A GUIDE TO

WEB PROGRAMMING WITH python

TABLE OF CONTENTS

INTRODUCTION

Python can be used to do almost anything you can dream up, including the seemingly old-school task of developing websites. Web development is far from dead, and many programmers end up liking Python for web development much better than the old standby languages.

This eBook offers you a great way to get started using Python for web development.

OBJECT ORIENTED PROGRAMMING I

Python can be used with or without Object Oriented Programming. What makes Python a strong language is the fact that it gives you the freedom to work the way you feel comfortable. However, it is very important to decide as to when one should opt for implementing object oriented programming.

Python is interpreted line by line which is why many people prefer to use it the way it is interpreted. To make the most out of Python it is best to identify objects. While working with only data it makes sense to stick to Python data structures but when we work on programs that are dynamic in nature, interactive programs, Application models, games etc. where many times we have to deal with behavior of entities and there is no stored data then it is a good idea to go for OOPS. An object oriented program will have classes which will consist of interacting objects. Every object can process data, and interact with other objects.

In Python you can define a class and in this class you can further define the attributes of an object; just as in case of any other object oriented programming language. You can also define class variables, data members, methods and yes, even inheritance is possible but for now we will just stick to the fundamentals of how a class is defined and how to define methods.

How to create a class:

```
class class_name:
```

- “class” statement is used to define a class. It should be followed by the name of the class and after the class name you must use colon ‘:’
- A class has a __init__() method. This is a special method that is called whenever an object is created. It is important to understand here that a method is different from function in the sense that it is defined within a class and the first parameter while defining a method is reference to the instance of the class and it is called “self”. However when a method of the class is called no value is passed for self.

```
def __init__(self):
```

The interesting thing that you need to know here is that “self” is not a reserved keyword, you can think of it as a strict rule. “self” refers to the newly created object of a class. Another important thing that you need to know is that the __init__() method is something similar but not exactly a constructor. You may have this doubt if you have sound knowledge any other object oriented languages. We say this because the object is created before __init__() method is called. However, there is no doubt about the fact that it is the __init__() method that is used to initialize the variables of an object.

So, with this information in mind we now proceed towards writing our first class in Python.

```
1  class Program():
2
3  def__init__(self,*args,**kwargs):
4
5  self.lang = input("What Language?: ")
6
7  self.version = float(input("Version?: "))
8
9  self.skill = input("What skill level?: ")
10
11 p1 = Program()
```

Here "input" is used to prompt the user to provide an input value. When you execute the program the user is prompted to provide information about language and the value is assigned to self.lang. Similarly self.version and self.skill take the next two values. Now if you access these values with p1.lang, p1.version and p1.self you will get the same values that you had provided.

With this we come to the end of our first chapter on Object Oriented Programming.

OBJECT ORIENTED PROGRAMMING II

Welcome to part 2 of object oriented programming in Python. In this chapter we will start from where we left in Object Oriented Programming part 1 and take the concept of OOPs a step further. So, let's take a look at the program that we had developed:

```
class Program():
      def__init__(self,*args,**kwargs):
            self.lang = input("What Language?: ")
            self.version = float(input("Version?: "))
            self.skill = input("What skill level?: ")

p1 = Program()
```

Now, in our next step we would define a new method with which we can update the object that we call the method for. So, we write the following method:

```
def upgrade(self):
new_version=input("what version?:")
print("We have updated the version for",self.lang)
self.version=new_version
```

On adding this piece of code our program would look something like this:

```
class Program():
      def__init__(self,*args,**kwargs):
            self.lang = input("What Language?: ")
            self.version = float(input("Version?: "))
            self.skill = input("What skill level?: ")

     def upgrade(self):
           new_version=input("what version?:")
           print("We have updated the version for",self.lang)
          self.version=new_version

p1 = Program()
```

Analysis of upgrade() method:

1. While defining the method we write it as

```
1 | def upgrade(self):
```

What's important to note here is that it is not important to pass 'self' as an object. As a matter of fact, you won't get any error if you don't pass self but we put it there to denote that it is a part of the object.

2. 'new_version' can also be defined as 'self.new_version'. The difference between the two is that the latter will be the attribute of the entire object whereas the former will remain an attribute internal to the method. When upgrade() method is called, it will prompt the user to provide the value for the new version and that value will be assigned to new_version attribute.

3. The next statement is:

```
1 | print("We have updated the version for",self.lang)
```

It prints "We have updated the version for" followed by the old version of the language. Notice that here, we make a call to self.lang which is not the attribute of this method however, it is an attribute of the object and so we can get away with it.

4. We then assign the value of new_version to self.version using:

```
1 | self.version=new_version
```

Which is the old version value defined in the __init__() method.

Execution of the program

Now, let's have a look at how the statements are executed when we run the program:

1. The moment you execute the program, the __init__() program will prompt you with "What Language?:" statement. Here let's say you respond with "Python". This string "Python" will be assigned to self.lang.

2. The next statement of the __init__() function will be called now and you will be asked about "Version?:". You say: "2.7". The value "2.7" is assigned to self.version.

3. Now the __init__() function will prompt you to answer the third question "What skill level?:". You can say "Beginner". The value "Beginner" is assigned to self.skill.

4. To check whether the values have been assigned as expected you can type p1.lang, p1.version or p1.skill to check if the value that you provided has been assigned correctly or not.

5. Now to call the upgrade method type "p1.upgrade()", you will now be taken to the upgrade() method and you will be prompted to answer ("what version?:") you say "3.4". You would now get a message "We have updated the version for Python". If you look at the code, in __init__() function we have defined self.lang as

```
1 | self.lang = input("What Language?: ")
```

Now if you look at the upgrade() method, it says print("We have updated the version for",self.lang). It refers to self.lang which is an attribute of the object defined in __init__() method and when we execute the upgrade method it refers to self.lang attribute to print

this statement so the last word of the message , i.e. “Python” is the value that we assigned to self.lang.

6. Now if you call p1.version you will see that it displays the value of 3.4. In step 2. We had assigned it a value of 2.7 and it step 5. It was changed to 3.4.

With this we come to the end of this chapter. It is good this to play around with the code and try to update the other values as well.

MODULES

When we close the interpreter after we have tried our Python code, all the functions and logic that we had defined are lost. So when we have to work on longer pieces of code we prefer to save our work in a file and when we have to execute the program it is better to provide the file name as input to the interpreter rather than typing the code directly on the interpreter.

When you work on a bigger project in Python it is a wise thing to split your program into different files so that you are able to easily maintain your code. You can define certain important functions, definitions and classes in a module so that it can be used whenever they are required in a program and when you want to access the functions of that particular module you just simply have to import it in your program file.

Many times Python developers code something interesting and then feel that it would be good to share the functions that they have defined with other developers so they create or package their code in a module and make it available to others online. So, any developer who is looking for something similar can simply down load the module and use it in his work. These modules are completely open source. When you are working in a commercial set up or a closed source you are working with python modules that come under restrictive license but there are several modules that don't have any restrictive license.

How to get modules?

Try to find answer to this question on Google and you will be confused by the response. There is too much of scattered information which is the reason why we have designed this chapter for you. In this chapter we are going to talk about the present day norm for getting modules and how to make them work in your python project.

First thing that you need to know is that all the important and major modules that we would be using for these series can be installed with the help of pip. Earlier pip was not used for the installation of python modules, there were several ways to install python packages and there was lot of confusion as well. However, now pip comes with python by default and most people prefer to use pip. Installing by pip is really simple you just have to give one command. So, if you have a Windows operating system you will have to give the following command on your command prompt:

```
pip install module_name
```

In Windows we give command on the command prompt whereas in Mac we will use bash and terminal on linux.

"When I give a pip command, my system says it's not a recognized command"

If you get such a message when you give a pip command then it means that pip is not in your path.

In order to add pip to the path (for windows, it will vary for other operating systems): go to **control panel> system and security>system> advanced system settings.** This will open "System Properties" windows, now go to the **advanced tab** and

click on Environmental variables. It will show you the path, click on **Edit** button to edit the path. Here, you have to look for the path where Python is installed such as C:/Python 34; is by default the path where Python 3.4 would be installed. SO, in the same way to access pip you need to find out where it is installed in your system. So, if pip is available at C:/Python34/Scripts/pip then give this path at your command prompt.

People can also have lot of problem in figuring out where their Python installation is. You can find out where your standard library is when you install a module. Suppose you import a module, say the io module:

```
1 | import io
```

Now give the following command:

```
1 | Import io.__file__
```

This will give you the path for where the file actually is. This seems easy on Windows but it can be little difficult to retrieve this information on Linux or Mac OS.

So, this is how you can install Python modules with pip. However, sometimes pip may not work especially if you have a 64-bit Windows operating system. In such a scenario please go to http://www.lfd.uci.edu/~gohlke/pythonlibs/

With this we come to the end of this chapter on Modules-I.

MODULES - II

In the previous chapter the main focus was on how to acquire the modules. In this chapter you will learn more about modules and how to interact with them. While dealing with modules it is important to know where it goes when it is downloaded. Once you are in the Python directory (we have learnt about that in the previous chapters), you will find a folder called lib. All the standard library modules are in this lib folder. All modules that came with the installation of Python can be found here. There are third party modules as well which are installed into “site packages” inside the “lib” folder. When you click on “site packages” folder you will see modules that have been personally installed.

When you import a module in your program it works as if all the lines of code in the module are written in your script. Modules are full of Python code and to know more about what a module is all about, you must go through its code. Most developers heavily comment in modules so that they can provide better understanding about the code. When you import a module, Python is going to look for it at different locations. The first location will be where your script is, the second location is the “lib” folder and third location is the “site packages”.

So, the primary location where the Python is going to look for the code is local, ie., where your script is installed. Now to create a local module, you have to create a python file (in our example we stored the file with the name sample_module.py) in the local directory where your code is installed.

To get a better understanding let's create a small sample module which will have the following code in it:

```
def output(text):
    Print text
```

So, the 'output' function in the module basically takes text as input and prints it. *Now open your main python program file and import the module as shown below:*

```
import sample_module
```

Now that you have imported this module you can reference things within it. *So, let's try to call the output function from our program as follows:*

```
sample_module.output('Hello')
```

So, when you execute the program "Hello" will be printed on the shell.

Now, another way to call the output function is:

```
import output from sample_module;
output('Hello')
```

The output for this will also be the same. However, we use this statement when we are sure that we just need to call one function from the module.

Now, another interesting thing is to rename a function of a module in your script:

```
1  import output from sample_module as o;
2  o('Hello')
```

This will give the same result as well.

*Suppose there are several functions in your module then you can use a * to import everything as follows:*

```
1  from sample_module import  *
2  output('Hello')
```

So, even this piece of code will have the same result.

In the end there is just one small advice regarding naming your modules. While creating a module be careful about naming them in order to avoid any sort of conflicts. As mentioned above, when you import a module, Python first looks for it in the local directory and if you already have some other old python file with the same name in that directory then it may happen that your program will import the wrong code. So, before naming your module make sure that no file of that name exists in the local, lib or sire packages folder.

With this we come to the end of this chapter and this section of 'Python Programming Review' as well.

SQLITE INTRO

Welcome to the first chapter of section 3- Basic Database (SQLite) with Python. Here we are going to learn about SQLite and databases. Whenever there is a need for data collection or analysis it is important to store the data that you are collecting. The data that you collect can be stored in a text or a csv file. However, the issue with storing string data in this manner is that these files can easily get corrupted and navigation through these files can also become little difficult and this is why developers prefer to use data bases. SQL database is actually a collection of tables and most people have vision of a table when they talk of databases. Every table has columns and rows. Columns are generally of different data types and rows are data entries.

It is easier to work with data bases as they help in organizing data, making navigation simpler and also allow locking mechanism. SQLite is considered to be one of the most easiest and light-weight databases. As the name suggests SQLite is a very light version of SQL database that allows you to use some of the major SQL queries such as inserting, selecting, deleting etc. So, you can easily do all the basic stuff with it. SQLite is included in the standard library and so, you don't have to install anything extra to access the database. For our chapter we are using SQLite 3.The version does not actually matter because most of the code used in this module is SQL which is altogether different programming language in itself and hence variation of versions is not going to make much of a difference.

A database is like a csv file having columns and rows and no other third dimension. Files are made for the purpose of viewing and using information all at once by the user. Database on the other hand is used to produce reports and yield results based on the information stored in a machine. Databases are used when lots of machines are interacting with each other and files are used when the information is to be directly accessed by a human.

Files can be used to store data but it has some serious drawbacks. Most file system fail to provide transaction support which is required mainly when several users try to access the information concurrently. Databases allow fast indexing which cannot be expected from a file. Indexing is trying to retrieve data based on the indexed attribute. This is a very strong feature of databases. Another thing is that files are stored in directories with some filename and that is it. In databases we can draw relation between entries in different table and with the help of small and very easy commands defined by structured query language (SQL) we are able to retrieve, create, delete or edit related data.

A table comprises of rows and columns. All entries are stored in rows. However, the columns need to be defined in advance. Adding or removing rows is very easy but since a table is defined and entries are made then it is difficult to add or remove columns. Therefore, it is important to plan the data types and columns for a table before creating it.

With that information in mind we now proceed to the next chapter to learn more about SQLite.

DATABASE & SQLITE TABLE

Welcome to the next chapter on SQLite where you will learn how to create a database and a SQLite table.

Creating a database

1. The first thing that you need to do in order to access SQLite is to import its module using the following statement.

```
import sqlite3
```

2. Next you need to specify a connection to an existing database.

```
conn=sqlite3.connnect('tutorial.db')
```

Here conn is the connection with the database and all the information within it.

3. We will now define the cursor

```
c=conn.cursor()
```

Cursor in sqlite is same as what you see in and MySQL. You can use the cursor to perform whatever actions that you need to do on the database.

4. Now close the connection

```
conn.close()
So, this is how your code looks like:
1
Import sqlite3
conn=sqlite3.connnect('tutorial.db')
c=conn.cursor()
conn.close()
```

5. Run this program to create a database by the name chapter.db.

Creating a Table in a Database

We will now make slight changes to the code mentioned above to demonstrate how a table is created. The new statements added have been highlighted with bold text for you to get a better understanding. Here, we will show have the 'CREATE' command of SQL can be used in python for creating tables.

1. To create a table you will have to create a function after the cursor is defined.

```
Import sqlite3
conn=sqlite3.connnect('tutorial.db')
c=conn.cursor()
def create_table():
conn.close()
```

2. In this function use the cursor to execute the SQL commands. Here, we will execute the SQL command to create a table by the name 'example'. Coders generally use all caps for SQL specific commands and no caps for the other things. It is just a standard that is followed to keep things convenient. So, you see that "CREATE TABLE" is in caps (because they carry meaning and values in SQL) and the name of the table "example" is non-caps.

This standard process is followed by most coders so that it is easier to read the code.

```
Import sqlite3
conn=sqlite3.connnect('tutorial.db')
c=conn.cursor()
def create_table():
        c.execute("CREATE TABLE example (Language VARCHAR, Version REAL, skill TEXT")
conn.close()
```

Language, Version and Skill are names of the columns and VARCHAR, REAL and TEXT define the datatype of the values that can be stored in these columns

3. Now to run the command to create table, call the function as shown below:

```
create_table()
```

4. Now run the code to create a table.

If you run the code again then you will face an "Operation Error" saying that the Table already exists in the database. You cannot create the same table again and again.

With this we come to the end of how to create database and tables and in the next chapter we will learn to insert data into the table.

INSERTING DATA

Welcome to the third chapter of SQLite where you are going to learn about how to insert data in SQLite table using Python. We are now going to take the same code that we worked on in the last chapter and then make changes in it to take it further.

This is the code that we are going to work on:

```
Import sqlite3
conn=sqlite3.connnect('tutorial.db')
c=conn.cursor()
def create_table():
        c.execute("CREATE TABLE example (Language VARCHAR, Version REAL, skill TEXT")
conn.close()
```

All the new code that will be added will be highlighted in bold for better understanding.

We will now create another function:

```
def enter_data():
```

We will now again use "c.execute()" function to execute the SQL query for inserting data. *The SQL command for inserting data has the following syntax:*

```
INSERT INTO TABLE_NAME VALUES(value1, value2….)
```

So, for this code we write:

```
c.execute("INSERT INTO example VALUES('Python', 2.7, 'Beginner')")
```

We will now add two more sets of data:

```
c.execute("INSERT INTO example VALUES('Python', 3.3, 'Intermediate')")
c.execute("INSERT INTO example VALUES('Python', 3.4, 'Expert')")
```

And then call enter_data() function

So, the final piece of code looks like this:

```
Import sqlite3
conn=sqlite3.connnect('tutorial.db')
c=conn.cursor()
def create_table():
       c.execute("CREATE TABLE example (Language VARCHAR, Version REAL, skill TEXT")

def enter_data():
      c.execute("INSERT INTO example VALUES('Python', 2.7, 'Beginner')")
      c.execute("INSERT INTO example VALUES('Python', 3.3, 'Intermediate')");
      c.execute("INSERT INTO example VALUES('Python', 3.4, 'Expert')")

enter_data()

#conn.close()
```

Now, execute the code. The conn.close() statement has been commented so that you can understand how things work. Now if you go to the source code you will find a chapter.db.journal file which is a temporary file. This file is created to store the values as they are being inserted into the database. Large applications are often accessed by several users at the same time. What if two people try to enter a post at exact same time? Well, the values are stored in this file and then at the end of the code when we call conn.close() statement the content of the temporary file move into the database.

So, now again if you run the code and remove the comment on the conn.close() statement:

```
Import sqlite3
conn=sqlite3.connnect('tutorial.db')
c=conn.cursor()
def create_table():
       c.execute("CREATE TABLE example (Language VARCHAR, Version REAL, skill TEXT")

def enter_data():
      c.execute("INSERT INTO example VALUES('Python', 2.7, 'Beginner')")
      c.execute("INSERT INTO example VALUES('Python', 3.3, 'Intermediate')")
      c.execute("INSERT INTO example VALUES('Python', 3.4, 'Expert')")

enter_data()

conn.close()
```

You will not find the temporary file in the database now.

You can also do the following instead of using conn.close():

```
Import sqlite3
conn=sqlite3.connnect('tutorial.db')
c=conn.cursor()
def create_table():
       c.execute("CREATE TABLE example (Language VARCHAR, Version REAL, skill TEXT")

def enter_data():
      c.execute("INSERT INTO example VALUES('Python', 2.7, 'Beginner')")
      c.execute("INSERT INTO example VALUES('Python', 3.3, 'Intermediate')")
      c.execute("INSERT INTO example VALUES('Python', 3.4, 'Expert')")
      conn.commit()
enter_data()
```

conn.close() closes the connection, However, you don't want to close the connection everytime and then reconnect to the database again. So, conn.commit() can be used after the SQL statements have been executed. This is same as edit and save, but not closing the file so that it can be used again. However if you check again you will not see the temporary file.

So, both conn.commit() and conn.close() move the content of the temporary file into the database.So, this is how we can insert hard coded data into the database. But what if we have to enter data dynamically? *i.e.* the values are passed through a function which is actually the case? We discuss that in the next chapter on inserting dynamic data.

INSERTING DYNAMIC DATA

Welcome to this chapter on inserting dynamic data. In real time scenario data is always entered dynamically and that's why programs are made. There is no point entering hardcoded data. To explain the concept we will create a function which will prompt user to insert values and those values will be inserted into the data base.

This is the code that we had written for inserting hardcoded data:

```
Import sqlite3
conn=sqlite3.connnect('tutorial.db')
c=conn.cursor()
def create_table():
       c.execute("CREATE TABLE example (Language VARCHAR, Version REAL, skill TEXT")

def enter_data():
      c.execute("INSERT INTO example VALUES('Python', 2.7, 'Beginner')")
      c.execute("INSERT INTO example VALUES('Python', 3.3, 'Intermediate')")
      c.execute("INSERT INTO example VALUES('Python', 3.4, 'Expert')")
      conn.commit()
enter_data()
```

We now define a new function:

```
def enter_dynamic_data():
      lang=input("What Language?")
      version=float(Input("What Version?")
      skill=input("What skill level?")
      c.execute("INSERT INTO example(Language, Version,skill)  VALUES(?,?,?)",(lang, version,skill))
      conn.commit()
```

So, the final code looks like this:

```
Import sqlite3
conn=sqlite3.connnect('tutorial.db')
c=conn.cursor()
def create_table():
       c.execute("CREATE TABLE example (Language VARCHAR, Version REAL, skill TEXT")

def enter_data():
      c.execute("INSERT INTO example VALUES('Python', 2.7, 'Beginner')")
      c.execute("INSERT INTO example VALUES('Python', 3.3, 'Intermediate')")
      c.execute("INSERT INTO example VALUES('Python', 3.4, 'Expert')")
      conn.commit()

def enter_dynamic_data():
      lang=input("What Language?")
      version=float(Input("What Version?")
      skill=input("What skill level?")
      c.execute("INSERT INTO example(Language, Version,skill)  VALUES(?,?,?)",(lang, version,skill))
      conn.commit()
enter_dynamic_data()
```

So, when you execute this program you will be prompted to feed in the values for Language, version and skill level. Your inputs will be inserted in the database. But how would you know if the values have been inserted in the database? For that you would have to move on to the next chapter on "Reading Data"

READING DATA

Welcome to the chapter on Reading Data from SQLite. Reading data from the database is important in order to access and verify the details that have been stored in it already. We will be continuing with the same code that we have used in the previous chapters.

To read data we will define the following function:

```
def read_data_from_database():
```

In next step we write the SQL query to read all the data from the example table. The data base query is:

SELECT * FROM example.

Here * stands for 'ALL'. *In python we will assign this command to a variable as follows:*

```
sql = "SELECT * FROM example"
```

Now to retrieve data from the table we will have to access each row. *Hence, we will use the 'for' loop as shown below:*

```
for row in c.execute(sql):
        print (row)
```

To get a particular value from row you can use indexing. *So, in the above function add the following statement:*

```
print(row[0])
```

So, the final function will look as follows:

```
def read_data_from_database():
        sql = "SELECT * FROM example"
        for row in c.execute(sql):
              print (row)
              print(row[0])
```

The final full code will be as follows:

```
Import sqlite3
conn=sqlite3.connnect('tutorial.db')
c=conn.cursor()
def create_table():
       c.execute("CREATE TABLE example (Language VARCHAR, Version REAL, skill TEXT")

def enter_data():
      c.execute("INSERT INTO example VALUES('Python', 2.7, 'Beginner')")
      c.execute("INSERT INTO example VALUES('Python', 3.3, 'Intermediate')")
      c.execute("INSERT INTO example VALUES('Python', 3.4, 'Expert')")
      conn.commit()

def enter_dynamic_data():
      lang=input("What Language?")
      version=float(Input("What Version?")
      skill=input("What skill level?")
      c.execute("INSERT INTO example(Language, Version,skill)  VALUES(?,?,?)",(lang, version,skill))
      conn.commit()

def read_data_from_database():
       sql = "SELECT * FROM example"
       for row in c.execute(sql):
             print (row)
             print(row[0])
read_data_from_database()
```

The output for this will be as follows:

```
('Python', 2.7, 'Beginner')
Python
('Python', 3.3, 'Intermediate')
Python
('Python', 3.4, 'Expert')
Python
```

To get a better understanding let's now try to read the data of only that row where the skill value is 'Beginner' For this we will make following changes to the read_data_from_database() function.

```
def read_data_from_database():
        sql = "SELECT * FROM example WHERE skill== 'Beginner'"
        for row in c.execute(sql):
              print (row)
```

Now if you execute the new code you will get the following output:

```
('Python', 2.7, 'Beginner')
```

Let's suppose that you want to read data dynamically for this we will have to do some modifications to the above program. We will now create a function that will prompt the user to enter the value for skill and then the output will be retrieved accordingly.

The read_data_from_database() function will now look like this:

```
def read_data_from_database():
        what_skill = input("What skill level are we looking for?")
        sql = "SELECT * FROM example WHERE skill=?"
        for row in c.execute(sql, [(what_skill)]):
              print (row)
```

If you now execute this function you will be prompted to enter the skill level, enter Beginner. What skill level are we looking for? 'Beginner'

The following result will be displayed:

```
('Python', 2.7, 'Beginner')
```

Now suppose you want to search based on two values – skill and language:

```
def read_data_from_database():
        what_skill = input("What skill level are we looking for?")
        what_language = input("What language?:")

        sql = "SELECT * FROM example WHERE skill=? AND Language=?"
        for row in c.execute(sql, [(what_skill),(what_language)]):
              print (row)
```

Now when you execute the code you will be prompted with two questions. *Answer as follows:*

```
What skill level are we looking for? Beginner
What language?: Python
```

Your output will be as follows:

```
('Python', 2.7, 'Beginner')
```

With this we come to the end of this chapter. The next would be based on Limit, update and delete.

LIMIT • UPDATE • DELETE

In this chapter we will learn about Limit, update and delete.

LIMIT

To understand the concept of Limit we make the following changes to the read_data_from_database() function as follows:

```
def read_data_from_database():
        sql = "SELECT * FROM example LIMIT 2"
        for row in c.execute(sql):
              print (row)
```

UPDATE

This will output first two rows of data:

```
('Python', 2.7, 'Beginner')
 ('Python', 3.3, 'Intermediate')
```

Limit is generally used while updating or deleting or sometimes you may want to have a look at some number of rows in the database.

Now, look at how the UPDATE command works. UPDATE is used to edit the information in the database. Let's say that we want to find out which rows have skill as "Beginner" and then change it to value of "beginner".

```
def read_data_from_database():
       sql = "UPDATE example SET skill='beginner ' where skill ='Beginner '"
      c.execute(sql):
       sql = "SELECT * FROM example"
       for row in c.execute(sql):
             print (row)
```

Now if you execute the code, you will get the following output:

```
('Python', 2.7, 'beginner')
('Python', 3.3, 'Intermediate')
('Python', 3.4, 'Expert')
```

DELETE

Now , let's try to delete the row that has skill value of 'beginner'.

```
def read_data_from_database():
        sql = "SELECT * FROM example"
       for row in c.execute(sql):
             print (row)
       print(20*'#')
      sql = "DELETE * FROM example where skill='Beginner'"
      c.execute(sql):
for row in c.execute(sql):
             print (row)
conn.commit()
```

So, the output now will be:

```
('Python', 2.7, 'beginner')
('Python', 3.3, 'Intermediate')
('Python', 3.4, 'Expert')
####################
('Python', 3.3, 'Intermediate')
('Python', 3.4, 'Expert')
```

URLLIB MODULE

Here we are going to learn about the urllib (stands for url library) in Python. 'urllib' is a very important Python internet module, often used in Python networking / Internet Programming. Whenever you have to deal with HTTP protocols(associated with webpages, port No. 80), you would most certainly think of using urllib module as it deals with handling of connection, basic authentication, reading data, transfer of data, cookies, proxies etc.
In order to make use of this module you will have to import urllib in your python file as shown below:

```
1 | import urllib
```

'urllib' offers some very important packages that help us work with urls. *Whenever there is a need to open and read URLS we can import urllib.request as shown below:*

```
1 | import urllib.request
```

As the name suggests urllib.request allows you to request data from a web server(port 80 by default) while accessing a url with urllib.request you can provide the domain name or an ip address of the web page as a parameter, both cases will work fine. *This module defines function and classes that help developers to access the HTTP and HTTPS web pages:*

```
1 | req=urllib.request.urlopen('https://www.google.com')
```

The parameter provided in urlopen function can be a string or a request object. Along with this you can provide two more parameters. When your HTTP request is POST instead of GET you need to provide additional data. You also need to define a timeout parameter which is in seconds and is used for blocking operations such as connection attempts. When timeout parameter is not defined, the global timeout setting is used.

Now to read the information use the read() function as follows:

```
print(req.read())
```

So, this is how the final piece of code will look like:

```
import urllib.request
req=urllib.request.urlopen('https://www.google.com')
print(req.read())
```

If you now execute your code, the output file displays the source code of the web page. Sometimes, websites don't like other programs visiting their sites and accessing their data. For, these purposes in other programming languages developers often modify the user-agent which is a variable of the header sent in. However, in case of Python you generally don't have to face any such issues because by default Python notifies the website that your piece of code is making use of urllib and it also mentions the Python's version that is being used.

The piece of code mentioned above expresses the simplest way of using the urllib module to access the internet. We have just imported the urllib.request module, opened the url and assigned the value to a variable. We then invoked the read() command. The

output file looks messy but do not panic. As you advance in Python you will learn how to extract important information from such files. For this purpose we have to make use of regular expressions. As you know HTML, JavaScript and CSS are used for designing a web page which is why the content of the file looks confusing however meaningful data can be easily retrieved from this file.

Thank you for reading!

We invite you to share your thoughts and reactions

STONE RIVER
eLEARNING