**Iterations, strings and Files in Python Application Programming**

2.1 ITERATION

Using the while clause, loops that never end, breaks, and "infinite loops" finishing iterations with Continue, using for-loops that are definitive, looping design,loops for adding and subtracting, Loops at the maximum and minimum

2.2 STRINGS

A sequence is a string. using len to determine a string's length, traversal via a looped string, slice strings, Strings cannot be changed, counting while looping, The operator for, string techniques for string comparison, string analysis Formatting command

2.3 FILES

Files, Persistence, launching a file, lines and text files, reading documents During a file search, allowing the user to select the filename Try, Except, and Open are used, writing documents Debugging

* 1. **ITERATION**

Iteration is a processing of repeating some task. In a real time programming, we require a set of statements to be repeated certain number of times and/or till a condition is met. Every programming language provides certain constructs to achieve the repetition of tasks. In this section, various such looping structures are discussed.

 **The *while* Statement**

The *while* loop has the syntax as below –

***while*** condition:

statement\_1 statement\_2

……………. statement\_n

statements\_after\_while

* + - Here, ***while*** is a keyword, the flow of execution for a while statement is as below.
		- The condition is evaluated first, yielding True or False
		- If the condition is false, the loop is terminated and statements after the loop will be executed.
		- If the condition is true, the body will be executed which comprises of the statement\_1 to statement\_n and then goes back to condition evaluation.

Consider an example –

* + - n=1

while n<=5:

print(n)

#observe indentation n=n+1

print("over")

The output of above code segment would be – 1

2

3

4

5

over

* In the above example, a variable n is initialized to 1. Then the condition n<=5 is being checked. As the condition is true, the block of code containing print statement print(n) and increment statement (n=n+1)are executed. After these two lines, condition is checked again. The procedure continues till condition becomes false, that is when n becomes 6. Now, the while-loop is terminated and next statement after the loop will be executed. Thus, in this example, the loop is ***iterated*** for 5 times.

Consider another example –

* + n=5

while n>0:

print(n)

#observe indentation n=n-1

print("Blast off!")

The output of above code segment would be – 5

4

3

2

1

Blast off!

Each time the body of the loop is executed, iteration is referred to. It should be noted that the variable n is initialised before the loop begins and that it is increased/decremented inside the loop. Iteration variables, often known as counter variables, are variables that change their value after each iteration and govern how the loop as a whole is executed. The loop may not end and continue running indefinitely if the count variable is not updated correctly during the loop.

**• Infinite Loops, Break and Continue**

• If the condition can never become false, a loop can run indefinitely.

 • For example,

n=1

while True:

print(n)

 n=n+1

In this case, the loop's condition is the constant True, which will never be satisfied. Sometimes the condition is written so that it can never fail, preventing the program's control from leaving the loop. • In some circumstances, we may wish to get out of the loop even before the regular termination of the loop. This situation may occur owing to either an incorrect condition or due to not updating the counter variable. Break statement is utilised for this.The use of break is seen in the example below. Up until a negative number is inputted, the values in this case are taken from the keyboard. The loop ends when it is determined that the input is negative.

while True:

x=int(input("Enter a number:"))

if x>= 0:

**print("You have entered ",x)**

else:

print("You have entered a negative number!!")

### break #terminates the loop

OUTPUT:

Enter a number:23

You have entered 23

Enter a number:12

You have entered 12

Enter a number:45

You have entered 45

Enter a number:0

You have entered 0

 Enter a number:-2

You have entered a negative number!!

•In the preceding example, the while-loop's condition was the constant True, which can never turn out to be untrue. So, an infinite loop was a possibility. By combining a break statement with a condition, this has been prevented.

• If the user input is a negative integer, the condition is maintained inside the loop so that the loop ends. This means that the loop could end after just one iteration if the user inputs a negative value for the first time, or it could take hundreds of iterations if the user continuously inputs positive numbers. Therefore, the number of iterations in this case is arbitrary.

• But we're making sure it won't be an endless loop by giving the user control over

Here is another illustration of a while with break statement: The user's input is collected by the code below until they write done:

while True:

line = input(">")

 if line == 'done':

break

 print(line)

print('Done!')

• In the aforementioned illustration, the loop runs continually until it reaches the break statement because the loop condition was True.

• Each time the user is prompted to enter data. The brak statement ends the loop if the user types done. Otherwise, the programme repeats everything the user inputs and returns to the loop's beginning.

•Output will be:

>hello hello

>finished

finished

>done Done!

Depending on a circumstance with the current iteration, a programmer may occasionally want to skip a few statements in the loop and continue to the next iteration. Continue statements are employed for this purpose.

**#Python program to demonstrate continue statement**

Take into account the scenario in which you must create a programme that publishes the numbers from 1 to 10 but not 6. It is stated that you must do this task using a loop, and only one loop may be used. Here is where the continue statement is used. What we can do is execute a loop from 1 to 10 times, comparing the value of the iterator with 6 each time. We will utilise the continue statement to skip to the next iteration if the value is equal to 6, otherwise we will output the value.

#loop from 1 to 10

for i in range(1, 11):

# If i is equals to 6,

# continue to next iteration # without printing

if i == 6:

continue

else:

# otherwise print the value

# of i

print(i, end = " ")

**Output:**

1 2 3 4 5 7 8 9 10

•Example of a loop that copies its input until the user types “done”, but treats lines that start with the hash character as lines not to be printed

while True:

 line=input('>')

 if line[0] == '#':

continue