



# **SNS COLLEGE OF ENGINEERING**

Kurumbapalayam (PO), Coimbatore - 641 107

Accredited by NAAC-UGC with 'A' Grade

Approved by AICTE, Recognized by UGC & Affiliated to Anna University, Chennai

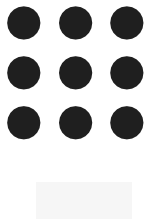
## **DEPARTMENT OF INFORMATION TECHNOLOGY**

### **COURSE NAME: 23ITB202-PYTHON PROGRAMMING**

#### **II YEAR/ III SEM**

#### **Unit : INTRODUCTION**

#### **Topic : Conditional statements and loop.**





# Algorithm

- Algorithm is a step-by-step procedure, which defines a set of instructions to be executed in a certain order to get the desired output.
- Algorithms are generally created independent of underlying languages, i.e. an algorithm can be implemented in more than one programming language.



# How to Write an Algorithm?

- There are no well-defined standards for writing algorithms. Rather, it is problem and resource dependent. Algorithms are never written to support a particular programming code.
- As we know that all programming languages share basic code constructs like loops (do, for, while), flow-control (if-else), etc. These common constructs can be used to write an algorithm.



# Example

**Problem – Design an algorithm to add two numbers and display the result.**

step 1 – START

step 2 – declare three integers a, b & c

step 3 – define values of a & b

step 4 – add values of a & b

step 5 – store output of step 4 to c

step 6 – print c

step 7 – STOP



# Alternative

step 1 – START ADD

step 2 – get values of a & b

step 3 –  $c \leftarrow a + b$

step 4 – display c

step 5 – STOP



# Flowcharts

- Flowcharts graphically represent the flow of a program. There are four basic shapes used in a flow chart. Each shape has a specific use:

oval: start / end

parallelogram: input / output

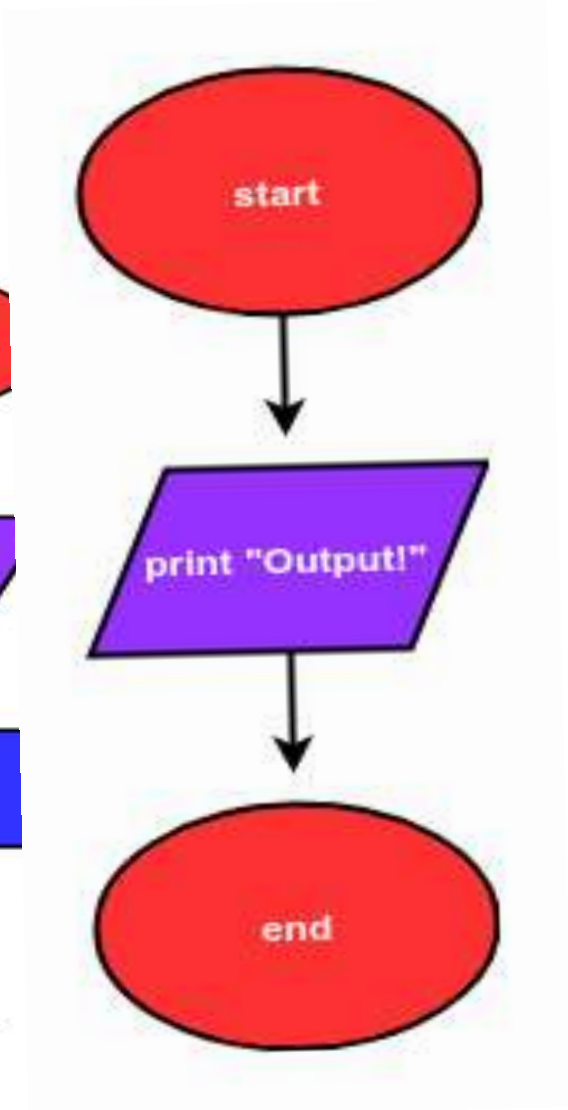
rectangle: calculations

diamond: selection structures



# Example

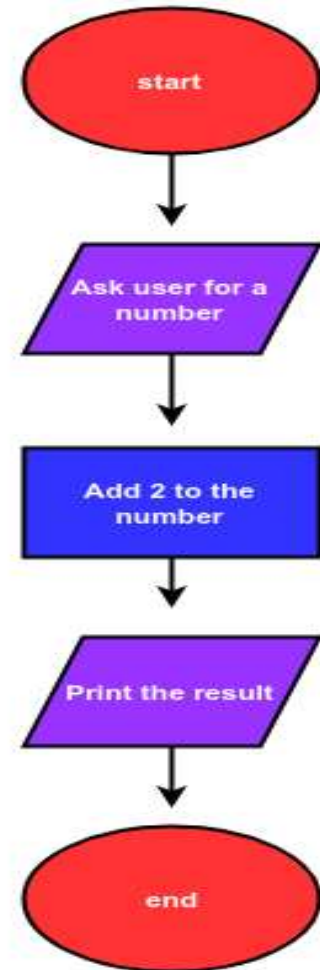
```
# start  
print("Output!")  
# end
```










# Example 2

```
# start  
num = input("Enter a number: ")  
num = float(num)  
num_plus_2 = num + 2  
print(num_plus_2)  
# end
```





Symbol	Name	Function
	Start/end	An oval represents a start or end point
	Arrows	A line is a connector that shows relationships between the representative shapes
	Input/Output	A parallelogram represents input or output
	Process	A rectangle represents a process
	Decision	A diamond indicates a decision



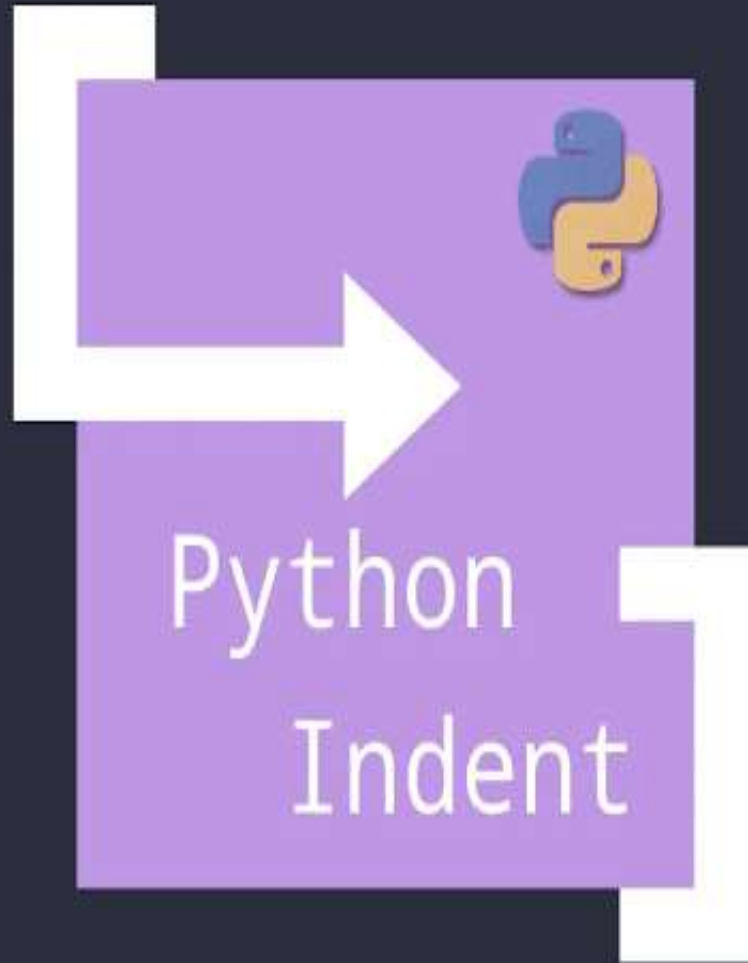
# Flow Chart Symbols

Reference: IBM Corp., "IBM Flowcharting Techniques GC20-8152-1", March 1970. <http://www.ibm.gov.de/~keine/histor/software/IBM-FlowchartingTechniques-GC20-8152-1.pdf>

	<b>Terminator</b> Start, Stop or Halt the process.		<b>Link/Connector   Inspection</b> like-labeled pair to link process sections on the same page <b>BPM:</b> Inspection point
	<b>Decision</b> select one of several paths based on a conditional statement		<b>Off Page Link/Connector</b> Indicates flow continues on another page
	<b>Delay</b> Waiting period, during which there is no activity.		<b>Manual Operation</b> offline operation or process adjustment done manually
	<b>Preparation</b> Any preparation process		<b>Manual Input</b> Data or command entry done manually
	<b>Process</b> Any operation or activity		<b>Display</b> Information to be displayed in any manner, such as a screen or printer
	<b>Predefined Process</b> Named process(es) defined elsewhere		<b>Transfer</b> Physical movement of item(s)
	<b>Alternate Process</b> Alternative to a standard process (rectangle). Usually connected by a dashed arrow line.		<b>Annotation</b>
	<b>Sort</b> Order items sequentially		<b>Flow Lines</b>
	<b>Collate</b> Merge & Extract: create multiple sets of items from multiple other sets	<b>Data / I/O</b> Input/Output: generate or receive some type of data	
	<b>Merge   Storage</b> Combine multiple sets into one <b>BPM:</b> raw material storage	Specific media: Document Multi-document Sequential Access Storage / Magnetic Tape	
	<b>Extract   Measurement</b> Remove specific set(s) of items from a larger set <b>BPM:</b> finished goods storage	<b>Online Storage / Stored Data</b> I/O using any type of online magnetic storage	
	<b>Or</b> Logical OR	Specific media for I/O: Direct Access Storage, Hard Disk Drive (HDD) Database or Data File Core, Internal Storage, Random Access Memory (RAM)	
	<b>Summing Function</b> Logical AND		

Revised 05-23-2008

Created by Spencer Selie





# Indentation

- Indentation refers to the spaces at the beginning of a code line.
- Where in other programming languages the indentation in code is for readability only, the indentation in Python is very important.
- Python uses indentation to indicate a block of code.

# Python Indentation



```
code
```

```
code
```

```
code
```

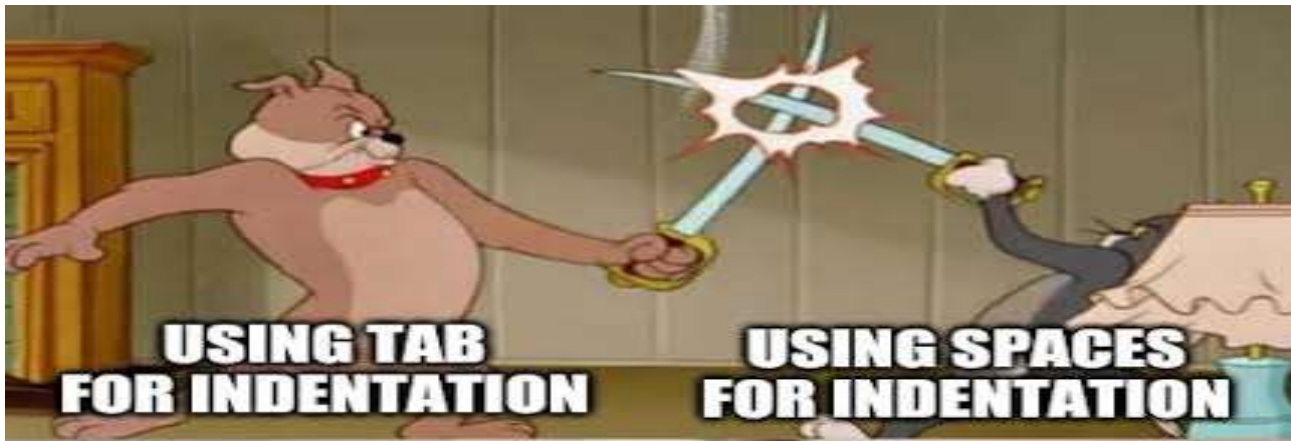


Python

## Statement, Indentation, and Comments







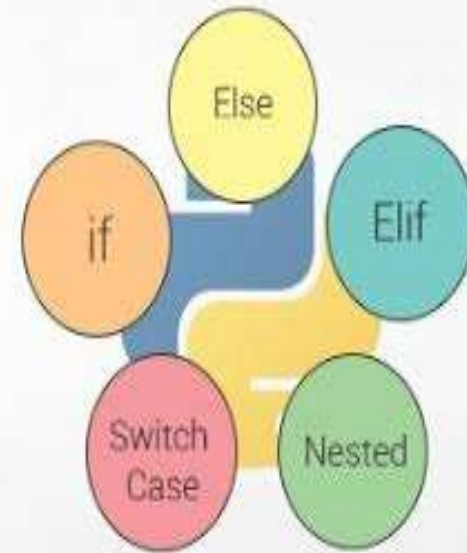
```
int main (int argc, char* arg[]) {  
    ;;;int i = 0  
    ;;;while (i < 10) {  
        ;;;;printf("%d\n", i)  
        ;;;;i ++  
    ;;;}  
    ;;;return 0  
}
```

lingtip.com



# Python

## Conditional Statements







# Conditional statements

- conditional statement as the name suggests itself, is used to handle conditions in your program. These statements guide the program while making decisions based on the conditions encountered by the program.
- Python has 3 key Conditional Statements that you should know:
  - if statement
  - if-else statement
  - if-elif-else ladder



# If statement

- The if statement is a conditional statement in python, that is used to determine whether a block of code will be executed or not.
- Meaning if the program finds the condition defined in the if statement to be true, it will go ahead and execute the code block inside the if statement.



# Syntax

if condition:

    # execute code block



# Flowchart for IF statement

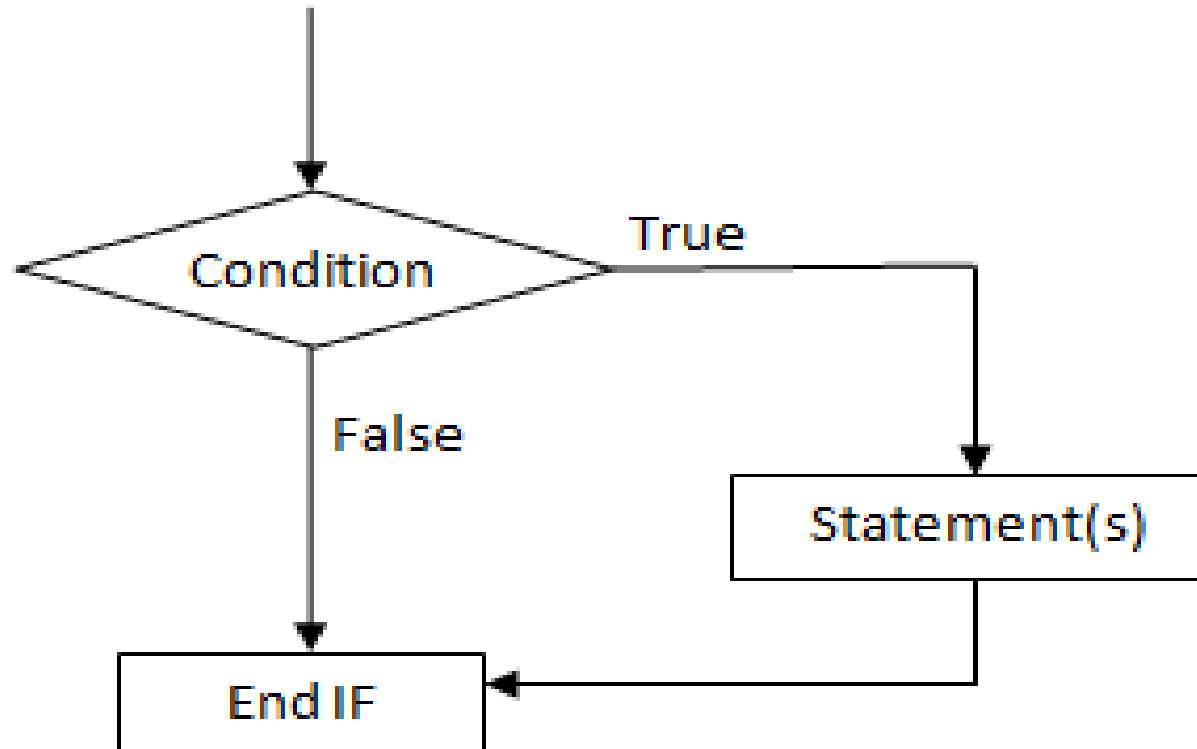


fig: Flowchart for if statement



# Working of IF

## Condition is True

```
number = 10  
if number > 0:  
    # code  
  
# code after if
```

## Condition is False

```
number = -5  
if number > 0:  
    # code  
  
# code after if
```



# Example

```
number = 10
# check if number is greater than 0
if number > 0:
    print('Number is positive.')
print('The if statement is easy')
```

```
Number is positive.
The if statement is easy
> |
```



# Example

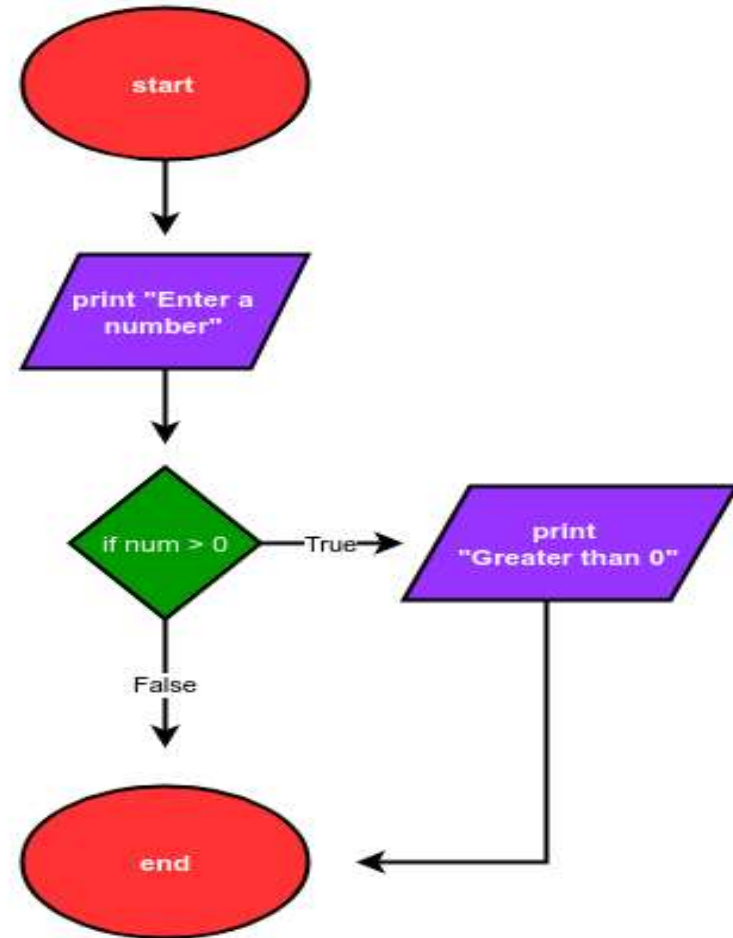
```
num = int(input("enter the number?"))  
if num%2 == 0:  
    print("Number is even")
```

```
enter the number? 56  
56  
Number is even  
> |
```

# Example

```
# start  
num = input("Enter a number: ")  
num = float(num)  
if num > 0:  
    print("Greater than 0")  
# end
```

```
main.py Shell  
Enter a number: 66  
Greater than 0  
Enter a number: -2  
> |  
7/25/2024
```







# Example

```
a = int(input("Enter a? "));  
b = int(input("Enter b? "));  
c = int(input("Enter c? "));  
if a>b and a>c:  
    print("a is largest");  
if b>a and b>c:  
    print("b is largest");  
if c>a and c>b:  
    print("c is largest");
```

```
Enter a? 5  
Enter b? 77  
Enter c? 88  
c is largest  
> |
```



# If else

- if statement executes the code block when the condition is true. Similarly, the else statement works in conjunction with the if statement to execute a code block when the defined if condition is false.



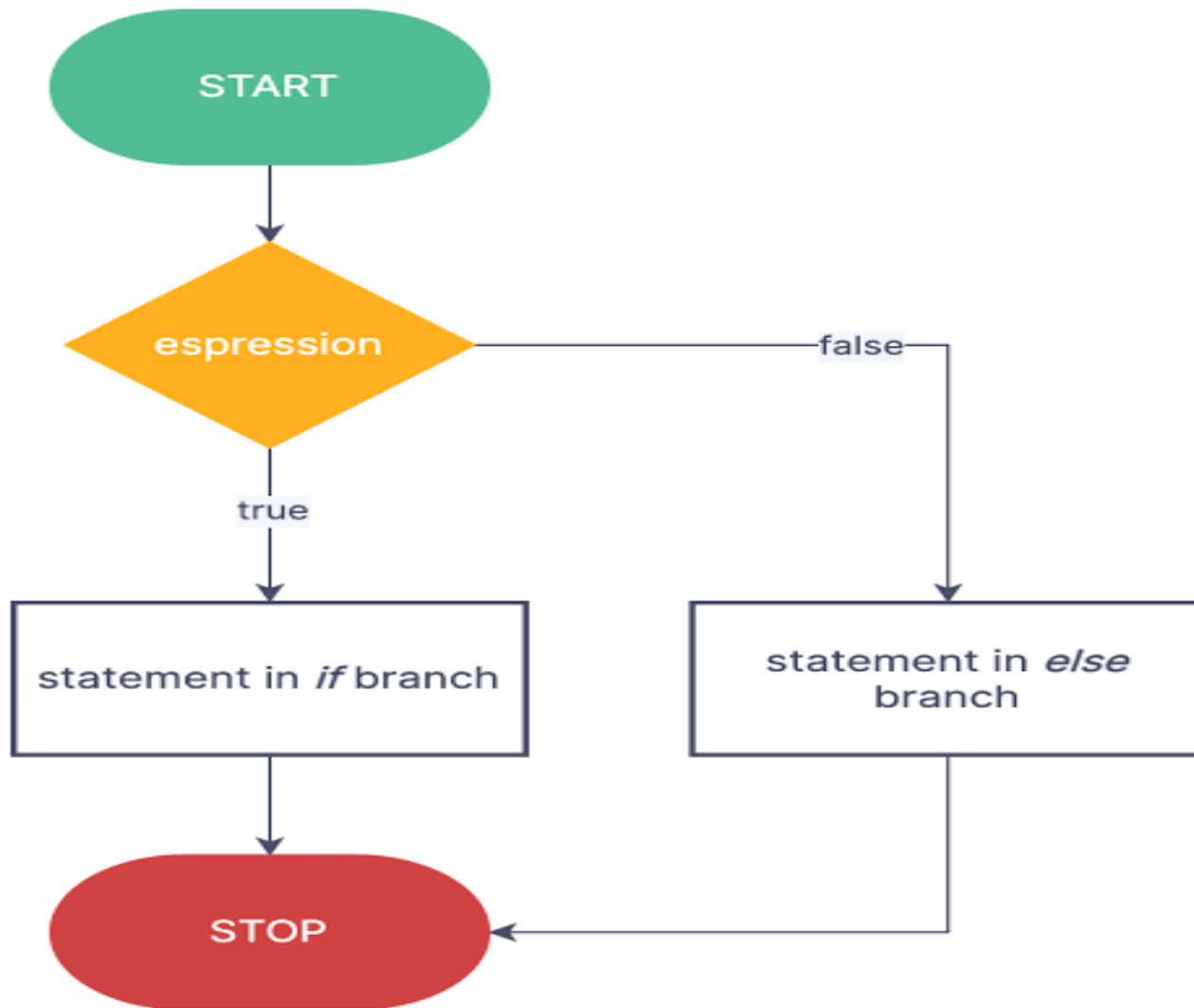
# Syntax

if condition:

```
# execute code if condition is true
```

else:

```
# execute code if condition if False
```





# Working process of IF-ELSE

## Condition is True

```
number = 10
```

```
if number > 0:
```

```
    # code
```

```
else:
```

```
    # code
```

```
# code after if
```

## Condition is False

```
number = -5
```

```
if number > 0:
```

```
    # code
```

```
else:
```

```
    # code
```

```
# code after if
```



# Example

```
number = 10
```

```
if number > 0:
```

```
    print('Positive number')
```

```
else:
```

```
    print('Negative number')
```

```
print('This statement is always executed')
```

```
Positive number
```

```
This statement is always executed
```

```
> |
```



# Example

```
age = int (input("Enter your age? "))  
if age>=18:  
    print("You are eligible to vote !!");  
else:  
    print("Sorry! you have to wait !!");
```

```
Enter your age? 29  
You are eligible to vote !!  
> |
```



# Elif

- The elif statement enables us to check multiple conditions and execute the specific block of statements depending upon the true condition among them.
- We can have any number of elif statements in our program depending upon our need. However, using elif is optional.





# Syntax

if condition1:

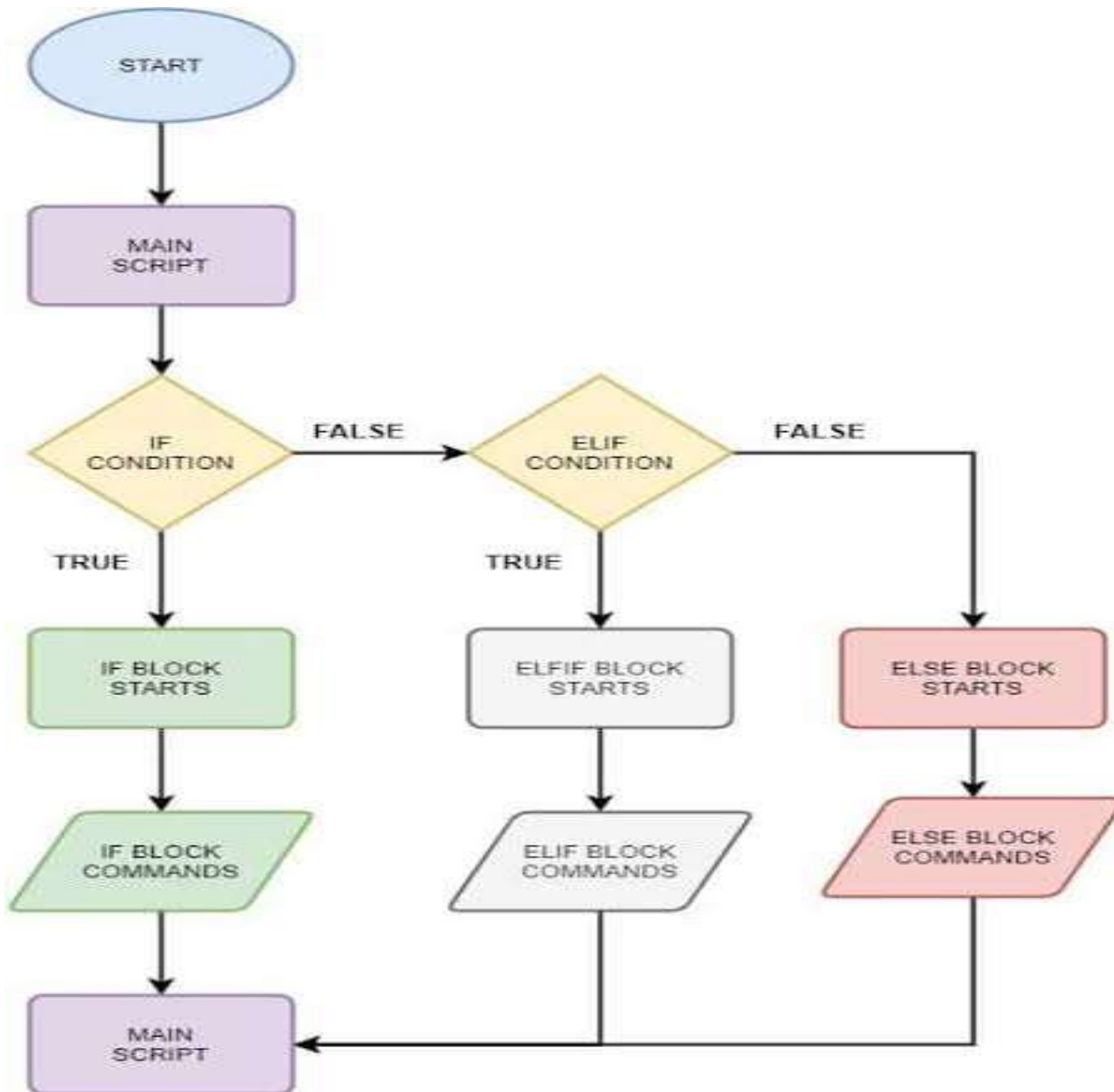
    # code block 1

elif condition2:

    # code block 2

else:

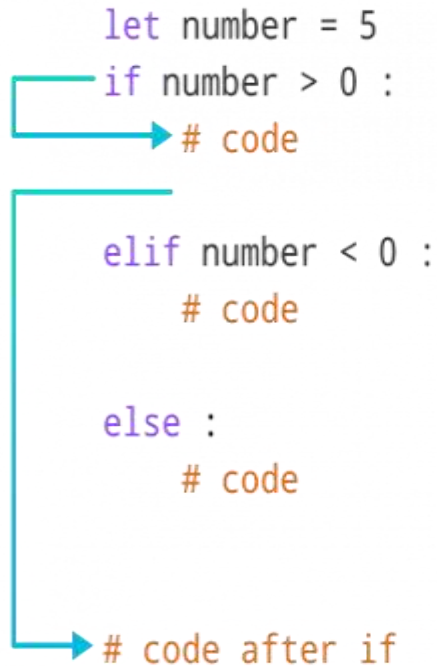
    # code block 3



# Working of elif

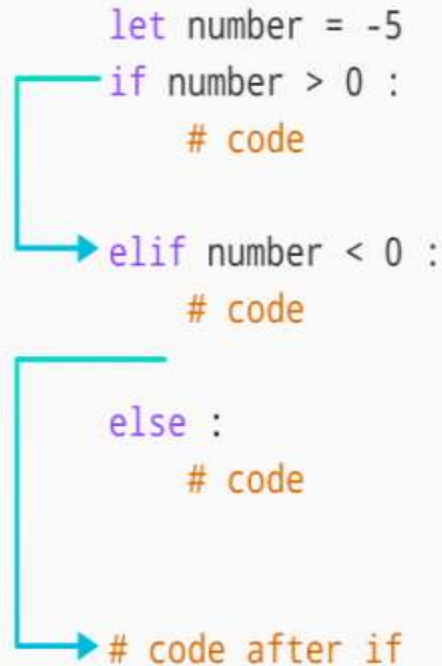
## 1st Condition is True

```
let number = 5
if number > 0 :
    # code
elif number < 0 :
    # code
else :
    # code
# code after if
```



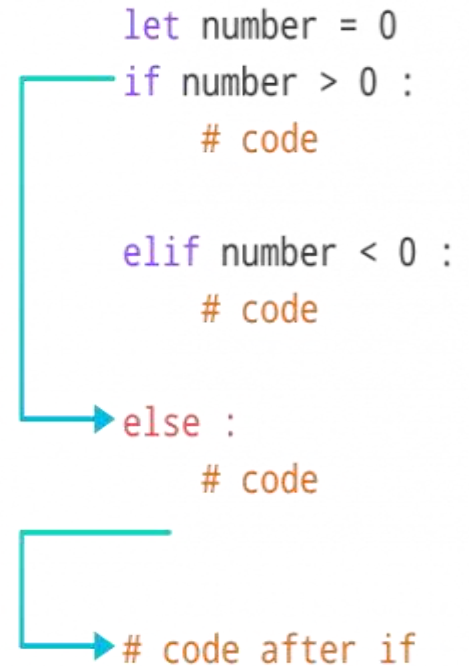
## 2nd Condition is True

```
let number = -5
if number > 0 :
    # code
elif number < 0 :
    # code
else :
    # code
# code after if
```



## All Conditions are False

```
let number = 0
if number > 0 :
    # code
elif number < 0 :
    # code
else :
    # code
# code after if
```





# Example

```
number = 0
if number > 0:
    print("Positive number")
elif number == 0:
    print('Zero')
else:
    print('Negative number')
print('This statement is always executed')
```

```
Zero
This statement is always executed
> |
```



# Python Nested if statements

- We can also use an if statement inside of an if statement. This is known as a nested if statement.



# Syntax

```
# outer if statement
```

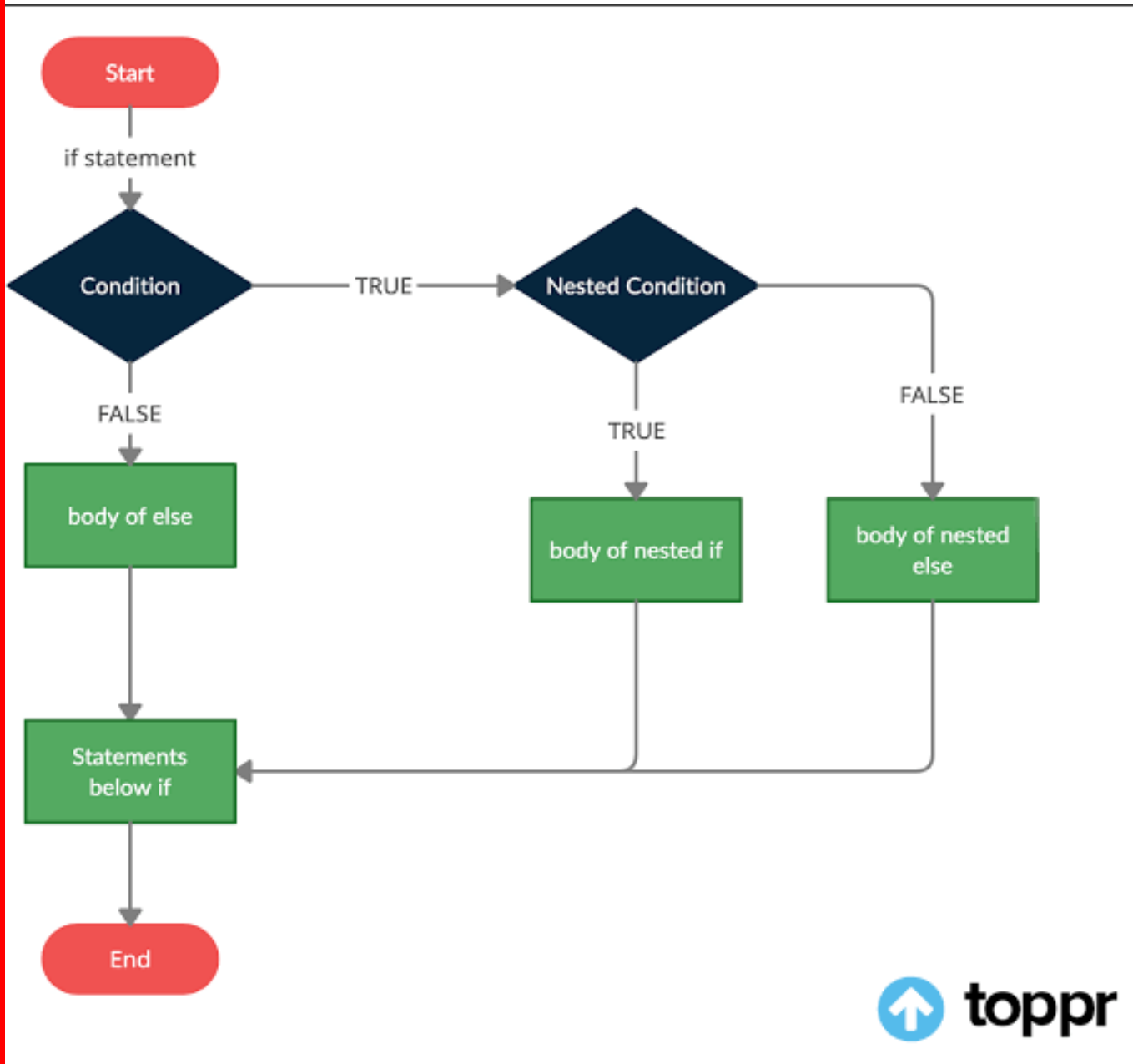
```
if condition1:
```

```
    # statement(s)
```

```
    # inner if statement
```

```
        if condition2:
```

```
            # statement(s)
```





# Example

```
number = 5
# outer if statement
if (number >= 0):
    # inner if statement
    if number == 0:
        print('Number is 0')

    # inner else statement
    else:
        print('Number is positive')
# outer else statement
else:
    print('Number is negative')
# Output: Number is positive
```

```
Number is positive
> |
```

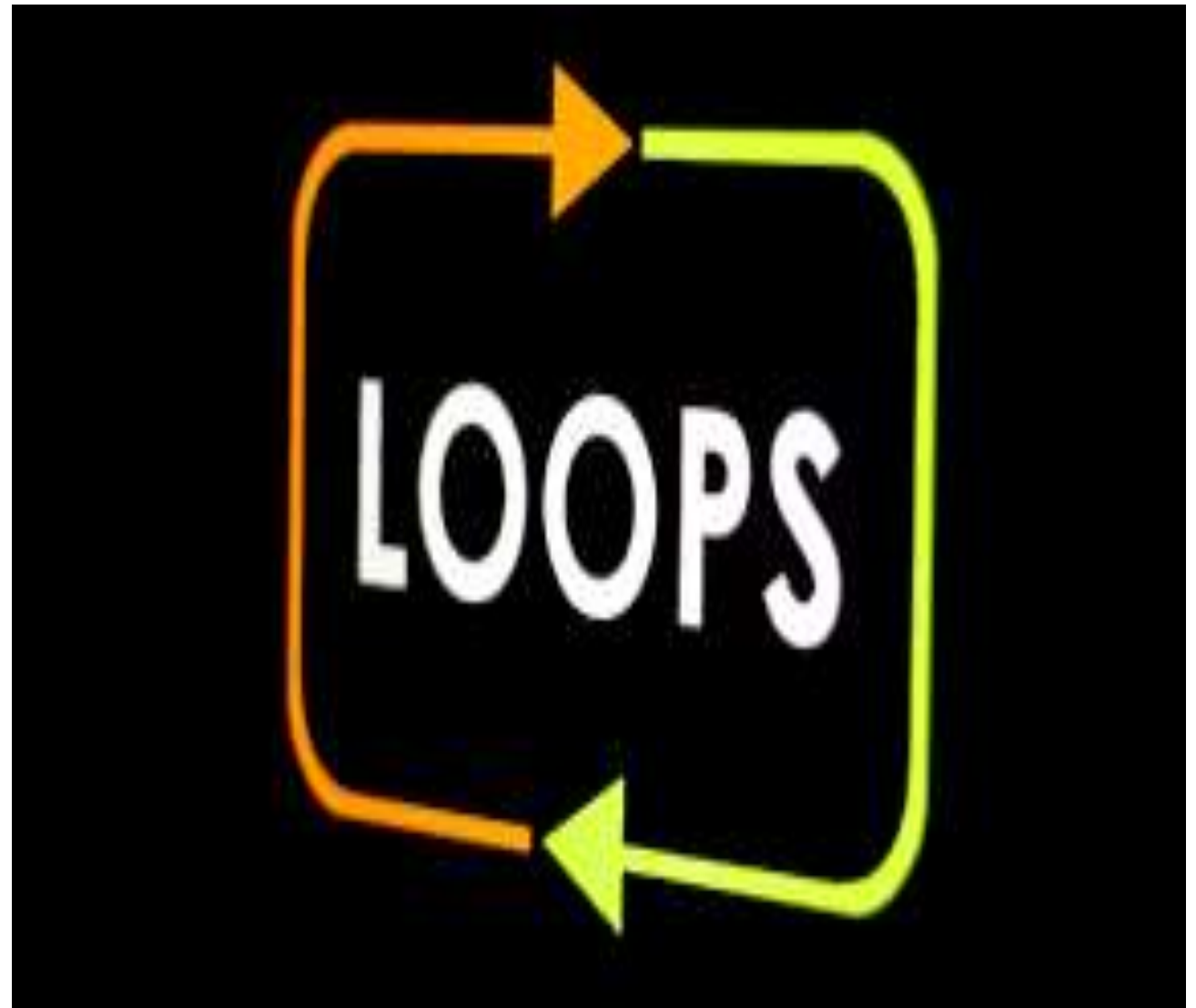




# Example

```
number = int(input("Enter the number?"))  
if number==10:  
    print("number is equals to 10")  
elif number==50:  
    print("number is equal to 50");  
elif number==100:  
    print("number is equal to 100");  
else:  
    print("number is not equal to 10, 50 or 100");
```

```
Enter the number? 6  
6  
number is not equal to 10, 50 or 100  
> |
```





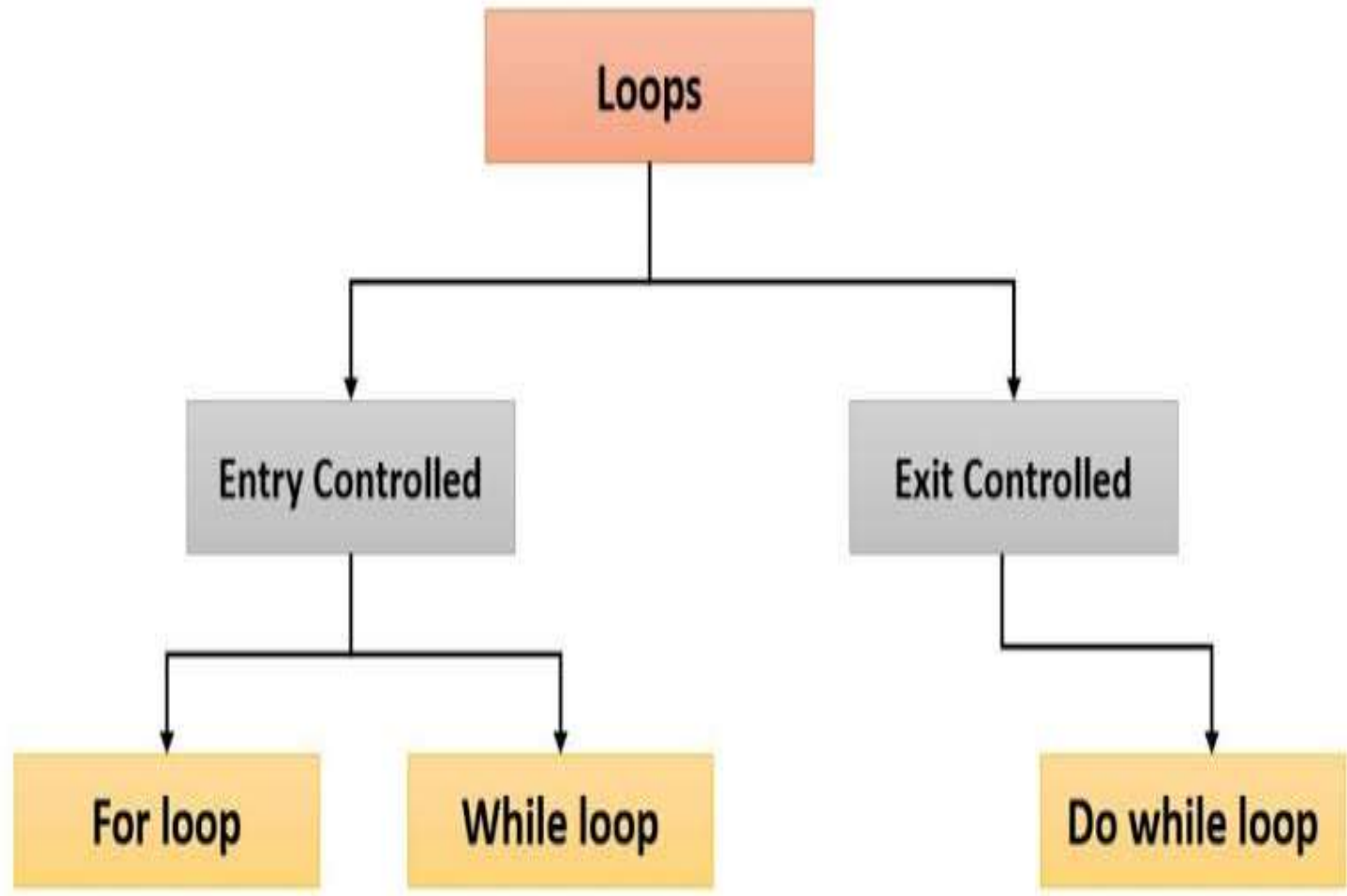
# Looping statement



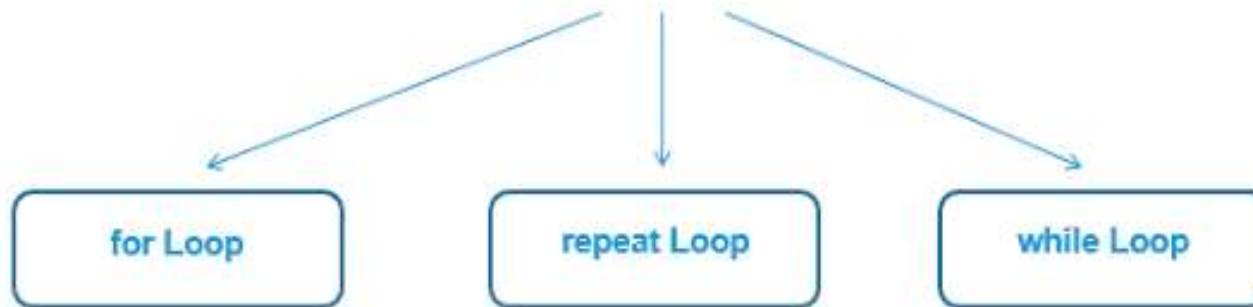


# JAVA Types of Loops

- There are three types of loops:
  - while Loops
  - for Loops
  - do Loops
- Each loop requires the following steps:
  - Initialization (get ready to start looping)
  - Condition (test if we should execute loop body)
  - Update (change something each time through)



# Types of Loops



- Iterates over the elements of any sequence (vector) **till the condition defined is true**
- **Number of iterations** are fixed and known in advance

- **Infinite loop** and used with **break statement** to exit the loop
- **Number of iterations** depends on the condition which is checked at the end of each iteration

- Repeats a statement or group of statements **until some condition is met**
- **Number of iterations** depends on the condition which is checked at the beginning of each iteration



# While loop

- while loop is used to execute a block of statements repeatedly until a given condition is satisfied. And when the condition becomes false, the line immediately after the loop in the program is executed.



# Syntax

Syntax :

while expression:

    statement(s)

All the statements indented by the same number of character spaces after a programming construct are considered to be part of a single block of code. Python uses indentation as its method of grouping statements.

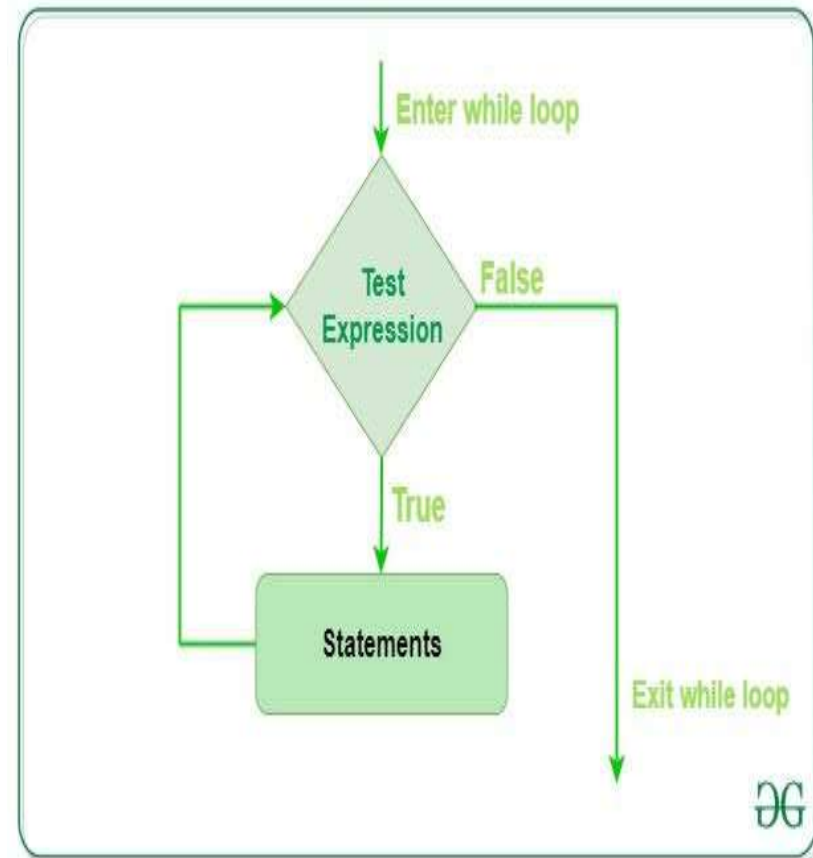
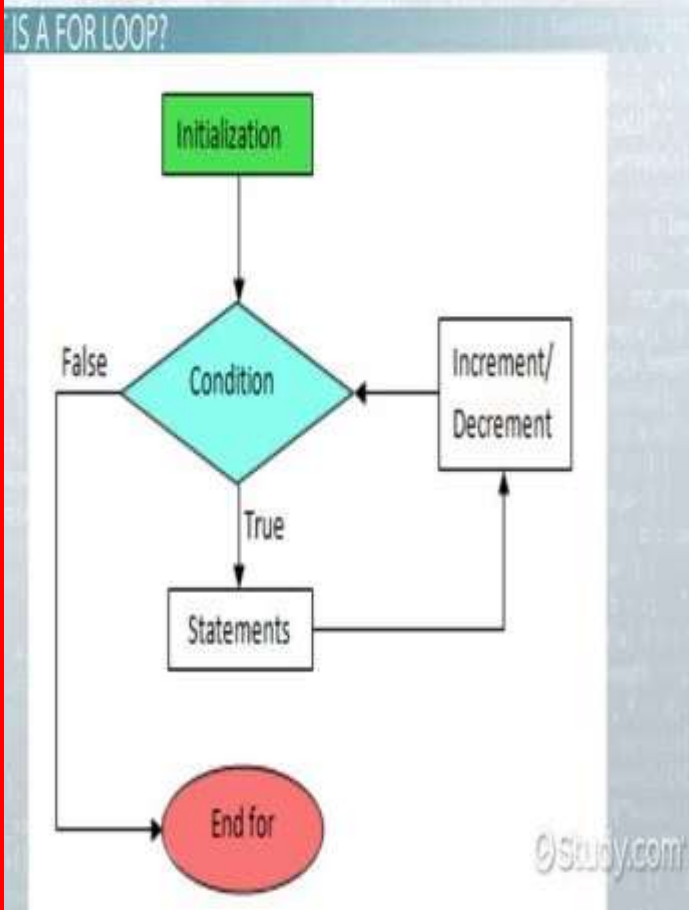




- The statements of the Python while loop are dictated by indentation.
- The code block begins when a statement is indented & ends with the very first unindented statement



# Flow chart





# Example

```
i = 1
while i < 6:
    print(i)
    i += 1
```

```
1
2
3
4
5
> |
```



## The Multiplication Table of: 12

$$12 \times 1 = 12$$

$$12 \times 2 = 24$$

$$12 \times 3 = 36$$

$$12 \times 4 = 48$$

$$12 \times 5 = 60$$

$$12 \times 6 = 72$$

$$12 \times 7 = 84$$

$$12 \times 8 = 96$$

$$12 \times 9 = 108$$

$$12 \times 10 = 120$$

> |



# Multiplication Table using While Loop

```
num =12
counter = 1
# we will use a while loop for iterating 10 times
for the multiplication table
print("The Multiplication Table of: ", num)
while counter <= 10: # specifying the condition
    ans = num * counter
    print (num, 'x', counter, '=', ans)
    counter += 1 # expression to increment the
counter
```

# While Loop Multiple Conditions

```
num1 = 17
```

```
num2 = -12
```

```
while num1 > 5 and num2 < -5 : # multiple conditions  
in a single while loop
```

```
    num1 -= 2
```

```
    num2 += 3
```

```
    print( (num1, num2) )
```

```
(15, -9)
```

```
(13, -6)
```

```
(11, -3)
```

```
> |
```

# While Loops in Python

## Learning Objectives:

Remember the following Programming Skills:

- FOR Loops
- WHILE Loops

## 'While Loops' in Python

The **x** is simply a variable. It could have any name.

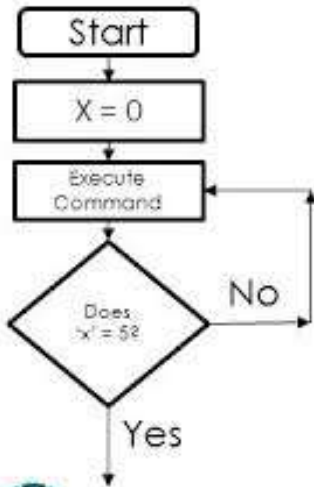
It is however a special kind of variable known as the '**most recent value**'

We must finish the statement with a colon

`while x == n:`  
`while x != n:`  
`while x > n:`  
`while x < n:`

The **n** represents a value that we want **x** to either equal, not equal, be greater than, etc. depending on the **condition** we want to use.

E.g. **n=5** and the condition **while x != 5** (not equal to 5) then the loop would repeat until **x** equals 5.





# Do while loop

- The do while construct consists of a process symbol and a condition. First the code within the block is executed. Then the condition is evaluated.
- If the condition is true the code within the block is executed again. This repeats until the condition becomes false.

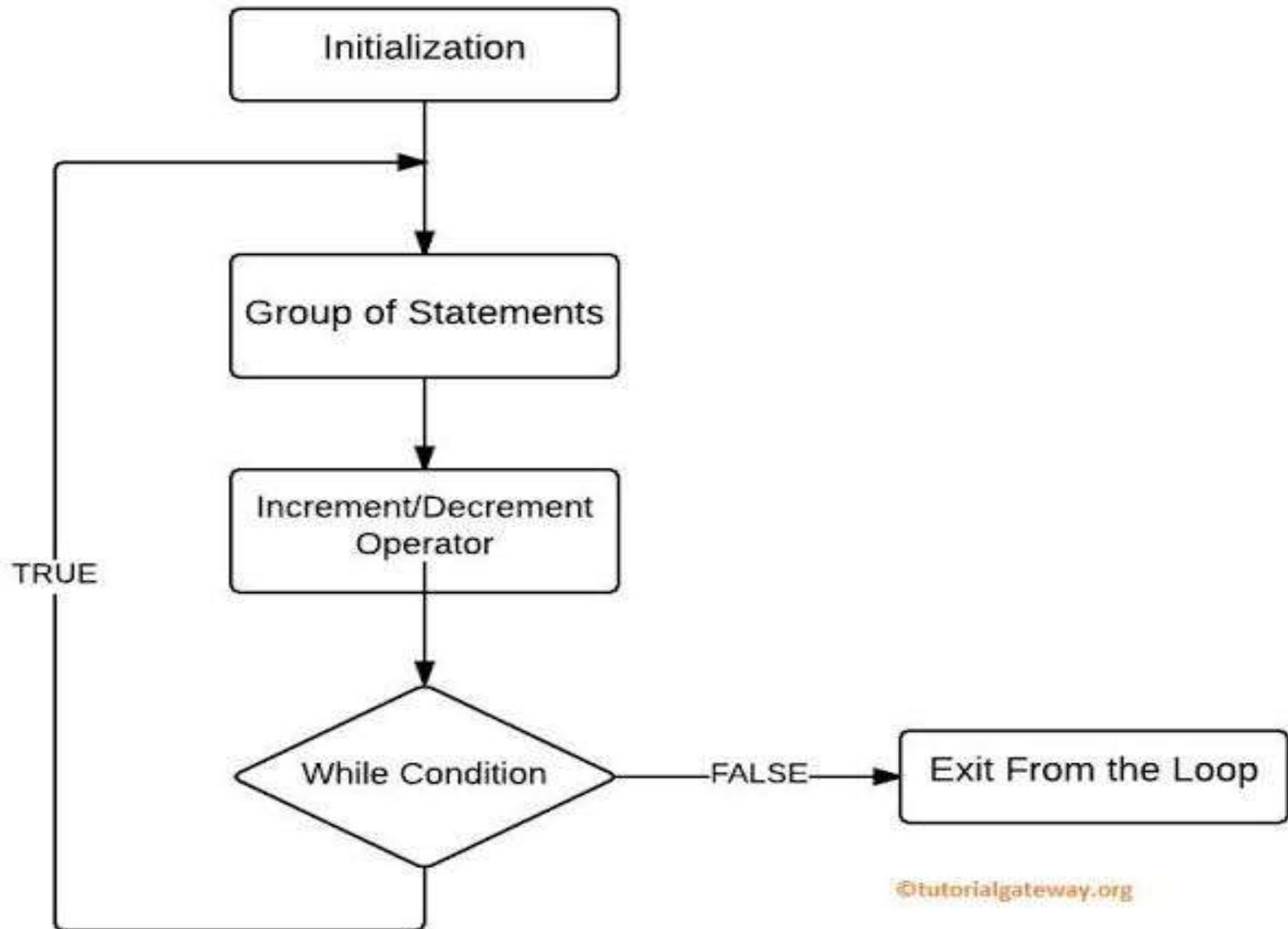




# Syntax

```
do {  
    loop block statement to be executed;  
}  
while(condition);
```

# Flowchart



©tutorialgateway.org



# Do while

- The while loop in python first checks for condition, and then the block is executed if the condition is true. The block is executed repeatedly until the condition is evaluated to false.
- Thus, in python, we can use a while loop with if/break/continue statements that are indented, but if we use do-while, it does not fit the indentation rule.
- Therefore we cannot use the do-while loop in python.



# Example

```
i = 1
while True:
    print(i)
    i = i + 1
    if(i > 5):
        break
```

```
1
2
3
4
5
> |
```



# For loop

- A for loop is used for iterating over a sequence (that is either a list, a tuple, a dictionary, a set, or a string).

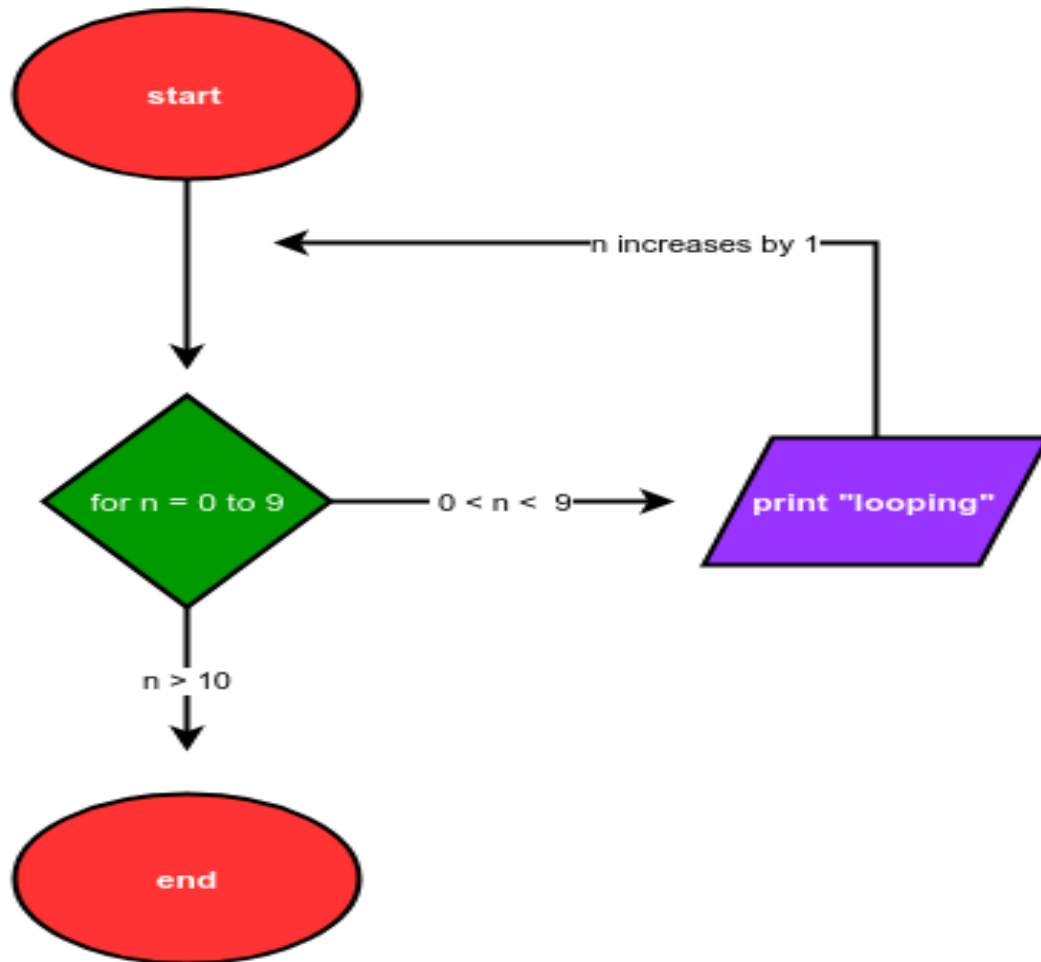


# Syntax

```
for val in sequence:  
    # statement(s)
```



# Flow chart





# Example

```
fruits = ["apple", "banana", "cherry"]
```

```
for x in fruits:
```

```
    print(x)
```

```
apple
banana
cherry
> |
```





# Python for loop

A for loop is **used for iterating over a sequence and iterables** (like range, list, a tuple, a dictionary, a set, or a string).





# Looping Through a String

```
for x in "banana":  
    print(x)
```

```
b  
a  
n  
a  
n  
a  
|
```



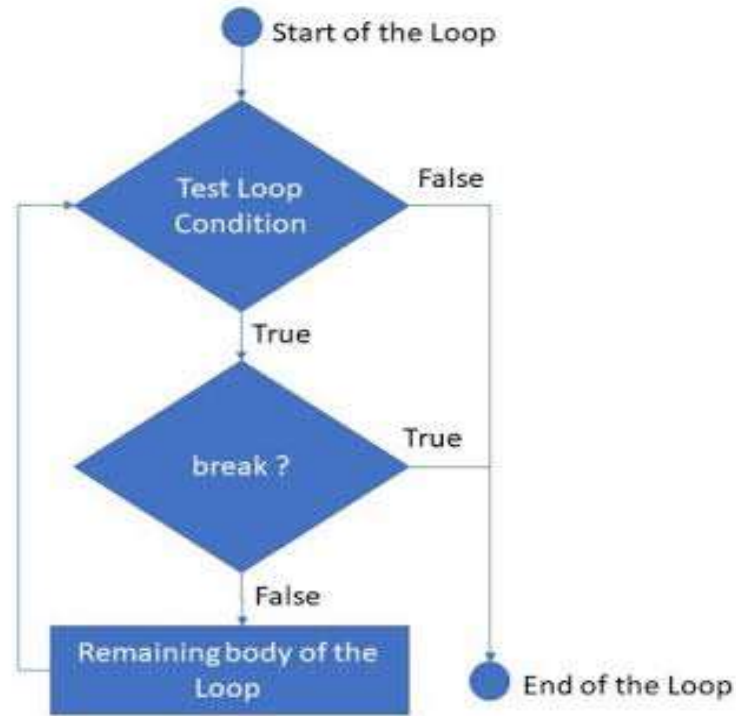
# The break Statement

```
fruits = ["apple", "banana", "cherry"]  
for x in fruits:  
    print(x)  
    if x == "banana":  
        break
```

```
apple  
banana  
> |
```



# Flow chart





Exit the loop when x is "banana",  
but this time the break comes  
before the print:

```
fruits = ["apple", "banana", "cherry"]  
for x in fruits:  
    if x == "banana":  
        break  
    print(x)
```

```
apple  
> |
```



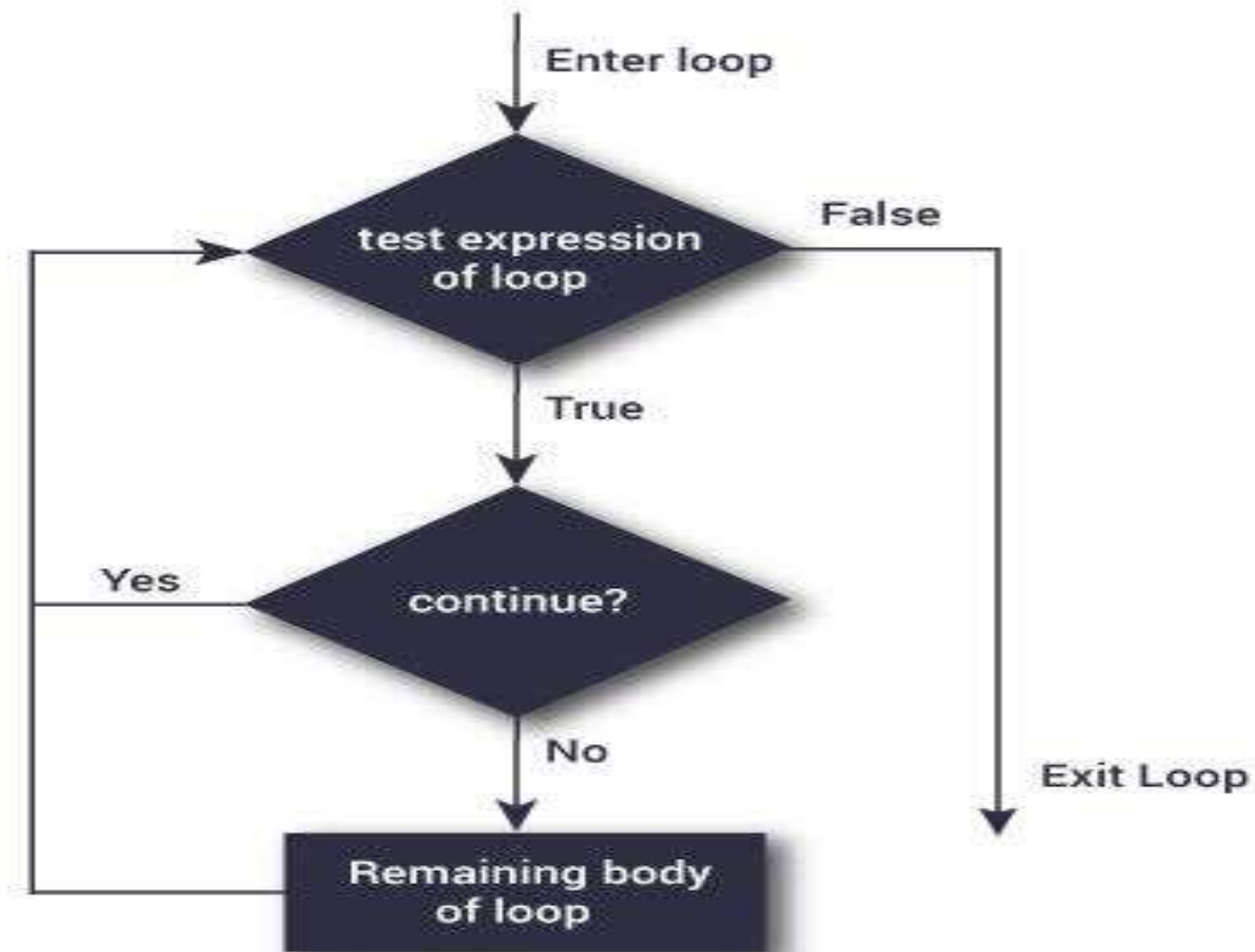
# The continue Statement

```
fruits = ["apple", "banana", "cherry"]  
for x in fruits:  
    if x == "banana":  
        continue  
    print(x)
```

```
apple  
cherry  
> |
```



# Flow chart





# The range() Function

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

```
for x in range(6):  
    print(x)
```

```
0  
1  
2  
3  
4  
5  
> |
```





# Using the start parameter:

```
for x in range(2, 6):  
    print(x)
```

```
2
```

```
3
```

```
4
```

```
5
```

```
> |
```



# Increment the sequence with 3 (default is 1):

```
for x in range(2, 30, 3):  
    print(x)
```

```
2  
5  
8  
11  
14  
17  
20  
23  
26  
29  
> |
```



# Else in For Loop

- The else keyword in a for loop specifies a block of code to be executed when the loop is finished:

```
for x in range(6):  
    print(x)  
else:  
    print("Finally finished!")
```

```
0  
1  
2  
3  
4  
5  
Finally finished!  
> |
```



# If the loop breaks, the else block is not executed.

```
for x in range(6):  
    if x == 3: break  
    print(x)  
else:  
    print("Finally finished!")
```

```
0  
1  
2  
> |
```



# Pass statement

- When the pass statement is executed, nothing happens, but you avoid getting an error when empty code is not allowed.
- Empty code is not allowed in loops, function definitions, class definitions, or in if statements.

for x in [0, 1, 2]:

pass

- # having an empty for loop like this, would raise an error without the pass statement



# Nested Loop in Python

A nested loop is a loop inside the body of the outer loop. The inner or outer loop can be any type, such as a while loop or for loop.

The outer loop can contain more than one inner loop. There is no limitation on the chaining of loops

In the nested loop, the number of iterations will be equal to the number of iterations in the outer loop multiplied by the iterations in the inner loop.

In each iteration of the outer loop inner loop execute all its iteration. For each iteration of an outer loop the inner loop re-start and completes its execution before the outer loop can continue to its next iteration.



# Syntax

```
# outer for loop
for element in sequence
    # inner for loop
    for element in sequence:
        body of inner for loop
    body of outer for loop
```

# Flow chart

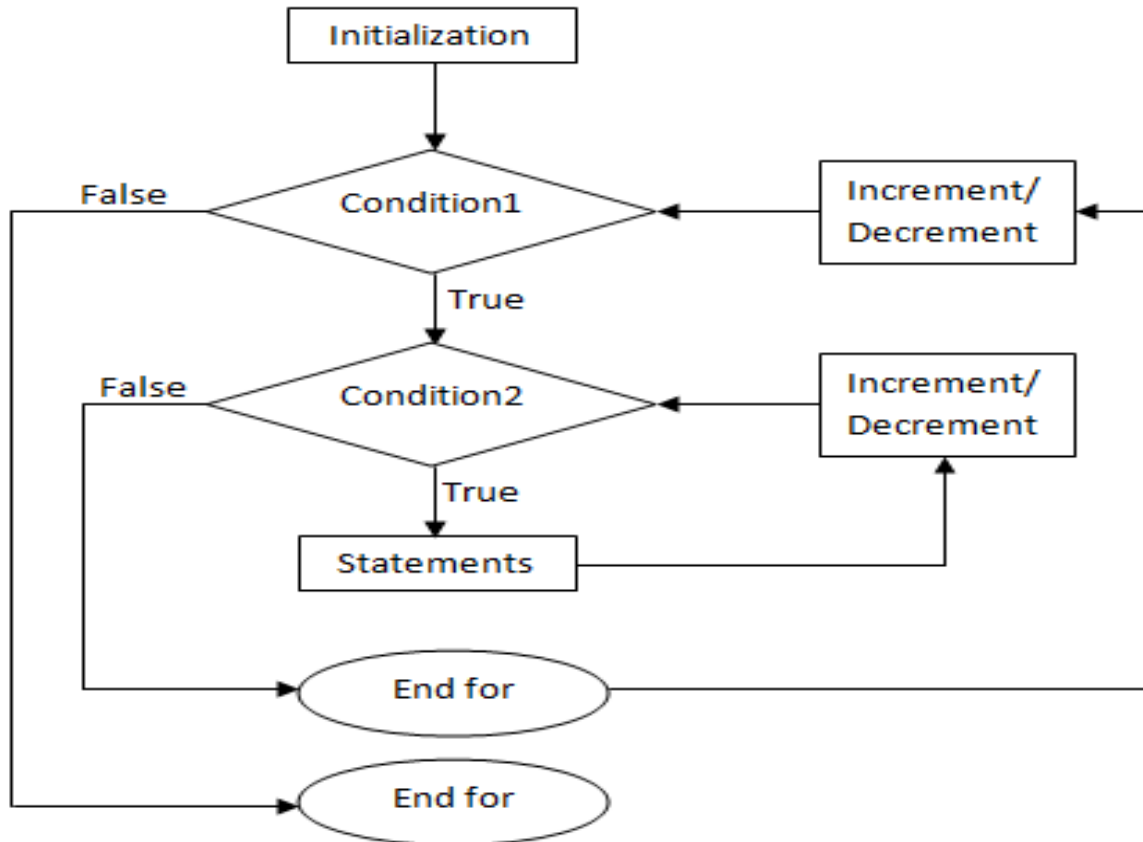


Fig: Flowchart for nested for loop





# Example

```
# outer loop
for i in range(1, 11):
    # nested loop
    # to iterate from 1 to 10
    for j in range(1, 11):
        # print multiplication
        print(i * j, end=' ')
    print()
```

```
1 2 3 4 5 6 7 8 9 10
2 4 6 8 10 12 14 16 18 20
3 6 9 12 15 18 21 24 27 30
4 8 12 16 20 24 28 32 36 40
5 10 15 20 25 30 35 40 45 50
6 12 18 24 30 36 42 48 54 60
7 14 21 28 35 42 49 56 63 70
8 16 24 32 40 48 56 64 72 80
9 18 27 36 45 54 63 72 81 90
10 20 30 40 50 60 70 80 90 100
> |
```



# Example

```
rows = 5
# outer loop
for i in range(1, rows + 1):
    # inner loop
    for j in range(1, i + 1):
        print("*", end=" ")
    print("")
```

```
*
* *
* * *
* * * *
* * * * *
> |
```



## Python Nested Loop

A Loop inside a loop is known as a nested loop.

In the nested loop, the number of iterations will be equal to the number of iterations in the outer loop multiplied by the iterations in the inner loop.

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

Diagram illustrating the structure of a nested loop:

- The outer loop is represented by a green bracket on the left, labeled "Outer loop".
- The inner loop is represented by a red bracket on the left, labeled "Inner loop".
- The body of the inner loop is represented by a purple bracket on the right, labeled "Body of inner for loop".
- The body of the outer loop is represented by a purple bracket on the right, labeled "Body of Outer loop".

PYnative.com



# Thank you