SYLLABUS - 40, 21, 10

Unit-I : Introduction to Python :

Features of Python - Installing and running Python - interpreter and interactive mode - Identifiers - Reserved keywords - variables - comments in Python. Data types - Numeric, string, list, sets, tuple, Dictionary, boolean; Operators - Arithmetic, Relational, Assignment, logical, Bitwise, Membership operator, identity operator. Statements and expressions, string operations; Boolean expressions, Data type conversion, type coercion; Input from keyboard - input function, raw - input function, Mutable and immutable objects; Illustrative programs.

Unit - II : Decision Making, Control structures and functions.

Decision Making: Simple if, if.... else and if... elif statement; control statement - for loop, range(), while, break, continue, pass.

functions: Built in functions - Mathematical functions, Date and time, dir(), help() function; User defined functions - Return values, parameters and Arguments, functions calls, local and global scope, function composition, recursion, anonymous functions.

Unit - III : Strings and lists:

Strings: strings in python, string functions and methods, string slicing, immutable property, string traversal, Escape characters, string formatting operators and functions.

List - Creation of list, Values and accessing elements, mutable property, traversing a list, Copying the list, altering values, deleting elements from list.

Built-in list operators and built-in methods.

Unit-IV : Tuples and Dictionaries.

Tuples - creating, accessing values, immutable property, assignment of tuples, returning tuples, tuples as arguments - variable length arguments - basic tuple operations, Built-in tuple operations, Built-in tuple functions.

Dictionaries: creating a dictionary, accessing values, updating dictionary, deleting elements from dictionary, dictionary keys properties, operations in dictionaries, Built-in dictionary methods, Illustrative Prog

Unit - V : files and Exception Handling:-

Files: Text files, opening a file, closing a file, reading from a file and writing in to a file, file opening modes, closing a file, file object Attributes, file positions, renaming deleting a file and files related Methods

Directory: Directory methods - mkdir(), chdir(), getcwd(), rmdir().

Exceptions in python: Definition - Built in exceptions, handling Exceptions - try....except, except with No exception, except with Multiple exceptions, try ... finally; User defined exceptions. Illustrative programs.

Unit - 1

Introduction to python and data types

Introduction:

* Python is an interpreted, object-oriented high level programming language having dynamic semantics.
* It was developed by Guido van Rossum in 1991 at Centrum Wiskunde and Informatica in Netherland.
* Python is an extension of the existing programming language ABC developed by CWI.
* Versions 2.x and 3.x are the most widely used versions.

Features of Python:

(i) easy to learn:-
Python has few keywords, simple structure and clearly defined syntax.

(ii) Expressive language:-
Python is easy to read and to understand. This is because a single line of python code can do more than a single line of code which does in most other languages.

(iii) free and opensource:-
Python can be downloaded free of cost from python's official website (http://www.python.org).

(iv) Object-oriented language:-
Python language supports program development using real world thinking.

(v) Extensible or Mixable :- Python code can be written into C (or) C++ programs and can be compiled using C or C++ compilers.

(vi) Graphical - User interface support (GUI) :- Python has a number of libraries for developing graphical user interface for user needs.

(vii) Interpreted language :- Python source code is converted automatically into an intermediate code called bytecode.

(viii) Portable :- Python bytecode written for one operating system can run on any other operating systems such as window, linux, etc..

(ix) Easy to use :- Python programs can be run directly without any intermediate compile and link steps in language like C, C++.

Installing and Running python :-

Python software is free and downloads are available in the website python.org.

(i) Open any web browser and give http://www.python.org in the address bar.

(ii) Click the downloads button. The following pull down menu appears.

(iii) Select windows from the pull down menu. the following windows appears. scroll down and go to python 3.7.3 and click on download x86 executable installer.

(iv) Run the .exe file to install python 3.7.3 the python software will be placed in C:\Users\user\AppData\Local\Programs\Python\Python37-32> python.

Interpreter and Interactive mode

* python is an interpreted object oriented language. An interpreter is a translator which translates the high level language program line by line into machine readable codes called byte codes.

In python there are two basic program execution methods.

(i) Interactive prompt execution.

(ii) python script execution.

(i) Interactive prompt execution:

To activate this mode navigate to start → All programs → python 3.7 software in the computer.

eg:- >>> "welcome, may god bless you"
welcome, may god bless you.

The following are the important drawbacks of interactive prompt.

(i) It is not possible to save the statements as a file.

(ii) It is not possible to write more statements as a block or module.

(ii) Python script execution:-

A python script or program is a set of python statements or expressions. Scripts can be stored as a file with .py extension and can be run directly.

(i) Navigate to start → All programs →

python 3.7 → IDLE.

(ii) Click the file option in the status bar in the interactive prompt window. A pull down menu appears as shown below.

(iii) Click new file option in the pull down menu. A new editor window appears as shown below to edit, run etc our programs.

Python token:

A token is an individual basic element in python language. A program is written using the available tokens.

(i) Keywords or Reserved words.
(ii) Identifiers
(iii) Constants or Literals
(iv) Operators
(v) Separators or delimiters.

eg:
```
>>> a = 10
>>> b = 20
>>> c = a+b
```
The different tokens are

a b c = + 10 20

Python Character Set:-

Characters are the smallest units of the python language. These are used to write tokens. The python characters are defined by Unicode character set.

Uppercase alphabets - A to Z
lowercase alphabets - a to z
numerical digits - 0 to 9
special characters - ( ) [ ] - = _ | \ > < + - / ?
@ # ~ ^ * % ! ' " . : ; ? { } $
white spaces - blank space, horizontal tab, new line.

Keywords:

* Keywords are reserved words which belong to the python language.
* They have standard predefined meaning.
* The users have no right to change its meaning.
* In python there are 33 keywords

false, None, true, and, as, assert, break, class, continue, def, del, elif, else, except, finally, for, from, global, if, import, in, is, lambda, nonlocal, not, or, pass, raise, return, try, while, with, yield.

Identifiers:- Identifiers are names given to variables, modules, functions, lists, classes, etc in a program. Identifiers do not change during program execution.

eg:-

(i) valid identifiers:-

Temp, basic_pay, firstName, _123, A10

(ii) Invalid identifiers and reason for invalidity

Identifies	reason for invalidity
A$	- Dollar symbol not allowed
9AD	- Must not begin with digit
a.b.8	- Delimiter . (dot) not allowed.

Variables:-

variables is a name given to a memory location to keep track of the data in the program.

eg: A0 - A10 are a basic pay.

Assigned values to variables:

Assigning values to the valid variables is called assigning.

Types (i) Assigning single value to single variables.
(ii) Assigning multiple values to multiple variables.
(iii) Assigning same value to multiple variables.

Comments in python:-

Comments are non-executable statements. These are included anywhere in the program for better Understanding.

There are two types of comments in python.

(i) Single line comment:-

Single line comments are written in a single line, these comments must begin with # (hash).

eg: >>> # program by Edwin.
>>> # python is simple.

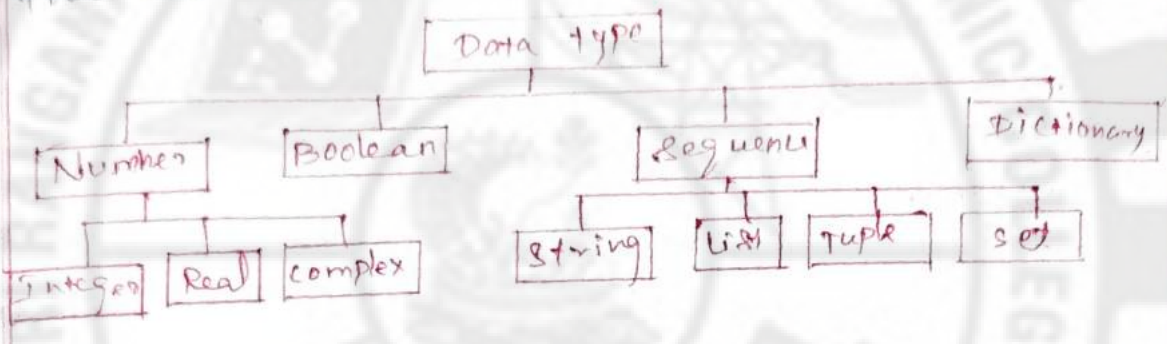
(ii) Multiline Comments:-

Multiline comments are Comments written in more than one line. These comments must begin and end with """ triple codes.

eg: >>> """ this program is used by calculated the salary of the employees".

Data type:

In python data takes the form of objects. The type of data may be either built-in type or User defined type. Data type specify the type of the value that are to be stored.



Number:

Number data type represents a value made up of digits and some special characters (+ - .). There three types,

(a) Integer:

Integer is a numeric value formed by digits without decimal point. Integer numbers are defined in python as int class.

Types of integer:

(a) Decimal integer (b) Octal integer

(c) Hexadecimal (d) Binary.

(B) Real or float:

Real or float is a numeric value by digits with one decimal point. Real numbers are defined in python as float class.

(a) fractional form eg: -0.571, .64, 2.4

(b) Exponent form or Scientific form

eg: .257 e5, .2751E-5

(iii) Complex Number:

A complex number is a number represented with real part and imaginary part. Complex numbers are defined in python as complex class.

eg syntax:

real part + or - imaginary part j or J

Eg:

$-9+7.8j$, $3+4j$, $3.7-7J$

Boolean:-

Boolean Datatype takes the value there true or false. It is used to compare two numeric numbers or expressions.

eg:- $10>9 \Rightarrow$ true

$7.5 < 6.2 \Rightarrow$ false.

Sequence:- A sequence is a collection of similar or different data types. This allows to store multiple values in an organized and efficient way.

(a) String:- String is a sequence of python Characters. the characters are enclosed within single or double or triple quotation marks.

eg:- (i) 'God loves you'

(ii) "X231ab".

(b) List:- list is a flexible ordered collection of objects. The objects are separated by comma and enclosed within square brackets [].

eg: (i) ['God', 'loves', 'you']

(ii) ["bat", 2, 3, 3.2]

(c) Tuple: Tuple is an ordered collection of objects. The objects are separated by comma and enclosed within parantheses ().

eg: (i) ("God", "loves", "you")

(ii) ("bat", 2, 3, 3.2)

(d) Set: Set is an unordered collection of objects. The objects are separated by comma and enclosed within curly brackets {}.

eg: (i) {"Monda", "Tuesday"}

(ii) {65, 90}.

Dictionary: Dictionary is an unordered collection of key and a value pair, separated by commas, enclosed within curly brackets {}.

eg: {"Age": 25, "D.O.B": "13-3-2020"}

Operators:

An operator is a symbol which represents some operations that can be performed on data.

(i) Arithmetic operators.

(ii) Relational or Comparision Operators.

(iii) logical operators.

(iv) Short hand assignment Operators.

(a) Binary operators

(b) Unary operators

(c) Ternary operators

(i) Arithmetic operators

Arithmetic operators are used to do Mathematical operations. These are used with numeric values.

(a) binary operators

(b) Unary operators

(a) Binary operators: Binary operators need two operands for operations.

Operator		example.
+ (add)	→	X = 5 + 3 = 8 Y = X + Z
- (sub)	→	X = 5 - 3 = 2 Y = X - Z
* (mul)	→	X = 5 * 3 = 15 Y = X * Z
% (mod)	→	X = 5 % 3 = 2
/ (div)	→	X = 5 / 3 = 1.6667.

(b) Unary operator:- Unary operator need only one operand for operation.

operator		example
- (Negation)	→	-X -10

(ii) Relational or comparision operator:-

Relational operators are used to compare the operand values on LHS and RHS of the operator. It gives the result as true or false.

Operator	example
< (less than)	A < B
> (greater than)	A > B
<= (less than or equal to)	A <= B
>= (greater than or equal to)	A >= B
== (equal to)	A == B
!= (not equal to)	A != B

logical operator :-

logical operators are used to compare the operands on the LHS and RHS of the operator.

Operator	Example.
logical And	a = 20 a < 50 and a < 100 o/p : true
logical or	a = 20 a < 50 or a < 100 true.
logical Not	a = 20 not (a < 50 and a < 100) false.

Assignment Operator :-

This operators are used to simplify the coding of certain type of assignment statement.

Syntax : Variable Operator variable.

Operator	example.
+=	x = 20 x = x + 10 written as x += 10 value of x is 30

-=	y = 20 x -= 10 value => 10
*=	y = 20 x *= 10 value => 200
/=	y = 20 x /= 10 value => 2.0
%=	x = 18 x = x %10 x %= 10 value = 8
//=	x = 17 x //= 3 value => 5.

(v) Bitwise operator :-

Bitwise operators are used to do bit by bit operations on integers.

Operator		Meaning
&	→	Bitwise AND
|	→	Bitwise OR
^	→	Bitwise exclusive OR
~	→	Bitwise one's complement.
>>	→	Bitwise right shift.
<<	→	Bitwise left shift.

[vi] Identity operator :-

Identity operators are used to compare the location of memory objects. This gives either true or false value.

operator	example.
is	x = 10, y = 10 x is y => true.
is not	x = 10 y = 10 x is not y => false.

Membership operator:-

Membership operators are used to check the membership of a member in a sequence such as strings, lists, tuples etc.

Operator	Example.
in	x = 'God' 'G' in x true.
not in	x = 'God' 'G' not in x false.

Statements and expressions:-

Expression:- An expression is a linear combination of identifiers, literals, strings, objects and operators. Each expression represents a value.

eg:
```
>>> x = (a+b) * (2-c)        # arithmetic.
>>> s = "welcome" + "you"    # String expression.
```

Statements:- Statements are defined as instructions written by the user that can be read and executed by the python interpreter.

types of statements:-

(i) Assignment statements:-

Assignment statements are used to assign data value to variables.

eg:
```
>>> x = 10
>>> y = x + 20
>>> y
30
>>>
```

(ii) Augmented assignment statements:-

Statements written using shorthand assignment operators also called augmented assignment statements.

eg:- >>> y=3
>>> y+=10
>>>

(iii) Conditional statements:- Conditional statements are used to skip or execute a group of statements based on the boolean result of the condition.

eg:- >>> a=30
>>> b=50
>>> if a<b: a # conditional statement
30
>>>

(iv) looping statements: Looping statements are used to execute a group of statements repeatedly until some condition is satisfied. (true)

(v) Control statement:

Control statements are used to control the execution of loops.

Types:-

(i) continue - to skip a part of loop statement

(ii) Break - to come out of loop even when the condition in the loop is true.

(iii) pass - to write empty control statements, functions and classes.

(vi) Multiline statements:-

A python statement written in more than one line is called multiline statement.

(a) Explicit line continuation
(b) Implicit line continuation.

(a) Explicit line continuation:

Explicit line continuation is a process of writing a statement in more than one line using explicit line continuation character \ (backslash).

eg:-
```
>> a = 1 + 2 +3 +\
     4 + 5 +\
     6 + 7
>>> a
  28
>>>
```

(b) Implicit line continuation:- Implicit line continuation is a process of writing a statement in more than one line using any one of the grouping symbols (), [], {}.

eg:-
```
>>> a = (1 + 2 + 3
        + 4 + 5 +
        6 + 7)
>>> a
  28
>>>
```

Boolean expression:-

Boolean expression is an expression that gives either true or false value. Internally true refers to integer 1 and false refers to integer 0.

eg:-
```
>>> a = 20
>>> a > 50
false
>>>
```

Data type Conversion:

Data type Conversion is a process of converting one data type into another data type.

types: (i) Automatic or implicit type conversion.

(ii) Explicit type conversion.

(i) Automatic type Conversion:

If a mathematical expression involves different data types, the lower data type is automatically converted to higher data type before execution to avoid data loss.

eg:-
```
>>> n1 = 10
>>> n2 = 5.75
>>> n3 = n1+n2
>>> print (n3)
15.75
>>>
```

(ii) Explicit type Conversion:

The process of converting on data type into another without affecting its original type using type Conversion function is called explicit Conversion or type casting.

Syntax:- required datatype (variables)

eg:
```
>>> n1 = 10
>>> n2 = 5.75
>>> print (n1 + int (n2))
15
>>>
```

Type conversion functions:

Type conversion functions are predefined functions. These are used to convert one data type to another type.

syntax : functionname (parameter)

functions:- int (x [, base]), float (x), ord (x), hex (x), oct (x), str(x), complex (x,y), tuple (x), set (x), list (x), dict (x), bin (x).

Input from keyboard:-

In python there in no direct statement to given values to variables through keyboard.

(a) Input ()
(b) raw - input ()

(a) Input ()

This function is available in python version 2.x and 3.x.

syntax : variable name = input(prompt)

eg:- (i) age = input ()

(ii) age = input ("Enter the age:")

When this statement executed, the following message is displayed.

Enter the age : 22.

(b) raw_input (): This function is available in python version 2.x. This reads the given input value of any data type as a string.

Syntax: variablename = raw_input (format)

Mutable and immutable data types

In python, all data values are considered as objects. This means, when the user assigns a value to the variable, the assigning value becomes objects and is stored in memory location.

(a) Mutable Objects:-

Mutable objects are objects whose content can be modified after creation.

eg:- List, dictionary, set.

(b) immutable objects:-

Immutable objects are objects whose content cannot be modified after creation.

eg:- Integer, float, Boolean, String, tuple.

Getting Output from the program:-

The output from the program can be displayed over the visual display unit by using the print() function.

syntax:- print ([object 1, object 2 ...][, sep=' '][end='\n'], [file=sys.stdout])

eg:-

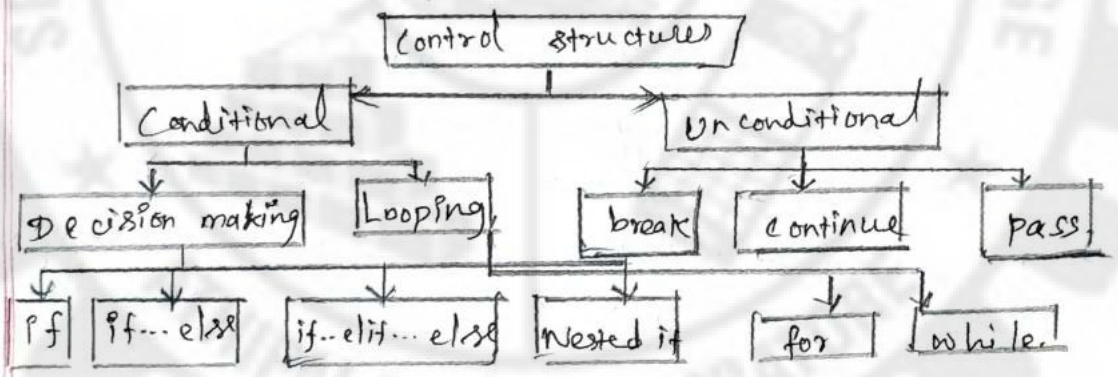
```
>>> a=3, b='Boy', c=8.9
>>> print (a, b, c)
3 Boy 8.9
>>>
```

Unit - II

Decision Making, Control Structures and Functions

Introduction:

* Python programs are executed statement by statement in a sequential order.
* In many situations it is necessary to alter this order.
* This is achieved using control structure. These structures are used to control the program execution order.



Indentation:-

Indent is defined as a space that is left when a block of text spaces inwards compared with other texts surrounding it.

In programming languages like C, C++, Java etc {}.

Conditional structures:-

There are two types of conditional structures.

(a) Decision making structures.

(b) looping structures.

(a) Decision making structures:-

Decision making structures are used to skip or to execute a group of statements or a single statement based

on the result of a test condition. The test condition may be a relational or logical expression giving either true or false value.

(i) if statement
(ii) if ... else statement.
(iii) if ... elif ... else statement.

(i) if statement :-

if statement is used to alter the sequential flow of execution according to the output of a test condition in the header.

If the output of the test condition is true, one or a group of statements called suit or block is executed followed by the next statement sequentially.

If the test condition value is false, if statement will not work and the control is transferred to the next statement skipping the suit of statements.

Syntax :- if test condition: → Header
    Statement 1
    Statement 2
    ....
    ....
    Statement n → true suit or block.
next statement

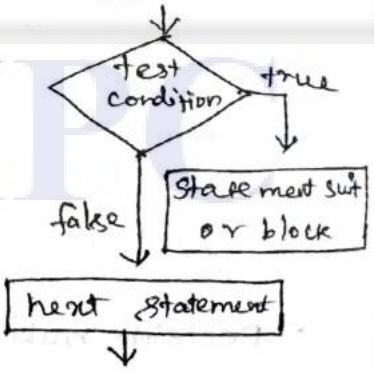
eg :-

```
mark = 70
grade = 'A'
if grade == 'A':
    mark = mark + 10
    print (mark)
print ('Program over')
```

O/P :-

```
80
Program over.
```

flow chart :-



(ii) if ... else statement :-

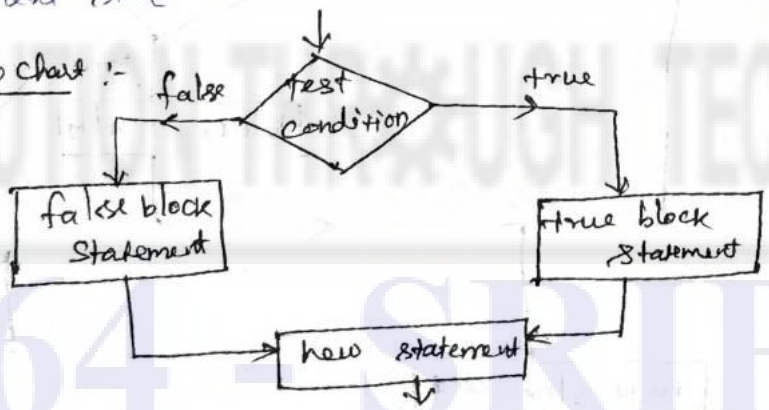
if ..... else statement is used to alter the sequential flow of execution according to the output of a test condition called header. If the output of the test condition is true, one or a group of statements called true suit or block is executed followed by the next statement in sequential order.

If the test condition is false, one or a group of statement called false suit or block in the else part is executed followed by next statement in sequential order.

eg :-

```
if test condition :          → Header
    Statement 1
    Statement 2              → True suit or block
    Statement n
else :
    Statement 1
    Statement 2              → true suit or block.
    Statement n
next statement
```

flow chart :-



eg :-

```
a = 10 ; b = 20
if a > b :
    print ('A is maximum')
else :
    print ('B is maximum')
```

O/P :-

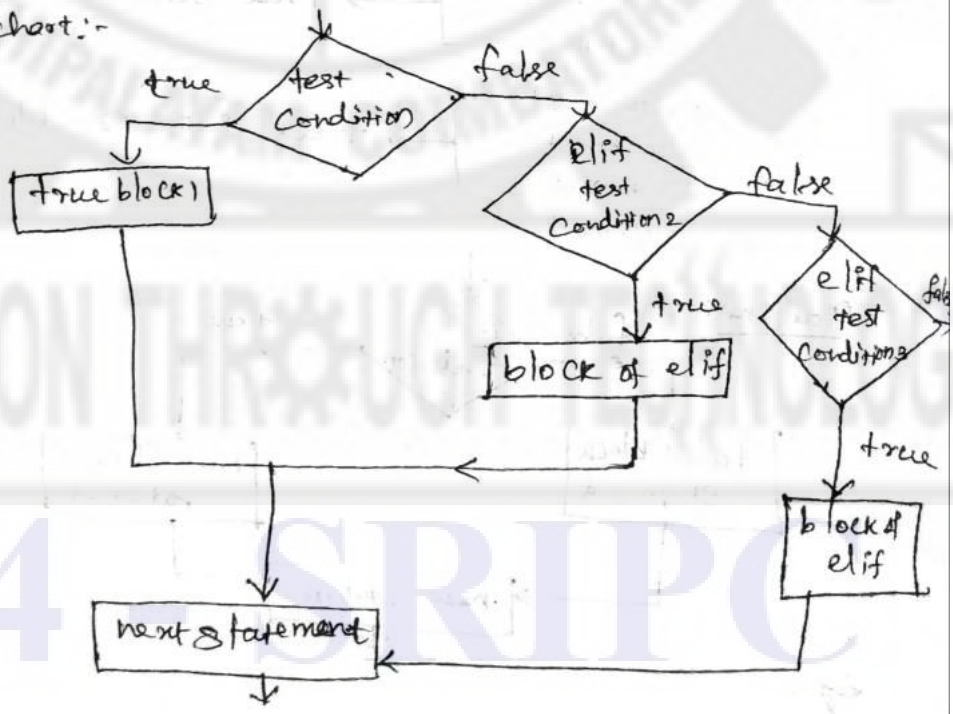
B is maximum.

(iii) if ... elif ... else statement :-

if ... elif ... else statement is used to check multiple test conditions. this statement is formed by joining any number of elif ... from in the if part of the if .... else statement.

syntax :- if test condition1:
    statement block 1
elif test condition 2 :
    statement block 2
elif test condition 3:
    statement block 3
    . . . . . .
    . . . . . .
elif test condition n:
    statement block n
else :
    statement block
next statement

flow chart :-



eg :-

```
a = -27
if a>0 :
    print ('Number is positive')
```

```
elif a == 0:
    print ('Number is zero')
elif a < 0:
    print ('Number is negative')
else:
    print ('Give a valid number')
```

O/P:- Number is negative.

Control Statement:-

Looping structures or iterative structures are used to execute a group of statements repeatedly for a known number of time. These structures need two entities namely, loop variable and loop terminator.

looping structures are:

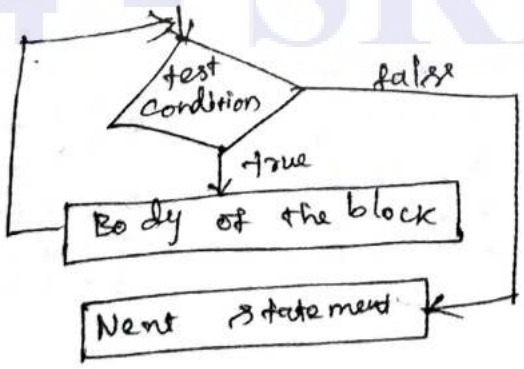
(i) while loop (ii) for loop.

(i) while loop:- While loop is used to execute a block of codes called block or body of the loop repeatedly until the given condition is true.

Syntax:-

```
While test condition :  ---> Header
    statement -1
    statement -2
    . . . . .
    statement -n        ---> Body or block of the loop.
next statement.
```

flow chart:-



eg :-

```
i = 1
while (i <= 3):
    print ('God loves you')
    i = i + 1
print ('program ended')
```

o/p:-

God loves you
God loves you
God loves you
program ended.

(ii) Nested while loop:- If one while loop is enclosed within another while loop then such loops are called nested while loops.

syntax:

```
while condition :
    while condition :
        .....
    .....
    .....
```

outer loop ← the whole block; Inner loop → the inner while block

eg :-

```
i = 1
j = 1
while i <= 3 :
    while j <= 5 :
        print (i, '*', j, '=', i * j)
        j = j + 1
    print ()
    i = i + 1
    j = 1
```

o/p:-

```
1 * 1 = 1
1 * 2 = 2
1 * 3 = 3
1 * 4 = 4
1 * 5 = 5

2 * 1 = 2
2 * 2 = 4
2 * 3 = 6
2 * 4 = 8
2 * 5 = 10

3 * 1 = 3
3 * 2 = 6
3 * 3 = 9
3 * 4 = 12
3 * 5 = 15
```

(iii) for loop:-

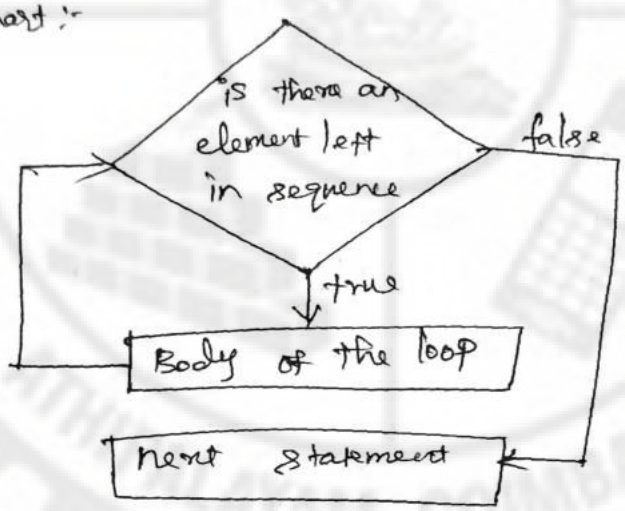
for loop is used to execute a group of statements repeatedly for a known number of times.

Syntax:-

```
for var in sequence
    statement - 1
    statement - 2
    ...
    statement - 2
next statement
```

→ Body or block of the for loop.

flowchart:-



eg:- 1

```
for i in range(3):
    print ('God love you')
print ('the program executed successfully')
```

o/p:- God love you
God love you
God love you
the program executed successfully.

eg:- 2

```
s = 'python'
for i in s:
    print(i)
```

o/p:-

p
y
t
h
o
n

(iv) Nested for loop:

A for loop within another for loop then such loops are called nested for loops. Each loop must follow its indentation. There is no limit on the number of loops that can be nested.

Syntax:-

```
for i in sequence 1:
    for j in sequence 2:
        body of the loop j
        . . . . .
    . . . . . .
    . . . . . .
```

eg:-

```
for i in range (1,4):
    for j in range (1,6):
        print (i,'*',j,'=', i*j)
    print ()
```

O/P:-

```
1 x 1 = 1
1 x 2 = 2
1 x 3 = 3
1 x 4 = 4
1 x 5 = 5

2 x 1 = 2
2 x 2 = 4
2 x 3 = 6
2 x 4 = 8
2 x 5 = 10

3 x 1 = 3
3 x 2 = 6
3 x 3 = 9
3 x 4 = 12
3 x 5 = 15
```

Unconditional control structures:-

The Unconditional Control statement are statement that are used to transfer the flow of execution for one part of the program to another part without any condition check.

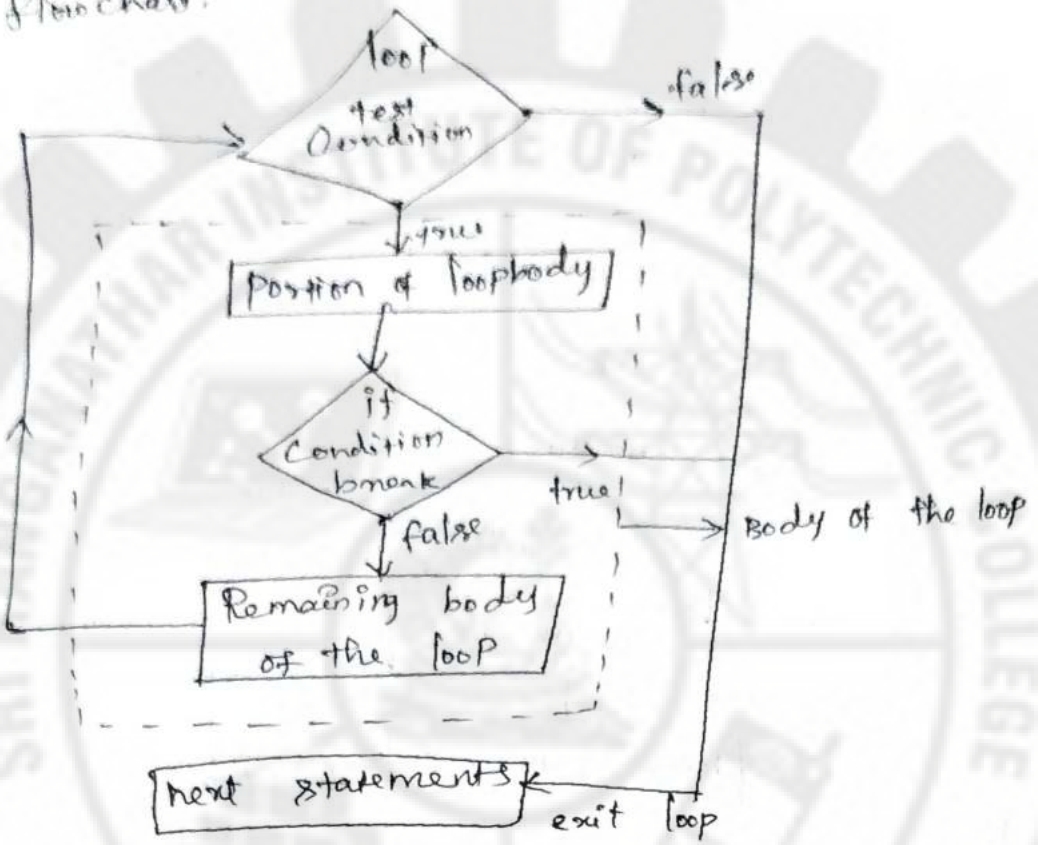
(i) break (ii) continue (iii) pass.

(i) break statement:-

break statement is used to exit from a loop when the test condition is true.

Syntax : **break**

Flowchart :-



eg :-

```
for i in range (10):
    if i == 4:
        break
    print ('The number in range is :', i)
print ('The loop breaks when i = 7')
```

O/P :-

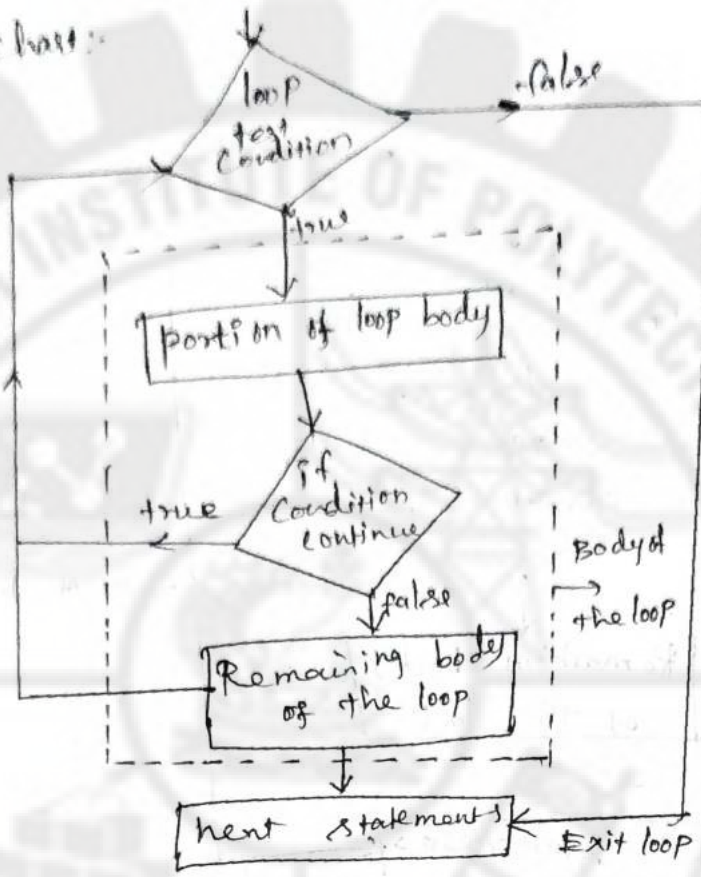
```
The number in range is : 0
The number in range is : 1
The number in range is : 2
The number in range is : 3
The loop breaks when i = 4
```

6) Continue statements :-

Continue statement is used to skip a part in the body of the loop when the test condition is true and control is transfered to the beginning of the loop. loop operation does not terminate but continues with the next iteration.

Syntax :- `Continue`

Flow chart:-



eg:-

```
S = 'Nagercoil'
for i in S:
    if i == 'e' or i == 'l':
        continue
    print (i, end = ' ')
```

O/P:-

Nagrcoi

(iii) pass statement :-

This statement acts as a place holder or an empty area and this can be used for writing codes in future.

Syntax :- pass

eg:-

```
If (a>b):
    pass
else:
    print ('a is big')
```

2.2 Functions

Built in functions:

Built-in functions are functions which are not to be written by the programmer But these are are available in the languages as modules. The programmers can import and use it depending on their need.

Mathematical functions:

Mathematical functions are predefined functions in the python language for performing Mathematical functions are listed below.

(a) Number functions
(b) Power and logarithmic functions
(c) Trignometric, angular Conversion and hyperbolic functions.
(d) Special functions
(e) Mathematical constants.

Syntax :- math. functionname (Parameter)

i) Number functions :-

① ceil(x) → Return the rounded up integer value greater or equal.
eg :- math.ceil(8.78) = 9.

② floor(x) → Return the rounded down integer value greater or equal.
eg :- math.floor(8.78) = 8.

③ copysign(x,y) → Return the value of x with sign of y. The returned values is float type.
eg :- math.copysign(-8,9) = 8.0

④ fabs(x) → Return the absolute value of x.
eg :- math.fabs(-5) = 5.0

⑤ factorial(x) → Return the factorial value of x. x >= 10.
eg :- math.factorial(4) = 24.

(ii) power and logarithmic functions:-

① exp(x) → Return e ** y

eg: math.exp(2) = 14.8.413159

② expm1(x) → Return e ** x - 1

eg: math.expm1(5) = 147.413.

③ log1p(x) → Return natural log of 1+x to base e

eg:- math.log1p(23) = 3.178053

④ log2p(x) → Return natural log of x to base 2.

eg:- math.log2(23) = 4.5235.

⑤ log10(x) → Return the log of x to base 10

eg:- math.log10(23) = 1.361727.

⑥ pow(x, y) → Return x raised to y.

(iii) Trigonometric and angular Conversion and Hyperbolic functions:-

① acos(x) → Return the arc cosine of x in radians. the value of x must be between -1 and 1.

eg:- math.acos(-0.1) = 1.6709.

② asine(x) → Return the arc sine of x in radians.

eg: math asine(-0.1) = -0.1001

③ cos(x) → Return the cosine of x in radians.

eg:- math.cos(34) = -0.8485

④ sin(x) → Return the sine of x in radians

eg: math.sin(24) = 0.5290

⑤ tan(x) → Return the tangent of x in Radians.

eg:- Math.tan(34) = -0.62

(iv) special functions:-

① erf(x) → Error function. This occurs while working with probability, statics, and partial differentiation.

eg:- math.erf(40) = 1.0
math.erf(-40) = -1.0.

② erfc(x) → Complementary error function. This is 1-erf(x) Return the Complementary error function of x.

eg:- math.erfc(40) = 0.0.

③ gamma(x) → Return the gamma function. x can be any number.

eg:- math.gamma(39) = 5.23022e61

④ lgamma(x) → Return the natural logarithmic of the absolute value of the gamma function x.

eg:- math.lgamma(30) = 5.33367.

(v) Mathematical constants:-

① pi → Return the mathematical constant π = 3.14.

eg:- math.pi = 3.141592

② tau → Return the mathematical constant τ = 6.28...

eg:- math.tau = 6.283185

③ inf → Return the floating point positive infinity.

eg:- math.inf = inf — math.inf = -inf (negative infinity)

④ nan → Return the floating point not a number.

eg:- math.nan = nan.

dir() function:-

dir() function is used to find the list of names of all present in the module.

Syntax: dir (object name)

eg:-

(i) import math

print (dir (math))

o/p:

['__doc__', '__loader__', '__name__', '__package__', '__spec__', 'acos', 'asin', 'cos', 'copyright' etc. ...]

(ii) import datetime

print (dir (datetime))

o/p:

['MAXYEAR', 'MINYEAR', 'date', etc..]

help()

This function is used to get the python help documentation for modules, functions, classes, keywords etc. The documentation will be printed on the screen.

Syntax: help(object)

eg: To get the documentation for math.

```
>>> help('math')
help on built-in module math:
NAME
  math
DESCRIPTION
  this module is always available.
FUNCTIONS
  acos(x, /)
  Return the arc cosine of x
  ....
  ....
```

User defined functions:

A user defined function is a group of related reusable code written by the user to perform a specific well defined task or job.

Function definition:

def key word is used to define a function.

syn:-

```
def function name (list of parameters):
    "function_docstring"
    statement -1
    statement -2
    .
    .
    .
    statement -n
    return (expression)
```

Header → def ... :

statement -1 ... statement -n → function body

Rules:-

(i) The key word def marks the function header.

(ii) list of arguments are optional.

Parameters and arguments:-

Parameters are variables present in the function definition. These variables receive value only at the time of function calling. These parameters are also called as dummy parameters.

```
def abc (i,j):
    k = i+j
    return (k)
```

In this function i and j are called parameters because the values of i and j are not available. so this function cannot be executed.

Arguments are values present in the function calling, these values are passed to the defined function when it is called.

Return statements:-

return statement is used to return a value or more than one value to the calling point and is optional.

Syntax:-

return [or] return (expression), (expression)....

All functions must return a value. If there is no expression or value to return back, it sends a special value of the expression. The paranthesies around the expression is optional.

There are two types of functions namely fruitful functions and void functions.

eg:- def example (a, b, c):
return a+b+c, a* b* c

function calls:-

To execute the already defined function it has to be called by specifying the name followed by list of objects or values called arguments enclosed within paranthesis.

Syntax:- functionname (list of Arguments).

Rules:-

(i) function name must be the same name used in the function definition as the re-assigned name.

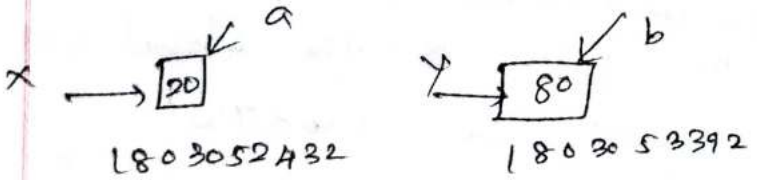
(ii) If the function call has no arguments an empty pair of parenthesis is a must.

eg:- program:- program using function to add two integer numbers

```
def add(a,b):
    c = a+b
    print ("The memory address of a is:", id(a))
    print ("The memory address of b is:", id(b))
    return(c)
x = int(input ('Enter the first number:'))
y = int(input ('Enter the second number:'))
z = add(x,y)
print ('the memory address of x is:', id(x))
print ('The memory address of y is:', id(y))
print ('The sum of x and y is:', z).
```

Output:-

```
Enter the first number: 20
Enter the second number: 80
The memory address of a is: 1803052432
the memory address of b is: 1803053392
the memory address of x is: 1803052432
The memory address of y is: 1803053392
The sum of x and y is: 100
```



Scope and life time of variables:-

Scope of variable means how widely a variable is known among the set of functions in programs.

life time of variable means how long a variable retains a given value during the execution of the program.

(i) local scope (ii) Global scope

(iii) Encoding scope (iv) Built-in scope.

(i) local scope :-

The variable defined inside a function are called local variable. These variables are local to the function and are visible to the function where they are defined.

eg :- def example ():
    a = 10
    b = 20
    print (a+b)
example ()

(ii) Global scope :-

* The variables defined out side all the functions in a file is called global variables.

* These variables are visible to all the functions in the file.

* The value of the global variables cannot be modified inside the function.

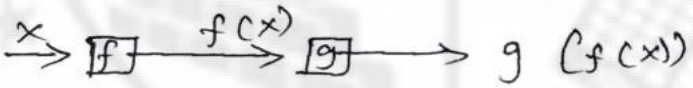
eg:-

```
x = 'Good Morning'
def example():
    y = ' To you all'
    z = x + y
    print (z)
example()
```

O/P :- Good Morning to you all.

Function Composition :-

Function composition is defined as a process of combining two or more functions in such a way that the output of one function becomes the input of second function and so on.



eg. program :-

```
def square (x):
    return (x * x)
def divide (x):
    return (x / 2)
out = divide (square (4))
print ("The output is :", out)
```

O/P :- The output is : 8.0

Recursive function :-

Recursive is a technique used to solve the problem using computer. In this, the given problem is defined as a function and this contains a statement to call itself repeatedly until the problem is solved.

eg. Program: to find the factorial of a number.

Program:

```
def factorial(n):
    if n==0:
        return 1
    else:
        fact = n * factorial(n-1)
        return fact
n = int(input("Enter a positive number:"))
fact = factorial(n)
print("The factorial of {} is {}.format(n, fact))
```

O/P:-

Give a positive number: 5

The factorial of 5 is 120

Anonymous function- Lambda function:-

Lambda or anonymous function is an expression form of function that generates a function object as ordinary function.

Syntax:- var=lambda arg1, arg2...: expression using arg1, arg2...

eg:-

```
Sum=lambda a,b,c: a+b+c
S=Sum(10,20,30)
print("the value sent by lambda is:", S)
```

O/P:-

The value sent by lambda is: 60

Unit III

Strings and Lists

Strings

String is a sequence of python characters. These characters are enclosed within any one of the following types of quotation marks.

(i) Single quotation marks :- string enclosed within single quotation marks is called single line string.

eg :- 'python language'

(ii) Double quotation marks :- single line string.

eg :- "python language"

(iii) Three quotation mark :- Multiline string.

eg :- """python is an object-oriented language introduced in the year 1991"""

Assigning string :-

The general form to assign string to a variable is:

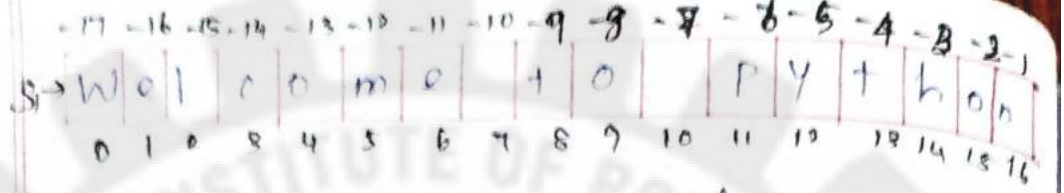
syntax :- variable = string literal

eg :-

```
S1 = 'welcome to python'
S2 = "DOB IS 13-03-2020"
S3 = '' # empty string.
```

Accessing a string :-

```
S1 = 'welcome to python'
```



0 to 16 → positive index

-1 to -17 → negative index.

The integer numbers 0 to 16 are called positive index and are used to access the characters from left to right. the integer numbers -1 to -17 are called negative index and are used to access the characters from right to left.

Types of accessing a string:-

(i) Accessing characters in a string.

(ii) slicing a string.

(i) Accessing characters in a string:-

Syntax:- variable name[index]

eg:- S1 = 'welcome to python'

(i) print(S1[6])

e

(ii) printing last character

print(S1[-1])

n.

(ii) slicing a string:-

slicing is a process of taking a range of characters from the given string.

syntax: variable name [start : stop]

eg:- S1 = 'welcome to python'

(i) print(S1[:6])

welcome

(ii) print(S1[2:9])

lcome t

Immutable Property

* Created string objects are immutable.
* It means the content of the string objects cannot be changed after it has been created.
* The created string can be empty.

eg:- Consider the string S = 'welcome'. If we alter the third character l to e as, S[2] = 'e'.

It gives an error as type error: 'str' object does not support item assignment.

Basic string operations :-

(i) finding the length
(ii) Concatenation of string
(iii) Repeating string
(iv) Iteration.

(i) finding the length :-

len() function is used to find the length of the string.

syntax : len(String variable)

eg:-

```
S1 = "God loves you"
print(len(S1))
```

18.

(ii) Concatenation of strings :-

Concatenation is a process of joining two strings to form a new string. This is carried out using the + operator.

Syntax :- string 3 = string 1 + string 2.

eg :-

```
S1 = 'MG CSE'
S2 = 'polytechnic'
S3 = S1+S2
print (S3)
```

O/P :-

MG CSE polytechnic

(iii) Repeating string :-

Repeating is a string process of adding a string to itself number of times. This is carried out using * operator.

Syntax : string * times.

eg :-

```
print ("Irene" * 4
```

O/P :- IreneIreneIreneIrene.

(iv) Iteration :- Strings can be used in for loop to do iteration operation.

Syntax : for loop variable in string variable:

eg :-

```
S = 'python'
for i in S:
    print (i)
```

O/P :-

```
p
y
t
h
o
n
```

Escape Sequence :-

Escape sequence is defined as a string literal having a beginning character \ (backslash) followed by one or more characters within single or double quotation marks.

Escape sequence	Example
(i) \a (Bell or audible alert)	print ('God \a love') o/p :- God • love
(ii) \b (Backspace)	print ('God\b love') o/p :- Go love
(iii) \' (single quotes)	print ('God \' s love') o/p :- God's love
(iv) \" (double quotes)	print ("sachin \" kolhi \" tendulkar") o/p :- sachin "kolhi" tendulkar
(v) \\ (Backslash)	print ('python\\\') o/p :- python\\
(vi) \t (Horizontal tab)	print ('1 2 3 4 5 6 7 8 9 0') print ('\t hello') o/p :- 1 2 3 4 5 6 7 8 9 0 H e l l o
(vii) \n (new line)	print ('Hello \n world') o/p :- Hello world

String traversal :-

String traversal is a process to visit all the characters in a string exactly once to do any one of the following.

(i) to read the character
(ii) to print the characters
(iii) to process the characters.

eg :- s = 'python'
for i in s:
print (i)

o/p :-

p
y
t
h
o
n

String Methods :-

In python there are a large variety of built-in methods it must be called by the string object.

syntax :- stringname.methodname()

(i) capitalize() :-
the first letter of the character will be capital letter. syntax : stringname.ca

eg :- a = "hello world"
a.capitalize()

o/p :-
Hello world.

(ii) center() :-
this method is used to centre the string with the specified length and fill it with the optional fill character.

syntax :- stringname.center(width [, fillch

eg :- b = "hello"
print(b)
b.center(11, "x")

o/p :- hello
xxx hello xxx

(iii) count() :-
This method is used to count the number of occurances of a substring in the string.

syntax :- S.count(str1 [, start] [, stop]).

eg :- a = "hello world"
a.count("o", 0, len(a))

o/p :-
2.

(iv) endswith ():-

syntax:- s. endswith (str, [start], [stop])

eg:- a = 'hello world'

a.endswith ("d", 0, len(a))

o/p:- true.

(v) find() :- to find the index value of the string.

syntax:- s. find (string, [start] [, stop])

eg:- a = "helloworld"

a. find ("o", 0, len(a))

o/p:- 4.

(vi) Index () :-

syntax:- s. index (str [, start] [, stop])

eg:- a = "hello world"

a. index ("w", 0, len(a))

o/p:- 6

(vii) islower () :-

syntax:- s. islower ()

eg:- a = "helloworld"

a. islower ()

o/p:- true.

(viii) isupper () :-

syntax:- s. isupper ()

eg:- a = "hello world"

a. isupper ()

o/p:- false.

(ix) istitle () :-

syntax:- s. istitle ()

eg:- a = "Hello World"

a. istitle ()

o/p:- true.

(x) length()

syntax:- len(str)

eg:- a = "hello world"
len(a)

O/P:- 10

(xi) lower()

syntax:- s.lower()

eg:- a = "Hello world"
a.lower()

O/P:- hello world

(xii) upper()

syntax:- s.upper()

eg:- a = "hello world"
a.upper()

O/P:- HELLO WORLD

(13) max():-

syntax: max(str)

eg:- a = "helloworld"
max(a)

O/P:- w

14) min():-

syntax:- min(str)

eg:- a = "hello world"
min(a)

O/P:- e

15) Startswith():-

syntax:- s.startswith(String [, start] [, stop])

eg:- a = "hello world"
a.startswith("hello", 0, len(a))

O/P:- true. //

String formatting operators and functions

By using print() function the user can display the output of a program onto the display unit without any form useful to the users understanding. But this can be achieved by formatting the strings.

(i) formatting with % Operators.

(ii) formatting with format() function.

(i) formating with % operators:-

format operator % is used to format the output to be printed according to the need of the user.

Syntax:-

'Message 1 % Conversation Character message 2 % Conversation Character ....' %(V1, V2, ...)

Character	Meaning
c	Single Character
d or i	signed decimal integer
e	floating point value in exponent form.
f	floating point value.
s	String
o	octal integer
u	unsigned decimal integer
x	hexa decimal integer.

eg:

```
name = 'Ram'
age = 57
weight = 77.5
height = 185
print ('Candidate name is %s, age is %d
        weight is %.1f kg and height is %.
    %(name, age, weight, height))
```

O/P:-

Candidate name is Ram, age is 57, weight is 77.5 and height his 185.00 cm

(ii) formatting with format() function:-

format() is a built-in string method and is used to format the output to be printed according to the need of the user.

Syntax:-

'message {index1: Specifier} message {index2: Specifier} ... message {Indexn: specifier}'. format (index1=v1, index2=v2 ...)

eg:-1

```
>>> print (1234, 56.98, 'Edwin')
```

O/P:- 1234 56.98 Edwin

eg:2:-

```
>>>print ('The float number is : {:*<+10.2f}'.
        format (2345567.6789))
```

O/P:- The float number is : +2,345,567.68.

List :-

Creation of Lists :-

List is a flexible ordered collection of objects. The objects are separated by commas and enclosed within square brackets.

The elements in list can be of any data type, such as:

* numbers
* strings
* other lists
* tuples
* sets
* dictionaries.

syntax :-

variable = [objects separated by comma]

The created list can be modified. That means lists are mutable.

eg :- (i) l = [1, 2, 3, 4, 5]

(ii) l1 = ['sam', 52, 35 'cat']

(iii) l2 = [] # empty list

properties of Lists :-

(i) Lists are ordered collections of arbitrary objects.

(ii) They follow left to right positional ordering.

(iii) Individual elements of the lists are accessed by using index.

(iv) List are mutable.

Accessing a List :-

l = [42, 35, 'Good', 72.5, 85.3]

42	35	'Good'	72.5	85.3
0	1	2	3	4
-5	-4	-3	-2	-1

0 - 4 --> positive index

-4 ... -1 --> negative index.

Types of list access:

(i) Accessing individual elements in a list

(ii) Slicing a list.

(i) Accessing individual elements in a list :-

Syntax:-

variable name [index]

Eg:-

(i) print (l [2]) = Good

(ii) print (l [0]) = 42

(iii) print (l [-5]) = 42

(ii) slicing a list:- slicing is a process of taking a range of list elements from the given list.

Syntax:- Variable name ([start] : [stop])

eg :-

(i) print (l [:3]) o/p:- [42, 55, 'Good']

(ii) print (l [2 :4]) o/p:- ['Good', 72.5]

(iii) print (l [-3 :]) o/p:- ['good', 72.5, 85.3]

Traversing a list:-

Traversing a list is a process to visit all elements in the list exactly once to do any one of the following.

* to read the element.
* to print the element.
* to process the element.

This can be implemented using for loop or while loop.

eg: l = [5, 8, 6, 7, 1]

for i in l:
    print(i, end = " ")

o/p: 5 8 6 7 1

copying the list:-

copy() method is used to take a copy of the existing list. The process of taking a copy of the existing list is also called as list cloning

Syntax:- newlist = listname.copy()

eg:- x = [1, 2, 3, 4]
y = x.copy()
print("the existing list is :", x)
print("the copied list is :", y)

o/p:-
The existing list is : [1, 2, 3, 4]
The copied list is : [1, 2, 3, 4]

Altering values in the list:-

Altering is nothing but changing or modifying the existing values in the list.

(a) altering single vale:-

syntax:- listname [index of the value to be modified] = new value.

l = [42, 35, 'good', 'bad', 'sad']

l[1] = 50

Now the lis becomes l = [42, 50, 'good', 'bad', 'sad'].

(b) Altering Multiple values :

Syntax :-

listname [start index : stop index of the values to the modified] = new value

l = [40, 35, 'good', 'bad', 'Sad']

l [2 : 5] = 'kumar', 'babu', 'Ramu']

o/p: l = [40, 35, 'kumar', 'babu', 'Ra

Deleting Elements from List :-

del () function is used to delete element or range of elements or a whole list from memory.

(a) to delete any element or range of el

del listname [index or range]

(b) To delete the whole list from memory.

del (list name)

eg :- t = [5, 4, 3, 6, 7, 8, 9]

(a) print (del (t[0]))

[4, 3, 6, 7, 8, 9]

(b) print (del (t[0:3]))

[6, 7, 8, 9]

(c) print (del (t))

Name error : name t is not defined.

Built-in List Operators :-

There are three important built-i

List operators.

(i) Membership operators in and not in

(ii) Concatenation operator +

(iii) Repetition operator *

i) Membership operator :-

The membership operators in and not in are used to find whether the given elements is present in the list or not.

syntax :- element operator list name.

eg:- x = [17, 'hai', 97.5, 'cat', 'dog']

(a) 'cat' in x
true

(b) 57 not in x
true

(c) 'elephant' in x
false

ii) concatenation of the lists :-

Concatenation is a process of joining two lists to form a new list. This is carried out using the operator +.

syntax:- List 3 = List 1 + List 2

eg:- l_1 = [1, 2, 3]
l_2 = [4, 5, 6]
l_3 = l_1 + l_2
print l_3
[1, 2, 3, 4, 5, 6]

iii) Repetition operator :-

Repetition is a process of adding a list to itself number of times. This carried out using * operator.

syntax :- List variable * times.

eg :-
l_3 = [1, 2, 3, 4, 5, 6]
print (l_3 * 2)

o/p:- [1, 2, 3, 4, 5, 6, 1, 2, 3, 4, 5, 6]

Built-in Methods:

The following are the various built-in methods to handle lists. To use these methods, it must be called by the list object.

Syntax:- listname.methodname()

(i) append() This method is used to add a new element at the end of the list. This method does not return any thing but updates the existing list.

Syntax:- listname.append(element).

eg:- l = [2, 4, 8, 16]
l.append(32)
print('updated list:', l)

o/p:-
Updated list: [2, 4, 8, 16, 32]

(ii) insert() This method is used to insert a new element at the specified index position in the given list.

Syntax:- listname.insert(index, element)

eg:- l = [2, 4, 8, 16, 32]
l.insert(2, 64)
Print('the new list is:', l)

o/p:-
the new list is: [2, 4, 64, 8, 16, 32]

(iii) Count():- This method is used to find the total occurance of a given element in the list.

Syntax: listname.count(element)

eg:
```
l = ['bird', 'cat', 'elephant', 'cat', 'bird', 'dog']
print (l.count('bird'))
```
o/p:- 2

(iv) extend():
This method is used to add the content of one list at the end of another list.

syntax:- l_1.extend(l_2)

eg:-
```
l1 = [1, 2, 3, 4]
l2 = [5, 6, 7]
l1.extend(l2)
l2.extend(l1)
print ('List l1 :', l1)
print ('List l2 :', l2)
```
o/p:-
List l_1 : [1, 2, 3, 4, 5, 6, 7]
List l_2 : [5, 6, 7, 1, 2, 3, 4]

(v) index():- This method is used to find the index of the first occurance of the element in the list.

Syntax :- listname.index(element)

eg:-
```
l = [1, 3, 7, 9, 3, 2, 11, 1, 9]
l.index(1)
```
o/p :-
0
```
l.index(9)
```
3

(vi) remove()

Syntax: listname.remove(element)

eg: l = [1, 2, 3, 4, 2, 4, 6, 7]

l.remove(2)

print(l)

o/p:- [1, 3, 4, 2, 4, 6, 7]

l.remove(100)

ValueError: list.remove(x): x not in list.

(vii) pop():-

Syntax: listname.pop(index)

eg:- l = [1, 2, 3, 4, 5, 6, 7]

l.pop()

o/p:- 7

(viii) reverse()

Syntax: listname.reverse()

eg:- l = [1, 2, 3, 4, 5, 6]

l.reverse()

print(l)

o/p:- [7, 6, 5, 4, 3, 2, 1]

(ix) sort()

syntax: listname.sort(reverse = true/false)

eg:- l = [33, 72, 27, 40, 11, 13]

l.sort()

print('the sorted list is:', l)

o/p:- the sorted list is: [11, 13, 27, 33, 40, 72]

(x) clear()

Syntax: listname.clear()

eg:- l = [2, 48, 'boy', [2, 3])

l.clear()

Print(l)

o/p:- []

Unit - IV

Tuples And Dictionaries

Tuples:

Creating a tuple:

Tuple is an ordered collection of objects. The objects are enclosed with in parantheses (). The elements in the tuple can be of any data type such as:

* numbers
* strings
* List
* other tuple
* Sets
* dictionaries.

Syntax:-

tuple variable = (objects separated by comma)

The created tuple cannot be modified.

Hence tuples are immutable.

eg:-

(i) t = (1, 2, 5, 10, 11, 14)

(ii) t1 = ('sam', 52.5, 27)

(iii) t6 = ([1, 2, 3], 23, 'ed')

Properties:-

(i) tuples are ordered collections of arbitrary objects.

(ii) Individual elements of the tuple are accessed by using index. So the user can do slicing and indexing.

(iii) tuples are immutable. So the tuples can't grow and shrink.

iv) tuples are best for data not to be modified throughout the life of the program.

Accessing a tuple:

l = (42, 35, 'Good', 72.5, 85.3)

42	35	'Good'	72.5	85.3
0	1	2	3	4
-5	-4	-3	-2	-1

0 - 4 → positive index

-1 - -5 → negative index

(i) Accessing individual elements in a tuple.

(ii) Slicing a tuple.

(i) Accessing individual elements in a tuple:-

syntax :- variable name [index]

eg :- l = (42, 35, 'Good', 72.5, 85.5)

(i) print (l[2]) = Good

print (l[-1]) = 85.3

print (l[-5]) = 42

(ii) slicing a tuple:-

Slicing is a process of taking a range of tuple elements from the given tuple.

syntax :- Variablename([start]:[stop])

eg :- l = (42, 35, 'Good', 72.5, 85.3)

(i) print (l[:3])

o/p :- (42, 35, 'Good')

(ii) print (l[2:4])

o/p :- ('Good', 72.5)

(iii) print (l[1:-3])

o/p :- 35

Assignment or Unpacking of tuples:-

Tuple can be considered as a packing structure of elements. Assigning or unpacking a tuple is a process of assigning the tuple elements to different variables.

There are three types of Unpacking.

(i) Basic Unpacking or assign

(ii) Unpacking Using _

(iii) Unpacking Using *

(i) Basic Unpacking :-

By writing variables separated by comma on the left hand side and tuple name on the right hand side, the tuple elements can be assigned to the variables.

eg :-

```
t = (27, 'God', 37.8)
a, b, c = t
print (a)
print (b)
print (c)
```

O/P:-
27
God
37.8

(ii) Unpacking Using _ (underscore)

If all the tuple elements are not needed, the unnecessary elements can be assigning to _ (underscore) instead of variables. It has no meaning.

eg:-

```
t = (27, 'God', 37.8)
a, _, b = t
print (a, _, b)
```

O/P:- 27, 'God', 37.8

(iii) Unpacking using *

If the number of variables on the left hand side is less than elements, the remaining tuple elements can be assigned as a list to a variable preceded by an * as * variable.

eg:-

```
t = (5, 32, 72, 'God', 37)
a, b, *c = t
print (a, b, c)
```

O/P:- 5 32 72, God, 37

```
print (*t)
```

O/P:- 5 32 72 God 37.

Basic tuple operations:-

(i) find whether a given element is present in the tuple or not.

(ii) Concatenation of the tuple.

(iii) Repeating

(iv) Iterating

(i) finds whether a Given Element is present in the tuple or not:-

The membership operators in and not in are used to find whether the given element is present in the tuple or not.

Syntax: element operator tuple name.

eg:- l = (17, 'bat', 97.5, 'cat', 'dog')

(a) 'cat' in l

true

(b) 57 not in l

true.

(ii) concatenation of the tuple :-

Concatenation is a process of joining two tuples to form a new tuple. This is carried out using the + operator.

Syntax :- tuple 3 = tuple 1 + tuple 2.

eg :- $t_1 = (1, 2, 3)$

$t_2 = (4, 5)$

$t_3 = t_1 + t_2$

print (t_3)

o/p :- (1, 2, 3, 4, 5)

(iii) Repeating the tuple :-

Repeatition is a process of adding a tuple to itself number of times. This is carried out using * operator.

Syntax :- tuplevariable * times.

eg :- $t_3 = (1, 2, 3, 4)$

$t_3 * 2$

o/p :- (1, 2, 3, 4, 1, 2, 3, 4)

(iv) Iteration :-

tuples can be used in for loop for doing iteration operation.

Syntax :-

for loop variable in tuple :

eg :- for i in (2, 4, 6, 8) :

print (i, end = ' ')

o/p :- 2 4 6 8.

Immutable property:

Created string objects are immutable. It means the content of the string objects cannot be changed after it has been created. The created string can be empty.

If we try to alter the value, it gives an error message.

eg:-

Consider the string S = 'welcome'. If we alter the third character 'l' to 'e' as, S[2] = 'e'.

It gives an error as type error: 'str' object does not support item assignment.

Built-in tuple functions:-

The following are the various built-in functions to handle tuples. To use a function, it must be called by its name.

syntax:- functionname()

(i) len() This function is used to find the length of the tuple.

syntax :- len (var)

eg:-

```
t = (2, 5, 8, 9, 4)
print (len (t))
```

o/p:- 5

(ii) max() This function is used to find the maximum value element in the tuple.

syntax : max (var)

eg:- t = (20, 7, 5, 49, 72)
max (t)
o/p:- 72

(iii) min()
This function is used to find the minimum value element in the tuple.
Syntax : min(var)

eg:- t = (20, 7, 5, 49, 72)
min (t)
o/p:- 5.

t = ('Edwin', 'shanthi', 'Reni', 'Irene')
min(t)
o/p:- 'Edwin'.

(iv) tuple()
This function is used to convert any sequence types such as string, list etc to tuples.
Syntax :- tuple (seq)

Eg:- s='python'
tuple (s)
o/p:- ('p', 'y', 't', 'h', 'o', 'n')

Returning tuples:-
like any other data type, a function can return a tuple to the calling point.
Syntax : return (tuple variable)

eg:-
```
def example(s):
    s = s + 'luck'
    t = tuple(s)
    return(t)
```

```
x = example ('Good')
print ('the returning tuple is:', x)
```

o/p:-

the returning tuple is: ('G', 'o', 'o', 'd', 'l', 'u', 'c', 'k')

Tuple as Arguments:

Tuple can be passed as arguments to the defined function.

Syntax:- functionname (tuple variable)

eg:-

```
def example (t):
    print (t)
t1 = (4, 8, 9, 10)
example (t1)
```

o/p:-

(4, 8, 9, 10).

Variable length Arguments:-

If the number of arguments are known during function definition it can be passed to function using positional arguments.

But in some situation the number of arguments are not known during function definition.

During this time, variable length parameters can be used to receive the variables.

Syntax:- def functionname (*args):

```
    statement 1
    . . . . .
    statement n
    return (expression)
```

* args is the name of the variable and it receives all the variables length arguments and store it as a tuple.

eg:- def example (*t):
    print (t)
example (1,2,3,4,5)

o/p:- (1,2,3,4,5)

From the output it is clear that * variable stores the variable length arguments as a tuple.

4.2 : Dictionaries:-

Creating dictionary:-

Dictionary is a flexible unordered collection of objects. The objects are enclosed with in curly brackets {}. Dictionaries are mutable.

general form:-

variable = {key : value , key : value, ...}

eg:-
(i) d = {}
(ii) d1 = {1001 : 'Ram', 1002 : 'Bala'}

Properties of Dictionary:-

(i) Dictionaries are unordered collections of arbitary objects.
(ii) The dictionary can be accessing using its key value.
(iii) Dictionaries are mutable. A dictionary element can be another dictionary.

Accessing dictionary elements:

A dictionary can have numbers, string, list, tuples, sets etc..

The elements of the dictionary can be accessed by giving elements key value inside square brackets[].

syntax:- dictionary name [key value].

eg:- d1 = {1:2, 2:8, 9, 'TOOl': 'pen', 'List': [1,2,3], 'Tuple': (9,8,7), 'Set': {5,6,7}}

(i) ~~del~~ d1[1]

o/p:- 2

(ii) d1['Tool']

o/p:- 'pen'

(iii) d1['List']

o/p:- [1,2,3]

Operations in dictionary:-

(i) finding the length of the dictionary

(ii) Incremental building of a dictionary

(iii) Changing dictionary elements in-place.

(iv) Iteration.

(v) Creating dictionary using any sequence of key-value pairs.

(i) finding the length of the dictionary:-

len()

len() function is used to find the length of the dictionary.

Syntax:- len(x)

eg:- (i) d = {}
x = len(d)
print ('length of d =', x)

o/p:- length of d = 0

ii) d1 = {1: 'dog', 2: 'cat'}
x = len(d1)
print ('length of d1 =', x)

o/p:- length of d1 = 2

(ii) Incremental building of a dictionary:-

The process of creating a dictionary by adding its elements one by one in an empty dictionary is called incremental building of a dictionary.

eg:-
d = {}
d[1001] = 'Ram'
d[1002] = 'Bala'
d[1003] = 'Kumar'
d[1004] = 'Shanthi'
print ('the created dictionary d =', d)

o/p:-
The created dictionary d = {1001: 'Ram', 1002: 'Bala', 1003: 'Kumar', 1004: 'shanthi'}

(iii) Changing dictionary elements in-place.

Since dictionary is mutable, the user can change the element value.

Syntax:-
dictionary name [key] = new value.

eg:-
d = {1001: 'Ram', 1002: 'Bala', 1003: 'Kumar', 1004: 'shanthi'}

d[1001] = 'mahesh'
# changed
print ("the changed dictionary d:", d)

O/P:- the changed dictionary d = {1001: 'Ram', 1002: 'mahesh', 1003: 'kumar', 1004: 'Shanthi'}

(iv) Iteration:-

Dictionary can be used in for loop for doing iteration operation.

* Iteration using keys
* Iterate using values.
* Iterate using key value pair.

eg:-1, Iterate using key

d = {1:100, 2:200, 3:300, 4:400}

```
for key in d:
    print (key, end = ' ')
```

O/P:-

1 2 3 4

eg:2 Iterate using values:-

```
for value in d.values():
    print (value, end = ' ')
```

O/P:- 100 200 300 400.

eg3: Iterate using key-value pair:-

```
for key, value in d.items():
    print ('{}:{}'.format(key, value))
```

O/P:-

1:100
2:200
3:300
4:400

(v) Creating dictionary using any sequence of key-value pair.

Using dict() built-in function dictionary can be created

- The argument of the dictionary must contain a sequence of key value pair.

syntax: dictionary name = dict (x)

eg:- creating dictionary list with tuple.

```
d = dict ([(1001,'Ram'), (1002, 'Bala'),
           (1003,'kumar'), (1004,'shanthi')])
print ('created dictionary d is', d)
```

o/p:-

created dictionary d is {1001:'Ram', 1002:'Bala', 1003:'kumar', 1004:'shanthi'}

eg:- creating dictionary using tuple with list as elements.

```
d = dict (([1, 100], [2, 200]))
print ['created dictionary d is', d)
```

o/p:- created dictionary d is {1:100, 2:200}

Updating dictionary:-

update() method is used to insert elements from a dictionary or an iteratable with key-value pair to another dictionary.

syntax:- dictionaryname.update(value)

eg:-

```
d = {1:100, 2:200, 3:300}
d1 = {4:400, 5:500}
d.update(d1)
Print(d)
```

o/p:- {1:100, 2:200, 3:300, 4:400, 5:500}

Deleting element from dictionary:

del() function is used to delete a key-value pair from dictionary.

Syntax: del dictionaryname[key]

eg: d = {1:100, 2:200, 3:300, 4:400}
del d[3]
print ('The dictionary after deleting key 3 is', d)

O/P: The dictionary after deleting key3 is
{1:100, 2:200, 4:400}

Built-in dictionary methods:

The following are the various built-in methods to handle dictionaries. To use these Methods, it must be called by the dictionary name

Syntax: dictionaryname.methodname()

(i) copy()

This method is used to take a copy of the dictionary.

Syntax: n = dictionaryname.copy()

eg:-

d = {1001 : 'Ramu', 1002 : 'babu', 1003: 'kumar', 1004 : 'shanthi'}
new = d.copy()
print ('Existing dictionary : d = ', d)
print ('copy of d is : new = ', new)

O/P:- Existing dictionary : d = {1001 : 'Ramu', 1002 : 'Babu', 1003 : 'Kumar', 1004 : 'shanthi'}

copy of d is : new = {1001 : 'Ramu', 1002 : 'Babu', 1003 : 'Kumar', 1004 : 'Shanthi'}

(ii) pop()

The method is used to remove an element specified by the key

Syntax:- dictionaryname.pop(key, default value)

eg:-

```
d = {1:100, 2:200, 3:300}
x = d.pop(2)
print("The removed element is:", x)
print("The dictionary after removing the element is:", d)
```

o/p:- The removed element is: 200
The dictionary after removing one element is:
{1:100, 2:200}

(iii) fromkeys:- This method is used to create a dictionary with the given keys.

Syntax: dict.fromkeys(key, value)

eg:-

```
l = [1, 2, 3, 4]
d = dict.fromkeys(l)
print("The created dictionary d =", d)
```

o/p:- The created dictionary d = {1: None, 2: None, 3: None, 4: None}

(iv) clear():- This method is used to remove all elements from the dictionary.

Syntax: dictionaryname.clear()

eg:-

```
d = {1:100, 2:200, 3:300}
d.clear()
print("The dictionary d is:", d)
```

o/p:-

The dictionary d is: {}

(v) get() This method is used to get the element value of the given key

Syntax: dictionaryname.get(key, value)

eg:
```
d = {1:100, 2:200, 3:300}
item = d.get(2, 'key not present')
print('the element value is:', item)
```
O/P:- the element value is 300

(vi) items() This method is used to view the key-value pairs of the given dictionary as tuples in a list.

Syntax: dictionaryname.items()

eg:-
```
d = {1:100, 2:200, 3:300}
x = d.items()
print(x)
```
O/P:- dict_items([(1,100), (2,200), (3,300)])

(vii) key() This method is used to return the keys in the dictionary as elements of list.

Syntax: dictionaryname.keys()

eg:
```
d = {1:100, 2:200, 3:300}
k = d.key()
Print(k)
```
O/P:- dict_keys([1, 2, 3]).

(viii) values()

This method is used to return the values in the dictionary as elements of list.

Syntax: dictionaryname.values()

eg: d = {1:100, 2:200, 3:300}
k = d.values()
print(k)

o/p:- dict_keys([100, 200, 300])

ix) popitem()

This method is used to remove the last item from the dictionary. The removed item will be returned as a tuple.

Syntax: dictionaryname.popitem()

eg: d = {1:100, 2:200, 3:300}
k = d.popitem()
Print ('The removed item:', x)

o/p:- the removed item : (3, 300)

x) Setdefault():

This method is used to return the value of the element for the given key. If the key is not in the dictionary, the given key and given value will be inserted in the dictionary.

Syntax: dictionaryname.Setdefault(key, value)

eg:
d = {1:100, 2:200, 3:300}
x = d.Setdefault(2, 500)

UNIT - V

Files and Exception handling

1. Files:

Introduction:

So far input and print function used to read data into the variables and write data from the variables using visual display unit.

Giving input and getting output to and from programs using functions is not efficient. To overcome this problem, data are stored permanently in secondary storage devices such as hard disk, dvd disc or in any other permanent storage device using the concept of files.

file types:-

(i) Text files

(ii) binary files.

(i) In text files data are stored in alphabet, number format. These files are human-readable.

eg: .pdf files, .text files etc.

(ii) Binary files stored data in binary format (0 to 1). These files cannot be read by humans.

eg: Image files, video files etc.

File Operation :-

A file is a collection of related data stored permanently in a particular area on the disk. Storing and Managing the data in the file is called file processing.

(i) opening the file (ii) Reading the file data (iii) writing data into file (iv) closing the file (v) Renaming a file (vi) Deleting a file.

Opening a file :-

A file must be opened before doing any file processing operation such as read, write etc.

Syntax : Variable = open (filename [,mode])

Reading modes :-

r - Open a text file for reading
rb - open a binary file for reading
r+ - open a text file for reading & writing.
rb+ - open a binary file for reading & writing

Writing modes :-

w - open a text file for writing
wb - open a binary file for writing
w+ - open a text file for writing & reading
wb+ - open a binary file for writing & reading
x - exclusive mode. open a text file for writing.

append modes :-

a - open a text file for appending
ab - open a binary file for appending
a+ - open a text file for appending and reading
ab+ - open a binary file for appending and reading

During execution of this statement, it returns a file object called file handle as file pointer and it is assigned to the left hand side variable. Using this object the user can perform operations on the opened file.

properties :-

(i) The default open mode is r. If the file does not existing it gives an error.

(ii) In read and write mode, the file pointer will be placed at the beginning of the file.

eg :-

(i) a = open ('Mydata.txt', 'r')

(ii) b = open ('test.txt', 'w')

(iii) c = open ('D:\users\.abc.txt')

ii) Closing the file :-

After all the operations on the opened file are completed, it must be closed. This is an optional process.

syntax : variable.close()

eg :- b = open ('abc.txt', 'r')

c = open ('xyz.txt', 'wt')

.....

b.close()

c.close()

(iii) writing Data to the file:

writing is defined as a process of storing string data in the opened file. the data is written from the place whome the file pointer is pointing.

There are two different methods

(a) write (b) writelines.

a) write :- this method is used to write a string data.

Syntax : fileobject. write (str)

eg:-

```
S=open ('student.txt', 'w')
S. write ('Ram')
S. write ('1001')
S. write ('2-computer')
print ('file student created sucessfully')
```

O/P:- Ram1001 2- computer
file student created successfully.

(b) writelines ()
This method is used to write a list of string to the file.

Syntax : fileobject. writelines (listvariables)

eg:-

```
K = Open ('example.txt', 'w')
L = ['Ram\n', '1001\n', '2-computer\n']
K. writelines (L)
Print ('file student using List
Created Successfully')
```

f.close()

O/P:- Ram
1001
2 - computer

file student using list created successfully

Reading data from the file:-

Reading is defined as a process of accessing the stored data from the opened file.

The following are the different methods available for reading:-

(a) read() (b) readline() (c) readlines()

(a) read():-

This method is used to read the whole file data or the specified size of data from the file.

Syntax: fileobject.read([size])

eg:-

```
S = open('student.txt', 'w')
S.write('Ram\n')
S.write('1001\n')
S.write('2 - computer\n')
S.close()
S = open('student.txt', 'r')
print(S.read(2))
Print(S.read())
Print(S.read())
S.close()
```

o/p: Ram → output for first read()
1001 → output for second read()
computer → o/p for third readline() empty string

(b) readline()

This method is used to read one line of data from the file or the specified size of data from the line.

Syntax: fileobject.readline([size])

eg:-

```
s=open('student.txt','r')
print(s.readline())
print(s.readline())
print(s.readline())
s.close()
```

o/p:-

```
Ram
1001
2-computer.
```

(c) readlines():

This method is used to read the entire file content as a list containing each line as its elements.

Syntax: fileobject.readlines()

eg:-

```
s=open('test.txt','w')
L=['apple\n','orange\n','guava\n','banana']
s.writelines(L)
s.close()
s=open('test.txt','r')
ll=s.readlines()
print(ll)
```

o/p:- ['apple\n', 'orange\n', 'guava', 'banana']

writing and reading files (using loop) statement:

To write/read a file with hundred lines of data, hundred write/read statements are needed.

It is cumbersome and so is not a useful programming practice. To overcome this, looping statements are used and this makes the process very easy.

During each iteration a line is read or written from or to the file.

To write n lines of data :-

Using while loop :-

```
S = open ('test.txt', 'w')
n = int (input ('Enter the number of data:'))
while n != 0 :
    data = input ('Enter the next data:')
    S.write (data + '\n')
    n = n - 1
S.close()
```

O/P :- Enter the number of data : 3
Enter the next data : 3
Enter the next data : 2
Enter the next data : 1

To read n lines of data :-

Using for loop :-

```
S = open ('test.txt', 'r')
for i in S:
    print (i)
S.close()
```

file renaming

file renaming is defined as a process to change the name of the already existing file into a new name this is implemented using the rename() method present in the OS module

Syntax: import os
os.rename(existing file location, new name).

eg:-
```
import OS
OS.rename('example.txt', 'Modified.txt')
print ('file renamed')
```
o/p:- file renamed.

file Deleting:-

file deleting is defined as a process to remove the file from the directory this is implemented using the remove() method present in the OS module.

Syntax: import os
os.remove(file location)

eg:-
```
import OS
OS.remove('D:/Perfect.txt')
print('file deleted successfully')
```
o/p:- file deleted successfully.

File object attributes:

In python there are predefined attributes to get the information about the opened file. The opened file, file object is used to get this information.

Syntax : fileobject.attributename.

name - variable to get the name of the opened file.

mode - variable to get the mode of the opened file.

Closed - variable to check whether the file is closed or not. If closed it returns true else false.

readable() - method to check whether the file is readable or not. If readable it returns true else false.

writable() - method to check whether the file is writable or not. If writable it returns true else false.

eg :-

```
p = open('xyz.txt', 'w')
print('Name of the file :', p.name)
print('Mode of the file :', p.mode)
print('Is readable :', p.readable)
print('Is writable :', p.writable)
print('Is closed :', p.closed)
p.closed()
print('Is closed :', p.closed)
```

o/p :-

```
Name of the file : xyz.txt
Mode of the file : w
Is readable : false.
```

is writable : true
is closed : false
is closed : true

File handling Methods and functions:

Apart from Methods used for writing and reading, there are many more methods and functions to do other file handling operations.

methods :-

(i) fileno() (ii) next() (iii) strip()

(iv) seek() (v) tell()

i) fileno() :- This method is used to get the file number called file descriptor.

file descriptor is an integer number which gives the file number in the open files table kept by the Operating System.

Syntax : fileobject.fileno()

eg :- s = open ('test.txt', 'r')
print (s.fileno())

o/p :- 4.

ii) next() :- This function is used to read the next line of data from the file. If it is used within looping statement all lines of data can be read from the file.

Syntax:- variable = next (fileobject)

eg:-

Consider the student.txt file with the data:

Ramu, 1001, Mechanical, second year

Rala, 4571, Computer, third year

Kumar, 2341, EEE, first year.

(i) S = open ('student.txt', 'r')

x = next (S)

print ('first line of data:', x)

O/P:-

first line of data: Ramu, 1001, Mechanical, second year.

(iii) strip():-

This method is used to remove the leading and trailing blank spaces including tabs (\t).

Syntax:- fileobject.strip()

eg:-

S = open ('student.txt', 'r')

for i in range (3):

x = next (S)

print ('line {} data is {}'.format (i, x.strip()))

O/P:-

Line 0 data is Ramu, 1001, Mechanical, second year.

Line 1 data is Rala, 4571, Computer, third year.

Line 2 data is Kumar, 2341, EEE, first year.

(iv) seek() :- This method is used to change the file object location to any desired location within the file.

Syntax : file object . seek (offset)

eg :-

```
s=open('student.txt','r')
s.seek(10)
print(s.read())
```

O/P :-

Mechanical, second year

Bala, 4371, computer, third year.

Kumar, 2341, EEE, first year.

v) tell() :- This method is used to return the current position of file object in the given file.

Syntax : fileobject . tell()

eg :-

```
s=open('student.txt','r')
print(s.readline().strip())
pos=s.tell()
print('Current file pointer position is:', pos)
```

O/P :-

Ramu, 1001, Mechanical, second year.

Current file pointer position is : 34

A.9: Directory:

A directory is a place where folders and files are stored. In a computer there can be more than one directory in the various drives of the hard disk.

The User can store their files in any one directory present in any one of the drives.

In python there are number of methods to manipulate directories.

These Methods are present in the built-in module named os.

Methods:-

(i) getcwd()
(ii) chdir()
(iii) Mkdir()
(iv) rmdir()
(v) rename()
(vi) Listdir()

(i) getcwd():- this method is used to display the current working directory.

General form: import os
os.getcwd()

eg:-

```
import os
cwd = os.getcwd()
print ("the Current working directory is:")
print (cwd)
```

O/P:-

The Current working directory is:

C:\Users\user\AppData\Local\Programs\Python\Python37-32

(ii) chdir():

This method is used to change the current working directory to another directory.

Syntax: import os
os.chdir(directory name)

eg: import os
os.chdir('D:\Downloads')
cwd = os.getcwd()
print(cwd)

O/P:-
D:\Downloads.

(iii) mkdir(): This method is used to create a new directory. If path is not specified the new directory will be created in the current working directory.

Syntax: import os
os.mkdir(name)

eg:-
import os
os.mkdir('D:\python')

This statement creates a new directory named python in the D:drive.

(iv) listdir():

This method is used to list all files and sub-directories inside the given directory.

If path is not specified the current directory content will be listed, else the directory content of the specified path is listed.

Syntax: import os
os.listdir(name)

eg:-
```
import os
print(os.listdir('D:/Downloads'))
```

O/P:-
```
['13288607_1700411585354600 4_3288 7214_
0.jpg', '13431522_1711698849637270_
n.jpg',
'. . . . . ',
'. . . . . ']
```

(v) rmdir() This method is used to delete an empty directory. If files are present inside the directory to be deleted, an error occurs.

Syntax: import os
os.rmdir(name)

Eg:-
```
import os
os.rmdir('D:/python')
print('the specified directory deleted')
```

O/P:-
The specified directory deleted.

Q.2 Exceptions in Python:

Errors:

Errors are mistakes in programs and these make the program behave abnormally. Normally the following errors occurs while writing programs:

* Syntax error (or) compiletime error
* Logical error
* Runtime error or exception

Syntax error (or) Compile time error:-

When the programmers violates the language rules. This happens because of poor understanding of the language. For example, in python if the colon (:) after the if statement is missed it gives a syntax or Compile time error.

Logical error:- Logical error occurs because of poor understanding of the problem by the programmer.

These errors can be identified by the wrong output for right input.

Runtime error or exception:

It is defined as an error that occurs during runtime. There are two types of exceptions namely, Synchronous and asynchronous.

(i) Synchronous exception:-

* division by zero

* accessing the maximum length
the index.
* And N memory.
* Disk space problem.
* Incompatible data.

(ii) asynchronous exception:
* file error
* keyboard problem.
* internal problem

eg: Python language exceptions:-

(i) >>> 87/0
traceback (most recent call last):
file "<pyshell#0>", line 1, in <module>
87/0
ZeroDivisionError: division by zero.

(ii) >>> a = 70
>>> b = a+b
traceback (most recent call last):
file "<pyshell#2>", line 1, in <module>
b=a+c
Name error: name 'c' is not defined.

(iii) >>> s = 'good'
>>> s[5]
traceback (most recent call last):
file "<pyshell#5>", line 1, in <module>
s[5]
IndexError: string index out of range.
>>>

Built-in EXCEPTIONS:

Built-in exception classes which belongs to python language for handling standard errors that occurs during run time. All these classes are derived from various base classes.

Exception	Description.
Assertion error	Raised when an assert statement fails.
Attribute error	Raised when an attribute reference or assignment fails.
EOFError	Raised when the input() function hits an end-of-file condition (EOF) without reading any data.
floating point Error	Raised when a floating point operation fails.
Import error.	Raised when the important statement fails when trying to load a module.
Index error.	Raised when a sequence subscript is out of range.
Key Error.	Raised when a mapping key is not found in the set of existing keys.

memory error	Raised when an operation runs out of memory but the situation may still be rescued.
Name error.	Raised when a local or global name is not found.
OS Error ()	This exception is raised when a system function returns a system related errors.
Runtime error.	Raised when an error is detected that does not fall in any of the other categories.
Syntax Error	Raised when the parser encounters a syntax error.
Indentation Error.	Base class for syntax errors related to incorrect indentation.
System Error.	Raised when the interpreter find an internal error.
Type error.	Raised when an operation or function is applied to an object of inappropriate type.
Value error.	Raised when a built-in operation or function receives an argument that has the right type but an inappropriate value

Need for exception handling :

A computer program or a software is a collection of statements. These statements are divided into two types.

* Normal statements
* Critical statements.

Normal statements are statements that will not give exceptions or runtime errors. Critical statements are statements which contain the posibility of runtime errors or exceptions.

eg :-
```
def example(a,b):
    c = a+b
    print ("The value of a+b is :", c)
    d = a/b
    print ("The value of a/b is :", d)
    e = a*b
    print ("The value of a+b is :", e)
a = int (input ('Give the value of first number :'))
b = int (input ('Give the value of second number :'))
example(a,b)
```

O/P :-
```
Give the value of first number : 56
Give the value of second number : 0
The value of a+b is : 56
Traceback (most recent call last):
  File "C:/Users/user/AppData/Local/Programs/Python/Python37-32/exp1.py", line 12, in <module>
    example(a,b)
```

File "C:/Users/User/AppData/Local/Programs/Python/Python37-32/exp1.py", line 4, in

example,

d = a/b

Zero division error: Division by zero.

from the output it is clear that the output for c = a+b = 5 is calculated normally. But for the statement d = a/b the python language by default gives a run time error as:

Zerodivision Error: division by zero.

Handling Exception:-

python has the following types of execution handling from to deal with the abnormal situation during run time of a program.

(a) try .. except with no exception.

(b) try .... except with multiple exception.

(c) try ..... except with no exception:-

This form is used to handle any type of exception.

Syntax:-

```
try:
    critical statements
except:
    exception handling statements
Next statements.
```

This statement catches all type of exceptions raised by the critical statement in the try block followed by next statement

But this does not give any information about the type of the error that occured

eg:
```
try:
    a = int (input ('Enter the value of a :'))
    b = int (input ('Enter the value of b :'))
    c = a/b
    print ('the value of a/b is :', c)
except :
    print ('Exception occcurs')
```

→ In this program there is a possibility for the following two errors to occurs during runtime.

(i) Value error
(ii) Zerodivision error.

O/P:- 1
Enter the value of a : 20
Enter the value of b : 8
the value of c is : 2.5

In this, no exception occurs during run time, so the program ends normally.

O/P: 2
Enter the value of a: P
Exception occurs.

In this, exception occurs during runtime because of character input instead of integer output. But the program ends normally by giving the message Exception occurs.

(b) try.... except with multiple exceptions:

- the general form of try statement to handle multiple exception using single except block is:

Syntax:

```
try:
    critical statements
except (exceptionname1 [, exceptionname2, ...
                exceptionnamen]):
    exception handling statements
Next statements.
```

- name of the excepted exceptions in the try block. the exception names with in square brackets [] are optional.

During the execution of the try block if any exception is raised, the raised exception is compared with the exception names in the except clause.

eg:-

```
try:
    a = int (input ('Enter the value of a:'))
    b = int (input ('Enter the value of b:'))
    c = a / b
    print ('the value of c is:', c)
except (ValueError, ZeroDivisionError):
    print ('Error occurs in the try block')
```

the output will be the same for the two types of exception because they are handle by the same except clause.

try.... except.... finally:-

Syntax:

```
try:
    critical statements
exception:
    statements to handle exception
finally:
    Statements to be executed whether
        exception occurs or not.
Next statements
```

finally clause is an optional block in exception handling statement and it must appear at the last. This block will be executed whether an exception is raised or not in the try block.

eg:-

```
try:
    a=int(input('Enter the value of a:'))
    b=int(input('Enter the value of b:'))
    c=a/b
    print('The value of a/b is:', c)
except Exception as abc:
    print('The description in variable abc is:', abc)
finally:
    d = a+b
    print('The value of a+b=', d)
```

O/P:-

Enter the value of a:90

Enter the value of b:0

the description in variable abc is: division by zero

the value of a+b = 90

Enter the value of a: 56

Enter the value of b: 8

the value of a/b : 7.8

the value of a+b = 64

From the outputs it is clear that whether exception occurs or not the finally block is executed.

Raising Exception:-

The user can explicitly raise exception using raise statement.

Syntax :- raise exception classname ([value])

The raise statement can be used in two ways.

(i) Used within try statement.

(ii) Used without try statement.

If used within try statement the raised exception will be handled by the corresponding except block. This behave like normal exception handing.

If used without try statement, the program comes to a halt and displays the raised exception.

(i) Raise statement within try block:-

```
try:
    a = int(input("Enter the value of a:"))
    b = int(input("Enter the value of b:"))
    if b==0:
        raise ZeroDivisionError
    c = a/b
    print("the value of c is:", c)
except ZeroDivisionError:
    print("Error occurs due to zero value for b")
except ValueError as x:
    print(x)
```

O/P:-

Enter the value of a: 90
Enter the value of b: 0
Error occurs due to zero value for b
Enter the value of a: t
invalid literal for int() with base 10: 't'.

(ii) Raise statement without try block:

```
x=20
y=90
if y>x:
    raise Exception("value y must be
    less than x")
```

O/P:-

```
Traceback (most recent call last):
  file "C:/Users/user/appData/local/Programs/
Python/Python37-32/ex.py", line 4, in <module>
Exception: value y must be less than x
```

User defined exceptions:

In the previous topics we have discussed the process of handling the exceptions raised in the programs using built-in exception classes.

But in python there is a facility to define our own exception classes and can be raised explicitly to meet our programming requirements.

eg:- An account holder must maintain a minimum balance of RS 5000/- in the account. write a code to raise an exception if the balance falls bellow 5000/-

```
wamont = float (input ('Enter the amount to
                withdraw :'))
balance = balance - wamont.
if balance < 5000:
    raise MinimumBalanceException()
```

In this code the minimum Balance exception is not a built in exception, so during execution the raise statement gives an error.

this can be overcome by informing the new user defined exception to the python language by defining it before using.

Defining User - defined exception:

The steps given below are followed to define User - defined exception classes.

i) Define a desired class in the name of the new exception using any one base class in the exception hierarchy either directly or indirectly. Generally in python the base class Exception is used.

Syntax:-

```
Class newexceptionname (Exception):
    pass
```

ii) The defined exception can be raised explicitly by using raise statement.

Syntax:

```
raise newexceptionname ([value])
```

iii) When the exception is raised the program terminates by giving the raised exception class name as output.

iv) If the description is given for the value Argument it will also be printed along with the exception name.

Eg:-

```
Class Usererception (Exception):
    pass
def example (x):
    if x == 0:
        raise Exception ('Number Zero Exception')
```

O/P:-

Enter the value of a: 2
Enter the value of b: 7
Enter the value of c: 7
There are two roots -4.0 and -14.0

(2) program to count the number of characters and words in a string.

```
S=input ('Enter the string:')
Characters = len(s)
S1 = S.split (' ')
words = len(S1)
print ('Number of characters:', Characters)
print ('Number of words:', words)
```

O/P:-

Enter the string: God loves you
Number of Charaters: 13
Number of words: 3

(3) Program to read a collection of words entered by the user and add it in the list without duplicate words until the user enters blank line.

```
a=[]
print ('Enter the words one by one or
        enter blank line to quit:')
x = (input ())
while x != '':
```

```
if x not in a:
    a.append(x)
    x=(input())
print("the created list without duplicates
                                is:")
print(a)
```

```
RESTART: C:/Users/Admin/Desktop/dup.py
Enter the words one by one or enter
blank line to quit:
ramu
raj
bala
somu
ramu
raj

the created list with out dupicate is:
['ramu', 'raj', 'bala', 'somu']
```

program to print the keys and the values of a dictionary separately.

```
n=int(input("Enter the value of n:"))
d=dict()
for i in range(1, n+1):
    d[i]=i**3
print("The dictionary is", d)
print()
for (key, value) in (d.items()):
    print('{} key of value {}'.format(key,value)
```

O/p:

Enter the value of n: 5

the dictionary is {1: 1, 2: 8, 3: 27, 4: 64, 5: 125}

1 key to value 1

2 key to value 8

3 key to value 27

4 key to value 64

5 key to value 125

program to multiply the given numbers by ignoring non-numeric input.

```
value = 1
v1 = input ('Enter the next value to multiply:')
while v1 != '':
    try:
        v1 = float(v1)
        value = value * v1
    except ValueError:
        print ('Give a valid number:')
        v1 = input ('Enter the next value to
                    multiply:')
print ('the product of the given numbers is:'
                                    , value)
```

O/p:-

Enter the next value to Multiply: 4

Enter the next value to multiply: 5

Enter the next value to multiply: p

Give a valid number

Enter the next value to multiply : 3

Enter the next value to multiply :

The product of given numbers is: 60.0

program to convert the data in the given file into Upper Case by including file not found exception.

```
name = input ('Enter the name of the file:')
try:
    with open (name, 'r') as f:
        for line in f:
            print (line.upper())
except file not found error:
    print ('Give a valid text file').
```

O/P:-

Enter the name of the file : test.txt

PLANTAIN

Enter the name of the file : xyz

Give a valid text file.