Python Module

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 Try Except

The try block lets you test a block of code for errors.

The except block lets you handle the error.

The finally block lets you execute code, regardless of the result of the try- and except blocks.

Exception Handling

When an error occurs, or exception as we call it, Python will normally stop and generate an error message.

These exceptions can be handled using the try statement:

Example

The try block will generate an exception, because x is not defined:

try:  
  print(x)  
except:  
  print("An exception occurred")

Since the try block raises an error, the except block will be executed.

Without the try block, the program will crash and raise an error:

Example

This statement will raise an error, because x is not defined:

print(x)

Many Exceptions

You can define as many exception blocks as you want, e.g. if you want to execute a special block of code for a special kind of error:

Example

Print one message if the try block raises a NameError and another for other errors:

try:  
  print(x)  
except NameError:  
  print("Variable x is not defined")  
except:  
  print("Something else went wrong")

Else

You can use the else keyword to define a block of code to be executed if no errors were raised:

Example

In this example, the try block does not generate any error:

try:  
  print("Hello")  
except:  
  print("Something went wrong")  
else:  
  print("Nothing went wrong")

Finally

The finally block, if specified, will be executed regardless if the try block raises an error or not.

Example

try:  
  print(x)  
except:  
  print("Something went wrong")  
finally:  
  print("The 'try except' is finished")

This can be useful to close objects and clean up resources:

Example

Try to open and write to a file that is not writable:

try:  
  f = open("demofile.txt")  
  f.write("Lorum Ipsum")  
except:  
  print("Something went wrong when writing to the file")  
finally:  
  f.close()

The program can continue, without leaving the file object open.

Raise an exception

As a Python developer you can choose to throw an exception if a condition occurs.

To throw (or raise) an exception, use the raise keyword.

Example

Raise an error and stop the program if x is lower than 0:

x = -1  
  
if x < 0:  
  raise Exception("Sorry, no numbers below zero")

The raise keyword is used to raise an exception.

You can define what kind of error to raise, and the text to print to the user.

Example

Raise a TypeError if x is not an integer:

x = "hello"  
  
if not type(x) is int:  
  raise TypeError("Only integers are allowed")