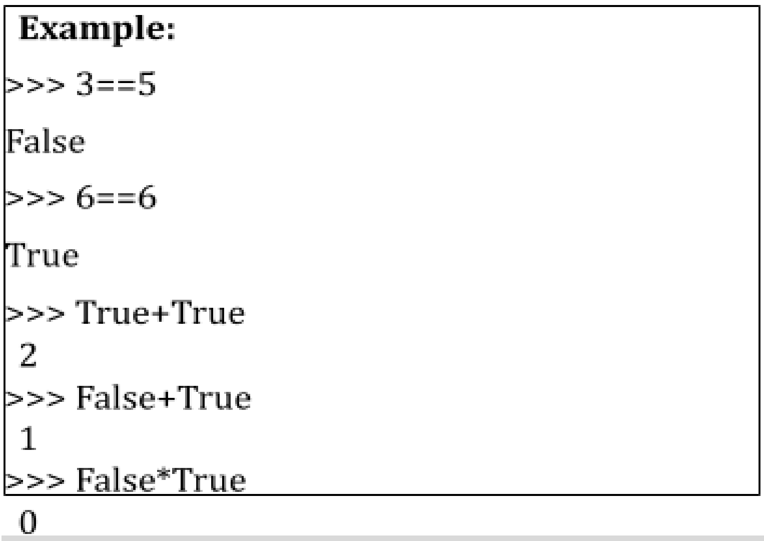
CONTROL FLOW, FUNCTIONS

BOOLEAN VALUES:

Boolean:

- ❖ Boolean data type have two values. They are 0 and 1.
- ❖ 0 represents False
- ❖ 1 represents True
- ❖ True and False are keyword.



Example:

```
>>> 3==5
False
>>> 6==6
True
>>> True+True
 2
>>> False+True
 1
>>> False*True
 0
```

OPERATORS:

- ❖ Operators are the constructs which can manipulate the value of operands.
- ❖ Consider the expression *4 + 5 = 9.* Here, 4 and 5 are called operands and + is called operator.

Types of Operators:

1. Arithmetic Operators
2. Comparison (Relational) Operators
3. Assignment Operators
4. Logical Operators
5. Bitwise Operators
6. Membership Operators
7. Identity Operators

Arithmetic operators:

They are used to perform mathematical operations like addition, subtraction, multiplication etc.

Operator	Description	Example a=10,b=20
+ Addition	Adds values on either side of the operator.	a + b = 30
- Subtraction	Subtracts right hand operand from left hand operand.	a – b = -10
* Multiplication	Multiplies values on either side of the operator	a * b = 200
/ Division	Divides left hand operand by right hand operand	b / a = 2
% Modulus	Divides left hand operand by right hand operand and returns remainder	b % a = 0
** Exponent	Performs exponential (power) calculation on operators	a**b =10 to the power 20
//	Floor Division - The division of operands where the result is the quotient in which the digits after the decimal point are removed	5//2=2

Comparison (Relational) Operators:

❖ Comparison operators are used to compare values.

❖ It either returns True or False according to the condition.

Operator	Description	Example a=10,b=20
==	If the values of two operands are equal, then the condition becomes true.	(a == b) is not true.
!=	If values of two operands are not equal, then condition becomes true.	(a!=b) is true
>	If the value of left operand is greater than the value of right operand, then condition becomes true.	(a > b) is not true.
<	If the value of left operand is less than the value of right operand, then condition becomes true.	(a < b) is true.
>=	If the value of left operand is greater than or equal to the value of right operand, then condition becomes true.	(a >= b) is not true.
<=	If the value of left operand is less than or equal to the value of right operand, then condition becomes true.	(a <= b) is true.

Assignment Operators:

Assignment operators are used in Python to assign values to variables.

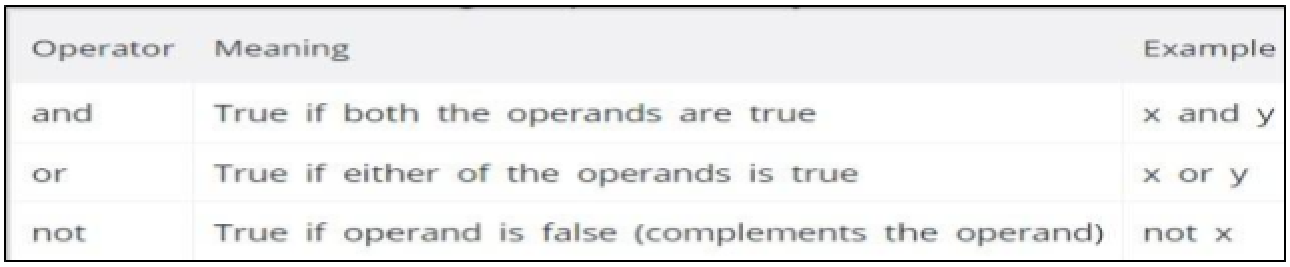
Operator	Description	Example

=	Assigns values from right side operands to left side operand	c = a + b assigns value of a + b into c
+= Add AND	It adds right operand to the left operand and assign the result to left operand	c += a is equivalent to c = c + a
-= Subtract AND	It subtracts right operand from the left operand and assign the result to left operand	c -= a is equivalent to c = c - a

*= Multiply AND	It multiplies right operand with the left operand and assign the result to left operand	c *= a is equivalent to c = c * a
/= Divide AND	It divides left operand with the right operand and assign the result to left operand	c /= a is equivalent to c = c / ac /= a is equivalent to c = c / a
%= Modulus AND	It takes modulus using two operands and assign the result to left operand	c %= a is equivalent to c = c % a
**= Exponent AND	Performs exponential (power) calculation on operators and assign value to the left operand	c **= a is equivalent to c = c ** a
//= Floor Division	It performs floor division on operators and assign value to the left operand	c //= a is equivalent to c = c // a

Logical Operators:

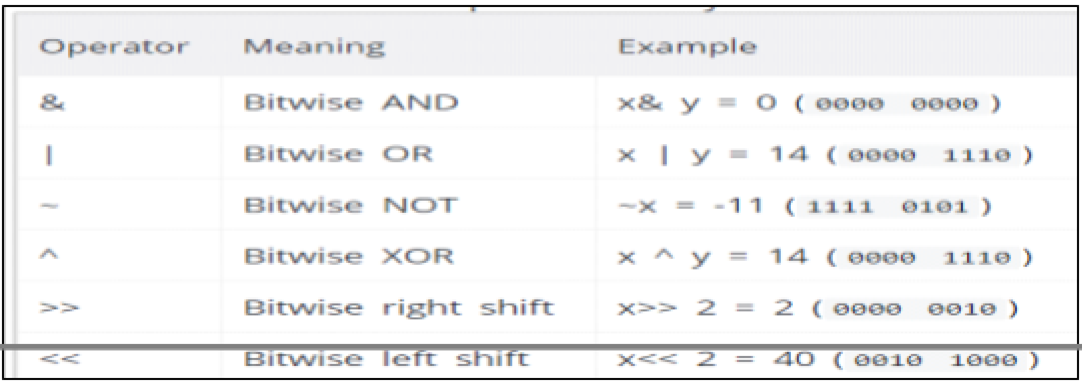
Logical operators are and, or, not operators.



Operator	Meaning	Example
and	True if both the operands are true	x and y
or	True if either of the operands is true	x or y
not	True if operand is false (complements the operand)	not x

Bitwise Operators:

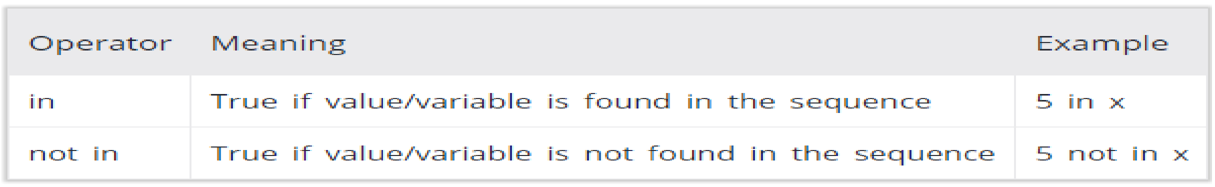
Let x = 10 (0000 1010 in binary) and y = 4 (0000 0100 in binary)



Operator	Meaning	Example
&	Bitwise AND	x& y = 0 (0000 0000)
|	Bitwise OR	x | y = 14 (0000 1110)
~	Bitwise NOT	~x = -11 (1111 0101)
^	Bitwise XOR	x ^ y = 14 (0000 1110)
>>	Bitwise right shift	x>> 2 = 2 (0000 0010)
<<	Bitwise left shift	x<< 2 = 40 (0010 1000)

Membership Operators:

- ❖ Evaluates to find a value or a variable is in the specified sequence of string, list, tuple, dictionary or not.
- ❖ To check particular element is available in the list or not.
- ❖ Operators are in and not in.



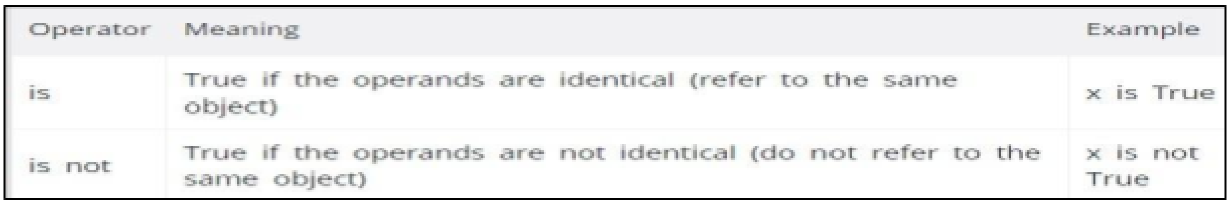
Operator	Meaning	Example
in	True if value/variable is found in the sequence	5 in x
not in	True if value/variable is not found in the sequence	5 not in x

Example:

```
x=[5,3,6,4,1]
>>> 5 in x
True
>>> 5 not in x
False
```

Identity Operators:

They are used to check if two values (or variables) are located on the same part of the memory.



Operator	Meaning	Example
is	True if the operands are identical (refer to the same object)	x is True
is not	True if the operands are not identical (do not refer to the same object)	x is not True

Example

```
x = 5
y = 5
a = 'Hello'
b = 'Hello'
print(x is not y) // False
print(a is b)//True
```

CONDITIONALS

- ❖ Conditional if
- ❖ Alternative if… else
- ❖ Chained if…elif…else
- ❖ Nested if….else

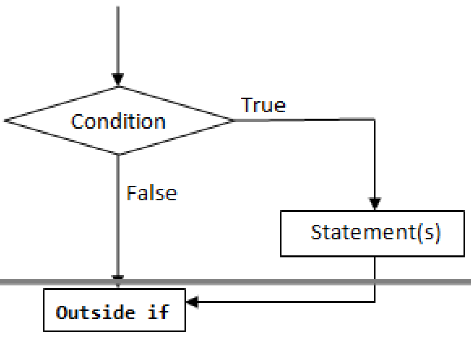
Conditional (if):

conditional (if) is used to test a condition, if the condition is true the statements inside if will be executed.

syntax:

```
if(condition 1):
        Statement 1
```

Flowchart:



Example:

1. Program to provide flat rs 500, if the purchase amount is greater than 2000.
2. Program to provide bonus mark if the category is sports.

Program to provide flat rs 500, if the purchase amount is greater than 2000.	**output**

purchase=eval(input("enter your purchase amount"))
if(purchase>=2000):
    purchase=purchase-500
print("amount to pay",purchase)
	enter your purchase amount
	2500
	amount to pay
	2000

Program to provide bonus mark if the category is sports	**output**

m=eval(input("enter ur mark out of 100"))
c=input("enter ur categery G/S")
if(c=="S"):
    m=m+5
print("mark is",m)
	enter ur mark out of 100
	85
	enter ur categery G/S
	S
	mark is 90

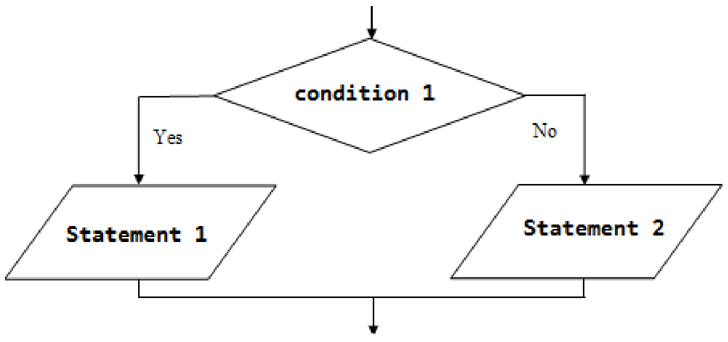
alternative (if-else)

In the alternative the condition must be true or false. In this **else** statement can be combined with **if** statement. The **else** statement contains the block of code that executes when the condition is false. If the condition is true statements inside the if get executed otherwise else part gets executed. The alternatives are called branches, because they are branches in the flow of execution.

syntax:

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

Flowchart:



Examples:

1. odd or even number
2. positive or negative number

3. leap year or not

4. greatest of two numbers
5. **5.** eligibility for voting

Odd or even number	**Output**

n=eval(input("enter a number"))
if(n%2==0):
   print("even number")
else:
   print("odd number")
	enter a number4
	even number

positive or negative number	**Output**

n=eval(input("enter a number"))
if(n>=0):
   print("positive number")
else:
   print("negative number")
	enter a number8
	positive number

leap year or not	**Output**

y=eval(input("enter a yaer"))
if(y%4==0):
   print("leap year")
else:
   print("not leap year")
	enter a yaer2000
	leap year

greatest of two numbers	**Output**

a=eval(input("enter a value:"))
b=eval(input("enter b value:"))
if(a>b):
   print("greatest:",a)
else:
   print("greatest:",b)
	enter a value:4
	enter b value:7
	greatest: 7

eligibility for voting	**Output**

age=eval(input("enter ur age:"))
if(age>=18):
   print("you are eligible for vote")
else:
   print("you are eligible for vote")
	enter ur age:78
	you are eligible for vote

Chained conditionals(if-elif-else)

- The elif is short for else if.
- This is used to check more than one condition.
- If the condition1 is False, it checks the condition2 of the elif block. If all the conditions are False, then the else part is executed.
- Among the several if...elif...else part, only one part is executed according to

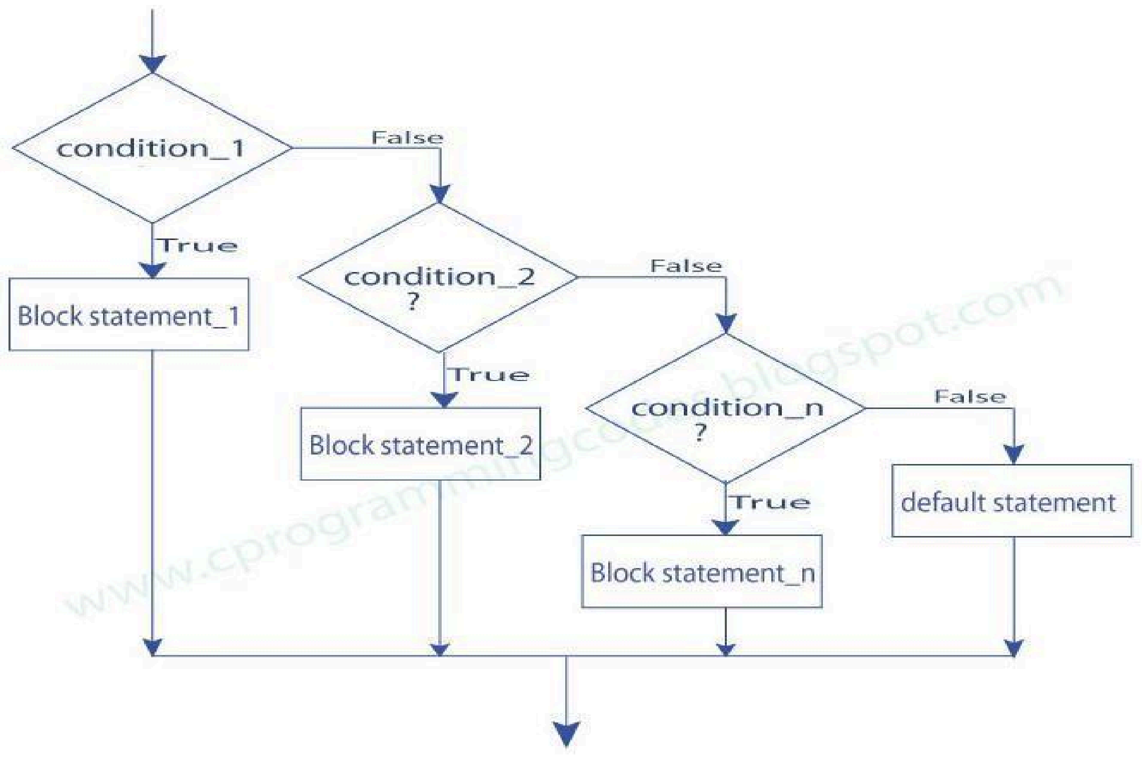
the condition.

- The if block can have only one else block.But it can have multiple elif blocks.
- The way to express a computation like that is a chained conditional.

syntax:

```
if(condition 1):
     statement 1
elif(condition 2):
     statement 2
elif(condition 3):
     statement 3
else:
    default statement
```

Flowchart:



Example:

1. student mark system
2. traffic light system
3. compare two numbers
4. roots of quadratic equation

student mark system	**Output**

mark=eval(input("enter ur mark:"))
if(mark>=90):
   print("grade:S")
elif(mark>=80):
   print("grade:A")
elif(mark>=70):
   print("grade:B")
elif(mark>=50):
   print("grade:C")
else:
   print("fail")
	enter ur mark:78
	grade:B

traffic light system	**Output**

colour=input("enter colour of light:")
if(colour=="green"):
   print("GO")
elif(colour=="yellow"):
   print("GET READY")
else:
   print("STOP")
	enter colour of light:green
	GO

compare two numbers	**Output**

x=eval(input("enter x value:"))
y=eval(input("enter y value:"))
if(x == y):
   print("x and y are equal")
elif(x < y):
   print("x is less than y")
else:
   print("x is greater than y")
	enter x value:5
	enter y value:7
	x is less than y

Roots of quadratic equation	**output**

a=eval(input("enter a value:"))
b=eval(input("enter b value:"))
c=eval(input("enter c value:"))
d=(b*b-4*a*c)
if(d==0):
   print("same and real roots")
elif(d>0):
   print("diffrent real roots")
else:
   print("imaginagry roots")
	enter a value:1
	enter b value:0
	enter c value:0
	same and real roots

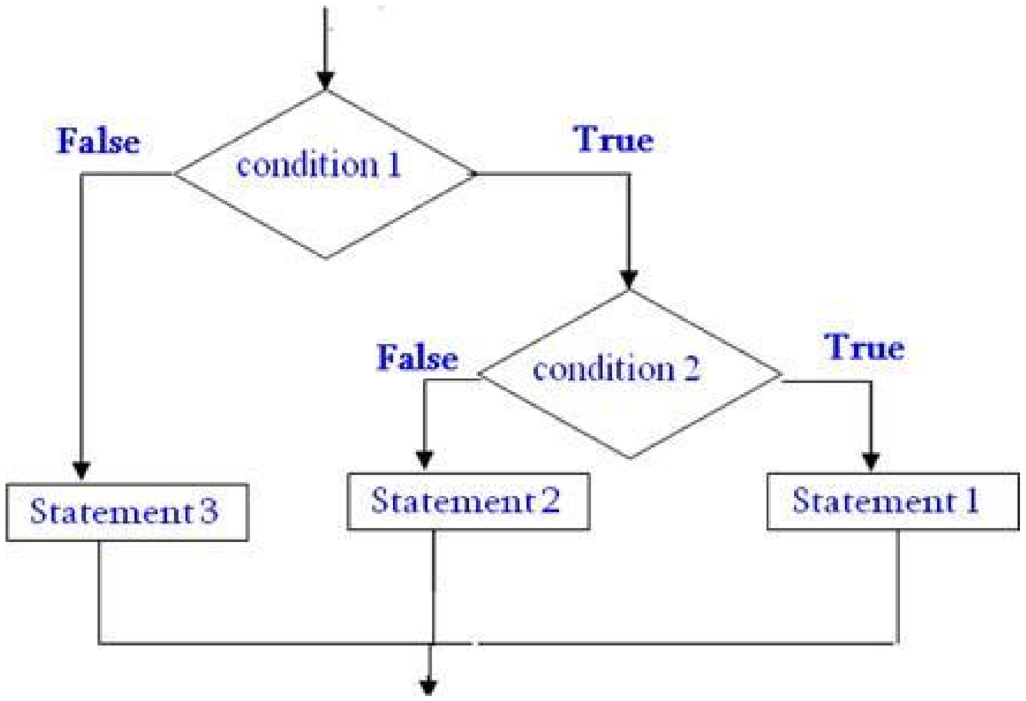
Nested conditionals

One conditional can also be nested within another. Any number of condition can be nested inside one another. In this, if the condition is true it checks another if condition1. If both the conditions are true statement1 get executed otherwise statement2 get execute. if the condition is false statement3 gets executed

Syntax:

```
if (condition):
      if(condition 1):
          statement 1
      else:
          statement 2
else:
        statement 3
```

Flowchart:



Example:

1. greatest of three numbers
2. **2.** positive negative or zero

greatest of three numbers	output

a=eval(input("enter the value of a"))
b=eval(input("enter the value of b"))
c=eval(input("enter the value of c"))
if(a>b):
    if(a>c):
        print("the greatest no is",a)
    else:
        print("the greatest no is",c)
	enter the value of a 9
	enter the value of a 1
	enter the value of a 8
	the greatest no is 9

```
else:
        if(b>c):
                print(“the greatest no is”,b)
        else:
                print(“the greatest no is”,c)
```
positive negative or zero	**output**

```
n=eval(input("enter the value of n:"))
if(n==0):
   print("the number is zero")
else:
   if(n>0):
      print("the number is positive")
   else:
      print("the number is negative")
```
	```
	enter the value of n:-9
	the number is negative
	```

ITERATION/CONTROL STATEMENTS:

- ❖ state
- ❖ while
- ❖ for
- ❖ break
- ❖ continue
- ❖ pass

State:

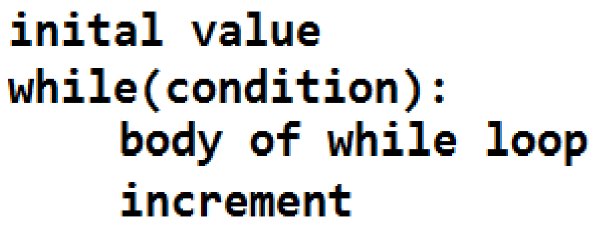
Transition from one process to another process under specified condition with in a time is called state.

While loop:

- While loop statement in Python is used to repeatedly executes set of statement as long as a given condition is true.
- In while loop, test expression is checked first. The body of the loop is entered only if the test_expression is True. After one iteration, the test expression is checked again. This process continues until the test_expression evaluates to False.

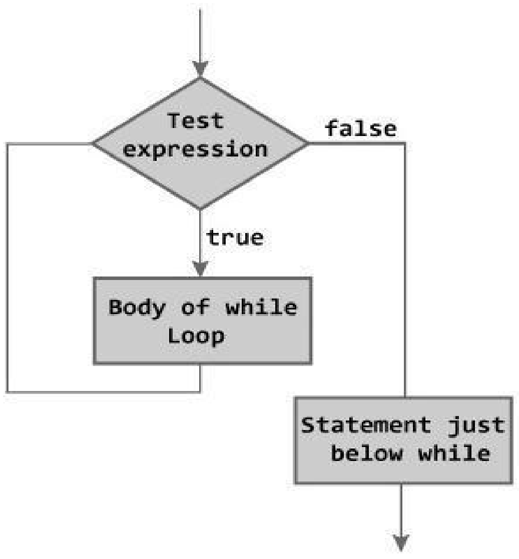
- In Python, the body of the while loop is determined through indentation.
- The statements inside the while starts with indentation and the first unindented line marks the end.

Syntax:



```
inital value
while(condition):
    body of while loop
    increment
```

Flowchart:



Examples:

1. program to find sum of n numbers:
2. program to find factorial of a number
3. program to find sum of digits of a number:
4. Program to Reverse the given number:
5. Program to find number is Armstrong number or not
6. Program to check the number is palindrome or not

Sum of n numbers:	**output**

n=eval(input("enter n"))
i=1
sum=0
while(i<=n):
   sum=sum+
   i i=i+1
print(sum)
	enter n
	10
	55

Factorial of a numbers:	**output**

n=eval(input("enter n"))
i=1
fact=1
while(i<=n):
   fact=fact*i
   i=i+1
print(fact)
	enter n
	5
	120

Sum of digits of a number:	**output**

n=eval(input("enter a number"))
sum=0
while(n>0):
	enter a number
	123
	6

```
   a=n%10
```

```
   sum=sum+
   a n=n//10
print(sum)
```

Reverse the given number: | **output**

```
n=eval(input("enter a number"))
sum=0
while(n>0):
   a=n%10
   sum=sum*10+a
   n=n//10
print(sum)
```

```
enter a number
123
321
```

Armstrong number or not | **output**

```
n=eval(input("enter a number"))
org=n
sum=0
while(n>0):
   a=n%10
   sum=sum+a*a*a
   n=n//10
if(sum==org):
   print("The given number is Armstrong
number")
else:
   print("The    given    number    is
not Armstrong number")
```

```
enter a number153
The given number is Armstrong number
```

Palindrome or not | **output**

```
n=eval(input("enter a number"))
org=n
sum=0
while(n>0):
   a=n%10
   sum=sum*10+a
   n=n//10
if(sum==org):
   print("The given no is palindrome")
else:
   print("The given no is not palindrome")
```

```
enter a number121
The given no is palindrome
```

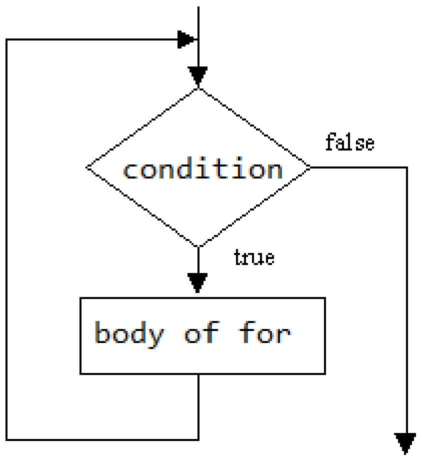
<u>For loop:</u>

- ***<u>for in range:</u>***
  - We can generate a sequence of numbers using range() function. range(10) will generate numbers from 0 to 9 (10 numbers).
  - In range function have to define the start, stop and step size as range(start,stop,step size). step size defaults to 1 if not provided.

<u>syntax</u>

```
for i in range(start,stop,steps):
        body of for loop
```

<u>Flowchart</u>:



<u>For in sequence</u>

- The for loop in Python is used to **iterate over a sequence (list, tuple, string).** Iterating over a sequence is called **traversal.** Loop continues until we reach the last element in the sequence.
- The body of for loop is separated from the rest of the code using indentation.

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

<u>Sequence can be a list, strings or tuples</u>

s.no	sequences	example	output

1.	For loop in string	for i in "Ramu":
		    print(i)
			R
			A
			M
			U

2.	For loop in list	for i in [2,3,5,6,9]:
		    print(i)
			2
			3
			5
			6
			9

3.	For loop in tuple	for i in (2,3,1):
		print(i)
			2
			3
			1

Examples:

1. print nos divisible by 5 not by 10:
2. Program to print fibonacci series.
3. Program to find factors of a given number
4. check the given number is perfect number or not
5. check the no is prime or not
6. Print first n prime numbers
7. Program to print prime numbers in range

print nos divisible by 5 not by 10	**output**

```
n=eval(input("enter a"))
for i in range(1,n,1):
   if(i%5==0 and i%10!=0):
      print(i)
```
	```
	enter a:30
	5
	15
	25
	```

Fibonacci series	**output**

```
a=0
b=1
n=eval(input("Enter the number of terms: "))
print("Fibonacci Series: ")
print(a,b)
for i in range(1,n,1):
   c=a+b
   print(c
   ) a=b
   b=c
```
	```
	Enter the number of terms: 6
	Fibonacci Series:
	0 1
	1
	2
	3
	5
	8
	```

find factors of a number	**Output**

```
n=eval(input("enter a number:"))
for i in range(1,n+1,1):
    if(n%i==0)
        : print(i)
```

```
enter a number:10
1
2
5
10
```

check the no is prime or not | **output**

```
n=eval(input("enter a number"))
for i in range(2,n):
   if(n%i==0):
      print("The num is not a prime")
      break
else:
   print("The num is a prime number.")
```

```
enter a no:7
The num is a prime number.
```

check a number is perfect number or not | **Output**

```
n=eval(input("enter a number:"))
sum=0
for i in range(1,n,1):
   if(n%i==0):
      sum=sum+i
if(sum==n):
   print("the number is perfect number")
else:
   print("the number is not perfect number")
```

```
enter a number:6
the number is perfect number
```

Program to print first n prime numbers | **Output**

```
number=int(input("enter no of prime numbers to be displayed:"))
count=1
n=2
while(count<=number):
   for i in range(2,n):
      if(n%i==0):
         break
   else:
      print(n)
      count=count+1
   n=n+1
```

```
enter no of prime numbers to be displayed:5
2
3
5
7
11
```

Program to print prime numbers in range | **output:**

```
lower=eval(input("enter  a  lower  range"))
upper=eval(input("enter a upper range"))
for n in range(lower,upper + 1):
  if n > 1:
     for i in range(2,n):
        if (n % i) == 0:
           break
     else:
        print(n)
```

```
enter a lower range50
enter a upper range100
53
59
61
67
71
73
79
83
89
97
```

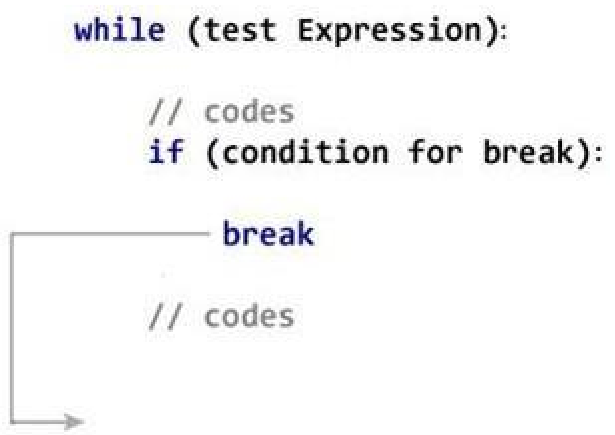
Loop Control Structures

BREAK

- ❖ Break statements can alter the flow of a loop.
- ❖ It terminates the current
- ❖ loop and executes the remaining statement outside the loop.
- ❖ If the loop has else statement, that will also gets terminated and come out of the loop completely.

Syntax:

break



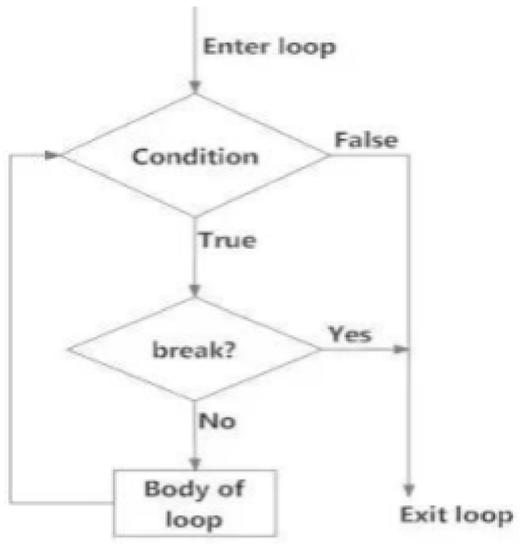
```
while (test Expression):

    // codes
    if (condition for break):

        break

    // codes
```

Flowchart



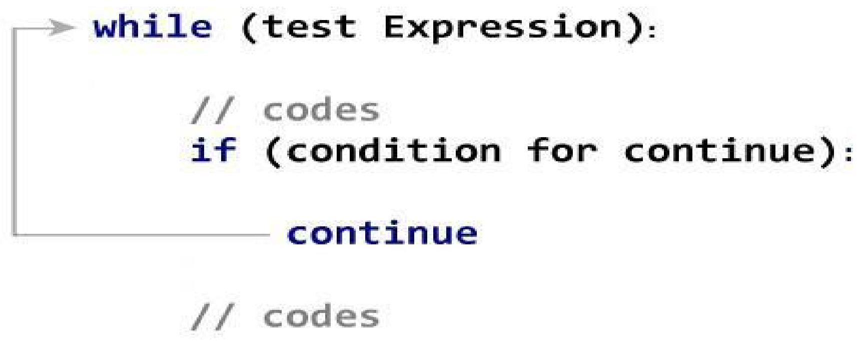
example	Output

for i in "welcome":
   if(i=="c"):
      brea
   k
   print(i)
	w
	e
	l

CONTINUE

It terminates the current iteration and transfer the control to the next iteration in the loop.

Syntax: Continue



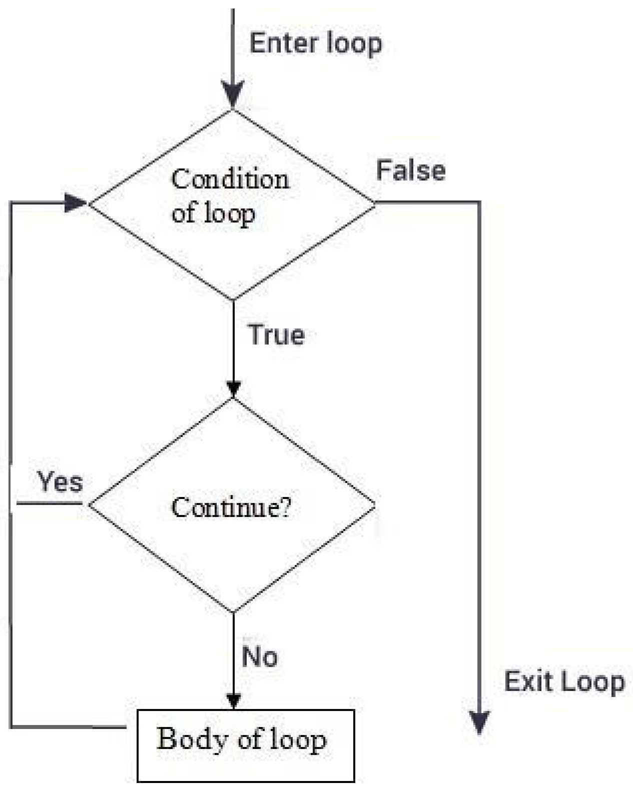
```
while (test Expression):

    // codes
    if (condition for continue):

        continue

    // codes
```

Flowchart



Example:	**Output**

for i in "welcome":
   if(i=="c"):
      continue
   print(i)
	w
	e
	l
	o
	m
	e

PASS

- ❖ It is used when a statement is required syntactically but you don't want any code to execute.
- ❖ It is a null statement, nothing happens when it is executed.

Syntax:

pass

break

Example	**Output**

for i in “welcome”:
    if (i == “c”):
        pas
    s
    print(i)
	w
	e
	l
	c
	o
	m
	e

Difference between break and continue

break	**continue**
It terminates the current loop and executes the remaining statement outside the loop.	It terminates the current iteration and transfer the control to the next iteration in the loop.
syntax: break	syntax: continue

for i in "welcome":
    if(i=="c"):
        break
    print(i)
	for i in "welcome":
	    if(i=="c"):
	        continue
	    print(i)

w e l	w e l o m e

else statement in loops:

else in for loop:

- ❖ If else statement is used in for loop, the else statement is executed when the loop has reached the limit.
- ❖ The statements inside for loop and statements inside else will also execute.

example	**output**

```python
for i in range(1,6):
   print(i)
else:
   print("the number greater than 6")
```

```
1
2
3
4
5 the number greater than 6
```

<u>else in while loop:</u>

- ❖ If else statement is used within while loop, the else part will be executed when the condition become false.
- ❖ The statements inside for loop and statements inside else will also execute.

Program	**output**

i=1
while(i<=5):
   print(i
) i=i+1
else:
   print("the number greater than 5")
	1
	2
	3
	4
	5
	the number greater than 5

Function

- ❖ Fruitful function
- ❖ Void function
- ❖ Parameters
- ❖ Local and global scope
- ❖ Function composition
- ❖ Recursion

<u>Fruitful function:</u>

A function that returns a value is called fruitful function.

<u>Example</u>:

```
        Root=sqrt(25)
```

<u>Example:</u>

```
def add():
        a=10
        b=20
        c=a+b
        return c
c=add(
)
print(c
)
```

Void Function

A function that perform action but don’t return any value.

Example:

```
        print(“Hello”)
```

Example:

```
def add():
        a=10
        b=20
```

add()	c = a + b p r i n t (c)

PARAMETERS / ARGUMENTS:

- ❖ Parameters are the variables which used in the function definition. Parameters are inputs to functions. Parameter receives the input from the function call.
- ❖ It is possible to define more than one parameter in the function definition.

Types of parameters/Arguments:

1. Required/Positional parameters
2. Keyword parameters
3. Default parameters
4. Variable length parameters

Required/ Positional Parameter:

The number of parameter in the function definition should match exactly with number of arguments in the function call.

Example	Output:

def student(name, roll):
        print(name,roll)
student("George",98)
	George 98

Keyword parameter:

When we call a function with some values, these values get assigned to the parameter according to their position. When we call functions in keyword parameter, the order of the arguments can be changed.

Example	Output:

def student(name,roll,mark):
        print(name,roll,mark)
student(90,102,"bala")
	90 102 bala

<u>Default parameter:</u>

Python allows function parameter to have default values; if the function is called without the argument, the argument gets its default value in function definition.

Example	**Output:**

def student(name, age=17):
   print (name, age)
student(“kumar”):
student(“ajay”):
	Kumar 17
	
	Ajay 17

<u>Variable length parameter</u>

- ❖ Sometimes, we do not know in advance the number of arguments that will be passed into a function.
- ❖ Python allows us to handle this kind of situation through function calls with number of arguments.
- ❖ In the function definition we use an asterisk (*) before the parameter name to denote this is variable length of parameter.

Example	**Output:**

def student(name,*mark):
        print(name,mark)
student (“bala”,102,90)
	bala (102 ,90)

<u>Local and Global Scope</u>

<u>Global Scope</u>

- ❖ The *scope* of a variable refers to the places that you can see or access a variable.
- ❖ A variable with global scope can be used anywhere in the program.
- ❖ It can be created by defining a variable outside the function.

Example	**output**

a=50
def add():
Global Variable
   b=20
   c=a+b
   print©
Local Variable
def sub():
   b=30
   c=a-b
   print
   ©
print(a)
	70
	
	20
	
	50

Local Scope A variable with local scope can be used only within the function .

Example	**output**

def add():
   b=20 → **Local Variable**
   c=a+b
   print©
def sub():
   b=30 → **Local Variable**
   c=a-b
   print©
print(a)
print(b)
	
	70
	
	20
	
	error
	error

Function Composition:

- ❖ Function Composition is the ability to call one function from within another function
- ❖ It is a way of combining functions such that the result of each function is passed as the argument of the next function.
- ❖ In other words the output of one function is given as the input of another function is known as function composition.

Example:	**Output:**

def
   add(a,b):
   c=a+b
   return c
def mul(c,d):
    e=c*d
    return e
c=add(10,20)
e=mul(c,30)
print(e)
	900

find sum and average using function composition	**output**

def sum(a,b):
   sum=a+b
   return sum
def avg(sum):
   avg=sum/2
   return avg
a=eval(input("enter a:"))
b=eval(input("enter b:"))
sum=sum(a,b)
avg=avg(sum)
print("the avg is",avg)
	enter a:4
	enter b:8
	the avg is 6.0

Recursion

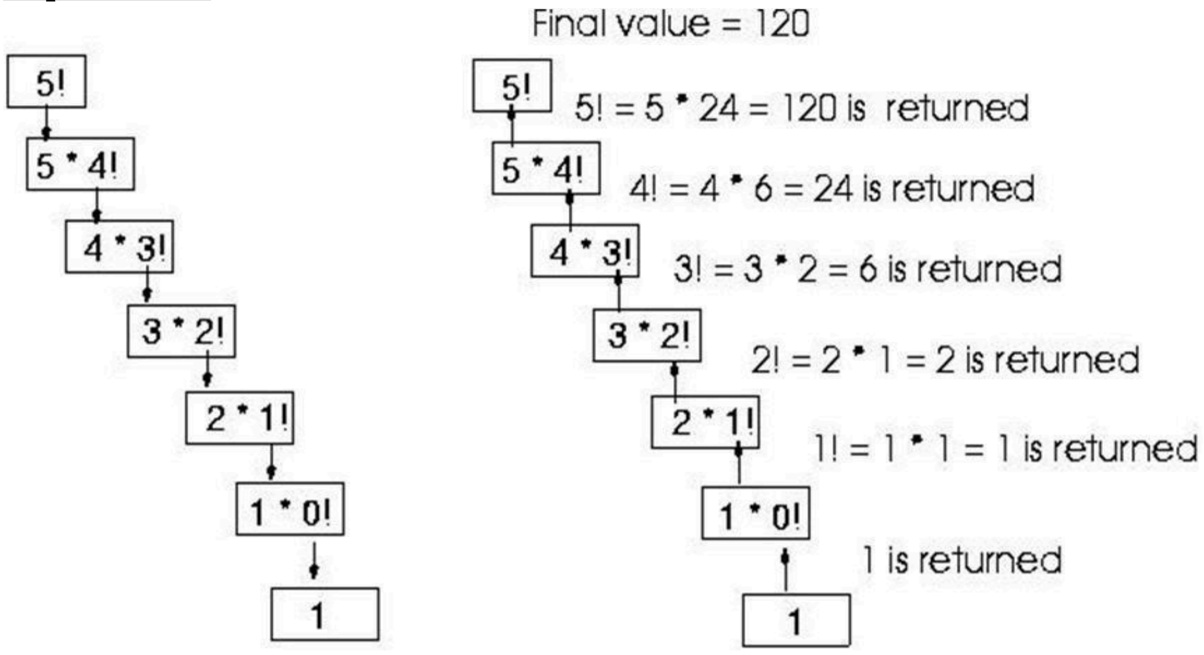
A function calling itself till it reaches the base value - stop point of function call.

Example: factorial of a given number using recursion

Factorial of n	**Output**

def fact(n):
   if(n==1)
   :
      return 1
   else:
      return n*fact(n-1)
 	
n=eval(input("enter no. to find fact:"))
fact=fact(n)
print("Fact is",fact)
	enter no. to find fact:5
	Fact is 120

Explanation


5!
5 * 4!
4 * 3!
3 * 2!
2 * 1!
1 * 0!
1
Final value = 120
5! = 5 * 24 = 120 is returned
4! = 4 * 6 = 24 is returned
3! = 3 * 2 = 6 is returned
2! = 2 * 1 = 2 is returned
1! = 1 * 1 = 1 is returned
1 is returned

Examples:

1. sum of n numbers using recursion
2. **2.** exponential of a number using recursion

Sum of n numbers	**Output**

```
def sum(n): if(n==1):
      return 1 else:
      return n+sum(n-1)


n=eval(input("enter    no.    to    find sum:"))
sum=sum(n)
print("Fact is",sum)
```
	```
	enter no. to find sum:10
	Fact is 55
	```

Definition and Usage

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

Syntax

range(*start, stop, step*)

Parameter	**Description**
start	Optional. An integer number specifying at which position to start. Default is 0
stop	Required. An integer number specifying at which position to stop (not included).
step	Optional. An integer number specifying the incrementation. Default is 1

```
x = range(3, 6)
for n in x:
  print(n)
```