TCP/IP & THREADS

TCP (TRANSMISSION CONTROL PROTOCOL)

TCP (TRANSMISSION CONTROL PROTOCOL)

- Network communication protocol

TCP (TRANSMISSION CONTROL PROTOCOL)

- Network communication protocol
- Uses IP addresses for addressing (IPv4 and IPv6)

TCP (TRANSMISSION CONTROL PROTOCOL)

- Network communication protocol
- Uses IP addresses for addressing (IPv4 and IPv6)
- Stream based

TCP (TRANSMISSION CONTROL PROTOCOL)

- Network communication protocol
- Uses IP addresses for addressing (IPv4 and IPv6)
- Stream based
- Reliable:

TCP (TRANSMISSION CONTROL PROTOCOL)

- Network communication protocol
- Uses IP addresses for addressing (IPv4 and IPv6)
- Stream based
- Reliable:
  - Messages cannot be reordered

TCP (TRANSMISSION CONTROL PROTOCOL)

- Network communication protocol
- Uses IP addresses for addressing (IPv4 and IPv6)
- Stream based
- Reliable:
  - Messages cannot be reordered
  - Messages are delivered exactly once

SOCKET

Abstraction of one endpoint of a comm. channel

SOCKET

Abstraction of one endpoint of a comm. channel

- **bind(address, port)** - assigns socket to an address

SOCKET

Abstraction of one endpoint of a comm. channel

- **bind(address, port)** - assigns socket to an address
- **listen()** - starts listening for client connections

SOCKET

Abstraction of one endpoint of a comm. channel

- **bind(address, port)** - assigns socket to an address
- **listen()** - starts listening for client connections
- **accept()** - accept client connection

SOCKET

Abstraction of one endpoint of a comm. channel

- **bind(address, port)** - assigns socket to an address
- **listen()** - starts listening for client connections
- **accept()** - accept client connection
- **connect(address, port)** - connect to an address

SOCKET

Abstraction of one endpoint of a comm. channel

- **bind(address, port)** - assigns socket to an address
- **listen()** - starts listening for client connections
- **accept()** - accept client connection
- **connect(address, port)** - connect to an address
- **sendall(data)** - send data

SOCKET

Abstraction of one endpoint of a comm. channel

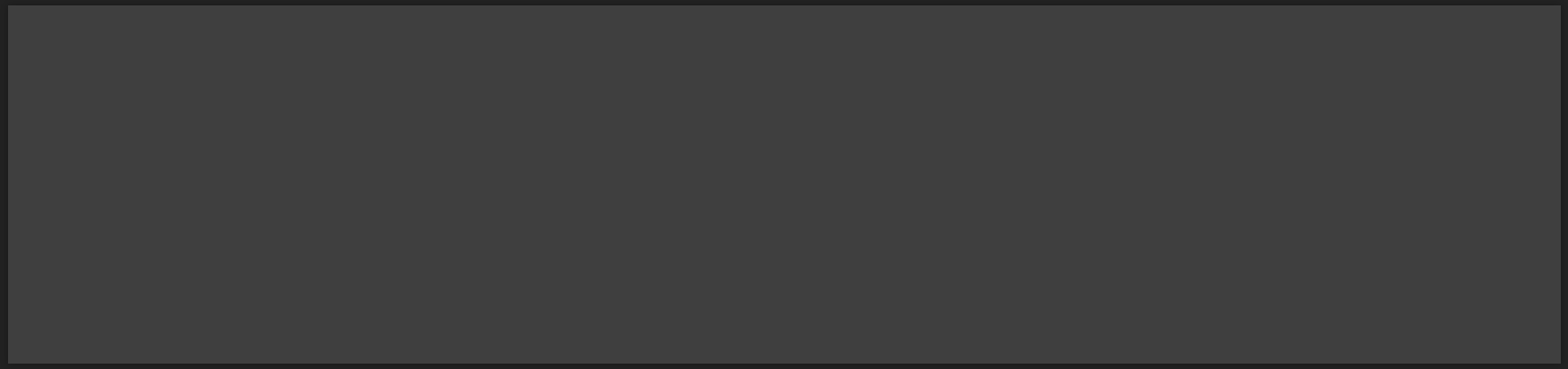
- **bind(address, port)** - assigns socket to an address
- **listen()** - starts listening for client connections
- **accept()** - accept client connection
- **connect(address, port)** - connect to an address
- **sendall(data)** - send data
- **recv(size)** - receive at most **size** bytes

SOCKET

Abstraction of one endpoint of a comm. channel

- **bind(address, port)** - assigns socket to an address
- **listen()** - starts listening for client connections
- **accept()** - accept client connection
- **connect(address, port)** - connect to an address
- **sendall(data)** - send data
- **recv(size)** - receive at most **size** bytes
- **close()** - end the communication

Server workflow



Server workflow

```
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

Server workflow

```
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
```

Server workflow

```
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
sock.bind(('127.0.0.1', 5555)) # assign to address
```

Server workflow

```
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
sock.bind(('127.0.0.1', 5555)) # assign to address
sock.listen() # start listening
```

Server workflow

```
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
sock.bind(('127.0.0.1', 5555)) # assign to address
sock.listen() # start listening
client, addr = sock.accept() # accept client
```

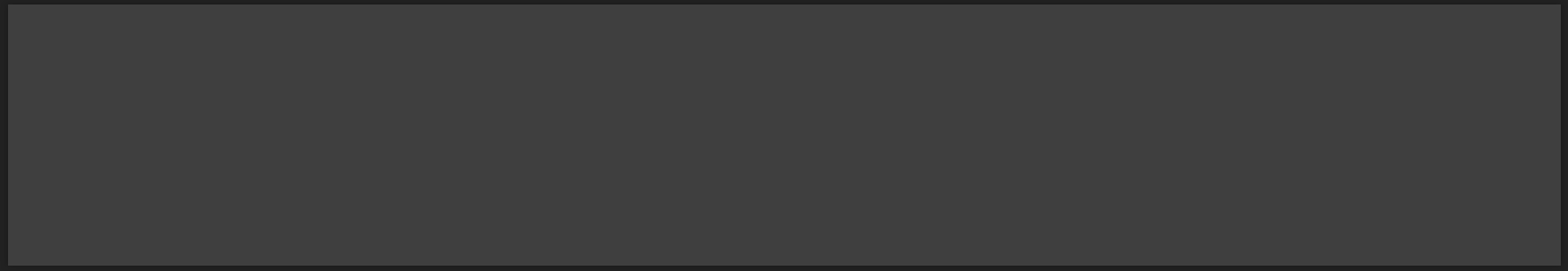
Server workflow

```
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
sock.bind(('127.0.0.1', 5555)) # assign to address
sock.listen() # start listening
client, addr = sock.accept() # accept client
client.sendall(b"hello")
```

Server workflow

```
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
sock.bind(('127.0.0.1', 5555)) # assign to address
sock.listen() # start listening
client, addr = sock.accept() # accept client
client.sendall(b"hello")
data = client.recv(512)
```

Client workflow



Client workflow

```
client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

Client workflow

```
client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.connect(('127.0.0.1', 5555)) # connect to server
```

Client workflow

```
client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.connect(('127.0.0.1', 5555)) # connect to server
client.sendall(b"hello")
```

Client workflow

```
client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.connect(('127.0.0.1', 5555)) # connect to server
client.sendall(b"hello")
data = client.recv(512)
```

Client workflow

```
client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.connect(('127.0.0.1', 5555)) # connect to server
client.sendall(b"hello")
data = client.recv(512)
client.close() # close connection
```

Gotchas

Stream based

```
# Client
client.sendall("msg1\n")
client.sendall("msg2\n")
client.sendall("msg3\n")
```

Gotchas

Stream based

```
# Client
client.sendall("msg1\n")
client.sendall("msg2\n")
client.sendall("msg3\n")

# Server
data = sock.recv(512)
# data may be b"msg1\nmsg2\nmsg3\n"
```

Gotchas

Stream based

```
# Client
client.sendall("msg1\n")
client.sendall("msg2\n")
client.sendall("msg3\n")

# Server
data = sock.recv(512)
# data may be b"msg1\nmsg2\nmsg3\n"
# but also b"ms", b"msg1", ...
```

Gotchas

Stream based

```
# Client
client.sendall("msg1\n")
client.sendall("msg2\n")
client.sendall("msg3\n")

# Server
data = sock.recv(512)
# data may be b"msg1\nmsg2\nmsg3\n"
# but also b"ms", b"msg1", ...
# => you have to delimit messages by yourself
```

How to recognize disconnect

```
data = sock.recv(512)
if data == 0:
  # error during transmission, disconnect
```

Operations are blocking by default

```
sock.accept()          # blocks until a client connects
sock.sendall(b"...")   # blocks until all data is sent
sock.recv(512)         # blocks until (some) data is received
```

Encoding

Encoding

- Sockets work with bytes, not strings

Encoding

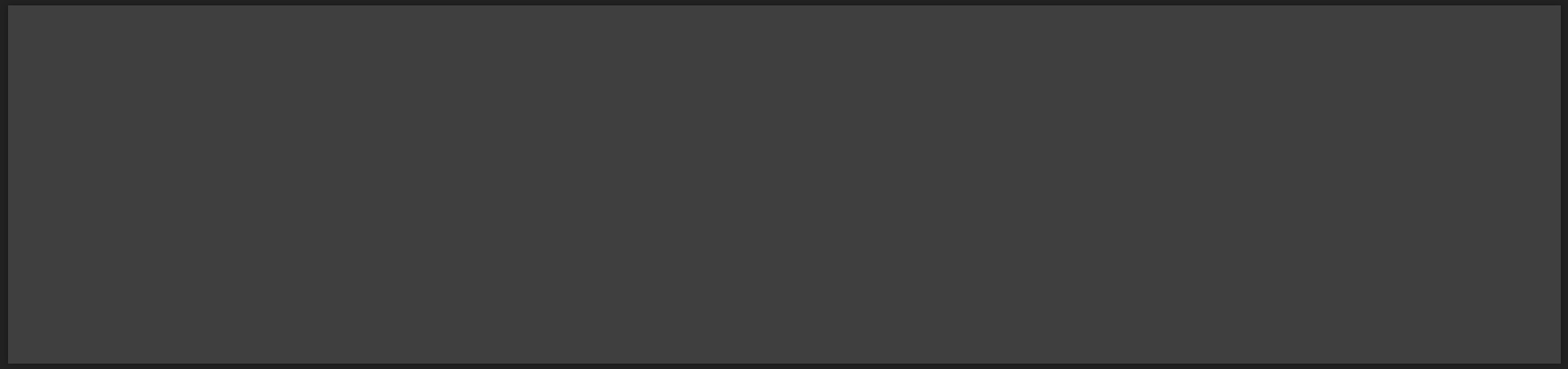
- Sockets work with bytes, not strings
- Before sending a string, you have to encode it

Encoding

- Sockets work with bytes, not strings
- Before sending a string, you have to encode it
- After receiving a string, you have to decode it

Encoding

- Sockets work with bytes, not strings
- Before sending a string, you have to encode it
- After receiving a string, you have to decode it



Encoding

- Sockets work with bytes, not strings
- Before sending a string, you have to encode it
- After receiving a string, you have to decode it

```
msg = "Hello world!\n"
```

Encoding

- Sockets work with bytes, not strings
- Before sending a string, you have to encode it
- After receiving a string, you have to decode it

```
msg = "Hello world!\n"
sock.sendall(msg.encode()) # by default uses UTF-8
```

Encoding

- Sockets work with bytes, not strings
- Before sending a string, you have to encode it
- After receiving a string, you have to decode it

```
msg = "Hello world!\n"
sock.sendall(msg.encode()) # by default uses UTF-8
# or you can use bytes literal
sock.sendall(b"Hello world\n")
```

Encoding

- Sockets work with bytes, not strings
- Before sending a string, you have to encode it
- After receiving a string, you have to decode it

```
msg = "Hello world!\n"
sock.sendall(msg.encode()) # by default uses UTF-8
# or you can use bytes literal
sock.sendall(b"Hello world\n")

byte_data = sock.recv(512)
```

Encoding

- Sockets work with bytes, not strings
- Before sending a string, you have to encode it
- After receiving a string, you have to decode it

```
msg = "Hello world!\n"
sock.sendall(msg.encode()) # by default uses UTF-8
# or you can use bytes literal
sock.sendall(b"Hello world\n")

byte_data = sock.recv(512)
string = byte_data.decode()
```

THREADS

THREADS

- Unit of scheduling

THREADS

- Unit of scheduling
- Registers + memory address space

THREADS

- Unit of scheduling
- Registers + memory address space

Useful for:

THREADS

- Unit of scheduling
- Registers + memory address space

Useful for:

- Concurrency

THREADS

- Unit of scheduling
- Registers + memory address space

Useful for:

- Concurrency
- Parallelism

THREADS IN PYTHON

THREADS IN PYTHON

- Only one thread may execute at any given time

THREADS IN PYTHON

- Only one thread may execute at any given time
- Interpreter is locked by GIL (global interpreter lock)

THREADS IN PYTHON

- Only one thread may execute at any given time
- Interpreter is locked by GIL (global interpreter lock)
- Concurrency ✓

THREADS IN PYTHON

- Only one thread may execute at any given time
- Interpreter is locked by GIL (global interpreter lock)
- Concurrency ✓
- Parallelism ×

THREADS IN PYTHON

- Only one thread may execute at any given time
- Interpreter is locked by GIL (global interpreter lock)
- Concurrency ✓
- Parallelism ×
- You have to use processes to get parallelism

USING THREADS

USING THREADS

```
def fun(num):
  for _ in range(5):
    print("Thread {}".format(num))
```

USING THREADS

```
def fun(num):
  for _ in range(5):
    print("Thread {}".format(num))

import threading
threads = []
```

USING THREADS

```
def fun(num):
  for _ in range(5):
    print("Thread {}".format(num))

import threading
threads = []
for i in range(3):
```

USING THREADS

```
def fun(num):
  for _ in range(5):
    print("Thread {}".format(num))

import threading
threads = []
for i in range(3):
  t = threading.Thread(target=fun, args=(i, ))
```

USING THREADS

```
def fun(num):
  for _ in range(5):
    print("Thread {}".format(num))

import threading
threads = []
for i in range(3):
  t = threading.Thread(target=fun, args=(i, ))
  t.start()
  threads.append(t)
```

USING THREADS

```
def fun(num):
  for _ in range(5):
    print("Thread {}".format(num))

import threading
threads = []
for i in range(3):
  t = threading.Thread(target=fun, args=(i, ))
  t.start()
  threads.append(t)

for t in threads:
  t.join() # block until thread ends
```