**Python Loops**

**Overview**

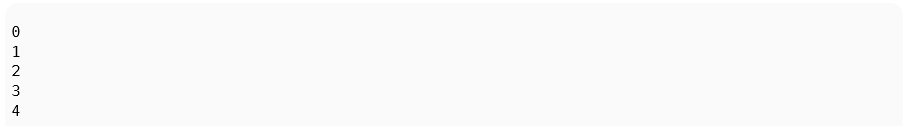
|  |
| --- |
| Loops enable developers to set certain portions of their code to repeat through a number of loops which are referred to as iterations. This handout covers using multiple types of loops and applications of loops in Python. |

**For loops**

For loops iterate over a collection of items, such as list or dict, and run a block of code with each element from the collection.



The above for loop iterates over a list of numbers. Each iteration sets the value i to the next element of the list. So first it will be 0, then 1, then 2, etc. The output is as follows:



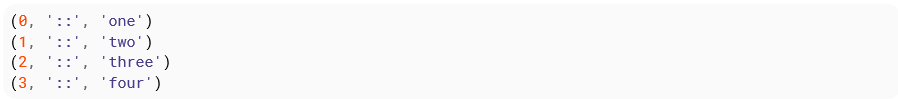
***range*** is a function that returns a series of numbers. It can be used in the for loops:

Gives the exact same result as the first for loop. Note that 5 is not printed as the range here is the first five numbers counting from 0.

If you want to loop through both the elements of a list and have an index for the elements as well, you can use Python’s **enumerate** function:

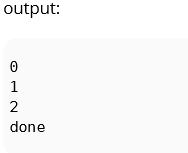
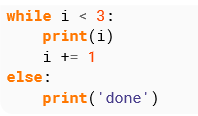
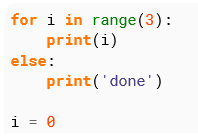


The above loop will print:



**Loop with an “else” clause**

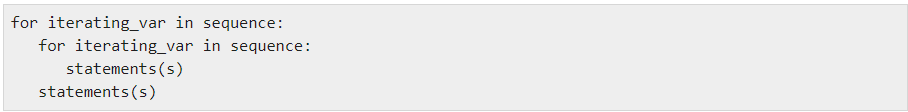
The for and while compound statements(loops) can optionally have an else clause (in practice, this usage is fairly rare). The **else** clause only executes after a **for** loop terminates by iterating to completion, or after a **while** loop terminates by its conditional expression becoming false. Have a look at the examples(s) below. Assume i is initialized to zero.



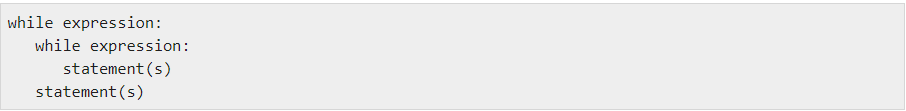
The else clause does not execute if the loop terminates some other way (through a break statement or by raising an exception)

Python programming language allows you to use one loop inside another loop. Following section shows a few examples to illustrate the concept.

**Nested Loop Syntax**



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



A final note on loop nesting is that you can put any type of loop inside of any other type of loop. For example a for loop can be inside a while loop or vice versa.

Nested loops are useful when for each pass through the outer loop, you need to repeat some action on the data in the outer loop. For example, you read a file line by line and for each line you must count how many times the word “the” is found

**The Pass Statement**

**pass** is a null statement for when a statement is required by Python syntax (such as within the body of a for or while loop), but no action is required by the programmer. This can be useful as a placeholder for code that is yet to be written.

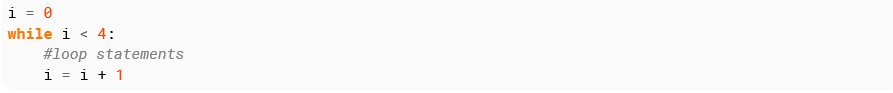
pass can be used in while loops, as well as in selections and function definition etc.



In the example above, nothing will happen. The while loop will complete without error, but no commands or code will be actioned.

**While Loop**

A while loop will cause the loop statements to be executed until the loop condition is falsey. The following code will execute the loop statements a total of 4 times.

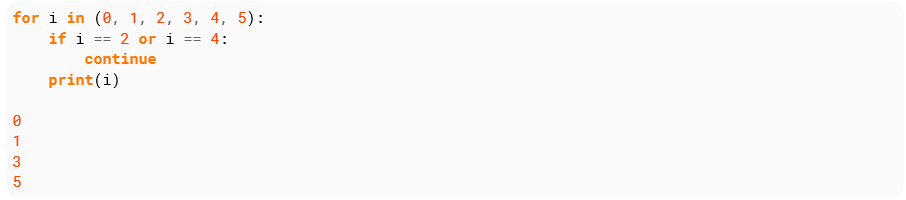


while loops can also run without a condition by using numbers(complex or real) or True.

If the condition is always true the while loop will run forever (infinite loop) if it is not terminated by a break or return statement or an exception.

**Continue statement**

A continue statement will skip to the next iteration of the loop bypassing the rest of the current block but continuing the loop. As with break, continue can only appear inside loops:



Note that 2 and 4 are not printed, this is because continue goes to the next iteration instead of continuing on to print(i) when i == 2 or i == 4.

**Exercise 1**

* Write a short program that prints each number from 1 to 100 on a new line.
* For each number, if it is divisible by 3, print “Fizz” instead of the number.
* If it’s divisible by 5, print “Buzz” instead of the number.
* For numbers which are divisible by both 3 and 5, print “FizzBuzz” instead of the number.

**Sample Output:**

1

2

Fizz

4

Buzz

.

.

etc

**Task 1**

Write a program that generates a random number (0-10) and ask you to guess it. You have three asserts.

* Define a random number between 0 - 10.
* Initialize **guesses\_left** to 3.
* Use a while loop to let the user keep guessing so long as guesses\_left is greater than zero
* Use an else: case after your while loop to print: You lose.

**Task 2**

Write a Python program to construct the following pattern, using a nested for loop.

\*

\* \*

\* \* \*

\* \* \* \*

\* \* \* \* \*

\* \* \* \*

\* \* \*

\* \*

\*