Computer Science (083)
CLASS-XII 2021-22

DISTRIBUTION OF MARKS:

UNIT	UNIT NAME	MARKS
I	**Computational Thinking and Programming - 2**	**40**
II	Computer Networks	10
III	Database Management	20
TOTAL		70

• Revision of Python topics covered in Class XI.
• Functions: types of function (built-in functions, functions defined in module, user defined functions), creating user defined function, arguments and parameters, default parameters, positional parameters, function returning value(s), flow of execution, scope of a variable (global scope, local scope)
• Introduction to files, types of files (Text file, Binary file, CSV file), relative and absolute paths
• Text file: opening a text file, text file open modes (r, r+, w, w+, a, a+), closing a text file, opening a file using with clause, writing/appending data to a text file using write() and writelines(), reading from a text file using read(), readline() and readlines(), seek and tell methods, manipulation of data in a text file

• Binary file: basic operations on a binary file: open using file open modes (rb, rb+, wb, wb+, ab, ab+), close a binary file, import pickle module, dump() and load() method, read, write/create, search, append and update operations in a binary file
• CSV file: import csv module, open / close csv file, write into a csv file using csv.writerow() and read from a csv file using csv.reader( )
• Python libraries: creating python libraries
• Recursion: simple programs with recursion: sum of first n natural numbers, factorial, fibonacci series
• Idea of efficiency: number of comparisons in Best, Worst and Average case for linear search
• Data Structure: Stack, operations on stack (push & pop), implementation of stack using list. Introduction to queue, operations on queue (enqueue, dequeue, is empty, peek, is full), implementation of queue using list.

Python: It is widely used general purpose,high level programming language.Developed by Guido van Rossum in 1991. Python can be used to follow both Procedural approach andObject Oriented approach of programming

Features of Python:

- Easy to use – Due to simple syntax rule
- Interpreted language – Code execution & interpretation line by line
- Cross-platform language – It can run on windows,linux,macinetosh etc. equally
- Expressive language – Less code to be written as it itself express the purpose of the code.
- Completeness – Support wide range of library
- Free & Open Source – Can be downloaded freely and source code can be modify for improvement
- Variety of Usage / Applications

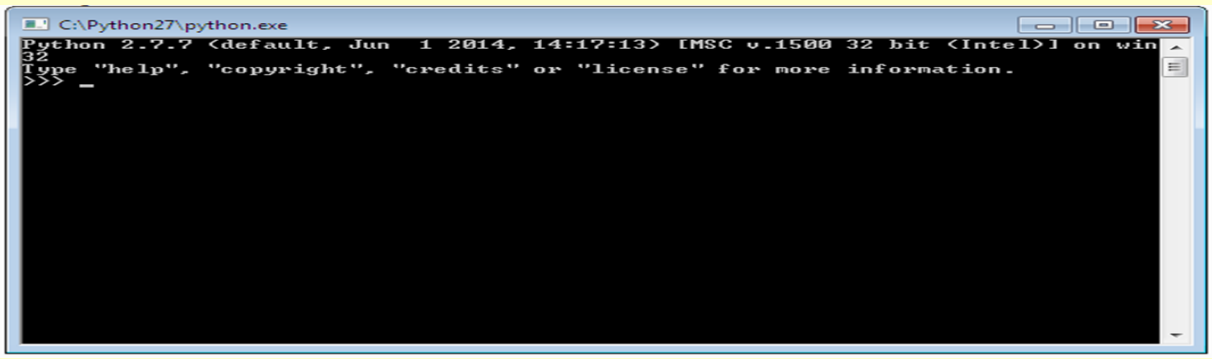
Limitation of Python:

- Not the fastest language (interpreted)
- Lesser Libraries than C, Java, Perl
- Not Strong on Type-binding
- Not easily convertible

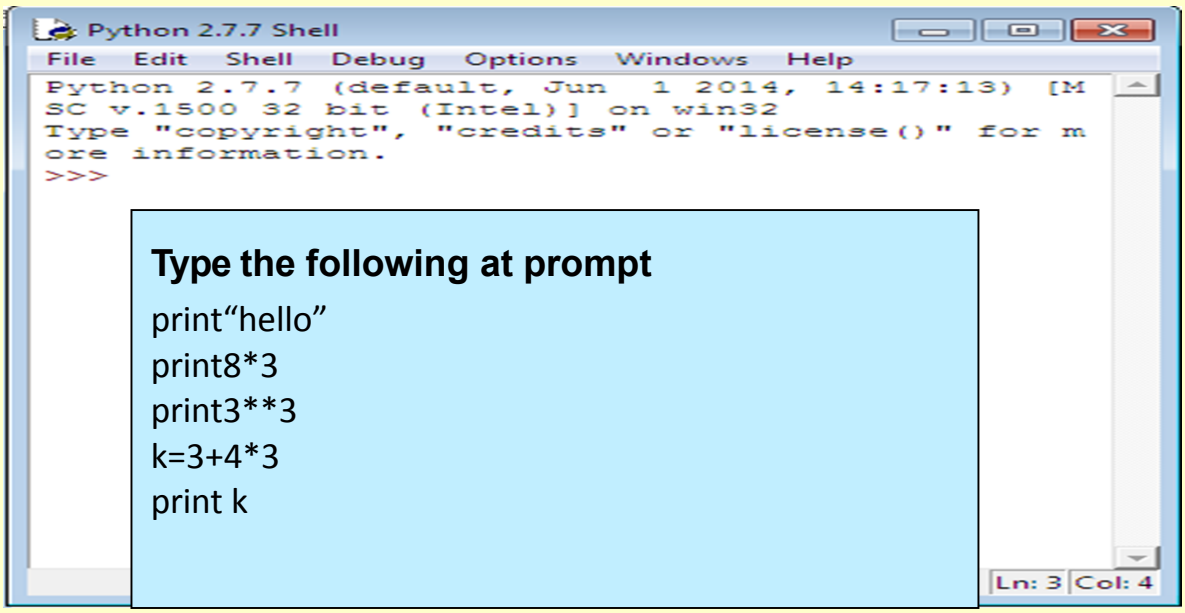
How to work in Python

(i) in Interactive mode

* Search the python.exe file in the drive in which it is installed.
If found double click it to start python in interactive mode



* Click start button -> All programs -> python<version>->IDLE(Python GUI)



(ii) in Script mode

Step 1 (Create program file)

Below steps are for simple hello world program
Click Start button->All Programs ->
Python<version>->IDLE
Now click File->New in IDLE Python Shell Now type
print “hello” print “world”
print “python is”,”object oriented programming lang.”
Click File->Save and then save the file with filename and .py extension

Step 2 (Run program file)

Click Open command from IDLE's File menu and select the file you have already saved
Click Run-> Run Module
It will execute all the commands of program file and display output in separate python shell window

Tokens

In a passage of text, individual words and punctuation marks are called tokens or lexical units or lexical elements. The smallest individual unit in a program is known as Tokens. Python has following tokens:

Keywords

Keywords are the reserved words and have special meaning for python interpreter. Every keyword is assigned specific work and it can be used only for that purpose.

and	del	from	not
while	as	elif	global
or	with	assert	else
if	pass	yield	break
except	import	print	class
exec	in	raise	continue
finally	is	return	def
for	lambda	try	

Identifiers

Are thenames given to different parts of program like variables, objects, classes, functions etc. Identifier forming rules of Python are :

- Is an arbitrarily long sequence of letters and digits
- The first character must be letter or underscore
- Upper and lower case are different
- The digits 0-9 are allowed except for first character
- It must not be a keyword
- No special characters are allowed other than underscore is allowed.

- Space not allowed

Valid Identifiers:
GradePay GRADEPAYFile_12_2018_ismarriedJAMES007_to_update

Invalid Identifiers:
Grade-Pay if12_2018_File RollNo.$JAMES007Roll No

Literal
Literals are data items that have a fixed value. Python supports several kinds of literals:

- String Literal
- Numeric Literals
- Boolean Literals
- Special Literals – ***None***

String Literal is a collection of character(s) enclosed in a double or single quotes. It can be either Single line strings or Multiline Strings

„123456“
„Hello How are your“
“Python”

"""1/7 PreetVihar
New Delhi
India"""

Numeric literals in Python can belong to any of the following numerical types:

Integer Literals: it contain at least one digit and must not contain decimal point. It may contain (+) or (-) sign.

Floating point Literals: Also known as real literals. Real literals are numbers having fractional parts.

Complex number Literals: Complex number in python is made up of two floating

Boolean literals in Python is used to represent one of the two Boolean values i.e. True or False

Python has one **Special literal**, which is None. It indicates absence of value. In other languages it is knows as NULL.

Operators

Type ofOperators	Symbols
Arithmetic	+, -, *, /, %, **,//
Relational	>, <, >=, <=, ==,!=
Logical	and,or
Identity	is, is not
Assignment	=
Membership	in, notin
Arithmetic Assignment	+=, -=, *=, /=, %=, **=,//=
Bitwise	&, ^, |, <<,>>

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: „ " # $ @ []{}=:;(),.

Data handling

Most of the computer programming language support data type, variables,operator and expression like fundamentals.Python also support these.

Data Types

Data Type specifies which type of value a variable can store. type() function is used to determine a variable's type in Python.

Data Types in Python

Number String Boolean List Tuple Set Dictionary

NumberIn Python:

It is used to store numeric values

Python has three numeric types:

- Integers
- Floating point numbers
- Complex numbers

(i)Integers

Integers or int are positive or negative numbers with no decimal point. Integers in Python 3 are of unlimited size.

e.g.

```
a= 100
b= -100
c= 1*20  print(a)  print(b)  print(c)
```

Output :-100
-100
200

Type Conversion of Integer

int() function converts any data type to integer. e.g.

```
a = "101" # string
b=int(a) # converts string data type to integer.
c=int(122.4) # converts float data type to integer.
print(b)
print(c)
```

Output :- 101
122

(ii)Floating point numbers

It is a positive or negative real numbers with a decimal point.e.g.

```
a = 101.2
b = -101.4
c = 111.23
d = 2.3*3
```

```
print(a)
print(b)
print(c)
print(d)
```

```
Output :-  101.2
           -101.4
           111.23
           6.8999999999999995
```

Type Conversion of Floating point numbers

float() function converts any data type to floating point number.e.g.

```
a='301.4' #string
b=float(a) #converts string data type to floating point number.
c=float(121) #converts integer data type to floating point number.
print(b)
print(c)
```

```
Output :-
301.4
121.0
```

(iii)Complex numbers

Complex numbers are combination of a real and imaginary part.Complex numbers are in the form ofX+Yj, where X is a real part and Y is imaginary part. e.g.

```
a = complex(5) # convert 5 to a real part val and zero imaginary part
print(a)
b=complex(101,23) #convert 101 with real part and 23 as imaginary part
print(b)
```

```
Output :-
(5+0j)
(101+23j)
```

String In Python

A string is a sequence of characters. In python we can create string using

single (' ') or double quotes (" ").Both are same in python. e.g.

```
str='computer science'
print('str-',str) # print string
print('str[0]-',str[0]) # print first char 'h'
print('str[1:3]-',str[1:3]) # print string from postion 1 to 3 'ell'
print('str[3:]-', str[3:]) # print string staring from 3rd char 'llo world'
print('str *2-',  str *2 ) # print string two times
print("str +'yes'-", str +'yes') # concatenated string
```

Output

```
str- computer science
str[0]- c
str[1:3]- om
str[3:]- puter science
```

Iterating through string

e.g.

```
str='comp sc'
for i in str:
        print(i)
```

Output

```
c
o
m
p

s
c
```

Boolean In Python

It is used to store two possible values either true or false

e.g.

```
str="comp sc"
boo=str.isupper() # test if string contains upper case
print(boo)
```

Output

```
False
```

List In Python

List are collections of items and each item has its own index value.
e.g. of list

```
list =[6,9]
list[0]=55
print(list[0])
print(list[1])
```

OUTPUT

```
55
9
```

Tuple In Python

List and tuple, both are same except,a list is mutable python objects and tuple is immutable Python objects. Immutable Python objects mean you cannot modify the contents of a tuple once it is assigned.

Set In Python

It is an unordered collection of unique and immutable (which cannot be modified)items.
e.g.

```
set1={11,22,33,22}
print(set1)
```

Output

```
{33, 11, 22}
```

Dictionary In Python

It is an unordered collection of items and each item consist of a key and a value.
e.g.

```
dict = {'Subject': 'comp sc', 'class': '11'}
print(dict)
print ("Subject : ", dict['Subject'])
print ("class : ", dict.get('class'))
```

Output

{'Subject': 'comp sc', 'class': '11'}
Subject : comp sc class : 11

Arithmetic Operator:

Operator	Meaning	Example
+	Add two operands or unaryplus	x +y +2
-	Subtract right operand from the left or unaryminus	x -y -2
*	Multiply twooperands	x *y
/	Divide left operand by the right one (always results intofloat)	x /y
%	Modulus - remainder of the division of left operand by theright	x % y (remainder ofx/y)
//	Floor division - division that results into whole number adjusted to the left in the numberline	x //y
**	Exponent - left operand raised to the power ofright	x**y (x to the power y)

Arithmetic operator e.g.

```
x = 5
y = 4
print('x + y =',x+y)
print('x - y =',x-y)
print('x * y =',x*y)
print('x / y =',x/y)
print('x // y =',x//y)
print('x ** y =',x**y)
```

OUTPUT

```
('x + y =', 9)
('x - y =', 1)
```

('x * y =', 20)
('x / y =', 1)
('x // y =', 1)
('x ** y =', 625)

Comparison operators:

Operator	Meaning	Example
>	Greater that - True if left operand is greater than theright	x >y
<	Less that - True if left operand is less than theright	x <y
==	Equal to - True if both operands areequal	x ==y
!=	Not equal to - True if operands are notequal	x !=y
>=	Greater than or equal to - True if left operand is greater than or equal to the right	x >=y
<=	Less than or equal to - True if left operand is less than or equal to theright	x <=y

Comparison operators e.g.

```
x = 101
y = 121
print('x > y is',x>y)
print('x < y is',x<y)
print('x == y is',x==y)
print('x != y is',x!=y)
print('x >= y is',x>=y)
print('x <= y is',x<=y)
```

Output

```
('x > y is', False)
('x < y is', True)
('x == y is', False)
('x != y is', True)
('x >= y is', False)
('x <= y is', True)
```

Logical operators:

Operator	Meaning	Example
and	True if both the operands aretrue	x andy
or	True if either of the operands istrue	x ory
not	True if operand is false (complements theoperand)	notx

```
x = True
y = False
print('x and y is',x and y)
print('x or y is',x or y)
print('not x is',not x)
```

Output

```
('x and y is', False)
('x or y is', True)
('not x is', False)
```

Bitwise operators

Operator	Meaning	Example
& Binary AND	Operator copies a bit to the result if it exists in both operands	(a & b) (means 0000 1100)
| Binary OR	It copies a bit if it exists in either operand.	(a | b) = 61 (means 0011 1101)
^ Binary XOR	It copies the bit if it is set in one operand but not both.	(a ^ b) = 49 (means 0011 0001)
~ Binary Ones Complement	It is unary and has the effect of 'flipping' bits.	(~a) = -61 (means 1100 0011 in 2's complement form due to a signed binary number.
<< Binary Left Shift	The left operands value is moved left by the number of bits specified by the right operand.	a << 2 = 240 (means 1111 0000)
>> Binary Right Shift	The left operands value is moved right by the number of bits specified by the right operand.	a >> 2 = 15 (means 0000 1111)

```
a = 60            # 60 = 0011 1100
b = 13            # 13 = 0000 1101
c = 0

c = a & b;        # 12 = 0000 1100
print "Line 1 - Value of c is ", c

c         # 61 = 0011 1101
print "Line 2 - Value of c is ", c

c = a ^ b;        # 49 =  0011  0001
print "Line 3 - Value of c is ", c

c = ~a;           # -61 = 1100 0011
print "Line 4 - Value of c is ", c

                              # 000011 11

c = a << 2;       # 240 = 1111 0000
print "Line 5 - Value of c is ", c

c = a >> 2;       # 15 = 0000 1111
print "Line 6 - Value of c is ", c
```

Output:

```
Line 1 - Value of c is 12
Line 2 - Value of c is 61
Line 3 - Value of c is 49
Line 4 - Value of c is -61
Line 5 - Value of c is 240
Line 6 - Value of c is 15
```

Membership Operators

Test for membership in a sequence

Operator	Description
in	Evaluates to true if it finds a variable in the specified sequence and falseotherwise.

notin	Evaluates to true if it does not finds a variable in the specified sequence and falseotherwise.

```
a=5
b = 10
list = [1, 2, 3, 4, 5 ]
if ( a in list ):
        print ("Line 1 - a is available in the given list")
else:
        print ("Line 1 - a is not available in the given list")
if ( b not in list ):
        print ("Line 2 - b is not available in the given list")
else:
        print ("Line 2 - b is available in the given list")
```

output

```
Line 1 - a is available in the given list
Line 2 - b is not available in the given list
```

Identity Operators

Operator	Description
is	Evaluates to true if the variables on either side of the operator point to the same object and false otherwise.
isnot	Evaluates tofalse if the variables on either side of the operator point to the same object and trueotherwise.

```
x = ["apple", "banana"]
y = ["apple", "banana"]
z = x

print(x is z)   # returns True because z is the same object as x

print(x is y)  # returns False because x is not the same object as y, even if
                    they have the same content
print(x == y)  # to demonstrate the difference betweeen "is" and "==": this
                     comparison returns True because x is equal to y
```

Output:

```
True
False
True
```

```
x = ["apple", "banana"]
y = ["apple", "banana"]
z = x
print(x is not z)   # returns False because z is the same object as x
print(x is not y)   # returns True because x is not the same object as y, even
                      if they have the same content

print(x != y)       # to demonstrate the difference betweeen "is not" and "!=":
                      this comparison returns False because x is equal to y
```

Output:

```
False
True
False
```

Operators Precedence: highest precedence to lowest precedence table

Operator	Description
**	Exponentiation (raise to thepower)
~ +-	Complement, unary plus and minus (method names for the last two are +@ and-@)
* / % //	Multiply, divide, modulo and floordivision
+-	Addition andsubtraction
>><<	Right and left bitwiseshift
&	Bitwise'AND'td>
^|	Bitwise exclusive `OR' and regular`OR'
<= <>>=	Comparisonoperators
<> ==!=	Equalityoperators
= %= /= //= -=+= *=**=	Assignment operators
is isnot	Identityoperators
in not in	Membershipoperators

not orand	Logical operators

Expression:

It is a valid combination of operators,literals and variable.

Arithmaticexpression :- e.g. c=a+b
Relational expression :- e.g. x>y
Logical expression :- a or b
String expression :- c="comp"+"sc"

Type Conversion:

The process of converting the value of one data type (integer, string, float, etc.) to anotherdata type is called type conversion.Python has two types of type conversion.

- Implicit Type Conversion
- Explicit Type Conversion

Implicit Type Conversion:

In Implicit type conversion, Python automatically converts one data type to another data type. This process doesn't need any user involvement.
e.g.

```
num_int = 12
num_flo = 10.23
num_new = num_int + num_flo
print("datatype of num_int:",type(num_int))
print("datatype of num_flo:",type(num_flo))
print("Value of num_new:",num_new)
print("datatype of num_new:",type(num_new))
```

OUTPUT

```
('datatype of num_int:',<type 'int'>)
('datatype of num_flo:',<type 'float'>)
('Value of num_new:', 22.23)
('datatype of num_new:',<type 'float'>)
```

Explicit Type Conversion:

In Explicit Type Conversion, users convert the data type of an object to requireddata type. We use the predefined functions like int(),float(),str() etc. e.g.

```
num_int = 12
num_str = "45"
print("Data type of num_int:",type(num_int))
print("Data type of num_str before Type Casting:",type(num_str))
num_str = int(num_str)
print("Data type of num_str after Type Casting:",type(num_str))
num_sum = num_int + num_str
print("Sum of num_int and num_str:",num_sum)
print("Data type of the sum:",type(num_sum))
```

OUTPUT

```
('Data type of num_int:',<type 'int'>)
('Data type of num_str before Type Casting:',<type 'str'>)
('Data type of num_str after Type Casting:',<type 'int'>)
('Sum of num_int and num_str:', 57)
('Data type of the sum:',<type 'int'>)
```

Control Flow Statements

Control statements are used to control the flow of execution depending upon the specified condition/logic. There are three types of control statements.

1. Decision Making Statements
2. Iteration Statements (Loops)
3. Jump Statements (break, continue, pass)

Decision making

Decision making statement used to control the flow of execution of program depending upon condition. (if)

if statement in Python is of many forms:

- if without false statement
- if with else
- if with elif
- Nested if

(a) In the simplest form if statement in Python checks the condition and execute the statement if the condition is true and do nothing if the condition is false.

Syntax:

```
if  condition:
    Statement1
    Statements ….
```

Points to be notes:

- It must contain valid condition which evaluates to either True or False
- Condition must followed by Colon (:) , it is mandatory
- Statement inside if must be at same indentation level.

Example: Input monthly sale of employee and give bonus of 10% if sale is more than 50000 otherwise bonus will be 0

```
bonus = 0
```

```
sale = int(input("Enter Monthly Sales :"))
if sale>50000:
        bonus=sale * 10 /100
print("Bonus = " + str(bonus))
```

Example:

```
x=1
y=2
if(x==1 and y==2):
        print("condition matching the criteria")
```

Example:

```
a=100
if not(a == 20):
        print('a is not equal to 20')
```

(b) if with else is used to test the condition and if the condition is True it perform certain action and alternate course of action if the condition is false.

Syntax:

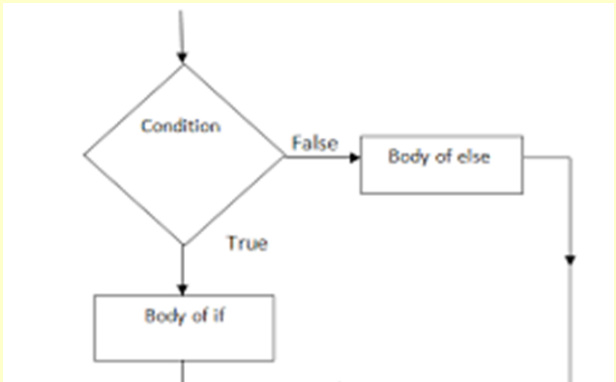
```
        if condition:
                Statements
        else:
                Statements
```

Example: Input Age of person and print whether the person is eligible for voting or not

```
age = int(input("Enter your age "))
if age>=18:
        print("Congratulation! you are eligible for voting ")
else:
        print("Sorry! You are not eligible for voting")
```

Example:

```
a=10
if(a < 100):
        print('less than 100')
```



```
else:
    print('more than equal 100')
```

(c) if with elif is used where multiple chain of condition is to be checked. Each elif must be followed by condition: and then statement for it. After every elif we can give else which will be executed if all the condition evaluates to false

Syntax:

```
if condition:
    Statements
elif condition:
    Statements
elif condition:
    Statements
else:
    Statement
```

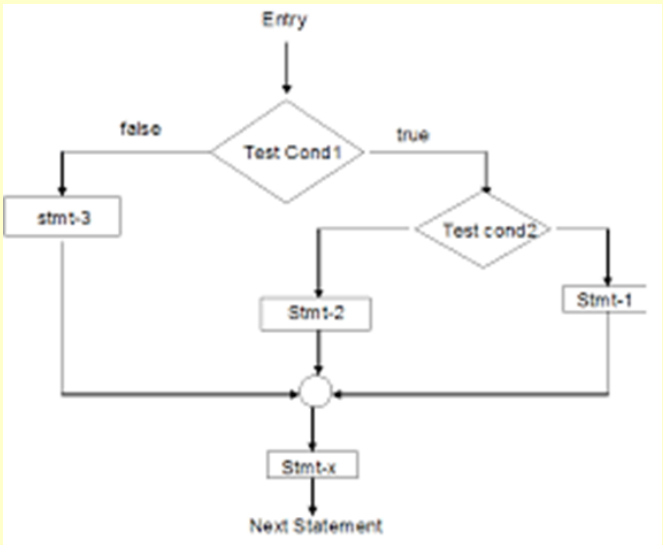
Example: Input temperature of water and print its physical state

```
temp = int(input("Enter temperature of water "))
if temp>100:
    print("Gaseous State")
elif temp<0:
    print("Solid State")
else:
    print("Liquid State")
```

(d) In nested type of "if" we put if within another if as a statement of it. Mostly used in a situation where we want different else for each condition.

Syntax:

```
if condition1:
    if condition2:
        statements
    else:
        statements
elif condition3:
    statements
else:
    statements
```



Example:

```
num = float(input("Enter a number: "))
if num>= 0:
        if num == 0:
                print("Zero")
        else:
                print("Positive number")
else:
        print("Negative number")
```

OUTPUT

```
Enter a number: 5
Positive number
```

LOOPING

To carry out repetition of statements Python provide 2 loop statements

Conditional loop (while)
Counting loop (for)

While Loop

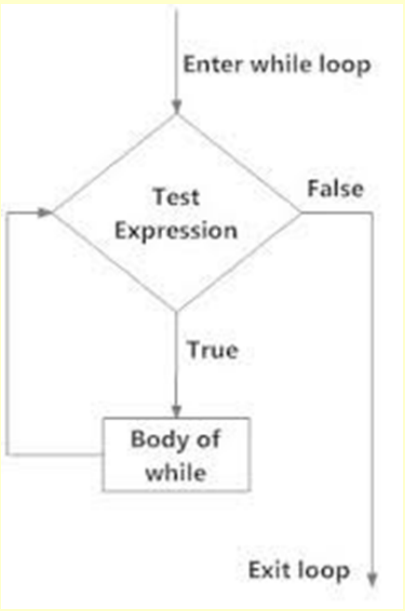
It is used to execute a block of statement as long as a given condition is true and when the condition become false, the control will come out of the loop. The condition is checked every time at the beginning of the loop.

Syntax

```
while (condition):
        statement [statements]
```

e.g.

```
x = 1
while (x <= 4):
        print(x)
        x = x + 1
```



Initialization: it is used to give starting value in a loop variable from where to start the loop
Test condition: it is the condition or last value up to which loop will be executed.
Body of loop: it specifies the action/statement to repeat in the loop
Update statement: it is the increase or decrease in loop variable to reach the test condition.

Example:

```
x = 1
while (x < 3):
        print('inside while loop value of x is ',x)
        x = x + 1
else:
        print('inside else value of x is ', x)
```

Output

```
inside while loop value of x is 1
inside while loop value of x is 2
inside else value of x is 5
```

Example:
e.g.

```
x = 5
while (x == 5):
        print('inside loop')
```

Output

```
Inside loop
Inside loop
…
…
```

FOR loop

for loop in python is used to create a loop to process items of any sequence like List, Tuple, Dictionary, String

It can also be used to create loop of fixed number of steps like 5 times, 10 times, n times etc using range() function.

Syntax

```
for val in sequence:
statements
```

```
School=["Principal","PGT","TGT","PRT"]
for sm in School:
        print(sm)
```

```
Code=(10,20,30,40,50,60)
for cd in Code:
        print(cd)
```

```
for ch in 'Plan':
print(ch)
```

Let us create a loop to print all the natural number from 1 to 100

```
for i in range(1,101):
        print(i, end='\t')
```

usage of Range()

Range (lower_limit, upper_limit)

The range function generate set of values from lower_limit to upper_limit-1

For e.g.

range(1,10) will generate set of values from 1-9

range(0,7) will generate [0-6]

Default step value will be +1 i.e.

range(1,10) means (1,2,3,4,5,6,7,8,9)

To change the step value we can use third parameter in range() which is step value

range(1,10,2) will produce **[1,3,5,7,9]**

Step value can be in –ve also to generate set of numbers in reverse order

range(10,0) will produce **[10,9,8,7,6,5,4,3,2,1]**

To create list from ZERO(0) we can use
range(10) it will produce **[0,1,2,3,4,5,6,7,8,9]**

The operator in and not in is used in for loop to check whether the value is in the range / list or not
e.g.

```
>>> 5 in [1,2,3,4,5]
True
>>> 5 in [1,2,3,4]
False
>>>'a' in 'apple'
True
>>>'national' in 'international'
True
```

```
for i in range(5,3,-1):
        print(i)

for i in range(3,5):
        print(i)

for i in range(1,3):
        for j in range(1,11):
                k=i*j
                print (k, end=' ')
        print()
```

Jump Statements

Jump statements are used to transfer the program's control from one location to another. Means these are used to alter the flow of a loop like - to skip a part of a loop or terminate a loop

There are three types of jump statements used in python.
1. break
2. continue
3. pass

1. break
it is used to terminate the loop.
e.g.

```
for val in "string":
        if val == "i":
                break
        print(val)

print("The end")
```

Output
s
t
r
The end

```
for i in range(1,20):
        if i % 6 == 0:
                break
        print(i, end='')
print("Loop Over")
```

The above code produces output
1 2 3 4 5
Loop Over

2. continue

It is used to skip all the remaining statements in the loop and move controls back to the top of the loop.

```
for val in "init":
        if val == "i":
                continue
        print(val)
print("The end")
```

Output

n

t

The end

```
for i in range(1,20):
        if i % 6 == 0:
                continue
        print(i, end = ' ')
print("Loop Over")
```

The above code produces output

1 2 3 4 5 7 8 9 10 11 13 14 15 16 17 19

Loop Over

when the value of i becomes divisible from 6, condition will becomes True and loop will skip all the *statement below continue* and continues in loop with next value of iteration

3. pass

This statement does nothing. It can be used when a statement is required syntactically but the program requires no action.

```
for i in 'initial':
        if(i == 'i'):
                pass
        else:
                print(i)
```

OUTPUT

```
n
t
a
L
```

NOTE: continue forces the loop to start at the next iteration while pass means "there is no code to execute here" and will continue through the remainder or the loop body.

String manipulation

Two basic operators + and * are allowed
+ is used for concatenation (joining)
e.g. "shakti" + "man"
OUTPUT: shaktiman
* Is used for replication (repetition)
e.g. "Bla" * 3
OUTPUT: BlaBlaBla

Note: you cannot multiply string and string using * only number*number or string*number is allowed

Membership operators (in and not in) are used to check the presence of character(s) in any string.

Example	Output
'a' in 'python'	False
'a' in 'java'	True
'per' in 'operators'	True
'men' in 'membership'	False
'Man' in 'manipulation'	False
'Pre' not in 'presence'	True

We can apply comparison operators (==,!=,>,<,>=,<=) on string. Comparison will be character by character

str1='program'
str2='python'
str3='Python'

Example	Output
str1==str2	False
str1!=str2	True
str2=='python'	True
str2>str3	True

str3<str1	True
Characters	Ordinal/ ASCII code
A-Z	65-90
a-z	97-122
0-9	48-57

Python allows us to find out the ordinal position single character using ord() function.

>>>ord('A') output will be 65

We can also find out the character based on the ordinal value using chr() function

>>>chr(66) output will be 'B'

String slicing

```
>>> str1="wonderful"
>>> str1[0:6]
'wonder'
>>> str1[0:3]
'won'
>>> str1[3:6]
'der'
>>> str1[-1:-3]
''
>>> str1[-3:-1]
'fu'
>>> str1[-3:0]
''
>>> str1[3:3]
''
>>> str1[3:4]
'd'
>>> str1[-5:-2]
'erf'
>>> str1[:-2]
```

'wonderf'
>>> str1[:4]
'wond'
>>> str1[-3:]
'ful'
>>>str1[::-1]
lufrednow

String Function

Function name	Purpose	Example
String.capitalize()	Return a copy of string with first character as capital	>>>'computer'.capitalize() Computer
String.find(str[,start[,end]])	Return lowest index of str in given string , -1 if not found	Str="johny johny yes papa" Sub="johny" Str.find(Sub) 0 Str.find(Sub,1) 6 Str.find('y',6,11) 10 Str.find('pinky') -1
String.isalnum()	Return True if the string is alphanumeric character	'hello123'.isalnum() True
String.isalnum()		S="ravi@gmail.com S.Isalnum() False
String.isalpha()	Return True if string contains only alphabets characters	Str1="hello" Str2="hello123" Str1.isalpha() True Str2.isalpha() False
String.isdigit()	Return True if string contains only digits	s1="123" s1.isdigit() True Str2.isdigit() False
String.islower()	Return True if all character in	Str1.islower() True

	string is in lower case	
String.upper()	Return True if all characters in the string are in upper case	S="HELLO" S.isupper() True
String.lower()	Return copy of string converted to lower case characters	S="INDIA" S.lower() india
String.upper()	Return copy of string converted to upper case characters	S="india" S.lower() INDIA