**Python Training Session-2**

* **Looping Statements:**
  + **While Loop:**

Syntax: while expression:

statements(s)

Ex:

#For loop  
 count=0  
 while(count<2):  
 count += 1  
 print("hello world")

* + **For Loop:**

Syntax: for iteration\_var in sequence:

statements(s)

Ex:

#while loop  
 # iterating over a list  
 L = ['this','is','a','while','loop','example']  
 for i in L:  
 print(i,end=' ')

* + **While loop with else:**

the else clause is executed when your while condition becomes false. If you break out of a loop or if an execution is raised, it wont be executed.

Ex:

#while-else loop  
  
j=0  
while j<4:  
 j += 4  
 print(j)  
else:  
 print("no break")  
print('\n')  
  
j=0  
while j<4:  
 j += 4  
 print(j)  
 break  
else:  
 #not executed bcs break is used  
 print("break")  
print('\n')

* **Control Statements:**
  + **if:**

If certain statement is true then a block statement is executed otherwise not

Ex:

i = 15  
if i>10:  
 print('i is the greater than 10')

* + **if-else:**

we can use the else statement with if statement to execute a block of code when the condition is false.

Ex:

i = 20  
if (i < 15):  
 print("i is smaller than 15")  
else:  
 print("i is greater than 15")

* + **if-elif-else:**

The if statements are executed from the top down. As soon as one of the conditions controlling the if is true, the statement associated with that if is executed, and the rest of the ladder is bypassed. If none of the conditions is true, then the final else statement will be executed.

Ex:

i = 20  
if (i == 10):  
 print("i is 10")  
elif (i == 15):  
 print("i is 15")  
elif (i == 20):  
 print("i is 20")  
else:  
 print("i is unknown")

* + **shorthand if and if-else:**

Whenever there is only a single statement to be executed inside the if or if-else block block then shorthand if can be used.

Ex:

i = 10  
if i<12: print('i is less than 12')  
  
print(True) if i>12 else print(False)

* **Flowcontrol Statements:**
  + Continue Statement:

It returns the control to the beginning of the loop

Ex:

for letter in 'justanexample':  
 if letter in 'aeiou':  
 continue  
 print('current letter: ',letter)  
print('\n')

* + Break:

It brings control out of the loop

Ex:

for letter in 'complexword':  
 if letter in 'aeiou':  
 break  
 print('current letter: ', letter)  
print('\n')

* + Pass:

Pass is used to write empty loops. Pass is also used for empty control statements, function and classes.

Ex:

for letter in 'justanexample':  
 pass  
print('last word:',letter)  
print('\n')

* **Operators:**

Operators are used to perform operations on values and variables.

* + **Arithmetic Operators:**

Arithmetic operators are used to performing mathematical operations like addition, subtraction, multiplication, and division.

+ addition

- subtraction

\* multiplication

/ division(float)

// division(floor)

% modulus

\*\* power

* + **Comparison Operators:**

It compares the values and returns True or False according to the condition.

> greater than

< less than

== equal to

!= not equal to

>= greater than or equal to

<= less than or equal to

* + **Logical Operators:**

It performs logical AND, logical OR and logical NOT operations. It is used to combine conditional statements.

and True if both are true

or True if either of the operands is true

not True if the operand is false

* + **Bitwise Operators:**

It acts on bits and performs the bit-by-bit operations. These are used to operate on binary numbers.

& Bitwise AND

| Bitwise OR

~ Bitwise NOT

^ Bitwise XOR

>> Bitwise right shift

<< Bitwise left shift

* + **Assignment Operators:**

They are used to assigning values to the variables.

= Assign value of right side of expression to left side operand

+= add AND add right-side operand with left side operand and then assign to left operands

-= subtract AND

\*= multiply AND

/= divide AND

%= modulus AND

//= divide(floor) AND

\*\*= exponent AND

&= perform Bitwise AND on operands and assign to left operands

|= Bitwise OR and assign to left operands

**^=** Bitwise XOR and assign to left operands

>>= Bitwise right shift and assign to left operands

<<= Bitwise left shift and assign to left operands

* + **Identity Operators:**

**is** and **is not** are the identity operators, both are used to check if two values are located on the same part of the memory.

is True if operands are identical

is not True if operands are not identical

* + **Membership Operators:**

**in** and **not in** are the membership operators. They are used to test whether a value or variable is in a sequence.

in True if value is in the sequence

not in True if the value is in the sequence