Conditional statements :

- These are fundamental concepts of programming language.
- Till now we have learned statements like Assignment and Arithmetic statements and these statements are called normal statements. Python executes them if syntax is correct.

* conditional statements are the statements that will only be executed based on the conditions.

For conditional statements in Python

Three techniques
- if conditional statement
- if-else conditional statement
- if-elif-else conditional statement.

If Conditional statement

If conditional statement is the simplest way to write the conditional statement.

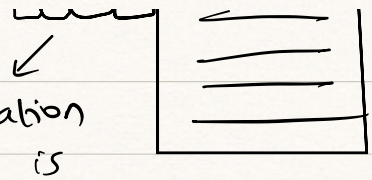
Syntax(Rule)

1) 'if' keyword is used here.

```
if _ conditions :
    ....
```

→ we can write n no.of conditions

→ conditions have to end with semicolon.



2) After if we have to leave space before conditions.

3) Conditions can be any number of conditions using logical, Arithmetic or other operators.

4) After the conditions end we need to use colon ':' to denote the conditions have ended.

5) Next line should start with indentations which represents a block of code.

6) Indentation can be single, double, triple or four white spaces. Default is four white spaces.

7) Indentation will have to be same for all the statements in that block of code otherwise we get indentation error.

8) The statements execute if the condition is True, if it is False it will not do anything.

ex:-

```
a=10
if a>5:
    x=1
    y=1
    z=x+y
    print(z)
```

↳ 2

```
a=10
if a>5 and a%2==0:
    x=1
    y=1
    z=x+y
    print(z)
```

1st inner loop

2nd in loo

↳ 2

Since statements do not execute if the condition is False we have other techniques.

if-else conditional statement:

If the condition is true it will execute a set of statements else it will execute a different set of statements.

Syntax (Rules)

```
if_conditions:
    ____
    ____
    ____
    ____
else:                    ← no conditions
    ____
    ____
    ____
    ____
```

↙ Indentation

1) Here 'if' and 'else keywords are used.
2) All the syntax of the 'if' condition is followed here also.
3) Else should always come after if.
4) Indentation in all the else statements should be same. It need not be same as if statement indentation.

5) If if block is executed then else block is not executed and vice versa.

6) else block should not have any conditions.

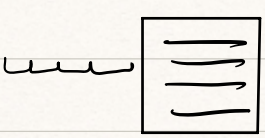
* Till now we were executing single set of statements if the condition is True and another set if the condition is False.

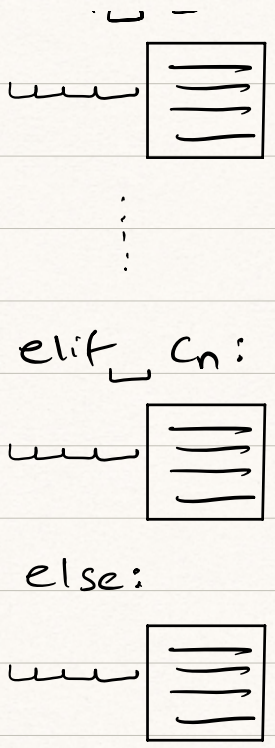
If we want execute different set of statements for different conditions then we use if-elif-else conditional statements.

if-elif-else Conditional statements

if condition 1 True execute statement set 1
elif " 2 " " " " 2
elif " 3 " " " " 3
:
elif " n " " " " n
else execute statement set(n+1).

Syntax (Rules)

```
if C1:
    
elif C2:
```



1) Here 'if', 'elif' and 'else' keywords are used
2) All the 'if' and 'if-else' conditional statements syntax have to be followed here.
3) elif (is short form for else-if) condition should always be written in between if and else.
4) elif blocks are used to write multiple conditions i.e we can write n number of elif blocks.
5) Indentation needs to be given inside each elif block.
6) First 'if' condition is checked, if True statements are executed. If False it will check first elif conditions, if True statements are executed. If False next elif condition is checked and

so on until last elif condition. If None are true then else statements are executed.

7) only one set of statements are executed depending on what condition was True.

ex:-

1) # To check if number is even or odd

```
number = int(input("Type any number: "))
if number%.2==0:
    print("The number {} is even".format(number))
if number%.2!=0:
    print("The number {} is odd".format(number))
else:
    print("sorry you have done a mistake")
```

2) Create a dynamic calculator:

```
# we will be accepting 3 inputs, nub1, operator, nub2
nub1 = int(input("Type number1: ")
opera = input("Type the operation -, +, /, %, //,
                                *, ** to be performed)
nub2 = int(input("Type number2: ")
if opera=="+":
    print("{}+{}={}".format(nub1, nub2,
                                nub1+nub2))
elif opera=='-':
    print("{}{}{}={}".format(nub1, opera, nub2,
```

nub1-nub2))

⋮

```
else:
    print("wrong inputs")
```

<u>Assignment</u> : Create a game for Rock, Paper, Scissors

\# we will be taking 2 inputs

\# Choice of person 1 (Rock/paper/scissor)

\# Choice of person 2 (Rock/paper/scissor)

Based on this we will get 9 combinations.

Person 1	Person 2	win
Rock	Rock	N/A
Rock	paper	Person 2
Rock	scissor	Person 1
Paper	Rock	Person 1
Paper	paper	N/A
paper	scissor	Person 2
scissor	Rock	Person 2
scissor	paper	Person 1
scissor	scissor	N/A

```
# code find below
P1 = input(# choice of person 1 (Rock/ paper/
                                    scissor))
P2 = input(# choice of person 2 (Rock/ paper/
                                    scissor))
print("person 1 chose {} and person 2 chose {}.
                        format(P1, P2))
if (P1=='ROCK' and P2=='ROCK') or (P1=='Paper' and P2==
        'paper') or (P1=='scissor and P2=='scissor'):
    print("No body wins play another round")

elif (P1=='ROCK' and P2=='scissor') or (P1=='Paper' and P2==
        'ROCK') or (P1=='scissor' and P2=='paper'):
    print("person 1 wins the hand")

elif (P1=='scissor' and P2=='ROCK') or (P1=='ROCK' and P2==
        'paper') or (P1=='paper' and P2=='scissor'):
    print("person 2 wins the hand")

else:
    print("wrong inputs are given")
```

Ternary operations:

Ternary operators also known as conditional expressions are operators that evaluate something based on a condition being True or False.

It simply allows testing a condition in a single line replacing the multiline if-else making the code compact.

Syntax:

[on-true statements] if [expression] else [on-false statements]

if we want to check multiple conditions

[cond 1 true statements] if [cond 1] else [cond 2 true statements] if [cond 2]

else [on-false statements]

Shorthand if:

* use this only when we are evaluating one expression and it has a single statement.
* we can use for multi statements by seperating the statements using comma ',' or semi-colon ';' but this is not recommended.

ex:-

Normal method

```
x=10
if x>5:
    print('hi')
```

Short hand if

```
if x>5: print('hi')
```

we can also use like: (not recommended)

```
if x>5: print('hi'), print('bye')
```

Short hand if-else:

* This should have only single statements inside the if expression.

```
ex:- print('hi') if x>5 else print('bye')
```

* These resemble the normal english language structure

```
ex:- print('hi') if (x>10 and x%.2==0) else print('bye')
```

Short hand elif if-else:

* Instead of elif keyword we use else-if.

Normal code

```
x=int(input())
if x>5:
```

```
        print ('hi')
    elif x%2==0:
        print ('welcome')
    else:
        print('bye')
```

Short hand

```
print('hi') if x>5 else print('welcome') if x%2==0
                    else print('bye')
```

* Reads just like english language logic.

loop statements :

Normal statement : Always executes as long as the syntax is correct.

Conditional statement : If the condition is True then the statements are executed. If the conditions are False statements are not executed.

looping statement : we use this if we want to execute a statement n number of times if the condition is true. The execution stops when the condition is false

two ways

- while loop (tricky to write)
- for loop (easy to write)

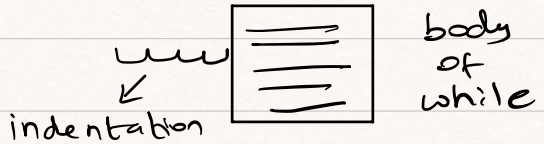
while loop :

To write a while loop we use a keyword 'while'

Syntax :

```
while␣condition:
```

space → ␣ ; colon at the end



body of while can contain print statements, normal statements, conditional statements or more looping statements.

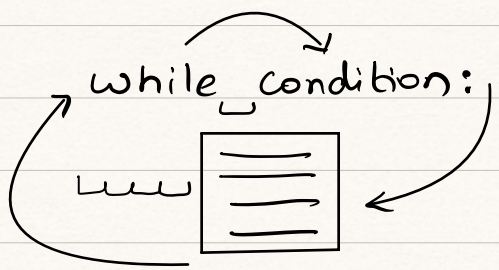
* When the condition True we enter the loop but if the condition is never False then we will never come out of the loop.

i.e if condition is True for ∞ no.of times then the while loop will be executed ∞ no.of times.

- In this scenario the memory becomes full and Python crash so very bad and we need to restart the kernel.

* It is the responsibility of the programmer to use logic to stop the while loop so that the condition is False after n iterations.

- One loop or one iteration



1) In this loop first keyword while is checked.
2) Then condition is check if it is True the statements are executed.
3) Then it goes back to checking the keyword.

This whole process is one loop or one iteration.

* For ease of writing while we can strictly follow the below steps.

1) Initialize a variable with some value.
2) write the while loop condition with the help of the variable.
3) Write the logic to make the condition False. so that we can come out of the loop.

ex:- a=1

```
while a<5:
    print("hi")
```

× here a is always <5 So ∞ iterations.

instead we have to write

```
a=1
while a<5:
    print("hi")
    a=a+1
```

✓

this a=a+1 incremental will help while condition to be False after 4 iterations.

1 loop	2 loop	3 loop	4 loop	5 loop
1<5	2<5	3<5	4<5	5<5 ×
hi	hi	hi	hi	
a=2	a=3	a=4	a=5	comes out of the loop

On the 5th loop the condition is False so program comes out of the loop.

we can use a decremental method also:

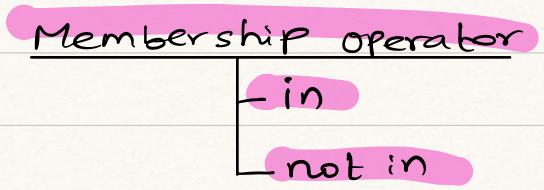
ex:-

```
a=10
while a>=1:
    print("hi")
    a=a-1
```

this will execute 10 times and in the 11th loop the program comes out of the loop.

Membership operator

They are the keywords by using which we can do membership operation on the data.



here 'in' and 'not' are keywords.

Membership operator checks whether a particular element is present in the data or not.

Output will be Boolean data type "True" or "False".

ex:- "a" in "abcd"

↳ True

"a" not in "abcd"

↳ False

'in' and 'not in' can be used for any data type.

ex:- 10 in [1, 'a', 'c', 2.2]

↳ False

They check whether 10 is a member of data or not.

* not in is generally not used in the loops.

For loop

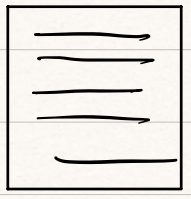
A for loop is used for iterating over a sequence (this is either a list, a tuple, a dictionary, a set, or a string).

With the for loop we can execute a set of statements, once for each item in a list, tuple, set etc.

The for loop does not require an indexing variable to be set beforehand.

Syntax

for ␣ variable ␣ in ␣ sequential data :



body of for loop

1) we use 'for' keyword here.
2) Here in is a membership operator.
3) for loop will not go to infinity loop as it will stop when sequential data stops.

```
ex:-  x = [1, 2, 3, 4]
      for y in x:
          Print("hi")
```

4) If it sees keyword 'for' the in operator will first check if data given is sequential data

type or not.

5) Next length of the data is taken to understand the no. of loop iterations.

6) Every loop one element from sequential data is given by data to variable to perform the loop.

ex:- x = [7, 8, 9, 10]

for y in x:

1st loop	2nd loop	3rd loop	4th loop	5th loop
y = 7	y = 8	y = 9	y = 10	x does not have anything to give to y so loop stops.
loop works	loop works	loop works	loop works	

No. of loops depends upon the length of the data given.

Two conditions checked in the for loop:-

1) whether it is sequential data.
2) whether element given to variable is present in data or not.

ex:- x = 1

for y in x:

print ("hi") X x is int so not iterable.

x = "abcd"

for y in x:

Print (y)

Check

x → sequential data

len(x) = 4

output

a

b

c

d

Issue for the 'for' loop

It is only running for the length of the data so if we want to run for n no. of iterations we do not need to create sequential data with length 'n' instead we can use range function.

Range Function (In built function)

It is an inbuilt function which inputs starting value and ending value and gives out sequential data type where ending value is not included.

range (SV, EV, step) → step size (default=1)

SV → starting value

EV → Ending value (this is not included)

ex:- range(1, 10) ← not included in the sequential data output.

↳ 1, 2, 3, 4, 5, 6, 7, 8, 9

So if we want to loop 1000 times in a for loop

```
for y in range(0, 1000):
    Print("hi")
```

output here will be hi printed 1000 times.

If we just mention range(EV) then SV is considered by default as 0.

so range(EV) is range(0, EV)

Step size can be used in range function also

range(1, 10, 1)

↳ 1, 2, 3, 4, 5, 6, 7, 8, 9

range(1, 10, 2)

↳ 1, 3, 5, 7, 9

so for y in range (1, 10, 3):

print(y)

↳ 1
4
7

range (-10, -1, -1)

↳ -10, -9, -8, -7, -6, -5, -4, -3, -2, -1

so x = "abcdef"

for y in range (len(x)-1, -1, -1):

print(y)

↳ f
e
d
c
b
a

Break and Continue in Python:

In Python, break and continue are loop control statements and keywords that are executed only inside the loop.

They cannot be executed outside the loop.

These statements either skip according to the conditions inside the loop or terminate the loop execution at some point.

Break statement

A break statement is used inside both while and for loops.

It terminates the loop immediately and transfers execution to the new statement after the loop.

i.e If we want to come out of the loop even if the condition is True, then we use Break keyword.

ex:-

```
a=1
while a<5:
    print("hi")
    if a==3:
        break
    a=a+1
```

1 loop	2 loop	3 loop
1<5	2<5	3<5
hi	hi	hi
1==3x	2==3x	3==3✓ → comes out of the loop as break is executed.
a=2	a=3	so here a=a+1 is not executed and while loop stops.

* break is like an off button.
* be careful with the indentation of while loop & other conditional loops inside the while loop. Each block of code needs to maintain its indentation.
* break - breaks the iteration and comes out of the loop.

we can use break statements in for loop also.

ex:-
```
x = "abcd"
for y in x:
    if y==b:
        break
    print(y)
```
↳ a

it breaks when y==b and comes out of the loop.

Continue statement

The continue statement causes the loop to skip its current execution at some point and move on to the next iteration.

Instead of terminating the loop like a break statement, it moves on to the subsequent execution

ex:-
```
a=1
while a<5:
    a=a+1      ← Incremental is used before continue because otherwise when a==3 it will not be incremented to make a=4 and execute 4 loop.
    if a==3:
        continue
    print("hi",a)
```

1 loop	2 loop	3 loop	4 loop	5 loop
1<5	2<5	3<5	4<5	5<5x
a=2	a=3	a=4	a=5	
2==3x	3==3✓	4==3x	5==3x	
hi, 2	↓	hi, 4	hi, 5	

Continue will be executed it will skip the code below for that iteration.
i.e did not execute print("hi",a) instead it will go to while loop for next iteration.

* So here unlike Break it will still run 4 loops but in 2nd loop hi, 3 is not printed.

Continue statement can be used in for loop also

```
ex:- x="abcd"
     for y in x:
         if y==b:
             continue
         print(y)
```

↳ a
c
d

(when y==b the iteration is skipped and output b is not printed)

Pass

Definition and usage:

The pass statement is used as a placeholder for future code.

When the pass statement is executed, nothing happens. But we 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.

```
ex:- def myFunction():
         pass
```

code to write a pattern for (A) using while loop.

```
a=1
while a<=5:
    if a==1:
        Print("*****")
    elif a==2:
        Print("*     *")
    elif a==3:
        Print("*****")
    elif a==4:
        Print("*     *")
    elif a==5:
        print("*     *")
    a=a+1
```

Output

code to write a pattern for (A) using for loop.

```
for y in range(1,6)
    if a==1:
        print("******")
    elif a==2:
        Print("*     *")
    elif a==3:
        Print("******")
    elif a==4:
        Print("*     *")
    elif a==5:
        print("*     *")
    a=a+1
```

output

```
******
*     *
******
*     *
*     *
```

Assignment: write code to print Alphabets (A-Z) using while loop.

Assignment: write code to print alphabets (a-z) using while loop.

Assignment : write code to print alphabets (a-z) using for loop.

Examples of for loop using range, break and continue.

ex:-

```
for z in range(1,6):
    if z==4:
        break
    print(z)
```

↳ 1
2
3

```
for z in range(1,6):
    if z==4:
        continue
    Print(z)
```

↳ 1
2
3
5

ex:-

```
x="abcdef"
for y in x:
    print(y)
```

↳ a
b
c
d
e
f

```
x="abcdef"
for y in range(len(x)):
    print(x[y])
```

↳ a
b
c
d
e
f

ex:- If we want to skip at 5th index value

```
x="abcdef"
for y in range(len(x)):
    if y==5:
        continue
    print(x[y])
```

(here f with index 5 is skipped)

a
b
c
d
e

Nested loops

A nested loop is a loop inside a loop.
The "inner loop" will be executed one time for each iteration of the outer loop.

Nested for loop: for loop inside the for loop

ex:-

```
for y in range(1,3):      → (1,2)
    for z in range(1,3):  → (1,2)
        print(y,z)
```

1st outer loop		2nd outer loop	
y=1		y=2	
1st inner loop	2nd inner loop	1st inner loop	2nd inner loop
z=1	z=2	z=1	z=2
Output (1,1)	(1,2)	(2,1)	(2,2)

* Inner loop will execute completely before moving back to the outer loop.

ex:-

```
for y in range(1,4):      → (1,2,3)
    for z in range(1,2):  → (1)
        print(y,z)
```

output

(1,1)

(2,1)

(3,1)

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

* The else block will not be executed if the loop is stopped by a break statement.

ex:-

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

→ 0
1
2
→ else not executed

```
for x in range(6)
    print(x)
else:
    Print ("finally finished")
```

→ 0
1
2
3
4
5
finally finished

while else: with the else statement we can run a block of code when the condition is no longer true.

ex:-

```
i=1
while i<6:
    print(i)
    i+=1
else:
    print("i is no longer less than 6")
```

↓

```
1
2
3
4
5
i is no longer less than 6
```

Practice questions of Basic python

Q) without using the length function try to find out the length of the data.

```
x= "abdf"
count =0
for y in x:
    count = count +1
print(count)
    ↳ 4
```

(if we use x=input( ) then code becomes dynamic)

1st loop	2nd loop	3rd loop	4th loop
count=1	count=2	count=3	count=4 ↓ comes out and then print(count) ↳ 4

Q) write a program in which we calculate the sum of all the values.

```
x=[1, 2, 3, 4, 5, 6, 7, 8]
count=0
for y in x:
    count= count+y
print(count)
    ↳ 36
```

Q) write a program in which we calculate the product of all the values.

```
x = [1, 2, 3, 4, 5, 6, 7, 8]
count = 1                    (here we use 1 because num multiplied by 1 is number itself)
for y in x:
    count = count * y
print(y)
    ↳ 40320
```

Q) write a program to convert dynamic input string to lower case.

```
x = input("Type string to be converted to lower case: ")
str = ""
for y in x:
    str = str + y.lower()
print (str)
```

Input	output
SUBHADRA	subhadra

(or)

```
x = input("Type string to be converted to lower case: ")
str = ""
```

```
for y in range(len(x))
    str = str + x[y].lower()
print(str)
```

Input	Output
SUBHADRA	Subhadra

Q) Write a python program to convert only odd indexed characters of dynamic input string to lower case. Ex:- PYTHON → PyThOn

```
x = input("Type string to be converted to lower case: ")
str = ""
for y in range(len(x))
    if y%2 != 0:
        str = str + x[y].lower()
    elif y%2 == 0
        str = str + x[y].upper()
print(str)
```

Input	Output
PYThon	pYtHoN

Q) Write a python program to convert only even indexed characters of dynamic input string to lower case. Ex:- PYTHON → pYtHoN

```
x = input("Type string to be converted to
str = ""                                  lower case: ")
for y in range(len(x))
    if y%2==0:
        str = str + x[y].lower()
    elif y%2!=0
        str = str + x[y].upper()
print(str)
```

Input	Output
PYThon	pYtHON

Q)
```
x = "abcd"
y = "ABCD"
```
we want a pairs of values

(a,A), (a,B), (a,C) ..... (d,D)

```
for z in x:
    for v in y:
        print(z,v)
```

Q) l1 = ["a", "b", 1, 2, 2.2]

1) Get only alphabets
2) append these alphabets to a new list l2=['a','b']
3) convert this list to a single string "ab".

```
l1 = ['a', 'b', 1, 2, 2.2]
l2 = []
for y in l1:                          # isalpha will only work with strings
    if str(y).isalpha():              # if True will go in
        l2.append(y)                  # ['a', 'b']
"".join(l2)
                                      # "ab"
```

Q) Create a calculator so that it will keep on asking inputs until we press "c".

```
a = "l"        # any alphabet other than c
while a != "c":
    n1 = int(input("Type first number: "))
    o1 = input("Type operator +,-,*,/,%,//,**: ")
    n2 = int(input("Type second number: "))
    if o1 == "+":
        print("{}+{}={}".format(n1,o1,n2,n1+n2))
    elif o1 == "-":
        print("{}-{}={}".format(n1,o1,n2,n1-n2))
    elif o1 == "*":
        print("{}*{}={}".format(n1,o1,n2,n1*n2))
```

```
elif o1 == "/":
    print("{}/{}={}".format(n1,o1,n2,n1/n2))
elif o1 == "%":
    print("{}%{}={}".format(n1,o1,n2,n1%n2))
elif o1 == "//":
    print("{}//{}={}".format(n1,o1,n2,n1//n2))
elif o1 == "**":
    print("{}**{}={}".format(n1,o1,n2,n1**n2))
else:
    Print("wrong inputs")
a = input("Type c for breaking: ")
```

Q) Write a Python Program to get only names from the string "name1.@gmail.com, name2.@gmail.com, name3.@gmail.com"

output = name1, name2, name3

```
x = "name1.@gmail.com, name2.@gmail.com,
     name3.@gmail.com"
list = x.split(",")
string = ""
for y in list:
    z = y.partition(".@")[0]
    string = string + z + ","
print(string)
```

↳ name1, name2, name3

Q) Given a string of odd length greater than 9, return a new string made of the middle three characters of a given string.

ex:- "mynameissan" ⟶ "mei"

```
x = input("Type your string here: ")
print(x[len(x)//2-1] + x[len(x)//2] +
                        x[len(x)//2+1])
```

Q) Write a python program to insert 2nd string in the middle of 1st string.

ex:- str1 = "myn"

str2 = "sa"

Output = "msayn"

```
Str1 = input("Type your first string : ")
Str2 = input("Type your second string : ")
Str1[0:len(str1)//2] + Str2 + Str1[len(str1)//2:]
```

Q) Write a program to remove vowels from the string.

```
x = input("Type your string here to remove vowels: ")
string = ""
for y in x:
    if y=='a' or y=='e' or y=='i' or y=="o" or y=="u":
```

```
        Continue
    else
        string = string + y
Print (string)
```

Assignment

(Binary search Algorithm)

If a sorted list of elements is given and we are looking for the position of a particular element.

Normally we would use a for loop to look element wise.

But if we have n elements and n is huge then the no. of comparisions we need might be n and lot of time is taken. This is called Linear search.

Instead we can use binary search to reduce the no. of comparisions.

So even if no. of elements increase there is no drastic change in no. of comparissions and time.

Binary Search Algorithm:

Step 1: Find the middle value in sorted array.

Step 2: If given num < middle value
search left half of array
If given num > middle value
Search right half of array
If given num = middle value
print the index of the middle value.

Keep repeating step2 until we get the answer.

Create function linearSearch (array, number)
binarySearch (array, number)

Check time taken using both the searches.