* What is Python?

Python is a popular programming language. It was created by Guido van Rossum, and released in 1991.

It is **used** for:

* web development (server-side),
* software development,
* mathematics,
* System scripting.
* What can Python do?
* Python can be used on a server to create web applications.
* Python can be used alongside software to create workflows.
* Python can connect to database systems. It can also read and modify files.
* Python can be used to handle big data and perform complex mathematics.
* Python can be used for rapid prototyping, or for production-ready software development.
* Why Python?
* Python works on different platforms (Windows, Mac, Linux, Raspberry Pi, etc).
* Python has a simple syntax similar to the English language.
* Python has syntax that allows developers to write programs with fewer lines than some other programming languages.
* Python runs on an interpreter system, meaning that code can be executed as soon as it is written. This means that prototyping can be very quick.
* Python can be treated in a procedural way, an object-orientated way or a functional way.
* **Good to know**
* The most recent major version of Python is Python 3, which we shall be using in this tutorial. However, Python 2, although not being updated with anything other than security updates, is still quite popular.
* **Execute Python Syntax - print()**

As we learned in the previous page, Python syntax can be executed by writing directly in the Command Line:

>>> print("Hello, World!")  
Hello, World!

## Python Indentation

Indentation refers to the spaces at the beginning of a code line.

Python uses indentation to indicate a block of code.

### Example

if 5 > 2:  
  print("Five is greater than two!")

Python will give you an error if you skip the indentation:

### Example

Syntax Error:

if 5 > 2:  
print("Five is greater than two!")

# Python Comments

1. Comments can be used to explain Python code.
2. Comments can be used to make the code more readable.
3. Comments can be used to prevent execution when testing code.

### Example

#This is a comment  
print("Hello, World!")

# Python Variables

## Creating Variables

1. Variables are containers for storing data values.
2. Unlike other programming languages, Python has no command for declaring a variable.
3. Variables do not need to be declared with any particular type and can even change type after they have been set.
4. A variable is created the moment you first assign a value to it.

### Example

x = 5  
y = "John"  
print(x)  
print(y)

# Python Data Types

* Variables can store data of different types, and different types can do different things.
* Python has the following data types built-in by default, in these categories:

|  |  |
| --- | --- |
| * Text Type: | Str |
| Numeric Types: | int, float, complex |
| Sequence Types: | list, tuple, range |
| Mapping Type: | Dict (dictionary) |
| Set Types: | set, frozenset |
| Boolean Type: | Bool |
| Binary Types: | bytes, bytearray, memoryview |

## Getting the Data Type

You can get the data type of any object by using the type() function:

### Example

Print the data type of the variable x:

x = 5  
print(type(x))

Python Numbers

There are three numeric types in Python:

* int
* float
* complex
* Bool (boolian)
  1. Variables of numeric types are created when you assign a value to them:

Example

x = 1    # int  
y = 2.8  # float  
z = 1j   # complex

w= True/False (print(10 > 9))

* Python Operators
* Operators are used to perform operations on variables and values.

Python divides the operators in the following groups:

1. Arithmetic operators
2. Assignment operators
3. Comparison operators
4. Logical operators
5. Identity operators
6. Membership operators
7. Bitwise operators

* Python Arithmetic Operators

Arithmetic operators are used with numeric values to perform common mathematical operations:

|  |  |  |  |
| --- | --- | --- | --- |
| **Operator** | **Name** | **Example** | **Try it** |
| + | Addition | x + y | [Try it »](https://www.w3schools.com/python/showpython.asp?filename=demo_oper_add) |
| - | Subtraction | x - y | [Try it »](https://www.w3schools.com/python/showpython.asp?filename=demo_oper_sub) |
| \* | Multiplication | x \* y | [Try it »](https://www.w3schools.com/python/showpython.asp?filename=demo_oper_mult) |
| / | Division | x / y | [Try it »](https://www.w3schools.com/python/showpython.asp?filename=demo_oper_div) |
| % | Modulus | x % y | [Try it »](https://www.w3schools.com/python/showpython.asp?filename=demo_oper_mod) |
| \*\* | Exponentiation | x \*\* y | [Try it »](https://www.w3schools.com/python/showpython.asp?filename=demo_oper_exp) |
| // | Floor division | x // y | [Try it »](https://www.w3schools.com/python/showpython.asp?filename=demo_oper_floordiv) |

* Python Assignment Operators

Assignment operators are used to assign values to variables:

|  |  |  |  |
| --- | --- | --- | --- |
| **Operator** | **Example** | **Same As** | **Try it** |
| = | x = 5 | x = 5 | [Try it »](https://www.w3schools.com/python/showpython.asp?filename=demo_oper_ass1) |
| += | x += 3 | x = x + 3 | [Try it »](https://www.w3schools.com/python/showpython.asp?filename=demo_oper_ass2) |
| -= | x -= 3 | x = x - 3 | [Try it »](https://www.w3schools.com/python/showpython.asp?filename=demo_oper_ass3) |
| \*= | x \*= 3 | x = x \* 3 | [Try it »](https://www.w3schools.com/python/showpython.asp?filename=demo_oper_ass4) |
| /= | x /= 3 | x = x / 3 | [Try it »](https://www.w3schools.com/python/showpython.asp?filename=demo_oper_ass5) |
| %= | x %= 3 | x = x % 3 | [Try it »](https://www.w3schools.com/python/showpython.asp?filename=demo_oper_ass6) |
| //= | x //= 3 | x = x // 3 | [Try it »](https://www.w3schools.com/python/showpython.asp?filename=demo_oper_ass7) |
| \*\*= | x \*\*= 3 | x = x \*\* 3 | [Try it »](https://www.w3schools.com/python/showpython.asp?filename=demo_oper_ass8) |
| &= | x &= 3 | x = x & 3 | [Try it »](https://www.w3schools.com/python/showpython.asp?filename=demo_oper_ass9) |
| |= | x |= 3 | x = x | 3 | [Try it »](https://www.w3schools.com/python/showpython.asp?filename=demo_oper_ass10) |
| ^= | x ^= 3 | x = x ^ 3 | [Try it »](https://www.w3schools.com/python/showpython.asp?filename=demo_oper_ass11) |
| >>= | x >>= 3 | x = x >> 3 | [Try it »](https://www.w3schools.com/python/showpython.asp?filename=demo_oper_ass12) |
| <<= | x <<= 3 | x = x << 3 | [Try it »](https://www.w3schools.com/python/showpython.asp?filename=demo_oper_ass13) |

* Python Comparison Operators

Comparison operators are used to compare two values:

|  |  |  |  |
| --- | --- | --- | --- |
| **Operator** | **Name** | **Example** | **Try it** |
| == | Equal | x == y | [Try it »](https://www.w3schools.com/python/showpython.asp?filename=demo_oper_compare1) |
| != | Not equal | x != y | [Try it »](https://www.w3schools.com/python/showpython.asp?filename=demo_oper_compare2) |
| > | Greater than | x > y | [Try it »](https://www.w3schools.com/python/showpython.asp?filename=demo_oper_compare4) |
| < | Less than | x < y | [Try it »](https://www.w3schools.com/python/showpython.asp?filename=demo_oper_compare5) |
| >= | Greater than or equal to | x >= y | [Try it »](https://www.w3schools.com/python/showpython.asp?filename=demo_oper_compare6) |
| <= | Less than or equal to | x <= y | [Try it »](https://www.w3schools.com/python/showpython.asp?filename=demo_oper_compare7) |

* Python Logical Operators

Logical operators are used to combine conditional statements:

|  |  |  |  |
| --- | --- | --- | --- |
| **Operator** | **Description** | **Example** | **Try it** |
| and | Returns True if both statements are true | x < 5 and  x < 10 | [Try it »](https://www.w3schools.com/python/showpython.asp?filename=demo_oper_logical1) |
| Or | Returns True if one of the statements is true | x < 5 or x < 4 | [Try it »](https://www.w3schools.com/python/showpython.asp?filename=demo_oper_logical2) |
| Not | Reverse the result, returns False if the result is true | not(x < 5 and x < 10) | [Try it »](https://www.w3schools.com/python/showpython.asp?filename=demo_oper_logical3) |

* Python Identity Operators

Identity operators are used to compare the objects, not if they are equal, but if they are actually the same object, with the same memory location:

|  |  |  |  |
| --- | --- | --- | --- |
| **Operator** | **Description** | **Example** | **Try it** |
| is | Returns true if both variables are the same object | x is y | [Try it »](https://www.w3schools.com/python/showpython.asp?filename=demo_oper_identity1) |
| is not | Returns true if both variables are not the same object | x is not y | [Try it »](https://www.w3schools.com/python/showpython.asp?filename=demo_oper_identity2) |

* Python Membership Operators

Membership operators are used to test if a sequence is presented in an object:

|  |  |  |  |
| --- | --- | --- | --- |
| **Operator** | **Description** | **Example** | **Try it** |
| in | Returns True if a sequence with the specified value is present in the object | x in y | [Try it »](https://www.w3schools.com/python/showpython.asp?filename=demo_oper_membership1) |
| not in | Returns True if a sequence with the specified value is not present in the object | x not in y | [Try it »](https://www.w3schools.com/python/showpython.asp?filename=demo_oper_membership2) |

* Python Bitwise Operators

Bitwise operators are used to compare (binary) numbers:

|  |  |  |
| --- | --- | --- |
| **Operator** | **Name** | **Description** |
| & | AND | Sets each bit to 1 if both bits are 1 |
| | | OR | Sets each bit to 1 if one of two bits is 1 |
| ^ | XOR | Sets each bit to 1 if only one of two bits is 1 |
| ~ | NOT | Inverts all the bits |
| << | Zero fill left shift | Shift left by pushing zeros in from the right and let the leftmost bits fall off |
| >> | Signed right shift | Shift right by pushing copies of the leftmost bit in from the left, and let the rightmost bits fall off |

* Python Collections (Arrays)

There are four collection data types in the Python programming language:

* **List** is a collection which is ordered and changeable. Allows duplicate members.
* **Tuple** is a collection which is ordered and unchangeable. Allows duplicate members.
* **Set** is a collection which is unordered and unindexed. No duplicate members.
* **Dictionary** is a collection which is unordered, changeable and indexed. No duplicate members.

When choosing a collection type, it is useful to understand the properties of that type. Choosing the right type for a particular data set could mean retention of meaning, and, it could mean an increase in efficiency or security.

List

A list is a collection which is ordered and changeable. In Python lists are written with square brackets.

Example

Create a List:

thislist = ["apple", "banana", "cherry"]  
print(thislist)

Access Items

You access the list items by referring to the index number:

Example

Print the second item of the list:

thislist = ["apple", "banana", "cherry"]  
print(thislist[1])

Negative Indexing

Negative indexing means beginning from the end, -1 refers to the last item, -2 refers to the second last item etc.

Example

Print the last item of the list:

thislist = ["apple", "banana", "cherry"]  
print(thislist[-1])

Range of Indexes

You can specify a range of indexes by specifying where to start and where to end the range.

When specifying a range, the return value will be a new list with the specified items.

Example

Return the third, fourth, and fifth item:

thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]  
print(thislist[2:5])

**Note:** The search will start at index 2 (included) and end at index 5 (not included).

Remember that the first item has index 0.

By leaving out the start value, the range will start at the first item:

Example

This example returns the items from the beginning to "orange":

thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]  
print(thislist[:4])

By leaving out the end value, the range will go on to the end of the list:

Example

This example returns the items from "cherry" and to the end:

thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]  
print(thislist[2:])

Range of Negative Indexes

Specify negative indexes if you want to start the search from the end of the list:

Example

This example returns the items from index -4 (included) to index -1 (excluded)

thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]  
print(thislist[-4:-1])

Change Item Value

To change the value of a specific item, refer to the index number:

Example

Change the second item:

thislist = ["apple", "banana", "cherry"]  
thislist[1] = "blackcurrant"  
print(thislist)

Loop Through a List

You can loop through the list items by using a for loop:

Example

Print all items in the list, one by one:

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

You will learn more about for loops in our [Python For Loops](https://www.w3schools.com/python/python_for_loops.asp) Chapter.

Check if Item Exists

To determine if a specified item is present in a list use the in keyword:

Example

Check if "apple" is present in the list:

thislist = ["apple", "banana", "cherry"]  
if "apple" in thislist:  
  print("Yes, 'apple' is in the fruits list")

List Length

To determine how many items a list has, use the len() function:

Example

Print the number of items in the list:

thislist = ["apple", "banana", "cherry"]  
print(len(thislist))

Add Items

To add an item to the end of the list, use the append() method:

Example

Using the append() method to append an item:

thislist = ["apple", "banana", "cherry"]  
thislist.append("orange")  
print(thislist)

To add an item at the specified index, use the insert() method:

Example

Insert an item as the second position:

thislist = ["apple", "banana", "cherry"]  
thislist.insert(1, "orange")  
print(thislist)

Remove Item

There are several methods to remove items from a list:

Example

The remove() method removes the specified item:

thislist = ["apple", "banana", "cherry"]  
thislist.remove("banana")  
print(thislist)

Example

The pop() method removes the specified index, (or the last item if index is not specified):

thislist = ["apple", "banana", "cherry"]  
thislist.pop()  
print(thislist)

Example

The del keyword removes the specified index:

thislist = ["apple", "banana", "cherry"]  
del thislist[0]  
print(thislist)

Example

The del keyword can also delete the list completely:

thislist = ["apple", "banana", "cherry"]  
del thislist

Example

The clear() method empties the list:

thislist = ["apple", "banana", "cherry"]  
thislist.clear()  
print(thislist)

Copy a List

You cannot copy a list simply by typing list2 = list1, because: list2 will only be a *reference* to list1, and changes made in list1 will automatically also be made in list2.

There are ways to make a copy, one way is to use the built-in List method copy().

Example

Make a copy of a list with the copy() method:

thislist = ["apple", "banana", "cherry"]  
mylist = thislist.copy()  
print(mylist)

Another way to make a copy is to use the built-in method list().

Example

Make a copy of a list with the list() method:

thislist = ["apple", "banana", "cherry"]  
mylist = list(thislist)  
print(mylist)

Join Two Lists

There are several ways to join, or concatenate, two or more lists in Python.

One of the easiest ways are by using the + operator.

Example

Join two list:

list1 = ["a", "b" , "c"]  
list2 = [1, 2, 3]  
  
list3 = list1 + list2  
print(list3)

Another way to join two lists are by appending all the items from list2 into list1, one by one:

Example

Append list2 into list1:

list1 = ["a", "b", "c"]  
list2 = [1, 2, 3]  
  
for x in list2:  
  list1.append(x)  
  
print(list1)

Or you can use the extend() method, which purpose is to add elements from one list to another list:

Example

Use the extend() method to add list2 at the end of list1:

list1 = ["a", "b", "c"]  
list2 = [1, 2, 3]  
  
list1.extend(list2)  
print(list1)

The list() Constructor

It is also possible to use the list() constructor to make a new list.

Example

Using the list() constructor to make a List:

thislist = list(("apple", "banana", "cherry")) # note the double round-brackets  
print(thislist)

List Methods

Python has a set of built-in methods that you can use on lists.

|  |  |
| --- | --- |
| **Method** | **Description** |
| [append()](https://www.w3schools.com/python/ref_list_append.asp) | Adds an element at the end of the list |
| [clear()](https://www.w3schools.com/python/ref_list_clear.asp) | Removes all the elements from the list |
| [copy()](https://www.w3schools.com/python/ref_list_copy.asp) | Returns a copy of the list |
| [count()](https://www.w3schools.com/python/ref_list_count.asp) | Returns the number of elements with the specified value |
| [extend()](https://www.w3schools.com/python/ref_list_extend.asp) | Add the elements of a list (or any iterable), to the end of the current list |
| [index()](https://www.w3schools.com/python/ref_list_index.asp) | Returns the index of the first element with the specified value |
| [insert()](https://www.w3schools.com/python/ref_list_insert.asp) | Adds an element at the specified position |
| [pop()](https://www.w3schools.com/python/ref_list_pop.asp) | Removes the element at the specified position |
| [remove()](https://www.w3schools.com/python/ref_list_remove.asp) | Removes the item with the specified value |
| [reverse()](https://www.w3schools.com/python/ref_list_reverse.asp) | Reverses the order of the list |
| [sort()](https://www.w3schools.com/python/ref_list_sort.asp) | Sorts the list |

## Tuple

A tuple is a collection which is ordered and **unchangeable**. In Python tuples are written with round brackets.

**Example**

Create a Tuple:

thistuple = ("apple", "banana", "cherry")  
print(thistuple)

## Access Tuple Items

You can access tuple items by referring to the index number, inside square brackets:

### Example

Print the second item in the tuple:

thistuple = ("apple", "banana", "cherry")  
print(thistuple[1])

### Negative Indexing

Negative indexing means beginning from the end, -1 refers to the last item, -2 refers to the second last item etc.

### Example

Print the last item of the tuple:

thistuple = ("apple", "banana", "cherry")  
print(thistuple[-1])

### Range of Indexes

You can specify a range of indexes by specifying where to start and where to end the range.

When specifying a range, the return value will be a new tuple with the specified items.

### Example

Return the third, fourth, and fifth item:

thistuple = ("apple", "banana", "cherry", "orange", "kiwi", "melon", "mango")  
print(thistuple[2:5])

**Note:** The search will start at index 2 (included) and end at index 5 (not included).

Remember that the first item has index 0.

### Range of Negative Indexes

Specify negative indexes if you want to start the search from the end of the tuple:

### Example

This example returns the items from index -4 (included) to index -1 (excluded)

thistuple = ("apple", "banana", "cherry", "orange", "kiwi", "melon", "mango")  
print(thistuple[-4:-1])

**Change Tuple Values**

Once a tuple is created, you cannot change its values. Tuples are **unchangeable**, or **immutable** as it also is called.

But there is a workaround. You can convert the tuple into a list, change the list, and convert the list back into a tuple.

### Example

Convert the tuple into a list to be able to change it:

x = ("apple", "banana", "cherry")  
y = list(x)   
y[1] = "kiwi"  
x = tuple(y)  
print(x)

## Loop Through a Tuple

You can loop through the tuple items by using a for loop.

### Example

Iterate through the items and print the values:

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

You will learn more about for loops in our [Python For Loops](https://www.w3schools.com/python/python_for_loops.asp) Chapter.

## Check if Item Exists

To determine if a specified item is present in a tuple use the in keyword:

### Example

Check if "apple" is present in the tuple:

thistuple = ("apple", "banana", "cherry")  
if "apple" in thistuple:  
  print("Yes, 'apple' is in the fruits tuple")

**Tuple Length**

To determine how many items a tuple has, use the len() method:

### Example

Print the number of items in the tuple:

thistuple = ("apple", "banana", "cherry")  
print(len(thistuple)

**Add Items**

Once a tuple is created, you cannot add items to it. Tuples are **unchangeable**.

### Example

You cannot add items to a tuple:

thistuple = ("apple", "banana", "cherry")  
thistuple[3] = "orange" # This will raise an error  
print(thistuple)

## Create Tuple With One Item

To create a tuple with only one item, you have add a comma after the item, unless Python will not recognize the variable as a tuple.

### Example

One item tuple, remember the commma:

thistuple = ("apple",)  
print(type(thistuple))  
  
#NOT a tuple  
thistuple = ("apple")  
print(type(thistuple))

## Remove Items

**Note:** You cannot remove items in a tuple.

 Tuples are **unchangeable**, so you cannot remove items from it, but you can delete the tuple completely:

### Example

The del keyword can delete the tuple completely:

thistuple = ("apple", "banana", "cherry")  
del thistuple  
print(thistuple) #this will raise an error because the tuple no longer exists

## Join Two Tuples

To join two or more tuples you can use the + operator:

### Example

Join two tuples:

tuple1 = ("a", "b" , "c")  
tuple2 = (1, 2, 3)  
  
tuple3 = tuple1 + tuple2  
print(tuple3)

**The tuple() Constructor**

It is also possible to use the tuple() constructor to make a tuple.

### Example

Using the tuple() method to make a tuple:

thistuple = tuple(("apple", "banana", "cherry")) # note the double round-brackets  
print(thistuple)

## Tuple Methods

Python has two built-in methods that you can use on tuples.

|  |  |
| --- | --- |
| **Method** | **Description** |
| [count()](https://www.w3schools.com/python/ref_tuple_count.asp) | Returns the number of times a specified value occurs in a tuple |
| [index()](https://www.w3schools.com/python/ref_tuple_index.asp) | Searches the tuple for a specified value and returns the position of where it was found |

## Set

A set is a collection which is unordered and unindexed. In Python sets are written with curly brackets.

### Example

Create a Set:

thisset = {"apple", "banana", "cherry"}  
print(thisset)

**Note:** Sets are unordered, so you cannot be sure in which order the items will appear.

## Access Items

You cannot access items in a set by referring to an index, since sets are unordered the items has no index.

But you can loop through the set items using a for loop, or ask if a specified value is present in a set, by using the in keyword.

### Example

Loop through the set, and print the values:

thisset = {"apple", "banana", "cherry"}  
  
for x in thisset:  
  print(x)

### Example

Check if "banana" is present in the set:

thisset = {"apple", "banana", "cherry"}  
  
print("banana" in thisset)

## Change Items

Once a set is created, you cannot change its items, but you can add new items.

## Add Items

To add one item to a set use the add() method.

To add more than one item to a set use the update() method.

### Example

Add an item to a set, using the add() method:

thisset = {"apple", "banana", "cherry"}  
  
thisset.add("orange")  
  
print(thisset)

### Example

Add multiple items to a set, using the update() method:

thisset = {"apple", "banana", "cherry"}  
  
thisset.update(["orange", "mango", "grapes"])  
  
print(thisset)

## Get the Length of a Set

To determine how many items a set has, use the len() method.

### Example

Get the number of items in a set:

thisset = {"apple", "banana", "cherry"}  
  
print(len(thisset))

**Remove Item**

To remove an item in a set, use the remove(), or the discard() method.

### Example

Remove "banana" by using the remove() method:

thisset = {"apple", "banana", "cherry"}  
  
thisset.remove("banana")  
  
print(thisset)

**Note:** If the item to remove does not exist, remove() will raise an error.

### Example

Remove "banana" by using the discard() method:

thisset = {"apple", "banana", "cherry"}  
  
thisset.discard("banana")  
  
print(thisset)

**Note:** If the item to remove does not exist, discard() will **NOT** raise an error.

You can also use the pop(), method to remove an item, but this method will remove the last item. Remember that sets are unordered, so you will not know what item that gets removed.

The return value of the pop() method is the removed item.

### Example

Remove the last item by using the pop() method:

thisset = {"apple", "banana", "cherry"}  
  
x = thisset.pop()  
  
print(x)  
  
print(thisset)

**Note:** Sets are unordered, so when using the pop() method, you will not know which item that gets removed.

### Example

The clear() method empties the set:

thisset = {"apple", "banana", "cherry"}  
  
thisset.clear()  
  
print(thisset)

### Example

The del keyword will delete the set completely:

thisset = {"apple", "banana", "cherry"}  
  
del thisset  
  
print(thisset)

## Join Two Sets

There are several ways to join two or more sets in Python.

You can use the union() method that returns a new set containing all items from both sets, or the update() method that inserts all the items from one set into another:

### Example

The union() method :

set1 = {"a", "b" , "c"}  
set2 = {1, 2, 3}  
  
set3 = set1.union(set2)  
print(set3)

### Example

The update() method inserts the items in set2 into set1:

set1 = {"a", "b" , "c"}  
set2 = {1, 2, 3}  
  
set1.update(set2)  
print(set1)

**Note:** Both union() and update() will exclude any duplicate items.

There are other methods that joins two sets and keeps ONLY the duplicates, or NEVER the duplicates, check the full list of set methods in the bottom of this page.

## The set() Constructor

It is also possible to use the set() constructor to make a set.

### Example

Using the set() constructor to make a set:

thisset = set(("apple", "banana", "cherry")) # note the double round-brackets  
print(thisset)

## Set Methods

Python has a set of built-in methods that you can use on sets.

|  |  |
| --- | --- |
| **Method** | **Description** |
| [add()](https://www.w3schools.com/python/ref_set_add.asp) | Adds an element to the set |
| [clear()](https://www.w3schools.com/python/ref_set_clear.asp) | Removes all the elements from the set |
| [copy()](https://www.w3schools.com/python/ref_set_copy.asp) | Returns a copy of the set |
| [difference()](https://www.w3schools.com/python/ref_set_difference.asp) | Returns a set containing the difference between two or more sets |
| [difference\_update()](https://www.w3schools.com/python/ref_set_difference_update.asp) | Removes the items in this set that are also included in another, specified set |
| [discard()](https://www.w3schools.com/python/ref_set_discard.asp) | Remove the specified item |
| [intersection()](https://www.w3schools.com/python/ref_set_intersection.asp) | Returns a set, that is the intersection of two other sets |
| [intersection\_update()](https://www.w3schools.com/python/ref_set_intersection_update.asp) | Removes the items in this set that are not present in other, specified set(s) |
| [isdisjoint()](https://www.w3schools.com/python/ref_set_isdisjoint.asp) | Returns whether two sets have a intersection or not |
| [issubset()](https://www.w3schools.com/python/ref_set_issubset.asp) | Returns whether another set contains this set or not |
| [issuperset()](https://www.w3schools.com/python/ref_set_issuperset.asp) | Returns whether this set contains another set or not |
| [pop()](https://www.w3schools.com/python/ref_set_pop.asp) | Removes an element from the set |
| [remove()](https://www.w3schools.com/python/ref_set_remove.asp) | Removes the specified element |
| [symmetric\_difference()](https://www.w3schools.com/python/ref_set_symmetric_difference.asp) | Returns a set with the symmetric differences of two sets |
| [symmetric\_difference\_update()](https://www.w3schools.com/python/ref_set_symmetric_difference_update.asp) | inserts the symmetric differences from this set and another |
| [union()](https://www.w3schools.com/python/ref_set_union.asp) | Return a set containing the union of sets |
| [update()](https://www.w3schools.com/python/ref_set_update.asp) | Update the set with the union of this set and others |

## Dictionary

A dictionary is a collection which is unordered, changeable and indexed. In Python dictionaries are written with curly brackets, and they have keys and values.

### Example

Create and print a dictionary:

thisdict = {  
  "brand": "Ford",  
  "model": "Mustang",  
  "year": 1964  
}  
print(thisdict)  
**Accessing Items**

You can access the items of a dictionary by referring to its key name, inside square brackets:

### Example

Get the value of the "model" key:

x = thisdict["model"]

There is also a method called get() that will give you the same result:

### Example

Get the value of the "model" key:

x = thisdict.get("model")  
**Change Values**

You can change the value of a specific item by referring to its key name:

### Example

Change the "year" to 2018:

thisdict = {  
  "brand": "Ford",  
  "model": "Mustang",  
  "year": 1964  
}  
thisdict["year"] = 2018  
**Loop Through a Dictionary**

You can loop through a dictionary by using a for loop.

When looping through a dictionary, the return value are the keys of the dictionary, but there are methods to return the values as well.

### Example

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

for x in thisdict:  
  print(x)

**Example**

Print all values in the dictionary, one by one:

for x in thisdict:  
  print(thisdict[x])

**Example**

You can also use the values() function to return values of a dictionary:

for x in thisdict.values():  
  print(x)

**Example**

Loop through both keys and values, by using the items() function:

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

**Check if Key Exists**

To determine if a specified key is present in a dictionary use the in keyword:

### Example

Check if "model" is present in the dictionary:

thisdict = {  
  "brand": "Ford",  
  "model": "Mustang",  
  "year": 1964  
}  
if "model" in thisdict:  
  print("Yes, 'model' is one of the keys in the thisdict dictionary")

**Dictionary Length**

To determine how many items (key-value pairs) a dictionary has, use the len() method.

### Example

Print the number of items in the dictionary:

print(len(thisdict))

**Adding Items**

Adding an item to the dictionary is done by using a new index key and assigning a value to it:

### Example

thisdict = {  
  "brand": "Ford",  
  "model": "Mustang",  
  "year": 1964  
}  
thisdict["color"] = "red"  
print(thisdict)  
**Removing Items**

There are several methods to remove items from a dictionary:

### Example

The pop() method removes the item with the specified key name:

thisdict = {  
  "brand": "Ford",  
  "model": "Mustang",  
  "year": 1964  
}  
thisdict.pop("model")  
print(thisdict)

**Example**

The popitem() method removes the last inserted item (in versions before 3.7, a random item is removed instead):

thisdict = {  
  "brand": "Ford",  
  "model": "Mustang",  
  "year": 1964  
}  
thisdict.popitem()  
print(thisdict)

### Example

The del keyword removes the item with the specified key name:

thisdict = {  
  "brand": "Ford",  
  "model": "Mustang",  
  "year": 1964  
}  
del thisdict["model"]  
print(thisdict)

### Example

The del keyword can also delete the dictionary completely:

thisdict = {  
  "brand": "Ford",  
  "model": "Mustang",  
  "year": 1964  
}  
del thisdict  
print(thisdict) #this will cause an error because "thisdict" no longer exists.

### Example

The clear() keyword empties the dictionary:

thisdict = {  
  "brand": "Ford",  
  "model": "Mustang",  
  "year": 1964  
}  
thisdict.clear()  
print(thisdict)

## Copy a Dictionary

You cannot copy a dictionary simply by typing dict2 = dict1, because: dict2 will only be a reference to dict1, and changes made in dict1 will automatically also be made in dict2.

There are ways to make a copy, one way is to use the built-in Dictionary method copy().

### Example

Make a copy of a dictionary with the copy() method:

thisdict = {  
  "brand": "Ford",  
  "model": "Mustang",  
  "year": 1964  
}  
mydict = thisdict.copy()  
print(mydict)

Another way to make a copy is to use the built-in method dict().

### Example

Make a copy of a dictionary with the dict() method:

thisdict = {  
  "brand": "Ford",  
  "model": "Mustang",  
  "year": 1964  
}  
mydict = dict(thisdict)  
print(mydict)

## Nested Dictionaries

A dictionary can also contain many dictionaries, this is called nested dictionaries.

### Example

Create a dictionary that contain three dictionaries:

myfamily = {  
  "child1" : {  
    "name" : "Emil",  
    "year" : 2004  
  },  
  "child2" : {  
    "name" : "Tobias",  
    "year" : 2007  
  },  
  "child3" : {  
    "name" : "Linus",  
    "year" : 2011  
  }  
}

Or, if you want to nest three dictionaries that already exists as dictionaries:

### Example

Create three dictionaries, than create one dictionary that will contain the other three dictionaries:

child1 = {  
  "name" : "Emil",  
  "year" : 2004  
}  
child2 = {  
  "name" : "Tobias",  
  "year" : 2007  
}  
child3 = {  
  "name" : "Linus",  
  "year" : 2011  
}  
  
myfamily = {  
  "child1" : child1,  
  "child2" : child2,  
  "child3" : child3  
}

## The dict() Constructor

It is also possible to use the dict() constructor to make a new dictionary:

### Example

thisdict = dict(brand="Ford", model="Mustang", year=1964)  
# note that keywords are not string literals  
# note the use of equals rather than colon for the assignment  
print(thisdict)

## Dictionary Methods

Python has a set of built-in methods that you can use on dictionaries.

|  |  |
| --- | --- |
| **Method** | **Description** |
| [clear()](https://www.w3schools.com/python/ref_dictionary_clear.asp) | Removes all the elements from the dictionary |
| [copy()](https://www.w3schools.com/python/ref_dictionary_copy.asp) | Returns a copy of the dictionary |
| [fromkeys()](https://www.w3schools.com/python/ref_dictionary_fromkeys.asp) | Returns a dictionary with the specified keys and values |
| [get()](https://www.w3schools.com/python/ref_dictionary_get.asp) | Returns the value of the specified key |
| [items()](https://www.w3schools.com/python/ref_dictionary_items.asp) | Returns a list containing a tuple for each key value pair |
| [keys()](https://www.w3schools.com/python/ref_dictionary_keys.asp) | Returns a list containing the dictionary's keys |
| [pop()](https://www.w3schools.com/python/ref_dictionary_pop.asp) | Removes the element with the specified key |
| [popitem()](https://www.w3schools.com/python/ref_dictionary_popitem.asp) | Removes the last inserted key-value pair |
| [setdefault()](https://www.w3schools.com/python/ref_dictionary_setdefault.asp) | Returns the value of the specified key. If the key does not exist: insert the key, with the specified value |
| [update()](https://www.w3schools.com/python/ref_dictionary_update.asp) | Updates the dictionary with the specified key-value pairs |
| [values()](https://www.w3schools.com/python/ref_dictionary_values.asp) | Returns a list of all the values in the dictionary |

# 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")

In this example we use two variables, a and b, which are used as part of the if statement to test whether b is greater than a. As a is 33, and b is 200, we know that 200 is greater than 33, and so we print to screen that "b is greater than a".

## 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

## Elif

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

### Example

a = 33  
b = 33  
if b > a:  
  print("b is greater than a")  
elif a == b:  
  print("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".

## Else

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")

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")

## Short Hand If

If you have only one statement to execute, you can put it on the same line as the if statement.

### Example

One line if statement:

if a > b: print("a is greater than b")

## Short Hand If ... Else

If you have only one statement to execute, one for if, and one for else, you can put it all on the same line:

### Example

One line if else statement:

a = 2  
b = 330  
print("A") if a > b else print("B")

[Try it Yourself »](https://www.w3schools.com/python/trypython.asp?filename=demo_if_else_short)

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

### Example

One line if else statement, with 3 conditions:

a = 330  
b = 330  
print("A") if a > b else print("=") if a == b else print("B")

[Try it Yourself »](https://www.w3schools.com/python/trypython.asp?filename=demo_if_else_short2)

## 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 »](https://www.w3schools.com/python/trypython.asp?filename=demo_if_and)

## 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 »](https://www.w3schools.com/python/trypython.asp?filename=demo_if_or)

## Nested If

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

### Example

x = 41  
  
if x > 10:  
  print("Above ten,")  
  if x > 20:  
    print("and also above 20!")  
  else:  
    print("but not above 20.")

[Try it Yourself »](https://www.w3schools.com/python/trypython.asp?filename=demo_if_nested)

## The pass Statement

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.

### Example

a = 33  
b = 200  
  
if b > a:  
  pass

# Python While Loops

Python Loops

Python has two primitive 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

**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

The continue Statement

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)

The else Statement

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")

# Python For Loops

## 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)

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)

## 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

### 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)

## 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)

## The range() Function

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.

### Example

Using the range() function:

for x in range(6):  
  print(x)

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

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):

### Example

Using the start parameter:

for x in range(2, 6):  
  print(x)

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**):

### Example

Increment the sequence with 3 (default is 1):

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

## Else in For Loop

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!")

## 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":

### 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)

## The pass Statement

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

# Python Function:

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()**

## 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"**)

Arguments are often shortened to args in Python documentations.

## Parameters or Arguments?

The terms parameter and argument can be used for the same thing: information that are passed into a function.

From a function's perspective:

A parameter is the variable listed inside the parentheses in the function definition.

An argument is the value that are sent to the function when it is called.

## Number of Arguments

By default, a function must be called with the correct number of arguments. Meaning that if your function expects 2 arguments, you have to call the function with 2 arguments, not more, and not less.

### Example

This function expects 2 arguments, and gets 2 arguments

def my\_function(fname, lname):  
  print(fname + " " + lname)  
  
my\_function("Emil", "Refsnes")

If you try to call the function with 1 or 3 arguments, you will get an error:

### Example

This function expects 2 arguments, but gets only 1:

def my\_function(fname, lname):  
  print(fname + " " + lname)  
  
my\_function("Emil")

## Arbitrary Arguments, \*args

If you do not know how many arguments that will be passed into your function, add a \* before the parameter name in the function definition.

This way the function will receive a tuple of arguments, and can access the items accordingly:

### Example

If the number of arguments is unknown, add a \* before the parameter name:

def my\_function(\*kids):  
  print("The youngest child is " + kids[2])  
  
my\_function("Emil", "Tobias", "Linus")

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

## Keyword Arguments

You can also send arguments with the key = value syntax.

This way the order of the arguments does not matter.

### Example

def my\_function(child3, child2, child1):  
  print("The youngest child is " + child3)  
  
my\_function(child1 = "Emil", child2 = "Tobias", child3 = "Linus")

The phrase Keyword Arguments are often shortened to kwargs in Python documentations.

**Arbitrary Keyword Arguments, \*\*kwargs**

If you do not know how many keyword arguments that will be passed into your function, add two asterisk: \*\* before the parameter name in the function definition.

This way the function will receive a dictionary of arguments, and can access the items accordingly:

### Example

If the number of keyword arguments is unknown, add a double \*\* before the parameter name:

def my\_function(\*\*kid):  
  print("His last name is " + kid["lname"])  
  
my\_function(fname = "Tobias", lname = "Refsnes")

Arbitrary Kword Arguments are often shortened to \*\*kwargs in Python documentations.

## Default Parameter Value

The following example shows how to use a default parameter value.

If we call the function without argument, it uses the default value:

### Example

def my\_function(**country = "Norway"**):  
  print("I am from " + country)  
  
my\_function("Sweden")  
my\_function("India")  
my\_function()  
my\_function("Brazil")

## Passing a List as an Argument

You can send any data types of argument to a function (string, number, list, dictionary etc.), and it will be treated as the same data type inside the function.

E.g. if you send a List as an argument, it will still be a List when it reaches the function:

### Example

def my\_function(food):  
  for x in food:  
    print(x)  
  
fruits = ["apple", "banana", "cherry"]  
  
my\_function(fruits)

## Return Values

To let a function return a value, use the return statement:

### Example

def my\_function(x):  
  **return 5 \* x**  
print(my\_function(3))  
print(my\_function(5))  
print(my\_function(9))

## The pass Statement

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

### Example

def myfunction():  
  pass

## Recursion

Python also accepts function recursion, which means a defined function can call itself.

Recursion is a common mathematical and programming concept. It means that a function calls itself. This has the benefit of meaning that you can loop through data to reach a result.

The developer should be very careful with recursion as it can be quite easy to slip into writing a function which never terminates, or one that uses excess amounts of memory or processor power. However, when written correctly recursion can be a very efficient and mathematically-elegant approach to programming.

In this example, tri\_recursion() is a function that we have defined to call itself ("recurse"). We use the k variable as the data, which decrements (-1) every time we recurse. The recursion ends when the condition is not greater than 0 (i.e. when it is 0).

To a new developer it can take some time to work out how exactly this works, best way to find out is by testing and modifying it.

### Example

Recursion Example

def tri\_recursion(k):  
  if(k>0):  
    result = k+tri\_recursion(k-1)  
    print(result)  
  else:  
    result = 0  
  return result  
  
print("\n\nRecursion Example Results")  
tri\_recursion(6)  
 **Yourself With Exercises**

# 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")

In this example we use two variables, a and b, which are used as part of the if statement to test whether b is greater than a. As a is 33, and b is 200, we know that 200 is greater than 33, and so we print to screen that "b is greater than a".

## 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

## Elif

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

### Example

a = 33  
b = 33  
if b > a:  
  print("b is greater than a")  
elif a == b:  
  print("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".

## Else

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")

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")

## Short Hand If

If you have only one statement to execute, you can put it on the same line as the if statement.

### Example

One line if statement:

if a > b: print("a is greater than b")

## Short Hand If ... Else

If you have only one statement to execute, one for if, and one for else, you can put it all on the same line:

### Example

One line if else statement:

a = 2  
b = 330  
print("A") if a > b else print("B")

[Try it Yourself »](https://www.w3schools.com/python/trypython.asp?filename=demo_if_else_short)

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

### Example

One line if else statement, with 3 conditions:

a = 330  
b = 330  
print("A") if a > b else print("=") if a == b else print("B")

[Try it Yourself »](https://www.w3schools.com/python/trypython.asp?filename=demo_if_else_short2)

## 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 »](https://www.w3schools.com/python/trypython.asp?filename=demo_if_and)

## 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 »](https://www.w3schools.com/python/trypython.asp?filename=demo_if_or)

## Nested If

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

### Example

x = 41  
  
if x > 10:  
  print("Above ten,")  
  if x > 20:  
    print("and also above 20!")  
  else:  
    print("but not above 20.")

[Try it Yourself »](https://www.w3schools.com/python/trypython.asp?filename=demo_if_nested)

## The pass Statement

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.

### Example

a = 33  
b = 200  
  
if b > a:  
  pass

# Python While Loops

Python Loops

Python has two primitive 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

**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

The continue Statement

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)

The else Statement

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")