Redes de Computadores

Lab03 - Data Serialization/Deserialization

Threads

Get directory

Python Data Serialization/Deserialization

- Packets exchanged between clients and servers contain arrays of bytes.
- How to put/retrieve more complex Python data structures (as objects)?
  - To do this we must **Serialize**/**Deserialize** objects
  - **Serialization**: Conversion from an object declared in a programming language to a byte array
  - **Deserialization**: Conversion from byte arrays to an object
- In Python, the methods available in the ***pickle*** package allow the serialization and deserialization of data structures (like tuples), in a very easy way.
  - To serialize we have the method ***dumps***
  - To deserialize we have the method ***loads***

Example: Serialization/Deserialization of Tuples in Python

```python
import pickle
...

request = (fileName, offset, blockSize)
#serialize the tuple request
req = pickle.dumps(request)
UDPSocket.sendto(req, endpoint)

...
```

```python
import pickle
...

message, address = sock.recvfrom(1024)
#deserialize recv message
request=pickle.loads(message)
fileName = request[0]
offset = request[1]
noBytes = request[2]
print(f'file= {fileName},offset={offset},
       noBytes={noBytes}')
...
```

Threads in Python

Example:

```
import logging
import threading
import time

def thread_fun(name):
    logging.info("Thread %s: starting", name)
    time.sleep(2)
    logging.info("Thread %s: finishing", name)

if __name__ == "__main__":
    format = "%(asctime)s: %(message)s"
    logging.basicConfig(format=format, level=logging.INFO,
                        datefmt="%H:%M:%S")
    logging.info("Main    : before creating thread")
    x = threading.Thread(target=thread_fun, args=(1,))
    logging.info("Main    : before running thread")
    x.start()
    logging.info("Main    : wait for the thread to finish")
    # x.join()
    logging.info("Main    : all done")
```

Multi-threaded Server Code

Why Use Multi-threading in Socket Programming?

- To serve multiple clients simultaneously without blocking
- To improve responsiveness of the server
- To separate logic per client (e.g., each client runs its session in parallel)

Multi-threaded Server Code

- This server code uses sockets and multi-threading to handle multiple client connections.
- Each client gets its own thread, and the server sends back the reversed message received from the client.

Multi-threaded Server Code

```python
import socket
import threading
import time

def handle_client(c):
    while True:
        data = c.recv(1024)
        dataD = data.decode()
        if not dataD:
            print('Bye')
            break
        time.sleep(5)
        print("received from client: ", dataD)
        c.send(data)
    c.close()

def main():
    host = ''
    port = 12345
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.bind((host, port))
    s.listen(5)
    print("Server running on port", port)
    while True:
        c, addr = s.accept()
        print('Connected to:', addr[0], ':', addr[1])
        tid =  threading.Thread(target=handle_client, args = (c,))
        tid.start()
        print("Created thread to deal with client")

if __name__ == '__main__':
    main()
```

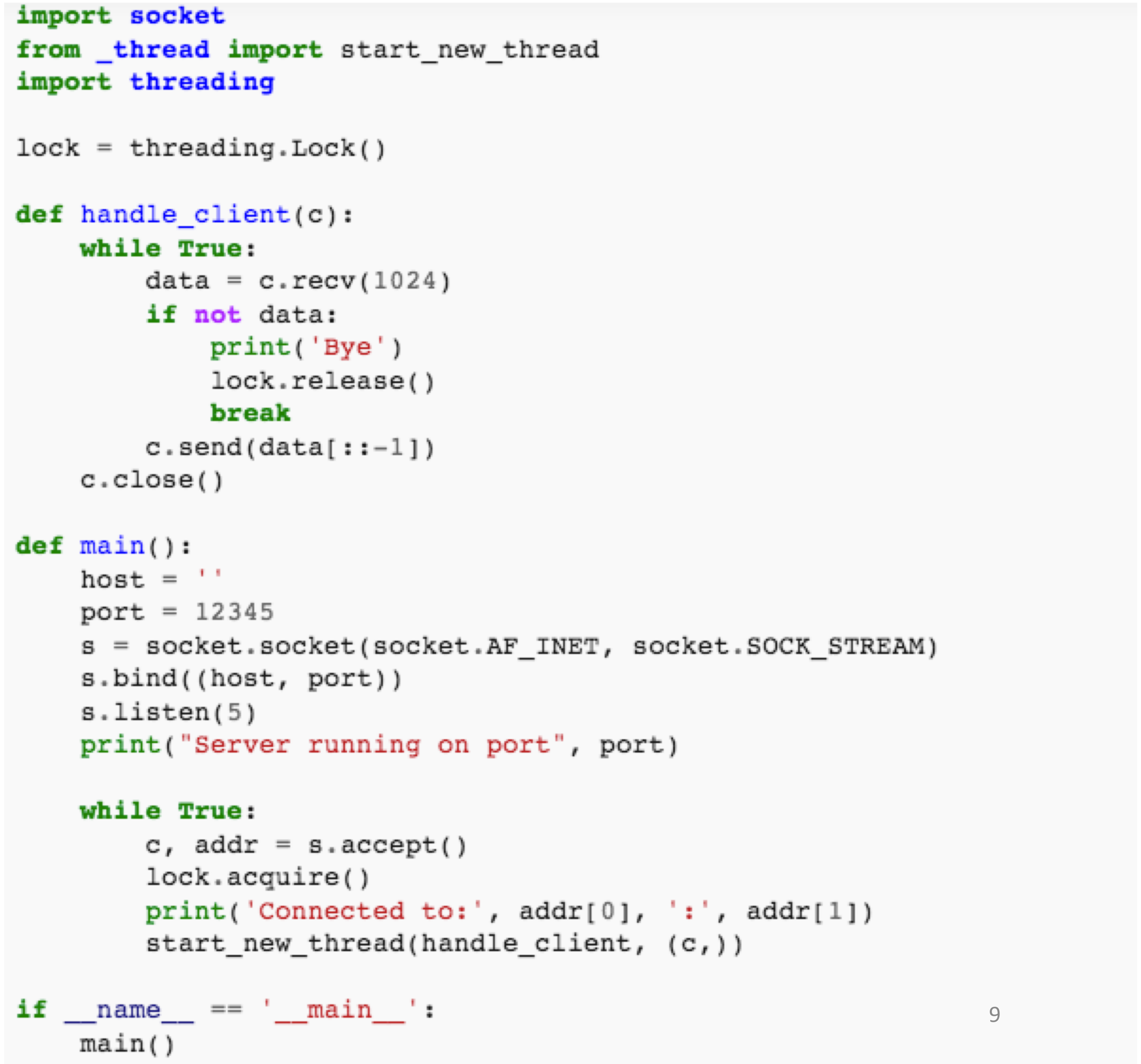
NOVA SCHOOL OF SCIENCE & TECHNOLOGY
DEPARTAMENTO DE INFORMÁTICA

Multi-threaded Client Code

```
import socket

def main():
    host = '127.0.0.1'
    port = 12345

    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.connect((host, port))


    msg = "hello from client"
    while True:
        s.send(msg.encode())
        data = s.recv(1024)
        print('Received from server:', data.decode())

        ans = input('Do you want to continue (y/n): ')
        if ans.lower() != 'y':
            break
    s.close()

if __name__ == '__main__':
    main()
```

NOVA SCHOOL OF SCIENCE & TECHNOLOGY
DEPARTAMENTO DE INFORMÁTICA

Multi-threaded Server Code

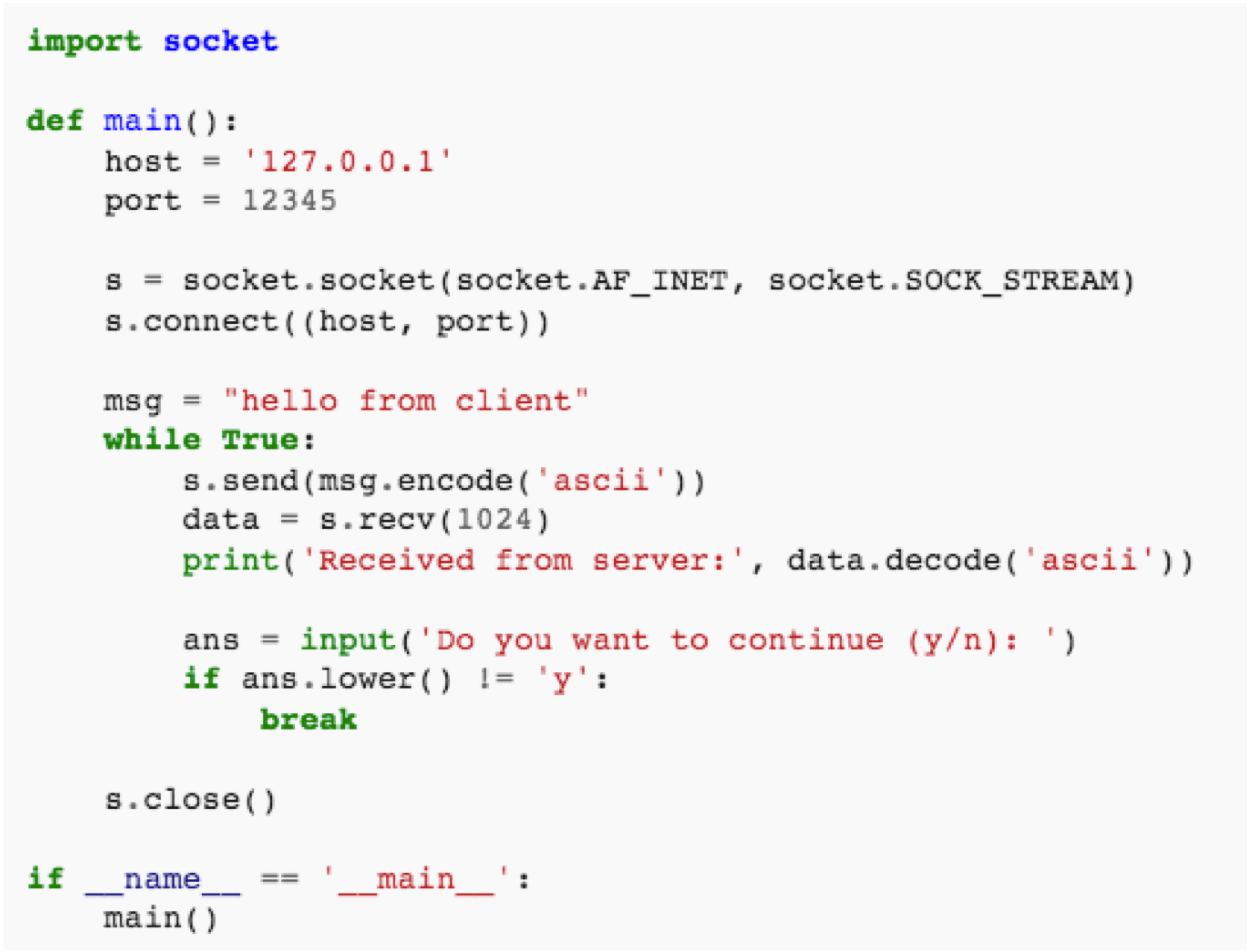


```python
import socket
from _thread import start_new_thread
import threading

lock = threading.Lock()

def handle_client(c):
    while True:
        data = c.recv(1024)
        if not data:
            print('Bye')
            lock.release()
            break
        c.send(data[::-1])
    c.close()

def main():
    host = ''
    port = 12345
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.bind((host, port))
    s.listen(5)
    print("Server running on port", port)

    while True:
        c, addr = s.accept()
        lock.acquire()
        print('Connected to:', addr[0], ':', addr[1])
        start_new_thread(handle_client, (c,))

if __name__ == '__main__':
    main()
```

https://www.geeksforgeeks.org/python/socket-programming-multi-threading-python/

NOVA SCHOOL OF
SCIENCE & TECHNOLOGY
DEPARTAMENTO DE INFORMÁTICA

Multi-threaded Client Code



```python
import socket

def main():
    host = '127.0.0.1'
    port = 12345

    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.connect((host, port))

    msg = "hello from client"
    while True:
        s.send(msg.encode('ascii'))
        data = s.recv(1024)
        print('Received from server:', data.decode('ascii'))

        ans = input('Do you want to continue (y/n): ')
        if ans.lower() != 'y':
            break

    s.close()

if __name__ == '__main__':
    main()
```

https://www.geeksforgeeks.org/python/socket-programming-multi-threading-python/

Lock Object - Thread Synchronization in Python

- In multithreading when multiple threads are working simultaneously on a shared resource some sort of locking mechanism should be used, to avoid concurrent modification errors. So when one thread is accessing a resource it takes a lock on that resource and only when it releases that lock other thread can access the same resource.
- In Python there are two basic methods: **`acquire()`** and **`release()`**.

Lock Object - Thread Synchronization in Python

`acquire(blocking=True, timeout=-1)`

- Used to acquire the lock. Without any optional argument, this method acquires the lock unconditionally, it blocks until the lock is unlocked.
  - If the *blocking* argument is present, the action depends on its value:
    - if it is false, the lock is only acquired if it can be acquired immediately without waiting.
    - if it is true, the lock is acquired unconditionally.
  - If the floating-point *timeout* argument is present and positive, it specifies the maximum wait time in seconds before returning. A negative *timeout* argument specifies an unbounded wait. You cannot specify a *timeout* if *blocking* is false.

Lock Object - Thread Synchronization in Python

`release()`

- Releases the lock. The lock must have been acquired earlier, but not necessarily by the same thread.
- It throws a RuntimeError if it is invoked on an unlocked lock.

Python - list all files in a directory

Code example (getDirFiles.py):

```python
import os

# get the list of all files and directories
dir_path = "."       # current path
dir_list = os.listdir(dir_path)

print("Files and directories in '", dir_path, "' :")

# print all files and subdirs
print("ALL")
for x in dir_list:
    print(x)

# list to store files
res = []

# print files only
for path in os.listdir(dir_path):
    # check if current path is a file
    if os.path.isfile(os.path.join(dir_path, path)):
        res.append(path)

print("FILES")
for f in res:
    print(f)
```

Python - Catch a Keyboard Interrupt

```python
import time

def main():
    print("Starting ...")

    while True:
        try:
            print(".", end=' ',flush=True)
            time.sleep(1)

        except KeyboardInterrupt:
            print("Exiting!")
            break

    print("Ending ")

main()
```