

Programming in Python

GAVRILUT DRAGOS

COURSE 8

Sockets

Sockets are implemented through **socket** module in Python.

A socket object in Python has the same functions as a normal socket from C/C++: accept, bind, close, connect, listen, recv, send, ...

Besides this several other functions are available for domain translation, time outs, etc

Documentation for Python socket module can be found on:

- Python 2: <https://docs.python.org/2/library/socket.html>
- Python 3: <https://docs.python.org/3/library/socket.html>

Sockets

How to build a simple server/client in Python:

Python 2.x/3.x (Server)

```
import socket
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind(("127.0.0.1", 1234))
s.listen(1)
(connection, address) = s.accept()
print ("Connectd address:", address);
while True:
    data = connection.recv(100).decode("UTF-8")
    if not data: break
    print("Received: ", data)
    if "exit" in data: break
connection.close()
print ("Server closed")
```

Sockets

How to build a simple server/client in Python:

Python 2.x/3.x (Server)

```
import socket
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind(("127.0.0.1", 1234))
s.listen(1)
(connection, address) = s.accept()
print ("Connectd address:", address);
while True:
    data = connection.recv(100).decode("UTF-8")
    if not data: break
    print("Received: ", data)
    if "exit" in data: break
connection.close()
print ("Server closed")
```



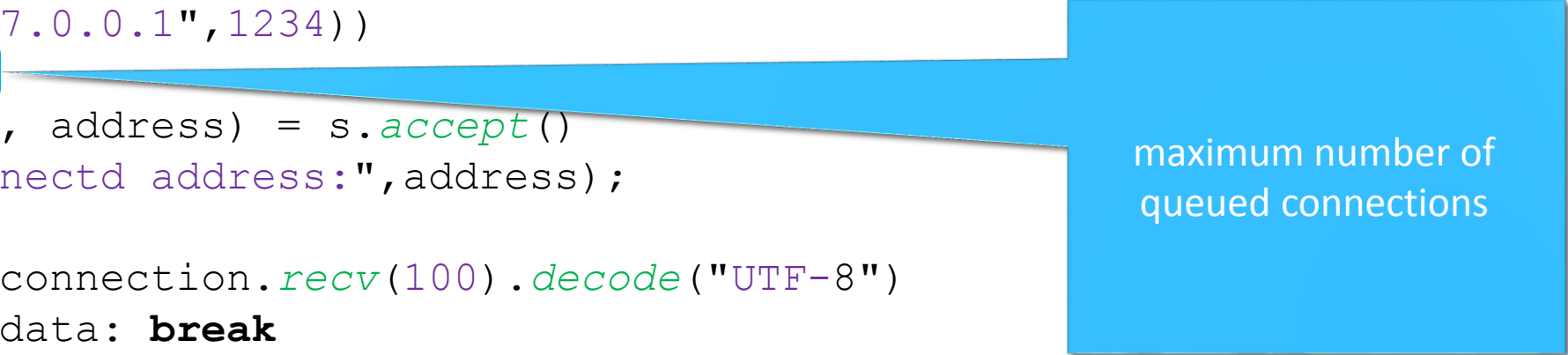
Address and port

Sockets

How to build a simple server/client in Python:

Python 2.x/3.x (Server)

```
import socket
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind(("127.0.0.1", 1234))
s.listen(1)
(connection, address) = s.accept()
print ("Connectd address:", address);
while True:
    data = connection.recv(100).decode("UTF-8")
    if not data: break
    print("Received: ", data)
    if "exit" in data: break
connection.close()
print ("Server closed")
```



maximum number of
queued connections

Sockets

How to build a simple server/client in Python:

Python 2.x/3.x (Server)

```
import socket
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind(("127.0.0.1", 1234))
s.listen(1)
(connection, address) = s.accept()
print ("Connectd address:", address);
while True:
    data = connection.recv(100).decode('utf-8')
    if not data: break
    print("Received: ", data)
    if "exit" in data: break
connection.close()
print ("Server closed")
```



Number of bytes to read

Sockets

How to build a simple server/client in Python:

Python 2.x/3.x (Server)

```
import socket
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind(("127.0.0.1", 1234))
s.listen(1)
(connection, address) = s.accept()
print ("Connectd address:", address);
while True:
    data = connection.recv(100).decode("UTF-8")
    if not data: break
    print("Received: ", data)
    if "exit" in data: break
connection.close()
print ("Server closed")
```

Use .decode("UTF-8") to convert a byte array to a string. Encoding in this case is done with UTF-8

Sockets

How to build a simple server/client in Python:

Python 2.x/3.x (Client)

```
import socket,time

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect(("127.0.0.1",1234))
s.send(b"Mesaj 1")
time.sleep(1)
s.send(b"Mesaj 2")
time.sleep(1)
s.send(b"exit")
s.close()
```

Output from the server

```
Connectd address: ('127.0.0.1', 61266)
Received:  Mesaj 1
Received:  Mesaj 2
Received:  exit
Server closed
```


Sockets

Getting current system IP

Python 2.x/3.x

```
import socket
print (socket.gethostbyname(socket.gethostname()))
```

Convert a host to an IP:

Python 2.x/3.x

```
import socket
print (socket.gethostbyname('uaic.ro'))
```

Getting the name associated with an IP:

Python 2.x/3.x

```
import socket
print (socket.gethostbyaddr("85.122.16.7"))
```

Output

```
('jad.uaic.ro', [],
['85.122.16.7'])
```

Sockets

Checking if a port is open:

Python 2.x/3.x (Client)

```
import socket

ip = "127.0.0.1"
ports = [20, 21, 23, 25, 80, 443, 530, 8080]
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.settimeout(3) #3 seconds timeout
for port in ports:
    if s.connect_ex((ip, port)) == 0:
        print("Port ", port, " is open")
    else:
        print("Port ", port, " is closed")
```

`connect_ex` returns an error code if the connection is not possible. `0` means no error.

URL modules

Python has several implementation for accessing the content of a URL. Some libraries are available only for Python 2.

The most widely used modules are urllib and urllib2 (available only for Python 2).

- Python 2: <https://docs.python.org/2/library/urllib2.html>
- Python 2: <https://docs.python.org/2/library/urllib.html>
- Python 3: <https://docs.python.org/3/library/urllib.html#module-urllib>

urllib2 module

Python 2 has a very simple way of downloading the content of an URL. The following script lists all professors from our faculty that are currently teaching.

Python 2.x

```
import urllib2,re

urlInfo = "http://profs.info.uaic.ro/~orar/orar_profesori.html"
r = re.compile("<li><a\shref=\"[^\"]\\.]*\\.html\">.*([A-Za-z\\.\\s,]+)</a>")

try:
    response = urllib2.urlopen(urlInfo).read()
    for mo in r.finditer(response):
        print (mo.group(1).strip())
except Exception as e:
    print ("Error -> ",e)
```

urllib module

Python 3 has another module (urllib). A similar module exists in Python 2. The following code extracts the name of the dean of our faculty.

Python 3.x

```
import urllib, re
from urllib import request
urlManagement = 'http://www.info.uaic.ro/bin/Structure/Management'
r = re.compile(b"Dean.*<a href=\"[^\"]+\">([A-Za-z\s]+)")

try:
    response = urllib.request.urlopen(urlManagement).read()
    obj = r.search(response)
    if obj:
        print ("Our dean is : ",obj.group(1))
except Exception as e:
    print ("Error -> ",e)
```

FTP module

Python has a module (ftplib) develop to enable working with FTP servers:

- Retrieve and store files
- Enumerate files from a FTP server
- Create folder on the FTP Server
- Support for password protected servers
- Support for custom FTP commands

Documentation

- Python 2: <https://docs.python.org/2/library/ftplib.html>
- Python 3: <https://docs.python.org/3/library/ftplib.html>

FTP module

The following snippet lists all directories from <ftp.debian.org> from folder debian.

Python 3.x

```
from ftplib import FTP
#drwxr-sr-x    18 1176      1176          4096 Sep 17 09:55 dists
def parse_line(line):
    if line.startswith("d"):
        print (line.rsplit(" ",1)[1])
try:
    client = FTP("ftp.debian.org")
    res = client.login()
    print (res)
    client.retrlines("LIST /debian/",parse_line)
    client.quit()
except Exception as e:
    print (e)
```

Output

```
230 Login successful.
dists
doc
indices
pool
project
tools
zzz-dists
```

FTP module

The following snippet downloads a file from a FTS server.

Python 3.x

```
from ftplib import FTP

cmdToDownload = "RETR /debian/extrfiles"
try:
    client = FTP("ftp.debian.org")
    res = client.login()
    f = open("debian_extrfiles", "wb")
    client.retrbinary(cmdToDownload ,lambda buf: f.write(buf))
    f.close()
    client.quit()
except Exception as e:
    print (e)
```


FTP module

List of all supported FTP commands:

| Command | Description |
|----------------|--|
| FTP.connect | Connect to a specified host on a specified port (default is 21) |
| FTP.login | Specifies the user name and password for ftp login |
| FTP.retrlines | Send a command to the FTP server and retrieves the results. The result is send line by line to a callback function |
| FTP.storbinary | Stores a file in a binary mode on the FTP server |
| FTP.retrbinary | Retrieves a binary file from the server. A callback must be provided to write the binary content. |
| FTP.rename | Rename a file/folder from the server |
| FTP.delete | Deletes a file from the server |
| FTP.rmd | Deletes a folder from the server. |

SMTP module

Python has a module (smtp) develop to enable working with emails.

While for simple emails (a subject and a text) the smtp module is enough, for more complex emails (attachment, etc) there is also another module (email.mime) that can be use to create an email.

Python 2 and Python 3 have some differences on how mime types can be used.

Documentation

- Python 2: <https://docs.python.org/2/library/smtplib.html>
- Python 2: <https://docs.python.org/2/library/email.mime.html>
- Python 3: <https://docs.python.org/3/library/smtplib.html>
- Python 3: <https://docs.python.org/3/library/email.mime.html>

SMTP module

The following can be used to send an email.

Python 2.x

```
import smtplib
from email.MIMEText import MIMEText
mail = smtplib.SMTP('smtp.gmail.com', 587)
mail.ehlo()
mail.starttls()
mail.login("<user_name>@gmail.com", "<Password>")
msg = MIMEText("My first email")
msg['Subject'] = "First email"
msg['From'] = "<user_name>@gmail.com"
msg['To'] = "<recipient email address>"
mail.sendmail("<from>", "<to>", msg.as_string())
mail.quit()
```

SMTP module

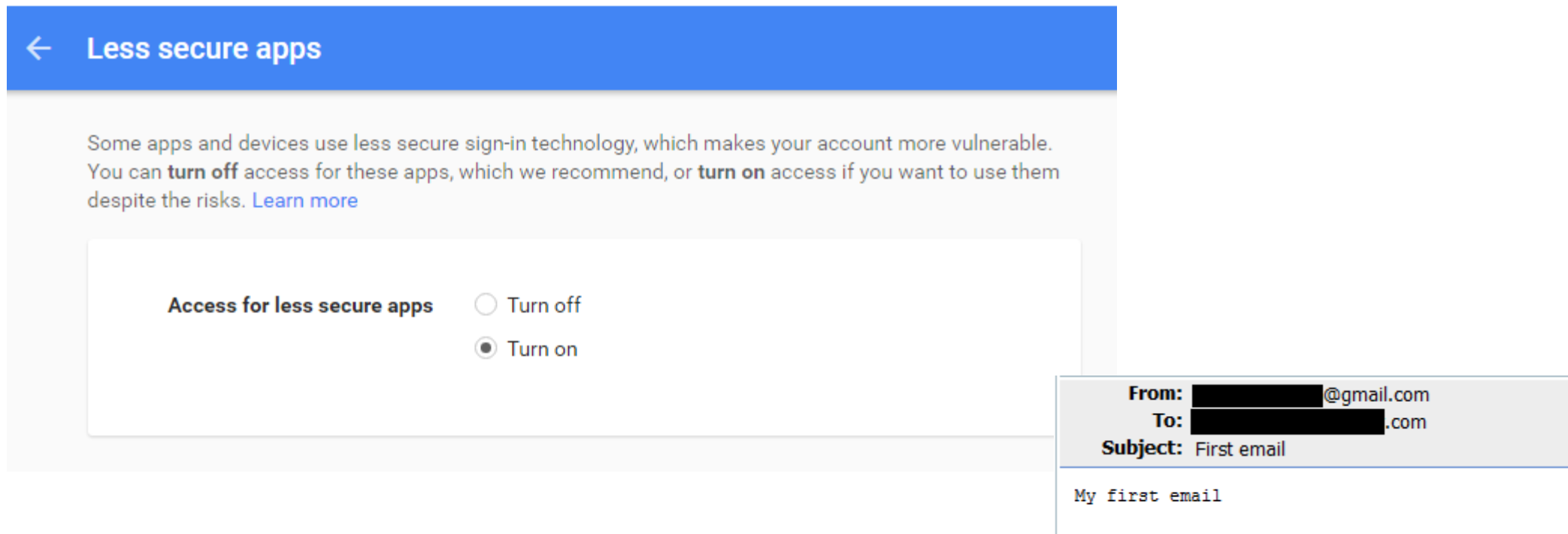
The following can be used to send an email.

Python 3.x

```
import smtplib
from email.mime.text import MIMEText
mail = smtplib.SMTP('smtp.gmail.com', 587)
mail.ehlo()
mail.starttls()
mail.login("<user_name>@gmail.com", "<Password>")
msg = MIMEText("My first email")
msg['Subject'] = "First email"
msg['From'] = "<user_name>@gmail.com"
msg['To'] = "<recipient email address>"
mail.sendmail("<from>", "<to>", msg.as_string())
mail.quit()
```

SMTP module

For the previous snippet to work with gmail servers you need to activate “Access for less secure apps” from you google account (<https://www.google.com/settings/security/lesssecureapps>)



SMTP module

Sending multiple attachments.

Python 3.x

```
import smtplib
from email.mime.multipart import MIMEMultipart
from email.mime.image import MIMEImage
mail = smtplib.SMTP('smtp.gmail.com', 587)
mail.ehlo()
mail.starttls()
mail.login("<user_name>@gmail.com", "<Password>")
msg = MIMEMultipart ()
msg['Subject'] = "First email"
msg['From'] = "<user_name>@gmail.com"
msg['To'] = "<recipient email address>"
msg.attach(MIMEImage(open("image.png", "rb").read()))
msg.attach(MIMEImage(open("image2.png", "rb").read()))
mail.send_message(msg)
mail.quit()
```

SMTP module (attachments + body)

Python 3.x

```
import smtplib
from email.mime.multipart import MIMEMultipart
from email.mime.image import MIMEImage
from email.mime.text import MIMEText
mail = smtplib.SMTP('smtp.gmail.com', 587)
mail.ehlo()
mail.starttls()
mail.login("<user_name>@gmail.com", "<Password>")
msg = MIMEMultipart("mixed")
msg['Subject'] = "First email"
msg['From'] = "<user_name>@gmail.com"
msg['To'] = "<recipient email address>"
msg.attach(MIMEText("The body of the email", 'plain'))
msg.attach(MIMEImage(open("image.png", "rb").read()))
mail.send_message(msg)
mail.quit()
```

Building a simple HTTP server

Python has some build in modules that can simulate a http server. There are some differences between Python 2 and Python 3 in terms of module names but the basic functionality is the same.

Python 2.x

```
import SimpleHTTPServer
import SocketServer

httpd = SocketServer.TCPServer(("127.0.0.1", 9000),
                               SimpleHTTPServer.SimpleHTTPRequestHandler)
httpd.serve_forever()
```

This script will create a HTTP server that listens on port 9000.

We will discuss more about these servers after we learn about classes.

Building a simple HTTP server

The default behavior for such a server is to produce a directory listing for the root where the script is.

However, if within the root a `index.html` file is found, that file will be loaded and send to the client.

These modules can also be executed automatically from the command line as follows:

- Python 2: **`python -m SimpleHTTPServer 9000`**
- Python 3: **`python -m http.server 9000`**

The last parameter from the command line is the port number.