Loops in Python

Distributed Network and Systems Laboratory,
Chonnam National University
Presenter : Shivani Kolekar

Recap

- ✓ **Decision Structures**
- ✓ **Boolean Expressions and Relational Operators**
- ✓ **If-else statements**

Today's Goals

- ✓ **Introduction to loops (for and while loops)**
- ✓ **Loop control statements (break and continue)**
- ✓ **Writing programs with loops**
- ✓ **Q & A**

Repetitive Actions and using Loops

- Often we need to do some (program) activity numerous times:

- So we might as well use cleverness to do it.
  That's what loops are for.

It doesn't have to be the exact same thing over and over.

And this is how we really harness the power of a computer that can perform tens of billions (or more) computations per second!

While Loop

While Loop

The majority of programming languages include syntax to **repeat** operations.

'while loop' is one option.

General form:

```
while condition:
    statement
    statement
    etc.
```

Meaning: as long as the condition remains true, execute the statements.

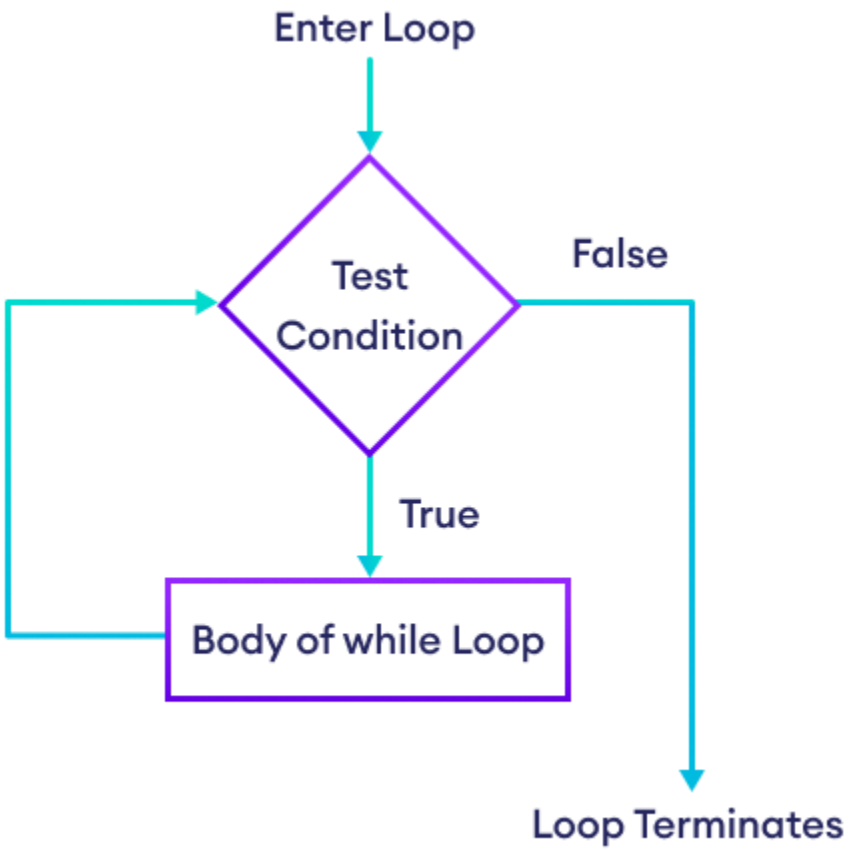

Figure: Flowchart - Logic of while loop

While Loop

Example : display numbers from 1 to 5.

```python
# program to display numbers from 1 to 5

# initialize the variable
i = 1
n = 5

# while loop from i = 1 to 5
while i <= n:
    print(i)
    i = i + 1
```

Output

```
1
2
3
4
5
```

How the loop works for this example

Variable		Condition: `i <= n`	Action
`i = 1`	`n = 5`	`True`	`1` is printed. `i` is increased to **2**.
`i = 2`	`n = 5`	`True`	`2` is printed. `i` is increased to **3**.
`i = 3`	`n = 5`	`True`	`3` is printed. `i` is increased to **4**.
`i = 4`	`n = 5`	`True`	`4` is printed. `i` is increased to **5**.
`i = 5`	`n = 5`	`True`	`5` is printed. `i` is increased to **6**.
`i = 6`	`n = 5`	`False`	The loop is terminated.

break statement

- ✓ The break statement is used to terminate the loop immediately when it is encountered.

```
while condition:
    # code
    if condition:
        break

    # code
```

Program: while_loop_with_break.py

```
counter = 0

while counter < 3:
    # loop ends because of break
    # the else part is not executed
    if counter == 2:
        break
    print('Inside loop')
    counter = counter + 1
else:
    print('Inside else')
```

Output

```
Inside loop
Inside loop
```

For Loop

for loop : A count controlled loop

CONCEPT: A count-controlled loop iterates a specific number of times.

- In Python, you use the for statement to write a count-controlled loop.

```
for variable in [value1, value2, etc.]:
    statement
    statement
    etc.
```

The for statement executes in the following manner:

1. The variable is assigned the first value in the list, then the statements that appear in the block are executed.
2. Then, variable is assigned the next value in the list, and the statements in the block are executed again.
3. This continues until variable has been assigned the last value in the list.

Understanding for loop iterations

Program: Simple_loop1.py

```
1  # This program demonstrates a simple for loop
2  # that uses a list of numbers.
3
4  print('I will display the numbers 1 through 5.')
5  for num in [1, 2, 3, 4, 5]:
6      print(num)
```

Program Output

```
I will display the numbers 1 through 5.
1
2
3
4
5
```

1st iteration:

```
for num in [1, 2, 3, 4, 5]:
    print(num)
```

2nd iteration:

```
for num in [1, 2, 3, 4, 5]:
    print(num)
```

3rd iteration:

```
for num in [1, 2, 3, 4, 5]:
    print(num)
```

4th iteration:

```
for num in [1, 2, 3, 4, 5]:
    print(num)
```

5th iteration:

```
for num in [1, 2, 3, 4, 5]:
    print(num)
```

Range function

✓ The range() function returns a sequence of numbers,
starting from 0 by default,
and increments by 1 (by default), and
stops before a specified number.

Program: for_loop_using_range.py

```
>>> for num in range(5):
...     print(num)
...
0
1
2
3
4
```

Program: for_loop_using_list.py

```
>>> for num in [0, 1, 2, 3, 4]:
...     print(num)
...
0
1
2
3
4
```

- The range function creates a type of object known as an iterable.

An iterable is an object that is similar to a list. It contains a sequence of values that can be iterated over with something like a loop.

Sequence

✓ A Python sequence holds multiple items stored one after another.

```
>>> seq = [2, 3, 5, 7, 11, 13] # a list
```

✓ The range function is a good way to generate a sequence.

`range(a, b)` : denotes the sequence `a, a+1, ..., b-1`.

`range(b)` : is the same as `range(0, b)`.

`range(a, b, c)` : generates `a, a+c, a+2c, ...., b'`,

where b' is the last value < b.

Range function Syntax

✓ **Syntax:**

```
range(start, stop, step)
```

Parameter	Description
start	Optional. An integer number specifying at which position to start. Default is 0
stop	Required. An integer number specifying at which position to stop (not included).
step	Optional. An integer number specifying the incrementation. Default is 1

Range(start,stop,step) : example

start, stop, step

```
>>> for i in range(11, 0, -3):
        print(i, end=" ")
...
11 8 5 2

>>> for i in range(0, 11, 3):
        print(i, end=" ")
...
0 3 6 9
```

continue statement

- ✓ The continue statement is used in loops (such as ‘for’ and ‘while’ loops) to alter the flow of the program and skip the current iteration of the loop when a specific condition is met.

- ✓ It allows you to bypass the rest of the current iteration and move to the next iteration of the loop immediately.

Program: continue_statement.py

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

Output

```
apple
cherry
```

Activity 1

✓ write a program that displays the numbers in `range(25, 101)` which have whole number square root values and their respective square roots, in a table similar to the following:

Number	Square root
25	5
36	6
49	7
64	8
81	9
100	10

Activity 2

✓ Refering to this Program and get the output for base= 21, and maximum power= 20.

Program: Powers_of_input_values.py

```
base = float(input("Enter the base: "))
n = int(input("Enter the maximum power (n): "))

print('Powers of', base,'up to', n,':')

for power in range(n + 1):
    result = base ** power
    print(base,"to the",power, "is:", result)
```

Output

```
Enter the base: 4
Enter the maximum power (n): 10
Powers of 4.0 up to 10:
4.0 to the 0 is: 1.0
4.0 to the 1 is: 4.0
4.0 to the 2 is: 16.0
4.0 to the 3 is: 64.0
4.0 to the 4 is: 256.0
4.0 to the 5 is: 1024.0
4.0 to the 6 is: 4096.0
4.0 to the 7 is: 16384.0
4.0 to the 8 is: 65536.0
4.0 to the 9 is: 262144.0
4.0 to the 10 is: 1048576.0
```

Lets Code!