure

sample.csv file structure

```
Name,       DOB,          City
Ram,        12-Jul-2001, Delhi
Mohan,      23-Jan-2005, Delhi
Suraj,      17-Dec-2002,Kolkata
```

Python CSV Module

- CSV Module is available in Python Standard Library.
- The CSV module contains classes that are used to read and write tabular form of data into CSV format.
- To work with CSV Files, programmer have to import CSV Module.

import csv
Methods of CSV Module :

- writer( )
- reader( )

Both the methods return an Object of writer or reader class. Writer Object again have two methods – writerow( ) , writerows( ).

writer( ) Methods

This function returns a writer object which is used for converting the data given by the user into delimited strings on the file object.

writer( ) Object Methods –

- w_obj . writerow(<Sequence>) : Write a Single Line
- w_obj . writerows (<Nested Sequence>) : Write Multiple Lines

Example:-

```
## writerow()
import csv
row=['Nikhil', 'CEO', '2', '9.0']
f=open("myfile.csv", 'w')
w_obj = csv.writer(f)
w_obj.writerow(row)
f.close()
```

writerows()

```
import csv
rows = [['Nikhil','CEO','2','9.0'],
      ['Sanchit','CEO','2','9.1']]
f=open("myfile.csv",'w')
w_obj = csv.writer(f)
w_obj.writerows(rows)
f.close()
```

reader( ) Methods
This function returns a reader object which will be used to iterate over lines of a given CSV file.
r_obj = csv.reader(csvfile_obj)

To access each row, we have to iterate over this Object.

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

Example:-

```
import csv
f=open("myfile.csv",'r')
r_obj = csv.reader(f)
for data in r_obj:
        print(data)
f.close()
```

If we consider the sample.csv file given above in the *CSV file structure* the output of the above code will be:
OUTPUT

```
['Name', 'DOB', 'City']
['Ram', '12-Jul-2001', 'Delhi']
['Mohan', '23-Jan-2005', 'Delhi']
['Suraj', '17-Dec-2002', 'Kolkata']
```