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`:

Example

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:

Example

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):

Example

Save this code in the file `mymodule.py`

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

Example

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:

Example

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.

Example

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:

Example

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.

Example

The module named `mymodule` has one function and one dictionary:

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

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

Example

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 math Module

Python has a built-in module that you can use for mathematical tasks.

The `math` module has a set of methods and constants.

Math Methods

Method	Description
math.acos()	Returns the arc cosine of a number
math.acosh()	Returns the inverse hyperbolic cosine of a number
math.asin()	Returns the arc sine of a number
math.asinh()	Returns the inverse hyperbolic sine of a number
math.atan()	Returns the arc tangent of a number in radians
math.atan2()	Returns the arc tangent of y/x in radians
math.atanh()	Returns the inverse hyperbolic tangent of a number
math.ceil()	Rounds a number up to the nearest integer

math.comb()	Returns the number of ways to choose k items from n items without repetition and order
math.copysign()	Returns a float consisting of the value of the first parameter and the sign of the second parameter
math.cos()	Returns the cosine of a number
math.cosh()	Returns the hyperbolic cosine of a number
math.degrees()	Converts an angle from radians to degrees
math.dist()	Returns the Euclidean distance between two points (p and q), where p and q are the coordinates of that point
math.erf()	Returns the error function of a number
math.erfc()	Returns the complementary error function of a number
math.exp()	Returns E raised to the power of x
math.expm1()	Returns E^x - 1
math.fabs()	Returns the absolute value of a number
math.factorial()	Returns the factorial of a number
math.floor()	Rounds a number down to the nearest integer
math.fmod()	Returns the remainder of x/y
math.frexp()	Returns the mantissa and the exponent, of a specified number
math.fsum()	Returns the sum of all items in any iterable (tuples, arrays, lists, etc.)

math.gamma()	Returns the gamma function at x
math.gcd()	Returns the greatest common divisor of two integers
math.hypot()	Returns the Euclidean norm
math.isclose()	Checks whether two values are close to each other, or not
math.isfinite()	Checks whether a number is finite or not
math.isinf()	Checks whether a number is infinite or not
math.isnan()	Checks whether a value is NaN (not a number) or not
math.isqrt()	Rounds a square root number downwards to the nearest integer
math.ldexp()	Returns the inverse of math.frexp() which is x * (2**i) of the given numbers x and i
math.lgamma()	Returns the log gamma value of x
math.log()	Returns the natural logarithm of a number, or the logarithm of number to base
math.log10()	Returns the base-10 logarithm of x
math.log1p()	Returns the natural logarithm of 1+x
math.log2()	Returns the base-2 logarithm of x
math.perm()	Returns the number of ways to choose k items from n items with order and without repetition
math.pow()	Returns the value of x to the power of y
math.prod()	Returns the product of all the elements in an iterable

math.radians()	Converts a degree value into radians
math.remainder()	Returns the closest value that can make numerator completely divisible by the denominator
math.sin()	Returns the sine of a number
math.sinh()	Returns the hyperbolic sine of a number
math.sqrt()	Returns the square root of a number
math.tan()	Returns the tangent of a number
math.tanh()	Returns the hyperbolic tangent of a number
math.trunc()	Returns the truncated integer parts of a number

Math Constants

Constant	Description
math.e	Returns Euler's number (2.7182...)
math.inf	Returns a floating-point positive infinity
math.nan	Returns a floating-point NaN (Not a Number) value
math.pi	Returns PI (3.1415...)
math.tau	Returns tau (6.2831...)

Python Random Module

Python has a built-in module that you can use to make random numbers.

The `random` module has a set of methods:

Method	Description
seed()	Initialize the random number generator
getstate()	Returns the current internal state of the random number generator
setstate()	Restores the internal state of the random number generator
getrandbits()	Returns a number representing the random bits
randrange()	Returns a random number between the given range
randint()	Returns a random number between the given range
choice()	Returns a random element from the given sequence
choices()	Returns a list with a random selection from the given sequence
shuffle()	Takes a sequence and returns the sequence in a random order
sample()	Returns a given sample of a sequence

random()	Returns a random float number between 0 and 1
uniform()	Returns a random float number between two given parameters
triangular()	Returns a random float number between two given parameters, you can also set a mode parameter to specify the midpoint between the two other parameters
betavariate()	Returns a random float number between 0 and 1 based on the Beta distribution (used in statistics)
expovariate()	Returns a random float number based on the Exponential distribution (used in statistics)
gammavariate()	Returns a random float number based on the Gamma distribution (used in statistics)
gauss()	Returns a random float number based on the Gaussian distribution (used in probability theories)
lognormvariate()	Returns a random float number based on a log-normal distribution (used in probability theories)
normalvariate()	Returns a random float number based on the normal distribution (used in probability theories)

vonmisesvariate()	Returns a random float number based on the von Mises distribution (used in directional statistics)
paretovariate()	Returns a random float number based on the Pareto distribution (used in probability theories)
weibullvariate()	Returns a random float number based on the Weibull distribution (used in statistics)

Python Requests Module

Example

Make a request to a web page, and print the response text:

```
import requests

x = requests.get('https://w3schools.com/python/demopage.htm')

print(x.text)
```

Definition and Usage

The `requests` module allows you to send HTTP requests using Python.

The HTTP request returns a Response Object with all the response data (content, encoding, status, etc).

Download and Install the Requests Module

Navigate your command line to the location of PIP, and type the following:

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

Syntax

```
requests.methodname(params)
```

Methods

Method	Description
delete(*url, args*)	Sends a DELETE request to the specified url
get(*url, params, args*)	Sends a GET request to the specified url
head(*url, args*)	Sends a HEAD request to the specified url
patch(*url*, *data, args*)	Sends a PATCH request to the specified url
post(*url, data, json, args*)	Sends a POST request to the specified url
put(*url*, *data, args*)	Sends a PUT request to the specified url
request(*method*, *url*, *args*)	Sends a request of the specified method to the specified

Python statistics Module

Python statistics Module

Python has a built-in module that you can use to calculate mathematical statistics of numeric data.

The `statistics` module was new in Python 3.4.

Statistics Methods

Method	Description
statistics.harmonic_mean()	Calculates the harmonic mean (central location) of the given data
statistics.mean()	Calculates the mean (average) of the given data
statistics.median()	Calculates the median (middle value) of the given data
statistics.median_grouped()	Calculates the median of grouped continuous data
statistics.median_high()	Calculates the high median of the given data
statistics.median_low()	Calculates the low median of the given data
statistics.mode()	Calculates the mode (central tendency) of the given numeric or nominal data
statistics.pstdev()	Calculates the standard deviation from an entire population
statistics.stdev()	Calculates the standard deviation from a sample of data

statistics.pvariance()	Calculates the variance of an entire population
statistics.variance()	Calculates the variance from a sample of data

Python Classes and 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`:

Example

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:

Example

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:

Example

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 belong to the object.

Let us create a method in the Person class:

Example

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 current instance of the class, and is used to access variables that belong to the class.

The self Parameter

The `self` parameter is a reference to the current instance of the class, 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:

Example

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:

Example

Set the age of p1 to 40:

```
p1.age = 40
```

Delete Object Properties

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

Example

Delete the age property from the p1 object:

```
del p1.age
```

Delete Objects

You can delete objects by using the `del` keyword:

Example

Delete the p1 object:

```
del p1
```

The pass Statement

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

Example

```
class Person:
  pass
```

Python User Input

User Input

Python allows for user input.

That means we are able to ask the user for input.

The method is a bit different in Python 3.6 than Python 2.7.

Python 3.6 uses the `input()` method.

Python 2.7 uses the `raw_input()` method.

The following example asks for the username, and when you entered the username, it gets printed on the screen:

Python 3.6

```
username = input("Enter username:")
print("Username is: " + username)
```

Python 2.7

```
username = raw_input("Enter username:")
print("Username is: " + username)
```

Python stops executing when it comes to the `input()` function, and continues when the user has given some input.

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 File read() Method

Example

Read the content of the file "demofile.txt":

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

Definition and Usage

The `read()` method returns the specified number of bytes from the file. Default is -1 which means the whole file.

Syntax

```
file.read()
```

Parameter Values

Parameter	Description

size	Optional. The number of bytes to return. Default -1, which means the whole file.

More examples

Example

Read the content of the file "demofile.txt":

```
f = open("demofile.txt", "r")
print(f.read(33))
```

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 exists

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.

Python File close() Method

Example

Close a file after it has been opened:

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

Definition and Usage

The `close()` method closes an open file.

You should always close your files, in some cases, due to buffering, changes made to a file may not show until you close the file.

Syntax

```
file.close()
```

Parameter Values

No parameters

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

Example

Open the file "demofile2.txt" and append content to the file:

```
f = open("demofile2.txt", "a")
f.write("Now the file has more content!")
f.close()

#open and read the file after the appending:
f = open("demofile2.txt", "r")
print(f.read())
```

Example

Open the file "demofile3.txt" and overwrite the content:

```
f = open("demofile3.txt", "w")
f.write("Woops! I have deleted the content!")
f.close()

#open and read the file after the appending:
f = open("demofile3.txt", "r")
print(f.read())
```

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

Example

Create a file called "myfile.txt":

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

Result: a new empty file is created!

Example

Create a new file if it does not exist:

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

Python File write() Method

Example

Open the file with "a" for appending, then add some text to the file:

```
f = open("demofile2.txt", "a")
f.write("See you soon!")
f.close()

#open and read the file after the appending:
f = open("demofile2.txt", "r")
print(f.read())
```

Definition and Usage

The `write()` method writes a specified text to the file.

Where the specified text will be inserted depends on the file mode and stream position.

`"a"`: The text will be inserted at the current file stream position, default at the end of the file.

`"w"`: The file will be emptied before the text will be inserted at the current file stream position, default 0.

Syntax

```
file.write(byte)
```

Parameter Values

Parameter	Description
byte	The text or byte object that will be inserted.

More examples

Example

The same example as above, but inserting a line break before the inserted text:

```
f = open("demofile2.txt", "a")
f.write("\nSee you soon!")
f.close()

#open and read the file after the appending:
f = open("demofile2.txt", "r")
print(f.read())
```

Python File read() Method

Example

Read the content of the file "demofile.txt":

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

Definition and Usage

The `read()` method returns the specified number of bytes from the file. Default is -1 which means the whole file.

Syntax

```
file.read()
```

Parameter Values

Parameter	Description
size	Optional. The number of bytes to return. Default -1, which means the whole file.

More examples

Example

Read the content of the file "demofile.txt":

```
f = open("demofile.txt", "r")
print(f.read(33))
```

Python File Methods

Python has a set of methods available for the file object.

Method	Description
close()	Closes the file

detach()	Returns the separated raw stream from the buffer
fileno()	Returns a number that represents the stream, from the operating system's perspective
flush()	Flushes the internal buffer
isatty()	Returns whether the file stream is interactive or not
read()	Returns the file content
readable()	Returns whether the file stream can be read or not
readline()	Returns one line from the file
readlines()	Returns a list of lines from the file
seek()	Change the file position
seekable()	Returns whether the file allows us to change the file position
tell()	Returns the current file position
truncate()	Resizes the file to a specified size

writable()	Returns whether the file can be written to or not
write()	Writes the specified string to the file
writelines()	Writes a list of strings to the file

Python File readline() Method

Example

Read the first line of the file "demofile.txt":

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

Definition and Usage

The `readline()` method returns one line from the file.

You can also specified how many bytes from the line to return, by using the size parameter.

Syntax

```
file.readline(size)
```

Parameter Values

Parameter	Description
size	Optional. The number of bytes from the line to return. Default -1, which means the whole line.

More examples

Example

Call `readline()` twice to return both the first and the second line:

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

Example

Return only the five first bytes from the first line:

```
f = open("demofile.txt", "r")
print(f.readline(5))
```