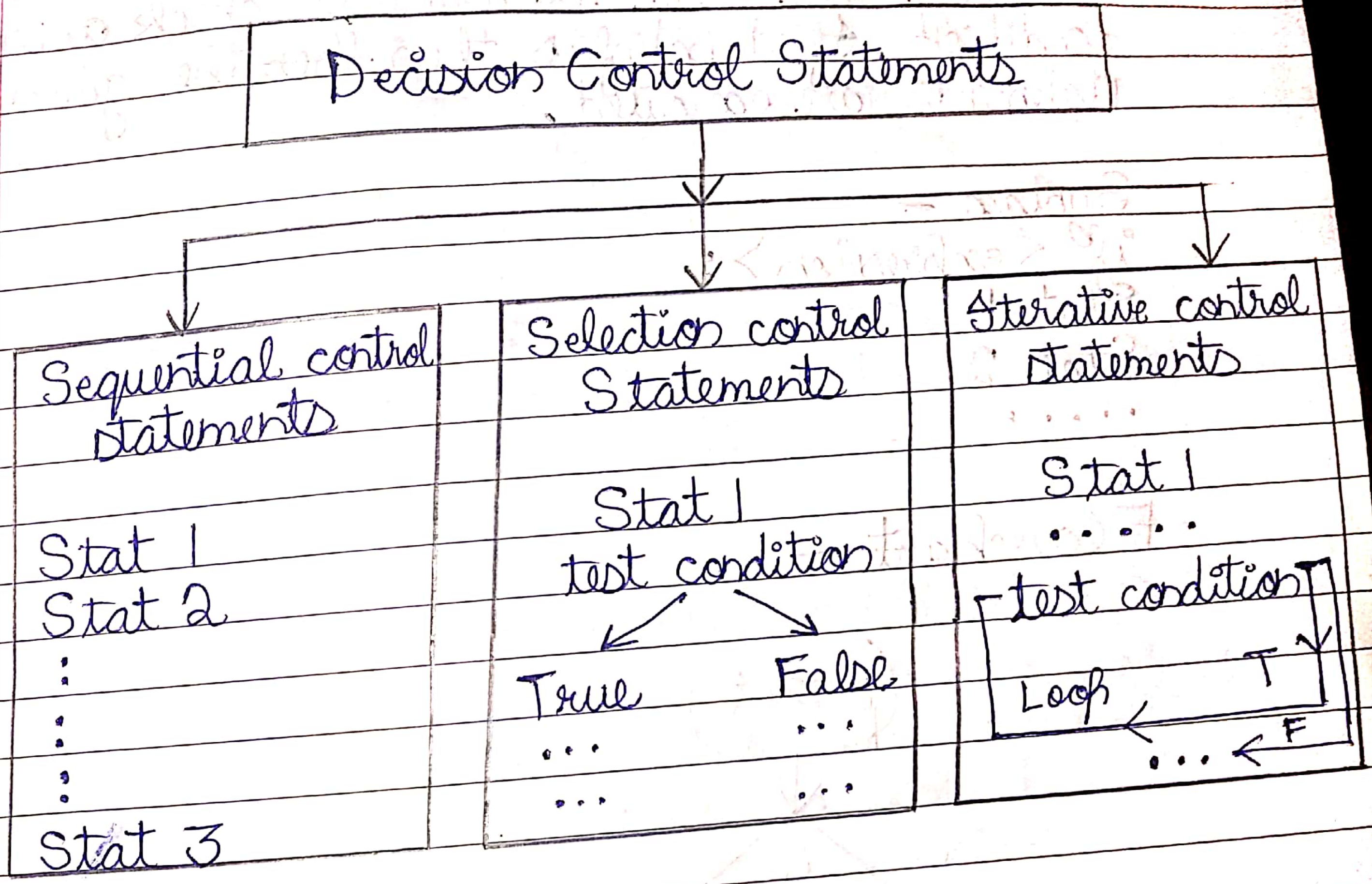
.09.19

Unit - II

Decision Control Statements

Introduction –


Decision Control Statements
Sequential control statements
Stat 1
Stat 2
:
:
:
Stat 3
Selection control Statements
Stat 1
test condition
True
False
...
...
...
...
Iterative control statements
Stat 1
....
test condition
Loop
T
F
...

1) (i) Sequential Control Statement –
Python program is executed sequentially from the first line of the program to its last line.
(ii) i.e. the second statement is executed after the first and so on. This method is known as sequential control flow.

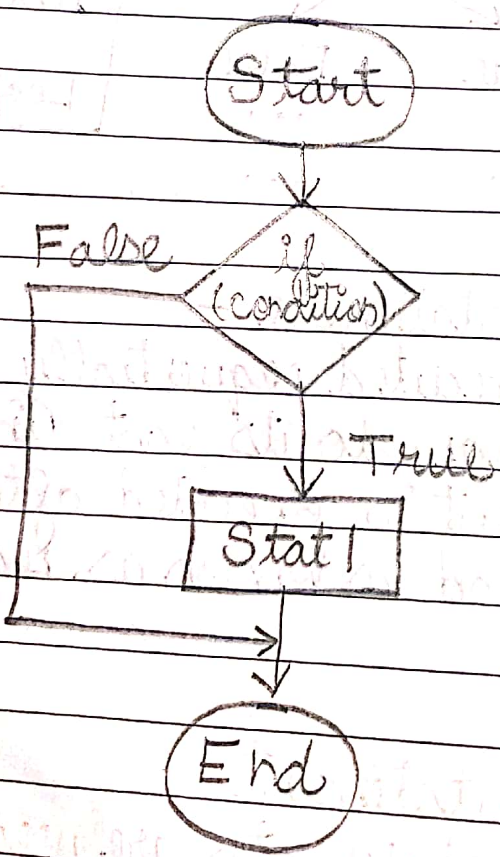
2) Selection Control Statement –
The decision making statements are used to check different conditions depending upon which the flow of control can be decided / made.

2.1 if statement -
If is a decision making statement. It is generally used when we want to check a single condition. If it satisfies then ~~then~~ the given statements are executed.

Syntax -

```
if <expression>:
Stat 1
Stat 2
.....
```

Flowchart -



03.09.19

example -

```
n = 110
if (n >100) :
print ("Number is greater than 100")
```

O/P Number is greater than 100

2.2 if - else statement -

the if - else statement is used to tell the compiler what to do in both the situations :-

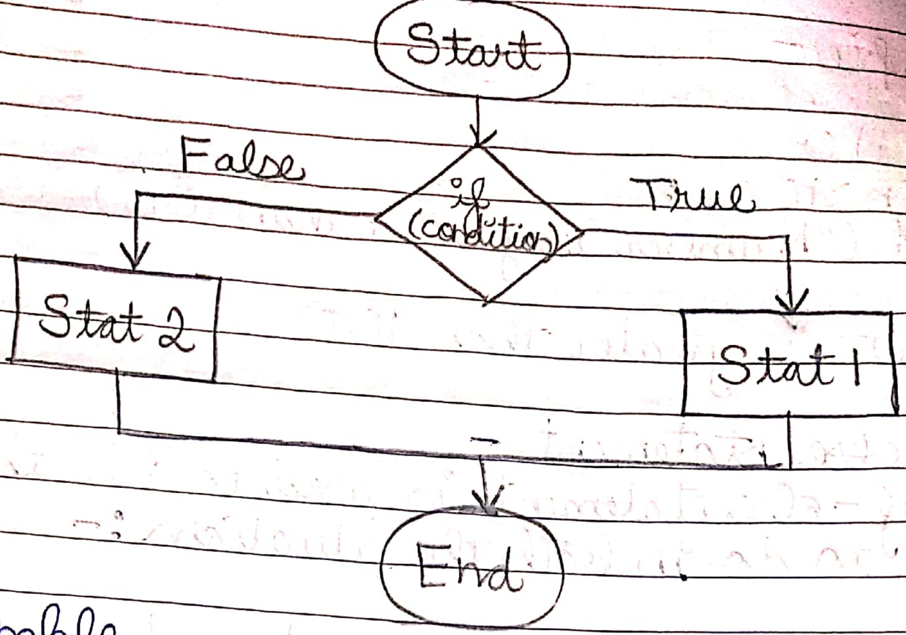
if the given condition is satisfied,
if the given condition does not satisfied.

if the given condition is true then the if block is executed otherwise else block is executed.

Syntax -

```
if expression :
Stat 1
else :
Stat 2
```

Flowchart -



example -

```
a = int (input ("Enter number"))
if ( a % 2 == 0):
print ("Number is even")
else :
print ("Number is odd")
```

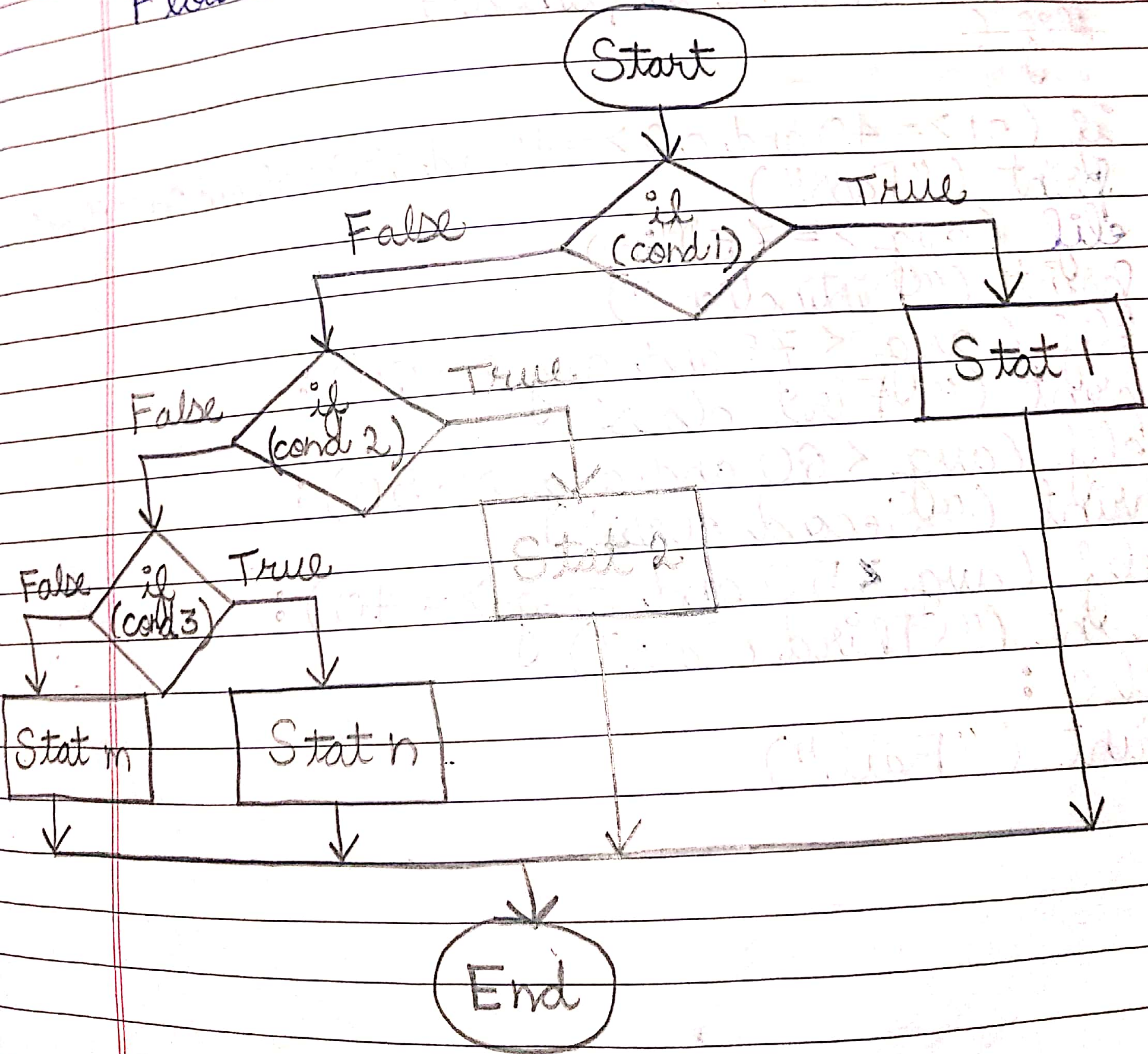
2.3 if - elif - else statement -:

the if - elif - else statement is used to test set of conditions in a sequence. It is also considered as multi way decision making statemen If any of the given condition is satisfied then th related statements are excecuted and the control exits from the elif ladder. That means further conditions are not going to be check.

Syntax –

```
if expression 1:
Stat 1
elif expression 2:
Stat 2
elif expression 3:
Stat 3
else:
Stat 4
```

Flowchart –



Ans

```
c1 = int (input ("Enter marks of course 1"))
c2 = int (input ("Enter marks of course 2"))
c3 = int (input ("Enter marks of course 3"))
c4 = int (input ("Enter marks of course 4"))
c5 = int (input ("Enter marks of course 5"))
avg = c1+c2+c3+c4+c5 * 100
              500
totmarks = c1+c2+c3+c4+c5
~~if (totmarks < 200):~~
~~print ("Student is failed")~~
~~else :~~
~~print ("Student is passed")~~
~~elif (~~

if (c1>= 40 and c2>=40 and c3>=40 and c4>=40 a
print ("Pass")
elif (avg >= 75):
print ("Distinction")
elif ( avg < 75 and avg >= 60):
print (" First class")
elif (avg < 60 and avg >= 50):
print ("Second class")
elif (avg < 50 and avg >= 40):
print ("Third class")
else :
print ("Fail")
```

29 March
24-April)

04.09.19

2.4 Nested if-else statement –

writing if statement inside another if is known as nested if. while checking no. of conditions we may need to nest if statement inside another if statement.

Syntax –

```
if exp1:
    Stat
    if exp 2:
        Stat
    else:
        Stat
else:
    Stat
```

18 12 14

o):

eg.

```
n1 = int (input ("Enter n1"))
n2 = int (input ("Enter n2"))
n3 = int (input ("Enter n3"))
if (n1 > n2):
    if(n1 > n3):
        print ("Max is n1", n1)
    else:
        print ("Max is n3", n3)
elif (n2 > n3):
    print ("Max is n2", n2)
else:
    print ("Max is n3", n3)
```

09.09.19

3) Basic loop structures / Iterative statements –

1) Basic loop structures or iterative statements are used to execute specific task repetedly in our program.

2) Rather than writing the code again and again we can use the concept of loop.

Looping statements in Python are :–

1) while loop
2) for loop

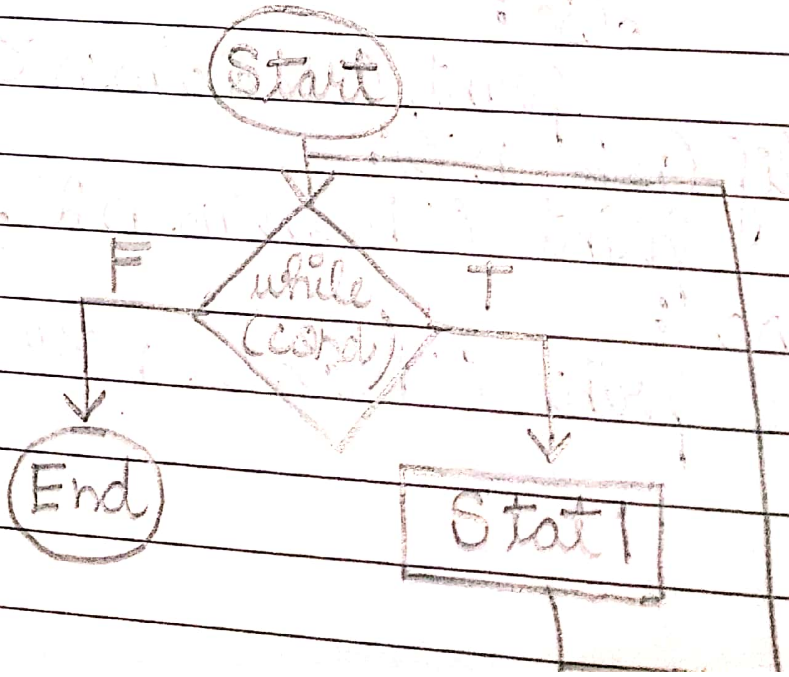
3.1) while loop –

while loop is used when we want to execute the set of statements repetedly until the given condition is satisfied. Once the condition becomes false the control exits the loop.

Syntax –

```
while expression :
 statements
```

Flowchart –



09.09.19

eg.
```
i=1
while (i<=10):
print ("Welcome to Python Prog")
  i=i+1
```

eg. Write a program to print 1 to 10 number in while loop.

=> ~~i=1~~
~~while (i<=10):~~
print (i)
i=i+1

```
i=1
while (i<=10):
    print(i)
  i=i+1;
```

eg. WAP to print factorial of a number.

=>
```
number = int (input ("Enter a number"))
f=1
while(number > 0):
f=f * number
number = number -1
print ("Factorial is:", f)
```

inside while →

O/P Factorial is: 120

- Infinite loop –
A loop becomes infinite when a condition is always true and never becomes false.
or
The results in a loop that never end are called as infinite loop.

09.09.19

eg.

```
n = 1
while (n == 1):
    num = int (input ("Enter a number"))
    print ("Square is :", num * num)
print ("Good Bye")
```

O/P

```
Enter a number 5
Square is : 25
Enter a number 3
Square is : 9
:
:
∞
```

* Else statement used with while loop -

(i) In Python it is possible to use else statement with loop statement.

(ii) If the else statement is used with a while lo the else statement is executed when the condi becomes false.

eg.

```
count = 0
while (count < 10):
    print ("B division")
    count = count + 1
else:
    print ("Message printed 10 times")
```

09.09.19

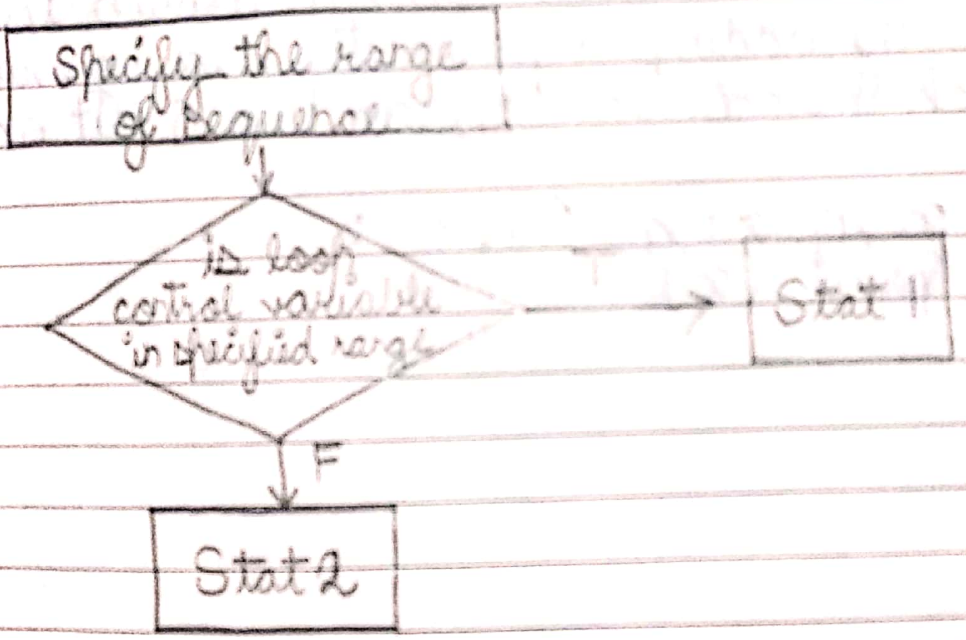
3.2) for loop –

The for loop in Python is used to iterate the statements or a part of the program several times. It is frequently used to traverse the data structures like list, tuple or dictionary.

Syntax –

```
for iterating_var in sequence:
statement(s)
```

Flowchart –



Specify the range of Sequence
is loop control variable in specified range
Stat 1
F
Stat 2

The range () –

i) Python provides range() to loop through a set of code for a specified no. of times.

ii) This function returns a sequence of numbers starting from 0 by default, and increaments by 1 (by default), and ends at a specified number.

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

O/P
```
0
1
2
3
4
5
```

By default the range() sets the starting value
0, however it is allowed to mention the star
value by adding a parameter, for eg. range(
which means values from 1 to 6 but exclud

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

O/P
```
2
3
4
5
```

In the range() by default value is increment
by 1, however it is allowed to specify ow
customised increment value by adding a thir
parameter.

09.09.19

eg.
```
for x in range (10, 21, 2):
    print (x)
```

O/P
```
10
12
14
16
18
20
```

else in for loop –

In for loop the else keyword is used to specify a block of code to be execated after completion of loop.

eg.
```
for x in range (11):
    print (x)
else:
    print ("Loop finished")
```

O/P
```
0
:
:
10
Loop finished
```

Selecting appropriate loop –

1) It is recommended to use for loop if you know exactly the no. of elements to be iterated over.

2) In contrast a while loop is better when you have a boolean expression to evaluate and not a list of elements to loop over.

09.09.19

- break statement –

1> The break statement is used to exit from a loop for a particular condition.
2> The break statement breaks the loop one by one.
3> In other words, it can be said that the break helps to abort the current execution of the respective program and sets the control to the next line after the loop.

Syntax –

```
#loop statements
break
```

eg.

```
for i in range(1,10):
    if i == 5:
        break  →  if i is equal to 5
                  then exit from loop
    print(i)
```

O/P

```
1
2
3
4
```

When the value of variable i becomes 5 the control comes out of the loop because of the break statement.

09.09.19

- continue statement -

1) The continue statement sets the control at the beginning of loop for a particular condition.

2) That means for the given condition the statements below the coitue statement are skipped and start with the next iteration.

3) It is mainly used for a particular condition inside the loop so that we can skip some specific code for that condition.

Syntax -

```
#loop statements
continue
# code to be skipped
```

eg.

```
for i in range(1,10):
    if i==5:
        continue
    print (i)
```

→ when i=5 the print statement will not execute and control sets at the beginning of loop

O/P

```
1
2
3
4
6
7
8
9
```

09.09.19

Continue doesn't mean continue the next code. The continue statement sets the control at the beginning of loop and the statements below the continue do not execute for the particular conditio In this eg. when the value of i becomes 5 the print statement which is below continue will not execute.

Difference between break and continue :-

Parameter	Break statement	Continue statement
Working	Break statement terminates the execution for given condition.	Continue statement doesn terminate the execution rather it skips the statem which appears after it for a particular condition
Program Control	When break statement appears the control goes out of loop.	When continue statement appears the control goes at the beginning of loop.

- pass statement –
  In Python pass statement acts as a place holder and is generally used when we don't want to write any code but a statement is still needed to make a code syntactically correct.

16.09.19

eg.
```
for num in range (1, 20):
        if num % 2 == 0:
              pass
        else:
              print (num)
```

For excample we want to print odd numbers upt
20 and do some operation on even numbers but
in future.
In such situation we cannot leave the specific part
empty as this would raise error, since it is
syntactically incorrect.
Here we can use pass statement which does
nothing but makes the code syntactically correct.

Pass statement vs Comment

A comment is not a place holder and it is
totally ignored by the interpreter whereas pass
is not ignored by interpreter ~~wh~~, it indicates
the interpreter to do nothing.

- Other datatypes —

1) Tuples
In Python a tuple is considered as a sequence
of immutable objects.
Creating a tuple is very easy as of just
setting different comma separated values.
It is also possible to put these comma separated
values between parenthesis also.

6.09.19

eg. tuple1=("Kunal", "Ishita", 2008, 2013)

tuple 2 = (11, 22, 33, 44, 55)

tuple 3 = ("a", "b", "c", "d")

Empty tuple =>

tup 1 = ( ) =>O/P

The empty tuple is written as two parenthes
containing nothing.

To write a tuple which is having only one val
we have to include a comma after d element
eg. tup 1 = (50,)

Tuple indices start at 0, that means the firs
element has index no. 0

eg. tup 1 = (10, 20, 30, 40, 50)

tup 1	10	20	30	40	50
	[0]	[1]	[2]	[3]	[4]

Accessing values in tuples =>

For accessing values in tuple we have to use the square bracket for slicing along with the index or indices to get the respective value present at particular index.

5.09.19

eg.
```
        0  1  2  3  4  5  6  7  8  9
tup1 = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
print ("Tup[3:6]=", tup1[3:6])
print ("Tup[:4]=", tup1[:4])
print ("Tup[4:]=", tup1[4:])
print ("Tup[:]=", tup1[:])
```

O/P ⇒

~~4 5 6~~

~~1 2 3 4~~

~~56 7 8 9 10~~

```
Tup[3:6] = (4, 5, 6)
Tup[:4]  = (1, 2, 3, 4)
Tup[4:]  = (5, 6, 7, 8, 9, 10)
Tup[:]   = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
```

NOTE Tuple [↓ : ↓]   P

from — by default [0]

to — ie. n-1 if nothing then upto last

```
Tup1 = (1, 2, 3, 4, 5)
Tup2 = (6, 7, 8, 9, 10)
Tup3 = Tup1 + Tup2
print (Tup3)
```

O/P ⇒ (1, 2, 3, 4, 5, 6, 7, 8, 9, 10)

Loop through a tuple =>

You can traverse through a tuple by using a loop.

★★ eg.
```
tup1 = (1, 2, 3, 4)
for x in tup1:
    print (x)
```

O/P => 1 ~~1, 2, 3, 4~~
2
3
4

Updating / Adding new element =>

Tuple are immutable, it indicates that it is not possible to change the values of tuple or add any new item to it.

eg.
```
tup1 = (1, 2, 3, 4, 5)
tup1[2] = 7
print (tup1)
```

O/P => Error

It will raise an error. New element can't be added.

Delete Tuple elements –

(i) Deleting individual elements from tuple is not allowed.

(ii) Python provides del statement to delete entire tuple.

```
eg. tup1 = (1, 2, 3, 4, 5)
    del tup1[3]
    print (tup1)
```

O/P => Error

Tuple object doesn't support item deletion

```
eg. tup1 = (1, 2, 3, 4, 5)
    del tup1
    print (tup1)
```

O/P => Error
Name tup1 is not defined

Check if item exists –

The in keyword is used to check whether a specified item is present in a tuple or not.

```
eg. tup1 = (1, 2, 3, 4, 5)
    if 3 in tup1:
            print ("3 is present")
    else:
            print ("3 is absent")
```

O/P => 3 is present

- Basic Tuple Operations -

1) Length-
Python provides len() to count total no. of items in a tuple.

```
eg. tup1 = (1,2,3,4,5)
     len(tup1)
```

O/P => 5

```
import keyword
print (keyword.kwlist)
```

2) Concatenation-
This operator contatenates the multiple tuples.

```
eg. tup1 = (1,2)
    tup2 = (3,4)
    tup3 = tup2+tup1
    print (tup3)
```

O/P => (3,4,1,2)

3) Repetition -

```
eg. tup1 = (1,2,3)
    tup1 * 3
```

17.09.19

O/P => (1, 2, 3, 1, 2, 3, 1, 2, 3)

The repetition operator is used to repeat the tuple elements multiple times.

4) Comparison -

5) Maximum -

eg. tup1 = (1, 2, 3, 4, 5, 99, 101)
print (max(tup1))

O/P => 101

6) Minimum -

eg. tup 1 = (1, 2, 3, 4, 5, 99, 101)
print (min (tup1))

O/P => 1

17.09.19

6) Conversion -

eg. tuple ("Hello")

O/P => ('H','e','l','l','o')

Tuple assignment -
It allows a tuple of variables on the left side of the assignment operator to be assigned values from a tuple on the right side of the assignment operator.

```
eg:(val1, val2, val3) = (1, 2, 3)
   print (val1, val2, val3)

   O/P => 1 2 3
```

-
  ```
  tup1 = (100, 200, 300)
  (val1, val2, val3) = tup1
  print (tup1)

  O/P =>(100 , 200, 300)
  ```

-
  ```
  (val1, val2, val3)

  O/P =>(100, 200, 300)
  ```

-
  ```
  (val1, val2, val3) = (2+4, 5/3+4, 9%6)
  print (val1, val2, val3)
  ```

O/P=>6 5.666666666666667 3

18.09.19

- (val1, val2, val3) = (1, 2)

O/P => ValueError: not enough values to unpack (expected 3, got 2)

- (val 1, val 2) = (1, 2, 3)

O/P => ValueError: too many values to unpack (expected 2)

Nested Tuple
Python allows you to define a tuple inside another tuple. This is a nested tuple.

eg.
```
toppers = (("Arav", "BSc", 92.0),
           ("Chaitanya", "BCA", 99.0), ("Dhruv", "BTech", 97))
    for i in toppers:
        print (i)
```

O/P =>
```
('Arav', 'BSc', 92.0)
('Chaitanya', 'BCA', 99.0)
('Dhruv', 'BTech', 97)
```

```
topper = ("Janvi", [94, 95, 96, 97])
print ("Class topper :", topper[0])
print ("Highest score in 4 subjects :", topper[1:])
```

O/P =>
```
Class topper : Janvi
Highest score in 4 subjects : ([94, 95, 96, 97],)
```

Index Method => to find index

The index of an element in the tuple can be obtained by using the index ().

Syntax -

```
tup. index (obj)
```

eg.
```
tup = (1, 2, 3, 4, 5, 6, 7, 8)
print (tup. index (4))
```

O/P => 3

```
tup = (1, 2, 3, 4, 5, 6, 7, 8)
print (tup. index (4-1))
```

O/P => 2

```
Students = ("Bhavya", "Era", "Falguni", "Huma")
index = Students. index ("Falguni")
print ("Falguni is present on location :", index)
index = Students. index ("Isha")
print (f" Isha is present at location:"; index)
```

O/P =>

```
Falguni is present on location : 2
ValueError : tuple . index (x): x not in tuple
```

18.09.19

• Escape sequences in Python	Purpose
1) \\	Prints backslash
2) \'	Prints single quote
3) \"	Prints double quote
4) \n	Prints new line character
5) \t	Prints tab
6) \a	Rings bell
7) \f	Prints form feed character
8) \o	Prints octal value
9) \x	Prints hex value

Count ( )

The count( ) is used to return the no. of elements with a specific value in a tuple.

```
eg. tup = "abcdxcxxcefxcxghijx"
    print (tup.count('x'))
```

O/P => 6

19.09.19

- List

i) A list is a collection of different Python objects such as integers, strings etc.

ii) It is similar to array concept of other programming languages.

iii) List is represented by [ ].

iv) List is mutable that means its content can be modified or updated unlike tuples which are considered as immutable.

v) Creating a list is very easy as of just ~~second~~ setting different, separated values in square brackets.

eg. lista = [1, 2, 3, 4, 5]
print (lista)

O/P => [1, 2, 3, 4, 5]

eg. lista = [1, 'a', "abcd"]
print (lista)

O/P => [1, 'a', 'abcd']

Accessing values in list –

List can also be sliced and contatenated. To access value in list, [ ] are used to slice alongwith index or indices to get value stored at that index.

9.09.19

eg.
```
numlist =[1,2,3,4,5,6,7,8,9,10]
print (numlist)
print (numlist[0])
print (numlist[2 : 5])
print (numlist[:: 2])
print (numlist[1:: 3])
```

```
O/P => [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
       1
       [3, 4, 5]
       [1, 3, 5, 7, 9]
       [2, 5, 8]
```

Updating values in list -

(i) Once created one or more elements of a list can be easily updated by giving the slice on the LHS of the assignment operator.

(ii) You can also appended new values in the list and remove exciting values from the list using the append () method and del respectively.

eg.
```
numlist =[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
print (numlist)
numlist [5] = 100
print (numlist)
numlist.append (200)
print (numlist)
del numlist [3]
print (numlist)
```

19.09.19

O/P =>
```
[1,2,3,4,5,6,7,8,9,10]
[1,2,3,4,5,100,7,8,9,10]
[1,2,3,4,5,100,7,8,9,10,200]
[1,2,3,5,100,7,8,9,10,200]
```

eg.
```
numlist = [1,2,3,4,5,6,7,8,9,10]
del numlist [2:4]
print (numlist)
```

O/P => `[1,2,5,6,7,8,9,10]`

eg.
```
numlist = [1,2,3,4,5,6,7,8,9,10]
del numlist [:]
print (numlist)
```

O/P => `[ ]`

eg.
```
numlist = [1,2,3,4,5,6,7,8,9,10]
del numlist
print (numlist)
```

O/P => NameError : name 'numlist' is not defined

eg.
```
numlist = [1,9,11,13,15]
print (numlist)
numlist [2] = [3,5,7]
print (numlist)
```

O/P =>
```
[1,9,11,13,15]
[1,9,[3,5,7],13,15]
```

Nested List—

It means a list within another list.

```
                0    1      2        3       4  5
eg. list1 = [1, 'a', "abc", [2,3,4,5], 8, 9]
       i = 0
       while i<(len(list1)):
           print("List1[", i,"] = ", list1[i])
           i = i+1
```

```
O/P => List1[0] = 1
       List1[1] = a
       List[2] = abc
       List[3] = [2,3,4,5]
       List[4] = 8
       List[5] = 9
```

Cloning List —

If you want to modify a list and also keep a copy of the original list then you should create a separate copy of the list. This process is called cloning. The slice operation is used to clone a list.

```
eg. list1 = [1,2,3,4,5,6,7,8,9,10]
       list2 = list1
       print (list1)
       print (list2)
       list3 = list1[2:6]
       print (list3)
```

20.09.19

Nested List

O/P => [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
[3, 4, 5, 6]

Basic List Operations

1) Length =

len (1, 2, 3, 4)

O/P => 4

2) Concatenation =

[1, 2, 3, 4] + [5, 6, 7, 8]

O/P => [1, 2, 3, 4, 5, 6, 7, 8]

3) Repetition =

"Hello" * 2

'Hello Hello' <= O/P

4) in =

'a' in ['a', 'e', 'i', 'o', 'u']

O/P => True

21.09.19

5) not in =

3 not in [0,2,4,6,8]

O/P → True

6) max =

```
list 1 = [6,3,7,0,1,2,4,9]
print (max (list1))
```

O/P → 9

7) min =

```
list 1 = [6,3,7,0,1,2,4,9]
print (min (list 1))
```

O/P → 0

8) sum =

```
list 1 = [1,2,3,4,5,6,7,8,9,10]
print (sum (list 1))
```

O/P ⇒ 55

9) all =
returns True if all elements of the list are true or if the list is empty.

eg.1 list1 = [1,2,3,4]

~~O/P~~ print(all(list1))

O/P => True

eg.2 list1 = [0,1,2,3,4]

print(all(list1))

O/P => False

eg.3 list1 = [0,-1,-2,-3,-4]

print(all(list1))

O/P => False

eg.4 list1 = []

O/P => True

10) list =

list1 = list ["HELLO")

print(list1)

O/P => ['H', 'E', 'L', 'L', 'O']

**

11) sorted=

list1 = [3,4,1,2,7,8]

list2 = sorted [list1)

print(list2)

21.09.19

** O/P => [1,2,3,4,7,8]

```
              0        1        2       3
list1 =["Hello","World","Good","Morning"]
print (list1[2])
print (list1[-3])
print (list1[1:])
```

```
O/P =>  Good
        World
        ['World','Good','Morning']
```

● List() methods =>

1> append = always on last

```
list1 = [1,2,3,4,5]
list1.append(10)
print (list1)
```

O/P => [1,2,3,4,5,10]

2> count = count no. of occurances of particular item

```
list1=[1,2,3,4,5,1]
print (list1.count(1))
```

O/P => 2

21.09.19

3) index = print index which is lowest

```
list1 = [6,3,0,3,7,6,0]
print(list1.index(7))
```

O/P => 4

**

```
list1 = [6,3,0,3,7,6,0]
print(list1.index(3))
```

O/P => 1

4) insert = insert (index, no. to be inserted.

```
         0 1 2 3 4 5 6
list1 = [6,3,0,3,7,6,0]
list1.insert(3,100)
print(list1)
```

```
         0 1 2  3  4 5 6 7
O/P => [6,3,0,100,3,7,6,0]
```

5) remove = remove the element

```
list1 = [6,3,0,3,7,6,0]
list1.remove(0)
print(list1)
```

O/P => [6,3,3,7,6,0]

21.09.19

6) reverse =

```
list1 = [1, 2, 3, 4, 5]
list1.reverse()
print(list1)
```

O/P => [5, 4, 3, 2, 1]

7) sort = same as sorted but ~~first have to~~ so directly

```
list1 = [6, 3, 4, 1, 5, 9, 2]
list1.sort()
print(list1)
```

O/P => [1, 2, 3, 4, 5, 6, 9]

8) extend = function

```
list1 = [1, 2, 3, 4, 5]
list2 = [6, 7, 8, 9, 10]
list1.extend(list2)
print(list1)
```

O/P => [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

23.09.19

- Dictionary -

It is a data structure in which we store values as a pair of key and value. Each key is separated from its value by a colo (:) and consecutive items are separated by commas. The items in a dictionary are enclosed in curly brackets.

Syntax -

```
dictionary_name = {key1 : val1, key2 : val2,
                   key3 : val3}
```

Creating a dictionary -

```
eg. Dict = { }
    print (Dict)

    O/P => { }
```

```
eg. dict1 = {'Roll no' : '1', 'Name' : 'Arav',
                  'Course' : 'BTech'}
  print (dict1)

  O/P => {'Rollno' : '1', 'Name' : 'Arav' : 'Course' : 'BTech
```

To create a dictionary with one or more key : value pairs you can also use the dict(). The dict() creates a dictionary directly from a sequence of key : value pairs.

23.09.19

** eg.
```
print(dict([('Rollno','1'),('Name','Arav'),
            ('Course','BTech')]))
```

```
O/P => {'Rollno':'1','Name':'Arav':'Course':'BTech'}
```

Accessing Values -
To access values in a dictionary square brack
([ ]) are used along with the key to obtain its value.

eg.
```
dict1={'Rollno':'15','Name':'Arav','Course':'BTech'}
 print("Dict[Rollno]=", dict1['Rollno'])
  print("Dict[Name]=", dict1['Name'])
  print("Dict[Course]=", dict1['Course'])
```

```
O/P =>  Dict[Rollno] = 15
        Dict[Name] = Arav
        Dict[Course] = BTech
```

eg.
```
dict={}
 print("Dict[Marks'=", dict['Marks'])
```

```
O/P => KeyError: 'Marks'
```

Adding and modifying an item in a dictionary

dictionary_variable[key] = val

23.09.19

eg.
```
dict1 = {'Rollno':'1', 'Name':'Arav', 'Course':'BTech
print ("Dict[Rollno] =", dict1['Rollno'])
print ("Dict[Name] =", dict1['Name'])
print ("Dict[Course] =", dict1['Course'])
dict1['Marks'] = 95
~~print (dict1)~~ print ("Dict[Marks] =", dict1['Mar
```

continuation

```
O/P => Dict[Rollno] = 1
       Dict[Name] = Arav
       Dict[Course] = BTech
       Dict[Marks] = 95
```

eg.
```
dict1['Course'] = 'BCA'
print ("Dict[Course] =", dict1['Course'])
```

```
O/P => Dict[Rollno] = 1
       Dict[Name] = Arav
       Dict[Course] = ~~BCA~~ BTech
       Dict[Marks] = 95
       Dict[Course] = BCA
```

Deleting Items in Dictionary -

eg.
```
dict1 = {'Rollno':'1', 'Name':'Apurwa', 'Course':'MTec
print ("Dict[Rollno] =", dict1['Rollno'])
print ("Dict[Name] =", dict1['Name'])
print ("Dict[Course] =", dict1['Course'])
del dict1['Course']
print (dict1)
~~pru~~ dict1.clear()
```

24.09.19

```
print(dict1)
del dict1
print(dict1)
```

O/P =>
```
Dict[Rollno] = 1
Dict[Name] = Apurva
Dict[Course] = MTech
{'Rollno': '1', 'Name': 'Apurva'}
{ }
NameError : name 'dict1' is not defined
```

*

NOTE: Keys must have unique values. Not even a single key can be duplicated in a dictionary. If you try to add a duplicate key, the last assignment is retained.

eg.
```
dict1 = {'Rollno': '1', 'Name': 'Apurva', 'Course':
         'MTech', 'Name': 'Kriti'}
print ("Dict[Rollno] =", dict1['Rollno'])
print ("Dict[Name] =", dict1['Name'])
print ("Dict[Course] =", dict1['Course'])
```

O/P ->
```
Dict[Rollno] = 1
Dict[Name] = Kriti
Dict[Course] = MTech
```

24.09.19

Program to check single key in dictionary :-

```
eg. dict = {'Rollno':'1','Name':'Apurva','Course':'MT
     if 'Course' in dict1:
         print(dict1['Course'])
```

O/P => MTech

Sorting items in a dictionary :-

The keys() of dictionary returns a list of all the keys used in the dictionary in an arbitrary order.
The sorted() is used to sort the keys.

```
eg. dict1 = {'Rollno':'1','Name':'Apurva','Course':'M
     print(sorted(dict1.keys()))
```

O/P => ['Course', 'Name', 'Rollno']

Looping over a dictionary :-

You can loop over a dictionary to access only values eg, only keys and both using a for loo

```
eg. dict1 = {'Rollno':'1','Name':'Apurva','Course':'MTec
     print("KEYS:", end='')
     for key in dict1:
         print(key, end='')
     print("\nVALUES:", end='')
```

24.09.19

```
for val in dict1.values():
    print(val, end=' ')
print("\n DICTIONARY:", end=' ')
for key, val in dict1.items():
    print(key, val, "\t", end='')
```

```
O/P => KEYS : Rollno   Name   Course
       VALUES: 1   Apurva   MTech
       DICTIONARY: Rollno 1   Name Apurva   Course MTech
```

Nested dictionary :

```
eg. Students={'Shiv': {'CS':90, 'DS':89, 'CSA':92},
              'Sadhvi': {'CS':91, 'DS':87, 'CSA':94},
              'Krish':{'CS':93, 'DS':92, 'CSA':88}}
    for key, val in Students.items():
        print(key, val)
```

```
O/P => Shiv {'CS':90, 'DS':89, 'CSA':92}
       Sadhvi {'CS':91, 'DS':87, 'CSA':94}
       Krish {'CS':93, 'DS':92, 'CSA':88}
```

Built in dictionary functions and methods :-

1) length – len()
2) string – str()
3) clear – clear()
4) copy – copy()
5) fromkeys – fromkeys()

24.09.19

** Fromkeys –
Creates a new dictionary with keys from sequences and values set to val.

~~eg. subjects = ['CSA',~~

6) get – get()
7) has – has() – in
8) items – items()
9) keys – keys()
10) setdefault()
11) update() – add or concatenate
12) values()
13) in and not in