Modules and Packages

Modules in Python are simply Python files with the .py extension, which implement a set of functions. Modules are imported from other modules using the import command. Before you go ahead and import modules, check out the full list of built-in modules in the Python Standard library.

When a module is loaded into a running script for the first time, it is initialized by executing the code in the module once. If another module in your code imports the same module again, it will not be loaded twice but once only - so local variables inside the module act as a "singleton" - they are initialized only once.

If we want to import module math, we simply import the module:

```
# import the library
import math
```

```
# use it (ceiling rounding)
math.ceil(2.4)
```

```
3
```

Exploring built-in modules

While exploring modules in Python, two important functions come in handy - the dir and help functions. dir functions show which functions are implemented in each module. Let us see the below example and understand better.

```
print(dir(math))
```

```
['__doc__', '__loader__', '__name__', '__package__', '__spec__', 'acos', 'acosh', 'as
```

When we find the function in the module we want to use, we can read about it more using the help function, inside the Python interpreter:

```
help(math.ceil)
```

```
Help on built-in function ceil in module math:

ceil(x, /)
    Return the ceiling of x as an Integral.
```

```
        This is the smallest integer >= x.
```

Writing modules

Writing Python modules is very simple. To create a module of your own, simply create a new .py file with the module name, and then import it using the Python file name (without the .py extension) using the import command.

Writing packages

Packages are name-spaces which contain multiple packages and modules themselves. They are simply directories, but with a twist.

The twist is, each package in Python is a directory which MUST contain a special file called ***_init_*.py**. This file can be empty, and it indicates that the directory it contains is a Python package, so it can be imported the same way a module can be imported.

If we create a directory called foo, which marks the package name, we can then create a module inside that package called bar. We also must not forget to add the ***_init_*.py** file inside the foo directory.

To use the module bar, we can import it in two ways:

```
# Just an example, this won't work
import foo.bar
```

```
# OR could do it this way
from foo import bar
```

In the first method, we must use the foo prefix whenever we access the module bar. In the second method, we don't, because we import the module to our module's name-space.

The ***_init_*.py** file can also decide which modules the package exports as the API, while keeping other modules internal, by overriding the ***_all_*** variable, like so:

```
__init__.py
__all__=[bar]
```

Errors and Exception Handling

In this section, we will learn about Errors and Exception Handling in Python. You've might have definitely encountered errors by this point in the course. For example:

```
print('hello)
```

Note how we get a SyntaxError, with the further description that it was an End of Line Error (EOL) while scanning the string literal. This is specific enough for us to see that we forgot a single quote at the end of the line. Understanding of these various error types will help you debug your code much faster.

This type of error and description is known as an Exception. Even if a statement or expression is syntactically correct, it may cause an error when an attempt is made to execute it. Errors detected during execution are called exceptions and are not unconditionally fatal.

You can check out the full list of built-in exceptions here. Now, let's learn how to handle errors and exceptions in our own code.

try and except

The basic terminology and syntax used to handle errors in Python is the **try** and **except** statements. The code which can cause an exception to occur is put in the *try* block and the handling of the exception are the implemented in the *except* block of code. The syntax form is:

```
try:
   You do your operations here...
   ...
except ExceptionI:
   If there is ExceptionI, then execute this block.
except ExceptionII:
   If there is ExceptionII, then execute this block.
   ...
else:
   If there is no exception then execute this block.
```

Using just except, we can check for any exception: To understand better let's check out a sample code that opens and writes a file:

```
from os import write
try:
  f=open('testfile','w')
  f.write('Test write this')
except IOError:
```

```
  print("Error: Could not find file or read data")
else:
  print("content written successfully")
  f.close()
```

```
   content written successfully
```

Now, let's see what happens when we don't have write permission? (opening only with 'r'):

```
try:
  f=open('testfile','r')
  f.write('Test write this')
except IOError:
  print("Error:Could not find file or read data")
else:
  print("Content written successfully")
  f.close()
```

```
   Error:Could not find file or read data
```

Notice, how we only printed a statement! The code still ran and we were able to continue doing actions and running code blocks. This is extremely useful when you have to account for possible input errors in your code. You can be prepared for the error and keep running code, instead of your code just breaking as we saw above.

We could have also just said except: if we weren't sure what exception would occur. For example:

```
try:
  f=open('testfile','w')
  f.write('test write this')
except:
  print("Error:Could not find file or read data")
else:
  print("Content written successfully")
  f.close()
```

```
   Content written successfully
```

Now, we don't actually need to memorize the list of exception types! Now what if we keep wanting to run code after the exception occurred? This is where **finally** comes in.

finally

The finally: Block of code will always be run regardless if there was an exception in the try code block. The syntax is:

```
try:
   Code block here
   ...
   Due to any exception, this code may be skipped!
finally:
   This code block would always be executed.
```

For example:

```
try:
  f=open("testfile","w")
  f.write("test write statement")
finally:
  print("Always execute finally code blocks")
```

```
Always execute finally code blocks
```

We can use this in conjunction with except. Let's see a new example that will take into account a user putting in the wrong input:

```
def askint():
  try:
    val=int(input("plese enter an integer:"))
  except:
    print("Looks like you did not enter an integer")
  finally:
    print("Finally,I executed")
  print(val)
```

```
askint
```

```
<function __main__.askint>
```

```
askint()
```

```
plese enter an integer:123
Finally,I executed
123
```

Check how we got an error when trying to print val (because it was properly assigned). Let's find the right solution by asking the user and checking to make sure the input type is an integer:

```
def askint():
  try:
    val=int(input("plese enter an integer:"))
  except:
```

```
    print("Looks like you did not enter an integer")
    val=int(input("Try again-plese enter an ineger:"))
  finally:
    print("Finally,I executed")
  print(val)


askint()
```

```
 plese enter an integer:str
 Looks like you did not enter an integer
 Try again-plese enter an ineger:123
 Finally,I executed
 123
```

Hmmm...that only did one check. How can we continually keep checking? We can use a while loop!

```
def askint():
  while True:
    try:
      val=int(input("plese enter an integer:"))
    except:
      print("Looks like you did not enter an integer")
      continue
    else:
      print('Yep thats an integer')
      break
    finally:
      print("Finally,I executed")
    print(val)


askint()
```

```
 plese enter an integer:asd
 Looks like you did not enter an integer
 Finally,I executed
 plese enter an integer:str
 Looks like you did not enter an integer
 Finally,I executed
 plese enter an integer:wee
 Looks like you did not enter an integer
 Finally,I executed
 plese enter an integer:abc
 Looks like you did not enter an integer
 Finally,I executed
 plese enter an integer:123
 Yep thats an integer
 Finally,I executed
```

Database connectivity and operations using Python.

For Example, the following is the example of connecting with MySQL database "my_database1" and creating table grades1 and inserting values inside it.

```
#!/usr/bin/python
import sqlite3
#connecting with the database.
db =sqlite3.connect("my_database1.db")
# Drop table if it already exist using execute() method.
db.execute("drop table if exists grades1")
# Create table as per requirement
db.execute("create table grades1(id int,name text,score int)")
#inserting values inside the created table
db.execute("insert into grades1(id, name, score) values(101,'John',99)")
db.execute("insert into grades1(id,name,score) values(102,'reddy',90)")
db.execute("insert into grades1(id,name,score) values(103,'savrav',80)")
db.execute("insert into grades1(id,name,score) values(104,'Catchy',85)")
db.execute("insert into grades1(id,name,score) values(105,'krish',95)")
```

```
<sqlite3.Cursor at 0x7f9728818960>
```

```
db.commit()
```

```
results=db.execute("select * from grades1 order by id")
for row in results:
  print(row)
print("-"*60)
```

```
(101, 'John', 99)
(102, 'reddy', 90)
(103, 'savrav', 80)
(104, 'Catchy', 85)
(105, 'krish', 95)
------------------------------------------------------------
```

```
results=db.execute("select * from grades1 where name ='reddy'")
for row in results:
  print(row)
print("-"*60)
```

```
(102, 'reddy', 90)
------------------------------------------------------------
```

```
results=db.execute("select * from grades1 where score>=90")
for row in results:
  print(row)
print("-"*60)
```

```
(101, 'John', 99)
(102, 'reddy', 90)
```

```
    (105, 'krish', 95)
    ------------------------------------------------------------
```

```
results=db.execute("select * from grades1 order by score desc")
for row in results:
  print(row)
print("-"*60)
```

```
    (101, 'John', 99)
    (105, 'krish', 95)
    (102, 'reddy', 90)
    (104, 'Catchy', 85)
    (103, 'savrav', 80)
    ------------------------------------------------------------
```

```
results=db.execute("select name,score from grades1 order by score")
for row in results:
  print(row)
print("-"*60)
```

```
    ('savrav', 80)
    ('Catchy', 85)
    ('reddy', 90)
    ('krish', 95)
    ('John', 99)
    ------------------------------------------------------------
```

```
results=db.execute("select name,score from grades1 order by score")
for row in results:
  print(row)
```

```
    ('savrav', 80)
    ('Catchy', 85)
    ('reddy', 90)
    ('krish', 95)
    ('John', 99)
```