Lecture 07

Iterative Structure (for and while)

Empowering For Professional Excellence

‘for’ loop

In Python, the for loop is often used to iterate over iterable objects such as lists, tuples, or strings. Traversal is the process of iterating across a series. If we have a section of code that we would like to repeat a certain number of times, we employ for loops. The for-loop is usually used on an iterable object such as a list or the in-built range function. The for statement in Python traverses through the elements of a series, running the block of code each time.

The for statement is in opposition to the "while" loop, which is employed whenever a condition requires to be verified each repetition or when a piece of code is to be repeated indefinitely.

‘for’ loop

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

This is less like the for keyword in other programming languages, and works more like an iterator method as found in other object-orientated programming languages.

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 set beforehand.

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
  print(x)
```

Break Statement

With the break statement we can stop the loop before it has looped through all the items: Exit the loop when x is "banana":

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
  print(x)
  if x == "banana":
    break
```

```
=== RESTART: C:/Users/He
apple
banana
>>>
=== RESTART: C:/Users/He
apple
>>>
```

Exit the loop when x is "banana", but this time the break comes before the print:

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
  if x == "banana":
    break
  print(x)
```

The continue Statement

With the continue statement we can stop the current iteration of the loop, and continue with the next:

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
  if x == "banana":
    continue
  print(x)
```

```
=== RESTART: C:/U
apple
cherry
>>>
```

The range() Function

To loop through a set of code a specified number of times, we can use the range() function, The range() function returns a sequence of numbers, starting from 0 by default, and increments by 1 (by default), and ends at a specified number.

Note that range(6) is not the values of 0 to 6, but the values 0 to 5.

```
>>> for x in range(6):
  print(x)


0
1
2
3
4
5
>>>
```

The range() Function

The range() function defaults to 0 as a starting value, however it is possible to specify the starting value by adding a parameter: range(2, 6), which means values from 2 to 6 (but not including 6):

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


2
3
4
5
>>>
```

The range() Function

The range() function defaults to increment the sequence by 1, however it is possible to specify the increment value by adding a third parameter: range(2, 30, 3):

```
>>> for x in range(2, 30, 3):
  print(x)


2
5
8
11
14
17
20
23
26
29
>>>
```

Else in For Loop

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

Example

Print all numbers from 0 to 5, and print a message when the loop has ended

```
for x in range(6):
  print(x)
else:
  print("Finally finished!")
```

```
=== RESTART: C:/Users/Hera N
0
1
2
3
4
5
Finally finished!
>>>
```

Note: The else block will NOT be executed if the loop is stopped by a break statement.

Break the loop when x is 3, and see what happens with the else block:

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

```
=== RESTART: C:/Use
0
1
2
>>>
```

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":

Example

Print each adjective for every fruit:

```
adj = ["red", "big", "tasty"]
fruits = ["apple", "banana", "cherry"]

for x in adj:
  for y in fruits:
    print(x, y)
```

```
=== RESTART: C:/Users/He
red apple
red banana
red cherry
big apple
big banana
big cherry
tasty apple
tasty banana
tasty cherry
>>>
```

The pass Statement

'For' loops cannot be empty, but if you for some reason have a 'for' loop with no content, put in the 'pass' statement to avoid getting an error.

Example

```
for x in [0, 1, 2]:
  pass
```

‘while’ loop

While loop falls under the category of indefinite iteration. Indefinite iteration means that the number of times the loop is executed isn’t specified explicitly in advance.

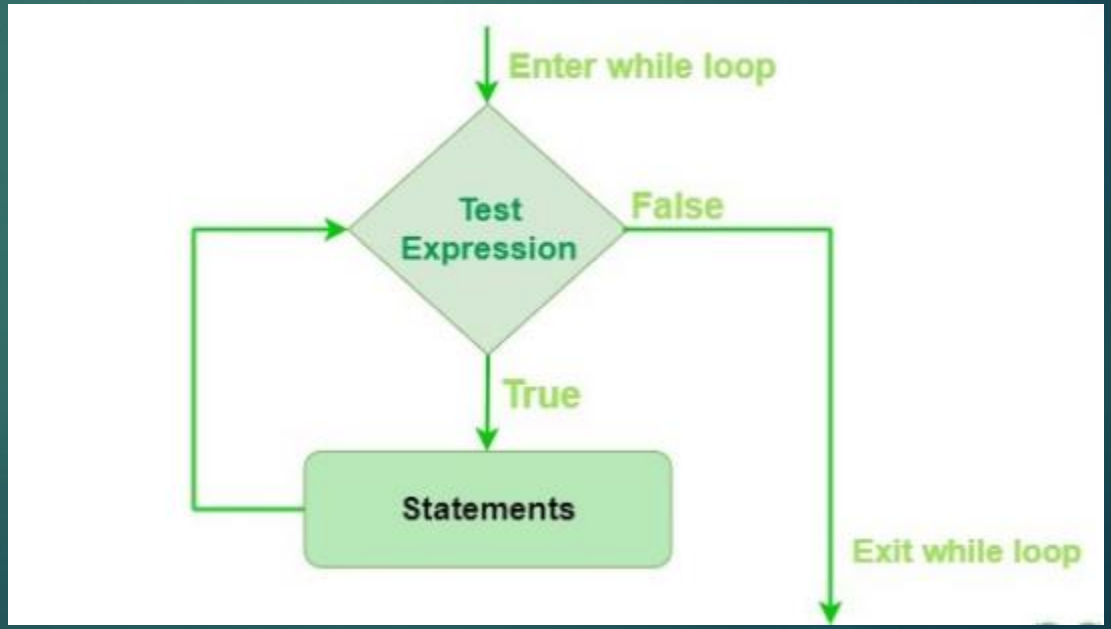
Statements represent all the statements indented by the same number of character spaces after a programming construct are considered to be part of a single block of code. Python uses indentation as its method of grouping statements. When a while loop is executed, expr is first evaluated in a Boolean context and if it is true, the loop body is executed. Then the expr is checked again, if it is still true then the body is executed again and this continues until the expression becomes false.

‘while’ loop

Python While Loop is used to execute a block of statements repeatedly until a given condition is satisfied. And when the condition becomes false, the line immediately after the loop in the program is executed.

Syntax:

while expression:
    statement(s)



Example
Print i as long as i is less than 6:

Note: remember to increment i, or else the loop will continue forever. The while loop requires relevant variables to be ready, in this example we need to define an indexing variable, i, which we set to 1.

```
i = 1
while i < 6:
  print(i)
  i += 1
```

```
>>>
== RESTART: C:/Users/Hera
1
2
3
4
5
>>>
```

The break Statement

With the break statement we can stop the loop even if the while condition is true:

Example
Exit the loop when i is 3:

```
i = 1
while i < 6:
  print(i)
  if i == 3:
    break
  i += 1
```

```
== RESTART: C:/Users/Her
1
2
3
>>>
```

The continue Statement

With the continue statement we can stop the current iteration, and continue with the next:

Example
Continue to the next iteration if i is 3:

```
i = 0
while i < 6:
  i += 1
  if i == 3:
    continue
  print(i)
```

```
== RESTART: C:/Users/Hera No
1
2
4
5
6
>>>
```

The else Statement

With the else statement we can run a block of code once when the condition no longer is true:

Example
Print a message once the condition is false:

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

```
== RESTART: C:/Users/Hera Noor/
1
2
3
4
5
i is no longer less than 6
>>>
```

Single statement while block

With the else statement we can run a block of code once when the condition no longer is true:

Example
Print a message once the condition is false:

```
count = 0
while (count < 5): count += 1; print("Hello World")
```

```
== RESTART: C:/Users/Hera
Hello World
Hello World
Hello World
Hello World
Hello World
>>>
```

Task – 7

1. Write a Python program to find those numbers which are divisible by 7 and multiple of 5, between 1500 and 2700 (both included).
2. Write a Python program to construct the following pattern, using a nested for loop.

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

Task – 7

3. Find the sum of squares of each element of the list using for loop.

numbers = [3, 5, 23, 6, 5, 1, 2, 9, 8]

Output:

The sum of squares is: 774

4. Write a Python program to count the number of even and odd numbers from a series of numbers. Sample numbers : numbers = (1, 2, 3, 4, 5, 6, 7, 8, 9)

Expected Output :

Number of even numbers : 5

Number of odd numbers : 4

Thank you!