

# Networking with Python

Thomas Schwarz, SJ

# Socket Programming In Python

- Python translates the UNIX socket interface
  - IPv4: Use a tuple IP-address, port number
- Sockets go through a life cycle:
  - Creation, Connection, Receiving / Sending, Closing
  - Creation, Binding, Listening, Closing

# Socket Programming In Python

- Example:
  - A simple writer to another process
  - Data is send as a byte stream
  - Using local-loop to avoid opening the firewall
    - Otherwise: Go to firewall administration and create a new rule to open port 61234 for TCP
    - Afterwards: remove this rule

# Socket Programming In Python

- **Server:**
  - Create socket
    - using ipv4 (AF\_INET) and TCP

```
import socket
```

```
HOST = '127.0.0.1' #Loopback interface
```

```
PORT = 65431 #Silly port
```

```
with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
```

# Socket Programming In Python

- Bind socket to port and listen

```
with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:  
    s.bind((HOST, PORT))  
    s.listen()  
    conn, addr = s.accept()
```

# Socket Programming In Python

- Receive data from client, stop when no data remains

```
with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:  
    s.bind((HOST, PORT))  
    s.listen()  
    conn, addr = s.accept()  
    print('Connection from:', conn)  
    while True:  
        data = conn.recv(1024)  
        if not data:  
            break  
        #conn.sendall(data) to return  
        print(data.decode('UTF-8'))
```

- Data is send in binary, as UTF-8

# Socket Programming In Python

- **Client:**
  - Instead of binding, we directly connect to the socket

```
import socket
```

```
HOST = '127.0.0.1' #Loopback interface
```

```
PORT = 65431 #Silly port
```

```
with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:  
    s.connect((HOST, PORT))
```

# Socket Programming In Python

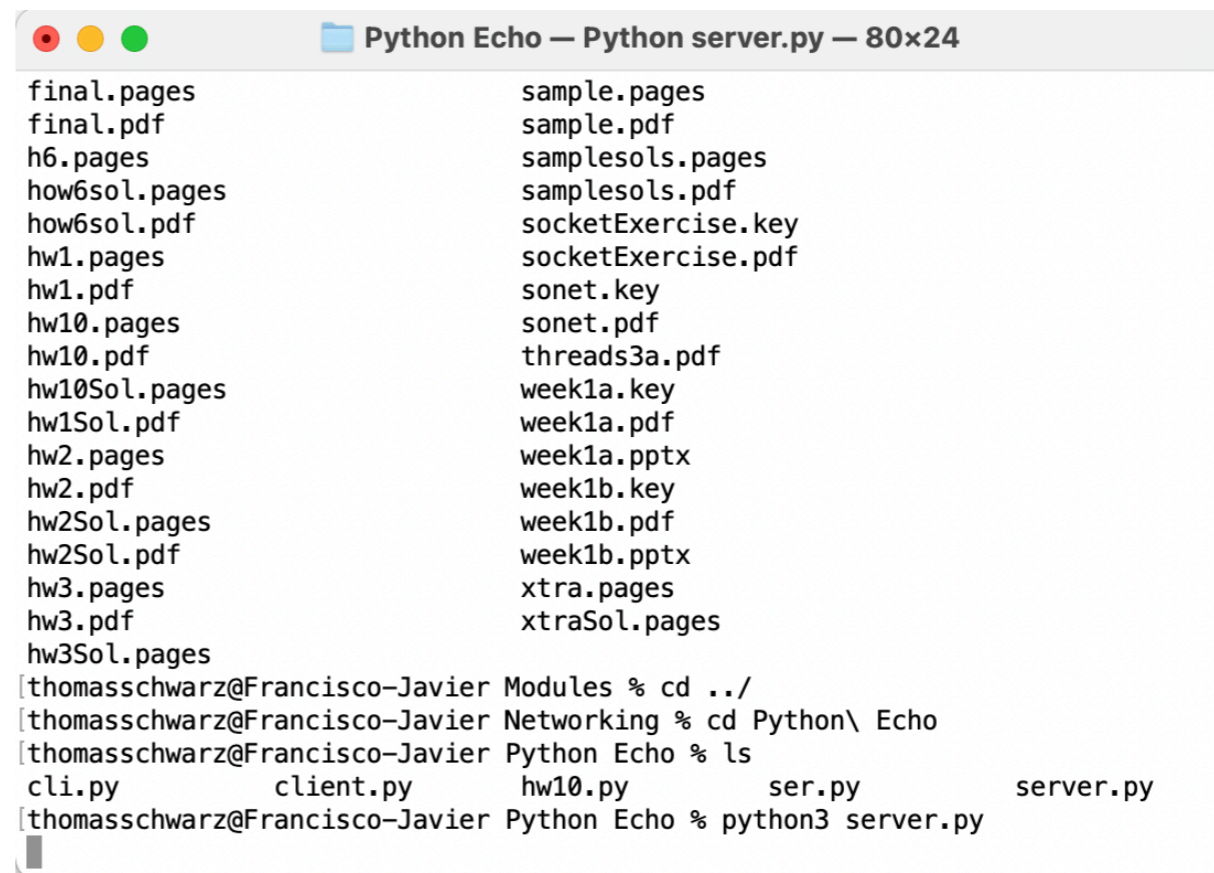
- **Client:**
  - Now we can write to the server

```
with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:  
    s.connect((HOST, PORT))  
    myinput = b'?:'  
    while myinput:  
        s.send(myinput)  
        myinput = bytes(input('?:'), 'utf-8')  
s.close() #not necessary
```



# Running in the Command Window

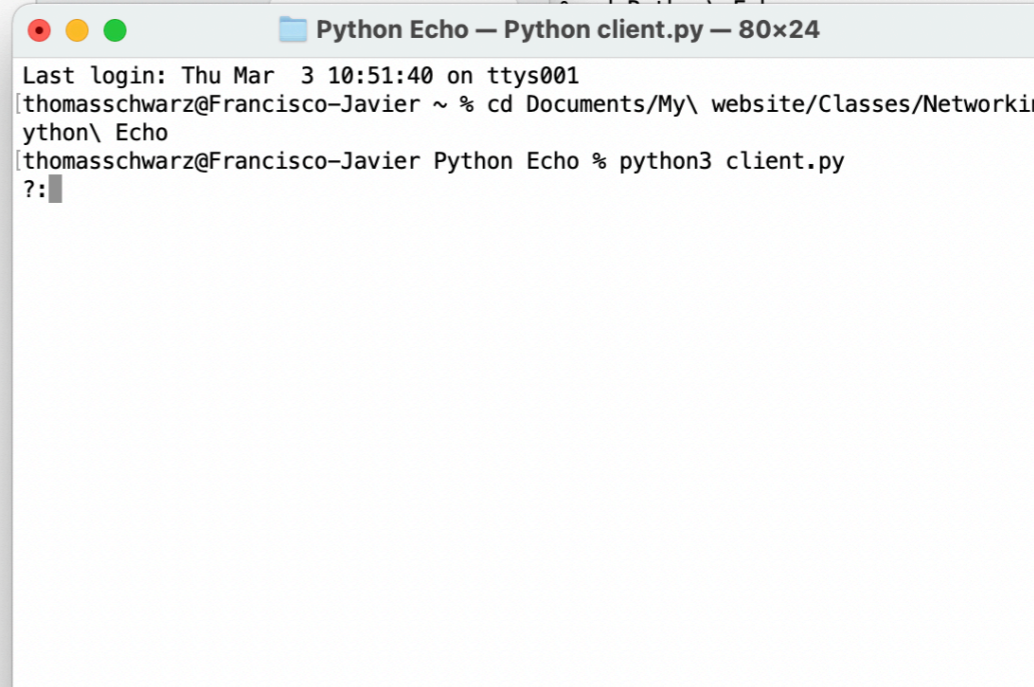
- Open a command window
- Navigate to your server program
- Invoke `python3 server.py`



```
Python Echo — Python server.py — 80x24
final.pages
final.pdf
h6.pages
how6sol.pages
how6sol.pdf
hw1.pages
hw1.pdf
hw10.pages
hw10.pdf
hw10Sol.pages
hw1Sol.pdf
hw2.pages
hw2.pdf
hw2Sol.pages
hw2Sol.pdf
hw3.pages
hw3.pdf
hw3Sol.pages
sample.pages
sample.pdf
samplesols.pages
samplesols.pdf
socketExercise.key
socketExercise.pdf
sonet.key
sonet.pdf
threads3a.pdf
week1a.key
week1a.pdf
week1a.pptx
week1b.key
week1b.pdf
week1b.pptx
xtra.pages
xtraSol.pages
[thomasschwarz@Francisco-Javier Modules % cd ../
[thomasschwarz@Francisco-Javier Networking % cd Python\ Echo
[thomasschwarz@Francisco-Javier Python Echo % ls
cli.py      client.py   hw10.py    ser.py     server.py
[thomasschwarz@Francisco-Javier Python Echo % python3 server.py
```

# Running in the Command Window

- Create another command window / terminal
- Navigate to your client program
- Type: `python3 client.py`
- Should respond with the prompt

A terminal window titled "Python Echo — Python client.py — 80x24" is shown. The terminal output includes the login message "Last login: Thu Mar 3 10:51:40 on ttys001", the user prompt "thomasschwarz@Francisco-Javier ~ %", the directory change command "cd Documents/My\ website/Classes/Networkir", the command "python\ Echo", and the execution of "python3 client.py" which results in a prompt "?:".

```
Python Echo — Python client.py — 80x24
Last login: Thu Mar 3 10:51:40 on ttys001
thomasschwarz@Francisco-Javier ~ % cd Documents/My\ website/Classes/Networkir
ython\ Echo
thomasschwarz@Francisco-Javier Python Echo % python3 client.py
?:
```

# Repetition

- Python allows us quickly create sockets

```
import socket
```

```
with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
```

# TCP Sockets in Python

- Setting up for listening:
  - Bind the socket to a host and port
    - If you do not want to fight firewalls, use the loopback address
    - Parameter is a tuple of Host and Port: Extra pair of ( )
  - Set the socket to listen for incoming packets
  - Accept segments

```
HOST = '127.0.0.1' #Loopback interface
PORT = 65431 #Silly port
with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
    s.bind((HOST, PORT))
    s.listen()
    conn, addr = s.accept()
    print('Connection from:', conn, 'Address', addr)
```

# TCP Sockets in Python

- Setting up for sending
  - Use connect and send / sendall

```
with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:  
    s.connect((HOST, PORT))  
    myinput = b'?:'  
    while myinput:  
        s.send(myinput)  
        myinput = bytes(input('?:'), 'utf-8')
```

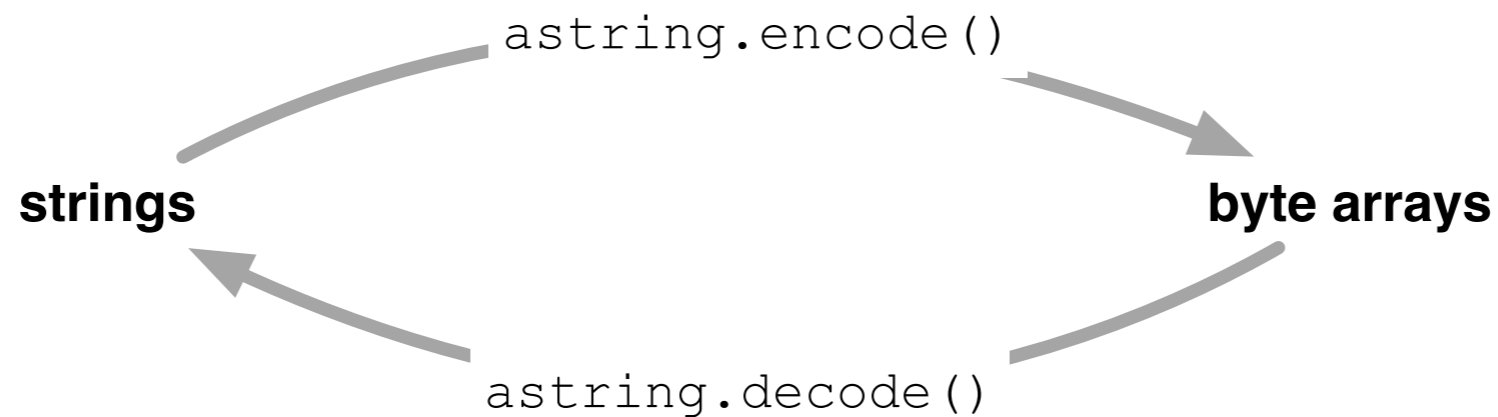
# TCP Sockets in Python

- Receiving communication:
  - Use a connection and receive
    - `conn.recv` blocks unless there is data
      - Parameter is the maximum number of bytes to receive at once

```
conn, addr = s.accept()
while True:
    data = conn.recv(1024)
    if not data:
        break
```

# Bytes vs. Strings

- The type of the message is an array of bytes
- To move between bytes and strings use encode and decode



# Bytes vs. Strings

- encode and decode have two optional parameters:
  - encoding
    - One of the many character encodings known to Python
    - Default is 'UTF-8'
  - errors
    - What to do if there is a bad character:
      - 'strict', 'ignore', 'replace'



# Bytes vs. Strings

- Examples:

- ```
>>> astring='hello world'
>>> astring.encode('utf-8')
b'hello world'
```

- ```
>>> bytestring = b'hello world'
>>> bytestring
b'hello world'
>>> bytestring.decode('utf-8')
'hello world'
```

# String Parsing Primer

- Whether we have a default string or a string of bytes:
  - For parsing:
    - Use split in order to create an array of components in a composite string
    - The default is splitting at white space
    - Example:
      - ```
>>> bs = b'12 13 14'  
>>> bs.split()  
[b'12', b'13', b'14']
```

# String Parsing Primer

- Whether we have a normal string or a string of bytes
  - For conversion:
    - String to Integer: use `int()`
    - String to Float: use `float()`

```
>>> bytestring = b'3.145 2.76'  
>>> for number in bytestring.split():  
    print(float(number))
```

```
3.145  
2.76
```

# String Parsing Primer

- To convert a number to a byte string
  - Convert to a string
  - Then convert the string using `bytes()`
    - But need to specify encoding
- Example:
  - ```
>>> bytes(str(3.145), encoding='utf-8')  
b'3.145'
```