## Python Tutorial

1. Python Intro
2. [Python Get Started](https://www.w3schools.com/python/python_getstarted.asp)
3. [Python Syntax](https://www.w3schools.com/python/python_syntax.asp)
4. [Python Variables](https://www.w3schools.com/python/python_variables.asp)
5. [Python Numbers](https://www.w3schools.com/python/python_numbers.asp)
6. [Python Casting](https://www.w3schools.com/python/python_casting.asp)
7. [Python Strings](https://www.w3schools.com/python/python_strings.asp)
8. [Python Operators](https://www.w3schools.com/python/python_operators.asp)
9. [Python Lists](https://www.w3schools.com/python/python_lists.asp)
10. [Python Tuples](https://www.w3schools.com/python/python_tuples.asp)
11. [Python Sets](https://www.w3schools.com/python/python_sets.asp)
12. [Python Dictionaries](https://www.w3schools.com/python/python_dictionaries.asp)
13. [Python Conditions](https://www.w3schools.com/python/python_conditions.asp)
14. [Python While Loops](https://www.w3schools.com/python/python_while_loops.asp)
15. [Python For Loops](https://www.w3schools.com/python/python_for_loops.asp)
16. [Python Functions](https://www.w3schools.com/python/python_functions.asp)
17. [Python Classes/Objects](https://www.w3schools.com/python/python_classes.asp)
18. [Python Modules](https://www.w3schools.com/python/python_modules.asp)
19. [Python Dates](https://www.w3schools.com/python/python_datetime.asp)
20. [Python PIP](https://www.w3schools.com/python/python_pip.asp)

## File Handling

1. [Python File Handling](https://www.w3schools.com/python/python_file_handling.asp)
2. [Python Read Files](https://www.w3schools.com/python/python_file_open.asp)
3. [Python Write/Create Files](https://www.w3schools.com/python/python_file_write.asp)
4. [Python Delete Files](https://www.w3schools.com/python/python_file_remove.asp)

## Python MySQL

1. [MySQL Get Started](https://www.w3schools.com/python/python_mysql_getstarted.asp" \t "_top)
2. [MySQL Create Database](https://www.w3schools.com/python/python_mysql_create_db.asp)
3. [MySQL Create Table](https://www.w3schools.com/python/python_mysql_create_table.asp)
4. [MySQL Insert](https://www.w3schools.com/python/python_mysql_insert.asp)
5. [MySQL Select](https://www.w3schools.com/python/python_mysql_select.asp)
6. [MySQL Where](https://www.w3schools.com/python/python_mysql_where.asp)
7. [MySQL Delete](https://www.w3schools.com/python/python_mysql_delete.asp)
8. [MySQL Drop Table](https://www.w3schools.com/python/python_mysql_drop_table.asp)
9. [MySQL Update](https://www.w3schools.com/python/python_mysql_update.asp)

**Python Introduction**

## What is Python?

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

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.
* In this tutorial Python will be written in a text editor. It is possible to write Python in an Integrated Development Environment, such as Thonny, Pycharm, Netbeans or Eclipse which are particularly useful when managing larger collections of Python files.

### Python Syntax compared to other programming languages

* Python was designed to for readability, and has some similarities to the English language with influence from mathematics.
* Python uses new lines to complete a command, as opposed to other programming languages which often use semicolons or parentheses.
* Python relies on indentation, using whitespace, to define scope; such as the scope of loops, functions and classes. Other programming languages often use curly-brackets for this purpose.

# Python Syntax

## Execute Python Syntax

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

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

Or by creating a python file on the server, using the .py file extension, and running it in the Command Line:

C:\Users\Your Name>python myfile.py

## Python Indentations

Where in other programming languages the indentation in code is for readability only, in Python the indentation is very important.

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

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

## Comments

Python has commenting capability for the purpose of in-code documentation.

Comments start with a #, and Python will render the rest of the line as a comment:

### Example

Comments in Python:

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

### Docstrings

Python also has extended documentation capability, called docstrings.

Docstrings can be one line, or multiline.

Python uses triple quotes at the beginning and end of the docstring:

### Example

Docstrings are also comments:

"""This is a   
multiline docstring."""  
print("Hello, World!")

# Python Variables

## Creating Variables

Unlike other programming languages, Python has no command for declaring a variable.

A variable is created the moment you first assign a value to it.

### Example

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

Variables do not need to be declared with any particular type and can even change type after they have been set.

### Example

x = 4 # x is of type int  
x = "Sally" # x is now of type str  
print(x)

## Variable Names

A variable can have a short name (like x and y) or a more descriptive name (age, carname, total\_volume). Rules for Python variables:

* A variable name must start with a letter or the underscore character
* A variable name cannot start with a number
* A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and \_ )
* Variable names are case-sensitive (age, Age and AGE are three different variables)

Remember that variables are case-sensitive

## Output Variables

The Python print statement is often used to output variables.

To combine both text and a variable, Python uses the + character:

### Example

x = "awesome"  
print("Python is " + x)

You can also use the + character to add a variable to another variable:

### Example

x = "Python is "  
y = "awesome"  
z =  x + y  
print(z)

For numbers, the + character works as a mathematical operator:

### Example

x = 5  
y = 10  
print(x + y)

If you try to combine a string and a number, Python will give you an error:

### Example

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

# Python Numbers

## Python Numbers

There are three numeric types in Python:

* int
* float
* complex

Variables of numeric types are created when you assign a value to them:

### Example

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

To verify the type of any object in Python, use the type() function:

### Example

print(type(x))  
print(type(y))  
print(type(z))

## Int

Int, or integer, is a whole number, positive or negative, without decimals, of unlimited length.

### Example

Integers:

x = 1  
y = 35656222554887711  
z = -3255522  
  
print(type(x))  
print(type(y))  
print(type(z))

## Float

Float, or "floating point number" is a number, positive or negative, containing one or more decimals.

### Example

Floats:

x = 1.10  
y = 1.0  
z = -35.59  
  
print(type(x))  
print(type(y))  
print(type(z))

Float can also be scientific numbers with an "e" to indicate the power of 10.

### Example

Floats:

x = 35e3  
y = 12E4  
z = -87.7e100  
  
print(type(x))  
print(type(y))  
print(type(z))

## Complex

Complex numbers are written with a "j" as the imaginary part:

### Example

Complex:

x = 3+5j  
y = 5j  
z = -5j  
  
print(type(x))  
print(type(y))  
print(type(z))

# Python Casting

## Specify a Variable Type

There may be times when you want to specify a type on to a variable. This can be done with casting. Python is an object-orientated language, and as such it uses classes to define data types, including its primitive types.

Casting in python is therefore done using constructor functions:

* int()- constructs an integer number from an integer literal, a float literal (by rounding down to the previous whole number) literal, or a string literal (providing the string represents a whole number)
* float() - constructs a float number from an integer literal, a float literal or a string literal (providing the string represents a float or an integer)
* str() - constructs a string from a wide variety of data types, including strings, integer literals and float literals

Integers: x = int(1)   # x will be 1  
y = int(2.8) # y will be 2  
z = int("3") # z will be 3

Floats: x = float(1)     # x will be 1.0  
y = float(2.8)   # y will be 2.8  
z = float("3")   # z will be 3.0  
w = float("4.2") # w will be 4.2

Strings: x = str("s1") # x will be 's1'  
y = str(2)    # y will be '2'  
z = str(3.0)  # z will be '3.0'

# Python Strings

## String Literals

String literals in python are surrounded by either single quotation marks, or double quotation marks.

'hello' is the same as "hello".

Strings can be output to screen using the print function. For example: print("hello").

Like many other popular programming languages, strings in Python are arrays of bytes representing unicode characters. However, Python does not have a character data type, a single character is simply a string with a length of 1. Square brackets can be used to access elements of the string.

Get the character at position 1:

a = "hello"  
print(a[1])

Substring. Get the characters from position 2 to position 5:

b = "world"  
print(b[2:5])

The strip() method removes any whitespace from the beginning or the end:

a = " Hello, World! "  
print(a.strip()) # returns "Hello, World!"

The lower() method returns the string in lower case:

a = "Hello, World!"  
print(a.lower())

The upper() method returns the string in upper case:

a = "Hello, World!"  
print(a.upper())

The replace() method replaces a string with another string:

a = "Hello, World!"  
print(a.replace("H", "J"))

The split() method splits the string into substrings if it finds instances of the separator:

a = "Hello, World!"  
print(a.split(",")) # returns ['Hello', ' World!']

## Command-line String Input

Python allows for command line input. That means we are able to ask the user for input.

The following example asks for the user's name, then, by using the input() method, the program prints the name to the screen:

demo\_string\_input.py

print("Enter your name:")  
x = input()  
print("Hello, " + x)

## **Python Operators**

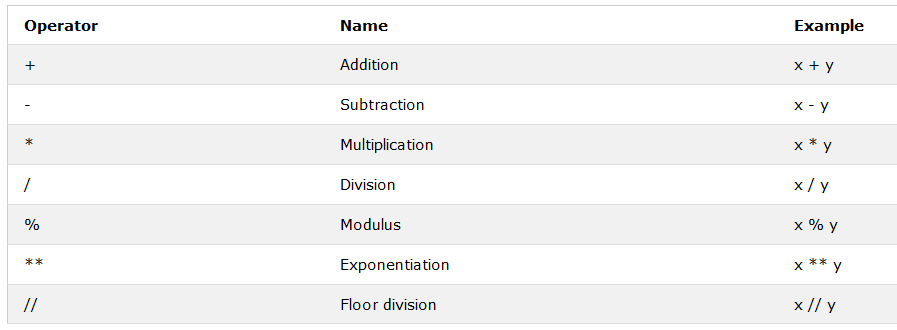
Operators are used to perform operations on variables and values.

Python divides the operators in the following groups:

* Arithmetic operators
* Assignment operators
* Comparison operators
* Logical operators
* Identity operators
* Membership operators
* Bitwise operators

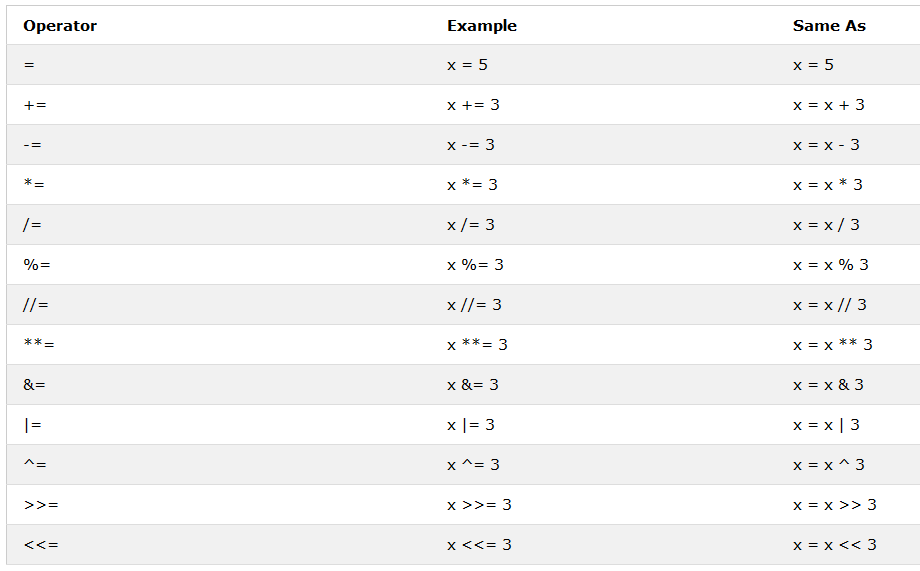
## Python Arithmetic Operators

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



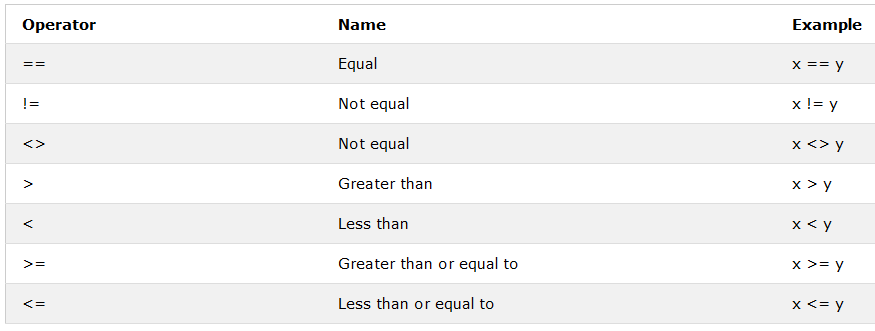
## Python Assignment Operators

Assignment operators are used to assign values to variables:



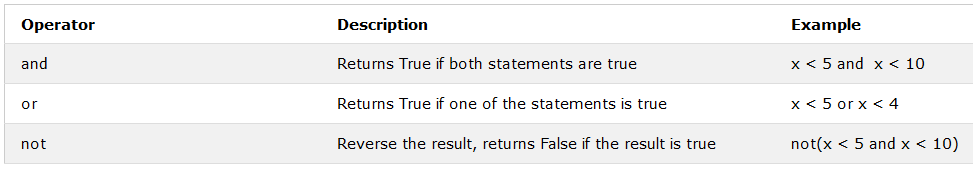
## Python Comparison Operators

Comparison operators are used to compare two values:



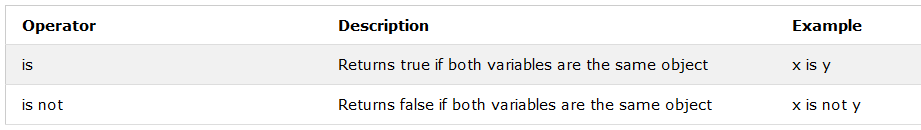
## Python Logical Operators

Logical operators are used to combine conditional statements:



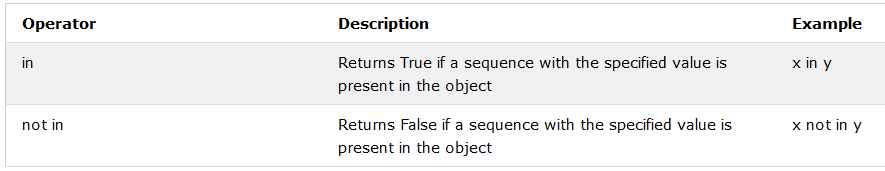
## Python Identity Operators

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:



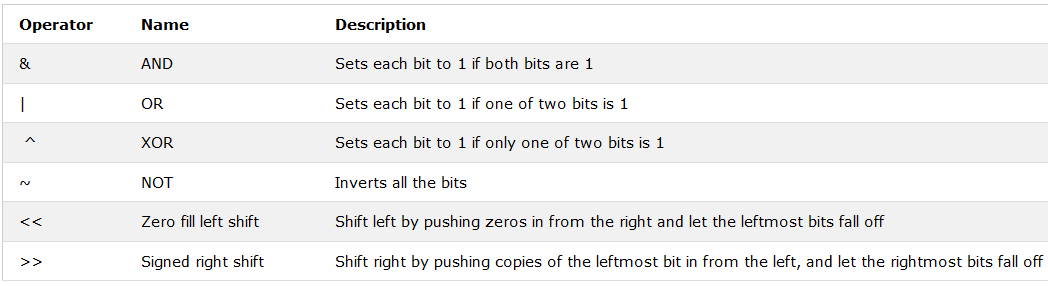
## Python Membership Operators

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



## Python Bitwise Operators

Logical operators are used to combine conditional statements:



# Python Lists

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.

Create a List:

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

Change the second item:

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

## The list() Constructor

It is also possible to use the list() constructor to make a list. To add an item to the list use append() object method. To remove a specific item use the remove() object method. The len() function returns the length of the list.

Using the list() constructor to make a List:

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

Using the append() method to append an item:

thislist = list(("apple", "banana", "cherry"))  
thislist.append("damson")  
print(thislist)

Using the remove() method to remove an item:

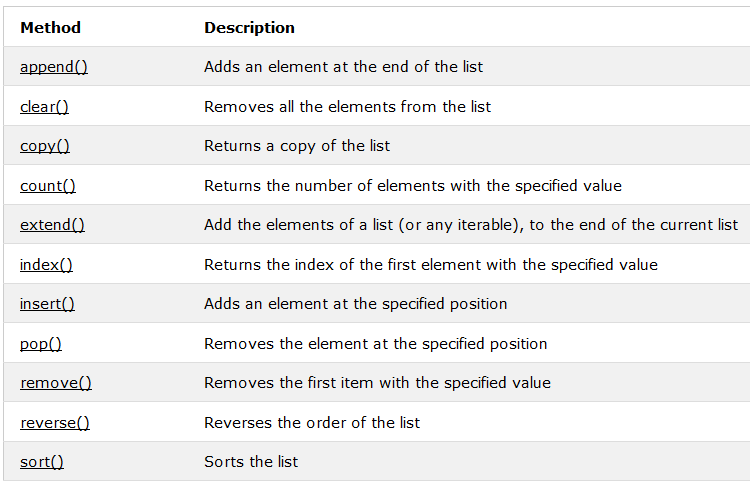
thislist = list(("apple", "banana", "cherry"))  
thislist.remove("banana")  
print(thislist)

The len() method returns the number of items in a list:

thislist = list(("apple", "banana", "cherry"))  
print(len(thislist))

## List Methods

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



## Tuple

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

Create a Tuple:

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

Return the item in position 1:

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

You cannot change values in a tuple:

thistuple = ("apple", "banana", "cherry")  
thistuple[1] = "blackcurrant" # test changeability  
print(thistuple)

## The tuple() Constructor

It is also possible to use the tuple() constructor to make a tuple. The len() function returns the length of the tuple.

Using the tuple() method to make a tuple:

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

The len() method returns the number of items in a tuple:

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

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

## Set

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

Create a Set:

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

**Note:** the set list is unordered, so the items will appear in a random order.

## The set() Constructor

It is also possible to use the set() constructor to make a set. You can use the add() object method to add an item, and the remove() object method to remove an item from the set. The len() function returns the size of the set.

Using the set() constructor to make a set:

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

Using the add() method to add an item:

thisset = set(("apple", "banana", "cherry"))  
thisset.add("damson")  
print(thisset)

Using the remove() method to remove an item:

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

Using the len() method to return the number of items:

thisset = set(("apple", "banana", "cherry"))  
print(len(thisset))

## 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.

Using the len() method to return the number of items:

thisdict = {  
  "apple": "green",  
  "banana": "yellow",  
  "cherry": "red"  
}  
print(thisdict)

Change the apple color to "red":

thisdict = {  
  "apple": "green",  
  "banana": "yellow",  
  "cherry": "red"  
}  
thisdict["apple"] = "red"  
print(thisdict)

## The dict() Constructor

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

thisdict = dict(apple="green", banana="yellow", cherry="red")  
# note that keywords are not string literals  
# note the use of equals rather than colon for the assignment  
print(thisdict)

## Adding Items

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

thisdict = dict(apple="green", banana="yellow", cherry="red")  
thisdict["damson"] = "purple"  
print(thisdict)

## Removing Items

Removing a dictionary item must be done using the del() function in python:

thisdict = dict(apple="green", banana="yellow", cherry="red")  
del(thisdict["banana"])  
print(thisdict)

## Get the Length of a Dictionary

The len() function returns the size of the dictionary:

thisdict = dict(apple="green", banana="yellow", cherry="red")  
print(len(thisdict))

# Python Conditions

## 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.

If statement:

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

n 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, using whitespace, to define scope in the code. Other programming languages often use curly-brackets for this purpose.

Statements on new lines MUST use indentations:

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

If statement, without indentation:

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 do this condition".

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.

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 to 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".

# 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.

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:

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:

Continue to the next iteration if i is 3:

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

## The For Loops

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

This is less like the for keyword in other programming language, 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.

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, as the for command itself allows for this.

## The break Statement

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

Exit the loop when i is 3:

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 wit the next:

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.

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):

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**):

Increment the sequence with 3 (default is 1):

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

# Python Functions

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:

def my\_function():  
  print("Hello from a function")

## Calling a Function

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

def my\_function():  
  print("Hello from a function")  
  
**my\_function()**

## Parameters

Information can be passed to functions as parameter.

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

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

def my\_function(**fname**):  
  print(fname + " Refsnes")  
  
my\_function(**"Emil"**)  
my\_function(**"Tobias"**)  
my\_function(**"Linus"**)

## Default Parameter Value

The following example shows how to use a default parameter value. If we call the function without parameter, it uses the default value:

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

## Return Values

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

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

## Lambda Functions

In python, the keyword lambda is used to create what is known as anonymous functions. These are essentially functions with no pre-defined name. They are good for constructing adaptable functions, and thus good for event handling.

An anonymous function that returns the double value of i:

myfunc = lambda i: i\*2  
print(myfunc(2))

Lambda defined functions can have more than one defined input, as shown here:

myfunc = lambda x,y: x\*y  
print(myfunc(3,6))

The power of lambda is better shown when you generate anonymous functions at run-time, as shown in the following example.

def myfunc(n):  
  return lambda i: i\*n  
  
doubler = myfunc(2)  
tripler = myfunc(3)  
val = 11  
print("Doubled: " + str(doubler(val)) + ". Tripled: " + str(tripler(val)))

Here we see the defined function, myfunc, which creates an anonymous function that multiplies variable i with variable n.

We then create two variables doubler and tripler, which are assigned to the result of myfunc passing in 2 and 3 respectively. They are assigned to the generated lambda functions.

# Python Classes and Objects

## Python Classes/Objects

Python is an object oriented programming language. Almost everything in Python is an object, with its properties and methods. A Class is like an object constructor, or a "blueprint" for creating objects.

## Create a Class

To create a class, use the keyword class:

Create a class named MyClass, with a property named x:

class MyClass:  
  x = 5

## Create Object

Now we can use the class named myClass to create objects:

Create an object named p1, and print the value of x:

p1 = MyClass()  
print(p1.x)

## The \_\_init\_\_() Function

The examples above are classes and objects in their simplest form, and are not really useful in real life applications.

To understand the meaning of classes we have to understand the built-in \_\_init\_\_() function.

All classes have a function called \_\_init\_\_(), which is always executed when the class is being initiated.

Use the \_\_init\_\_() function to assign values to object properties, or other operations that are necessary to do when the object is being created:

Create a class named Person, use the \_\_init\_\_() function to assign values for name and age:

class Person:  
  def \_\_init\_\_(self, name, age):  
    self.name = name  
    self.age = age  
  
p1 = Person("John", 36)  
  
print(p1.name)  
print(p1.age)

**Note:** The \_\_init\_\_() function is called automatically every time the class is being used to create a new object.

## Object Methods

Objects can also contain methods. Methods in objects are functions that belongs to the object.

Let us create a method in the Person class:

Insert a function that prints a greeting, and execute it on the p1 object:

class Person:  
  def \_\_init\_\_(self, name, age):  
    self.name = name  
    self.age = age  
  
  def myfunc(self):  
    print("Hello my name is " + self.name)  
  
p1 = Person("John", 36)  
p1.myfunc()

**Note:** The self parameter is a reference to the class itself, and is used to access variables that belongs to the class.

## The self Parameter

The self parameter is a reference to the class itself, and is used to access variables that belongs to the class.

It does not have to be named self , you can call it whatever you like, but it has to be the first parameter of any function in the class:

Use the words mysillyobject and abc instead of self:

class Person:  
  def \_\_init\_\_(mysillyobject, name, age):  
    mysillyobject.name = name  
    mysillyobject.age = age  
  
  def myfunc(abc):  
    print("Hello my name is " + abc.name)  
  
p1 = Person("John", 36)  
p1.myfunc()

## Modify Object Properties

You can modify properties on objects like this:

Set the age of p1 to 40:

p1.age = 40

## Delete Object Properties

You can delete properties on objects by using the del keyword:

Delete the age property from the p1 object:

del p1.age

## Delete Objects

You can delete objects by using the del keyword:

Delete the p1 object:

del p1

# Python Modules

## What is a Module?

Consider a module to be the same as a code library.

A file containing a set of functions you want to include in your application.

## Create a Module

To create a module just save the code you want in a file with the file extension .py:

Save this code in a file named mymodule.py

def greeting(name):  
  print("Hello, " + name)

## Use a Module

Now we can use the module we just created, by using the import statement:

Import the module named mymodule, and call the greeting function:

import mymodule  
  
mymodule.greeting("Jonathan")

**Note:** When using a function from a module, use the syntax: module\_name.function\_name.

## Variables in Module

The module can contain functions, as already described, but also variables of all types (arrays, dictionaries, objects etc):

Save this code in the file mymodule.py

person1 = {  
  "name": "John",  
  "age": 36,  
  "country": "Norway"  
}

Import the module named mymodule, and access the person1 dictionary:

import mymodule  
  
a = mymodule.person1["age"]  
print(a)

## Naming a Module

You can name the module file whatever you like, but it must have the file extension .py

## Re-naming a Module

You can create an alias when you import a module, by using the as keyword:

Create an alias for mymodule called mx:

import mymodule as mx  
  
a = mx.person1["age"]  
print(a)

## Built-in Modules

There are several built-in modules in Python, which you can import whenever you like.

Import and use the platform module:

import platform  
  
x = platform.system()  
print(x)

## Using the dir() Function

There is a built-in function to list all the function names (or variable names) in a module. The dir() function:

List all the defined names belonging to the platform module:

import platform  
  
x = dir(platform)  
print(x)

**Note:** The dir() function can be used on all modules, also the ones you create yourself.

## Import from Module

You can choose to import only parts from a module, by using the from keyword.

The module named mymodule has one function and one dictionary:

def greeting(name):  
  print("Hello, " + name)  
  
person1 = {  
  "name": "John",  
  "age": 36,  
  "country": "Norway"  
}

Import only the person1 dictionary from the module:

from mymodule import person1  
  
print (person1["age"])

**Note:** When importing using the from keyword, do not use the module name when referring to elements in the module. Example: person1.age, **not** ~~mymodule.person1.age~~

# Python Datetime

## Python Dates

A date in Python is not a data type of its own, but we can import a module named datetime to work with dates as date objects.

Import the datetime module and display the current date:

import datetime  
  
x = datetime.datetime.now()  
print(x)

## Date Output

When we execute the code from the example above the result will be:

2018-06-15 11:41:45.269189

The date contains year, month, day, hour, minute, second, and microsecond. The datetime module has many methods to return information about the date object.

Here are a few examples, you will learn more about them later in this chapter:

Return the year and name of weekday:

import datetime  
  
x = datetime.datetime.now()  
  
print(x.year)  
print(x.strftime("%A"))

## Creating Date Objects

To create a date, we can use the datetime() class (constructor) of the datetime module.

The datetime() class requires three parameters to create a date: year, month, day.

Create a date object:

import datetime  
  
x = datetime.datetime(2020, 5, 17)  
  
print(x)

The datetime() class also takes parameters for time and timezone (hour, minute, second, microsecond, tzone), but they are optional, and has a default value of 0, (None for timezone).

## The strftime() Method

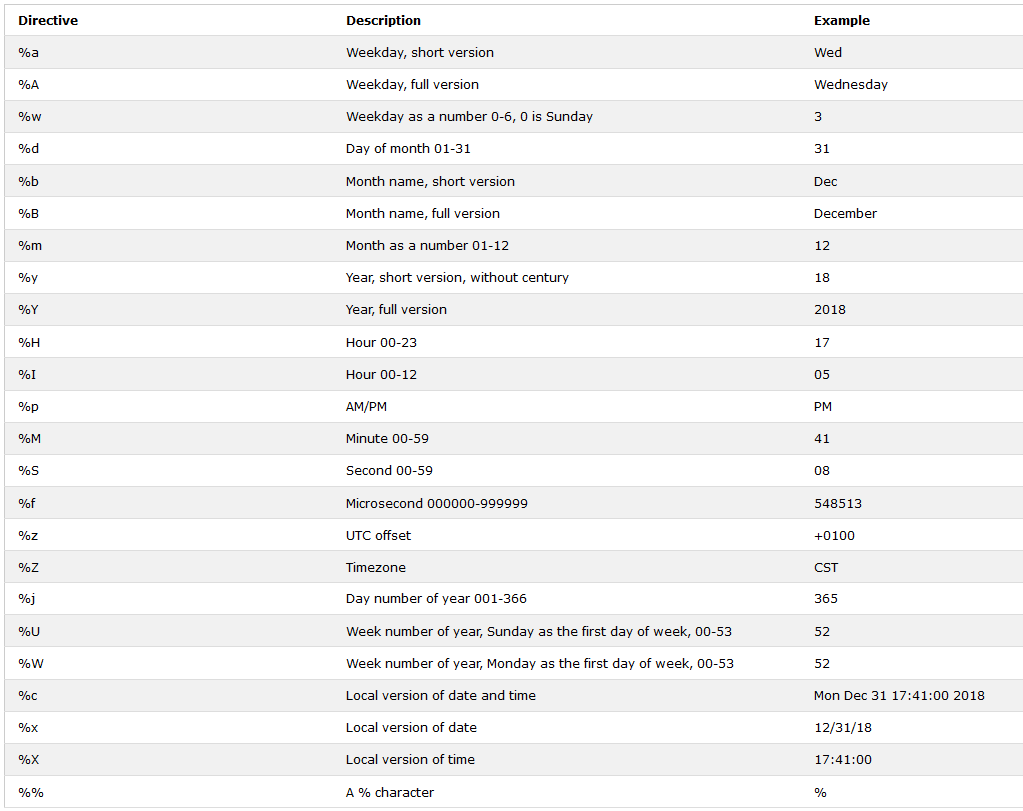
The datetime object has a method for formatting date objects into readable strings.

The method is called strftime(), and takes one parameter, format, to specify the format of the returned string:

Display the name of the month:

import datetime  
x = datetime.datetime(2018, 6, 1)  
print(x.strftime("%B"))

A reference of all the legal format codes:



# Python PIP

## What is PIP?

PIP is a package manager for Python packages, or modules if you like.

**Note:** If you have Python version 3.4 or later, PIP is included by default.

## What is a Package?

A package contains all the files you need for a module. Modules are Python code libraries you can include in your project.

## Check if PIP is Installed

Navigate your command line to the location of Python's script directory, and type the following:

Check PIP version:

C:\Users\Your Name\AppData\Local\Programs\Python\Python36-32\Scripts>pip --version

## Install PIP

If you do not have PIP installed, you can download and install it from this page: <https://pypi.org/project/pip/>

## Download a Package

Downloading a package is very easy.

Open the command line interface and tell PIP to download the package you want.

Navigate your command line to the location of Python's script directory, and type the following:

Download a package named "camelcase":

C:\Users\Your Name\AppData\Local\Programs\Python\Python36-32\Scripts>pip install camelcase

Now you have downloaded and installed your first package!

## Using a Package

Once the package is installed, it is ready to use. Import the "camelcase" package into your project. Import and use "camlecase":

import camelcase  
c = camelcase.CamelCase()  
txt = "hello world"  
print(c.hump(txt))

## Find Packages

Find more packages at <https://pypi.org/>.

# Python File Open

File handling is an important part of any web application.

Python has several functions for creating, reading, updating, and deleting files.

## File Handling

The key function for working with files in Python is the open() function.

The open() function takes two parameters; filename, and mode.

There are four different methods (modes) for opening a file:

"r" - Read - Default value. Opens a file for reading, error if the file does not exist

"a" - Append - Opens a file for appending, creates the file if it does not exist

"w" - Write - Opens a file for writing, creates the file if it does not exist

"x" - Create - Creates the specified file, returns an error if the file exist

In addition, you can specify if the file should be handled as binary or text mode

"t" - Text - Default value. Text mode

"b" - Binary - Binary mode (e.g. images)

## Syntax

To open a file for reading it is enough to specify the name of the file:

f = open("demofile.txt")

The code above is the same as:

f = open("demofile.txt", "rt")

Because "r" for read, and "t" for text are the default values, you do not need to specify them.

**Note:** Make sure the file exists, or else you will get an error.

## Open a File on the Server

Assume we have the following file, located in the same folder as Python:

demofile.txt

Hello! Welcome to demofile.txt  
This file is for testing purposes.  
Good Luck!

To open the file, use the built-in open() function.

The open() function returns a file object, which has a read() method for reading the content of the file:

f = open("demofile.txt", "r")  
print(f.read())

## Read Only Parts of the File

By default the read() method returns the whole text, but you can also specify how many character you want to return:

Return the 5 first characters of the file:

f = open("demofile.txt", "r")  
print(f.read(**5**))

## Read Lines

You can return one line by using the readline() method:

Read one line of the file:

f = open("demofile.txt", "r")  
print(f.readline())

By calling readline() two times, you can read the two first lines:

Read two lines of the file:

f = open("demofile.txt", "r")  
print(f.readline())  
print(f.readline())

By looping through the lines of the file, you can read the whole file, line by line:

Loop through the file line by line:

f = open("demofile.txt", "r")  
for x in f:  
  print(x)

# Python File Write

## Write to an Existing File

To write to an existing file, you must add a parameter to the open() function:

"a" - Append - will append to the end of the file

"w" - Write - will overwrite any existing content

Open the file "demofile.txt" and append content to the file:

f = open("demofile.txt", "a")  
f.write("Now the file has one more line!")

Open the file "demofile.txt" and overwrite the content:

f = open("demofile.txt", "w")  
f.write("Woops! I have deleted the content!")

**Note:** the "w" method will overwrite the entire file.

## Create a New File

To create a new file in Python, use the open() method, with one of the following parameters:

"x" - Create - will create a file, returns an error if the file exist

"a" - Append - will create a file if the specified file does not exist

"w" - Write - will create a file if the specified file does not exist

Create a file called "myfile.txt":

f = open("myfile.txt", "x")

Result: a new empty file is created!

Create a new file if it does not exist:

f = open("myfile.txt", "w")

# Python Delete File

## Delete a File

To delete a file, you must import the OS module, and run its os.remove() function:

Remove the file "demofile.txt":

import os  
os.remove("demofile.txt")

## Check if File exist:

To avoid getting an error, you might want to check if the file exists before you try to delete it:

Check if file exist, then delete it:

import os  
if os.path.exists("demofile.txt"):  
  os.remove("demofile.txt")  
else:  
  print("The file does not exists")

## Delete Folder

To delete an entire folder, use the os.rmdir() method:

Remove the folder "myfolder":

import os  
os.rmdir("myfolder")

**Note:** You can only remove empty folders.

# Python MySQL

Python can be used in database applications. One of the most popular database is MySQL.

## MySQL Database

To be able experiment with the code examples in this tutorial, you should have MySQL installed on your computer.

You can download a free MySQL database at <https://www.mysql.com/downloads/>.

## Install MySQL Driver

Python needs a MySQL driver to access the MySQL database. In this tutorial we will use the driver "MySQL Connector". We recommend that you use PIP to install "MySQL Connector".

PIP is most likely already installed in your Python environment. Navigate your command line to the location of PIP, and type the following:

Download and install "MySQL Connector":

C:\Users\Your Name\AppData\Local\Programs\Python\Python36-32\Scripts>python -m pip install mysql-connector

Now you have downloaded and installed a MySQL driver.

## Test MySQL Connector

To test if the installation was successful, or not, create a Python page with the following content:

demo\_mysql\_test.py:

import mysql.connector

If the above code was executed with no errors, "MySQL Connector" is installed and ready to be used.

## Create Connection

Start by creating a connection to the database.

Use the username and password from your MySQL database:

demo\_mysql\_connection.py:

import mysql.connector  
  
mydb = mysql.connector.connect(  
  host="localhost",  
  user="yourusername",  
  passwd="yourpassword"  
)  
  
print(mydb)

Now you can start querying the database using SQL statements.

# Python MySQL Create Database

## Creating a Database

To create a database in MySQL, use the "CREATE DATABASE" statement:

### Example

create a datebase named "mydatabase":

import mysql.connector  
  
mydb = mysql.connector.connect(  
  host="localhost",  
  user="yourusername",  
  passwd="yourpassword"  
)  
  
mycursor = mydb.cursor()  
  
mycursor.execute("CREATE DATABASE mydatabase")

If the above code was executed with no errors, you have successfully created a database.

## Check if Database Exists

You can check if a database exist by listing all databases in your system by using the "SHOW DATABASES" statement:

### Example

Return a list of your system's databases:

import mysql.connector  
  
mydb = mysql.connector.connect(  
  host="localhost",  
  user="yourusername",  
  passwd="yourpassword"  
)  
  
mycursor = mydb.cursor()  
  
mycursor.execute("SHOW DATABASES")  
  
for x in mycursor:  
  print(x)

Or you can try to access the database when making the connection:

### Example

Try connecting to the database "mydatabase":

import mysql.connector  
  
mydb = mysql.connector.connect(  
  host="localhost",  
  user="yourusername",  
  passwd="yourpassword",  
**database="mydatabase"**  
)

If the database does not exist, you will get an error.

# Python MySQL Create Table

## Creating a Table

To create a table in MySQL, use the "CREATE TABLE" statement.

Make sure you define the name of the database when you create the connection

### Example

create a datebase named "mydatabase":

import mysql.connector  
  
mydb = mysql.connector.connect(  
  host="localhost",  
  user="yourusername",  
  passwd="yourpassword",  
**database: "mydatabase"**  
)  
  
mycursor = mydb.cursor()  
  
mycursor.execute("CREATE TABLE customers (name VARCHAR(255), address VARCHAR(255))")

If the above code was executed with no errors, you have now successfully created a table.

## Check if Table Exists

You can check if a database exists by listing all tables in your database by using the "SHOW TABLES" statement:

### Example

Return a list of your system's databases:

import mysql.connector  
  
mydb = mysql.connector.connect(  
  host="localhost",  
  user="yourusername",  
  passwd="yourpassword",  
  database: "mydatabase"  
)  
  
mycursor = mydb.cursor()  
  
mycursor.execute("SHOW TABLES")  
  
for x in mycursor:  
  print(x)

## Primary Key

When creating a table, you should also create a column with a unique key for each record.

This can be done by defining a PRIMARY KEY.

We use the statement "INT AUTO\_INCREMENT PRIMARY KEY" which will insert a unique number for each record. Starting at 1, and increased by one for each record.

### Example

Create primary key when creating the table:

import mysql.connector  
  
mydb = mysql.connector.connect(  
  host="localhost",  
  user="yourusername",  
  passwd="yourpassword",  
  database: "mydatabase"  
)  
  
mycursor = mydb.cursor()  
  
mycursor.execute(sql = "CREATE TABLE customers (id INT AUTO\_INCREMENT PRIMARY KEY, name VARCHAR(255), address VARCHAR(255))")

If the table already exists, use the ALTER TABLE keyword:

### Example

Create primary key on an existing table:

import mysql.connector  
  
mydb = mysql.connector.connect(  
  host="localhost",  
  user="yourusername",  
  passwd="yourpassword",  
  database: "mydatabase"  
)  
  
mycursor = mydb.cursor()  
  
mycursor.execute("ALTER TABLE customers ADD COLUMN id INT AUTO INCREMENT PRIMARY KEY")

# Python MySQL Insert into Table

## Insert Into Table

To fill a table in MySQL, use the "INSERT INTO" statement.

### Example

Insert a record in the "customers" table:

import mysql.connector  
  
mydb = mysql.connector.connect(  
  host="localhost",  
  user="yourusername",  
  passwd="yourpassword",  
  database="mydatabase"  
)  
  
mycursor = mydb.cursor()  
  
sql = "INSERT INTO customers (name, address) VALUES (%s, %s)"  
val = ("John", "Highway 21")  
mycursor.execute(sql, val)  
 **mydb.commit()**  
print(mycursor.rowsaffected, "record inserted.")

**Important!** Notice the statement: mydb.commit(). It is required to make the changes, otherwise no changes are made to the table.

## Insert Multiple Rows

To insert multiple rows into a table, use the executemany() method.

The second parameter of the executemany() method is a list of tuples, containing the data you want to insert:

### Example

Fill the "customers" table with data:

import mysql.connector  
  
mydb = mysql.connector.connect(  
  host="localhost",  
  user="yourusername",  
  passwd="yourpassword",  
  database="mydatabase"  
)  
  
mycursor = mydb.cursor()  
  
sql = "INSERT INTO customers (name, address) VALUES (%s, %s)"  
val = [  
  ('Peter', 'Lowstreet 4'),  
  ('Amy', 'Apple st 652'),  
  ('Hannah', 'Mountain 21'),  
  ('Michael', 'Valley 345'),  
  ('Sandy', 'Ocean blvd 2'),  
  ('Betty', 'Green Grass 1'),  
  ('Richard', 'Sky st 331'),  
  ('Susan', 'One way 98'),  
  ('Vicky', 'Yellow Garden 2'),  
  ('Ben', 'Park Lane 38'),  
  ('William', 'Central st 954'),  
  ('Chuck', 'Main Road 989'),  
  ('Viola', 'Sideway 1633')  
]  
  
mycursor.executemany(sql, val)  
  
mydb.commit()  
  
print(mycursor.rowsaffected, "was inserted.")

## Get Inserted ID

You can get the id of the row you just inserted by asking the cursor object.

**Note:** If you insert more than one row, the id of the last inserted row is returned.

### Example

Insert one row, and return the ID:

import mysql.connector  
  
mydb = mysql.connector.connect(  
  host="localhost",  
  user="yourusername",  
  passwd="yourpassword",  
  database="mydatabase"  
)  
  
mycursor = mydb.cursor()  
  
sql = "INSERT INTO customers (name, address) VALUES (%s, %s)"  
val = ("Michelle", "Blue Village")  
mycursor.execute(sql, val)  
  
mydb.commit()  
  
print("1 record inserted, ID:", mycursor.lastrowid)

# Python MySQL Select From

## Select From a Table

To select from a table in MySQL, use the "SELECT" statement:

### Example

Select all records from the "customers" table, and display the result:

import mysql.connector  
  
mydb = mysql.connector.connect(  
  host="localhost",  
  user="yourusername",  
  passwd="yourpassword",  
  database="mydatabase"  
)  
mycursor = mydb.cursor()  
mycursor.execute("SELECT \* FROM customers")  
myresult = mycursor.fetchall()  
  
for x in myresult:  
  print(x)

**Note:** We use the fetchall() method, which fetches all rows from the last executed statement.

## Selecting Columns

To select only some of the columns in a table, use the "SELECT" statement followed by the column name(s):

### Example

Select only the name and address columns:

import mysql.connector  
  
mydb = mysql.connector.connect(  
  host="localhost",  
  user="yourusername",  
  passwd="yourpassword",  
  database="mydatabase"  
)  
mycursor = mydb.cursor()  
mycursor.execute("SELECT name, address FROM customers")  
myresult = mycursor.fetchall()  
  
for x in myresult:  
  print(x)

## Using the fetchone() Method

If you are only interested in one row, you can use the fetchone() method.

The fetchone() method will return the first row of the result:

### Example

Fetch only one row:

import mysql.connector  
  
mydb = mysql.connector.connect(  
  host="localhost",  
  user="yourusername",  
  passwd="yourpassword",  
  database="mydatabase"  
)  
  
mycursor = mydb.cursor()  
mycursor.execute("SELECT \* FROM customers")  
myresult = mycursor.fetchone()  
print(myresult)

# Python MySQL Where

## Select With a Filter

When selecting records from a table, you can filter the selection by using the "WHERE" statement:

### Example

Select record(s) where the address is "Park Lane 38": result:

import mysql.connector  
  
mydb = mysql.connector.connect(  
  host="localhost",  
  user="yourusername",  
  passwd="yourpassword",  
  database="mydatabase"  
)  
  
mycursor = mydb.cursor()  
sql = "SELECT \* FROM customers WHERE address ='Park Lane 38'"  
mycursor.execute(sql)  
myresult = mycursor.fetchall()  
  
for x in myresult:  
  print(x)

## Wildcard Characters

You can also select the records that starts, includes, or ends with a given letter or phrase.

Use the % to represent wildcard characters:

### Example

Select records where the address contains the word "way":

import mysql.connector  
  
mydb = mysql.connector.connect(  
  host="localhost",  
  user="yourusername",  
  passwd="yourpassword",  
  database="mydatabase"  
)  
mycursor = mydb.cursor()  
sql = "SELECT \* FROM customers WHERE address LIKE '%way%'"  
mycursor.execute(sql)  
myresult = mycursor.fetchall()  
  
for x in myresult:  
  print(x)

## Prevent SQL Injection

When query values are provided by the user, you should escape the values.

This is to prevent SQL injections, which is a common web hacking technique to destroy or misuse your database.

The mysql.connector module has methods to escape query values:

### Example

Escape query values by using the placeholder %s method:

import mysql.connector  
  
mydb = mysql.connector.connect(  
  host="localhost",  
  user="yourusername",  
  passwd="yourpassword",  
  database="mydatabase"  
)  
mycursor = mydb.cursor()  
  
sql = "SELECT \* FROM customers WHERE address = %s"  
adr = ("Yellow Garden 2", )  
  
mycursor.execute(sql, adr)  
mycursor.execute(sql)  
myresult = mycursor.fetchall()  
  
for x in myresult:  
  print(x)

# Python MySQL Delete From By

## Delete Record

You can delete records from an existing table by using the "DELETE FROM" statement:

### Example

Delete any record where the address is "Mountain 21":

import mysql.connector  
  
mydb = mysql.connector.connect(  
  host="localhost",  
  user="yourusername",  
  passwd="yourpassword",  
  database="mydatabase"  
)  
  
mycursor = mydb.cursor()  
  
sql = "DELETE FROM customers WHERE address = 'Mountain 21'"  
  
mycursor.execute(sql)  
  
mydb.commit()  
  
print(mycursor.rowcount, "record(s) deleted")

**Important!:** Notice the statement: mydb.commit(). It is required to make the changes, otherwise no changes are made to the table.

**Notice the WHERE clause in the DELETE syntax:** The WHERE clause specifies which record(s) that should be deleted. If you omit the WHERE clause, all records will be deleted!

## Prevent SQL Injection

It is considered a good practice to escape the values of any query, also in delete statements.

This is to prevent SQL injections, which is a common web hacking technique to destroy or misuse your database.

The mysql.connector module uses the placeholder %s to escape values in the delete statement:

### Example

Escape values by using the placholder %s method:

import mysql.connector  
  
mydb = mysql.connector.connect(  
  host="localhost",  
  user="yourusername",  
  passwd="yourpassword",  
  database="mydatabase"  
)  
mycursor = mydb.cursor()  
sql = "DELETE FROM customers WHERE address = %s"  
adr = ("Yellow Garden 2", )  
mycursor.execute(sql, adr)  
  
mydb.commit()  
  
print(mycursor.rowcount, "record(s) deleted")

# Python MySQL Update Table

## Update Table

You can update existing records in a table by using the "UPDATE" statement:

### Example

Overwrite the address column from "Valley 345" to "Canyoun 123":

import mysql.connector  
  
mydb = mysql.connector.connect(  
  host="localhost",  
  user="yourusername",  
  passwd="yourpassword",  
  database="mydatabase"  
)  
  
mycursor = mydb.cursor()  
sql = "UPDATE customers SET address = 'Canyon 123 WHERE address = 'Valley 345'"  
mycursor.execute(sql)  
mydb.commit()  
  
print(mycursor.rowcount, "record(s) affected")

**Important!** Notice the statement: mydb.commit(). It is required to make the changes, otherwise no changes are made to the table.

**Notice the WHERE clause in the UPDATE syntax:** The WHERE clause specifies which record or records that should be updated. If you omit the WHERE clause, all records will be updated!

## Prevent SQL Injection

It is considered a good practice to escape the values of any query, also in update statements.

This is to prevent SQL injections, which is a common web hacking technique to destroy or misuse your database.

The mysql.connector module uses the placeholder %s to escape values in the delete statement:

### Example

Escape values by using the placeholder %s method:

import mysql.connector  
  
mydb = mysql.connector.connect(  
  host="localhost",  
  user="yourusername",  
  passwd="yourpassword",  
  database="mydatabase"  
)  
  
mycursor = mydb.cursor()  
sql = "UPDATE customers SET address = %s WHERE address = %s"  
val = ("Valley 345", "Canyon 123")  
mycursor.execute(sql, val)  
  
mydb.commit()  
  
print(mycursor.rowcount, "record(s) affected")

# Python Built in Functions

Python has a set of built-in functions.

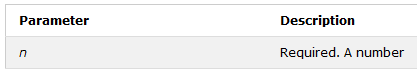
# Python abs() Function

The abs() function returns the absolute value of the specified number.

## Syntax

abs(n)

## Parameter Values



Return the absolute value of a number:

x = abs(-7.25) # 7.25

x = abs(3+5j) # 5.830951894845301

# Python all() Function

### Example

Check if all items in a list are True:

mylist = [True, True, True]  
x = all(mylist)

## Definition and Usage

The all() function returns True if all items in an iterable are true, otherwise it returns False.

If the iterable object is empty, the all() function also returns True.

## Syntax

all(iterable)

## Parameter Values

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| iterable | An iterable object (list, tuple, dictionary) |

## More Examples

### Example

Check if all items in a list are True:

mylist = [0, 1, 1]  
x = all(mylist)

### Example

Check if all items in a tuple are True:

mytuple = (0, True, False)  
x = all(mytuple)

### Example

Check if all items in a set are True:

myset = {0, 1, 0)  
x = all(myset)

### Example

Check if all items in a dictionary are True:

mydict = {0 : "Apple", 1 : "Orange"}  
x = all(mydict)

**Note:** When used on a dictionary, the all() function checks if all the keys are true, not the values.

# Python any() Function

### Example

Check if any of the items in a list are True:

mylist = [False, True, False]  
x = any(mylist)

## Definition and Usage

The any() function returns True if any item in an iterable are true, otherwise it returns False.

If the iterable object is empty, the any() function will return False.

## Syntax

any(iterable)

## Parameter Values

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| iterable | An iterable object (list, tuple, dictionary) |

## More Examples

### Example

Check if any item in a tuple is True:

mytuple = (0, 1, False)  
x = any(mytuple)

### Example

Check if any item in a set is True:

myset = {0, 1, 0)  
x = any(myset)

### Example

Check if any item in a dictionary is True:

mydict = {0 : "Apple", 1 : "Orange"}  
x = any(mydict)

**Note:** When used on a dictionary, the any() function checks if any of the keys are true, not the values.

# Python ascii() Function

### Example

Escape non-ascii characters:

x = ascii("My name is Ståle")

## Definition and Usage

The ascii() function returns a readable version of any object (Strings, Tuples, Lists, etc).

The ascii() function will replace any non-ascii characters with escape characters:

å will be replaced with \xe5.

## Syntax

ascii(object)

## Parameter Values

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| object | An object, like String, List, Tuple, Dictionary etc. |

# Python bin() Function

### Example

Return the binary version of 36:

x = bin(36)

## Definition and Usage

The bin() function returns the binary version of a specified integer.

The result will always start with the prefix 0b.

## Syntax

bin(n)

## Parameter Values

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| n | Required. An integer |

# Python bool() Function

### Example

Return the boolean value of 1:

x = bool(1)

## Definition and Usage

The bool() function returns the boolean value of a specified object.

The object will always return True, unless:

The object is empty, like [], (), {}  
The object is False  
The object is 0  
The object is None

## Syntax

bool(object)

## Parameter Values

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| object | Any object, like String, List, Number etc. |

# Python bytearray() Function

### Example

Return an array of 4 bytes:

x = bytearray(4)

## Definition and Usage

The bytearray() function returns a bytearray object.

It can convert objects into bytearray objects, or create empty bytearray object of the specified size.

## Syntax

bytearray(x, encoding, error)

## Parameter Values

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| x | A source to use when creating the bytearray object.  If it is an integer, an empty bytearray object of the specified size will be created.  If it is a String, make sure you specify the encoding of the source. |
| encoding | The encoding of the string |
| error | Specifies what to do if the encoding fails. |

# Python bytes() Function

### Example

Return an array of 4 bytes:

x = bytes(4)

## Definition and Usage

The bytes() function returns a bytes object.

It can convert objects into bytes objects, or create empty bytes object of the specified size.

The difference between bytes() and bytearray() is that bytes() returns an object that cannot be modified, and bytearray() returns an object that can be modified.

## Syntax

bytes(x, encoding, error)

## Parameter Values

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| x | A source to use when creating the bytes object.  If it is an integer, an empty bytes object of the specified size will be created.  If it is a String, make sure you specify the encoding of the source. |
| encoding | The encoding of the string |
| error | Specifies what to do if the encoding fails. |

# Python callable() Function

### Example

Check if a function is callable:

def x():  
  a = 5  
  
print(callable(x))

## Definition and Usage

The callable() function returns True if the specified object is callable, otherwise it returns False.

## Syntax

callable(object)

## Parameter Values

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| object | The object you want to test if it is callable or not. |

## More Examples

### Example

A normal variable is not callable:

x = 5  
  
print(callable(x))

# Python chr() Function

### Example

Get the character that represents the unicode 97:

x = chr(97)

## Definition and Usage

The chr() function returns the character that represents the specified unicode.

## Syntax

chr(number)

## Parameter Values

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| number | An integer representing a valid Unicode code point |

# Python compile() Function

### Example

Compile text as code, and the execute it:

mytext = 'print(55)'  
x = compile('mytext', 'test', 'eval')  
exec(x)

## Definition and Usage

The compile() function returns the specified source as a code object, ready to be executed.

## Syntax

compile(source, filename, mode, flag, dont\_inherit, optimize)

## Parameter Values

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| source | Required. The source to compile, can be a String, a Bytes object, or an AST object |
| filename | Required. The name of the file that the source comes from. If the source does not come from a file, you can write whatever you like |
| mode | Required. Legal values: eval - if the source is a single expression exec - if the source is a block of statements single - if the source is a single interactive statement |
| flags | Optional. How to compile the source. Default 0 |
| dont-inherit | Optional. How to compile the source. Default False |
| optimize | Optional. Defines the optimization level of the compiler. Default -1 |

## More Examples

### Example

Compile more than one statement, and the execute it:

mytext = 'print(55)\nprint(88)'  
x = compile('mytext', 'test', 'exec')  
exec(x)

# Python complex() Function

### Example

Convert the number 3 and imaginary number 5 into a complex number:

x = complex(3, 5)

## Definition and Usage

The complex() function returns a complex number by specifying a real number and an imaginary number.

## Syntax

complex(real, imaginary)

## Parameter Values

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| real | Required. A number representing the real part of the complex number. Default 0. The real number can also be a String, like this '3+5j', when this is the case, the second parameter should be omitted. |
| imaginary | Optional. A number representing the imaginary part of the complex number. Default 0. |

## More Examples

### Example

Convert a string into a complex number:

x = complex('3+5j')

# Python delattr() Function

### Example

Delete the "age" property from the "person" object:

class Person:  
  name = "John"  
  age = 36  
  country = "Norway"  
  
delattr(Person, 'age')

## Definition and Usage

The delattr() function will delete the specified attribute from the specified object.

## Syntax

delattr(object, attribute)

## Parameter Values

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| object | Required. An object. |
| attribute | Required. The name of the attribute you want to remove |

# Python dict() Function

### Example

Create a dictionary containing personal information:

x = dict(name = "John", age = 36, country = "Norway")

## Definition and Usage

The dict() function creates a dictionary.

A dictionary is a collection which is unordered, changeable and indexed.

Read more about dictionaries in the chapter: [Python Dictionaries](https://www.w3schools.com/python/python_dictionaries.asp).

## Syntax

dict(keyword arguments)

## Parameter Values

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| keyword arguments | Required. As may keyword arguments you like, separated by comma: key = value, key = value ... |

# Python dir() Function

### Example

Display the content of an object:

class Person:  
  name = "John"  
  age = 36  
  country = "Norway"  
  
print(dir(Person))

## Definition and Usage

The dir() function returns all properties and methods of the specified object, without the values.

This function will return all the properties and methods, even built-in properties which are default for all object.

## Syntax

dir(object)

## Parameter Values

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| object | The object you want to see the valid attributes of |

# Python divmod() Function

### Example

Display the quotient and the remainder of 5 divided by 2:

x = divmod(5, 2)

## Definition and Usage

The divmod() function returns a tuple containing the quotient  and the remainder when argument1 (divident) is divided by argument2 (divisor).

## Syntax

divmod(divident, divisor)

## Parameter Values

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| divident | A Number. The number you want to divide |
| divisor | A Number. The number you want to divide with |

# Python enumerate() Function

### Example

Convert a tuple into an enumerate object:

x = ('apple', 'banana', 'cherry')  
y = enumerate(x)

## Definition and Usage

The enumerate() function takes a collection (e.g. a tuple) and returns it as an enumerate object.

The enumerate() function adds a counter as the key of the enumerate object.

## Syntax

enumerate(iterable, start)

## Parameter Values

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| iterable | An iterable object |
| start | A Number. Defining the start number of the enumerate object. Default 0 |

# Python eval() Function

### Example

Evaluate the expression 'print(55)':

x = 'print(55)'  
eval(x)

## Definition and Usage

The eval() function evaluates the specified expression, if the expression is a legal Python statement, it will be executed.

## Syntax

eval(expression, globals, locals)

## Parameter Values

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| expression | A String, that will be evaluated as Python code |
| globals | Optional. A dictionary containing global parameters |
| locals | Optional. A dictionary containing local parameters |

# Python exec() Function

### Example

Execute a block of code:

x = 'name = "John"\nprint(name)'  
exec(x)

## Definition and Usage

The exec() function executes the specified Python code.

The exec() function accepts large blocks of code, unlike the eval() function which only accepts a single expression

## Syntax

exec(object, globals, locals)

## Parameter Values

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| object | A String, or a code object |
| globals | Optional. A dictionary containing global parameters |
| locals | Optional. A dictionary containing local parameters |

# Python filter() Function

### Example

Return the array with only values above 18:

ages = [5, 12, 17, 18, 24, 32]  
  
def myFunc(x):  
  if x < 18:  
    return False  
  else:  
    return True  
  
adults = filter(myFunc, ages)  
  
for x in adults:  
  print(x)

## Definition and Usage

The filter() function returns an iterator were the items are filtered through a function to test if the item is accepted or not.

## Syntax

filter(function, iterable)

## Parameter Values

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| function | A Function to be run for each item in the iterable |
| iterable | The iterable to be filtered |

# Python float() Function

### Example

Convert the number 3 into a floating point number:

x = float(3)

## Definition and Usage

The float() function converts the specified value into a floating point number.

## Syntax

float(value)

## Parameter Values

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| value | A number or a string that can be converted into a floating point number |

## More Examples

### Example

Convert a string into a floating point number:

x = float("3.500")

# Python format() Function

### Example

Format the number 0.5 into a percentage value:

x = format(0.5, '%')

## Definition and Usage

The format() function formats a specified value into a specified format.

## Syntax

format(value, format)

## Parameter Values

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| value | A value of any format |
| format | The format you want to format the value into. Legal values: '<' - Left aligns the result (within the available space) '>' - Right aligns the result (within the available space) '^' - Center aligns the result (within the available space) '=' - Places the sign to the left most position '+' - Use a sign to indicate if the result is positive or negative '-' - Use a sign for negative values only ' ' - Use a leading space for positive numbers ',' - Use a comma as a thousand separator '\_' - Use a underscore as a thousand separator 'b' - Binary format 'c' - Converts the value into the corresponding unicode character 'd' - Decimal format 'e' - Scientific format, with a lower case e 'E' - Scientific format, with an upper case E 'f' - Fix point number format 'F' - Fix point number format, upper case 'g' - General format 'G' - General format (using a upper case E for scientific notations) 'o' - Octal format 'x' - Hex format, lower case 'X' - Hex format, upper case 'n' - Number format '%' - Percentage format |

## More Examples

### Example

Format 255 into a hexadecimal value:

x = format(255, 'x')

# Python frozenset() Function

### Example

Freeze the list, and make it unchangeable:

mylist = ['apple', 'banana', 'cherry']  
x = frozenset(mylist)

## Definition and Usage

The frozenset() function returns an unchangeable frozenset object (which is like a set object, only unchangeable).

## Syntax

frozenset(iterable)

## Parameter Values

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| iterable | An iterable object, like list, set, tuple etc. |

## More Examples

### Example

Try to change the value of a frozenset item.

This will cause an error:

mylist = ['apple', 'banana', 'cherry']  
x = frozenset(mylist)  
x[1] = "strawberry"

# Python getattr() Function

### Example

Get the value of the "age" property of the "Person" object:

class Person:  
  name = "John"  
  age = 36  
  country = "Norway"  
  
x = getattr(Person, 'age')

## Definition and Usage

The getattr() function returns the value of the specified attribute from the specified object.

## Syntax

getattr(object, attribute, default)

## Parameter Values

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| object | Required. An object. |
| attribute | The name of the attribute you want to get the value from |
| default | Optional. The value to return if the attribute does not exist |

## More Examples

### Example

Use the "default" parameter to write a message when the attribute does not exist:

class Person:  
  name = "John"  
  age = 36  
  country = "Norway"  
  
x = getattr(Person, 'page', 'my message')

# Python globals() Function

### Example

Dispaly the global symbol table:

x = globals()  
print(x)

## Definition and Usage

The globals() function returns the global symbol table as a dictionary.

A symbol table contains necessary information about the current program

## Syntax

globals()

## Parameter Values

No parameters

## More Examples

### Example

Get the filename of the current program:

x = globals()  
print(x["\_\_file\_\_"])

# Python hasattr() Function

### Example

Check if the "Person" object has the "age" property:

class Person:  
  name = "John"  
  age = 36  
  country = "Norway"  
  
x = hasattr(Person, 'age')

## Definition and Usage

The hasattr() function returns True if the specified object has the specified attribute, otherwise False.

## Syntax

hasattr(object, attribute)

## Parameter Values

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| object | Required. An object. |
| attribute | The name of the attribute you want to check if exists |

|  |  |
| --- | --- |
| hash() | Returns the hash value of a specified object |
| help() | Executes the built-in help system |

# Python hex() Function

### Example

Convert 255 into hexadecimal value:

x = hex(255)

## Definition and Usage

The hex() function converts the specified number into a hexadecimal value.

The returned string always starts with the prefix 0x.

## Syntax

hex(number)

## Parameter Values

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| number | An Integer |

# Python id() Function

### Example

Return the unique id of a tuple object:

x = ('apple', 'banana', 'cherry')  
y = id(x)

## Definition and Usage

The id() function returns a unique id for the specified object.

All objects in Python has its own unique id.

The id is assigned to the object when it is created.

The id is the object's memory address, and will be different for each time you run the program. (except for some object that has a constant unique id, like integers from -5 to 256)

## Syntax

id(object)

## Parameter Values

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| object | Any object, String, Number, List, Class etc. |

# Python input() Function

### Example

Ask for the user's name and print it:

print('Enter your name:')  
x = input()  
print('Hello, ' + x)

## Definition and Usage

The input() function allows user input.

## Syntax

input(prompt)

## Parameter Values

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| prompt | A String, representing a default message before the input. |

## More Examples

### Example

Use the prompt parameter to write a message before the input:

x = input('Enter your name:')  
print('Hello, ' + x)

# Python int() Function

### Example

Convert the number 3.5 into an integer:

x = int(3.5)

## Definition and Usage

The int() function converts the specified value into an integer number.

## Syntax

int(value, base)

## Parameter Values

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| value | A number or a string that can be converted into an integer number |
| base | A number representing the number format. Default value: 10 |

## More Examples

### Example

Convert a string into an integer:

x = int("12")

# Python isinstance() Function

### Example

Check if the number 5 is an integer:

x = isinstance(5, int)

## Definition and Usage

The isinstance() function returns True if the specified object is of the specified type, otherwise False.

If the type parameter is a tuple, this function will return True if the object is one of the types in the tuple.

## Syntax

isinstance(object, type)

## Parameter Values

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| object | Required. An object. |
| type | A type or a class, or a tuple of types and/or classes |

## More Examples

### Example

Check if "Hello" is one of the types described in the type parameter:

x = isinstance("Hello", (float, int, str, list, dict, tuple))

### Example

Check if y is an instance of myObj:

class myObj:  
  name = "John"  
  
y = myObj()  
  
x = isinstance(y, myObj)

## Related Pages

The [issubclass()](https://www.w3schools.com/python/ref_func_issubclass.asp) function, to check if an object is a subclass of another object.

# Python issubclass() Function

### Example

Check if the class "myAge" is a subclass of "myObj":

class myAge:  
  age = 36  
  
class myObj(myAge):  
  name = "John"  
  age = myAge  
  
x = issubclass(myObj, myAge)

## Definition and Usage

The issubclass() function returns True if the specified object is a subclass of the specified object, otherwise False.

## Syntax

issubclass(object, subclass)

## Parameter Values

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| object | Required. An object. |
| subclass | A class object, or a tuple of class objects |

# Python iter() Function

### Example

Create an iterator object, and print the items:

x = iter(["apple", "banana", "cherry"])  
print(next(x))  
print(next(x))  
print(next(x))

## Definition and Usage

The iter() function returns an iterator object.

## Syntax

iter(object, subclass)

## Parameter Values

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| object | Required. An iterable object |
| sentinel | Optional. If the object is a callable object the iteration will stop when the returned value is the same as the sentinel |

# Python len() Function

### Example

Return the number of items in a list:

mylist = ["apple", "banana", "cherry"]  
x = len(mylist)

## Definition and Usage

The len() function returns the number of items in an object.

When the object is a string, the len() function returns the number of characters in the string.

## Syntax

len(object)

## Parameter Values

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| object | Required. An object. Must be a sequence or a collection |

## More Examples

### Example

Return the number of characters in a string:

mylist = "Hello"  
x = len(mylist)

# Python list() Function

### Example

Create a list containing fruit names:

x = list(('apple', 'banana', 'cherry'))

## Definition and Usage

The list() function creates a list object.

A list object is a collection which is ordered and changeable.

Read more about list  in the chapter: [Python Lists](https://www.w3schools.com/python/python_lists.asp).

## Syntax

list(iterable)

## Parameter Values

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| iterable | Required. A sequence, collection or an iterator object |

# Python locals() Function

### Example

Dispaly the local symbol table:

x = locals()  
print(x)

## Definition and Usage

The locals() function returns the local symbol table as a dictionary.

A symbol table contains necessary information about the current program.

## Syntax

locals()

## Parameter Values

No parameters

## More Examples

### Example

Get the filename of the current program:

x = locals()  
print(x["\_\_file\_\_"])

# Python map() Function

### Example

Calculate the length of each word in the tuple:

def myfunc(n):  
  return len(n)  
  
x = map(myfunc, ('apple', 'banana', 'cherry'))

## Definition and Usage

The map() function executes a specified function for each item in a iterable. The item is sent to the function as a parameter.

## Syntax

map(function, iterables)

## Parameter Values

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| function | Required. The function to execute for each item |
| iterable | Required. A sequence, collection or an iterator object. You can send as many iterables as you like, just make sure the function has one parameter for each iterable. |

## More Examples

### Example

Make new fruits by sending two iterable objects into the function:

def myfunc(a, b):  
return a + b  
  
x = map(myfunc, ('apple', 'banana', 'cherry'), ('orange', 'lemon', 'pineapple'))

# Python max() Function

### Example

Return the largest number:

x = max(5, 10)

## Definition and Usage

The max() function returns the item with the highest value, or the item with the highest value in an iterable.

If the values are strings, an alphabetically comparison is done.

## Syntax

max(n1, n2, n3, ...)

Or:

max(iterable)

## Parameter Values

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| n1, n2, n3, ... | One or more items to compare |

Or:

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| iterable | An iterable, with one or more items to compare |

## More Examples

### Example

Return the name with the highest value, ordered alphabetically:

x = max("Mike", "John", "Vicky")

### Example

Return the item in a tuple with the highest value:

a = (1, 5, 3, 9)  
x = max(a)

# Python memoryview() Function

### Example

Create and print a memoryview object:

x = memoryview(b"Hello")  
  
print(x)  
  
#return the Unicode of the first character  
print(x[0])  
  
#return the Unicode of the second character  
print(x[1])

## Definition and Usage

The memoryview() function returns a memory view object from a specified object.

## Syntax

memoryview(obj)

## Parameter Values

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| obj | A Bytes object or a Bytearray object. |

# Python min() Function

### Example

Return the lowest number:

x = max(5, 10)

## Definition and Usage

The min() function returns the item with the lowest value, or the item with the lowest value in an iterable.

If the values are strings, an alphabetically comparison is done.

## Syntax

min(n1, n2, n3, ...)

Or:

min(iterable)

## Parameter Values

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| n1, n2, n3, ... | One or more items to compare |

Or:

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| iterable | An iterable, with one or more items to compare |

## More Examples

### Example

Return the name with the lowest value, ordered alphabetically:

x = min("Mike", "John", "Vicky")

### Example

Return the item in a tuple with the lowest value:

a = (1, 5, 3, 9)  
x = min(a)

# Python next() Function

### Example

Create an iterator, and print the items one by one:

mylist = iter(["apple", "banana", "cherry"])  
x = next(mylist)  
print(x)  
x = next(mylist)  
print(x)  
x = next(mylist)  
print(x)

## Definition and Usage

The next() function returns the next item in an iterator.

You can add a default return value, to return if the iterable has reached to its end.

## Syntax

next(iterable, default)

## Parameter Values

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| iterable | Required. An iterable object. |
| default | Optional. An default value to return if the iterable has reached to its end. |

## More Examples

### Example

Return a default value when the iterable has reached to its end:

mylist = iter(["apple", "banana", "cherry"])  
x = next(mylist, "orange")  
print(x)  
x = next(mylist, "orange")  
print(x)  
x = next(mylist, "orange")  
print(x)  
x = next(mylist, "orange")  
print(x)

# Python object() Function

### Example

Create an empty object:

x = object()

## Definition and Usage

The object() function returns an empty object.

You cannot add new properties or methods to this object.

This object is the base for all classes, it holds the built-in properties and methods which are default for all classes.

## Syntax

object()

# Python oct() Function

### Example

Convert the number 12 into an octal value:

x = oct(12)

## Definition and Usage

The oct() function converts an integer into an octal string.

Octal strings in Python are prefixed with 0o.

## Syntax

oct(int)

## Parameter Values

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| int | An Integer Number |

# Python open() Function

### Example

Open a file and print the content:

f = open("demofile.txt", "r")  
print(f.read())

## Definition and Usage

The open() function opens a file, and returns it as a file object.

Read more about file handling in our chapters about [File Handling](https://www.w3schools.com/python/python_file_handling.asp).

## Syntax

open(file, mode)

## Parameter Values

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| file | The path and name of the file |
| mode | A string, define which mode you want to open the file in:  "r" - Read - Default value. Opens a file for reading, error if the file does not exist  "a" - Append - Opens a file for appending, creates the file if it does not exist  "w" - Write - Opens a file for writing, creates the file if it does not exist  "x" - Create - Creates the specified file, returns an error if the file exist  In addition you can specify if the file should be handled as binary or text mode  "t" - Text - Default value. Text mode  "b" - Binary - Binary mode (e.g. images) |

# Python ord() Function

### Example

Return the iteger that represents the character "h":

x = ord("h")

## Definition and Usage

The ord() function returns the number representing the unicode code of a specified character.

## Syntax

ord(character)

## Parameter Values

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| character | String, any character |

# Python pow() Function

### Example

Return the value of 4 to the power of 3 (same as 4 \* 4 \* 4):

x = pow(4, 3)

## Definition and Usage

The pow() function returns the value of x to the power of y (xy).

If a third parameter is present, it returns x to the power of y, modulus z.

## Syntax

pow(x, y, z)

## Parameter Values

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| x | A number, the base |
| y | A number, the exponent |
| z | Optional. A number, the modulus |

## More Examples

### Example

Return the value of 4 to the power of 3, modulus 5 (same as (4 \* 4 \* 4) % 5):

x = pow(4, 3, 5)

# Python print() Function

### Example

Print a message onto the screen:

 print("Hello World")

## Definition and Usage

The print() function prints the specified message to the screen, or other standard output device.

The message can be a string, or any other object, the object will be converted into a string before written to the screen.

## Syntax

print(object(s), separator=separator, end=end, file=file, flush=flush)

## Parameter Values

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| object(s) | Any object, and as many as you like. Will be converted to string before printed |
| sep='separator' | Optional. Specify how to separate the objects, if there is more than one. Default is '' |
| end='end' | Optional. Specify what to print at the end. Default is '\n' (line feed) |
| file | Optional. An object with a write method. Default is sys.stdout |
| flush | Optional. A Boolean, specifying if the output is flushed (True) or buffered (False). Default is False |

## More Examples

### Example

Print more than one object:

print("Hello", "how are you?")

### Example

Print a tuple:

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

### Example

Print two messages, and specify the separator:

print("Hello", "how are you?", sep=" ---")

|  |  |
| --- | --- |
| **property()** | Gets, sets, deletes a property |
|  |  |

# Python range() Function

### Example

Create a sequence of numbers from 0 to 5, and print each item in the sequence:

x = range(6)  
for n in x:  
  print(n)

## Definition and Usage

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.

## Syntax

range(start, stop, step)

## Parameter Values

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| start | Optional. An integer number specifying at which position to start. Default is 0 |
| stop | Optional. An integer number specifying at which position to endt. |
| step | Optional. An integer number specifying the incrementation. Default is 1 |

## More Examples

### Example

Create a sequence of numbers from 3 to 5, and print each item in the sequence:

x = range(3, 6)  
for n in x:  
  print(n)

### Example

Create a sequence of numbers from 3 to 20, but increment by 2 instead of 1:

x = range(3, 20, 2)  
for n in x:  
  print(n)

|  |  |
| --- | --- |
| **repr()** | Returns a readable version of an object |

# Python reversed() Function

### Example

Reverse the sequence of a list, and print each item:

alph = ["a", "b", "c", "d"]  
ralph = reversed(alph)  
for x in ralph:  
  print(x)

## Definition and Usage

The reversed() function returns a reversed iterator object.

## Syntax

reversed(sequence)

## Parameter Values

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| sequence | Required. Any iterable object |

# Python round() Function

### Example

Round a number to only two decimals:

x = round(5.76543, 2)  
print(x)

## Definition and Usage

The round() function returns a floating point number that is a rounded version of the specified number, with the specified number of decimals.

The default number of decimals is 0, meaning that the function will return the nearest integer.

## Syntax

round(number, digits)

## Parameter Values

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| number | Required. The number to be rounded |
| digits | Optional. The number of decimals to use when rounding the number. Default is 0 |

## More Examples

### Example

Round to the nearest integer:

x = round(5.76543)  
print(x)

# Python set() Function

### Example

Create a set containing fruit names:

x = set(('apple', 'banana', 'cherry'))

## Definition and Usage

The set() function creates a set object.

The items in a set list are unordered, so it will appear in random order.

Read more about sets in the chapter [Python Sets](https://www.w3schools.com/python/python_sets.asp).

## Syntax

set(iterable)

## Parameter Values

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| iterable | Required. A sequence, collection or an iterator object |

# Python setattr() Function

### Example

Change the value of the "age" property of the "person" object:

class Person:  
  name = "John"  
  age = 36  
  country = "Norway"  
  
setattr(Person, 'age', 40)

## Definition and Usage

The setattr() function sets the value of the specified attribute of the specified object.

## Syntax

setattr(object, attribute, value)

## Parameter Values

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| object | Required. An object. |
| attribute | Required. The name of the attribute you want to set |
| value | Required. The value you want to give the specified attribute |

## Related Pages

The [delattr()](https://www.w3schools.com/python/ref_func_delattr.asp) function, to remove an attribute

The [getattr()](https://www.w3schools.com/python/ref_func_getattr.asp) function, to get the value of an attribute

The [hasattr()](https://www.w3schools.com/python/ref_func_hasattr.asp) function, to check if an attribute exist

# Python slice() Function

### Example

Create a tuple and a slice object. Use the slice object to get only the two first items of the tuple:

a = ("a", "b", "c", "d", "e", "f", "g", "h")  
x = slice(2)  
print(a[x])

## Definition and Usage

The slice() function returns a slice object.

A slice object is used to specify how to slice a sequence. You can specify where to start the slicing, and where to end. You can also specify the step, which allows you to e.g. slice only every other item.

## Syntax

slice(start, end, step)

## Parameter Values

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| start | Optional. An integer number specifying at which position to start the slicing. Default is 0 |
| end | An integer number specifying at which position to end the slicing |
| step | Optional. An integer number specifying the step of the slicing. Default is 1 |

## More Examples

### Example

Create a tuple and a slice object. Start the slice object at position 3, and slice to position 5, and return the result:

a = ("a", "b", "c", "d", "e", "f", "g", "h")  
x = slice(3, 5)  
print(a[x])

### Example

Create a tuple and a slice object. Use the step parameter to return every third item:

a = ("a", "b", "c", "d", "e", "f", "g", "h")  
x = slice(0, 8, 3)  
print(a[x])

# Python sorted() Function

### Example

Sort a tuple:

a = ("a", "b", "c", "d", "e", "f", "g", "h")  
x = slice(2)  
print(a[x])

## Definition and Usage

The sorted() function returns a sorted list of the specified iterable object.

You can specify ascending or descending order. Strings are sorted alphabetically, and numbers are sorted numerically.

**Note:** You cannot sort a list that contains BOTH string values AND numeric values.

## Syntax

sorted(iterable, key=key, reverse=reverse)

## Parameter Values

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| iterable | Required. The sequence to sort, list, dictionary, tuple etc. |
| key | Optional. A Function to execute to decide the order. Default is None |
| reverse | Optional. A Boolean. False will sort ascending, True will sort descending. Default is False |

## More Examples

### Example

Sort numeric:

a = (1, 11, 2)  
x = sorted(a)  
print(a[x])

### Example

Sort ascending:

a = ("a", "b", "c", "d", "e", "f", "g", "h")  
x = sort(a, reverse=True)  
print(a[x])

# Python str() Function

### Example

Convert the number 3.5 into an string:

x = str(3.5)

## Definition and Usage

The str() function converts the specified value into a string.

## Syntax

str(object, encoding=encoding, errors=errors)

## Parameter Values

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| object | Any object. Specifies the object to convert into a string |
| encoding | The encoding of the object. Default is UTF-8 |
| errors | Specifies what to do if the decoding fails |

## More Examples

### Example

Convert a string into an integer:

x = int("12")

# Python sum() Function

### Example

Add all items in a tuple, and return the result:

a = (1, 2, 3, 4, 5)  
x = sum(a)

## Definition and Usage

The sum() function returns a number, the sum of all items in an iterable.

## Syntax

sum(iterable, start)

## Parameter Values

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| iterable | Required. The sequence to sum |
| start | Optional. A value that is added to the return value |

## More Examples

### Example

Start with the number 7, and add all the items in a tuple to this number:

a = (1, 2, 3, 4, 5)  
x = sum(a, 7)

|  |  |
| --- | --- |
| **@staticmethod()** | Converts a method into a static method |

# Python tuple() Function

### Example

Create a tuple containing fruit names:

x = tuple(('apple', 'banana', 'cherry'))

## Definition and Usage

The tuple() function creates a tuple object.

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

Read more about sets in the chapter [Python Tuples](https://www.w3schools.com/python/python_tuples.asp).

## Syntax

tuple(iterable)

## Parameter Values

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| iterable | Required. A sequence, collection or an iterator object |

# Python type() Function

### Example

Return the type of these objects:

a = ('apple', 'banana', 'cherry')  
b = "Hello World"  
c = 33  
  
x = type(a)  
y = type(b)  
z = type(c)

## Definition and Usage

The type() function returns the type of the specified object

## Syntax

type(object, bases, dict)

## Parameter Values

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| object | Required. If only one parameter is specified, the type() function returns the type of this object |
| bases | Optional. Specifies the base classes |
| dict | Optional. Specifies the namespace with the definition for the class |

# Python vars() Function

### Example

Return the \_\_dict\_\_ atribute of an object called Person:

class Person:  
  name = "John"  
  age = 36  
  country = "norway"  
  
x = vars(Person)

## Definition and Usage

The vars() function returns the \_\_dic\_\_ attribute of an object.

The \_\_dict\_\_ attribute is a dictionary containing the object's changeable attributes.

**Note:** calling the vars() function without parameters will return a dictionary containing the local symbol table.

## Syntax

vars(object)

## Parameter Values

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| object | Any object with a \_\_dict\_\_attribute |

# Python zip() Function

### Example

Join two tuples together:

a = ("John", "Charles", "Mike")  
b = ("Jenny", "Christy", "Monica", "Vicky")  
  
x = zip(a, b)

## Definition and Usage

The zip() function returns a zip object, which is an iterator of tuples where the first item in each passed iterator is paired together, and then the second item in each passed iterator are paired together etc.

If the passed iterators have different lengths, the iterator with least items decides the length of the new iterator.

## Syntax

zip(iterator1, iterqator2, iterator3 ...)

## Parameter Values

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| iterator1, iterqator2, iterator3 ... | Iterator objects that will be joined together |

## More Examples

### Example

If one tuple contains more items, these items are ignored:

a = ("John", "Charles", "Mike")  
b = ("Jenny", "Christy", "Monica", "Vicky")  
  
x = zip(a, b)