M110: Python Programming

Meeting #4

Control Structures-2: Repetition Structures

AOU
الجامعة العربية المفتوحة
Arab Open University
Lebanon

Prepared by Dr. Ahmad Mikati

Contents

4.1 Introduction to repetition Structures
4.2 The ***for*** loop
4.3 The ***while*** loop
4.4 Sentinels
4.5 Input Validation loops
4.6 Break and Continue statements
4.7 Nested loops

AOU
الجامعة العربية المفتوحة
Arab Open University
Lebanon

Repetition

On many occasions, programmers need to perform certain tasks repeatedly. It can be tedious to perform such tasks in a sequential way.
It would be much easier if the programmer completes such tasks using loop instructions. Almost all programming languages facilitate the use of control loop statements to repeatedly execute a block of code until a certain condition is satisfied.

In this lecture, we will look at two broad categories of loops:

- count-controlled
- condition-controlled

A **count-controlled** loop repeats a specific number of times.
A **condition-controlled** loop uses a true/false condition to control the
number of times that it repeats.

In Python, you use the ***for*** statement to write a count-controlled loop, and you use the ***while*** statement to write a condition-

The For Loop

As we have said, a count-controlled loop iterates a specific number of times. In Python, we use the for statement to write a count-controlled loop.
A for loop is used for iterating over a sequence (a string, a list, a tuple, a range object, etc..), and execute a set of statements, once for each item in a sequence.

- The structure of a for loop is as follows:

```
for <variable name> in <sequence>:
    statements to be repeated
```

- **Example 1:**
- The following program will print the characters of the string:

```
for i in "Stars":
    print(i)
```

OUTPUT

```
S
t
a
r
s
```

- The following program will print Hello, along with the value of i, 5 times:

```
for i in range(5): #(range will be explained in the next slides)
    print('Hello', i)
```

OUTPUT

```
Hello 0
Hello 1
Hello 2
Hello 3
Hello 4
```

AOU
الجامعة العربية المفتوحة
Arab Open University
Lebanon

Using the range Function with the for Loop

- 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.

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.
step	Optional. An integer number specifying the incrementation. Default is 1

- The values we put in the **range** function determines how many times we will loop.
- The **range** function produces a list of numbers from zero (by default, unless other is specified) to the value minus one.

Using the range Function with the for Loop

Here is an example of a for loop that uses the range function:

```
for num in range(5):
    print(num)
```

```
0
1
2
3
4
```

Notice instead of using a list of values, we call to the **range** function passing 5 as an argument. In this statement, the **range** function will generate an iterable sequence of integers in the range of 0 up to (but not including) 5.

This code works the same as the following:

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

```
0
1
2
3
4
```

For Loop

Example 2 The program below asks the user for a number and prints its square. It does this three times and then prints: 'The loop is done'.

```
for i in range(3):
    num = eval(input('Enter a number: '))
    print ('The square of your number is', num*num)
print('The loop is now done.')
```

No indentation here; so, it is outside the loop

The output:

```
Enter a number: 3
The square of your number is 9
Enter a number: 5
The square of your number is 25
Enter a number: 23
The square of your number is 529
The loop is now done.
```

For Loop

Example 3 The program below will print A, then B, then it will C's and D's five times and then finish with the letter E once.

```
print('A')
print('B')
for i in range(5):
    print('C')
    print('D')
print('E')
```


```
A
B
C
D
C
D
C
D
C
D
C
D
E
```

More about the range function

- If we want the list of values to start at a value other than 0, we can do that by specifying the starting value.

  range(1,5) will produce the list 1, 2, 3, 4.

- Another thing we can do is to get the list of values to go up by more than one at a time. To do this, we can specify an optional step as the third argument.

  range(1,10,**2**) steps through the list by twos, producing 1, 3, 5, 7, 9.

- To get the list of values to go backwards, we can use a step of -1.

  range(5,1,**-1**) will produce the values 5, 4, 3, 2

Note that the **range** function stops one short of the ending value.

The range function

Examples:

for i in range(1,7):
    print (i, i**2, i**3, i**4)

```
1 for i in range(1,7):
2     print (i, i**2, i**3, i**4)
```

```
1 1 1 1
2 4 8 16
3 9 27 81
4 16 64 256
5 25 125 625
6 36 216 1296
```

for i in range(5,0,-1):
    print(i,end=‘ ')
print('Done!')

```
1 for i in range(5,0,-1):
2     print(i,end=' ')
3 print('Done!')
```

```
5 4 3 2 1 Done!
```

AOU
الجامعة العربية المفتوحة
Arab Open University
Lebanon

The range function

Example: The below codes show 2 different ways to print a string in reverse. One is done using slicing and the other is done using the range function.

Using Slicing

```
text = "Hello World"
revText= text[::-1]
print(revText)
```

OUTPUT

dlroW olleH

Using range

```
text= "Hello World"
for i in range(len(text)-1,-1,-1):
    print(text[i],end="")
```

0	1	2	3	4	5	6	7	8	9	10	Positive Indexing
H	e	l	l	o		W	o	r	l	d	
-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1	Negative Indexing

Recall that in Slicing, by leaving out ***start***, the range will start at the first character. By leaving out the ***stop***, the range will go to the end.

The range function

Example:

Write a program that prompts the user to enter 10 positive numbers, finds and prints the largest number entered by the user, using only one ***for*** loop.

N.B: Assume that the user will input only non-negative numbers.

Solution:

```
print('Enter 10 positive numbers: ')
largest = -1000
for i in range(10):
    num = int(input())
    if num>largest:
        largest = num
print('The largest number is:
',largest)
```

Sample output:

```
Enter 10 positive numbers:
12
2
3
4
22
0
4
6
7
6
The largest number is:  22
```

The while Loop

- We have already learned about for loops, which allow us to repeat things a specified number of times.
- Sometimes, though, we need to repeat something, but we don't know ahead of time exactly how many times it must be repeated. This introduces the need of using the condition-controlled loop.
- A condition-controlled loop causes a statement or set of statements to repeat as long as a condition is true.
- In Python, you use the **while** statement to write a condition-controlled loop.

The while Loop

Here is the general format of the **while** loop in Python:

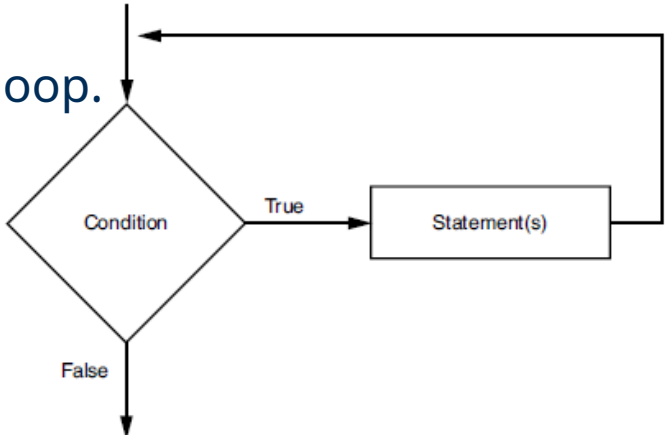
```
while condition:
  block
```

The while loop gets its name from the way it works:

while a condition is true, do some task.

The loop iterates while the condition evaluates to true; when the condition becomes false the loop terminates.

The loop has two parts:
(1) a **condition** that is tested for a true or false value, and
(2) a **block** (statement or set of statements) that is repeated if the condition is true.
The below Figure shows the logic of a while loop.

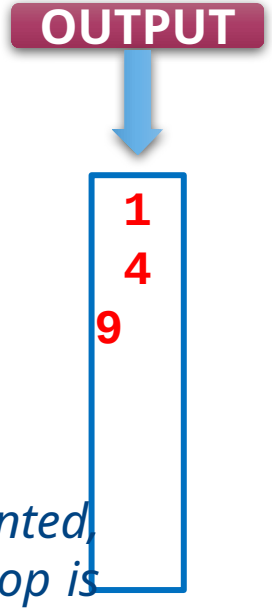


The `while` loop

```
while condition:
  block
```

Example:

```
x = 1
while x < 4:    # as long as x < 4...
   print (x**2)    # print the square of x
   x = x+1         # increment x by +1
```



Note that only the squares of 1, 2, and 3 are printed, because once x = 4, the condition is false, and the loop is terminated

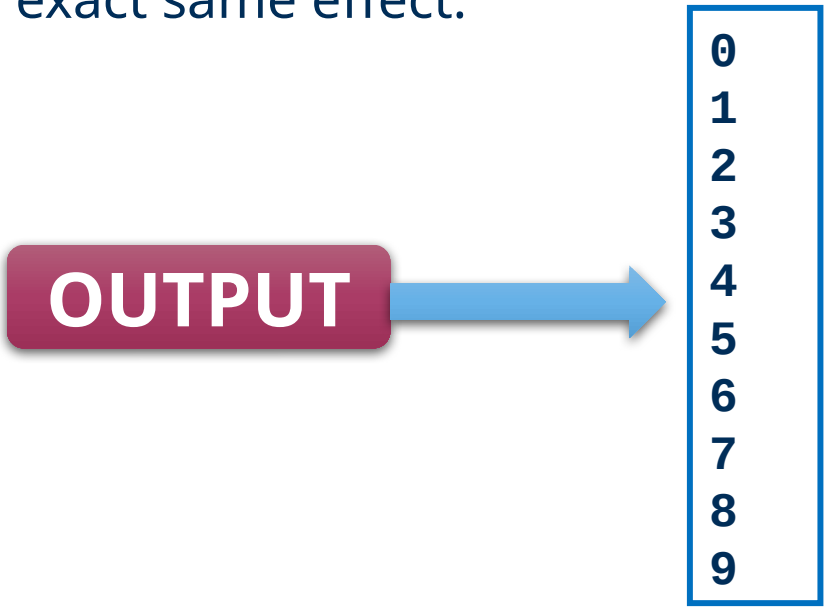
The `while` loop

The following `while` and `for` loops are equivalent

```
for i in range(10):
    print(i)
```

```
i=0
while i<10:
    print(i)
    i=i+1
```

They have the exact same effect:



The `while` loop

The while Loop Is a Pretest Loop

- The while loop is known as a pretest loop, which means it tests its condition before performing an iteration. Because the test is done at the beginning of the loop, you usually have to perform some steps prior to the loop to make sure that the loop executes at least once.

Example:

```
ans ="y"
# loop will repeat itself as long as  ans="y"
while ans=="y":
    sal=eval(input("Enter the salary: "))
    tax=0.05*sal
    print("The tax will be:",tax )
    ans=input('Do you want to enter another salary? (Enter y
for yes):')
print("Thank you!")
```

OUTPUT

```
Enter the salary: 500
The tax will be: 25.0
Do you want to enter another salary? (Enter y for
yes)y
Enter the salary: 1500
The tax will be: 75.0
Do you want to enter another salary? (Enter y for
yes)n
Thank you!
```

AOU
الجامعة العربية المفتوحة
Arab Open University
Lebanon

The `while` loop

Sentinel

- When processing a long sequence of values with a loop, perhaps a better technique is to use a sentinel.
A sentinel is a special value that marks the end of a sequence of items.
A sentinel value must be distinctive enough that it will not be mistaken as a regular value in the sequence.
When a program reads the sentinel value, it knows it has reached the end of the sequence, so the loop terminates.

```
# loop will repeat itself as long as salary>0
sal=eval(input("Enter the salary, any nonpositive number to
stop: "))
while sal >0:
    tax=0.05*sal
    print("The tax will be:",tax )
    sal=eval(input('Enter the salary, any nonpositive number
to stop:'))
print("Thank you!")
```

OUTPUT

```
Enter the salary, any nonpositive number to stop:
125
The tax will be: 6.25
The net salary will be: 118.75
Enter the salary, any nonpositive number to stop: 0
Thank you!
```

The `while` loop

Input Validation

Input validation is the process of inspecting data that has been input to
a program, to make sure it is valid before it is used in a computation.
Input validation is commonly done with a loop that iterates as long as an input variable references bad data.

```
# loop will repeat itself as long as the entered value is invalid
grade=eval(input("Enter the grade (between 0 & 20 included) "))
while grade<0 or grade>20:
    print('Wrong entry!')
    grade=eval(input("Enter the grade (between 0 & 20 included) "))
fgrade=grade*5
print("The grade out of hundred is",fgrade )
print("Thank you!")
```

OUTPUT

```
Enter the grade (between 0 & 20 included) #21
Wrong entry!
Enter the grade (between 0 & 20 included) # 14
The grade out of hundred is 70
Thank you!
```

The `while` loop

Pitfall to avoid:

while statements are intended to be used with changing conditions.

- If the condition in a *while* statement does not change, the program will be in an indefinite loop until the user hits **ctrl-c**.
- Indefinite loops usually occur when the programmer forgets to write code inside the loop that makes the test

```
x = 1
while x == 1:
    print('Hello world‘)
                    # Indefinite loop! Python will keep printing
                            # “Hello world”  because x does not
change!
```

The while loop

- The optional **else** clause runs only if the loop exits normally (not by break)

```
x = 1
while x < 3 :
    print (x)
    x = x + 1
else:
    print ('The loop ended normally')
```


```
1
2
The loop ended normally
```

The while Loop

Example:

Write a program that reads the grades of several students in a class and calculates the grade-point average for that class. The number of students is unknown, and the program should stop when the user enters any negative number.

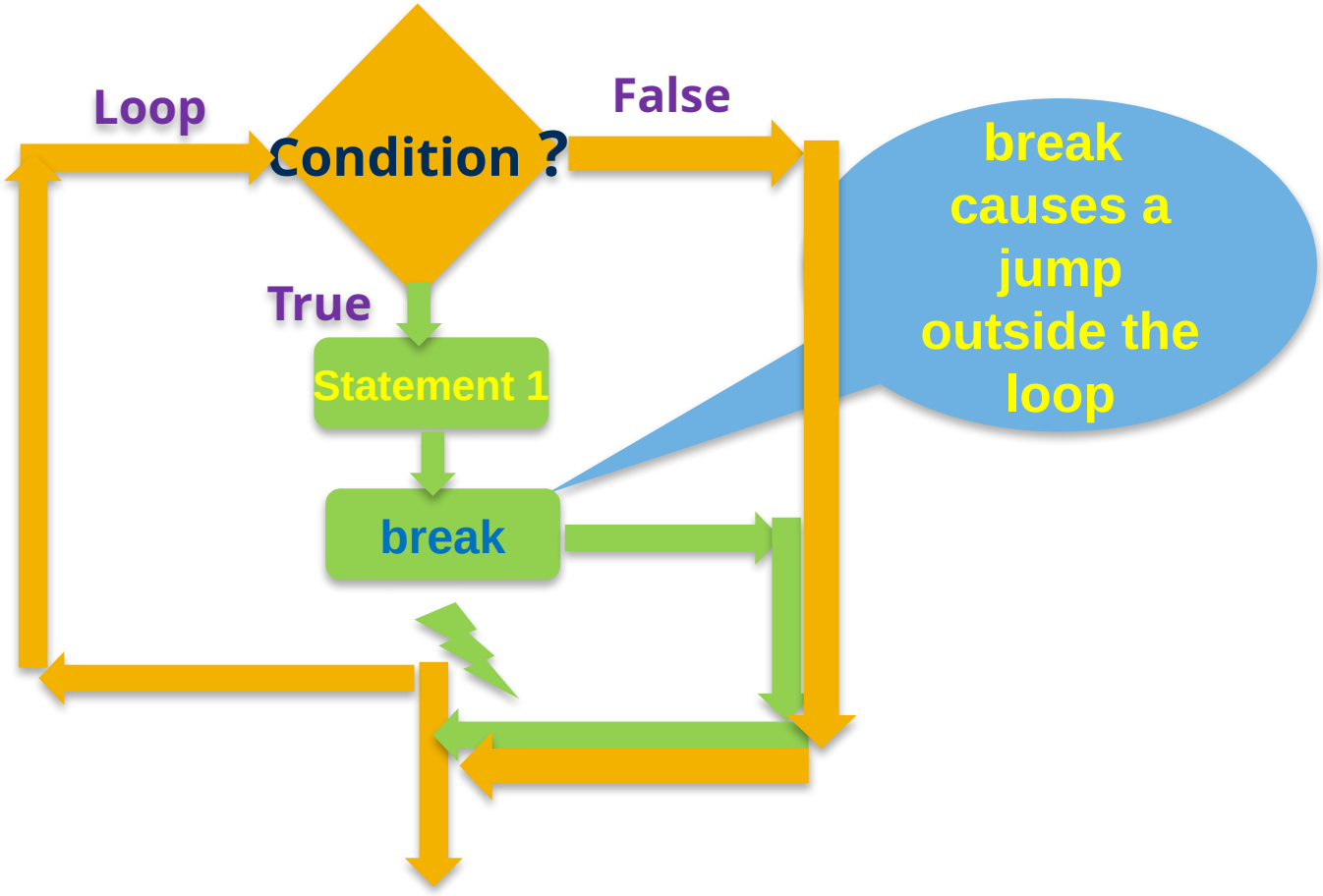
```
count = 0
Sum = 0
grade = float(input('Enter student grade; any negative number to stop:
'))
while grade >=0:
    Sum += grade
    count += 1
    grade = float(input('Enter student grade; any negative number to
stop: '))
average = Sum/count
print("The average is: ",format(average, '.2f'))
```

OUTPUT

```
Enter student grade; any negative number to stop: 15
Enter student grade; any negative number to stop: 12
Enter student grade; any negative number to stop: 16
Enter student grade; any negative number to stop: 0
Enter student grade; any negative number to stop: 5
Enter student grade; any negative number to stop: -15
The average is:  9.60
```

The break statement

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



The `break` Statement

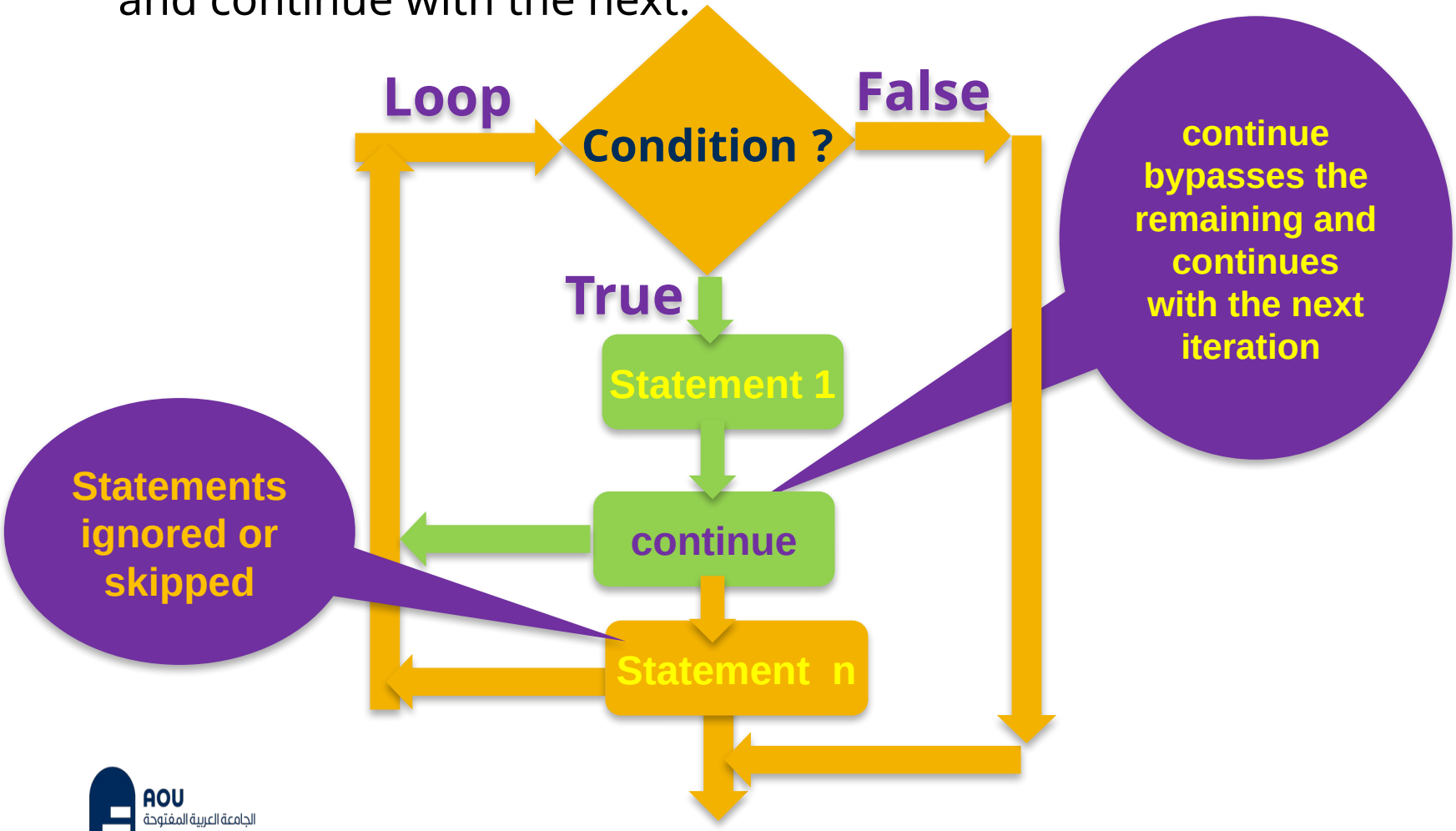
- Now, consider this program with **break**

```
x = 1
while x < 5 :
    print (x)
    x = x + 1
    break
else :
    print (The loop ended normally')
print("Done!")
```


```
1
Done!
```

The continue Statement

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



AOU
الجامعة العربية المفتوحة
Arab Open University
Lebanon

The continue Statement

- Now, consider this program with **continue**

```
x = 0
while x < 5 :
    x+=1
    if x==3:
        continue
    print(x)

else :
    print ('The loop ended
normally')
print("Done!")
```

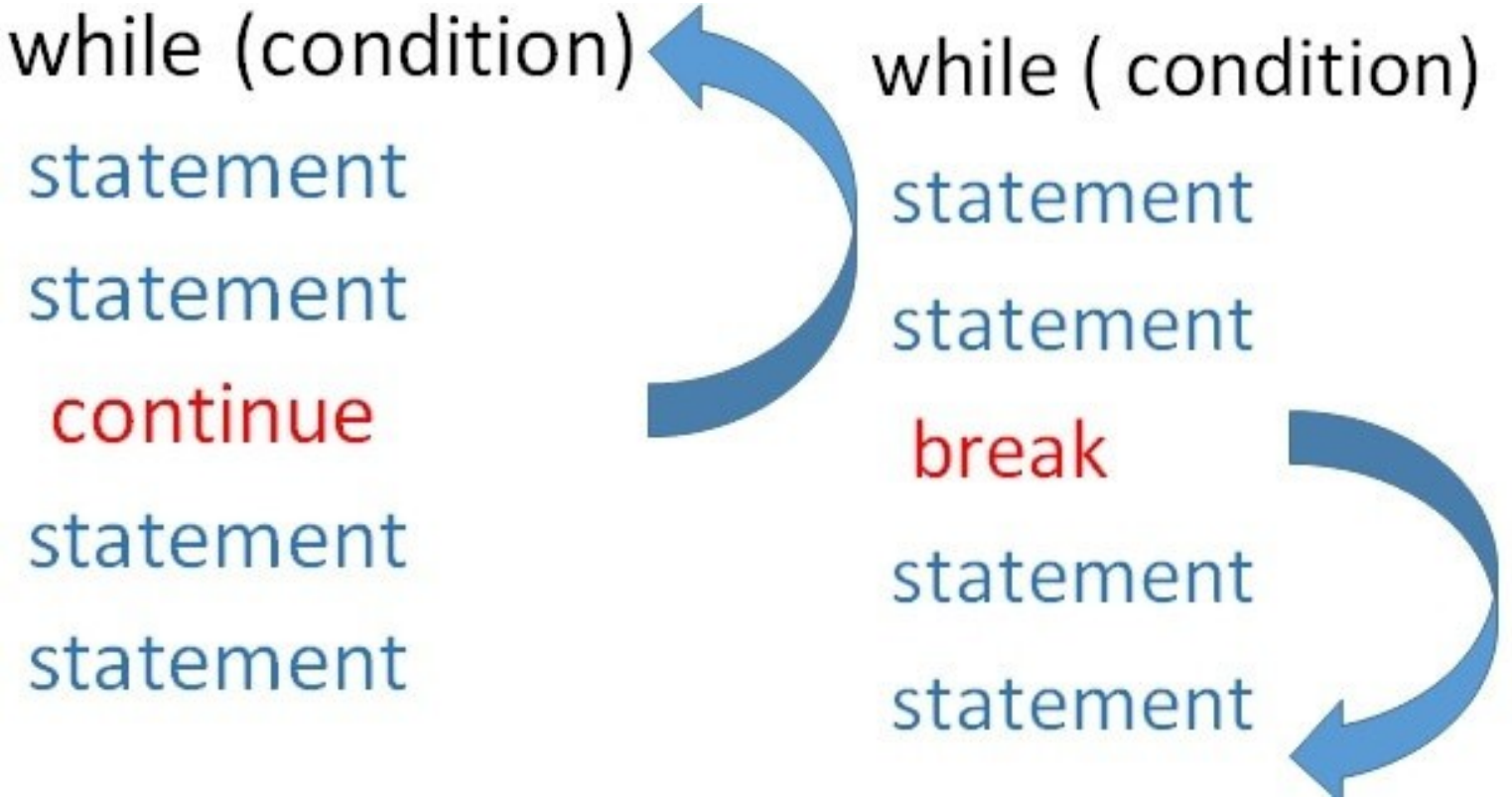

```
1
2
4
5
The loop ended normally
Done!
```

Notice how the number 3 hasn't been printed!

Comparison between **break** and **continue**

break	continue
It terminates the execution of remaining iteration of the loop.	It terminates only the current iteration of the loop.
'break' resumes the control of the program to the end of loop enclosing that 'break'.	'continue' resumes the control of the program to the next iteration of that loop enclosing 'continue'.
It causes early termination of loop.	It causes early execution of the next iteration.
'break' stops the continuation of loop.	'continue' does not stop the continuation of loop, it only stops the current iteration.

Comparison between break and continue



Comparison between **break** and **continue**

Example:

Exit the loop when i is 3:
break

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

OUTPUT

```
1
2
```

Continue to the next iteration if i is 3:
continue

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

OUTPUT

```
1
2
4
5
6
```

Nested loops

- A nested loop is an inner loop in the loop body of the outer loop. The inner or outer loop can be any type, such as a while loop or for loop. For example, the outer for loop can contain a while loop and vice versa.

- The outer loop can contain more than one inner loop. There is no limitation on the chaining of loops.
- In the nested loop, the number of iterations will be equal to the number of iterations in the outer loop multiplied by the iterations in the inner loop.
- In each iteration of the outer loop inner loop execute all its iteration.

- For each iteration of an outer loop the inner loop re-start and completes its execution before the outer loop can continue to its next iteration.

Nested loops

Nested for Loop

In Python, the for loop is used to iterate over a sequence.

Syntax of using a nested for loop in Python

```
# outer for loop
for element in sequence:
    # inner for loop
    for element in sequence:
        body of inner for loop
    body of outer for loop
```

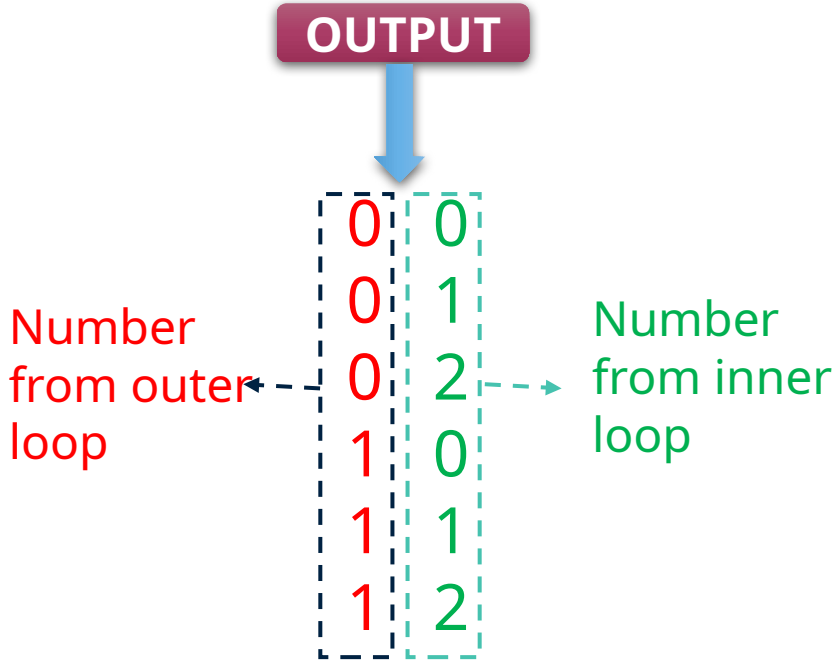
Nested loops

Example:

```
for i in range (2):
    for j in range (3):
        print(i," ",j)
```

Outer loop

Inner loop



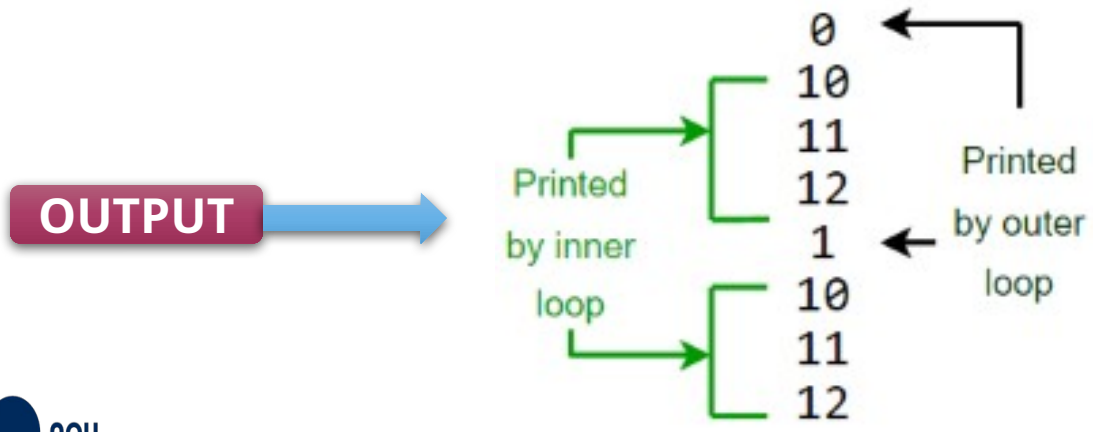
Nested loops

Example:

```
for i in range(2):
    print(i)
    for j in range(10,13):
        print(j)
```

Outer loop

Inner loop

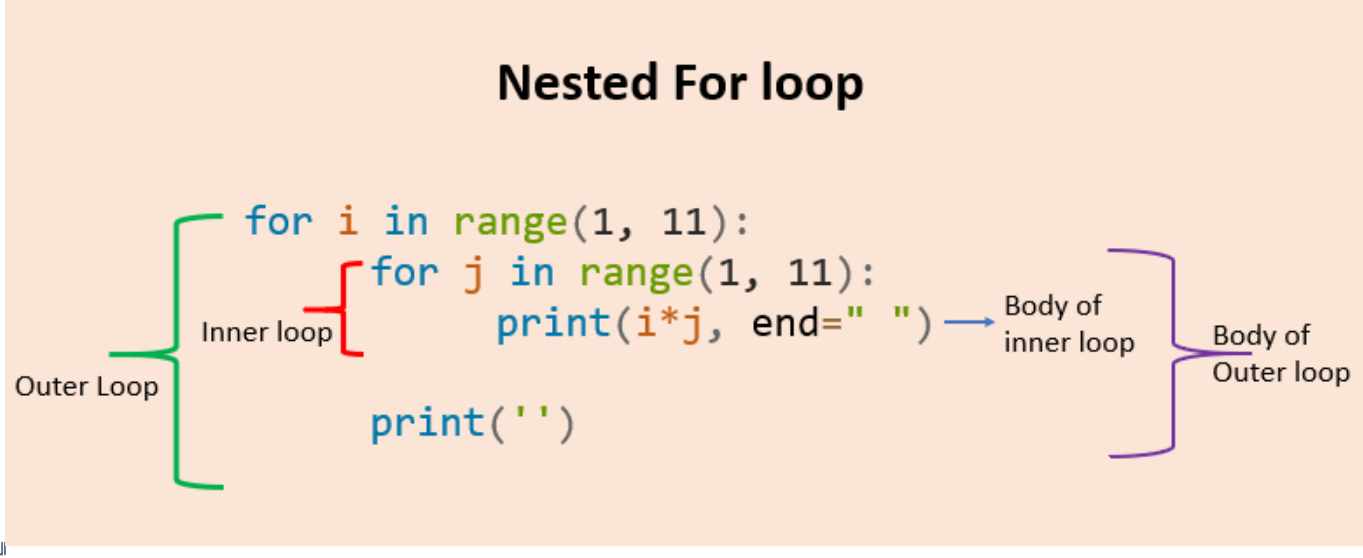


Nested loops

Nested for Loop- Example

```
for i in range(1,11):
    for j in
range(1,11):
        print(i*j, end="
")
    print('')
```

```
1 2 3 4 5 6 7 8 9 10
2 4 6 8 10 12 14 16 18 20
3 6 9 12 15 18 21 24 27 30
4 8 12 16 20 24 28 32 36 40
5 10 15 20 25 30 35 40 45 50
6 12 18 24 30 36 42 48 54 60
7 14 21 28 35 42 49 56 63 70
8 16 24 32 40 48 56 64 72 80
9 18 27 36 45 54 63 72 81 90
10 20 30 40 50 60 70 80 90 100
```



Nested loops

Nested while Loop

In Python, the while loop can be used to iterate over a sequence as well.

Syntax of using a nested while loop in Python

```
# outer while loop
while expression:
    # inner while loop
    while expression:
        body of inner while
loop
    body of outer while loop
```

Nested loops

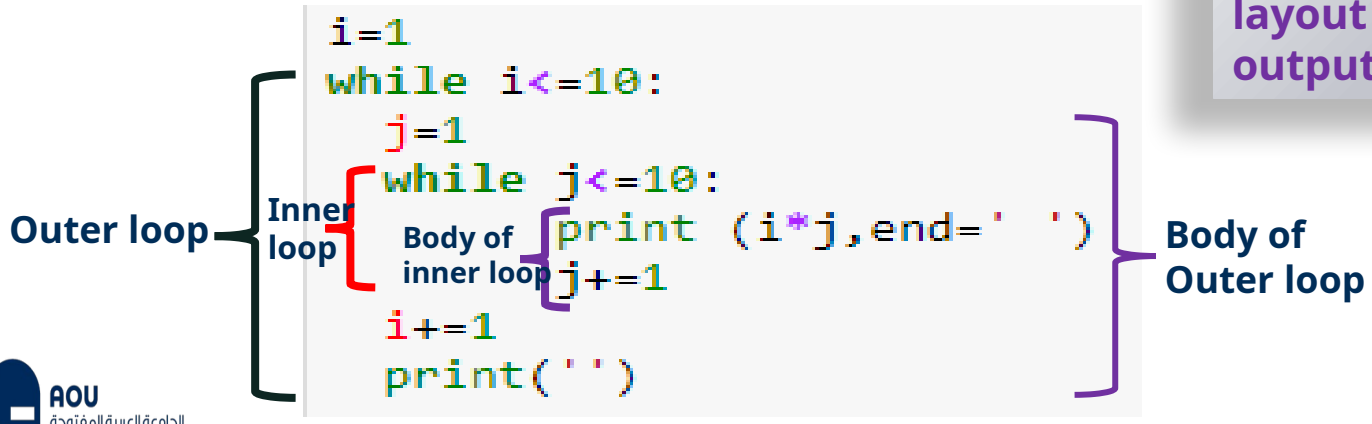
Nested while Loop- Example

```
i=1
while i<=10:
    j=1
    while j<=10:
        print (i*j,
end=' ')
        j+=1
    i+=1
    print('')
```

OUTPUT

```
1 2 3 4 5 6 7 8 9 10
2 4 6 8 10 12 14 16 18 20
3 6 9 12 15 18 21 24 27 30
4 8 12 16 20 24 28 32 36 40
5 10 15 20 25 30 35 40 45 50
6 12 18 24 30 36 42 48 54 60
7 14 21 28 35 42 49 56 63 70
8 16 24 32 40 48 56 64 72 80
9 18 27 36 45 54 63 72 81 90
10 20 30 40 50 60 70 80 90 100
```

Try to enhance the layout of the output!



Extra Exercises

1. Find the output of the following codes:

```
for i in range (3):
    for j in range (5):
        print("*",end=" ")
    print()
```

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

```
for i in range (4):
    for j in range (3):
        print(i+j,end=" ")
    print()
```

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

Extra Exercises

1. Find the output of the following codes:

```
for i in range (5):
    for j in range (i):
        print("*",end=" ")
    print()
```

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

Why this empty line?

```
for i in range (5):
    for j in range (i+1):
        print("*",end=" ")
    print()
```

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

```
for i in range(5,0,-1):
    for j in range(i):
        print('*', end=" ")
    print()
```

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

Extra Exercises

2. Write a program that prints the multiplication table for 6 as follows:
   1 x 6 = 6
   2 x 6 = 12
   3 x 6 = 18
   ...
   10 x 6 = 60

```
number = 6
for i in range(1, 11):
    result = i * number
    print(f"{i} x {number} =
{result}")
```

3. Write a program that prints all odd numbers from 1 to 100.

```
for num in range(1, 101, 2):
    print(num)
```

AOU
الجامعة العربية المفتوحة
Arab Open University
Lebanon

Extra Exercises

4. Write a program that prompts the user to enter 10 positive numbers, finds and prints the smallest number entered by the user. You should validat

Why None here?

```
minimum = None

for i in range(1, 5):
    number = eval(input(f"Enter positive number {i}: "))

    while number <= 0:
        print("Invalid input! Please enter a positive
number.")
        number = float(input(f"Enter positive number {i}:
"))

    if minimum==None or number < minimum:
        minimum = number

print("The smallest number entered is:", minimum)
```

Extra Exercises

5. Write a program that prompts the user to enter 5 numbers, counts and prints the number of positive numbers, negative numbers and zeros entered by the user.

```
positive_count = 0
negative_count = 0
zero_count = 0
for i in range(1, 6):
    number = float(input(f"Enter number {i}: "))
    if number > 0:
        positive_count += 1
    elif number < 0:
        negative_count += 1
    else:
        zero_count += 1
print("Positive numbers entered:",
positive_count)
print("Negative numbers entered:",
negative_count)
print("Zeroes entered:", zero_count)
```

AOU
الجامعة العربية المفتوحة
Arab Open University
Lebanon

Extra Exercises

6. Write a program that prompts the user to enter positive integers (0 to stop). The program should count and print the maximum integer and the minimum integer entered by the user.
7. Write a program that prompts the user to enter positive integers (0 to stop). The program then should count and print the number of odd integers and even integers entered by the user.
8. Write a program that reads from the user a sentence. The program should count the number of occurrences of the character ‘a’ in the sentence.
9. Write a program that reads from the user 10 words. The program then should count and print the number of words that start with character ‘a’.
10. Write a program that reads words (“finish” to stop). The program then should count and print the number of words that start with character ‘a’.
11. Write a program that reads from the user 10 words. The program then should find and print the longest word.
12. Write a program that reads from the user 10 words. The program then should find and print the shortest word.