

Programming Fundamentals and Python

Square Root

```
square root.py - C:/Users/Hera Noor/AppData/Local/Programs/Python/Python38/square root
File Edit Format Run Options Window Help
num = float(input('Enter a number: '))
num_sqrt = num ** 0.5
print('The square root of %0.3f is %0.3f'%(num ,num_sqrt))
|
```

OUTPUT

```
IDLE Shell 3.8.10
File Edit Shell Debug Options Window Help
Python 3.8.10 (tags/v3.8.10:3d8993a, May
AMD64) on win32
Type "help", "copyright", "credits" or "li
>>>
= RESTART: C:/Users/Hera Noor/AppData/Loca
.PY
Enter a number: 4
The square root of 4.000 is 2.000
>>> |
```

Decision Making and Conditional Operator

Decision Making Structures

4

Normally, the program flows along line by line in the order in which it appears in source code. But, it is sometimes required to execute a particular portion of code only if certain condition is true; or false i.e. you have to make decision in the program.

General Format

<code>if test1:</code>	<code># if test</code>
<code>statements1</code>	<code># Associated block</code>
<code>elif test2:</code>	<code># Optional elifs</code>
<code>statements2</code>	
<code>else:</code>	<code># Optional else</code>
<code>statements3</code>	

- The indentation (blank whitespace all the way to the left of the two nested statements here) is the factor that defines which code block lies within the condition statement.
- Python doesn't care how indents can be inserted (either spaces or tabs may be used), or how much a statement can be indented (any number of spaces or tabs can be used). In fact, the indentation of one nested block can be totally different from that of another.

- The syntax rule is only that for a given single nested block, all of its statements must be indented the same distance to the right. If this is not the case, a syntax error will appear, and code will not run until its indentation is repaired to be consistent. Python almost forces programmers to produce uniform, regular, and readable code.
- The one new syntax component in Python is the colon character (:). All Python compound statements that have other statements nested inside them—follow the same general pattern of a header line terminated in a colon, followed by a nested block of code usually indented underneath the header line.

Python Conditions and If statements

7

Python supports the usual logical conditions from mathematics:

Equals: $a == b$

Not Equals: $a != b$

Less than: $a < b$

Less than or equal to: $a \leq b$

Greater than: $a > b$

Greater than or equal to: $a \geq b$

These conditions can be used in several ways, most commonly in "if statements" and loops.

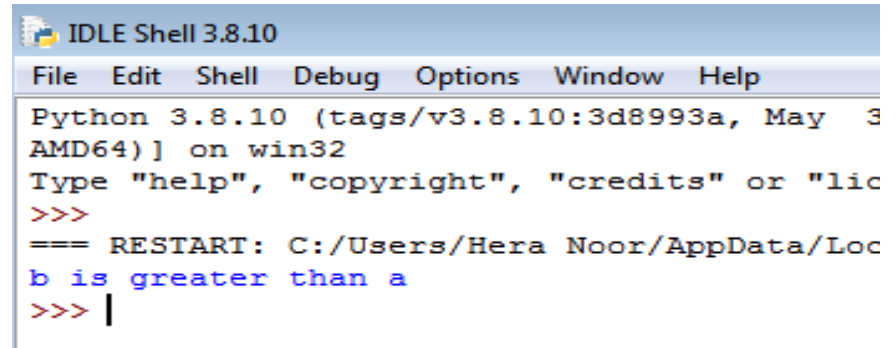
An "if statement" is written by using the if keyword.

Python relies on indentation (whitespace at the beginning of a line) to define scope in the code. Other programming languages often use curly-brackets for this purpose.

CODE

```
a = 86
b = 300
if b > a:
    print("b is greater than a")
|
```

OUTPUT



```
IDLE Shell 3.8.10
File Edit Shell Debug Options Window Help
Python 3.8.10 (tags/v3.8.10:3d8993a, May 3
AMD64)] on win32
Type "help", "copyright", "credits" or "lic
>>>
=== RESTART: C:/Users/Hera Noor/AppData/Loc
b is greater than a
>>> |
```


The **'elif'** keyword is python's way of saying "if the previous conditions were not true, then try this condition".

CODE

```
a = 33
b = 33
if b > a:
    print("b is greater than a")
elif a == b:
    print("a and b are equal")
```

OUTPUT

```
==== RESTART: C:/Users/Hera
a and b are equal
>>> |
```

In this example **a** is equal to **b**, so the first condition is not true, but the elif condition is true, so we print to screen that "a and b are equal".

CODE

```
File Edit Format Run Options Window Help
```

```
a = int(input('enter first number'))  
b = int(input('enter second number'))  
if b>a:  
    print("b is greater than a")  
elif a>b:  
    print("a is greater than b")  
|
```

OUTPUT

```
=== RESTART: C:/Users/Hera 1  
enter first number32  
enter second number12  
a is greater than b  
>>> |
```

In this example **a** and **b** are variables, so any number can be entered, the result will depend on variables and thus, we print to screen that “a is greater than b” or “b is greater than a” .

Else

11

It is used as the final condition respective to 'if' operator. Note: no condition can be defined within else statement.

CODE

```
a = 200
b = 33
if b > a:
    print("b is greater than a")
else:
    print("b is not greater than a")
```

OUTPUT

```
==== RESTART: C:/Users/Hera Noor/A
b is not greater than a
>>> |
```

We cannot use 'elif' operator after 'else' operator, but multiple 'elif' and multiple 'else' operator can be used in a single code to define the condition properly.

CODE

```
File Edit Format Run Options Window Help
a = int(input('enter first number'))
b = int(input('enter second number'))
if b>a:
    print("b is greater than a")
elif a>b:
    print("a is greater than b")
else:
    print("a is equals to b")
|
```

OUTPUT

```
==== RESTART: C:/Users/Hera
enter first number33
enter second number33
a is equals to b
>>>
==== RESTART: C:/Users/Hera
enter first number34
enter second number43
b is greater than a
>>>
==== RESTART: C:/Users/Hera
enter first number55
enter second number15
a is greater than b
>>>
...
```

Ternary Operators or Shorthand 'if'

13

If you have only one statement to execute, you can put it on the same line as the if statement. This technique is known as Ternary Operators, or Conditional Expressions.

CODE

```
File Edit Format Run Options Window Help
a = int(input('enter first number'))
b = int(input('enter second number'))
if b>a:    print("b is greater than a")
elif a>b:    print("a is greater than b")
else:      print("a is equals to b")
|
```

OUTPUT

```
==== RESTART: C:/Users/Hera
enter first number87
enter second number98
b is greater than a
>>> |
```

You can also have multiple else statements on the same line.

CODE

```
File Edit Format Run Options Window Help
a = int(input('enter first number'))
b = int(input('enter second number'))
print("A") if a > b else print("=") if a == b else print("B")
|
```

OUTPUT

```
==== RESTART: C:/Users/Heri
enter first number54
enter second number87
B
>>>
==== RESTART: C:/Users/Heri
enter first number54
enter second number45
A
>>>
==== RESTART: C:/Users/Heri
enter first number66
enter second number66
=
... |
```

The and keyword is a logical operator, and is used to combine conditional statements

CODE

File Edit Format Run Options Window Help

```
a = int(input('enter first number'))
b = int(input('enter second number'))
c = int(input('enter third number'))
if a > b and b > c:
    print("Both conditions are True")
else:
    print("the condition is false")
|
```

OUTPUT

```
>>>
==== RESTART: C:/Users/Hera
enter first number56
enter second number67
enter third number78
the condition is false
>>>
==== RESTART: C:/Users/Hera
enter first number98
enter second number87
enter third number76
Both conditions are True
>>> |
```

The 'or' keyword is a logical operator, and is used to combine conditional statements.

CODE

```
File Edit Format Run Options Window Help
a = int(input('enter first number'))
b = int(input('enter second number'))
c = int(input('enter third number'))
if a > b or b > c:
    print("At least one of the conditions is True")
else:
    print("the condition is false")
|
```

OUTPUT

```
==== RESTART: C:/Users/Hera
enter first number98
enter second number76
enter third number99
At least one of the conditio
>>>
==== RESTART: C:/Users/Hera
enter first number67
enter second number78
enter third number99
the condition is false
>>> |
```


Nested 'if'

17

You can have 'if' statements inside 'if' statements, this is called nested 'if' statements.

CODE

```
File Edit Format Run Options Window Help
x = int(input('enter number'))
if x > 10:
    print("Above ten,")
    if x > 20:
        print("and also above 20!")
        if x > 30:
            print("and also above 30!")
            if x > 40:
                print("and also above 40!")
                if x > 50:
                    print("and also above 50!")
else:
    print("but not above 20.")
```

OUTPUT

```
==== RESTART: C:/Users/Hera
enter number1
>>>
==== RESTART: C:/Users/Hera
enter number12
Above ten,
but not above 20.
>>>
==== RESTART: C:/Users/Hera
enter number24
Above ten,
and also above 20!
>>>
==== RESTART: C:/Users/Hera
enter number35
Above ten,
and also above 20!
and also above 30!
>>>
```

Pass statement

18

'if' statements cannot be empty, but 'if' you for some reason have an 'if' statement with no content, put in the 'pass statement' to avoid getting an error.

```
a = 33
b = 200

if b > a:
    pass
```

Iterative Structure (for and while)

In Python, the for loop is often used to iterate over iterable objects such as lists, tuples, or strings. Traversal is the process of iterating across a series. If we have a section of code that we would like to repeat a certain number of times, we employ for loops. The for-loop is usually used on an iterable object such as a list or the in-built range function. The for statement in Python traverses through the elements of a series, running the block of code each time.

The for statement is in opposition to the "while" loop, which is employed whenever a condition requires to be verified each repetition or when a piece of code is to be repeated indefinitely.

'for' loop

21

A for loop is used for iterating over a sequence (that is either a list, a tuple, a dictionary, a set, or a string).

This is less like the for keyword in other programming languages, and works more like an iterator method as found in other object-orientated programming languages.

With the for loop we can execute a set of statements, once for each item in a list, tuple, set etc. The for loop does not require an indexing variable to set beforehand.

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

Break Statement

22

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
```

```
=== RESTART: C:/Users/Hei
apple
banana
>>>
=== RESTART: C:/Users/Hei
apple
>>>
```

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)
```

The continue Statement

23

With the continue statement we can stop the current iteration of the loop, and continue with the next:

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

```
=== RESTART: C:/U
apple
cherry
>>>
```

The range() Function

24

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.

Note that range(6) is not the values of 0 to 6, but the values 0 to 5.

```
>>> for x in range(6):  
    print(x)
```

```
0
```

```
1
```

```
2
```

```
3
```

```
4
```

```
5
```

```
>>> |
```


The range() Function

25

The range() function defaults to 0 as a starting value, however it is possible to specify the starting value by adding a parameter: range(2, 6), which means values from 2 to 6 (but not including 6):

```
>>> for x in range(2, 6):  
    print(x)
```

```
2  
3  
4  
5
```

```
>>> |
```

The range() Function

26

The range() function defaults to increment the sequence by 1, however it is possible to specify the increment value by adding a third parameter: range(2, 30, 3):

```
>>> for x in range(2, 30, 3):  
    print(x)  
  
2  
5  
8  
11  
14  
17  
20  
23  
26  
29  
>>> |
```

Else in For Loop

27

The else keyword in a for loop specifies a block of code to be executed when the loop is finished

Example

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!")
```

```
=== RESTART: C:/Users/hera N  
0  
1  
2  
3  
4  
5  
Finally finished!  
>>> |
```

Note: 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!")
```

```
=== RESTART: C:/Use  
0  
1  
2  
>>> |
```

Nested Loops

29

A nested loop is a loop inside a loop.

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

Example

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)
```

```
=== RESTART: C:/Users/He
red apple
red banana
red cherry
big apple
big banana
big cherry
tasty apple
tasty banana
tasty cherry
>>>
```

The pass Statement

30

‘For’ loops cannot be empty, but if you for some reason have a ‘for’ loop with no content, put in the ‘pass’ statement to avoid getting an error.

Example

```
for x in [0, 1, 2]:  
    pass
```

While loop falls under the category of indefinite iteration. Indefinite iteration means that the number of times the loop is executed isn't specified explicitly in advance.

Statements represent all the statements indented by the same number of character spaces after a programming construct are considered to be part of a single block of code. Python uses indentation as its method of grouping statements. When a while loop is executed, `expr` is first evaluated in a Boolean context and if it is true, the loop body is executed. Then the `expr` is checked again, if it is still true then the body is executed again and this continues until the expression becomes false.

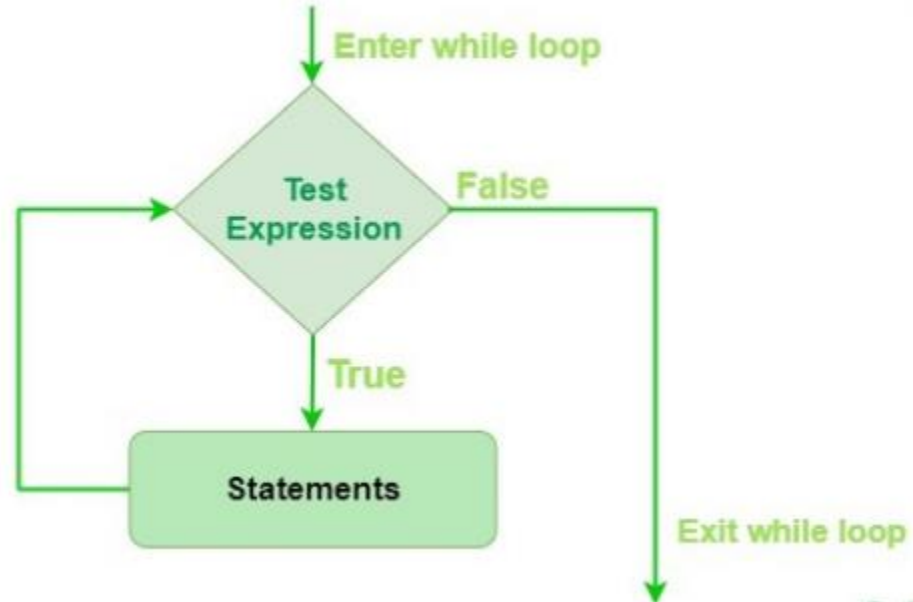
'while' loop

32

Python While Loop is used to execute a block of statements repeatedly until a given condition is satisfied. And when the condition becomes false, the line immediately after the loop in the program is executed.

Syntax:

while expression:
 statement(s)



Example

Print *i* as long as *i* is less than 6:

Note: remember to increment *i*, or else the loop will continue forever. The while loop requires relevant variables to be ready, in this example we need to define an indexing variable, *i*, which we set to 1.

```
i = 1
while i < 6:
    print(i)
    i += 1
|
```

```
=== RESTART: C:/Users/Hera
1
2
3
4
5
>>>
```

The break Statement

34

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

Example

Exit the loop when i is 3:

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

```
...
== RESTART: C:/Users/Her
1
2
3
>>> |
```

The continue Statement

35

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

Example

Continue to the next iteration if i is 3:

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

```
== RESTART: C:/Users/Hera No
1
2
4
5
6
>>>
```

The else Statement

36

With the else statement we can run a block of code once when the condition no longer is true:

Example

Print a message once the condition is false:

```
i = 1
while i < 6:
    print(i)
    i += 1
else:
    print("i is no longer less than 6")
```

```
== RESTART: C:/Users/Hera Noor/
1
2
3
4
5
i is no longer less than 6
>>> |
```

Single statement while block

37

With the else statement we can run a block of code once when the condition no longer is true:

Example

Print a message once the condition is false:

```
count = 0
while (count < 5): count += 1; print("Hello World")
```

```
== RESTART: C:/Users/Hera
Hello World
Hello World
Hello World
Hello World
Hello World
>>>
```