

## Python Tutorial

### Python HOME

- Python Intro
- Python Get Started
- Python Syntax
- Python Comments
- Python Variables
- Python Data Types
- Python Numbers
- Python Casting
- Python Strings
- Python Booleans
- Python Operators
- Python Lists
- Python Tuples
- Python Sets
- Python Dictionaries

- Python If...Else
- Python While Loops
- Python For Loops
- Python Functions
- Python Lambda

# Python Tutorial

[← Home](#)

Python is a programming language.

Python can be used on a server to create web applications.

[Start learning Python now »](#)

## Learning by Examples

With our "Try it Yourself" editor, you can edit the code and view the result.

# Python If ... Else

---

# Python Conditions and If statements

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 <= b`
- Greater than: `a > b`
- Greater than or equal to: `a >= 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.

## Example

If statement:

```
a = 33
b = 200
if b > a:
    print("b is greater than a")
```

[Try it Yourself »](#)

## Indentation

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.

## Example

If statement, without indentation (will raise an error):

```
a = 33
b = 200
if b > a:
print("b is greater than a") # you will get an error
```

[Try it Yourself »](#)

---

---

# Else and Elif

The `else` keyword catches anything which isn't caught by the preceding conditions.

## Example

```
a = 200
b = 33
if b > a:
    print("b is greater than a")
elif a == b:
    print("a and b are equal")
else:
    print("a is greater than b")
```

Try it Yourself »

In this example `a` is greater than `b`, so the first condition is not true, also the `elif` condition is not true, so we go to the `else` condition and print to screen that "a is greater than b".

You can also have an `else` without the `elif`:

## Example

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

Try it Yourself »

---

---

# And

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

## Example

Test if `a` is greater than `b`, AND if `c` is greater than `a`:

```
a = 200
b = 33
c = 500
if a > b and c > a:
    print("Both conditions are True")
```

[Try it Yourself »](#)

---

# Or

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

## Example

Test if `a` is greater than `b`, OR if `a` is greater than `c`:

```
a = 200
b = 33
c = 500
if a > b or a > c:
    print("At least one of the conditions is True")
```

[Try it Yourself »](#)

---

# Python Loops

Python has two loop commands:

- `while` loops
- `for` loops

## The while Loop

With the `while` loop we can execute a set of statements as long as a condition is true.

### Example

Print i as long as i is less than 6:

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

[Try it Yourself »](#)

**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.

## The break Statement

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
```

[Try it Yourself »](#)

## Python For Loops

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.

## Example

Print each fruit in a fruit list:

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

[Try it Yourself »](#)

The `for` loop does not require an indexing variable to set beforehand.

## Looping Through a String

Even strings are iterable objects, they contain a sequence of characters:

## Example

Loop through the letters in the word "banana":

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

[Try it Yourself »](#)

## The break Statement

With the `break` statement we can stop the loop before it has looped through all the items:

### Example

Exit the loop when `x` is "banana":

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

[Try it Yourself »](#)

### Example

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)
```

[Try it Yourself »](#)

# The continue Statement

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

## Example

Do not print banana:

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

[Try it Yourself »](#)

# Python Functions

[← Previous](#) [Next →](#)

A function is a block of code which only runs when it is called.

You can pass data, known as parameters, into a function.

A function can return data as a result.

## Creating a Function



In Python a function is defined using the `def` keyword:

## Example

```
def my_function():  
    print("Hello from a function")
```

## Calling a Function

To call a function, use the function name followed by parenthesis:

## Example

```
def my_function():  
    print("Hello from a function")
```

```
my_function()
```

[Try it Yourself »](#)

# Arguments

Information can be passed into functions as arguments.

Arguments are specified after the function name, inside the parentheses. You can add as many arguments as you want, just separate them with a comma.

The following example has a function with one argument (fname). When the function is called, we pass along a first name, which is used inside the function to print the full name:

## Example

```
def my_function(fname):  
    print(fname + " Refsnes")
```

```
my_function("Emil")  
my_function("Tobias")  
my_function("Linus")
```

[Try it Yourself »](#)

*Arguments* are often shortened to *args* in Python documentations.