Python Iteration

Introduction to Programming

Comp07027

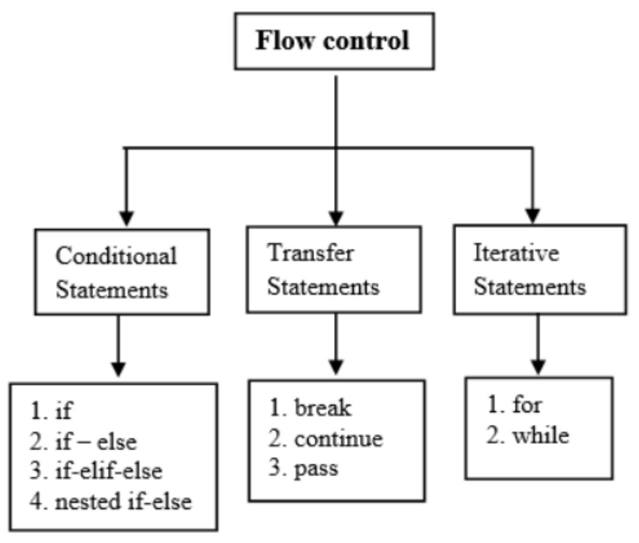
Lecture 5: For Loop

Dr Muhammad Aslam
Lecturer UWS Wuxi
Muhammad.Aslam@uws.ac.uk

Control Flow Statements

- The flow control statements are divided into **three categories**

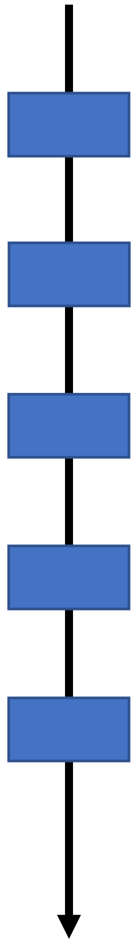
1. Conditional statements
2. Iterative statements.
3. Transfer statements



Python control flow statements

Sequence Statements

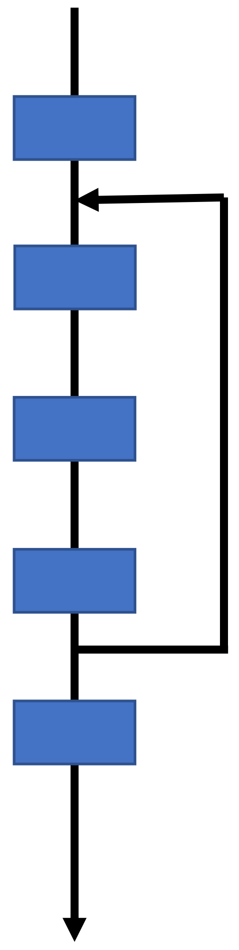
- This is the simplest construct.
- It means carrying out a set
- of instructions in a fixed order,
- and that order never changes.



Iteration

Iteration is sometimes called looping and, as the name suggests, involves performing some of the instructions more than once.

We can now look at how we do this.



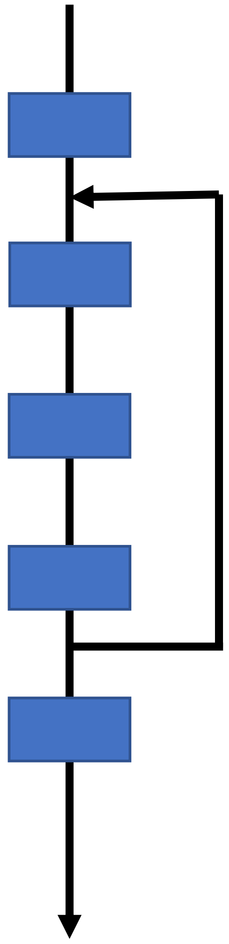
Python Iteration Types

There are two types loops in Python:

For loops

While loops

We'll start by looking at For loops



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).
- With the for loop we can execute a set of statements, once for each item in a list, tuple, set etc.
- Print each fruit in a fruit list:

-
  ```
  fruits = ["apple", "banana", "cherry"]
  for x in fruits:
    print(x)
  ```
- The for loop does not require an indexing variable to set beforehand.
  ```
  mylist = ['python', 'programming', 'examples', 'programs']
  for x in mylist:
    print(x)
  ```

-
  ```
  for i in range(4)
   print (mylist[i])
  ```

-
  ```
  for i in range(4)  # To print in same line
   print (mylist[i], end=' ' )
  ```

For Loop Example Program

- # Program to find the sum of all numbers stored in a list # List of numbers

```
numbers = [6, 5, 3, 8, 4, 2, 5, 4, 11]
# variable to store the sum
sum = 0
# iterate over the list
for val in numbers:
   sum = sum+val
 print("The sum is", sum)
```

For Loop for Tuple and Set

- Iterate through the items and print the values:

  -
    ```
    thistuple = ("apple", "banana", "cherry")
    for x in thistuple:
      print(x)
    ```

- Loop through the set, and print the values:

  -
    ```
    myset = {“Football", “Hockey", “Tennis”}
    for x in myset:
      print(x)
    ```

For Loop for Dictionary

- Print all key names in the dictionary, one by one:

  -
    ```
    mydict = {
      “class": “CS",
      “suject": “Programming",
      "year": 2021
    }
    ```

  -
    ```
    for x in mydict:
      print(x)
    for x in mydict.values():
      print(x)
    ```

  -
    ```
    for x in mydict.keys():
      print(x)
    ```

  -
    ```
    for x, y in thisdict.items():
      print(x, y)
    ```

For Loop for strings

- Even strings are iterable objects, they contain a sequence of characters:
- Loop through the letters in the word "banana":

-
  ```
  for x in "banana":
    print(x)
  ```

The 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
```

The break Statement Before print

- 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)
  ```

-
  ```
  number = [1, 5, 6, 8, 9, 10]
  for x in number:
    if x == 6:
      break
    print(x)
  ```

The continue Statement

- With the continue statement we can stop the current iteration of the loop, and continue with the next:
- Do not print banana:

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

-
  ```
  number = [1, 5, 6, 8, 9, 10]
  for x in number:
    if x == 6:
      continue
    print(x)
  ```

Range() function

We introduce the **range()** function.

The range() function allows us to access and use a sequence of numbers without having to type all of them in. By default it starts at 0 and increments by 1 unless we tell it different.

Here are some examples:

Range() function Example

<u>**Range()**</u>

Range is pretty flexible:

range(10) is 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

range(10) starts at **0**, stops one integer before **10** and increments by **1**

Range() function Flexible Range

Range is pretty flexible:

range(10)	is	0, 1, 2, 3, 4, 5, 6, 7, 8, 9
range(2,12)	is	2, 3, 4, 5, 6, 7, 8, 9, 10, 11

range(2,12) starts at **2**, stops one integer before **12** and increments by **1**

Range() function Flexible Range

Range()

Range is pretty flexible:

range(10) is 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

range(2,12) is 2, 3, 4, 5, 6, 7, 8, 9, 10,
11

range(2,12,3) is 2, 5, 8, 11

range(2,12,3) starts at **2**, stops one integer before **12** and increments by **3**

Range() function Flexible Range

Range()

Range is pretty flexible:

range(10) is 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

range(2,12) is 2, 3, 4, 5, 6, 7, 8, 9, 10, 11

range(2,12,3) is 2, 5, 8, 11

range(12,2,-2) is 12, 10, 8, 6, 4

range(12,2,-2) starts at **12**, stops one integer before **2** and decrements by **2**

Range() function Flexible Range

Range()

Range is pretty flexible:

range(10)	is	0, 1, 2, 3, 4, 5, 6, 7, 8, 9
range(2,12)	is	2, 3, 4, 5, 6, 7, 8, 9, 10, 11
range(2,12,3)	is	2, 5, 8, 11
range(12,2,-2)	is	12, 10, 8, 6, 4
range(7,-6,-3)	is	7, 4, 1, -2, -5

range(7,-6,-3) starts at **7**, stops one integer before **-6** and decrements by **3**

Use of Range() function in For Loop

The **range()** function is very useful when we want to create a (sometimes complicated) sequence of numbers.

The values in a range are all the same data type.

We're now going to use the range() function with a FOR loop.

For Loop using the Range() function

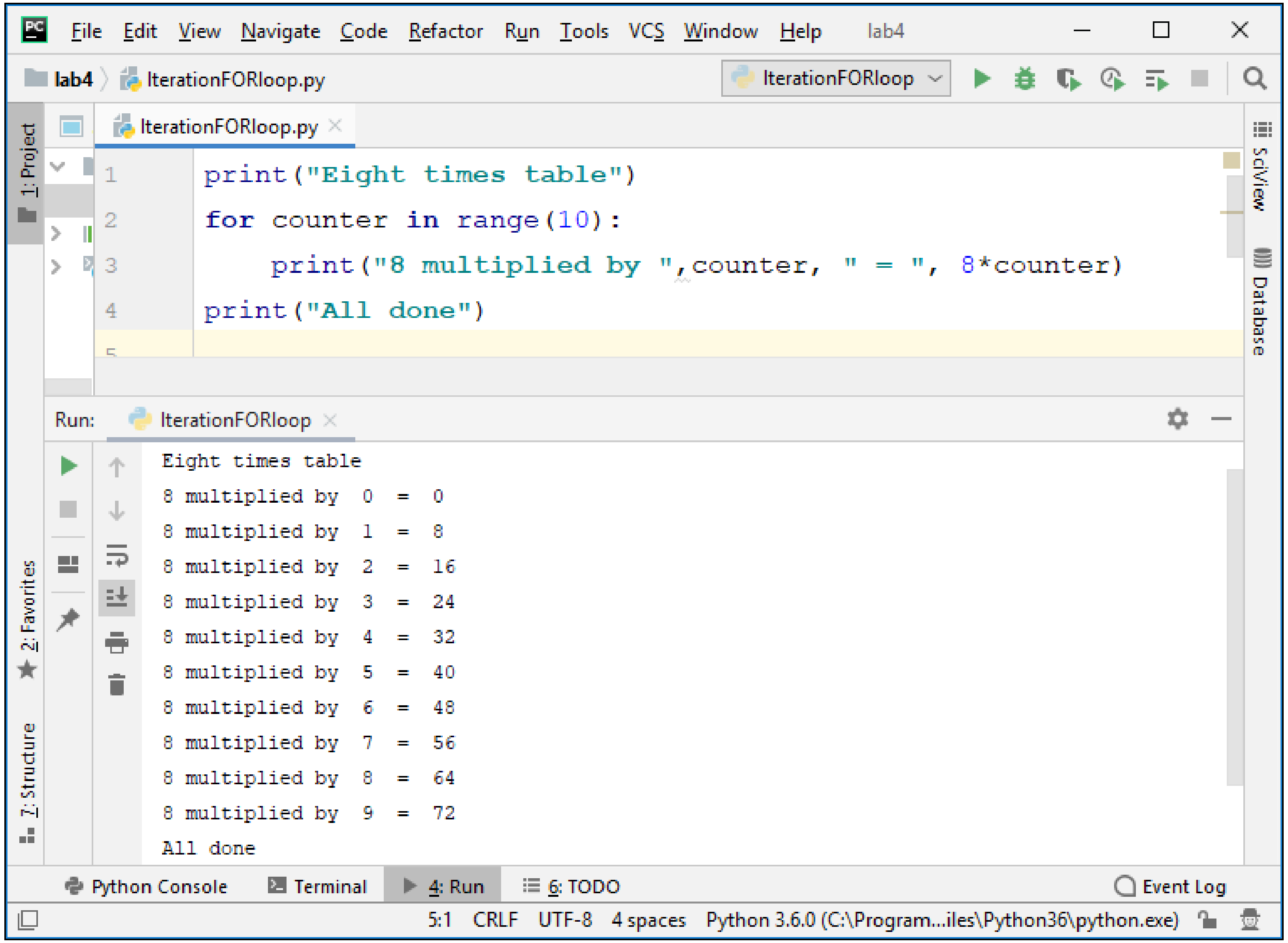
- The `range()` function returns a sequence of numbers, starting from 0 by default, and increments by 1 (by default), and ends at a specified number

-
  ```
  for x in range(20):
    print(x)
  ```
- Note that `range(20)` is not the values of 0 to 20, but the values 0 to 19
- Increment the sequence with 3 (default is 1)

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

Iteration using FOR loops

Let's have a look at the code in a bit more detail


File Edit View Navigate Code Refactor Run Tools VCS Window Help lab4
lab4 IterationFORloop.py
IterationFORloop
IterationFORloop.py
Run: IterationFORloop
Python Console
Terminal
4: Run
6: TODO
Event Log
5:1 CRLF UTF-8 4 spaces Python 3.6.0 (C:\Program...iles\Python36\python.exe)

```python
print("Eight times table")
for counter in range(10):
    print("8 multiplied by ",counter, " = ", 8*counter)
print("All done")
```

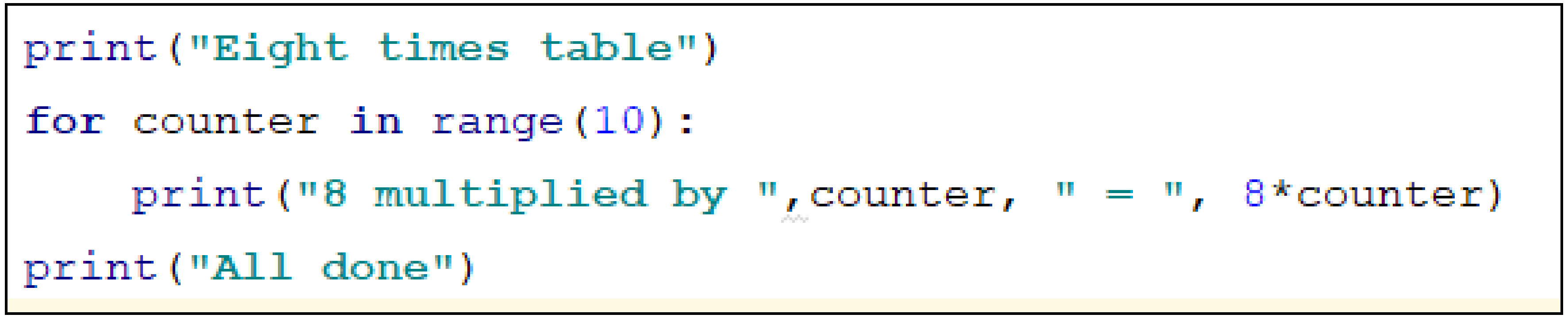
```
Eight times table
8 multiplied by  0  =  0
8 multiplied by  1  =  8
8 multiplied by  2  =  16
8 multiplied by  3  =  24
8 multiplied by  4  =  32
8 multiplied by  5  =  40
8 multiplied by  6  =  48
8 multiplied by  7  =  56
8 multiplied by  8  =  64
8 multiplied by  9  =  72
All done
```

Iteration using FOR loops

The header is output once

This is the ***FOR*** loop

The footer is output once



```
print("Eight times table")
for counter in range(10):
    print("8 multiplied by ",counter, " = ", 8*counter)
print("All done")
```

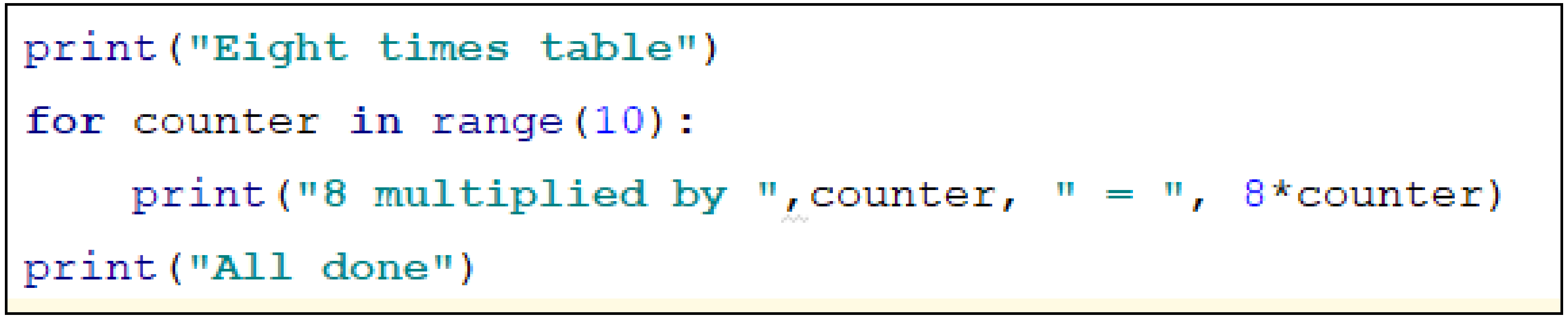
Iteration using FOR loops

counter is a dummy variable, which takes each of the values in ***range()*** in turn

The header is output once

This is the ***FOR*** loop

The footer is output once



```
print("Eight times table")
for counter in range(10):
    print("8 multiplied by ",counter, " = ", 8*counter)
print("All done")
```

Iteration using FOR loops

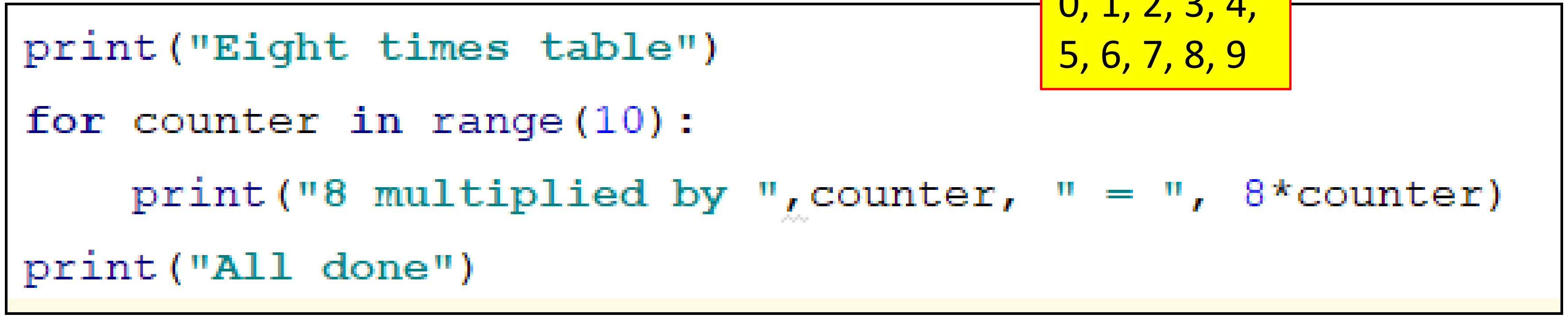
counter is a dummy variable, which takes each of the values in ***range()*** in turn

range(10) is 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

The header is output once

This is the ***FOR*** loop

The footer is output once



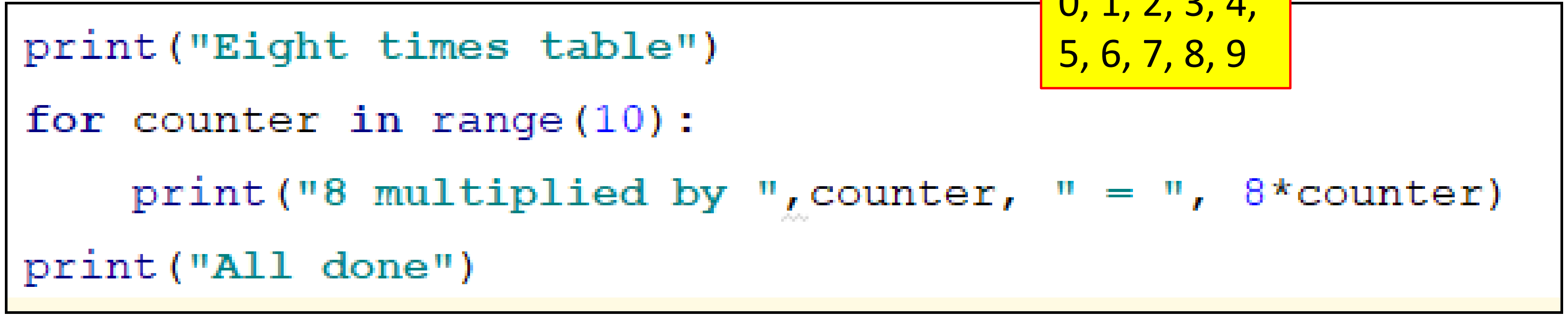
```
print("Eight times table")
for counter in range(10):
    print("8 multiplied by ",counter, " = ", 8*counter)
print("All done")
```

Iteration using FOR loops

counter is a dummy variable, which takes each of the values in ***range()*** in turn

The header is output once

range(10) is 0, 1, 2, 3, 4, 5, 6, 7, 8, 9



```
print("Eight times table")
for counter in range(10):
    print("8 multiplied by ",counter, " = ", 8*counter)
print("All done")
```

This is the ***FOR*** loop

The footer is output once

In this case the code within the ***FOR*** loop (indented code) is executed 10 times (because ***counter*** takes 10 values, 0 to 9)

Iteration using FOR loops

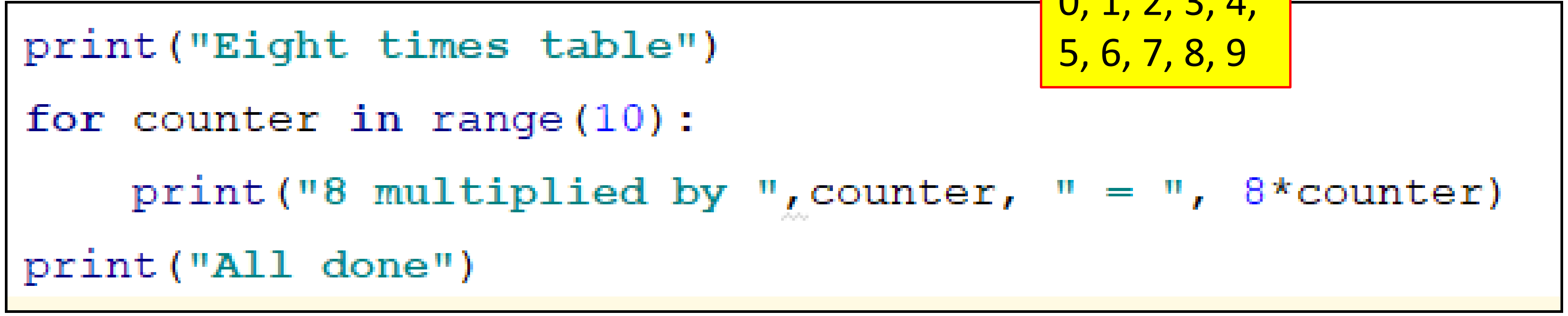
counter is a dummy variable, which takes each of the values in ***range()*** in turn

range(10) is 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

The header is output once

This is the ***FOR*** loop

The footer is output once



```
print("Eight times table")
for counter in range(10):
    print("8 multiplied by ",counter, " = ", 8*counter)
print("All done")
```

In this case the code within the ***FOR*** loop (indented code) is executed 10 times (because ***counter*** takes 10 values, 0 to 9)

The value of ***counter*** can simply be displayed, or used in a calculation

Else in For Loop

- With the continue statement we can stop the current iteration of the loop, and continue with the next:
- 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!")
  ```

Else in For Loop

- 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!")
```

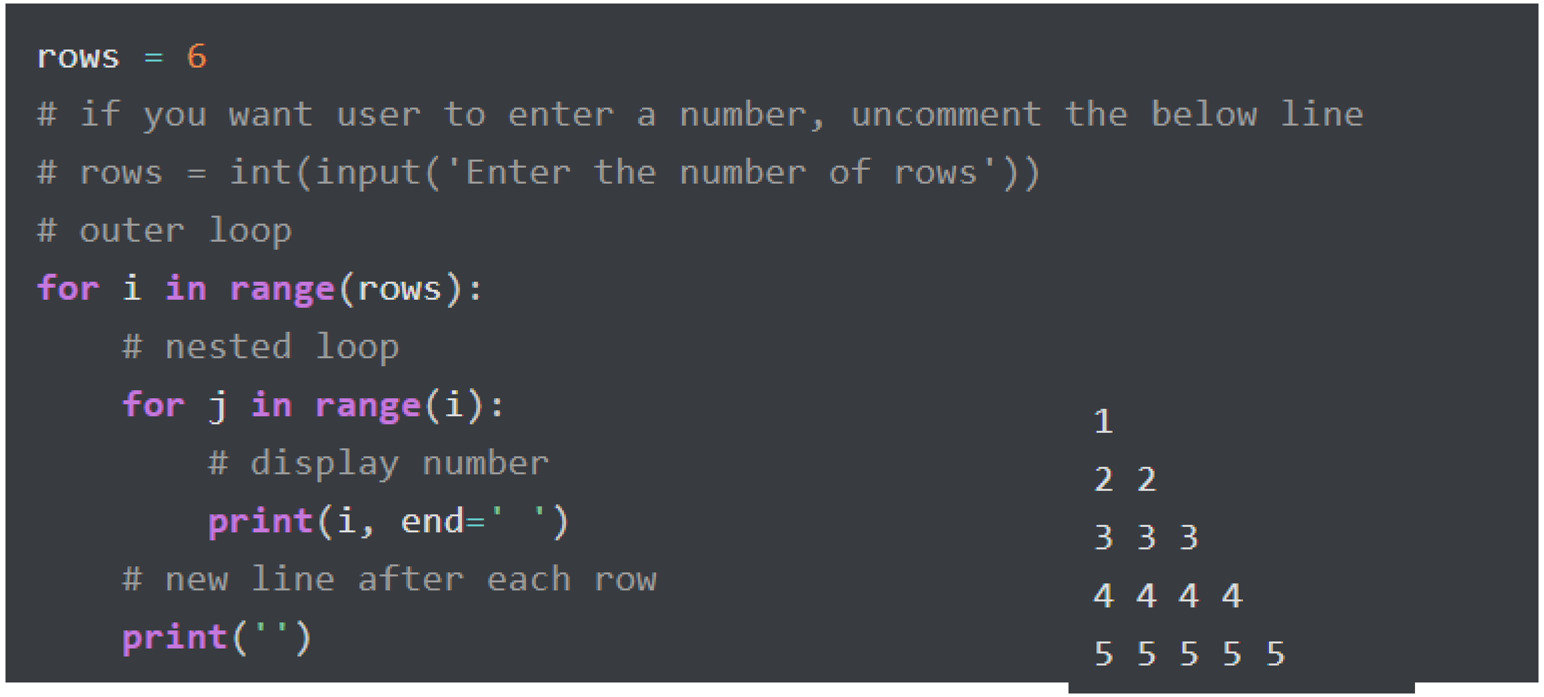
Nested Loops

- A nested loop is a loop inside a loop.
- The "inner loop" will be executed one time for each iteration of the "outer loop":
- 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)
```

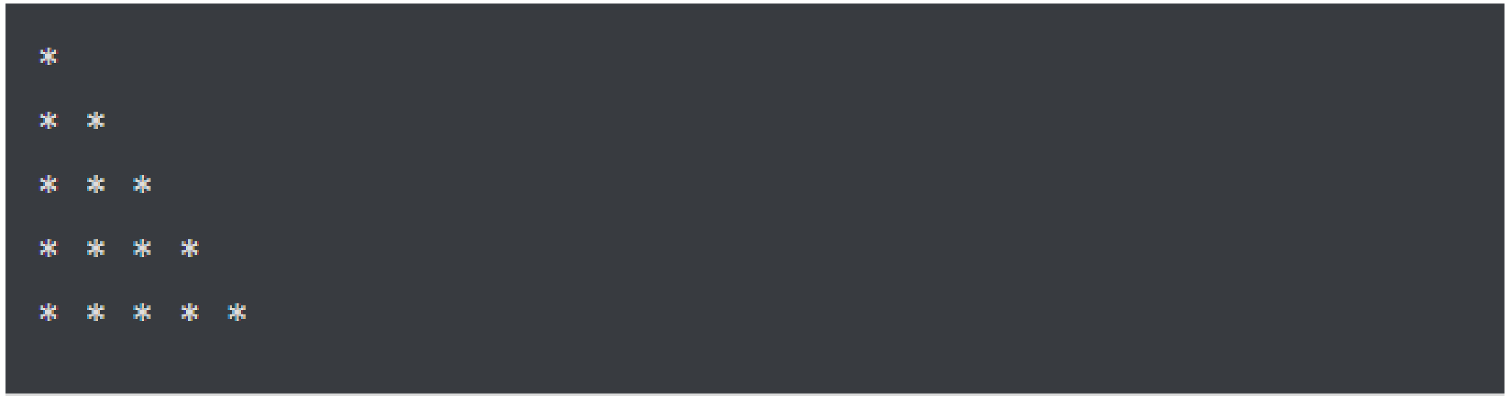
Nested Loops: Programs to print number pattern



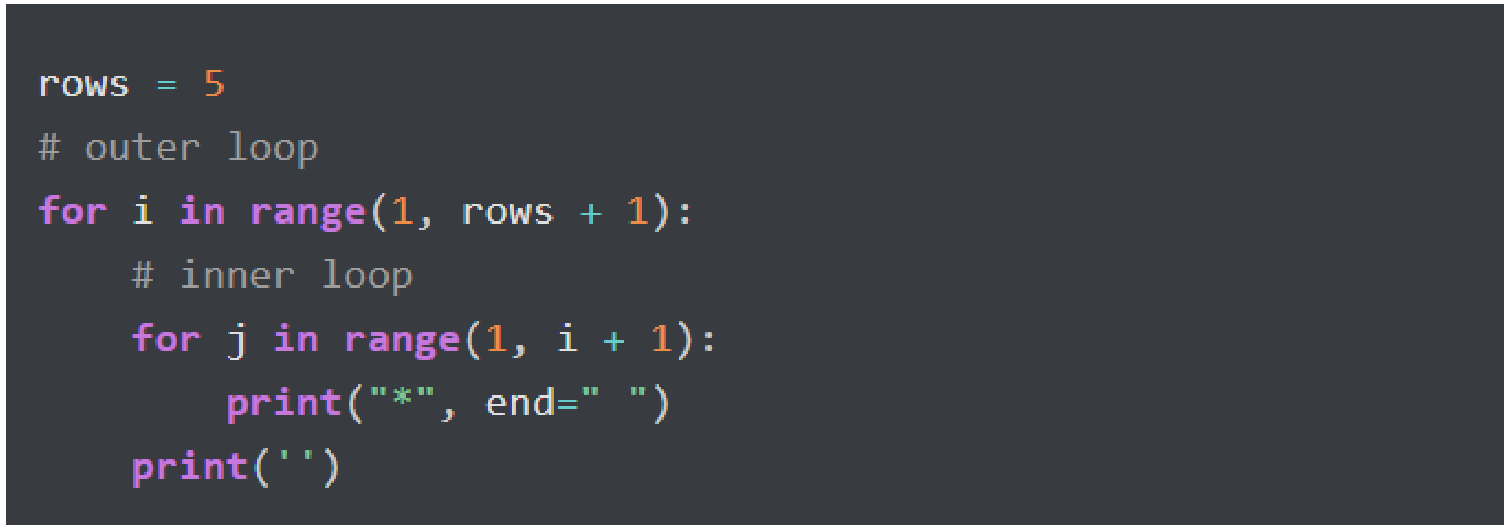
```
rows = 6
# if you want user to enter a number, uncomment the below line
# rows = int(input('Enter the number of rows'))
# outer loop
for i in range(rows):
    # nested loop
    for j in range(i):
        # display number
        print(i, end=' ')
    # new line after each row
    print('')
```

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

Nested Loops: Programs to print the star pattern

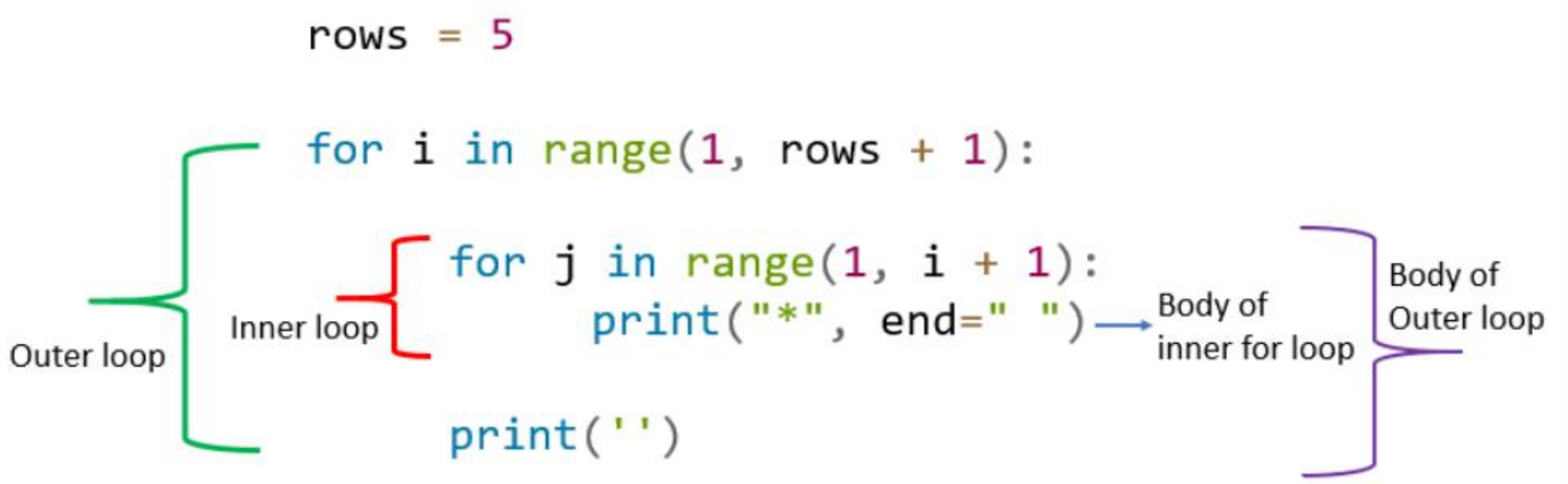


```
*
* *
* * *
* * * *
* * * * *
```



```python
rows = 5
# outer loop
for i in range(1, rows + 1):
    # inner loop
    for j in range(1, i + 1):
        print("*", end=" ")
    print('')
```

Nested Loops: How outer and inner loop work



```
rows = 5

for i in range(1, rows + 1):

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

    print('')
```

Nested Loops: Nested for loop to print the following pattern

- In this program, the outer loop is the number of rows print.
- The number of rows is five, so the outer loop will execute five times
- Next, the inner loop is the total number of columns in each row.
- For each iteration of the outer loop, the columns count gets incremented by 1
- In the first iteration of the outer loop, the column count is 1, in the next it 2. and so on.
- The inner loop iteration is equal to the count of columns.
- In each iteration of an inner loop, we print star

Backward Iteration using the reversed() function

- We can use the built-in function reversed() with for loop to change the order of elements, and this is the simplest way to perform a reverse looping.

```
list1 = [10, 20, 30, 40]
for num in reversed(list1):
     print(num)
```

Reverse for loop using range()

```
num = 5 #
start = 5 # stop = -1 # step = -1
for num in (range(num, -1, -1)):
print(num)
```

Questions??