Chp03B Loops using **for** and **while** structures

References

1. https://www.w3schools.com/python/python_while_loops.asp w3school Python Tutorial (while loop)

2. https://www.w3schools.com/python/python_for_loops.asp w3school Python Tutorial (for loop)

3. https://www.tutorialspoint.com/python3/python_loops.htm Loops (while and for structures)

4. https://docs.python.org/3/tutorial/controlflow.html *Read sections 4.1-4.4. Can also be obtained from Python manual*
5. *Learning Python, 5th (mark Lutz Oreilly 2013) – chapter 13: while and for Loops*

Loops

A loop statement allows to execute a statement or group of statements multiple times.

Python allows two types of loops:

- The **while** repetition structure
- The **for** repetition structure

while loop - I

Syntax:

```
while relational_expression_is_true:
      statement(s)
```

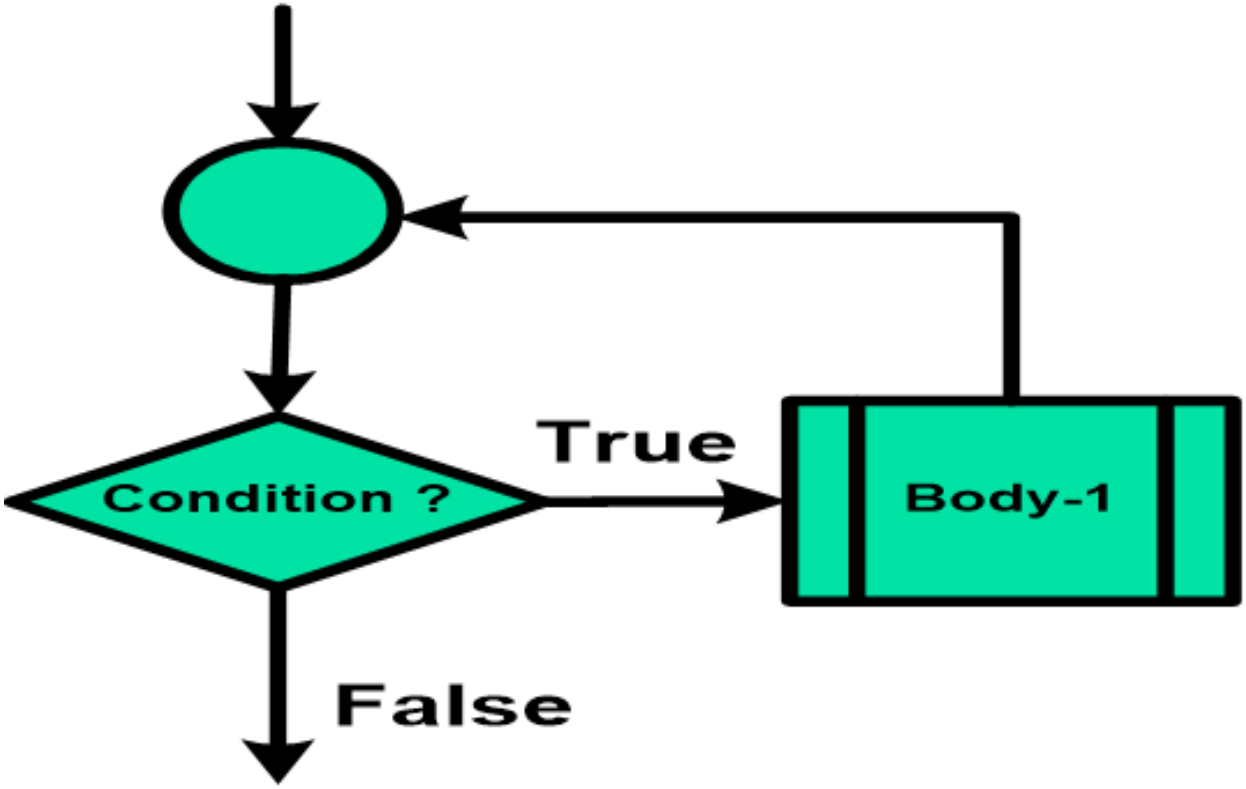
A while loop repeatedly executes a target statement as long as a given condition is True. It tests the condition before executing the loop body.

As it was for `if..elif..else` structure, Python uses indentation as its method of grouping statements.

Note: The loop might not ever run. When the condition is tested and the result is false, the loop body will be skipped and the first statement after the while loop will be executed.

While loop -II

Flow diagram of while loop is as follows:



While loop - III

Example: display 1,2,3,..10 in one line
Numbers are separated by comma

chp03Bex01while01A

```
number=1
while number<=10:
    print(number,end=",")
    number=number+1
```

Output:

1,2,3,4,5,6,7,8,9,10,

The program include comma after 10! See next slide on how this can be corrected.

While loop - IV

Example:

chp03Bex02while01B

```
number=1
while number<=10:
    if number<=9:
        print(number,end=",")
    else:
        print(number)
    number=number+1
```

Output:

1,2,3,4,5,6,7,8,9,10

The program takes care not to include comma after 10

While loop -V

Example: Calculate and display the **sum** and **average** of numbers 1..10

chp03Bex03while02

```
number=1;sum=0
while number<=10:
    sum=sum+number
    number=number+1 #number +=1
average=sum/10
print(f"The sum is {sum} and average is {average:.2f} ")
```

Output:

```
The sum is 55 and average is 5.50
```

Using else statement with while -I

Python supports having an **else** statement associated with a loop statement. If the **else** statement is used with a **while** loop, the **else** statement is executed when the Boolean Expression becomes false.

The following example illustrates the combination of an else statement with a while statement that prints a number as long as it is less than 5, otherwise the else statement gets executed.

Using else statement with while - II

```
chp03Bex04while03
number=1
while number<5:
    print (f"{number} is less than 5")
    number = number + 1
else:
    print (f"{number} is not less than 5")
```

Output:

```
1  is  less than 5
2  is  less than 5
3  is  less than 5
4  is  less than 5
5  is not less than 5
```

for loop

for loop - I

- A for loop is used for iterating over a sequence (that is either a **list**, a **tuple**, a **dictionary**, or a string). List, tuple and dictionary will be discussed in a later chapter.
- With the for loop, a set of statements can be executed, once for each item in a list, tuple, string, etc.
- **Iterator** is an object which allows a programmer to traverse through all the elements of a collection

for loop -II

Syntax:

```
for iteratingVar in sequence:
    statement(s)
```

```
chp03Bex05for01
#iterate and display one character at a time
str= "Python"
for letter in str:
    print(letter)
else:
    print("All characters displayed")
```

for loop - III

Output of previous slide is:

```
P
y
t
h
o
n
All characters displayed
```

Like while loop, If the **else** statement is used with a **for** loop, the **else** statement is executed when the loop has exhausted iterating the list as shown in example `chp03Bex05for01`

The break statement - I

The **break** statement is used for premature termination of the current loop. After abandoning the loop, execution at the next statement is resumed.

The most common use of break is when some external condition is triggered requiring a hasty exit from a loop.

The **break** statement can be used in both ***while*** and ***for*** loops.

The break statement - II

Example: chp03Bex06break

```
number=1
while number<5:
    print (number)
    if number==3: break
    number=number+1
print()    #insert blank line
str="Python"
for letter in str:
    print(letter,end="")
    if letter=="h": break
```

See output on the next slide.

The break statement - III

The output will be:

```
1

2

3


Pyth
```

The continue statement -I

- The **continue** statement in Python returns the control to the beginning of the current loop. When encountered, the loop starts next iteration without executing the remaining statements in the current iteration.

- The **continue** statement can be used in both ***while*** and ***for*** loops.

The continue statement -II

```
chp03Bex07continue
str="Python"
for letter in str:
    if letter=="h": # skip letter h
        continue
    print(letter,end="")
```

Output:

Pyton

Note: skips letter h

Using range function

The range function -I

- To loop through a set of code a specified number of times, the **range()** function is used
- 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

Syntax: range(start, stop[, step])

where step is optional. Parameters must be integers and can be negative. Use help("range") for details

```
range(6)≡range(0,6) => 0,1,2,3,4,5
range(1,6) => 1,2,3,4,5
range(1,6,2) => 1,3,5
range(1,7,2) => 1,3,5
```

The range function-II

chp03Bex08range01

```
print("range(6):",end=" ")
for x in range(6): print(x, end=" ")

print("\nrange(1,6):",end=" ")
for x in range(1,6): print(x, end=" ")

print("\nrange(1,6,2):", end=" ")
for x in range(1,6,2): print(x, end=" ")

print("\nrange(1,7,2):", end=" ")
for x in range(1,7,2): print(x, end=" ")
```

The range function-III

Output of chp03Bex08range01

```
range(6): 0 1 2 3 4 5
range(1,6): 1 2 3 4 5
range(1,6,2): 1 3 5
range(1,7,2): 1 3 5
```

The range function - IV

To iterate over the indices of a sequence, combine range() and len() as following example demonstrates:

chp03Bex09range02 and chp03Bex10range03

```
#print vowels in a string and number
str="Python is a High Level Programming Language"
for x in range(len(str)):
……..  # Refer to complete code
```

Output:

```
oiaieeoaiauae
There are 13 vowels in str
```

Nested loops -I

- A nested loop is a loop inside a loop. Syntax is:

```
for iteratingVar1 in sequence1:
    for iteratingVar2 in sequence2:
        statements(s)  #inner loop statements
    statements(s)      #outer loop statements
```

- The "inner loop" will be executed one time for each iteration of the "outer loop"

Example: chp03Bex11nestedloop

Shows how to form multiplication table.

Code is on the next slide.

Output =>

```
   1   2   3
   -------
1  1   2   3
2  2   4   6
3  3   6   9
```

chp03Bex11nestedloop

```
for outer in range(1,4):
    print(f" {outer:2d}",end="") print()
print(f" {'-'*8}")

for outer in range(1,4): # outer loop
    print(outer,end=" ")
    for inner in range(1,4): # inner loop
        print(f"{outer*inner:2d}", end=" ")
print()
```

Nested loops -II

The syntax for a nested while loop statement in Python programming language is as follows:

```
while expression:
   statement(s)
   while expression:
      statement(s)
   statement(s)
```

Note: When using loop nesting, you can put any type of loop **outside** and any other type of loop in the **inside**. For example a **for** loop can be inside a **while** loop or vice versa.

do..while -I

The **do..while** loop is used to check condition after executing the statement. It is like while loop but it is executed at least once.

General do..while Loop Syntax:

```
do {
      statement(s)
} while (condition)
```

- What are uses of do..while loop?
- Python does not have built in do..while loop, but it can be emulated.

do..while -II

do..while structure can be emulated using following pseudocode:

```
while True:
    do_something
    if certain_condition_is_true:
        break
    else:
        do_something_else
#execute this line if
certain_condition_is_true==True
```

The above loop won't exit unless certain_condition_is_true==True. See next slide for example.

do..while -III

chp03Bex12doWhile

```
while True:
    marks=float(input("Enter marks (0..100) "))
    if marks>=0 and marks<=100:
        break
    else:
      print(f"Entered marks {marks:.2f} is
outside the range, Try again")

print(f"Entered marks {marks:.2f} is valid")
```

do..while -IV

chp03Bex13doWhile_alternative1

```
validMarks=False
while validMarks==False:
    marks=float(input("Enter marks (0..100)
"))
    if marks>=0 and marks<=100: #valid
        validMarks=True
     else:
        print(f"Entered marks {marks:.2f}
                is outside the range, Try
                again")
print(f"Entered marks {marks:.2f} is valid")
```

do..while -V

chp03Bex14doWhile_alternative2

```
marks=float(input("Enter marks (0..100) "))
while marks<0 or marks>100:
    print(f"Entered marks {marks:.2f} is
    outside the range, Try again")
    marks=float(input("Enter marks (0..100) "))

print(f"Entered marks {marks:.2f} is valid")
```

==End of chapter 3B ==

Next topic:

Chapter 04 Working with Lists,Tuples and Dictionaries