**CONDITIONALS :**

A way to check for something. One of the example of conditionals is

1. if (CONDITIONAL IS TRUE)statement -] Do this

Example

age = 19

if age >= 18:

print("You are old enough to vote!")

print("Have you registered to vote yet?")

The conditional test passes, and both print statements are indented, so both lines are printed:

You are old enough to vote!

Have you registered to vote yet?

If the value of age is less than 18, this program would produce no output.

1. else - Now comes the else statement , it works great with the if statement . It is used when anything is passed by the ‘if’ statement or doesn’t meet the criteria

Catches everything that does NOT meet prior conditionals.

if (….)

……

else:

……

Example

We’ll display the same message we had previously if the person is old enough to vote, but this time we’ll add a message for anyone who is not old enough to vote:

age = 17

if age >= 18:

print("You are old enough to vote!")

print("Have you registered to vote yet?")

else:

print("Sorry, you are too young to vote.")

print("Please register to vote as soon as you turn 18!")

If the conditional test at u passes, the first block of indented print statements is executed. If the test evaluates to False, the else block at v is executed. Because age is less than 18 this time, the conditional test fails and the code in the else block is executed:

Sorry, you are too young to vote.

Please register to vote as soon as you turn 18!

This code works because it has only two possible situations to evaluate:

a person is either old enough to vote or not old enough to vote. The if-else structure works well in situations in which you want Python to always execute one of two possible actions. In a simple if-else chain like this, one of the two actions will always be executed.

1. elif – Combination of else and if. It comes AFTER if statement means it fit right in b/w else and if. It’s used when some other more conditionals are need to be checked.

It Setsup another conditionals.

Example

age = 12

if age < 4:

print("Your admission cost is $0.")

elif age < 18:

print("Your admission cost is $5.")

else:

print("Your admission cost is $10.")

The if test at u tests whether a person is under 4 years old. If the test passes, an appropriate message is printed and Python skips the rest of the tests. The elif line at v is really another if test, which runs only if the previous test failed. At this point in the chain, we know the person is at least 4 years old because the first test failed. If the person is less than 18, an appropriate message is printed and Python skips the else block. If both the if and elif tests fail, Python runs the code in the else block at w. In this example the test at u evaluates to False, so its code block is not executed. However, the second test evaluates to True (12 is less than 18) so its code is executed. The output is one sentence, informing the user of the admission cost:

Your admission cost is $5.

Any age greater than 17 would cause the first two tests to fail. In these situations, the else block would be executed and the admission price would be $10.

**LOOPS :**

Loops are the way to repeat a single action over and over.

Ex : real world example of loop ‘ll be wash a dish over and over. U wash a single dish then u grab another dish to wash.

# **while Loop Statements**

1. **while** : (condition should be true)

A while loop implements the repeated execution of code based on a given [Boolean](https://www.digitalocean.com/community/tutorials/understanding-boolean-logic-in-python-3) condition. The code that is in a while block will execute as long as the while statement evaluates to True.

You can think of the while loop as a repeating [conditional statement](https://www.digitalocean.com/community/tutorials/how-to-write-conditional-statements-in-python-3-2). After an if statement, the program continues to execute code, but in a while loop, the program jumps back to the start of the while statement until the condition is False.

As opposed to [**for loops**](https://www.digitalocean.com/community/tutorials/how-to-construct-for-loops-in-python-3) that execute a certain number of times, while loops are conditionally based, so you don’t need to know how many times to repeat the code going in.

Syntax

The syntax of a **while** loop in Python programming language is −

while expression:

statement(s)

### Examples

Let’s create a small program that executes a while loop. In this program, we’ll ask for the user to input a password. While going through this loop, there are two possible outcomes:

* If the password is correct, the while loop will exit.
* If the password is not correct, the while loop will continue to execute.

We’ll create a file called password.py in our text editor of choice, and begin by initializing the variable password as an empty [string](https://www.digitalocean.com/community/tutorial_series/working-with-strings-in-python-3):

password.py

password = ''

The empty string will be used to take in input from the user within the while loop.

Now, we’ll construct the while statement along with its condition:

password.py

password = ''

**while** password != 'password':

Here, the while is followed by the variable password. We are looking to see if the variable password is set to the string password (based on the user input later), but you can choose whichever string you’d like.

This means that if the user inputs the string password, then the loop will stop and the program will continue to execute any code outside of the loop. However, if the string that the user inputs is not equal to the string password, the loop will continue.

Next, we’ll add the block of code that does something within the while loop:

password.py

password = ''

**while** password != 'password':

print('What is the password?')

password = input()

Inside of the while loop, the program runs a print statement that prompts for the password. Then the variable password is set to the user’s input with the input() function.

The program will check to see if the variable password is assigned to the string password, and if it is, the while loop will end. Let’s give the program another line of code for when that happens:

password.py

password = ''

**while** password != 'password':

print('What is the password?')

password = input()

print('Yes, the password is ' + password + '. You may enter.')

The last print() statement is outside of the while loop, so when the user enters password as the password, they will see the final print statement outside of the loop.

However, if the user never enters the word password, they will never get to the last print() statement and will be stuck in an infinite loop.

An **infinite loop** occurs when a program keeps executing within one loop, never leaving it. To exit out of infinite loops on the command line, press CTRL + C.

Save the program and run it:

You’ll be prompted for a password, and then may test it with various possible inputs. Here is sample output from the program:

Output

What is the password?

hello

What is the password?

sammy

What is the password?

PASSWORD

What is the password?

password

Yes, the password is password. You may enter.

Keep in mind that strings are case sensitive unless you also use a [string function](https://www.digitalocean.com/community/tutorials/an-introduction-to-string-methods-in-python-3) to convert the string to all lower-case (for example) before checking.

### Example

count = 0

while (count < 9):

print 'The count is:', count

count = count + 1

print "Good bye!"

When the above code is executed, it produces the following result −

The count is: 0

The count is: 1

The count is: 2

The count is: 3

The count is: 4

The count is: 5

The count is: 6

The count is: 7

The count is: 8

Good bye!

### Example

#!/usr/bin/python

count = 10

while (count > 0):

print 'The count is:', count

count = count - 1

print "Good bye!"

When the above code is executed, it produces the following result −

The count is: 10

The count is: 9

The count is: 8

The count is: 7

The count is: 6

The count is: 5

The count is: 4

The count is: 3

The count is: 2

The count is: 1

Good bye!

## **The Infinite Loop**

A loop becomes infinite loop if a condition never becomes FALSE. You must use caution when using while loops because of the possibility that this condition never resolves to a FALSE value. This results in a loop that never ends. Such a loop is called an infinite loop.

An infinite loop might be useful in client/server programming where the server needs to run continuously so that client programs can communicate with it as and when required.

#!/usr/bin/python

var = 1

while var == 1 : # This constructs an infinite loop

num = raw\_input("Enter a number :")

print "You entered: ", num

print "Good bye!"

When the above code is executed, it produces the following result −

Enter a number :20

You entered: 20

Enter a number :29

You entered: 29

Enter a number :3

You entered: 3

Enter a number between :Traceback (most recent call last):

File "test.py", line 5, in <module>

num = raw\_input("Enter a number :")

KeyboardInterrupt

Above example goes in an infinite loop and you need to use CTRL+C to exit the program.

## **Using else Statement with Loops**

Python supports to have an **else** statement associated with a loop statement.

* If the **else** statement is used with a **for** loop, the **else** statement is executed when the loop has exhausted iterating the list.
* If the **else** statement is used with a **while** loop, the **else** statement is executed when the condition 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 else statement gets executed.p>

#!/usr/bin/python

count = 0

while count < 5:

print count, " is less than 5"

count = count + 1

else:

print count, " is not less than 5"

When the above code is executed, it produces the following result −

0 is less than 5

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 Statements**

It has ability to iterate over the item of any sequence , such as ‘list’ or a ‘string’.

### Syntax

for iterating\_var in sequence:

statements(s)

If a sequence contains an expression list, it is evaluated first. Then, the first item in the sequence is assigned to the iterating variable *iterating\_var*. Next, the statements block is executed. Each item in the list is assigned to *iterating\_var*, and the statement(s) block is executed until the entire sequence is exhausted.

### Example

#!/usr/bin/python

for letter in **'Python':** # First Example

print 'Current Letter :', letter

fruits = ['banana', 'apple', 'mango']

for fruit in **fruits**: # Second Example

print 'Current fruit :', fruit

print "Good bye!"

When the above code is executed, it produces the following result −

Current Letter : P

Current Letter : y

Current Letter : t

Current Letter : h

Current Letter : o

Current Letter : n

Current fruit : banana

Current fruit : apple

Current fruit : mango

Good bye!

## **Iterating by Sequence Index**

An alternative way of iterating through each item is by index offset into the sequence itself. Following is a simple example −

#!/usr/bin/python

fruits = ['banana', 'apple', 'mango']

for index in range(len(fruits)):

print 'Current fruit :', fruits[index]

print "Good bye!"

When the above code is executed, it produces the following result −

Current fruit : banana

Current fruit : apple

Current fruit : mango

Good bye!

Here, we took the assistance of the len() built-in function, which provides the total number of elements in the tuple as well as the range() built-in function to give us the actual sequence to iterate over.

**range** : now combine the power of for loop with range. A range is a key word function that creates a sequence of numbers and by default it goes from 0 up to, but not including the last value you put in.

Ex – range(6)

range(2,7)

range(10,20,2)

Remember range takes upper bound one less and lower bound from the same point of initialization.

print range(6) # here 6 is behaving as upper bound

print range(2,7) # and 2 is behaving as lower bound

print range(10,20,2)

OUTPUT

[0,1,2,3,4,5]

[2,3,4,5,6]

[10,12,14,16,18]

## **Loop Control Statements**

Loop control statements change execution from its normal sequence. When execution leaves a scope, all automatic objects that were created in that scope are destroyed.

Python supports the following control statements. Click the following links to check their detail.

|  |  |
| --- | --- |
| **Control Statement** | **Description** |
| [**break statement**](https://www.tutorialspoint.com/python/python_break_statement.htm) | Terminates the loop statement and transfers execution to the statement immediately following the loop. |
| [**continue statement**](https://www.tutorialspoint.com/python/python_continue_statement.htm) | Causes the loop to skip the remainder of its body and immediately retest its condition prior to reiterating. |
| [**pass statement**](https://www.tutorialspoint.com/python/python_pass_statement.htm) | The pass statement in Python is used when a statement is required syntactically but you do not want any |

# **Python break statement**

It terminates the current loop and resumes execution at the next statement, just like the traditional break statement in C.

The most common use for 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.

### Syntax

The syntax for a **break** statement in Python is as follows −

break

### Example

#!/usr/bin/python

for letter in 'Python': # First Example

if letter == 'h':

break

print 'Current Letter :', letter

var = 10 # Second Example

while var > 0:

print 'Current variable value :', var

var = var -1

if var == 5:

break

print "Good bye!"

When the above code is executed, it produces the following result −

Current Letter : P

Current Letter : y

Current Letter : t

Current variable value : 10

Current variable value : 9

Current variable value : 8

Current variable value : 7

Current variable value : 6

Good bye!

# **Python continue statement**

It returns the control to the beginning of the while loop.. The **continue**statement rejects all the remaining statements in the current iteration of the loop and moves the control back to the top of the loop.

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

### Syntax

continue

### Example

#!/usr/bin/python

for letter in 'Python': # First Example

if letter == 'h':

continue

print 'Current Letter :', letter

var = 10 # Second Example

while var > 0:

var = var -1

if var == 5:

continue

print 'Current variable value :', var

print "Good bye!"

When the above code is executed, it produces the following result −

Current Letter : P

Current Letter : y

Current Letter : t

Current Letter : o

Current Letter : n

Current variable value : 9

Current variable value : 8

Current variable value : 7

Current variable value : 6

Current variable value : 4

Current variable value : 3

Current variable value : 2

Current variable value : 1

Current variable value : 0

Good bye!

Example

Program to exclude the number which are divisible by 3 till range 30.

for i in range:

if (i%3==0):

continue # Third Example

print i

When the above code is executed, it produces the following result −

1

2

4

5

7

8

10

11

13

14

16

17

19

20

22

23

25

26

28

29

* Now try to make similar program with the help of while loop.

# **Python pass Statement**

It is used when a statement is required syntactically but you do not want any command or code to execute.

The **pass** statement is a *null* operation; nothing happens when it executes. The **pass** is also useful in places where your code will eventually go, but has not been written yet (e.g., in stubs for example):

### Syntax

pass

### Example

#!/usr/bin/python

for letter in 'Python':

if letter == 'h':

pass

print 'This is pass block'

print 'Current Letter :', letter

print "Good bye!"

When the above code is executed, it produces following result −

Current Letter : P

Current Letter : y

Current Letter : t

This is pass block

Current Letter : h

Current Letter : o

Current Letter : n

Good bye!