

Network Programming in Python

Prepared by Dr. Hamada I. AbdulWakel

Revision Lecture



Basics of Computer Networking

Lecture 1

Prepared by
Dr. Hamada I. AbdulWakel

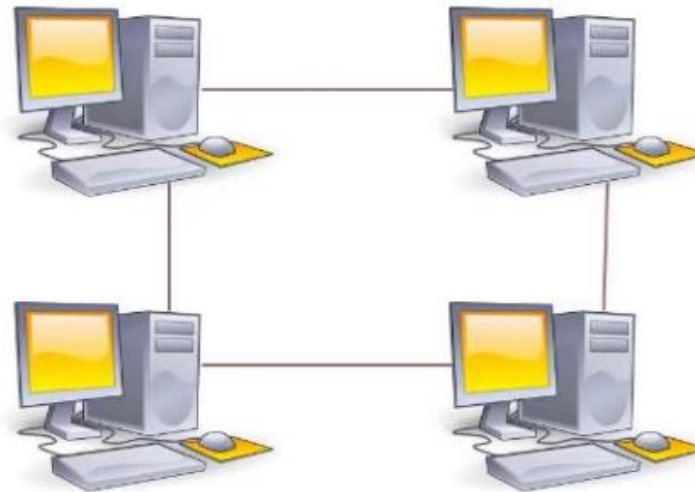
www.midocean.edu.km

Lecture

01

NETWORK DEFINITION

- A collection of devices connected to one another allowing for data to be shared and used.
 - Devices count.
 - At least 2 devices.
 - Device type:
 - Any device.
 - Connection type:
 - Wired.
 - Wireless.



Lecture

01

IMPORTANCE OF NETWORKS

Cont'd

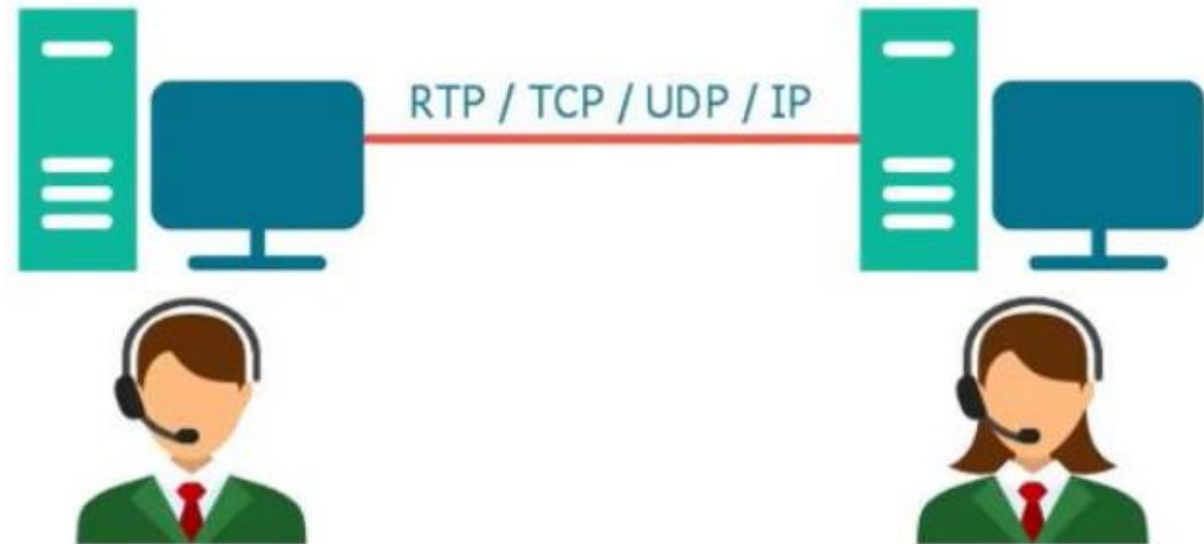
- Share resources such as Data and Hardware.
- Save a lot of money.
- Increase workgroups collaborations.
- Increase productivity.
- Protect company data.

Lecture

01

NETWORK COMPONENTS

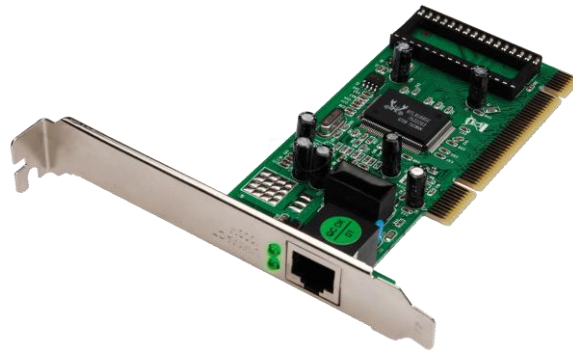
- Any network must have:
 - Computer systems.
 - Network media.
 - Cables:
 - Electrical signals.
 - Pulses of light.
 - Wireless:
 - Radio frequencies.



NETWORK COMPONENTS

Cont'd

- Any network must have:
 - Network interface.
 - A point of interconnection between a computer and a network.
 - Physical form or implemented in software.
 - Convert data to suitable formats for media used.



NETWORK COMPONENTS

Cont'd

- Any network must have:
 - Network protocol.
 - **A protocol** is a standard rules for how the connected devices are communicate and exchange information.
 - IP: a set of standards for addressing and routing data on the Internet.

106.241.27.244



133.83.189.60



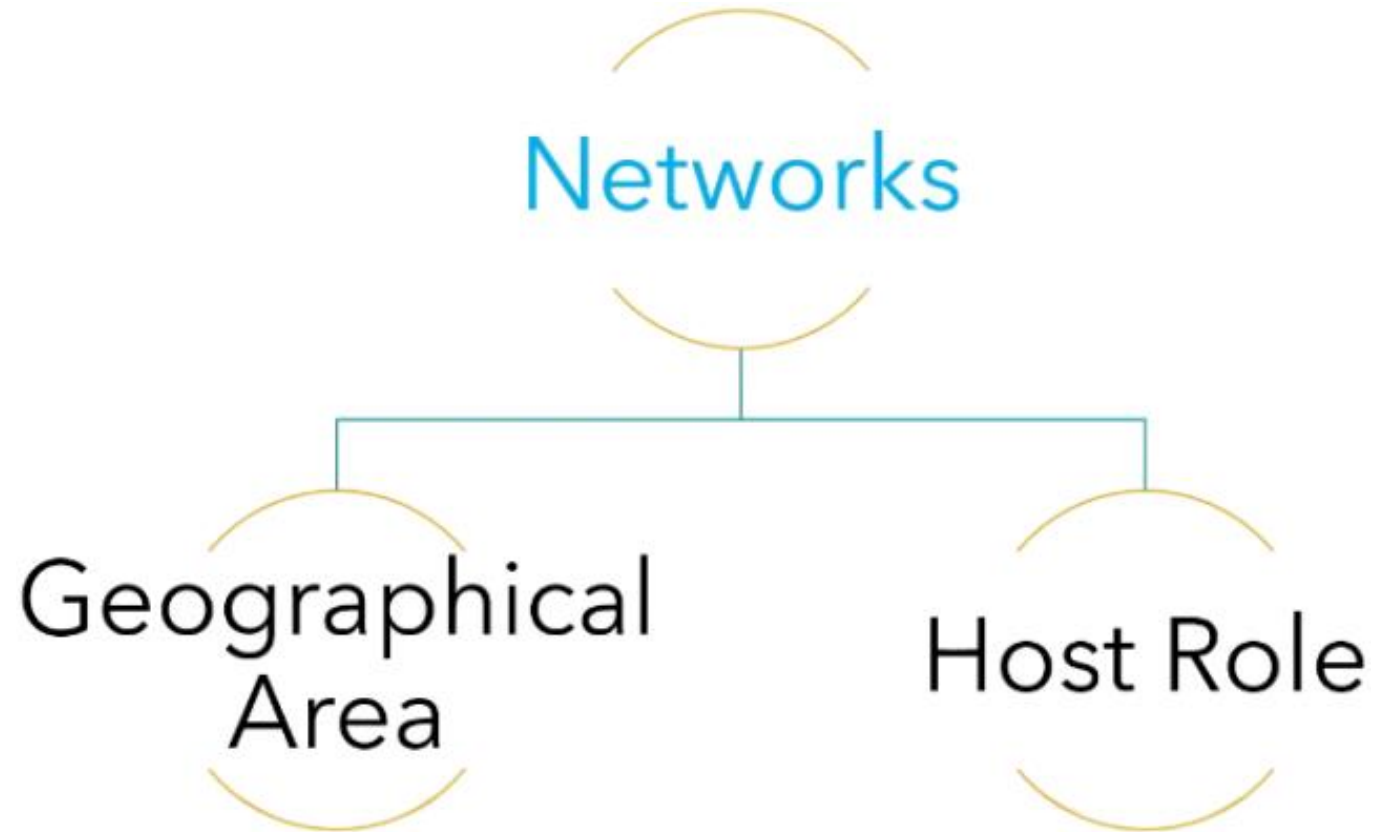
240.101.163.19



Lecture

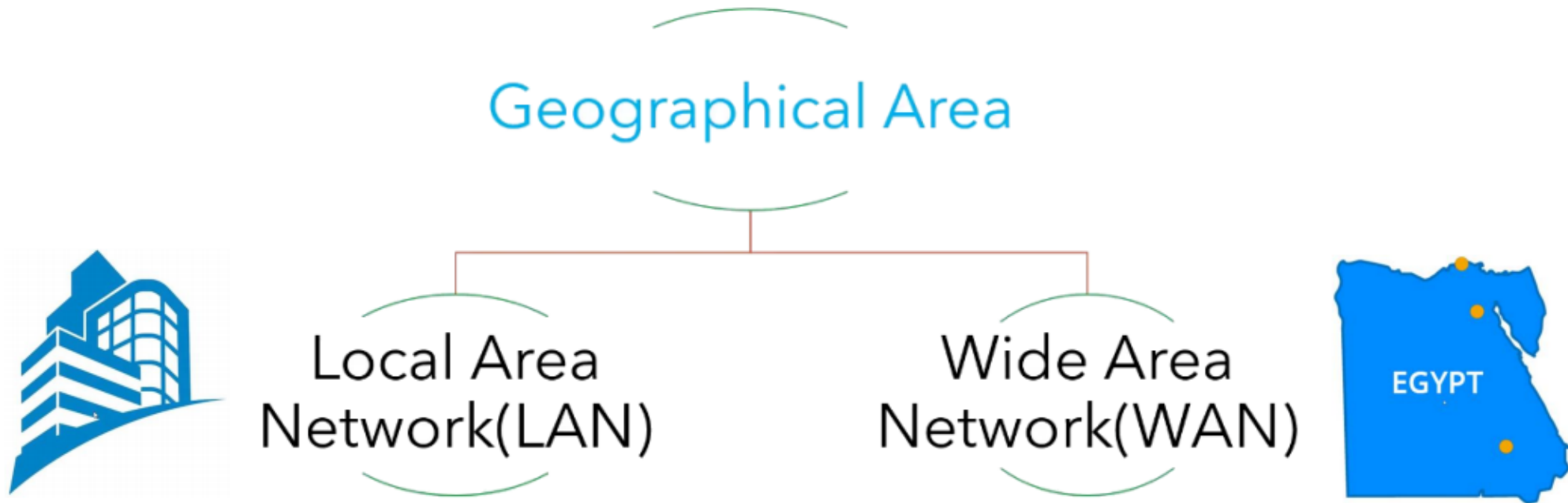
01

NETWORK TYPES



NETWORK TYPES

Cont'd

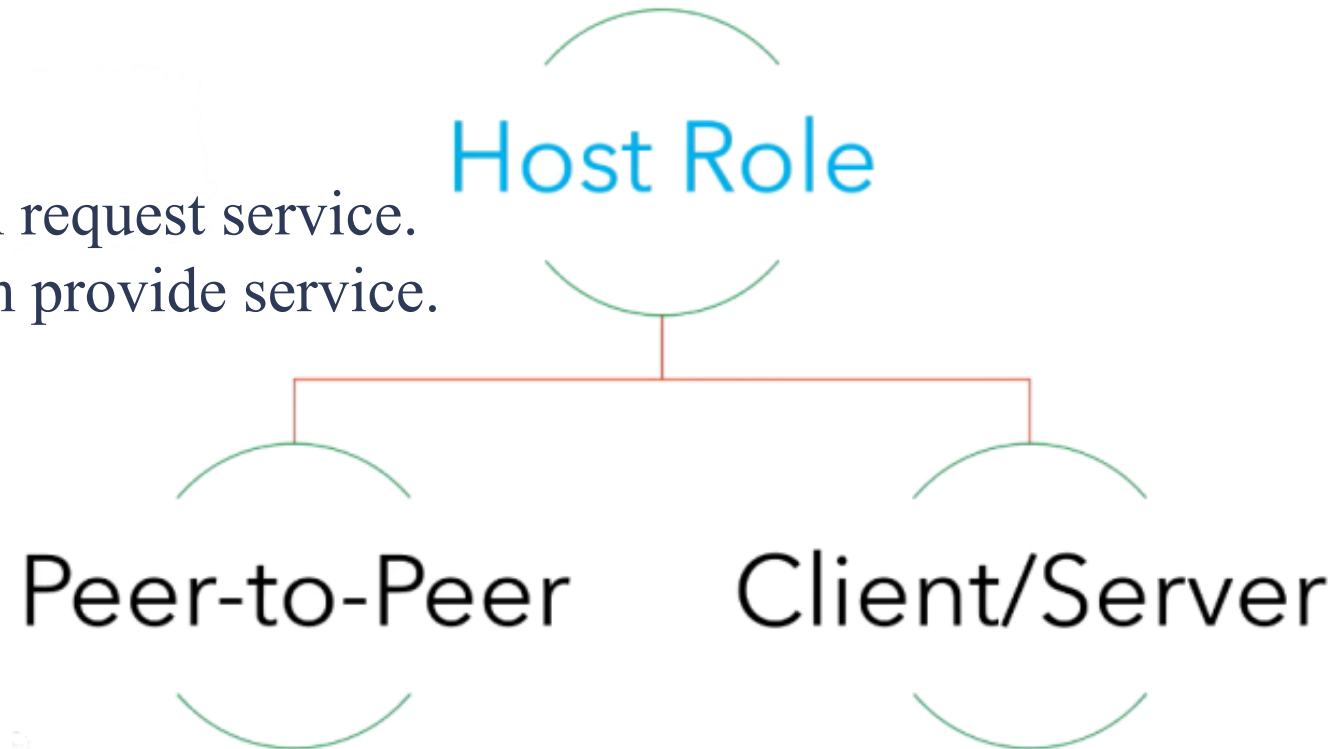


NETWORK TYPES

Cont'd

Host Roles:

- 1- Client which request service.
- 2- Server which provide service.



Lecture

01

NETWORK TYPES

Cont'd

- The peer-to-peer network features:
 - Simple network forms.
 - Less costs to forming.
 - Roles could be switched.
 - No centralized administration.
 - Security is its main problem.

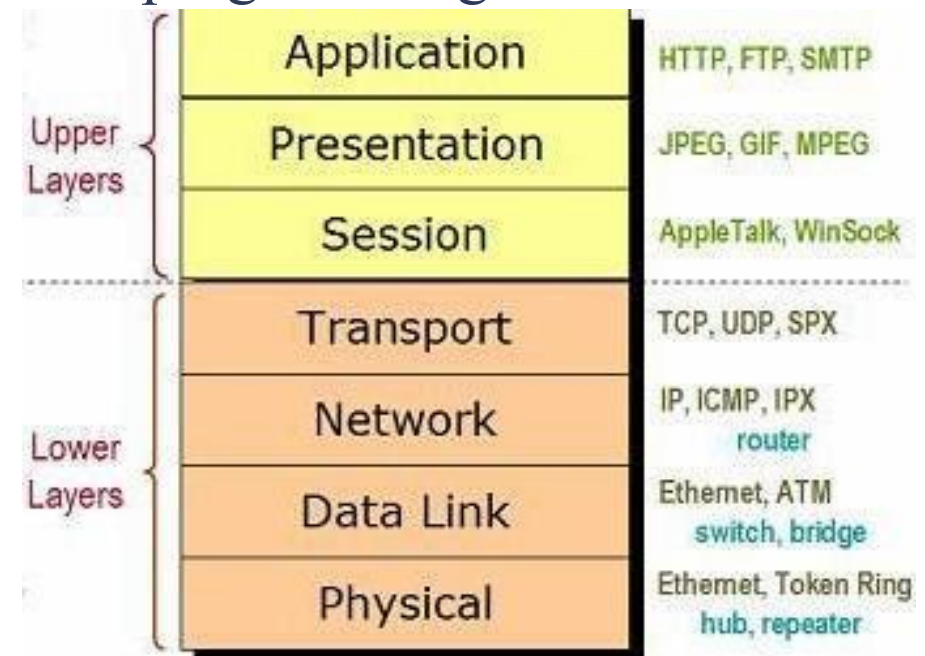
NETWORK TYPES

Cont'd

- The client/server network features:
 - Scalable network forms.
 - The server costs is high and requires experience to manage.
 - Roles do not switch.
 - The server considers the centralized administration.
 - Security maintained by the server.

NETWORK PROGRAMMING LEVELS

- Python provides two levels of access to network programming:
 - **Low-level access.**
 - **High-level access.**



NETWORK PROGRAMMING LEVELS

Cont'd

- Python provides two levels of access to network programming:
 - **Low-level access:**
 - Access the basic `socket` support for the operating system using Python's libraries.
 - Implement client and server for both connection-oriented and connectionless protocols.
 - **High-level access:**
 - Implement protocols like HTTP, FTP, etc. using Python's libraries.

Lecture

01

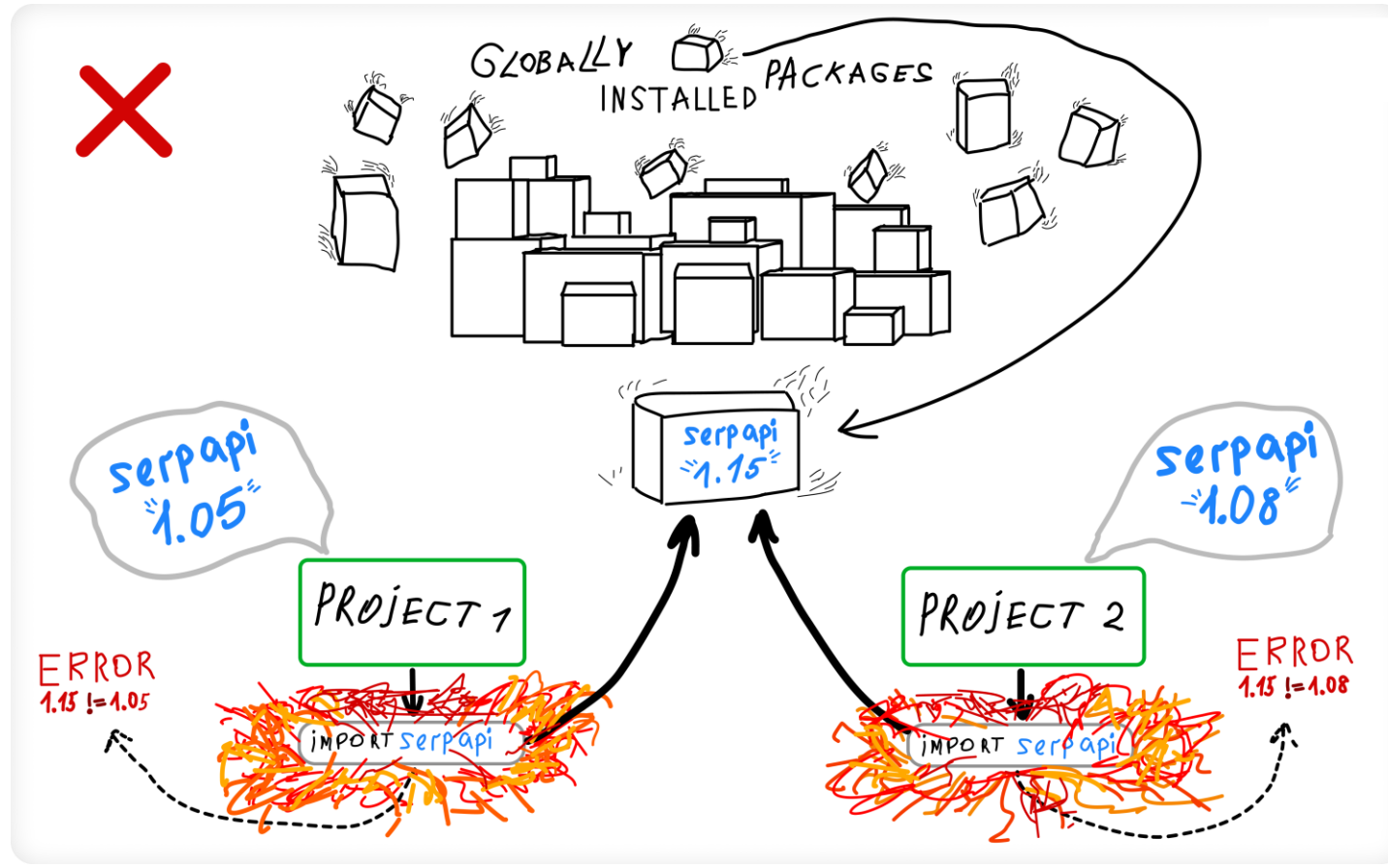
REQUIREMENTS FOR PYTHON NETWORK PROGRAMMING

- Python basics and OOP.
- PyPi [website](#) to find, install Python packages.
- Python package installer (pip).
 - Linux OS:
`sudo apt install python3-pip`
 - Windows OS: can be installed from the pip official [website](#).
- Integrated Development Environment such as [VScode](#) or [Pycharm](#).

Lecture

01

VIRTUAL ENVIRONMENTS



VIRTUAL ENVIRONMENTS

Cont'd

- **Virtual environment (Venv)** is a tool that helps to keep dependencies required by different projects separate by creating **isolated** python virtual environments for them.
- **Isolation** means any changes on dependencies installed in a virtual environment don't affect the dependencies of the other virtual environments or the system-wide (default python) libraries.
- We can create **multiple** virtual environments with different/same Python versions and different/same libraries versions on the same machine.

A collage of five vertical panels with rounded ends. The first panel shows hands typing on a laptop. The second panel shows a graduation cap icon. The third panel shows a play button icon. The fourth panel shows a target icon and a magnifying glass icon. The fifth panel shows a book icon and a magnifying glass icon.

Basics of Network Programming – Part I

Lecture 2

Prepared by
Dr. Hamada I. AbdulWakel

SERVICE-ORIENTED ARCHITECTURE (SOA)

Cont'd

- **SOA** is a method of software development that uses software components called services to create business applications.
- Each **service** provides a business capability, and services can also communicate with each other across platforms and languages.
- Developers use SOA to **reuse services** in different systems or combine several independent services to perform complex tasks.

COMPONENTS OF SOA

- **Services:** the basic building blocks of SOA.
 - They can be private — available only to internal users of an organization
 - Or public — accessible over the internet to all.
- **Heterogeneous application** which developed using different platform cannot communicate with each other because they did not know the language of each other.
- **Web services** are language independent and platform independent. Web services are mainly used for cross communication between heterogeneous technologies.

- **Service provider:** creates, maintains, and provides one or more services that others can use.
- **Service consumer:** requests the service provider to run a specific service. It can be an entire system, application, or other service.
- **The service contract:** defines the nature of the service and its associated terms and conditions, such as **the prerequisites** for using the service, service cost, and quality of service provided.

Lecture

02

APPLICATION PROGRAMMING INTERFACE (API)

- **API** could be a **software system interface** that enables two applications to act with one another with **no user intervention**.
 - APIs offer a **secure and standardized** means for applications to communicate.
- It defines:
 - **URL** style to make requests.
 - The **required data** for the service.
 - The **data formats** that will be received.
- All web services are APIs, but not all APIs are web services (e.g., when Photoshop needs to **save as** the image).

Lecture

02

WIRE PROTOCOL

- A **protocol** is a set of rules that all parties follow, so that we can predict each other's behavior.
- Whenever you transmit information over a network, that information is transformed from a binary representation into another representation that was designed for network use.
- The form that it takes for network transmission is called its **wire representation**.

- We need to **unify the format** between programs so that program can expect this format and thus extract the information from it easily.
- A **wire protocol** refers to a way of getting data from point to point.
- There are two common wire representations:
 - eXtensible Markup Language (XML) format.
 - JavaScript Object Notation (JSON) format.

Lecture

02

XML WIRE PROTOCOL

Python program
uses a dictionary
data structure

```
<car>  
  <type>  
    Mercedes  
  </type>  
  <color>  
    Black  
  </color>  
</car>
```

PHP program
uses an array
data structure

Lecture

02

JSON WIRE PROTOCOL

Python program
uses a dictionary
data structure

```
{  
  "type": "Mercedes",  
  "color": "Black"  
}
```

PHP program
uses an array
data structure

Lecture

02

JSON IN PYTHON

- Python has a built-in package/module called `json`, which can be used to work with JSON data.
- The `json.loads()` method is used to **parse** a JSON string and the result is a Python dictionary.
- The `json.dumps()` method is used to **convert** a Python dictionary to a JSON string.

Lecture

02

WEB PAGE ACCESSING AND
READING

<https://en.wikipedia.org/wiki/URL?key=value&life=42#History>

- Web pages are full of original, high-quality content that could be used for different reasons.
- URL parts:
 - Protocol part, such as **https**.
 - Host name part, such as **en.wikipedia.org**.
 - Port number part, getting from the protocol.
 - Path part, such as **wiki/URL**.
 - Querystring, such as **key and life**.
 - Fragment, such as **#History**.

Lecture

02

URLLIB MODULE

- **Urllib** is a Python package that allows you to access and interact with web pages using their URL's (Uniform Resource Locator).
- This facility of urllib provides the flexibility to run GET, POST, and PUT methods and mock the JSON data in the Python code.
- Urllib module is not a built-in module, you need to install it manually.

`pip install urllib3`

URLLIB MODULE COMPONENTS

Cont'd

- **Urllib.request**
 - Used for opening and reading web pages. The main method is `urlopen()` to open the specified URL.
- **Urllib.parse**
 - Used for parsing the parts of the URLs.
- **Urllib.error**
 - Used for catching exceptions encountered when opening URLs and other errors.

Lecture

02

URLLIB DESCRIPTION

- Code:

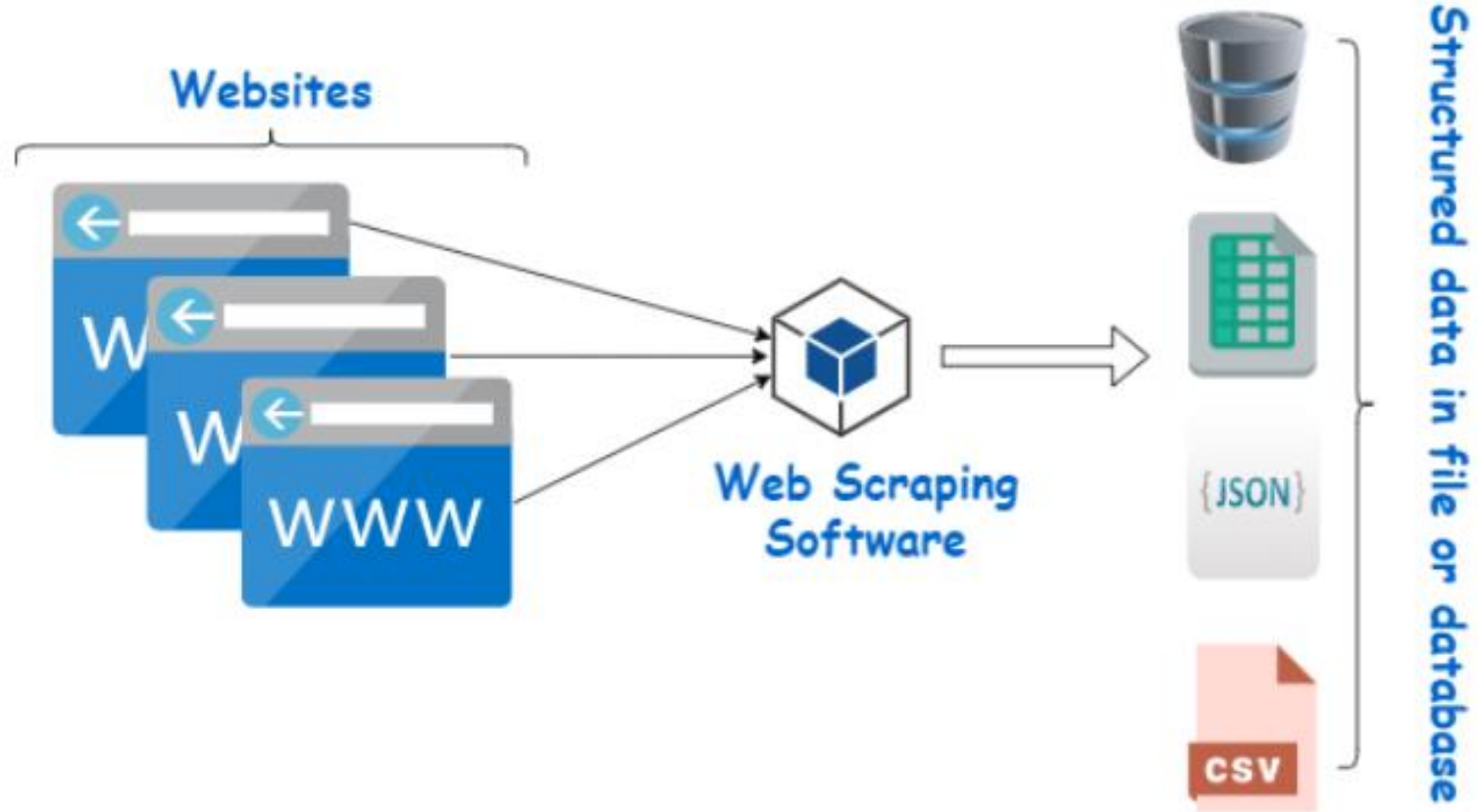
```
import urllib
from urllib import request
```

```
help(urllib)
print(dir(request))
resp = request.urlopen('https://www.wikipedia.org')
print(type(resp))
print(dir(resp))
```

Lecture

02

WEB SCRAPING



- A **web scraping** software will automatically crawl and extract data from multiple pages of websites based on your requirement.
 - Usually, the web pages are described as unstructured data.
- Web Scraping is an automatic way to retrieve unstructured data from a website and store them in a structured format.
 - By automating this process, these software can extract huge amounts of data in a very short time.

- **A web scraper** works as following:
 - Step 1: Making an HTTP request to a server.
 - Step 2: Extracting and parsing (or breaking down) the website's code.
 - Step 3: Saving the relevant data locally

The background features a collage of educational icons overlaid on a blurred image of hands typing on a laptop. The icons include a graduation cap, a target with an arrow, an open book, a play button, a magnifying glass, and a group of people. The icons are semi-transparent and arranged in a vertical sequence.

Basics of Network Programming – Part II

Lecture 3

Prepared by
Dr. Hamada I. AbdulWakel

GOOGLE MAPS API

- **Geocoding:** the process of converting human-readable addresses into geographic coordinates (i.e., latitude and longitude), which you can use to place markers on a map.
- **Reverse geocoding:** the process of converting geographic coordinates into a human-readable address.
- Map APIs allow you to create location-aware infographics, mind maps and visualization aids.

Lecture

03

WEB SCRAPING

- When a program pretends to be a web browser and retrieves web pages, extracts information, and then looks at more pages.
- Uses:
 - Monitor a site for new information.
 - Make a database for a search engine.
 - Get your data from a system that has no export feature.
 - Pull social data.

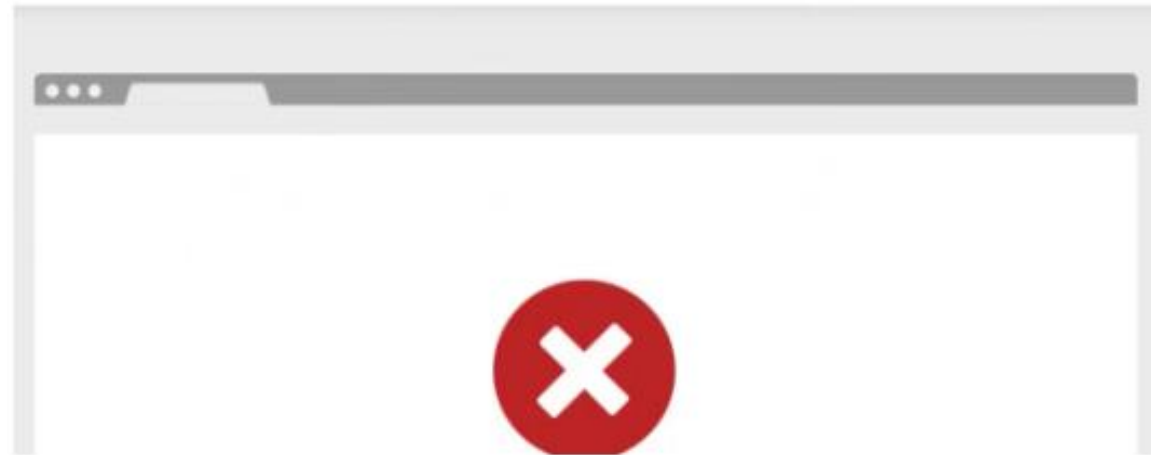
Lecture

03

BE CAREFUL WITH WEB SCRAPING

- There is some controversy about web scraping.
- Do not republishing copyrighted material.
- Do not violate terms of service.

Sorry, you have been blocked
You are unable to access youm7.com



BEAUTIFULSOUP MODULE

Cont'd

- **BeautifulSoup** is a Python module for pulling data out of HTML and XML files.
- One of the important pieces of HTML document are **tags**, which may contain other tags/strings (tag's children).
- BeautifulSoup provides different ways to navigate and iterate over these tags.
- BeautifulSoup is the object created when we try to **scrape a web resource**.

Lecture

03

INSTALLATION OF BEAUTIFULSOUP MODULE

- Installation:
 - PyPi: `pip install beautifulsoup4`
- Beautiful Soup supports different HTML parser such as:
 - **Html.parser**: the default HTML parser included in Python's standard library.
 - lxml
 - `pip install lxml`
 - Html5lib
 - `pip install html5lib`

WEB SCRAPING STEPS

Cont'd

- **Step 1: Inspect your data source:**
 - A manual step.
 - Open the website you want to pull important content from it.
 - Open inspection.
 - On Mac:
`Cmd+Alt+I`
 - On Windows/Linux:
`Ctrl+Shift+I`
 - Understand the website structure to know where these important data.

WEB SCRAPING STEPS

Cont'd

- **Step 2: Scrape HTML content from a page:**
 - Create the soup object.
 - Read the web page content using this object.
- **Step 3: Parse HTML code with BeautifulSoup:**
 - Parse the web page content to make it more accessible.
 - Pick out the data from the required HTML tags.

Lecture

03

BEAUTIFULSOUP MODULE METHODS

- **Beautifulsoup(content, parser):** creates a soup object.
- **soupObj.prettify():** gives the visual representation of the parsed URL. It arranges all the tags in a parse-tree manner with better readability.
- **soupObj.title:** gives the title tag with its content.
- **soupObj.title.string:** gives the title content only without the tag.

BEAUTIFULSOUP MODULE METHODS

Cont'd

- `soupObj.find("tagName")`: gives the first occurrence of the given tag name.
- `soupObj.find_all("tagName")`: gives a list of all occurrences of the given tag name with their contents.
 - `firstTag = soupObj.find_all("tagName")[0]`
 - If the firstTag contains other tags:
`firstTag.find_all("otherTagName")`

BEAUTIFULSOUP MODULE METHODS

Cont'd

- **soupObj.find_all("tagName", { css style })**: gives a list of all occurrences of the given tag name with they contents which have the same given css style.
- **tag.get("propertyName")**: gives the value of the given property for the selected tag.
- **tag.text**: gives the content of the selected tag.

Lecture

03

SOCKETS

- **Sockets** are the endpoints of a bidirectional communications channel between connection devices.
- Sockets are the most popular form of inter-process communication especially for cross-platform communication.
- Sockets may communicate within a process, between processes on the same machine, or between processes on different continents.

Lecture

03

THE SOCKET MODULE

- Python's standard library consists of various built-in modules that support inter-process communication and networking.
- The **socket** module defines how server and client machines can communicate at hardware level using socket endpoints on top of the operating system.
- The socket module provides various objects, constants, functions and related exceptions for building full-fledged network applications including client and server programs.

THE SOCKET MODULE

Cont'd

- To program a connection between two end-points, you need to configure both end-points sockets to know each other.
- The socket configuration includes:
 - Socket family: represents the IP version used.
 - **AF_INET** for IPv4
 - **AF_UNIX** for IPv6
 - Socket type: represents the type of connection.
 - **SOCK_STREAM** for connection-oriented type (TCP).
 - **SOCK_DGRAM** for connectionless type (UDP).

Lecture

03

THE SOCKET MODULE METHODS

- **socket()**: creates sockets objects for the server and the client.
- **gethostname()**: returns a string containing the hostname.
- **gethostbyname()**: returns IP address.
- **bind()**: binds/associates the address to the created socket object.

THE SOCKET MODULE METHODS

Cont'd

- **connect():** takes a two-item tuple object as argument. The two items are IP address and port number of the server.
- **listen():** starts the server and runs it into a listen loop looking for connection request from client.
- **accept():** when a connection request is intercepted by server, this method accepts it and identifies the client socket with its address.
- **close():** marks the socket as closed.

Lecture

03

THE SOCKET CREATION

- Code:

```
import socket
```

```
# create a TCP socket
```

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

```
print("Socket successfully created!")
```

Lecture

03

THE SOCKET CREATION WITH EXCEPTION HANDLING

- Code:

```
import socket
target_host = input("Enter the target host name to connect: ")
target_port = input("Enter the target port: ")
try:
    sock.connect((target_host, int(target_port)))
    print("Socket Connected to", target_host, "on port:", target_port)
except socket.error as err:
    print("Failed to connect to", target_host, "on port", target_port)
    print("Reason:", str(err) )
```

Lecture

03

SIMPLE SERVER USING SOCKETS

- Code:

```
import socket
```

```
soc = socket.socket()  
print("socket created")
```

```
# the server will be in the local machine  
soc.bind((socket.gethostname(), 5485))  
soc.listen(3) #accepts three connection  
print("Waiting for connections .....")
```

Socket Programming Using Python – Part I

Lecture 4

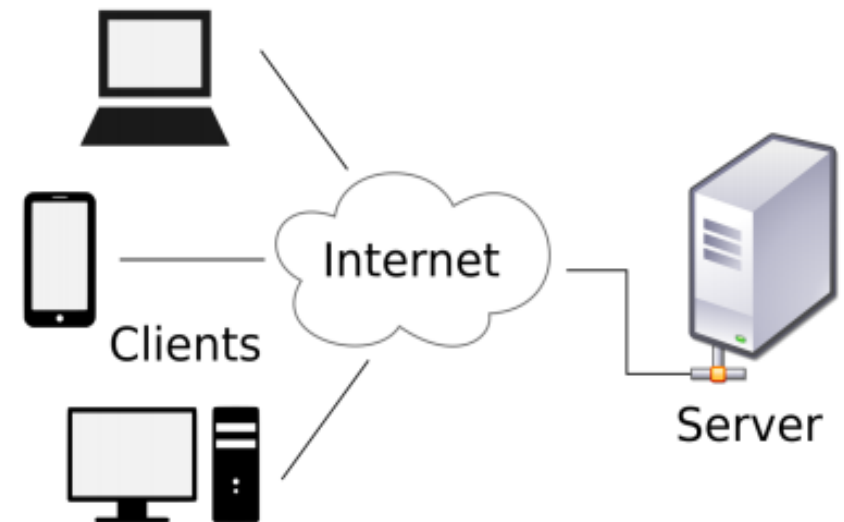
Prepared by
Dr. Hamada I. AbdulWakel

Lecture

04

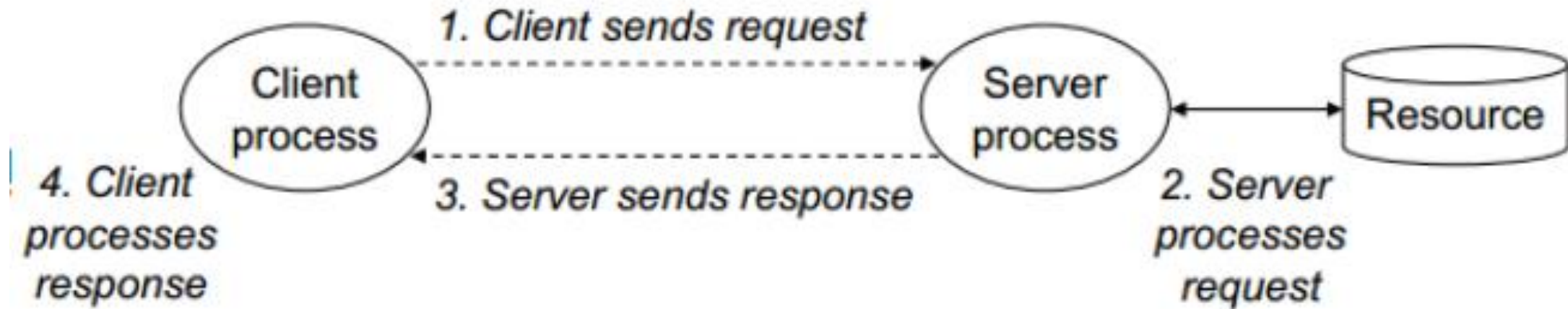
THE CLIENT/SERVER PROGRAMMING MODEL

- Every network application is based on the client-server model.
- An application consists of a server process and one or more client processes.
- A server manages some resource, and it provides some service for its clients by manipulating that resource.



THE CLIENT/SERVER PROGRAMMING MODEL

Cont'd



Lecture

04

THE COOMUNICATION PROTOCOLS

- Any computer usually has:
 - A single physical connection to the network.
 - Data could be sent by different applications through the use of ports configured on the same physical network connection.
- A computer's address is a 32-bit IP address.
- The port number is a 16-bit number ranging from 0 to 65,535, with ports 0-1023 restricted by well-known applications like HTTP and FTP.

THE COMMUNICATION PROTOCOLS

Cont'd

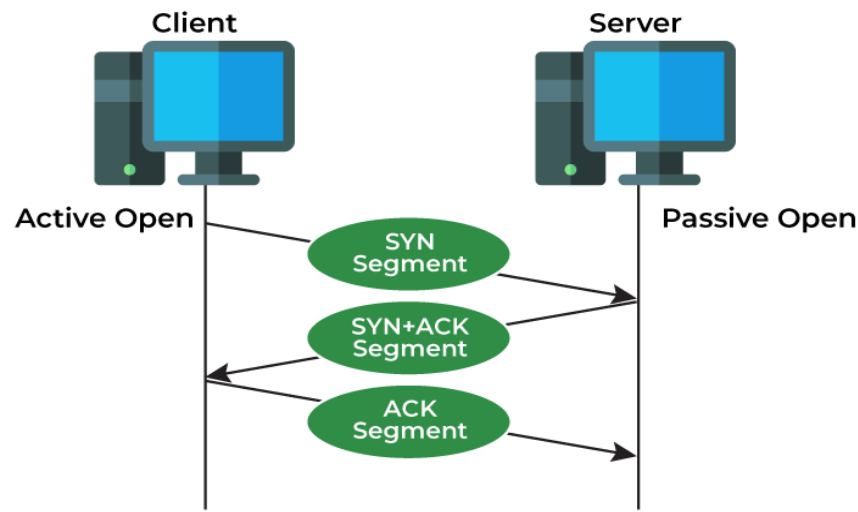
- Every packet of data has:
 - Source IP, Source Port, Destination IP, and Destination Port.
 - Allow computing devices and applications to send data via a network and verify its delivery.
- There are two types of client-server application communication protocol:
 - Transmission Control Protocol (TCP).
 - User Datagram Protocol (UDP).

Lecture

04

THE TRANSMISSION CONTROL PROTOCOL (TCP)

- A connection-oriented protocol that provides a reliable flow of data between end-to-end connection.
 - Three-way handshake. "Wanna connect?" "Sure!" "Alright then!".



Lecture

04

THE TRANSMISSION CONTROL PROTOCOL (TCP)

Cont'd

- TCP features:
 - Slow overhead time of setting up.
 - Retransmission.
 - Bigger overhead.
 - In-order delivery.
 - Congestion control.
 - Delays transmission when network is congested.
 - Data doesn't always get sent out immediately (i.e., voice chat).

THE USER DATAGRAM PROTOCOL (UDP)

- A Connection-less protocol that sends independent packets of data (datagrams), from one computer to another.
 - No connection to create and maintain.



THE USER DATAGRAM PROTOCOL (UDP)

Cont'd

- UDP features:
 - Faster since no overhead of setting up a connection.
 - Small packet sizes.
 - No compensation for lost packets.
 - Packets can arrive out of order.
 - No error recovery.
 - No congestion control.

Lecture

04

THE TCP AND UDP IN SOCKET MODULE

- Both protocols (TCP and UDP) are supported by socket module.
 - A socket is a file-like object.
 - Allows data to be sent and received across the network like a file.
 - But it also includes functions to accept and establish connections.
- Before two machines can establish a connection, both must create a socket object.

SOME SOCKET METHODS

- **gethostname():** returns the host name of the current system under which the Python interpreter is executed.

```
print( socket.gethostname() )
```

- **gethostbyname():** returns the IP address of the host.

```
print( socket.gethostbyname("www.github.com") )
```

SOME SOCKET METHODS

Cont'd

- **gethostbyaddr():** returns a tuple containing (Host Name, Alias list for the IP address if any, IP address of the host)

```
ipAddress = "192.0.43.8"  
hostName = socket.gethostbyaddr(ipAddress)  
print( hostName )
```
- **getprotobyname():** takes a protocol string like TCP, UDP, or FTP and returns the associated constant for the protocol as defined by the socket module.

```
print( socket.getprotobyname("TCP") )
```

SOME SOCKET METHODS

Cont'd

- **getservbyname():** takes a network service name and an underlying protocol name and returns the port number on which the service is defined.
`print(socket.getservbyname("https", "tcp"))`
- **getservbyport():** returns the name of the service for a given port number.
`print(socket.getservbyport(21, "tcp"))`

SOME SOCKET METHODS

Cont'd

- **recv(bufsize):** receives the TCP message from socket. The argument **bufsize** stands for buffer size and defines the maximum data.
 - You should use the **decode()** method when receiving data from the socket to convert it to a string.
- **send(bytes):** sends data to the socket which is connected to the remote machine. The argument **bytes** will give the bytes sent to the socket.
 - You should use the **encode()** method to convert the string to bytes.

Lecture

04

A SIMPLE CLIENT/SERVER APPLICATION USING TCP

- Code: 1- the server

```
import socket
soc = socket.socket()
print("socket created")
soc.bind( (socket.gethostname(), 5485) )
soc.listen()
print("Waiting for connections .....")
```

A SIMPLE CLIENT/SERVER APPLICATION USING TCP

Cont'd

- Code: 1- the server
 while True:
 #return client socket and address
 clientSoc, clientAddre = soc.accept()
 name = clientSoc.recv(1024).decode()
 print("Connected with", clientAddre, name)
 clientSoc.send(("Welcome we are connected :) " + name).encode())
 clientSoc.close()

A SIMPLE CLIENT/SERVER APPLICATION USING TCP

Cont'd

- Code: 2- the client

```
import socket
soc = socket.socket()
# the address is the same as in the server
soc.connect( (socket.gethostname(), 5485) )
name = input("Enter your name? ")
soc.send(name.encode())
print(soc.recv(1024).decode())
soc.close()
```

SOME SOCKET METHODS

- **recvfrom(bufsize):** receives the maximum data size, bufsize. Two pair (data, address) value is returned by this method.
 - Data defines the received data and address specifies the address of socket sending the data..
 - You should use the `decode()` method when receiving data from the socket to convert it to a string.
- **sendto(message, address):** sends message as bytes to the socket specified with the given address.
 - You should use the `encode()` method to convert the string to bytes.

Lecture

04

A SIMPLE CLIENT/SERVER
APPLICATION USING UDP

- Code: 1- the server

```
import socket
localIP = "127.0.0.1"
localPort = 20001
UDPsoc = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
UDPsoc.bind((localIP, localPort))
print("UDP server up and listening")
```

Socket Programming Using Python – Part II

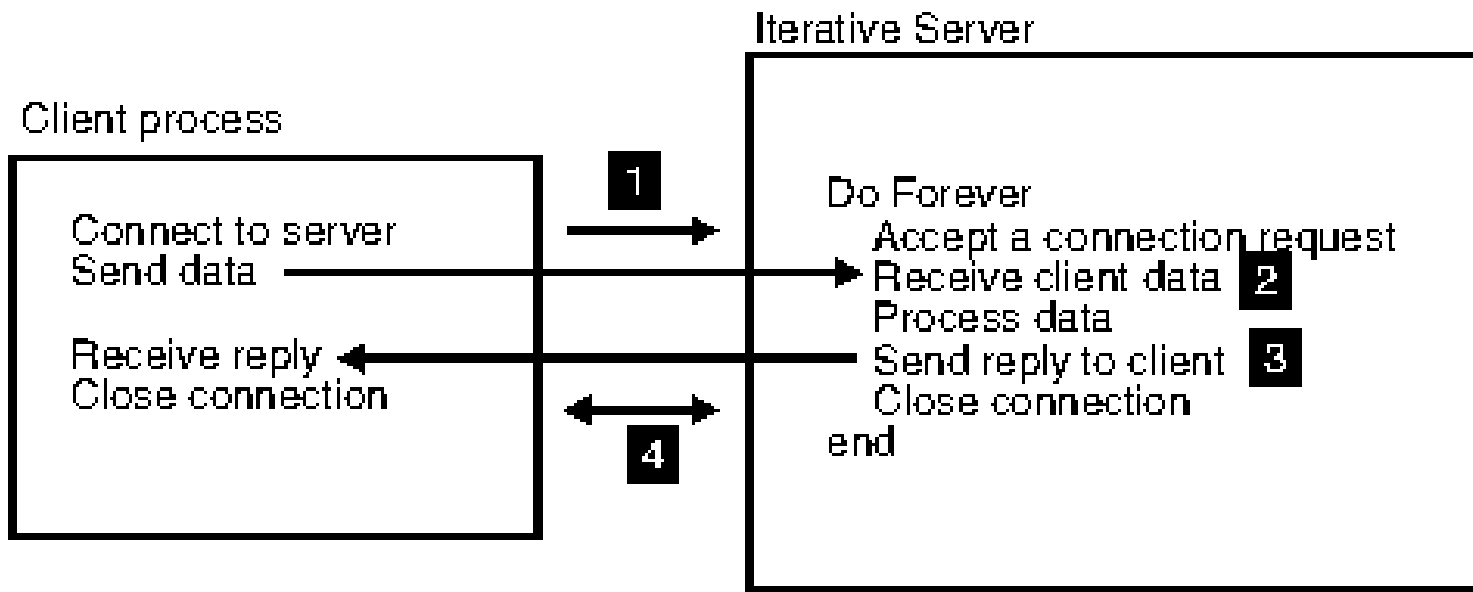
Lecture 5

Prepared by
Dr. Hamada I. AbdulWakel

Lecture

05

THE ITERATIVE SERVER



THE ITERATIVE SERVER

Cont'd

- **An iterative server** iterates through each client, handling it one at a time.
- The following list describes the iterative server socket process.
 - When a connection request arrives, it accepts the connection, receives the client data and does whatever has to be done to build a reply.
 - The server sends the data back to the client.
 - The server closes the socket and waits for the next connection request from the network.

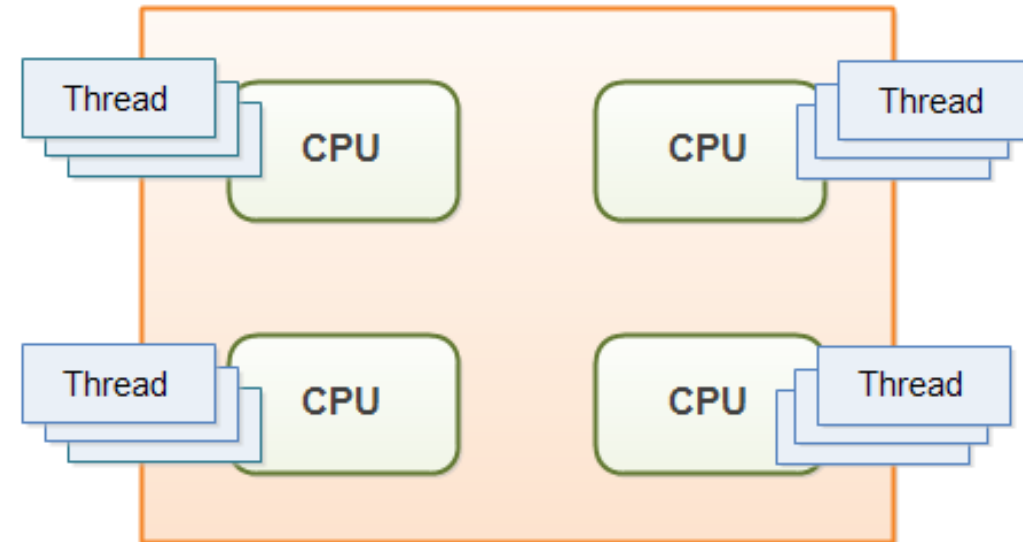
SOCKETS AND CONCURRENCY

Cont'd

- **A concurrent server** accepts a client connection, delegates the connection to a child process of some kind, and immediately signals its availability to receive the next client connection.
- The following list describes the concurrent server process:
 - When a connection request arrives, it schedules a child process and forwards the connection to the child process.
 - The child process takes the connection from the main process.
 - The child process receives the client request and processes it.
 - The connection is closed, and the child process signals to the main process that it is available for a new connection.

THE THREADING PROGRAMMING

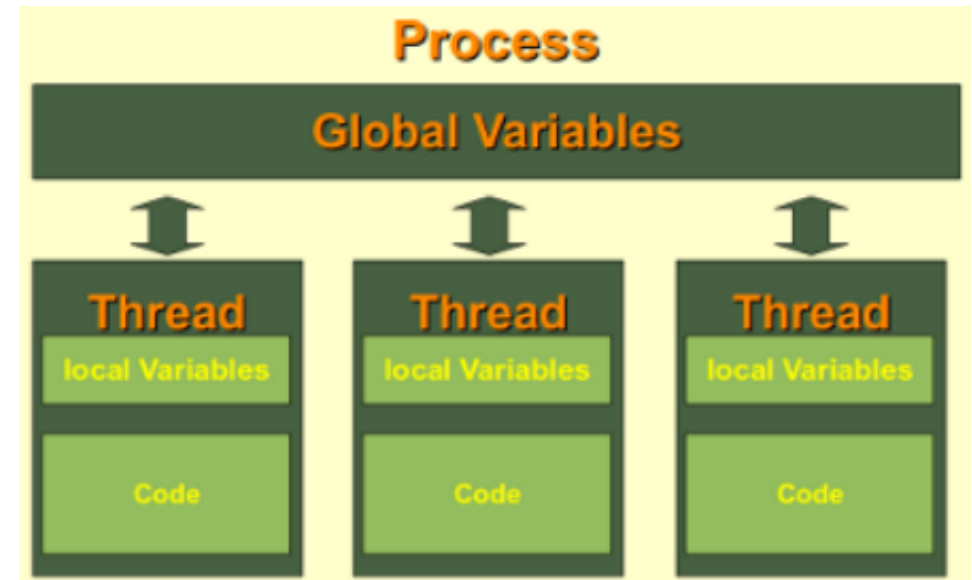
- A **thread** is defined in computer science as **the smallest unit** that can be scheduled in an operating system.
- The ability of a process to execute multiple threads parallelly is called **Multithreading**.



THE THREADING PROGRAMMING

Cont'd

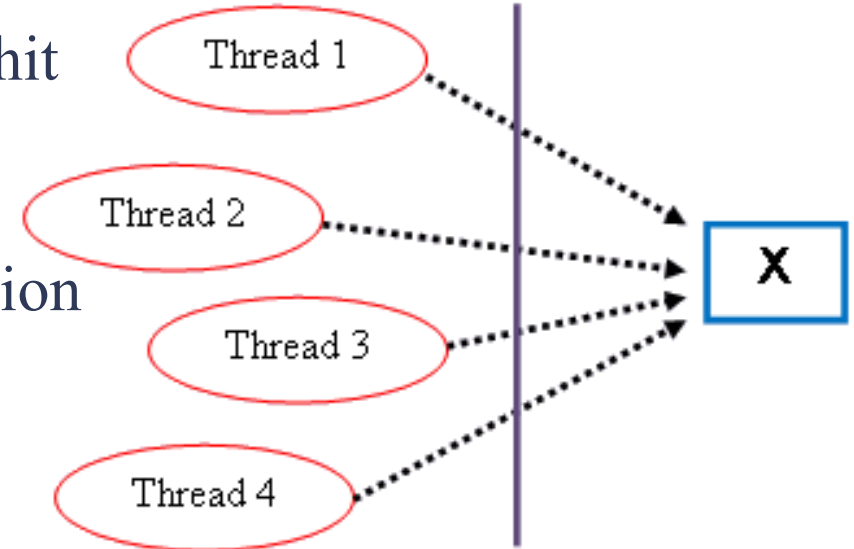
- Advantages:
 - Significantly improve the performance of any program.
 - A program can remain responsive while one thread waits for input.
 - All the threads of a process have access to its global variables.



THE THREADING PROGRAMMING

Cont'd

- Disadvantages:
 - On 1-CPU system, multithreading would not hit the speed of computation.
 - Synchronization for preventing mutual exclusion of shared resources.
 - Increases the complexity of the program.
 - Thread safe problem.



Lecture

05

THREADING IN PYTHON USING FUNCTIONS

- A function is a set of instructions that could be performed n times according to function calling in the code.

```
def myFun1(a, b):  
    c = a + b  
    print(c)
```

```
myFun1(1, 2)
```

THREADING IN PYTHON USING FUNCTIONS

Cont'd

- Performing/calling a function using threads using the threading module in Python.

```
import threading
def myFun1(a, b):
    c = a + b
    print(c)
thread1 = threading.Thread(target = myFun1, args = (12, 10))
thread1.start()
```

THREADING IN PYTHON USING FUNCTIONS

Cont'd

- Code:

```
import threading, time
def myFun1(a, b):
    time.sleep(1) #seconds
    c = a + b
    print(c)
def myFun2(a):
    c = a * 2
    print(c)
```

Lecture

05

CASTING

- Many applications transmit the same data at one time to multiple receivers:
 - Broadcasts of Radio or Video.
 - Video conferencing.
 - Shared applications.
- **The cast** term signifies some data (stream of packets) is being transmitted to the client(s) from a server over the communication channel.

Lecture

05

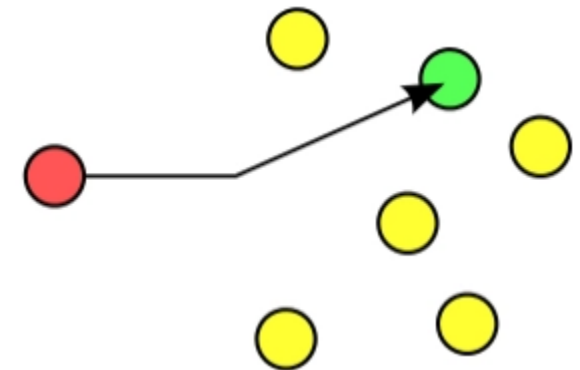
CASTING/COMMUNICATION TYPES

- There are three casting types:
 - Unicasting.
 - Multicasting.
 - Broadcasting.

CASTING/COMMUNICATION TYPES

Cont'd

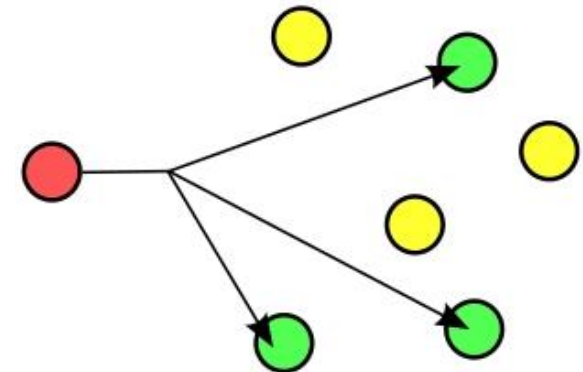
- **Unicasting.**
 - A type of information transfer and it is used when there is a participation of single sender and single recipient.
 - Represents the one-to-one communication.
 - Example:
 - Browsing a website.
 - Downloading a file from a FTP server



CASTING/COMMUNICATION TYPES

Cont'd

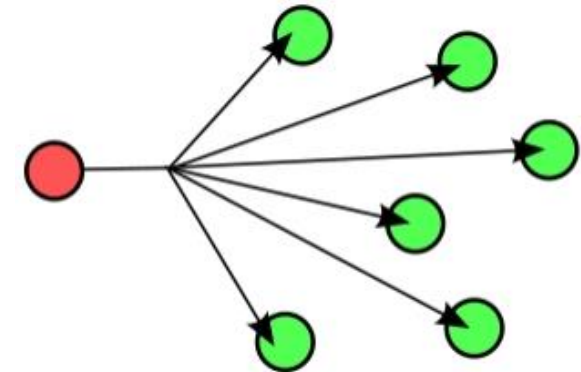
- **Multicasting.**
 - A type of information transfer and it is used when there is a participation of single sender and multiple recipient.
 - Represents the **one-to-many** communication.
 - The multiple recipient must join to that a **multicast group** to receive data.
 - Only supported by UDP.
 - Example:
 - Sending an email to a mailing list.



CASTING/COMMUNICATION TYPES

Cont'd

- **Broadcasting.**
 - A type of information transfer and it is used when there is a participation of single sender and all recipient.
 - Represents the **one-to-all** communication.
 - The recipients should be in a **local network**.
 - Example:
 - Video and audio broadcasting.



Sockets Security in Python

Lecture 6

Prepared by
Dr. Hamada I. AbdulWakel

Lecture

06

SECURING CONNECTIONS

- A website URL starts with http:// or https://.
- The "s" in https:// signifies users are visiting a secure website, and the connection is encrypted and protected.
- HTTPS combines HTTP (Hyper Transfer Protocol Secure) and SSL (Secure Socket Layer).



Course overview

Lecture

06

SECURING CONNECTIONS

Cont'd

- Network security is important for office, home, and business networks.
- **Network Attacker** completely owns the network:
 - Control Wi-Fi, DNS, routers, his own websites.
 - Listen to any packet, modify packets in transit.
 - Inject his own packets into the network.

Lecture

06

SECURITY TERMINOLOGIES

- **Encryption:** the process of converting human-readable plain text to incomprehensible text, also known as ciphertext.
- **Decryption:** the process of converting cipher text to a human-readable plain text.

SECURITY TERMINOLOGIES

Cont'd

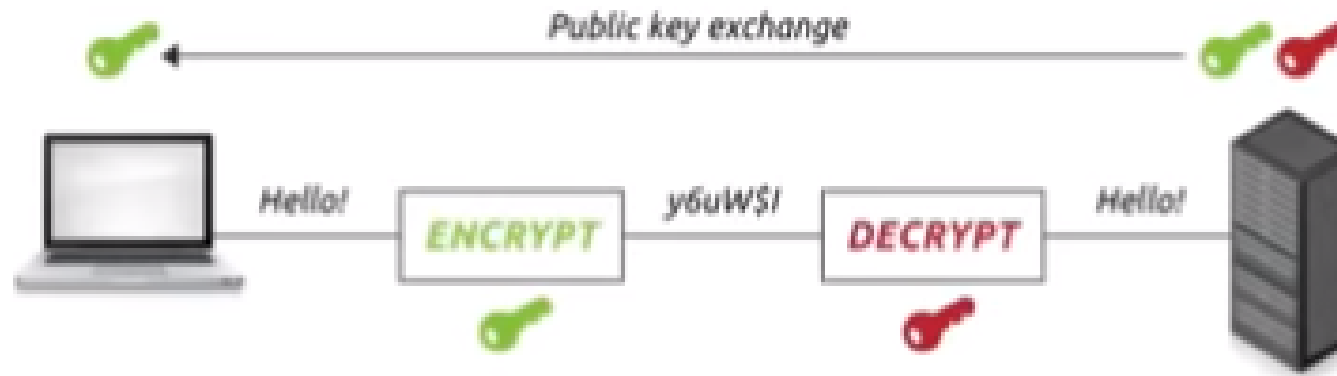
- Types of encryption:
 - **Symmetric-key encryption:** the message is encrypted and decrypted by using the same key.



SECURITY TERMINOLOGIES

Cont'd

- Types of encryption:
 - Asymmetric-key encryption:** it based on public and private keys.



Lecture

06

GETTING AN SSL CERTIFICATE

- To get an SSL certificate, you need to go through the following steps:
 - Start by verifying your website information and ensuring it is the same as the information you are submitting to the CA.
 - Certificate Authority (CA): a trusted entity that issues digital certificates which certifies the owner of a public key.
 - Then, generate the certificate signing request (CSR) through an online generator or with the help of a hosting company.
 - Submit the CSR to the certificate authority to validate the domain.
 - Once the process is complete, install and use the SSL certificate on your website.

Lecture

06

OPENSSL TOOLKIT

- [Install](#) the [OpenSSL](#) toolkit.
- Create the Private Key and Certificate Signing Request (CSR) files.
- Submit CSR to your certificate authority (CA) of choice.
- Retrieve your certificate from the certificate authority.
- Download the “root certificate” from the same certificate authority.

Lecture

06

TCP-SSL CLIENT/SERVER EXAMPLE

- Code: 1- the server

```
import socket, ssl
host = "127.0.0.1"
port = 10500
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
#context creation
ctx = ssl.SSLContext(ssl.PROTOCOL_TLS_SERVER)
ctx.load_cert_chain("/path/to/certServer.pem", "/path/to/certKey.pem")
sock.bind((host, port))
```

TCP-SSL CLIENT/SERVER EXAMPLE

Cont'd

- Code: 2- the client

```
import socket, ssl
host = "127.0.0.1"
port = 10500
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
#context creation
ctx = ssl.SSLContext(ssl.PROTOCOL_TLS_CLIENT)
ctx.load_verify_locations("/path/to/caCert.pem")
```

Domain Name System

Lecture 7

Prepared by
Dr. Hamada I. AbdulWakel

Lecture

07

NAME RESOLUTION

- Although TCP/IP uses IP addresses to locate and connect to hosts, users typically prefer to use friendly names.



Lecture

07

NAME RESOLUTION

Cont'd



- **Name resolution systems** provide the translation between alphanumeric names and numerical addresses.

NAME RESOLUTION

Cont'd

- There are two common methods for implementing name resolution:
 - A **static file** on each host on the network, containing all the name-to-address translations.
 - Like HOSTS file in Windows/Linux.
 - If a change occurs, every device's HOSTS file must be updated.
 - A **centralized server** that all hosts on the network connect to for name resolution.
 - All devices point to this centralized DNS server for name resolution, ensuring that changes only need to occur in one place.

Lecture

07

DOMAIN NAME SYSTEM

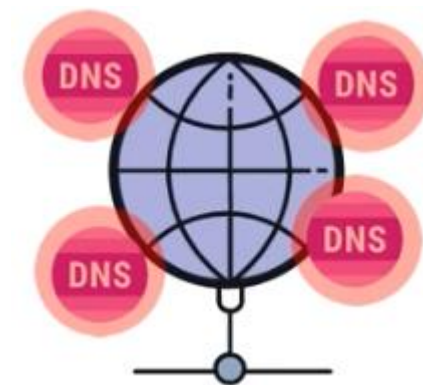
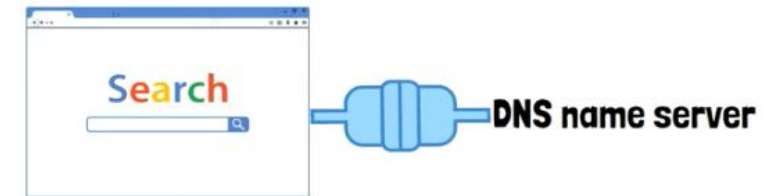
- **DNS** is used on the Internet to provide a standard naming convention for locating IP-based computers.
- **DNS servers:** connected servers have powerful computing resources, storing data, and providing IP addresses for requested domains.
- On Windows/Linux
 - **NOTE:** IP may vary
`nslookup google.com`

```
drhamada@drhamada-Latitude-E6520:~$ nslookup google.com
Server:          127.0.0.53
Address:         127.0.0.53#53

Non-authoritative answer:
Name:   google.com
Address: 142.250.200.206
Name:   google.com
Address: 2a00:1450:4006:80c::200e
```

DNS SERVERS

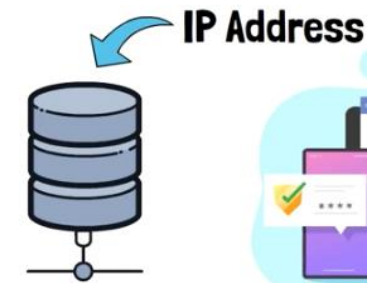
- There are four types of DNS servers:
 - **DNS Resolver:**
 - Provided by Internet Service Provider (ISP).
 - Connects machines with DNS name server.
 - **Root Name Server (RNS):**
 - Identical 13 DNS servers around the world.
 - Keep tracking of new added domains names.



DNS SERVERS

Cont'd

- There are four types of DNS servers:
 - **Top-level Domain Server (TLD):**
 - Stores all domain names with the same extension.
 - Domain extensions like .com, .net, .info, .org, .edu,
 - Extension together with country code like ae form the set of possible suffixes for valid domain names.
 - **Authoritative Name Server (ANS):**
 - Stores the IP address of the requested website.
 - Ensures if the user has the permissions to access IP or not.



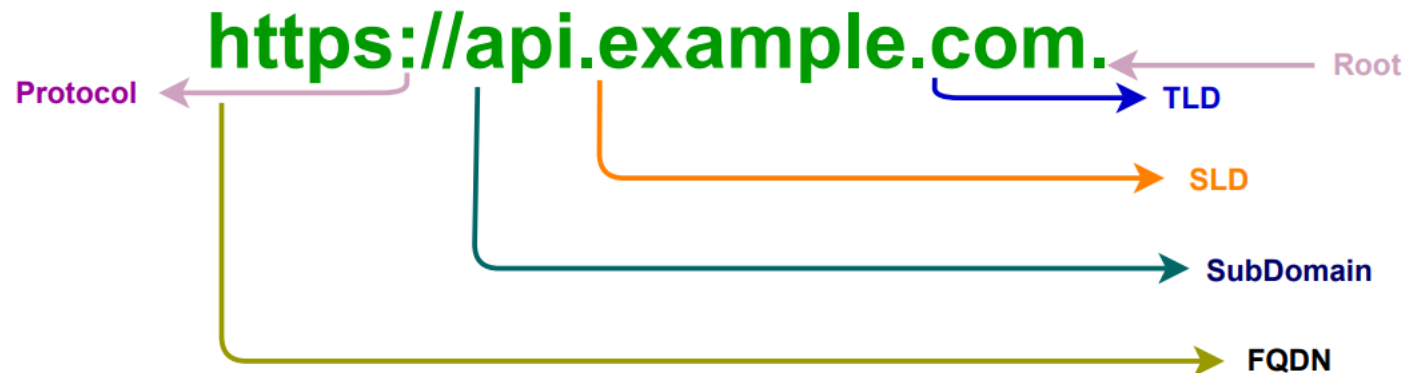
INTERNAL WORK OF DNS

Cont'd

- The naming system on which DNS is based is a hierarchical and logical tree structure called **the domain namespace**.
- Every node in the DNS domain tree can be identified by a fully qualified domain name (FQDN).
- To allow computers to properly recognize a fully qualified domain name:
 - Dots are placed between each part of the name.
 - All resolvers start their lookups at the root.
 - The resolver navigates its way down the tree until it gets to the left-most part of the domain name.

FULLY QUALIFIED DOMAIN NAME

- Fully qualified domain name (FQDN): names an internet site or host by appending its organization's full domain name to the name of a particular machine in that organization.



Lecture

07

DNS RECORD INFORMATION

- **A:** used for converting domain name to IPv4 address.
- **AAAA:** used for converting domain name to IPv6 host address.
- **PTR (Pointer)** : used for converting IP to a domain name.
- **CNAME (Canonical Name):** used to map a domain name as an alias for another domain.
- **MX (Mail eXchanger):** specifies a mail server responsible for accepting email messages on behalf of a recipient's domain.
- **NS (Name Server):** gives a name for each group of servers used in a domain.
- **TXT (Descriptive Text):** lets the administrator insert into the DNS record.

DNS RECORD INFORMATION

Cont'd

- Code:

```
import dns.resolver
def lookup(name):
    for qtype in 'A', 'AAAA', 'CNAME', 'MX', 'NS':
        answer = dns.resolver.query(name, qtype, raise_on_no_answer=False)
        for val in answer:
            print(val)
```

```
lookup('google.com')
```

SOCKETS WITH GETADDRINFO METHOD

- The `getaddrinfo()` uses the DNS protocol to talk to your name servers, which are configured using their IP address.
- The `getaddrinfo()` method returns a list of tuples - `[((), ())]` - for a given network service with which socket objects can be created to connect to that service.

```
socket.getaddrinfo(host, port, family=0, type=0, proto=0, flags=0)
```

SOCKETS WITH GETADDRINFO METHOD

Cont'd

```
socket.getaddrinfo(host, port, family=0, type=0, proto=0, flags=0)
```

- Host: IP address or host name.
- Port: could be a number or service name like http and www.
- Family: IP version.
- Type: Stream or Datagram.
- Proto: TCP or UDP.
- Flags: used for filtering the addresses.

Email Composition and Decoding

Lecture 8

Prepared by
Dr. Hamada I. AbdulWakel

Lecture

08

E-MAIL DEFINITION

- Sending and receiving of information through internet is known as an E-mail.
- Today, email is one of the most popular methods of digital communication.
- Sharing an email is done through the use of email address.
- Email is a critical means of communication in today's world, but it can be time-consuming and tedious to send out emails manually.

- **A Mail User Agent (MUA):** the client application running on a computer that is used to send and receive mail.
 - Examples: Apple Mail and Microsoft Outlook.
- **A Mail Transfer Agent (MTA):** accepts messages from a sender and routes them along to their destinations.
 - Examples: Sendmail, Postfix, and Microsoft Exchange.
- **A Mail Delivery Agent (MDA):** receives email from an MTA for delivery to a mailbox.

Lecture

08

E-MAIL PARTS

- **Email headers:** consists of the sender, receiver, subject, and timestamp. Headers are part of every email and is has many usages.
- **Email message:**
 - The message or body of the email is where a lot of the information about the email exists.
 - Using this information, we can extract key details about the purpose of the message.

- **Email attachment:**
 - Files that are part of the email body. Since data is sent in various forms, parsing attachments sent through email is important to create a powerful user experience.
- **Email HTML:**
 - Emails generally come in plain text and HTML.
 - Parsing HTML emails is a powerful way to get extra information to display in a different format and from various sources such as newsletters and transactional emails.

E-MAIL HEADER INFORMATION

Cont'd

- **An Email Header:** is metadata that accompanies every email and contains detailed information, such as the sender, receiver, route, timestamp, and more.
- Mailbox and email service providers use email headers to authenticate the email senders and properly allocate an email in the inbox.
- Each field in the header has a corresponding name, followed by a separator character (:) and a value identifier. The main fields located in the email message header are the **from, to, subject, and date**.

Lecture

08

E-MAIL HEADER INFORMATION

Cont'd

- **Message-ID:** is a unique identifier, generated automatically to prevent multiple deliveries. It includes various (around 50) letters and numbers.

Lecture

08

E-MAIL AUTHENTICATION

- **Email authentication** helps keep all of us safe from spam and email attacks. There are multiple email authentication protocols. Each protocol protects against a specific type of email forging.
 - **Sender policy framework (SPF)** authentication keeps malicious senders from spoofing emails and IP addresses. It used as a spam detection mechanism.
 - **Domain Keys Identified Mail (DKIM)** authentication protects against emails that have been altered in transit by a bad actor between the sender and the recipient. The email is signed with a digital signature, which can be verified by checking the sender's public key in the DNS records of the sender's domain.

Lecture

08

E-MAIL MODULE

- The email module is an email message management library.
`import email`
- It has three major components:
 - The central component is an “object model” that represents email messages.
 - The parser component takes the serialized version of an email message (a stream of bytes) and converts it into a `EmailMessage` object.
 - The generator component takes a `EmailMessage` object and turns it back into a serialized byte stream.

Lecture

08

E-MAIL MODULE

Cont'd

- It is used to create the EmailMessage object and add or remove email subcomponents. We cannot send an email message using the email module alone.
- The EmailMessage dictionary-like interface is indexed by the header names. The values of the dictionary are strings with some extra methods.
- The email module also provides support for parsing e-mail messages. Reconstructing the EmailMessage object.

Lecture

08

E-MAIL MODULE METHODS

- **email.message** submodule used to create instances of the EmailMessage class.
 - **EmailMessage()**: create a message object.
 - **set_payload()**: set the body of a message object.
 - **as_string()**: returns the message object as string.
 - **is_multipart()**: check if the message object includes a list of sub-EmailMessage objects.
 - **keys()**: return a list of all the message's header field names.
 - **values()**: return a list of all the message's header values.

- **email.utils** submodule methods:
 - **make_msgid()**: returns a string represents ant Message-ID header.
 - **formatdate()**: returns a date time as string.
 - The localtime argument used to return time related to the local timezone.
 - **parsedate_tz()**: returns either None or a 10-tuple datetime.
 - **mktime_tz()**: create a Universal Time Coordinated (UTC) timestamp from a 10-tuple datetime.
 - Unix time: the number of seconds that have elapsed since 00:00:00 UTC on 1 January 1970.

Lecture

08

CREATING AN E-MAIL EXAMPLE

- Code:

```
import email.utils, email.message
text = " Hello, This is a test message. I hope you enjoy it! "
msg = email.message.EmailMessage()
msg['To'] = 'recipient@example.com'
msg['From'] = 'sender@example.com'
msg['Subject'] = 'Test Message'
msg['Date'] = email.utils.formatdate(localtime = 1)
msg['Message-ID'] = email.utils.make_msgid()
msg.set_payload(text);  print(msg.as_string())
```

Lecture

08

THE MULTIPURPOSE INTERNET MAIL EXTENSIONS

- **MIME** is a set of rules for encoding data, rather than deal with simple plain text only inside e-mails.
- **MIME** message can include several different parts (e.g., a plain-text message, an image file attachment, a PDF attachment) into the message body.
- **MIME** operates by adding headers to an e-mail message and then giving it content that looks like plain text to the machine.

Lecture

08

CREATING A MIME EMAIL

- To compose a message with attachments, you will generally follow these steps:
 1. Create a **MIMEMultipart** object and set its message headers.
 2. Create a **MIMEText** object with the message body text and attach it to the **MIMEMultipart** object.
 3. Create appropriate **MIME** objects for each attachment and attach them to the **MIMEMultipart** object.
 4. Finally, call `as_string()` on the **MIMEMultipart** object to write out the resulting message.

MIME TYPES EXAMPLE

Cont'd

- Code:

```
message = "Hello, This is a test message. I hope you enjoy it!"  
msg = email.mime.multipart.MIMEMultipart()  
msg['To'] = 'recipient@example.com'  
msg['From'] = 'sender@example.com'  
msg['Subject'] = 'Test Message'  
msg['Message-ID'] = email.utils.make_msgid()  
body = email.mime.text.MIMEText(message, 'plain')  
msg.attach(body)  
attachment(msg, 'Document2.pdf');    print(msg.as_string())
```

The background features a collage of educational icons overlaid on a blurred image of hands typing on a laptop. The icons include a graduation cap, a target with an arrow, an open book, a play button, a magnifying glass, a group of people, and a person at a desk. The icons are semi-transparent and arranged in a vertical sequence.

Email Handling

Lecture 9

Prepared by
Dr. Hamada I. AbdulWakel

EMAIL PROTOCOLS

- **Simple Mail Transfer Protocol (SMTP):** It can only be used by the MTA for sending e-mails from one host to another.
- **Post Office Protocol 3 (POP3):** It provides a simple and standardized way for the users to gain access to the mailboxes and then download the messages.
- **Internet Message Access Protocol (IMAP):** It provides a simple and standardized way for accessing your e-mail. It requires only a small data transfer; this scheme works well even over a slow connection.

Lecture

09

POP3 VS. IMAP

- With IMAP:
 - All your mail stays on the server in multiple folders.
 - You can connect to any computer and see all your mail and mail folders.
 - If you have a dedicated connection to the Internet or you like to check your mail from various locations.
- With POP3:
 - You only have one folder, the Inbox folder.
 - When you open your mailbox, new mail is moved from the host server and saved on your computer.

Lecture

09

EMAIL MODULES

- Python has three modules:
 - **smtplib** for SMTP protocol.
 - **poplib** for POP3 protocol.
 - **imaplib** for IMAP protocol.
- Each module has options for transmitting the information securely by using the Transport Layer Security (TLS) protocol.
- Each protocol also uses some form of authentication for ensuring the confidentiality of the data.

Lecture

09

SMTPLIB METHODS

- **SMTP()**: returns an object of SMTP class.
- **ehlo()**: identifies the client to the server. An extended hello message will be sent by the client to the server.
- **starttls()**: starts unsecured SMTP connection with encryption for the traffic.
- **login()**: takes username and password arguments to login to SMTP server.

Lecture

09

SMTPLIB METHODS

Cont'd

- **sendmail()**: performs an entire mail transaction.
- **quit()**: terminates SMTP session.

LOCAL SMTP SERVER

Cont'd

- Code:

```
import smtplib
session = smtplib.SMTP('localhost', 1025)
subject = 'Test'
body = "A test message to check the email sending using STMP"
msg = f"Subject: {subject} \n\n{body}"
```

```
session.sendmail("hamada@midocean.edu.km", "hamada_mbsoft@yahoo.com",
msg)
```

Lecture

09

LESS SECURE APPS

- An App Password gives permission to a non-Google app to access Google's SMTP server.
- App Passwords can only be used with accounts that have 2-step verification turned on.

To help keep your account secure, from **May 30, 2022**, Google no longer supports the use of third-party apps or devices which ask you to sign in to your Google Account using only your username and password.

Important: This deadline does not apply to Google Workspace or Google Cloud Identity customers. The enforcement date for these customers will be announced on the Workspace blog at a later date.

For more information, continue to read.

- Visit: <https://support.google.com/accounts/answer/6010255?hl=en>

Lecture

09

POP3 METHODS

- **POP3()**: creates a POP3 object.
- **POP3_SSL()**: creates a secured POP3 object.
- **user()**: sets the username to get the mailbox.
- **pass_()**: sets the password for the username.
- **stat()**: returns mailbox status in a tuple format (message count, mailbox size).

POP3 METHODS

Cont'd

- **list():**
 - Retrieves a list of messages.
 - Returns a tuple of:
(responsecode, msglist, sizeofresponse)
- **retr():**
 - Retrieves a single message.
 - Returns a tuple of:
(responsecode, lines, size)
 - Marks the Email as read.

Lecture

09

EMAIL ACCESS CLIENT APP

- Code:

```
import getpass, poplib
popserver = 'pop.gmail.com'
popport = 995
username = input("Enter your email user ID: ")
# "tgrjnuccuhsjjzji"
password = getpass.getpass(prompt="Enter your email password: ")
mailbox = poplib.POP3_SSL(popserver , popport)
mailbox.user(username)
mailbox.pass_(password)
```

Lecture

09

IMAP METHODS

- **IMAP4_SSL()**: creates a secure IMAP object.
- **login()**: logs in to the server.
- **list()**:
 - Lists the folders present in mailbox.
 - Returns a tuple of:
(**'Ok'**, [**list**])

IMAP METHODS

Cont'd

- **select():**
 - Selects a specific mail folder to get messages from it.
 - Returns a tuple of:
(**'Ok'**, [list of one item specify number of emails in this folder])
- **fetch():**
 - Returns a particular message.
 - The 1st argument is the number of message, the second is the parsing format.
 - Returns a tuple of:
(**'Ok'**, [list of message header]).

Lecture

09

EMAIL ACCESS CLIENT APP

- Code:

```
import imaplib, email
imapServer = 'imap.gmail.com'
sender = 'hamada.mbsoft@gmail.com'
password = 'tgrjnuccuhsjjzji'
conn = imaplib.IMAP4_SSL(imapServer)
conn.login(sender, password)
print(conn.list())
print(conn.select('INBOX'))
```

File Transfer Protocol In Python

Lecture 10

Prepared by
Dr. Hamada I. AbdulWakel

Lecture

10

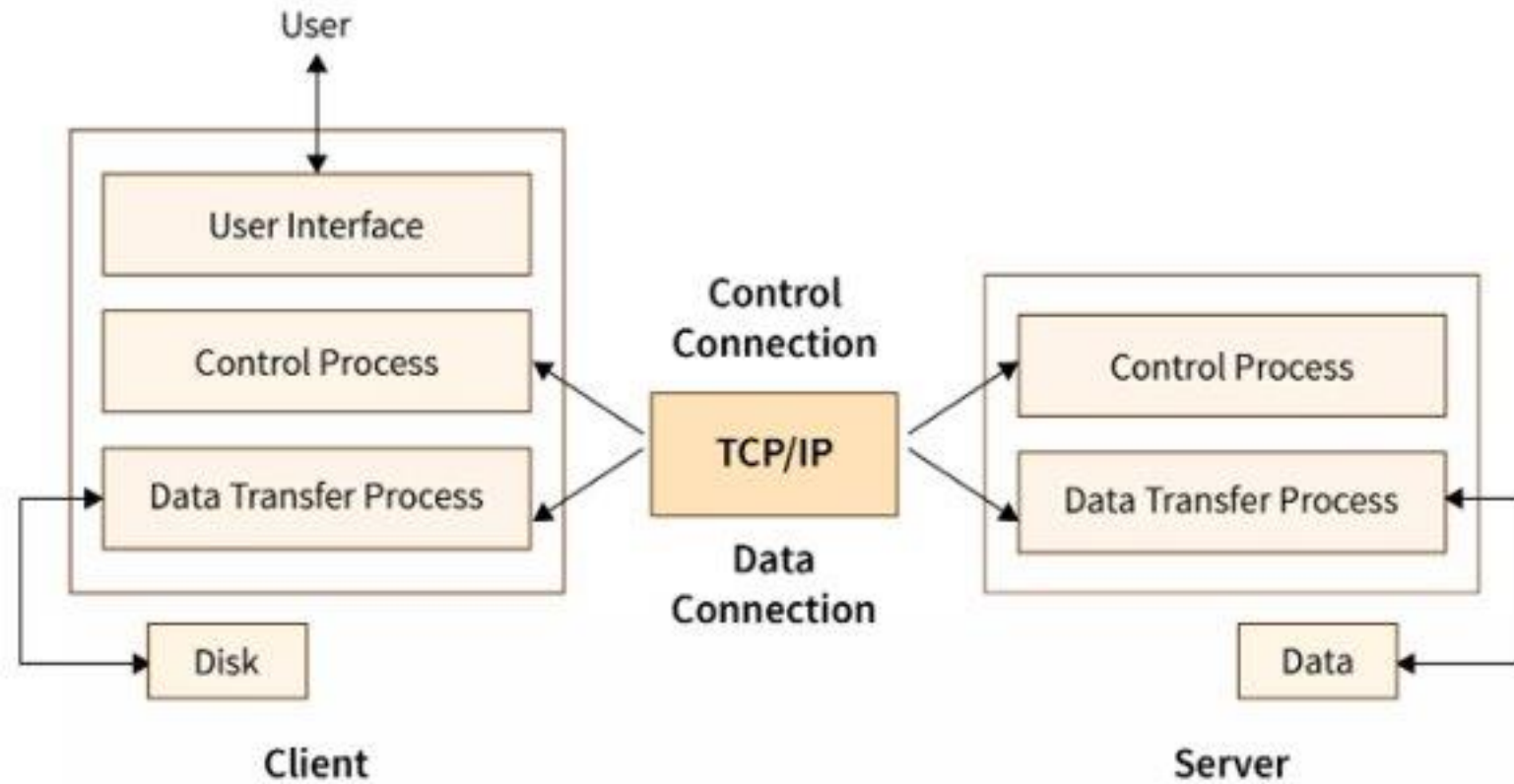
FILE TRANSFER PROTOCOL

- 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 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.

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

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.

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.

- 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")
```



Thank You

