Smarter education for smarter planet

IBM Innovation Centre for Education

Unit 4:Standard Libraries

Unit objectives

- After completing this unit, you should be able to:
- How to use Standard Libraries
- Data Compression and Archiving libraries
- Web Development, HTTP server
- Network programming, Sockets and other functions
- Client Server programming
- Extending and Embedding Python in C

Python Standard libraries

- The Python’s standard library is very large and huge and written mostly in C language
- Offers users many functionalities to integrate that in the Python apps
- Availability helps developers to pursue using Python for advanced applications
- To install packages, use the ‘pip install <package_name>’

Creating modules

- Modules are imported using the “import “ command at the beginning of the file
- Example:
- import os. # Imports the system OS module
- import printmodule # Imports the User defined print module

step1#The printmodule is going to be written as follows:

```
$ cat printmodule.py

def print_hello_world () :
        print ("hello world from print module”)
```

Step2#The calling python program:

```
$ cat testmodule.py
import printmodule.   # This imports the above module into this program.

printmodule.print_hello_world().  # Notice the prefix to the function calling the user defined module.
```

step3#The module being imported can be either written as :

```
 import <module name>
or
from <library name >import <function>
e.g:
from printmodule import print_hello_world
```

Creating modules Contd

- The library module can have functions, variables and other attributes, or even a Object Oriented Class written in it.

- Now, we extend the library printmodule to include a new Class and some variables along with the print_hello_world() function.

Here is the extended module with a Class and a function definition..

```
$ cat printmodule.py
def print_hello_world () :
        print ("hello world from demo module")


class Employee:
        def __init__(self, name,age):
                self.name = name
                self.age = age

        def getName(self):
                print (" In the getName(), returning " + self.name)
                return self.name

        def getAge (self):
                return self.age
```

Example calling program

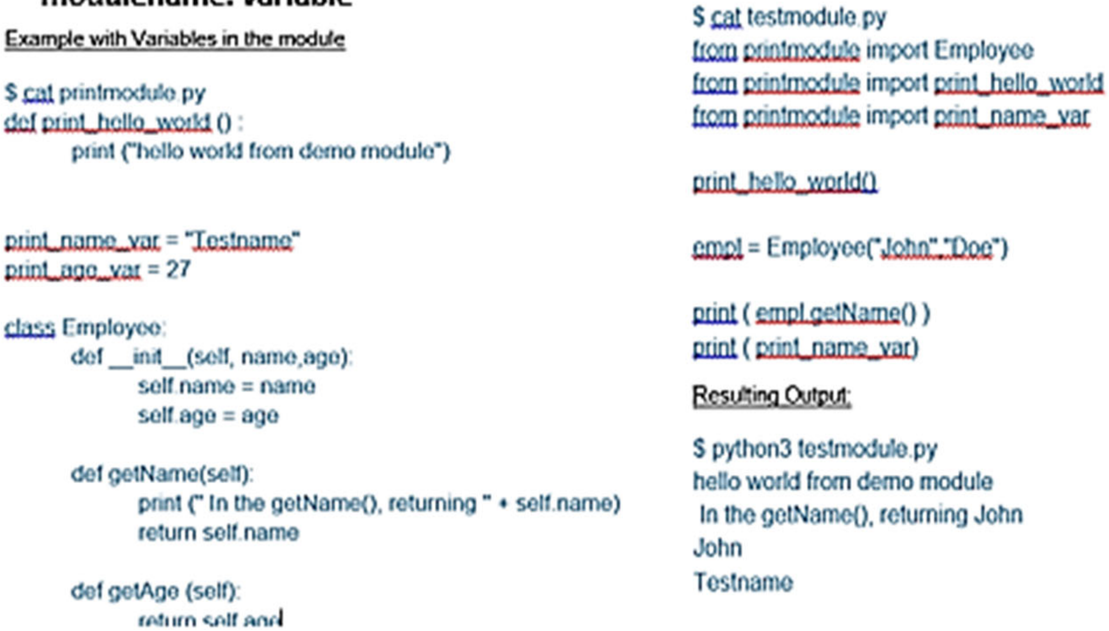
```
$ cat testmodule.py
import printmodule

printmodule.print_hello_world()

empl = printmodule.Employee("John","Doe")

print ( empl.getName() )
```

Resulting Output :

```
$ python3 testmodule.py
hello world from demo module
 In the getName(), returning John
John
```

Adding variables in the Module

- Variables can be added to the module, just like the Class and functions
- These variables can be referenced in the calling program, by referencing the **modulename. variable**



Example with Variables in the module

```
$ cat printmodule.py
def print_hello_world () :
        print ("hello world from demo module")


print_name_var = "Testname"
print_age_var = 27

class Employee:
        def __init__(self, name,age):
                self.name = name
                self.age = age

        def getName(self):
                print (" In the getName(), returning " + self.name)
                return self.name

        def getAge (self):
                return self.age
```

```
$ cat testmodule.py
from printmodule import Employee
from printmodule import print_hello_world
from printmodule import print_name_var

print_hello_world()

empl = Employee("John","Doe")

print ( empl.getName() )
print ( print_name_var)
```

Resulting Output:

```
$ python3 testmodule.py
hello world from demo module
 In the getName(), returning John
John
Testname
```

Order of search for the modules in the code

To search for the list of pre-installed standard library, a small program to print the sys.path is shown below.

Example program:

```
$ cat syspath.py
import sys
print(sys.path)
```

Resulting Output: (lists all the folders where the modules are going to be searched when it hits a import statement)

```
['/Users/user/nodetest',
'/Library/Frameworks/Python.framework/Versions/3.6/lib/python36.zip',
'/Library/Frameworks/Python.framework/Versions/3.6/lib/python3.6',
'/Library/Frameworks/Python.framework/Versions/3.6/lib/python3.6/lib-dynload',
'/Users/Library/Python/3.6/lib/python/site-packages',
'/Library/Frameworks/Python.framework/Versions/3.6/lib/python3.6/site-packages']
```

Standard Libraries (SYS Module)

- The sys Module has system related functions that offers users to communicate with the system attributes
- **sys.argv** -This tells the arguments passed to the program. argv[0] is the name of the program itself. argv[1] is the next argument, argv[2] (if any) is the following argument and so on.

Example of sys.argv

```
$ cat printpath.py
import sys
print ( len( sys.argv))

for item in sys.argv:
        print item


$ python3 printpath.py  one two three
4
printpath.py
one
two
three
```

sys.exit()

This is the most often used command by the Python command interpreter to get out of the Python command prompt and back to the OS console (or terminal)

Example :

```
#!/usr/bin/python3
import sys
print ("hello world")
sys.exit(254)
```

Resulting Output

```
$ python3 testsys.py
hello world


$ echo $?
254
```

Standard Libraries (SYS Module) Contd

IBM ICE (Innovation Centre for Education)

sys.float_info

- This is a structure and contains the internal max and min value representation of the float data type

```
$cat getfloatinfo.py
import sys
print ( sys.float_info)


$ python getfloatinfo.py
sys.float_info(max=1.7976931348623157e+308,
max_exp=1024,
max_10_exp=308,
min=2.2250738585072014e-308,
min_exp=-1021,
min_10_exp=-307,
dig=15,
mant_dig=53,
epsilon=2.220446049250313e-16,
radix=2,
rounds=1)
```

sys.getsizeof ()

This function returns the size of the Object in bytes. helpful to determine how much size the object is occupying in the OS. Usually a block of bytes is allocated for a Objects.

Example code:

```
$ cat getsizeof.py
import sys
class Employee :
        def __init__ ( self, name,age ):
                self.name = name
                self.age = age

        def  printName (self):
                print ("Inside the print Name " + self.name )
                return self.name

x = Employee ( "John", "Doe")
print ( x.printName () )

print( "Size of Employee Object in bytes " , sys.getsizeof(Employee) )
```

Resulting Output:

```
$ python3 getsizeof.py
Inside the print Name John
John
Size of Employee Object in bytes  1056
```

Standard Libraries (SYS Module) Contd

IBM ICE (Innovation Centre for Education)

sys.modules ()

This function lists the loaded modules names to modules when the python interpreter is bein

```
$ cat sysmodules.py
import sys
print (sys.modules)
```

Resulting Output : (only a snapshot is provided) This can vary platform to platform.

```
're': <module 're' from
'/Library/Frameworks/Python.framework/Versions/3.6/lib/python3.6/re.py'>, 'enum':
<module 'enum' from
                    ...
```

sys.platform

This returns the platform that this Python interpreter is being run. Helpful when determining the OS so that OS specific attributes can be aligned to the program.

Example Program

```
$ cat platform.py
import sys
print (sys.platform)
```

Resulting Output: (this is on a Mac) Your answer may vary.

```
$ python3 platform.py
darwin
```

Data Compression and Archiving

- This module is available for python developers to aid in creating zip files using python libraries.
- Popular compression techniques are zipfile, tarfile, gzip, and bz2.

ZIP File Module : Tools to view, read, write, append, and check contents of the zip file

Example : Here the archive.zip exists. Returns True.

```
import zipfile
print (zipfile.is_zipfile("archive.zip"))
```

zipfile.BadZipFile

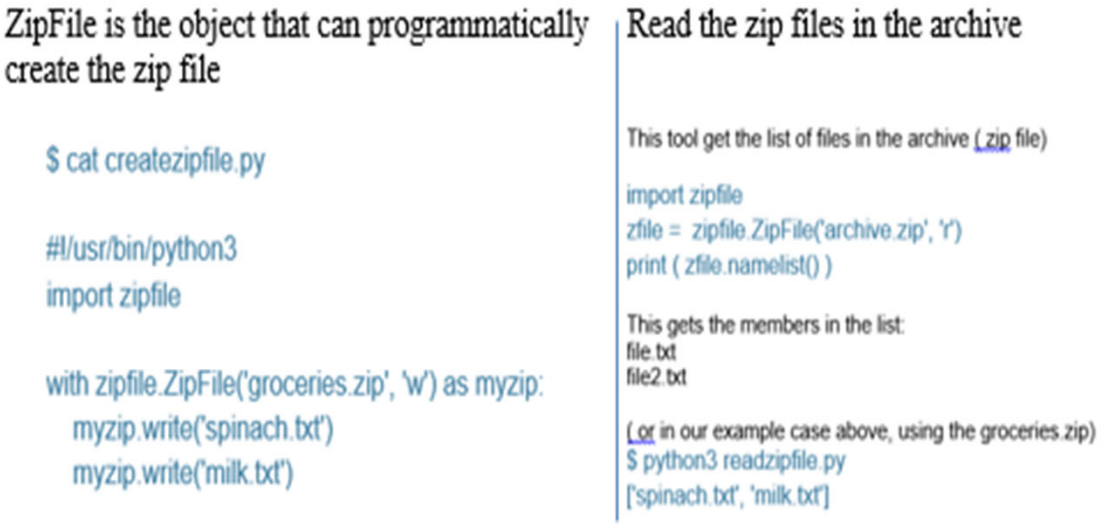
This is an exception that is raised when the file is corrupt or not able to read.

Example code :

```
#!/usr/bin/python3
import zipfile
try :
        print (zipfile.is_zipfile("achive.zip"))

except BadZipFile:  # Can be BadZipfile in earlier versions of Python.
```

Creating zip files using the ZipFile Object

IBM ICE (Innovation Centre for Education)



ZipFile is the object that can programmatically create the zip file

```
$ cat createzipfile.py

#!/usr/bin/python3
import zipfile

with zipfile.ZipFile('groceries.zip', 'w') as myzip:
   myzip.write('spinach.txt')
   myzip.write('milk.txt')
```

Read the zip files in the archive

This tool get the list of files in the archive (.zip file)

```
import zipfile
zfile =  zipfile.ZipFile('archive.zip', 'r')
print ( zfile.namelist() )
```

This gets the members in the list:

```
file.txt
file2.txt
```

(.or in our example case above, using the groceries.zip)

```
$ python3 readzipfile.py
['spinach.txt', 'milk.txt']
```

Tar file tools

- Tar file tools offers ability to read, write/create the tar files programmatically using the Python
- **Is_tarfile (name)** -This function returns a True or a False, whether the name/path is a tar file or not !

Example :

```
$ cat istarfile.py

#!/usr/bin/python3
import tarfile
print ( tarfile.is_tarfile( "archive.tar"))
```

Resulting Output:

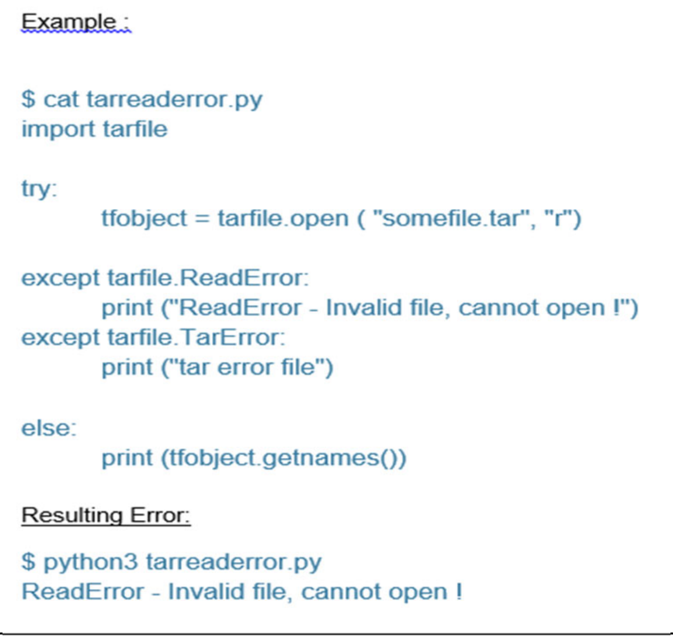
```
$ python3 istarfile.py
True
```

getnames()

```
$cat readtarfile.py
import tarfile

print ( tarfile.is_tarfile( "archive.tar"))

try :
______tfobject = tarfile.open ( "archive.tar", "r")
exception TarError:
        print "Error reading"
else :
     print (tfobject.getnames())


$ python3 readtarfile.py

True
['one.txt', 'two.txt', 'three.txt']
```

Tar file Exceptions

- exception tarfile.TarError -This is the base class of all tar file operations. This can be the last resort to catch any exception that happens during the tar file read/write/extract operation.
- exception tarfile.ReadError-This exception can happen if the file is a invalid or a corrupted and cannot be read



Example :

```
$ cat tarreaderror.py
import tarfile

try:
        tfobject = tarfile.open ( "somefile.tar", "r")

except tarfile.ReadError:
        print ("ReadError - Invalid file, cannot open !")
except tarfile.TarError:
        print ("tar error file")

else:
        print (tfobject.getnames())
```

Resulting Error:

```
$ python3 tarreaderror.py
ReadError - Invalid file, cannot open !
```

Gzip

This Python Module is a tool is used to compress and uncompress files in gzip, and gunzip format.

The format to open and close is similar to the tools listed above.
The open call returns a GZIP class that has methods to extract and compress data.

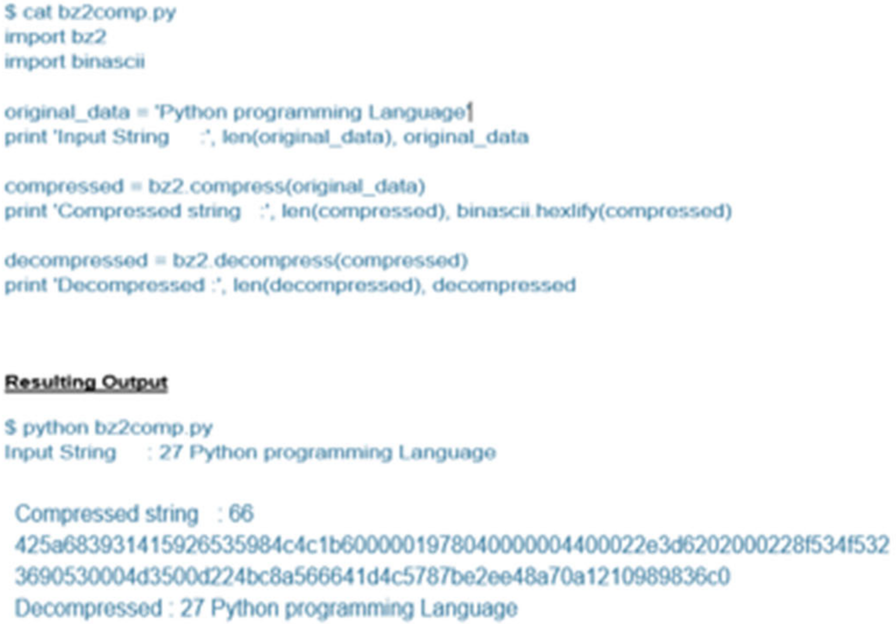
```
gzip.open(filename, mode='rb', compresslevel=9, encoding=None, errors=None, newline=None)¶
```

Example program to read a gzipped file and read the contents.

```
$ cat gzipp.py

import gzip
with gzip.open('theschool.txt.gz', 'rb') as f:
    file_content = f.readlines()
    print ( file_content)
```

Bzip2

- Bzip2 tool is for compression, data transmission and for storage



```
$ cat bz2comp.py
import bz2
import binascii

original_data = 'Python programming Language'
print 'Input String      :', len(original_data), original_data

compressed = bz2.compress(original_data)
print 'Compressed string   :', len(compressed), binascii.hexlify(compressed)

decompressed = bz2.decompress(compressed)
print 'Decompressed :', len(decompressed), decompressed
```

Resulting Output

```
$ python bz2comp.py
Input String      : 27 Python programming Language

 Compressed string   : 66
 425a68393141592653598 4c4c1b600000019780400000004400022e3d6202000228f534f532
 3690530004d3500d224bc8a566641d4c5787be2ee48a70a1210989836c0
 Decompressed : 27 Python programming Language
```

Python Web Development

- There are many modules that support web development
- Url ,lib.requests, http, and “requests” are different mechanisms to access the web.
- “requests” module that offers more features, and uses urllib3 a library that is powerful http client for Python
- Install requests, and import that in the Python code
- pip install requests

Request module :Access web page

```
import requests

#This call to make a 'get' request is as follows, It needs the destination location URL.

r = requests.get('https://finance.yahoo.com/quote/INFY?p=INFY')

print (r.url)
print (r.text)
```

- The resulting response is in r.text (which is the entire blob of response)
- Interestingly all the data is going to be decoded automatically.

Example of a code that cannot connect to the remote

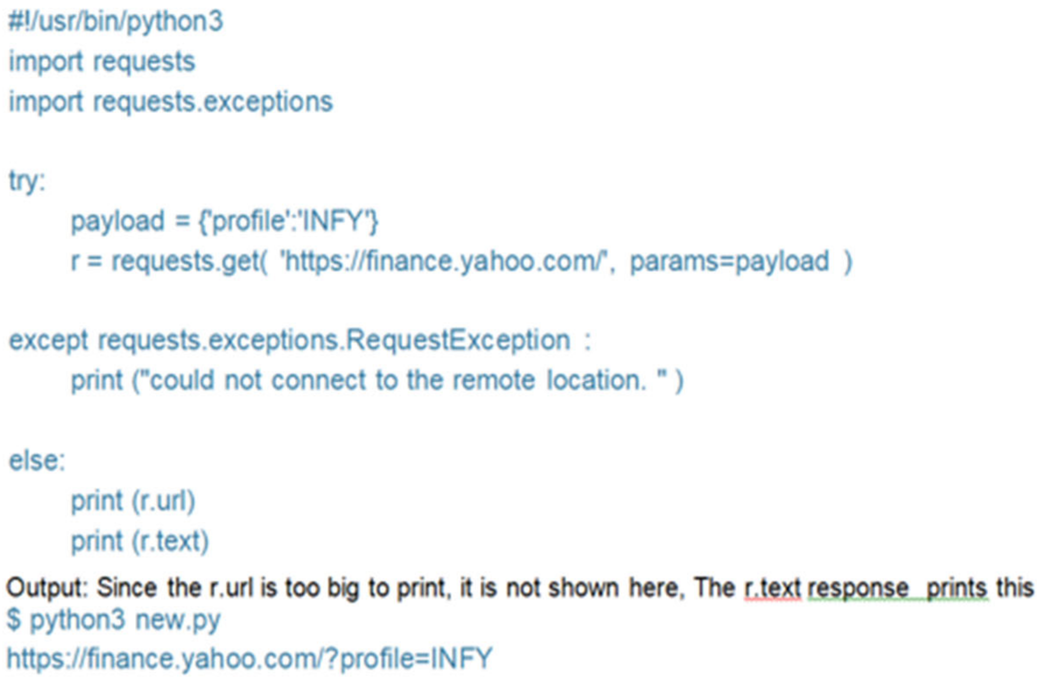
```
import requests
import requests.exceptions

try:
        r = requests.get('https://financ.yahoo.com/quote/INFY?p=INFY')

except requests.exceptions.RequestException :
        print ("could not connect to the remote location. " )

else:
        print (r.url)
        print (r.text)
```

- Timeouts can be specified by the timeout argument in the call.
- requests.get('http://', timeout=0.001)

- Params can be passed to the requests as payload as shown below.

- payload = **{**'key1'**:** 'value1'**,** 'key2'**:** 'value2'**}**
- r = requests.get**(**'<URL>'**,** params=payload**)**

Request module :passing the payload



```
#!/usr/bin/python3
import requests
import requests.exceptions

try:
        payload = {'profile':'INFY'}
        r = requests.get( 'https://finance.yahoo.com/',  params=payload  )

except requests.exceptions.RequestException  :
        print ("could not connect to the remote location. " )

else:
        print (r.url)
        print (r.text)
```

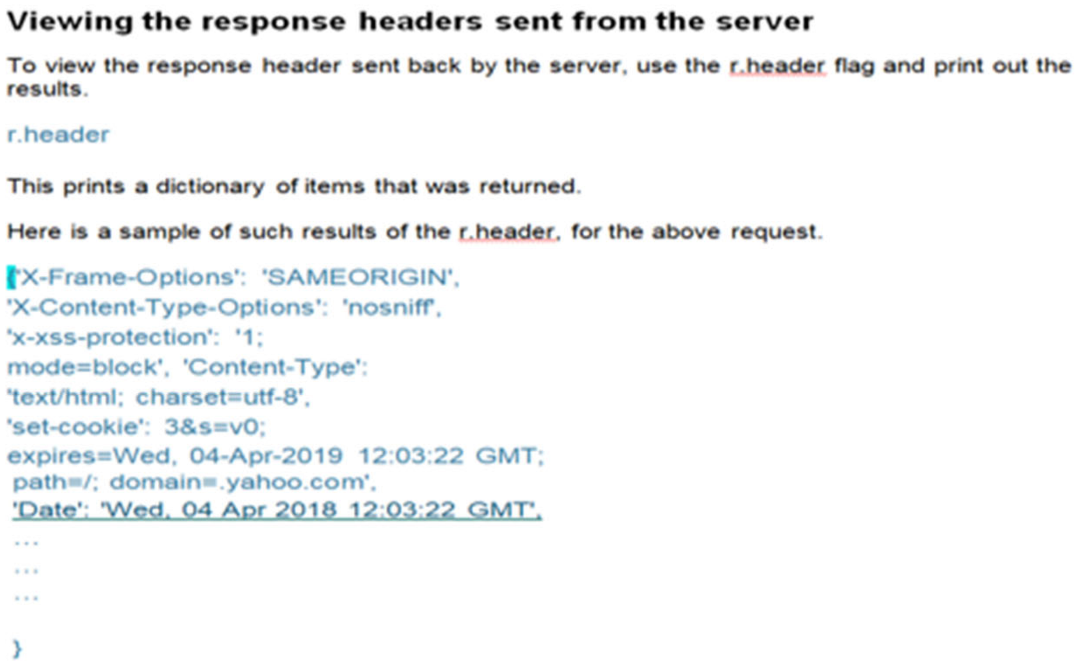
Output: Since the r.url is too big to print, it is not shown here, The r.text response prints this

```
$ python3 new.py
https://finance.yahoo.com/?profile=INFY
```

Request module: To view the status code

```
import requests
import requests.exceptions

try:
        payload = {'quote':'INFY'}
        r = requests.get( 'https://finance.yahoo.com/', params=payload )

except requests.exceptions.RequestException :
        print ("could not connect to the remote location. " )

else:
        print (r.url)
        print (r.status_code)
```

Response Module



Viewing the response headers sent from the server

To view the response header sent back by the server, use the r.header flag and print out the results.

```
r.header
```

This prints a dictionary of items that was returned.

Here is a sample of such results of the r.header, for the above request.

```
{'X-Frame-Options': 'SAMEORIGIN',
'X-Content-Type-Options': 'nosniff',
'x-xss-protection': '1;
mode=block', 'Content-Type':
'text/html; charset=utf-8',
'set-cookie': 3&s=v0;
expires=Wed, 04-Apr-2019 12:03:22 GMT;
 path=/; domain=.yahoo.com',
 'Date': 'Wed, 04 Apr 2018 12:03:22 GMT',
 ...
 ...
 ...

 }
```

http Module: Overview

- http module has multiple modules within it for working with the HTTP protocol.
- http.client :User should call this instance when making a connection to the http server
- http.server :provides a http web server for development
- http.cookies and cookies jar are the major modules in this package.

http Module : http client

- This module has the class for the client side of things

Step 1 :Creating the Instance

- class http.client.HTTPConnection(host, port=None, [timeout,]source_address=None)

Step2 : Response

- class http.client.HTTPResponse(sock, debuglevel=0, method=None, url=None)

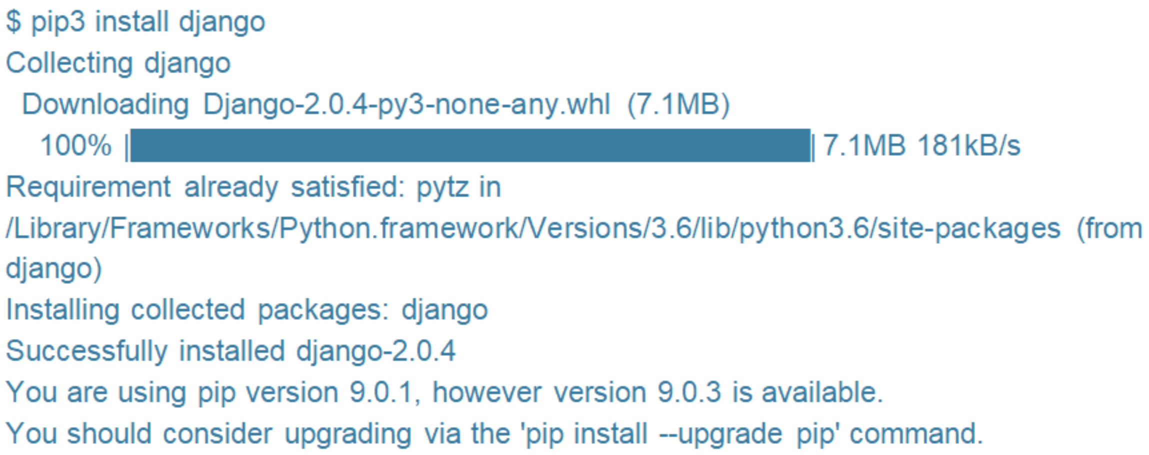
Step3:3.HTTP Exceptions :

- exception http.client.HTTPException

Django - Web Development framework

- Django is an open source tool
- Django requires less coding, and uses Object Oriented methodologies in the backend
- Django allows localisation so that users can develop applications in several languages
- It has a built in web server and a web interface

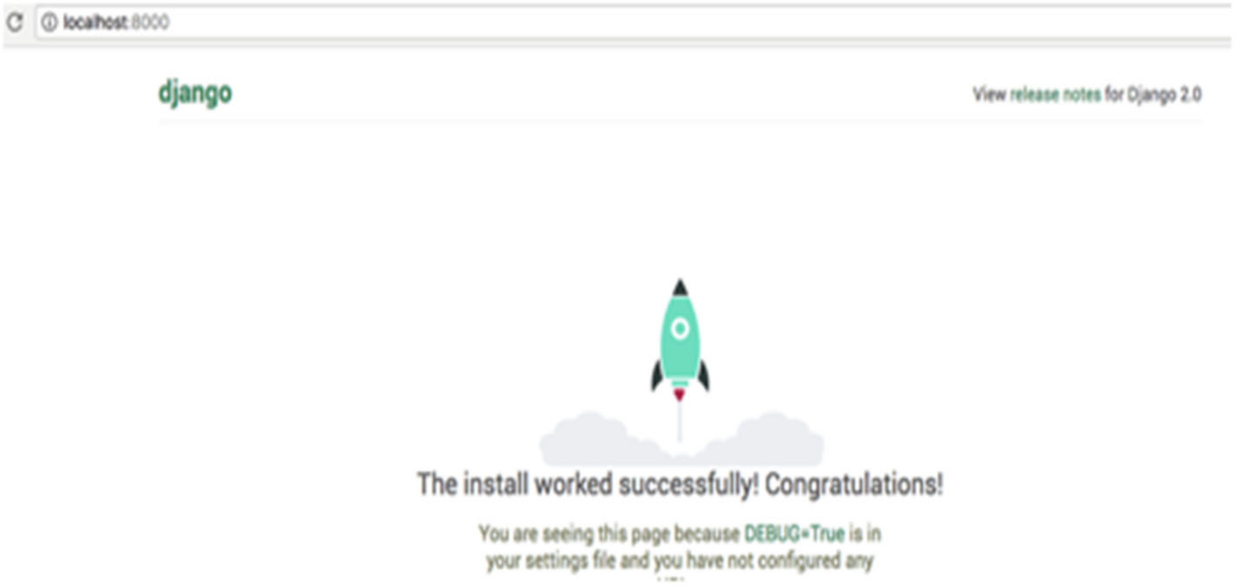
Django - Installation



```
$ pip3 install django
Collecting django
  Downloading Django-2.0.4-py3-none-any.whl (7.1MB)
    100% |████████████████████████████████| 7.1MB 181kB/s
Requirement already satisfied: pytz in
/Library/Frameworks/Python.framework/Versions/3.6/lib/python3.6/site-packages (from
django)
Installing collected packages: django
Successfully installed django-2.0.4
You are using pip version 9.0.1, however version 9.0.3 is available.
You should consider upgrading via the 'pip install --upgrade pip' command.
```

Django Interface

Django interface – After successful installation



localhost:8000

django

View release notes for Django 2.0

The install worked successfully! Congratulations!

You are seeing this page because DEBUG=True is in your settings file and you have not configured any

Django -Creating an app

Run this command below: Ensure that you you are in the same folder as the manage.py file is in.

```
python3 manage.py startapp polls
```

A new folder "polls" gets created

```
$ ls -R polls
__init__.py    apps.py            models.py    views.py
admin.py       migrations    tests.py

polls/migrations:
total 0
-rw-r--r--   1 user  staff  0 Apr  5 15:57 __init__.py
```

The folder structure has files for models, views .

Create a view for the app.

```
cd polls
vi views.py.
```

Add these few lines.

```
from django.http import HttpResponse

def index(request):
   return HttpResponse("Hello, world. This message is from views.py code")
```

In the polls/urls.py

Add these few lines.

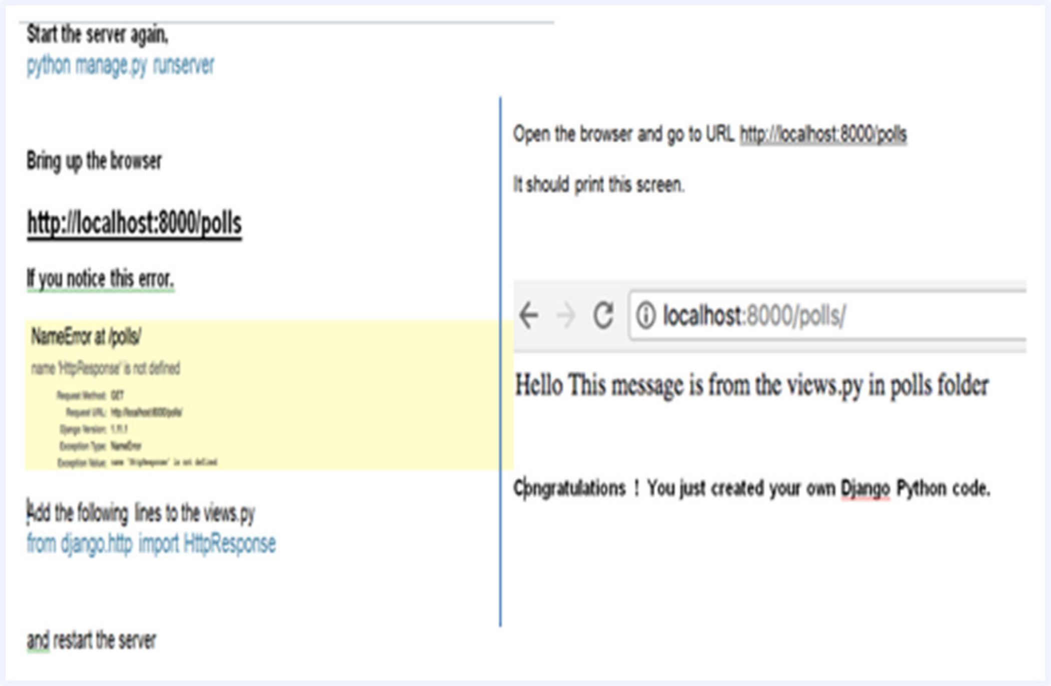
```
from django.urls import path

from . import views

urlpatterns  = [
    path('', views.index, name='index'),
]
```

In the mysite/urls.py

Add these few lines.

```
from django.contrib import admin
from django.urls import include, path

urlpatterns  = [
    path('polls/', include('polls.urls')),
    path('admin/', admin.site.urls),
]
```

Django -Creating an app Contd



Start the server again.

```
python manage.py runserver
```

Bring up the browser

http://localhost:8000/polls

If you notice this error.

```
NameError at /polls/
name 'HttpResponse' is not defined

      Request Method:  GET
         Request URL:  http://localhost:8000/polls/
      Django Version:  1.11.1
      Exception Type:  NameError
     Exception Value:  name 'HttpResponse' is not defined
```

Add the following lines to the views.py

```
from django.http import HttpResponse
```

and restart the server

Open the browser and go to URL http://localhost:8000/polls

It should print this screen.

```
localhost:8000/polls/

Hello This message is from the views.py in polls folder
```

Cpngratulations ! You just created your own Django Python code.

Flask : Web Development Framework

- Flask is yet another popular framework for Python amongst others
- This is for smaller applications, lesser requirements and faster developments cycles
- If the application to be developed does not need a database, admin page, and User management, then Flask is a better choice than Django.
- For a Quickstart formal documentation, go to http://flask.pocoo.org/docs/0.12/quickstart/

Flask Contd

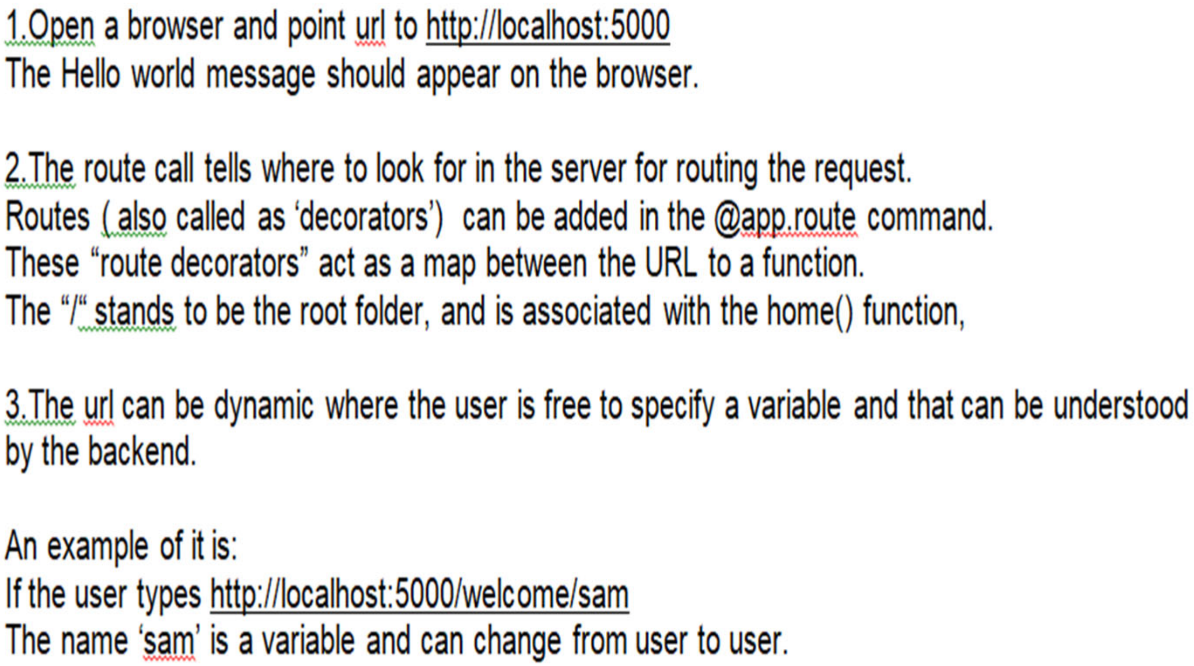
Prerequisites :
It is assumed that the user is familiar with MVC, frameworks in general, writing basic html templates.

Flask uses a template engine called “Jinga2’ for dynamic web page development.

To get started and see how this works, create a folder, cd to that folder and create a file with this content.

```
from flask import Flask
app = Flask(__name__)

@app.route('/')
def hello_world():
  return 'Hello World'

if __name__ == '__main__':
  app.run()
```

Flask Contd



1.Open a browser and point url to http://localhost:5000
The Hello world message should appear on the browser.

2.The route call tells where to look for in the server for routing the request.
Routes (also called as 'decorators') can be added in the @app.route command.
These "route decorators" act as a map between the URL to a function.
The "/" stands to be the root folder, and is associated with the home() function,

3.The url can be dynamic where the user is free to specify a variable and that can be understood by the backend.

An example of it is:
If the user types http://localhost:5000/welcome/sam
The name 'sam' is a variable and can change from user to user.

Flask Code

This can be addressed in the Flask code as

```
@app.route('/sayhello/<name>')
def sayhello_name(name):
   return ‘Hello, My name is %s!' % name
```

If the user types in ‘sam’ in the URL as described above then the “Hello Sam” is returned back to the browser.

Likewise, if the user wants to types in http://localhost:5000/sam
then the appropriate code is to define a route with “/<name>”

The <name> is a string variable.
Other data types like int, float and are accepted as well.

The requirement is that it has to be preceded by a datatype indicating the type.

```
@app.route(‘/stock/<int:Volume>')
@app.route(‘/open/<float:Open>')
```

Flask Code Contd

Examples of int variable in the route

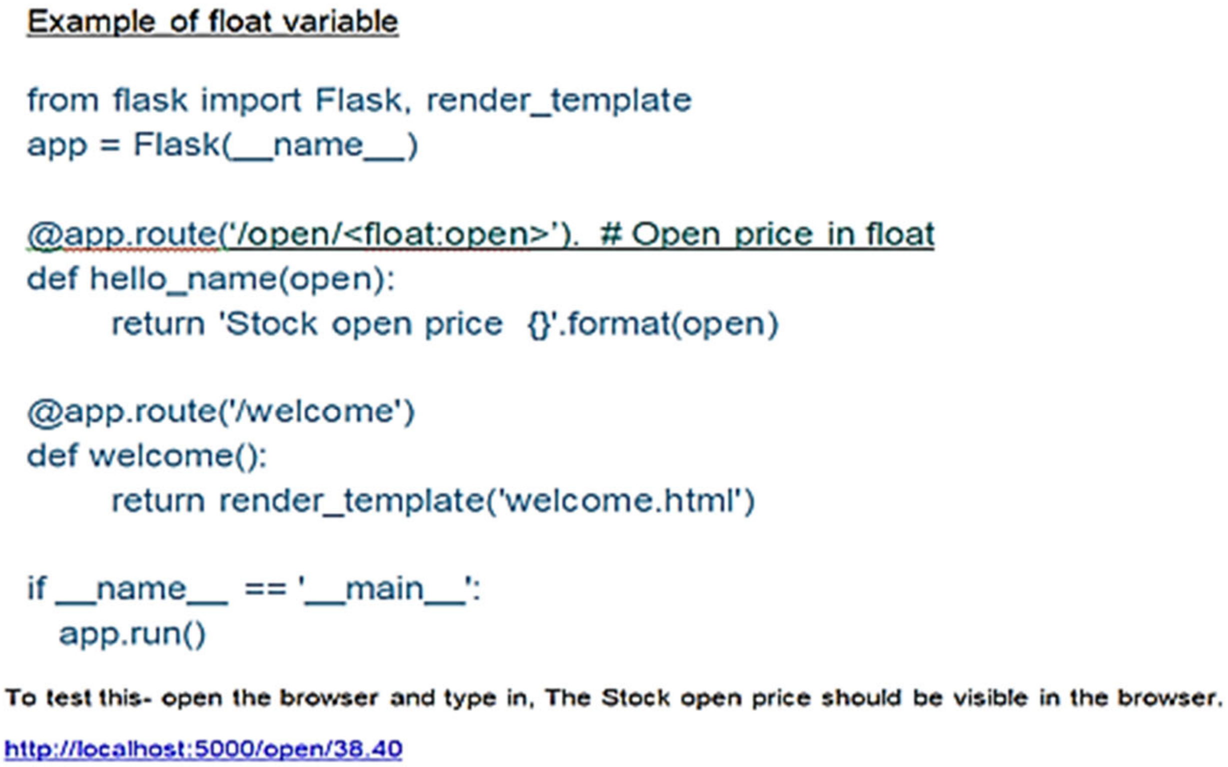
```
from flask import Flask, render_template
app = Flask(__name__)

@app.route('/stock/<int:volume>').  # Note the int variable.
def hello_name(volume):
        return 'Volume of Stock {}'.format(volume)

@app.route('/welcome')
def welcome():
        return render_template('welcome.html')

if __name__  == '__main__':
   app.run()
```

Run the app

Spython3 flaskapp.py

To test this, open the browser and type in.
http://localhost:5000/stock/500

Flask Code Contd



Example of float variable

```
from flask import Flask, render_template
app = Flask(__name__)

@app.route('/open/<float:open>').  # Open price in float
def hello_name(open):
        return 'Stock open price  {}'.format(open)

@app.route('/welcome')
def welcome():
        return render_template('welcome.html')

if __name__  == '__main__':
   app.run()
```

To test this- open the browser and type in, The Stock open price should be visible in the browser.

http://localhost:5000/open/38.40

Flask -Templates

Python Flask has an option of calling the templates which are html files to be rendered during call-ins in the route () decorator functions.

These html files are stored in the templates folder which has to be created in the current folder where the Flask server is being run.

A simple html file could be having very little information, or having an extensive html code. Flask should be able to call in this template file and process.

The format is

```
@app.route('/welcome')
def welcome():
        return render_template('welcome.html')
```

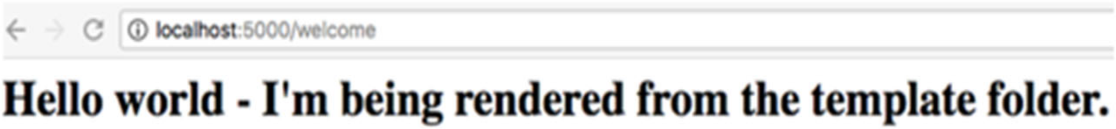
The render_template() function looks for the welcome.html in the templates folder.

Flask –Templates Contd

Create a template folder, and in that create the welcome.html and having a few lines

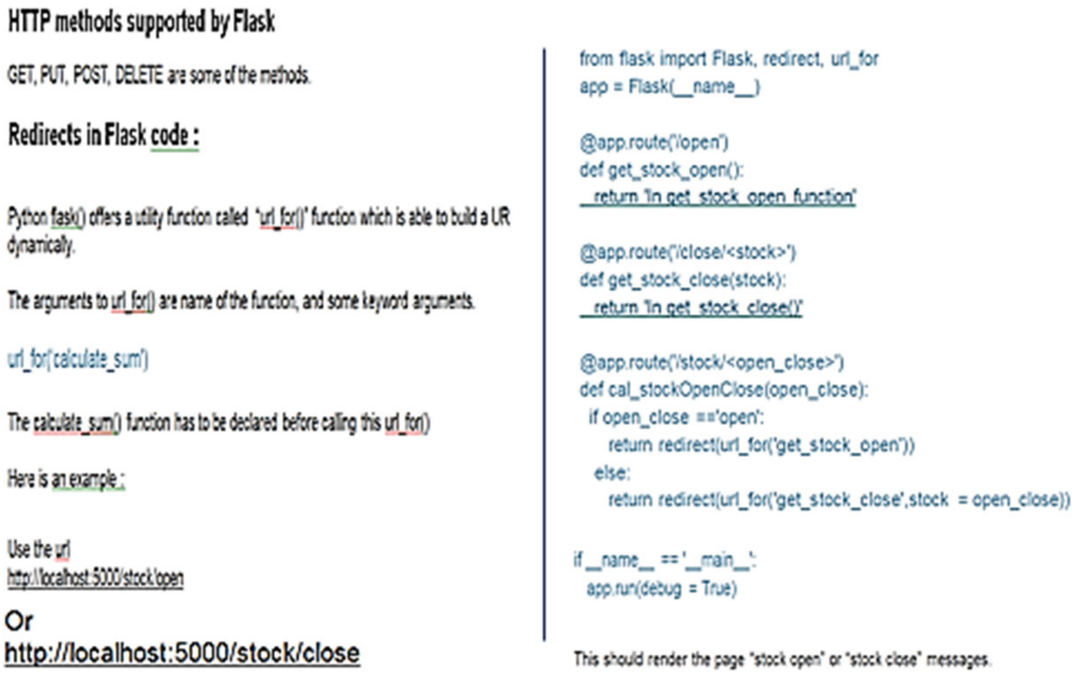
```
<html>
<h1>
Hello world - I'm being rendered from the template folder
</h1>
</html>
```

Launch the flask python app server.
and open the browser, and go to http://localhost:5000/welcome


localhost:5000/welcome

Hello world - I'm being rendered from the template folder.

Flask –Templates contd



HTTP methods supported by Flask

GET, PUT, POST, DELETE are some of the methods.

Redirects in Flask code :

Python fask() offers a utility function called 'url_for()' function which is able to build a UR dynamically.

The arguments to url_for() are name of the function, and some keyword arguments.

url_for('calculate_sum')

The calculate_sum() function has to be declared before calling this url_for()

Here is an example :

Use the url
http://localhost:5000/stock/open

Or
http://localhost:5000/stock/close

```
from flask import Flask, redirect, url_for
app = Flask(__name__)

@app.route('/open')
def get_stock_open():
  return 'In get_stock_open_function'

@app.route('/close/<stock>')
def get_stock_close(stock):
  return 'In get_stock_close()'

@app.route('/stock/<open_close>')
def cal_stockOpenClose(open_close):
  if open_close =='open':
     return redirect(url_for('get_stock_open'))
   else:
     return redirect(url_for('get_stock_close',stock  = open_close))

if __name__ == '__main__':
  app.run(debug = True)
```

This should render the page "stock open" or "stock close" messages.

Flask -Template Engine : Jinga

- Templates are just like text files with html code and has ability to manipulate data using programming constructs like for, while etc.

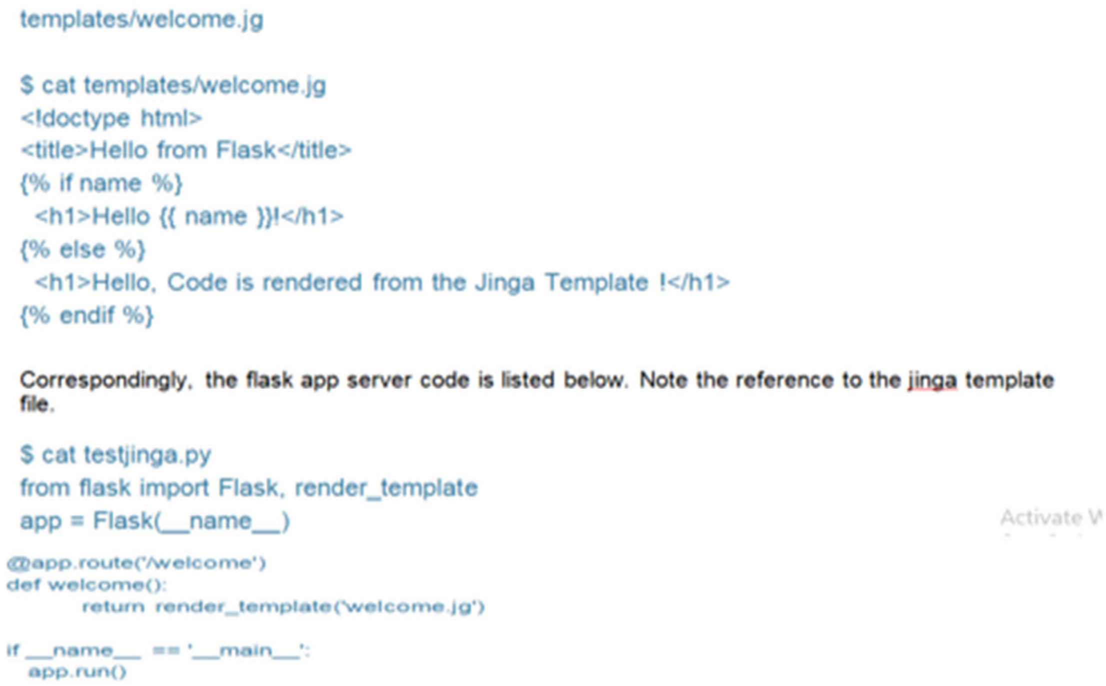
- Template engines read this input file and create html pages to be rendered on web pages.

- The output will be consistent, and code is reusable multiple times and efficient.

- Python Flask uses a certain template engine called Jinga, which reads the flask code, and processes the template specified in the code.

Flask – Jinga Template creation

IBM ICE (Innovation Centre for Education)



templates/welcome.jg

```
$ cat templates/welcome.jg
<!doctype html>
<title>Hello from Flask</title>
{% if name %}
  <h1>Hello {{ name }}!</h1>
{% else %}
  <h1>Hello, Code is rendered from the Jinga Template !</h1>
{% endif %}
```

Correspondingly, the flask app server code is listed below. Note the reference to the jinga template file.

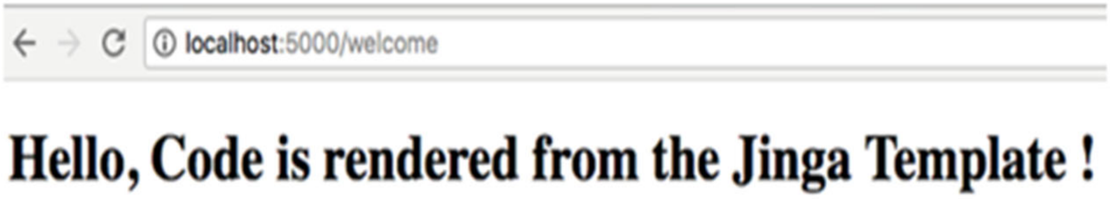
```
$ cat testjinga.py
from flask import Flask, render_template
app = Flask(__name__)

@app.route('/welcome')
def welcome():
        return render_template('welcome.jg')

if __name__ == '__main__':
   app.run()
```

Flask – Jinga Template

Start the python app server.

```
python3 testing.py
```

Open the browser and watch the rendering from the template file.
http://localhost:5000/welcome


localhost:5000/welcome

Hello, Code is rendered from the Jinga Template !

Congratulations, You created your first flask framework web page with a template Engine

Network programming

- Python offers libraries to do socket programming
- Lower level :user can implement the basic socket level where the sockets are created and data is sent and received
- Higher level: use the existing libraries or modules like HTTP, FTP
- Server listens to incoming connections from the client, and the client at the other end sends requests and waits for responses

Network programming Contd

To write apps, user would import the socket module in their python code.

```
import socket
```

The socket class

```
s = socket.socket (socket_family, socket_type, protocol=0)
```

The family can be AF_INET,(for IPv4) or AF_INET6 (for IPv6) or AF_UNIX.
The socket type can be SOCK_STREAM or SOCK_DGRAM (SOCK_STREAM) is the most familiar type often used for TCP connections.

The protocol is a default option and is usually not mentioned.

Next, connect the socket to the remote site, it can be a web site, or a remote client.

To communicate use the send() and recv() functions.

Finally to end - use the close() function.

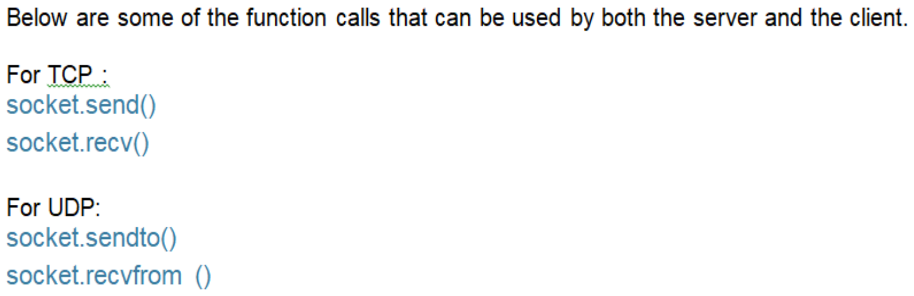
Network programming Contd

Network server setup:

- A) Create the socket
- B) bind the socket to a hostname or a port using the bind() function , and then set up to listen using the listen() call

Network client:

- A)create the socket and connect to the server.
- B)Now, use the send() , recv() to send and receive communications



Below are some of the function calls that can be used by both the server and the client.

For TCP :
socket.send()
socket.recv()

For UDP:
socket.sendto()
socket.recvfrom ()

Echo Server and a Client using sockets in Python

echoserver.py

```
$ cat echoserver.py
# echo_server.py
import socket


host = ''       # Symbolic name meaning all available interfaces
port = 12345    # Arbitrary non-privileged port
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind((host, port))
s.listen(1)
conn, addr = s.accept()
print('Connected by', addr)
while True:
   data = conn.recv(1024)
   if not data: break
   conn.sendall(data)
```

echo_client.py

Open a new terminal and execute this client code.

```
# echo_client.py
import socket


host = socket.gethostname()
port = 12345            # The same port as used by the server
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((host, port))
s.sendall(b'Hello, world')
data = s.recv(1024)
s.close()
print('Received', repr(data))
```

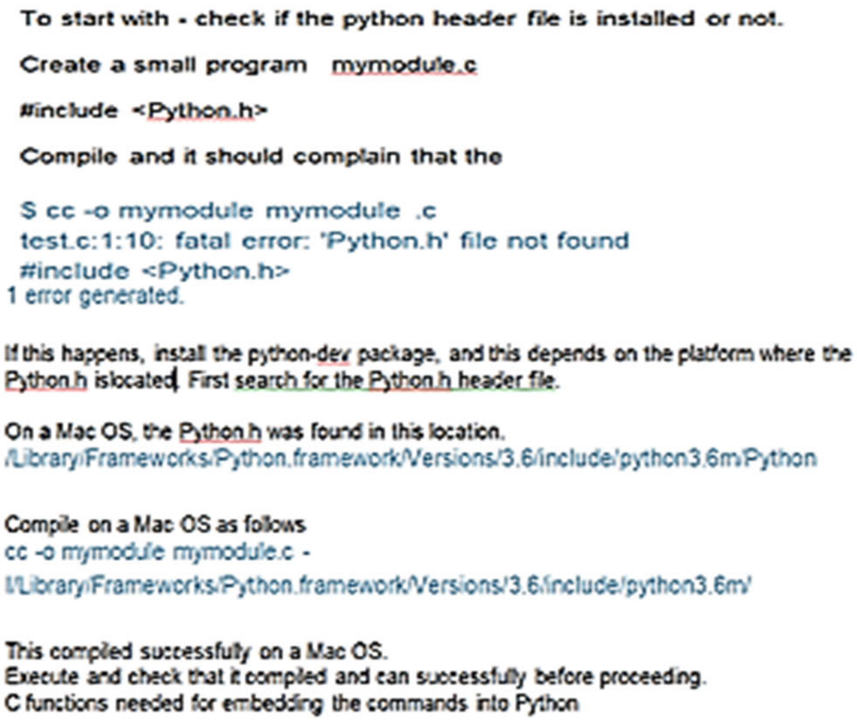
```
$python3 echo_client.py
```

The message 'Hello, world' is received back from the server, each time the program is run.

Extending and Embedding Python using C

- C functions that used often can be embed into the Python Programming language
- Python interpreter with the C functions or modules would be a good strategy.
- Possible to embed the python interpreter into another app
- Cpython is the reference implementation of Python
- Reference:https://docs.python.org/3/extending/index.html

Example Program



To start with - check if the python header file is installed or not.

Create a small program mymodule.c

#include <Python.h>

Compile and it should complain that the

```
$ cc -o mymodule mymodule .c
test.c:1:10: fatal error: 'Python.h' file not found
#include <Python.h>
1 error generated.
```

If this happens, install the python-dev package, and this depends on the platform where the Python.h islocated. First search for the Python.h header file.

On a Mac OS, the Python.h was found in this location.
/Library/Frameworks/Python.framework/Versions/3.6/include/python3.6m/Python

Compile on a Mac OS as follows

```
cc -o mymodule mymodule.c -
I/Library/Frameworks/Python.framework/Versions/3.6/include/python3.6m/
```

This compiled successfully on a Mac OS.
Execute and check that it compiled and can successfully before proceeding.
C functions needed for embedding the commands into Python

Custom Function

This is the main function that one has to develop to create the function.

```
static PyObject *my_function( PyObject *self, PyObject *args );
```

- User would write the contents of the custom function according to the requirements.
- This function call should return a pointer to a Python Object.
- When done writing the function, return with a value, or if there is no return value then return a constant called Py_RETURN_NONE.

Mapping Functions

- This custom function should be mapped to a table which is an array of structures of type PyMethodDef. (Array of structures is the "[ ]" in C programming language)
- The structure consists of some elements as shown

```
struct PyMethodDef {
 __char *ml_name;
   PyCFunction ml_meth;
   int ml_flags;
 __char *ml_doc;
};
```

The **ml_name** is function name that the user would be calling in the Python Interpreter after importing the custom module.

ml_meth is the address of the function. i.e the function name in the custom C Code that you are writing.

ml_flags is a flag, and it indicates which format the ml_meth is using. It can be METH_VARARGS, or METH_KEYWORDS, or METH_NOARGS indicating that the function will not take any arguments.

ml_doc : is the document string for the function.

Initialization Function

- The Python interpreter calls this function when the module is loaded. If the function name is helloworld, then the initialization function is called inithelloworld.

- The internal function is

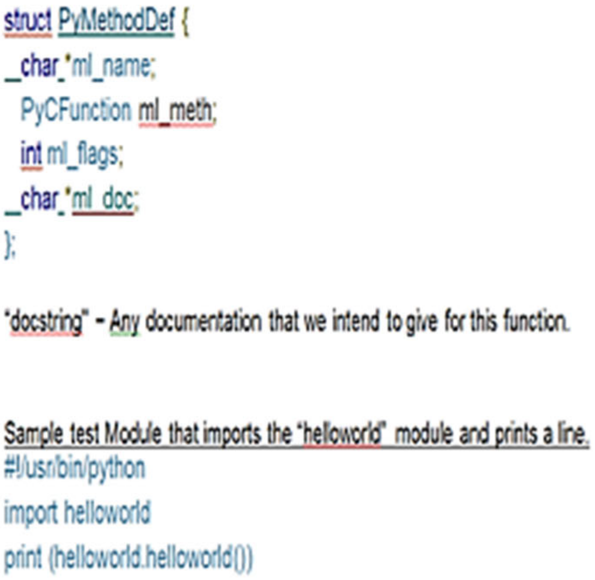
  Py_InitModule3(func, module_methods, “docstring...");

- The arguments are :

“func" − Function name to be exported to the Python interpreter.

“module_methods” − The array of PyMethodDef Structure. This is the mapping table that defines the function names, arguments, i.e the array of the structure

Sample Program

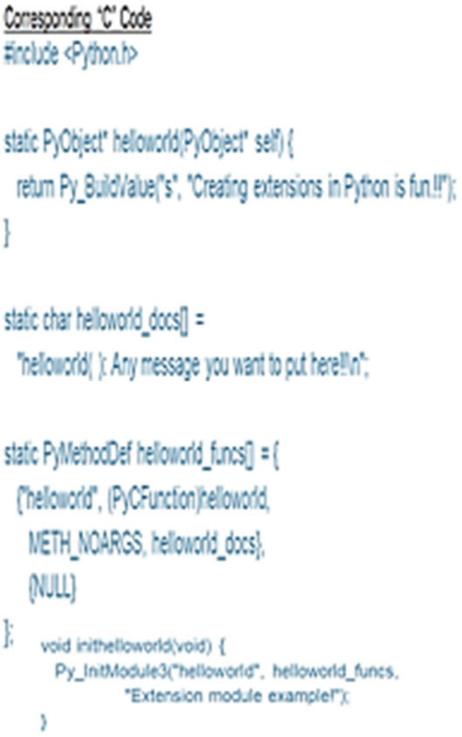


```
struct PyMethodDef {
char *ml_name;
  PyCFunction ml_meth;
  int ml_flags;
char *ml_doc;
};
```

"docstring" – Any documentation that we intend to give for this function.

Sample test Module that imports the "helloworld" module and prints a line.

```
#!/usr/bin/python
import helloworld
print (helloworld.helloworld())
```

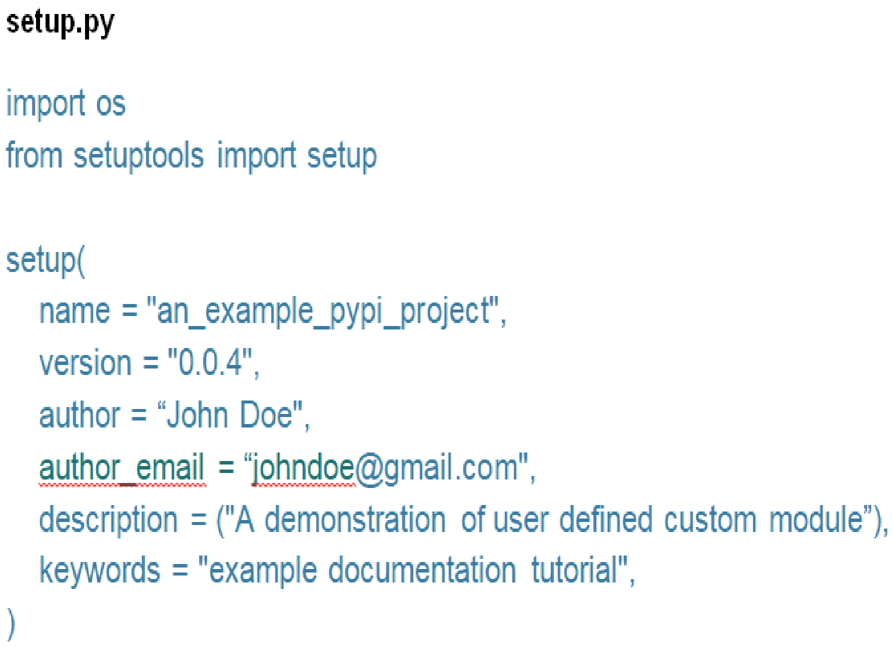


Corresponding "C" Code

```
#include <Python.h>

static PyObject* helloworld(PyObject* self) {
  return Py_BuildValue("s", "Creating extensions in Python is fun!!");
}

static char helloworld_docs[] =
  "helloworld( ): Any message you want to put here!!\n";

static PyMethodDef helloworld_funcs[] = {
  {"helloworld", (PyCFunction)helloworld,
   METH_NOARGS, helloworld_docs},
   {NULL}
};  void inithelloworld(void) {
      Py_InitModule3("helloworld", helloworld_funcs,
                     "Extension module example!");
    }
```

Building and Installing Extensions

- Modules are built and installed by a standard Python script called setup.py



setup.py

```
import os
from setuptools import setup

setup(
    name = "an_example_pypi_project",
    version = "0.0.4",
    author = “John Doe",
    author_email = “johndoe@gmail.com",
    description = ("A demonstration of user defined custom module”),
    keywords = "example documentation tutorial",
)
```

Building and Installing Extensions Contd

- In order to run this and install it in standard Python installation location, user would need super user permissions.

- In a Mac OS where the standard Python (v2.7) libraries are installed, this script builds the modules and installs it in /Library/Python/2.7/site-packages/

- This location may depend on platform to platform and installation to installation.

- More information on this can be found at :

https://pythonhosted.org/an_example_pypi_project/setuptools.html

- Building the custom module is as follows:
- 1]cd to the folder where the C module is located
- 2]Create a small script called setup.py with just a few lines as shown above.
- 3] run the command. A snapshot of build and install is shown below.

Example Module

If the user would like to install as super user, then sudo python setup.py install.

```
$ sudo python setup.py install
Password:
running install
running build
running build_ext
running install_lib
running install_egg_info
Removing /Library/Python/2.7/site-packages/mymodule-1.0-py2.7.egg-info
Writing /Library/Python/2.7/site-packages/mymodule-1.0-py2.7.egg-info
```

Now, check by loading the module and calling the function.

```
$ python testmodule.py
Creating extensions in Python is fun.!!
```

Congratulations ! You have created your own module and extended that to be used in the Python Interpreter.

Checkpoint

1 mark questions :

1. How does the user check if a certain package of Python is installed or not. e.g: Flask

A.pip list |grep Flask.

B.pip show Flask

2. What is the standard python standard library location URL, where users can search for new and download libraries?

A. http://www.python.org

B. http://pypi.python.org/pypi.

3. There are two types of libraries, Standard library and User Defined Library.

True.

False

4. It is mandatory to have the module to be imported in the same folder as the python program that is calling it ?

True

False.

5. Identify the correct way of Class definition in a module, or a standalone class python module.

A.Class Employee():

B.class Employee:

C.class Employee

Checkpoint

- **6. In the class definition above, identify this is a valid statement in the testmodule.py**

Can this format be used

from printmodule import Employee

from printmodule import print_hello_world

- A.True
- B.False

- **7. Identify the right format for importing modules into the code.**

from printmodule import Employee

from printmodule import print_hello_world

from printmodule import print_name_var

Or Just

import print module.

- Does both the formats work ?
- A.True
- B.False

Checkpoint

- **8. What is the system specific Python module that helps in defining the location of the module residing folder ?**

A.import os

B.import sys.

C.import printmodule

- **9. What is the sys function that helps in identifying the path or the folder where the new module resides ?**

A.sys.path.folderlocation

B.sys.path.append.

C.sys.args

10. What is the sys function that indicates the search path of the libraries in that order.

- A. sys.path ().
- B. sys.path.append ()
- C. sys.showpath()

Unit summary

Having completed this unit, you should be able to know:

- How to use Standard Libraries
- Data Compression and Archiving libraries
- Web Development, HTTP server
- Network programming, Sockets and other functions
- Client Server programming
- Extending and Embedding Python in C