ITERATIONS, STRINGS AND FILES IN PYTHON APPLICATION PROGRAMMING

Abstract

In Python application programming, iterations, strings, and files play crucial roles. Iterations, achieved through loops, allow efficient repetitive tasks. Strings, sequences of characters, facilitate text manipulation and processing. File operations enable reading from and writing to external storage, enabling data persistence. These concepts collectively empower developers to create versatile and powerful applications.

Keywords: Iterations, Strings and Files Python Application Programming

Author

Dr. Renuka Sagar
Associate Professor
Department of Artificial Intelligence and Machine Learning
Ballari Institute of Technology and Management
Jnana Gangothri Campus
Hospet, Bellary, Karnataka, India.
renukasagar83@gmail.com

I. 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

II. 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

III. 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

IV. ITERATION

Iteration is the process of repeatedly doing a task. Software development calls for a group of related statements to be repeated a predetermined many number of times or until a situation is satisfied. Every form of programming has a specific set of features that enable task repetition. Such looping structures are examined in this section in various forms.

1. *While* Statement: *while* loop syntax is as below –

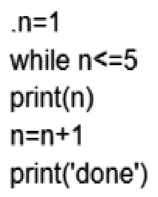
```
while condition:

        statement_1
        statement_2
        ……………
        statement_n

    statements_after_while
```

In this instance, the word "while" is a keyword, and the while statement's execution proceeds as shown below.

- After assessing the circumstance, a True or False result is produced.
- If the condition is false, the loop is ended and the statements that follow it are run.
- The body, which consists of statements_1 through_n, will be executed if the condition is true, and the assessment of the condition will then resume.Consider an example –

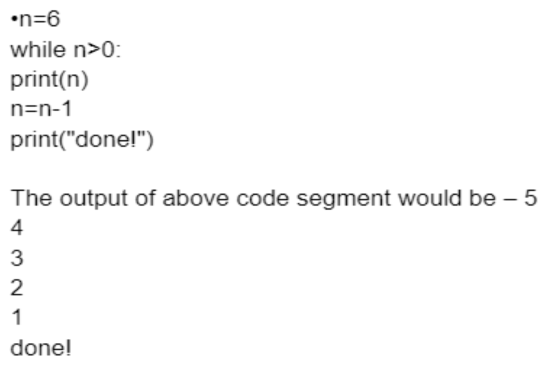


```
.n=1
while n<=5
print(n)
n=n+1
print('done')
```

The output of above code segment would be –1
2
3
4
5
done

In the above illustration, the initial value of the variable n is 1. Then it is determined if the condition n=5 is true. The increment statement (n=n+1) and the print statement (print(n)) are executed because the condition is true. The condition is tested once more after these next two lines. The process continues until condition is false, at which point n is equal to 6. The while-loop is now over, and the subsequent statement will be executed. The loop is iterated five times in this case.

Consider another example –



```
•n=6
while n>0:
print(n)
n=n-1
print("done!")
```

The output of above code segment would be – 5
4
3
2
1
done!

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.

2. 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("Enter ",x)
else:
print("You have entered a negative number!!")
```

break

```
OUTPUT:
Enter a number:45
You have entered 45
Enter a number:13
You have entered 13
Enter a number:6
You have entered 6
Enter a number:0
You have entered 0
Enter a number:-1
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.

V. #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 they 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 hash character as lines not to be printed

```
while True:
 line=input('>')
 if line[0] == '#':
continue
```