

Network Programming in Python

Prepared by Dr. Hamada I. AbdulWakel

Lecture 10



File Transfer Protocol In Python

Lecture 10

Prepared by
Dr. Hamada I. AbdulWakel

Lecture

10

RECAP: LECTURE 9

- Sending Emails.
- Securing Emails.
- Retrieving Emails.

Lecture

10

LECTURE MAIN POINTS

- The File Transfer Protocol.
- FTP Python Module.
- Creating an FTP Client.

Lecture

10

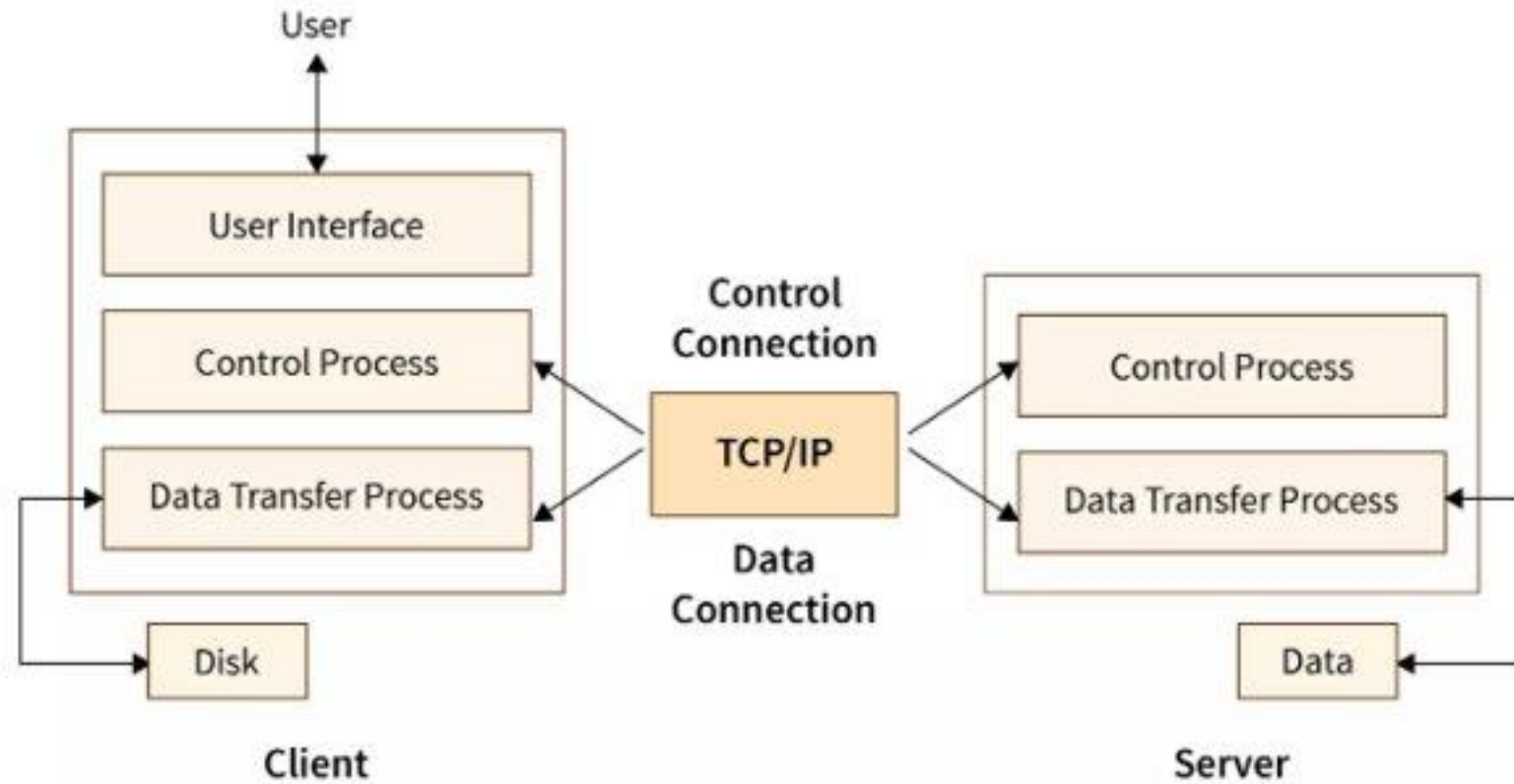
FILE TRANSFER PROTOCOL

- FTP is a standard internet protocol provided by TCP/IP used for transmitting/downloading files between client and server.
- FTP provides the sharing of files.
- FTP overcomes the problem of transferring and translating file types between different operating systems.

Lecture

10

FTP PARTS



Lecture

10

FTP PARTS

Cont'd

- FTP client has three components:
 - The user interface.
 - The control process.
 - The data transfer process.
- The server has two components:
 - The server control process.
 - The server data transfer process.

Lecture

10

FTP COMMUNICATION CONNECTIONS

- FTP actually uses two TCP connections during operation.
- The control connection:
 - Carries commands and the resulting acknowledgments or error codes.
 - Uses very simple rules for communication.
 - Remains opened during the entire interactive FTP session.

Lecture

10

FTP COMMUNICATION CONNECTIONS Cont'd

- The data transfer connection:
 - Used for transmitting file data or other blocks of information, such as directory listings.
 - Uses very complex rules as data types may vary.
 - Opens when a command comes for transferring the files and closes when the file is transferred.

Lecture

10

FTP CONTROL CONNECTION COMMANDS

- **OPEN:** used to open new connection with the server.
- **USER:** sends the user identification to the server.
- **PASS:** sends the user password to the server.
- **CWD:** allows the user to work with a different directory without altering his login information.

Lecture

10

FTP CONTROL CONNECTION COMMANDS

Cont'd

- **MKD:** causes the directory specified in the path-name to be created as a directory.
- **PUT:** used to put a file in the server.
- **CLOSE:** used to close connection with the server.

Lecture

10

LOCAL FTP SERVER

- [FileZilla](#) is a free and open-source, cross-platform FTP application.
- Run the executable and complete the installation.
- While installation, set the server administration password and remember the server administration port.

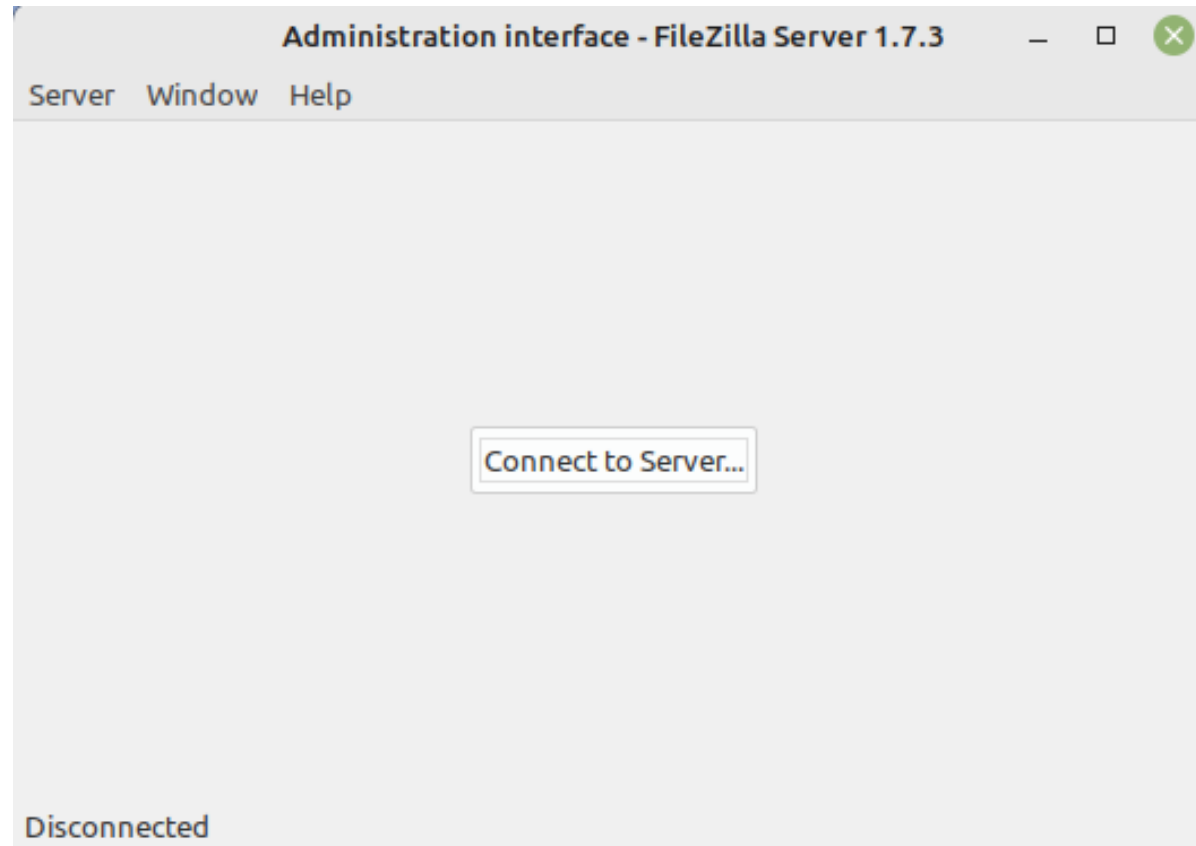


Lecture

10

LOCAL FTP SERVER

Cont'd

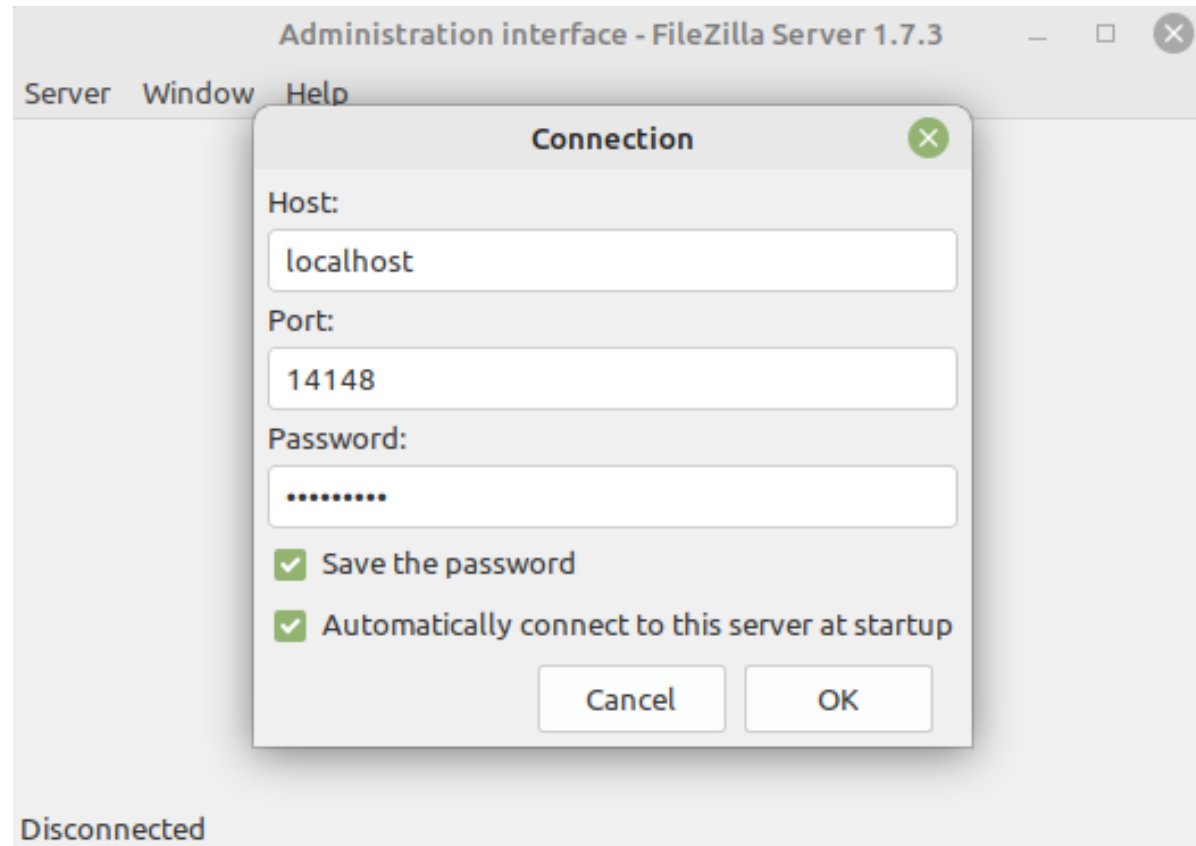


Lecture

10

LOCAL FTP SERVER

Cont'd

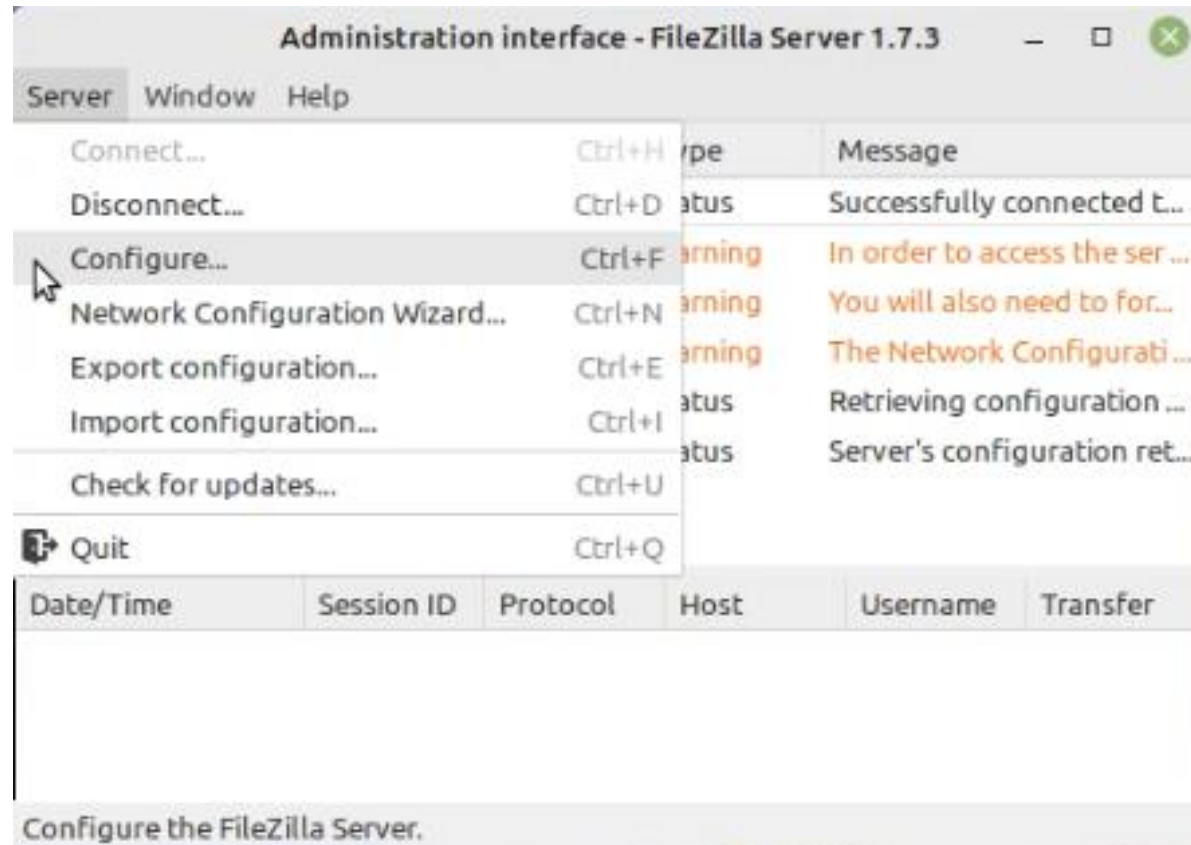


Lecture

10

LOCAL FTP SERVER

Cont'd



Lecture

10

LOCAL FTP SERVER

Cont'd

Settings for server localhost:14148

Select a page: Rights management / Users

Server listeners

[-] Protocols settings

 FTP and FTP over TLS (FTPS)

[-] Rights management

 Groups

Users

Administration

Logging

Let's Encrypt®

Updates

Available users

<system user>

hamada

General Filters Speed Limits

☒ User is enabled

Authentication:

Require a password to log in

.....

Member of groups:

Mount points:

Virtual path	Native path	Mount options
/	/mnt/367D9E	Access mode: Read + Write ☒ Apply permissions to subdirectories ☒ Writable directory structure ☐ Create native directory if it does not exist

Add Remove

Description:

Add Remove Duplicate Rename

Apply Cancel OK

Lecture

10

FTP MODULE

- Python provides FTP module in its standard library called **ftplib** that implements the client side of the FTP protocol.
- Features:
 - Connecting to an FTP server.
 - Navigating its structure.
 - Downloading files from the FTP server.
 - Uploading files to an FTP server.

Lecture

10

FTP MODULE METHOD

- **FTP()**: create an FTP client instance.
- **FTP_TLS()**: create an FTP client instance under a secure connection.
- **connect()**: connect to an FTP server.
- **login()**: login to an FTP server.
- **prot_p()**: used with a secure FTP connection.

Lecture

10

FTP MODULE METHOD

Cont'd

- **pwd()**: get current working directory in FTP server.
- **cwd()**: change the working directory.
- **nlst()**: return a list of file and folder names in a working directory.
- **storbinary()**: used to upload a file to an FTP server.
- **retrbinary()**: used to download a file from an FTP server.

Lecture

10

FTP MODULE METHOD

Cont'd

- **rename()**: rename a file or folder in an FTP server.
- **delete()**: delete a file or folder from an FTP server.
- **rmd()**: delete a folder from an FTP server.
- **mkd()**: create a folder in an FTP server.
- **size()**: return the size of a file in an FTP server.
- **quit()**: close an FTP server connection.

Lecture

10

FTP CLIENT

- Connection steps:
 - Set up connection parameters.
 - Create an FTP client instance.
 - Connect to FTP server.
 - Login to FTP server.
 - Perform required operation on FTP server.
 - Close connection.

Lecture

10

FTP CLIENT

Cont'd

- Code:

```
import ftplib
ftphost = "localhost"
ftpport = 21
ftpusername = "yourUserName"
ftppassword = "youPassword"
ftp = ftplib.FTP()
ftp.connect(ftphost, ftpport)
ftp.login(ftpusername, ftppassword)
ftp.quit(); print("FTP handling performed successfully")
```

Lecture

10

FTP CLIENT

Cont'd

Server Window Help					
Date/Time	Info		Type	Message	
Thu 07 Dec 2023 ...			Status	Settings written to /opt/filezilla-server/etc/users.xml.	
Thu 07 Dec 2023 ...			Status	Settings written to /opt/filezilla-server/etc/disallowed_ips.xml.	
Thu 07 Dec 2023 ...			Status	Settings written to /opt/filezilla-server/etc/settings.xml.	
Thu 07 Dec 2023 ...			Status	Settings written to /opt/filezilla-server/etc/allowed_ips.xml.	
Thu 07 Dec 2023 ...	FTP Session 2	127.0.0.1	Response	220-FileZilla Server 1.7.3	
Thu 07 Dec 2023 ...	FTP Session 2	127.0.0.1	Response	220 Please visit https://filezilla-project.org/	
Thu 07 Dec 2023 ...	FTP Session 2	127.0.0.1	Command	USER hamada	
Thu 07 Dec 2023 ...	FTP Session 2	127.0.0.1	Response	331 Please, specify the password.	
Thu 07 Dec 2023 ...	FTP Session 2	127.0.0.1	Command	PASS ****	
Thu 07 Dec 2023 ...	FTP Session 2	127.0.0.1	Response	230 Login successful.	
Thu 07 Dec 2023 ...	FTP Session 2	127.0.0.1	Command	QUIT	
Date/Time	Session ID	Protocol	Host	Username	Transfer
Connected to localhost					

Lecture

10

SECURED FTP CLIENT

Settings for server localhost:14148

Select a page:

- Server listeners
- Protocols settings
 - FTP and FTP over TLS (FTPS)
- Rights management
 - Groups
 - Users
- Administration
- Logging
- Let's Encrypt®
- Updates

Address	Port	Protocol
0.0.0.0	21	Require explicit FTP over TLS
::	21	Explicit FTP over TLS and insecure plain FTP

Add Remove

Apply Cancel OK

Lecture

10

SECURED FTP CLIENT

Cont'd

- Code:

```
import ftplib
ftphost = "localhost"
ftpport = 21
ftpusername = "yourUserName"
ftppassword = "youPassword"
ftp = ftplib.FTP_TLS()
ftp.connect(ftphost, ftpport)
ftp.login(ftpusername, ftppassword)
ftp.prot_p(); ftp.quit(); print("FTP handling performed successfully")
```

Lecture

10

FTP CLIENT

Cont'd

Administration interface - FileZilla Server 1.7.3					
Server Window Help					
Date/Time	Info	Type	Message		
Thu 07 Dec 2023 ...	FTP Session 3 127.0.0.1	Command	USER hamada		
Thu 07 Dec 2023 ...	FTP Session 3 127.0.0.1	Response	331 Please, specify the password.		
Thu 07 Dec 2023 ...	FTP Session 3 127.0.0.1	Command	PASS ****		
Thu 07 Dec 2023 ...	FTP Session 3 127.0.0...	Response	230 Login successful.		
Thu 07 Dec 2023 ...	FTP Session 3 127.0.0...	Command	PBSZ 0		
Thu 07 Dec 2023 ...	FTP Session 3 127.0.0...	Response	200 PBSZ=0		
Thu 07 Dec 2023 ...	FTP Session 3 127.0.0...	Command	PROT P		
Thu 07 Dec 2023 ...	FTP Session 3 127.0.0...	Response	200 Protection level set to P		
Thu 07 Dec 2023 ...	FTP Session 3 127.0.0...	Command	QUIT		
Thu 07 Dec 2023 ...	FTP Session 3 127.0.0...	Response	221 Goodbye		
Date/Time	Session ID	Protocol	Host	Username	Transfer
Connected to localhost					

Lecture

10

WORKING ON FTP SERVER

- Code:

```
import ftplib
ftphost = "localhost"; ftpport = 21
ftpusername = "yourUserName"
ftppassword = "youPassword"
ftp = ftplib.FTP()
ftp.connect(ftphost, ftpport)
ftp.login(ftpusername, ftppassword)
ftp.cwd('/your/configured/path') #ftplib.error_perm
```

Lecture

10

WORKING ON FTP SERVER

- Code:

```
print( ftp.pwd() )  
fileNames = ftp.nlst()  
for file in fileNames:  
    print(file)  
ftp.quit()
```

Lecture

10

WORKING ON FTP SERVER

Cont'd

- Code:

```
with open("message.txt", 'rb') as file:
    rescode = ftp.storbinary("STOR message.txt", file, blocksize= 1024*1024)
    if rescode.startswith("226"):
        print("File uploaded seccussefully")
    else:
        print("File not uploaded!")
ftp.quit()
```

Lecture

10

WORKING ON FTP SERVER

Cont'd

- Code:

```
with open("message.txt", 'wb') as file:
    rescode = ftp.retrbinary("RETR message.txt", file.write)

if rescode.startswith("226"):
    print("File downloaded seccussefully")
else:
    print("File not uploaded!")
ftp.quit()
```

- Code:

```
ftp.rename("oldFileName.extension", "newFileName.extension")
ftp.delete("fileName.extension")
ftp.mkd("folderName")
ftp.rmd("folderName")
print(ftp.size("fileName.extension"))
print(ftp.dir())
ftp.quit()
```

Lecture

10

QUESTIONS FOR ONLINE DISCUSSION

- Write a program to create your own FTP client to upload and download files to a specific location in your machine.

Lecture

10

PYTHON UPDATES in NETWORK PROGRAMMING

Lecture

10

SYNCHRONOUS vs. ASYNCHRONOUS

- Synchronous (The Old Way):
 - Code executes **line-by-line**. If one line involves a network request that takes 2 seconds, the entire program freezes for 2 seconds. This is called **Blocking**.
 - Blocking: Code executes line-by-line. The program stops and waits for I/O (like a network request or database query) to finish before moving to the next line.

Lecture

10

SYNCHRONOUS vs. ASYNCHRONOUS

- Asynchronous (The Modern Way):
 - The program initiates the 2-second request and puts a "bookmark" on it. It then continues executing other code. When the 2 seconds are up and data arrives, it jumps back to that bookmark to process the result. This is called Non-Blocking.
 - Non-Blocking: The program initiates a task (like a download) and immediately moves on to other work while waiting for the result in the background.

Lecture

10

THE EVENT LOOP

- **Asyncio:** The Event Loop is the central scheduler in Python's **asyncio** module. It manages and distributes the execution of different tasks.
- How it works:
 - It keeps a list of tasks that need to run.
 - It picks a task and runs it until that task says, "I have to wait for something (I/O)."
 - The loop then pauses that task and instantly switches to another task that is ready to run.
 - When the first task's wait is over (e.g., data arrived), the loop resumes it.

Lecture

10

SYNTAX REVOLUTION

- **async def:**
 - Defines a function as a "**Coroutine**."
 - Calling this function doesn't run it immediately; it returns a coroutine object that can be scheduled.
- **await:**
 - It tells Python: "Pause this function here, go do other work, and come back when this specific task is finished."
 - Example: `data = await download_file()`



Thank You

