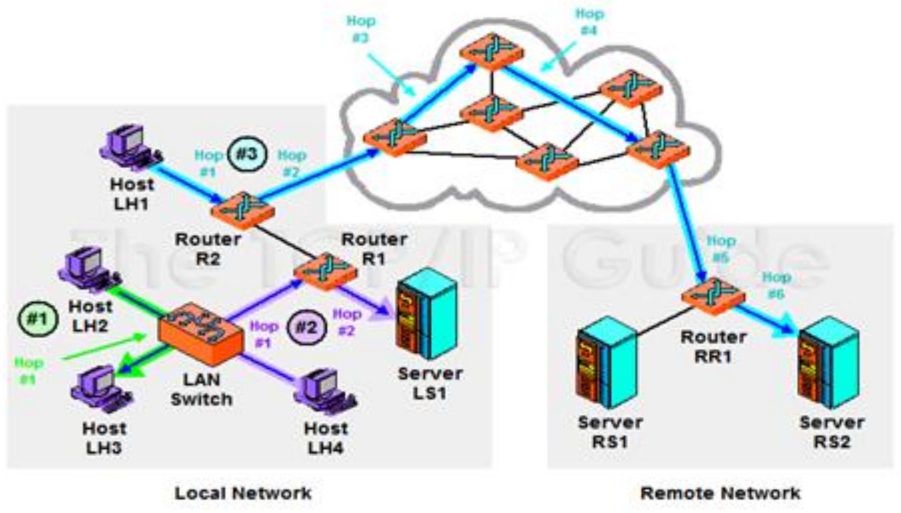
DSCI 510
PRINCIPLES OF PROGRAMMING FOR DATA SCIENCE

Itay Hen

NETWORKED PROGRAMS

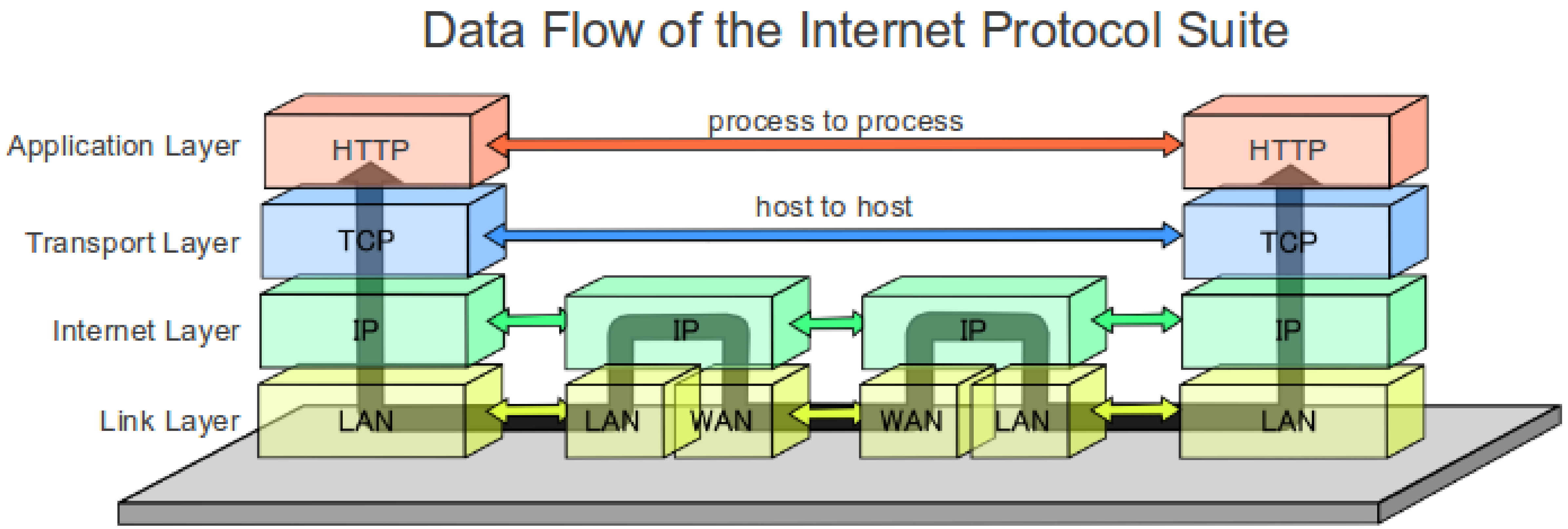
Internet in a Nutshell

- The internet is a network of networks.
- The IP layer: Each computer needs an address, i.e., an IP (Internet Protocol) address
  - IPv4: 128.9.35.232 (2^{32} addresses)
  - IPv6: 2001:0db8:0000:0000:0000:ff00:0042:8329 (2^{128} addresses)
- Domain Name System (DNS): maps IP addresses to nice hierarchical names
  - 128.9.36.51 == globulin.isi.edu == www.nimhgenetics.org
- To transmit data across the network from computer A to B:
  1. Break the data into small pieces: packets
  2. Send each packet with destination address to a computer you are connected to that you think it will get it closer to the destination (*Routing*). Repeat.
  3. Destination computer re-assembles the packets.



Internet: The TCP/IP framework

The **TCP/IP framework:** a suite of communication protocols that organizes how data is packaged, transmitted, and received over the internet



Internet: The TCP/IP framework

Layer (Top to Bottom)	Main Function	Common Protocols / Examples
5. Application Layer	Provides network services directly to user applications (e.g., web, email, file transfer).	HTTP, HTTPS, FTP, SMTP, DNS, SSH, Telnet
4. Transport Layer	Ensures end-to-end communication, reliability, and flow control between hosts.	TCP, UDP
3. Network (Internet) Layer	Routes packets across networks; handles logical addressing.	IP, ICMP, ARP
2. Data Link Layer	Manages node-to-node communication and error detection on the same network.	Ethernet, Wi-Fi (IEEE 802.11), PPP
1. Physical Layer	Transmits raw bits over a physical medium; defines hardware, signaling, and cabling.	Cables, Hubs, Repeaters, Fiber optics

Transmission Control Protocol (TCP)

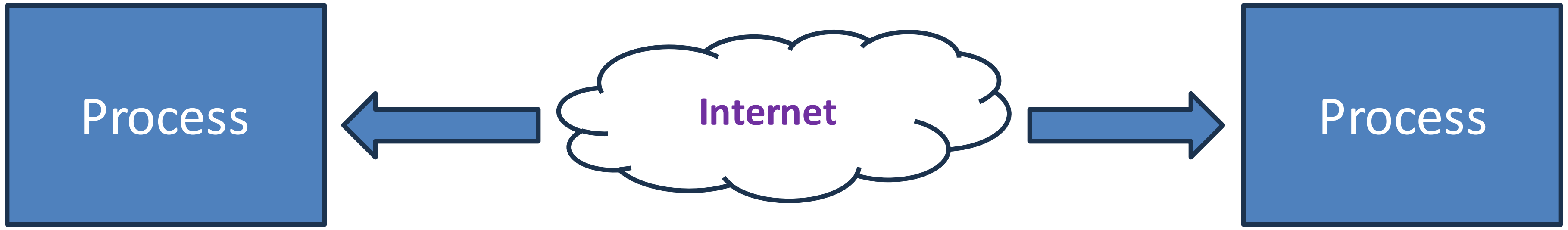
- TCP is a **connection-oriented** protocol operating at the **Transport Layer** of the Internet Protocol Suite.
- Ensures **reliable**, **ordered**, and **error-checked** delivery of data between applications.

Key Features

- **Connection Establishment**
- **Reliability:** Implements **acknowledgements (ACKs)** and **retransmissions** for lost packets.
- **Flow Control:** Uses a **sliding window** to regulate data flow between sender and receiver.
- **Congestion Control:** Dynamically adjusts data transmission rate to prevent network congestion.
- **Segmentation & Reassembly:** Breaks large messages into **segments**, reassembles them in order at the receiver.

TCP Connections / Sockets

- **TCP sockets** are the **programming interface** that allows applications to use the **Transmission Control Protocol (TCP)** for communication over a network.
- A **TCP socket** is an **endpoint** of a two-way communication link between two programs running on the network.
- Each endpoint is identified by a **(IP address, port number)** pair.

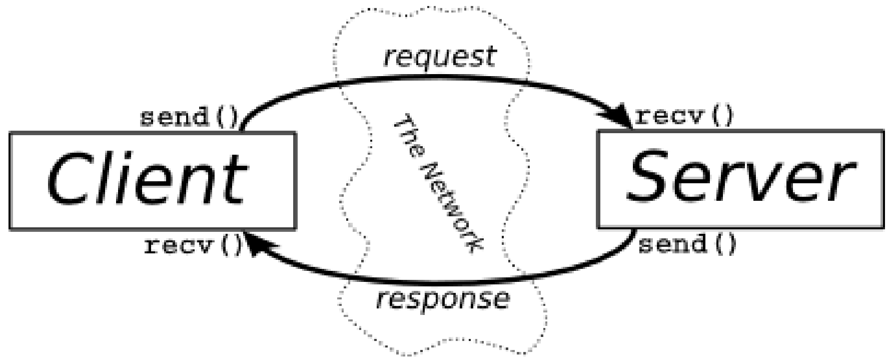


This is the layer where programmers like us come in whenever we want to establish communication with the outside world.

TCP Connections / Sockets

The two **endpoints** of a **TCP connection** are typically referred to as the **client** and the **server**.

- The **server** is the machine (or process) that **waits for incoming connections**.
  It's "passive" — it listens on a known **port number** (e.g., 80 for HTTP, 22 for SSH).
- The **client** is the machine (or process) that **initiates** the connection.
  It's "active" — it knows the server's **IP address** and **port number** and requests a connection.



1. **Server side:**
   - Creates a socket (`socket()`).
   - Binds it to an address and port (`bind()`).
   - Listens for incoming connections (`listen()`).
   - Accepts a connection (`accept()`).

2. **Client side:**
   - Creates a socket (`socket()`).
   - Connects to the server's address and port (`connect()`).

Once connected, **both sides can send and receive data** using:

- `send()` / `sendall()`
- `recv()`

TCP Port Numbers

- A **port** is an application-specific or process-specific software communications endpoint
- It allows multiple networked applications to coexist on the same server

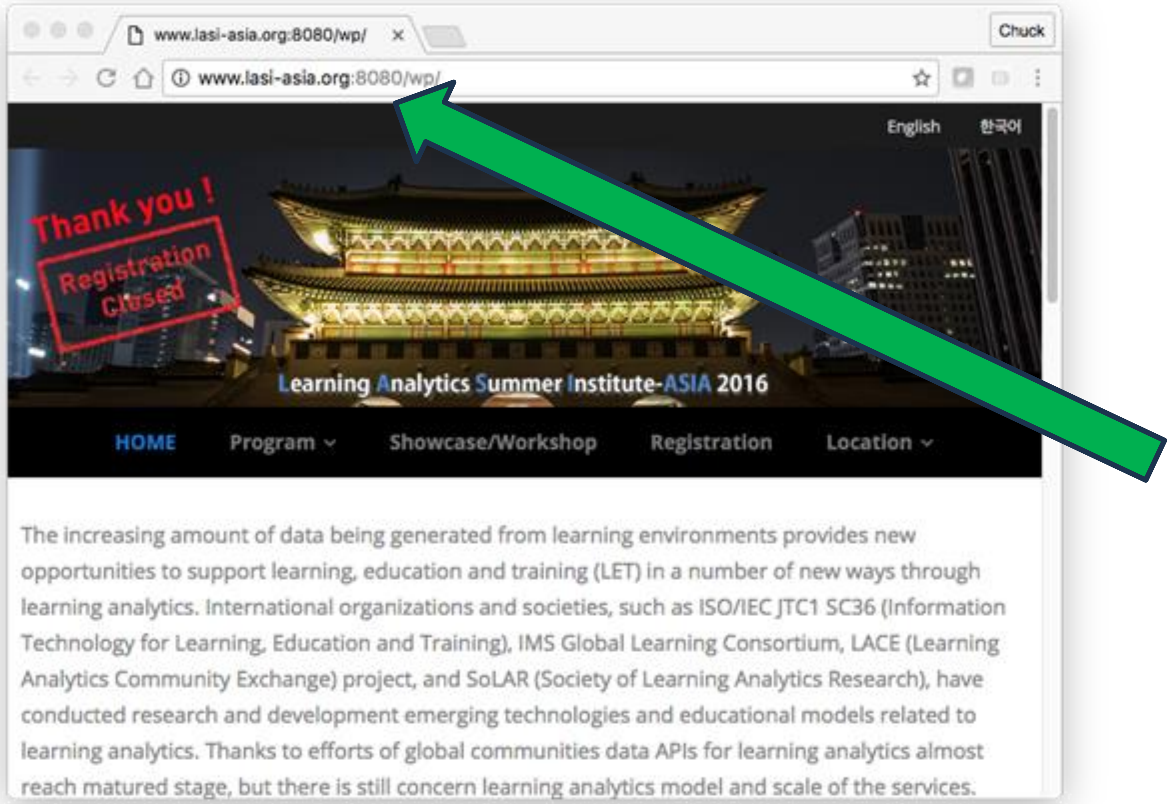
Port Number	Assignment
21	File Transfer Protocol (FTP)
22	Secure Shell (SSH)
23	Telnet remote login service
25	Simple Mail Transfer Protocol (SMTP)
80	Hypertext Transfer Protocol (HTTP)
443	HTTP Secure (HTTPS)
110	Post Office Protocol (POP3)

Example Applications

- **HTTP/HTTPS** (web traffic)
- **SMTP/IMAP** (email)
- **FTP/SFTP** (file transfer)
- **SSH** (secure shell)

https://en.wikipedia.org/wiki/List_of_TCP_and_UDP_port_numbers

Port Numbers in URL


www.lasi-asia.org:8080/wp/
Chuck
www.lasi-asia.org:8080/wp/
English
한국어
Thank you !
Registration Closed
Learning Analytics Summer Institute-ASIA 2016
HOME
Program
Showcase/Workshop
Registration
Location

The increasing amount of data being generated from learning environments provides new opportunities to support learning, education and training (LET) in a number of new ways through learning analytics. International organizations and societies, such as ISO/IEC JTC1 SC36 (Information Technology for Learning, Education and Training), IMS Global Learning Consortium, LACE (Learning Analytics Community Exchange) project, and SoLAR (Society of Learning Analytics Research), have conducted research and development emerging technologies and educational models related to learning analytics. Thanks to efforts of global communities data APIs for learning analytics almost reach matured stage, but there is still concern learning analytics model and scale of the services.

Sometimes we see the port number in the URL if the web server is running on a "non-standard" port.

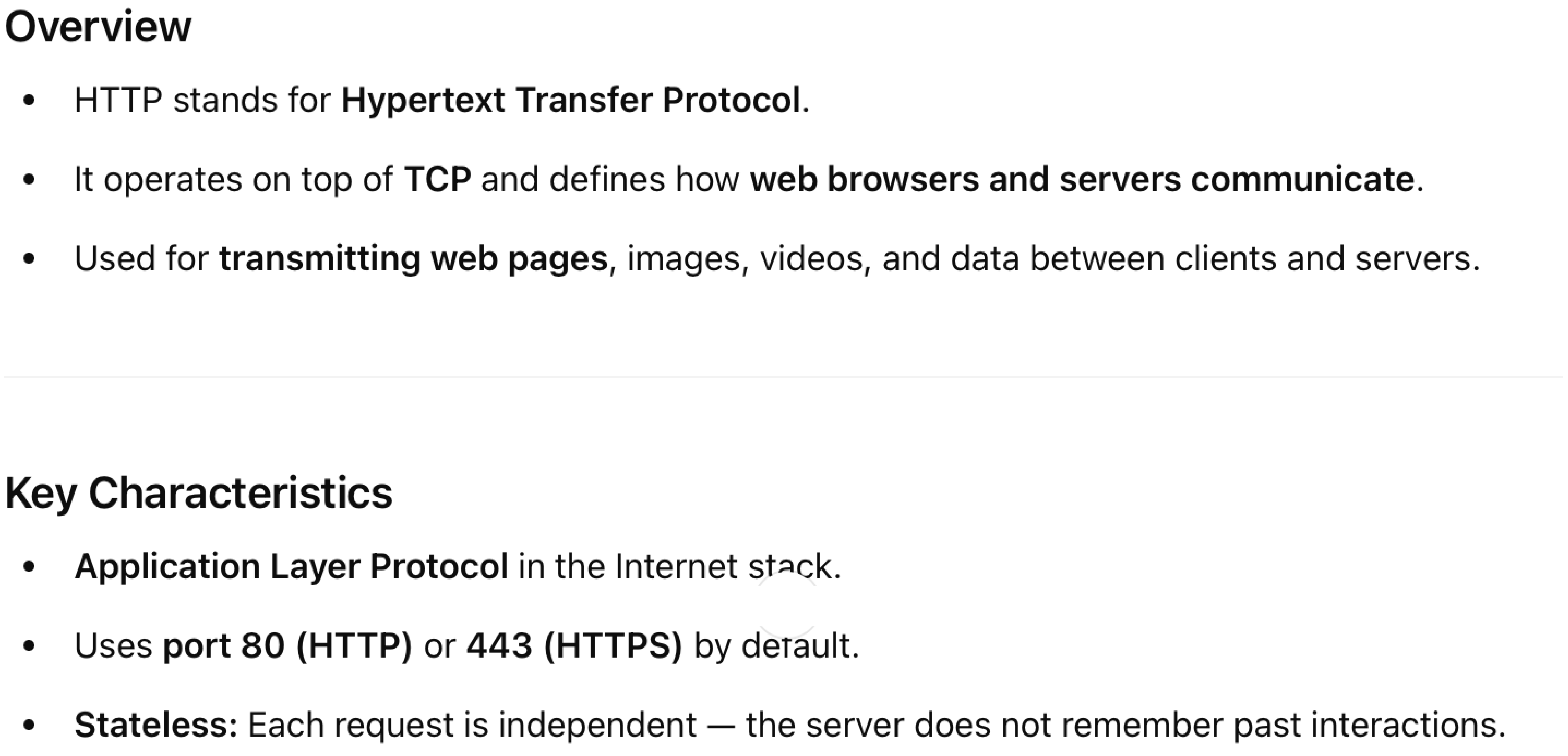
What is a Protocol ?

- A set of rules that all parties follow so we can predict each other's behavior
- And not bump into each other
  - On two-way roads in USA, drive on the right-hand side of the road
  - On two-way roads in UK, drive on the left –hand side of the road

Hypertext Transfer Protocol (HTTP)

- HTTP is an application layer protocol for transmitting hypermedia documents, such as HTML. HTTP is the foundation of data communication for the World Wide Web.
- Python has built-in support for HTTP called **sockets** which makes it very easy to make network connections and retrieve data over those **sockets** in a Python program.
- A **socket** is much like a file, except that a single **socket** provides a two-way connection between two programs. You can both read and write to the same **socket**.
- If you try to read a **socket** when the program on the other end of the **socket** has not sent any data, you just sit and wait.
- If the programs on both ends of the **socket** simply wait for some data without sending anything, they will wait for a very long time, so an important part of programs that communicate over the internet is to have some sort of protocol.
- A protocol is a set of precise rules that determine who is to go first, what they are to do, and then what the responses are to that message, and who sends next, and so on.
- In a sense, the two applications at either end of the socket are engaging in a dance, making sure not to step on each other's toes.
- HTTP description: https://www.w3.org/Protocols/rfc2616/rfc2616.txt

Hypertext Transfer Protocol (HTTP)



Overview

- HTTP stands for **Hypertext Transfer Protocol**.
- It operates on top of **TCP** and defines how **web browsers and servers communicate**.
- Used for **transmitting web pages**, images, videos, and data between clients and servers.

Key Characteristics

- **Application Layer Protocol** in the Internet stack.
- Uses **port 80 (HTTP)** or **443 (HTTPS)** by default.
- **Stateless:** Each request is independent — the server does not remember past interactions.

Sockets in Python

Python has built-in support for TCP sockets

```
import socket

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

Sockets in Python

Python has built-in support for TCP sockets

```
import socket

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

- `socket.AF_INET`
  - Specifies the **address family** — in this case, **IPv4** (Internet Protocol version 4).
  - If you were using IPv6, you'd write `socket.AF_INET6`.
- `socket.SOCK_STREAM`
  - Specifies the **socket type** — here it means **TCP**, a *stream-oriented*, reliable protocol.
  - For **UDP**, you'd use `socket.SOCK_DGRAM`.

Sockets in Python

Python has built-in support for TCP sockets

```
import socket

my_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

my_socket.connect(('data.pr4e.org', 80))
```

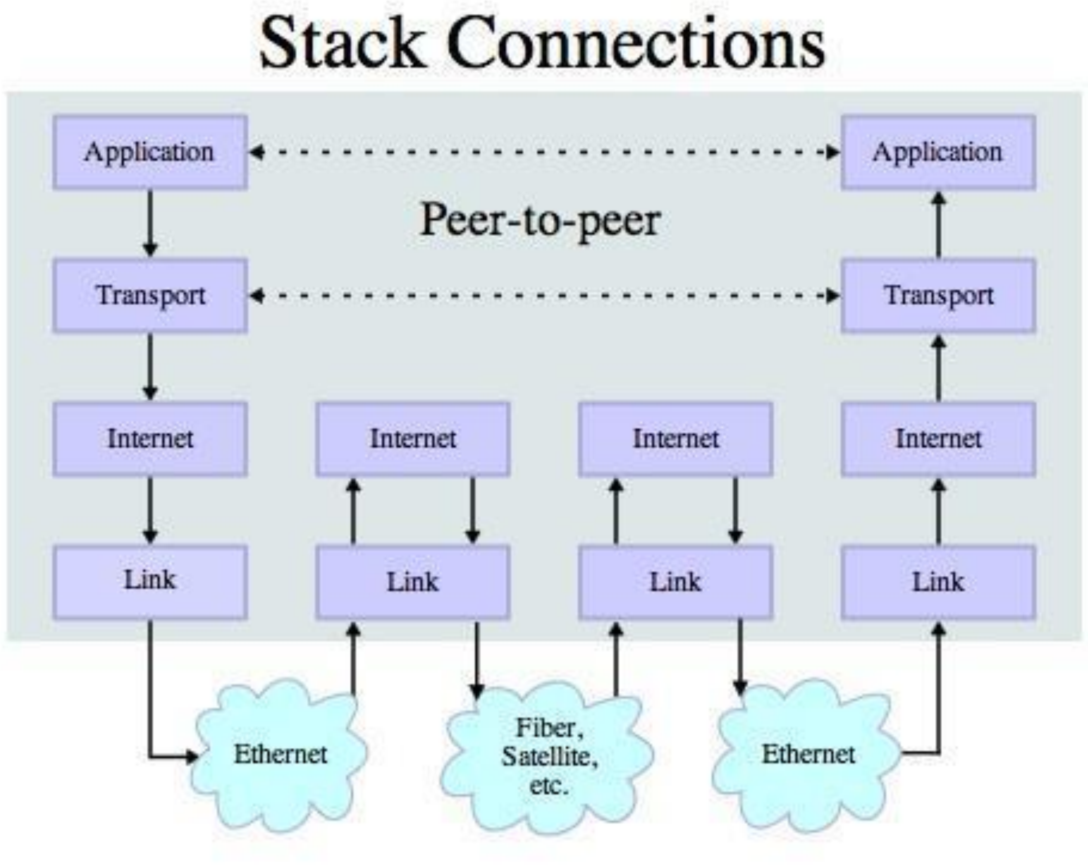
Host/IP port

This tells your socket to **initiate a TCP connection** to a remote server.

- `'data.pr4e.org'` is the **domain name** of the remote host.
  Python internally resolves it to an IP address (via DNS).
- `80` is the **port number** — by convention, port **80** is used by **HTTP servers** (web traffic).

Application Protocol

- Since TCP (and Python) gives us a reliable transport (socket), what do we want to do with the socket? What problem do we want to solve?
- Application Protocols
  - – Mail
  - – World Wide Web
  - – Secure Login



Source: http://en.wikipedia.org/wiki/Internet_Protocol_Suite

TCP ← → HTTP

- Once we have a TCP connection (via a socket), we can send **HTTP-formatted text** over it.
- For example:

```
cmd = 'GET /index.html HTTP/1.1\r\nHost: example.com\r\n\r\n'
my_socket.send(cmd.encode())
```

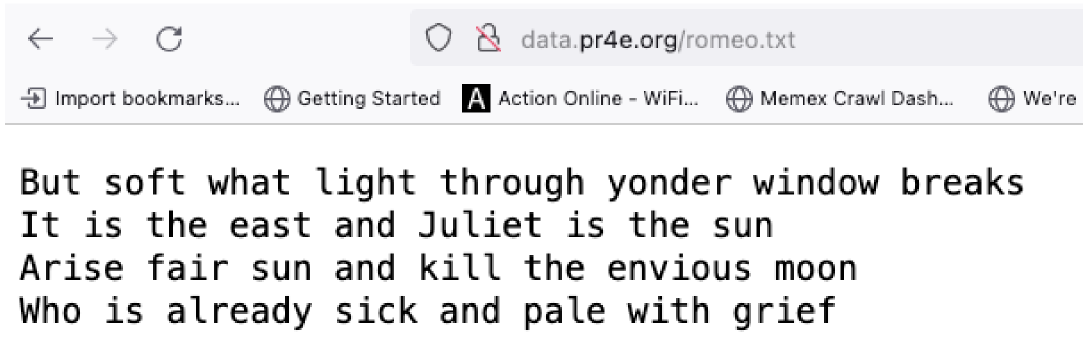
- This text follows the **HTTP protocol format** — it's not understood by TCP itself, but by the **web server** on the other end (which is expecting HTTP requests).

Layer	Example Role	Example Component
Application Layer	Defines request meaning	HTTP (`GET`, `POST`, etc.)
Transport Layer	Moves data reliably	TCP
Network Layer	Routes packets	IP
Link Layer	Moves bits physically	Ethernet / Wi-Fi

Making an HTTP Request

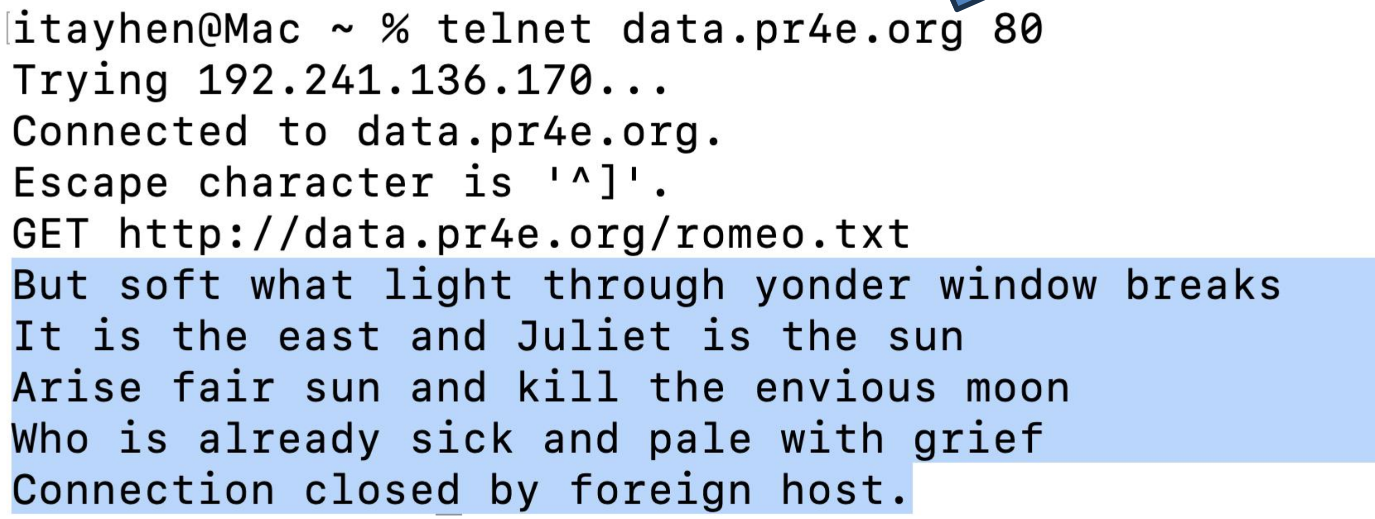
There are several ways to retrieve web data from the internet using the HTTP protocol.

Browser



```
But soft what light through yonder window breaks
It is the east and Juliet is the sun
Arise fair sun and kill the envious moon
Who is already sick and pale with grief
```

Command Line



```
itayhen@Mac ~ % telnet data.pr4e.org 80
Trying 192.241.136.170...
Connected to data.pr4e.org.
Escape character is '^]'.
GET http://data.pr4e.org/romeo.txt
But soft what light through yonder window breaks
It is the east and Juliet is the sun
Arise fair sun and kill the envious moon
Who is already sick and pale with grief
Connection closed by foreign host.
```

***telnet:** a network protocol and corresponding software tool that allows us to remotely access (and control) another computer over a network through a text-based command-line interface.

Socket send() and recv() commands

`send()` — Sending Data

Syntax

```
socket.send(bytes)
```

Description

- Sends **binary data** (bytes) through the socket.
- You must encode any string before sending, since sockets deal in **bytes**, not text.

```
cmd = 'GET /romeo.txt HTTP/1.0\r\nHost: data.pr4e.org\r\n\r\n'
s.send(cmd.encode())     # Convert string → bytes and send
```

Socket send() and recv() commands

`recv()` — Receiving Data

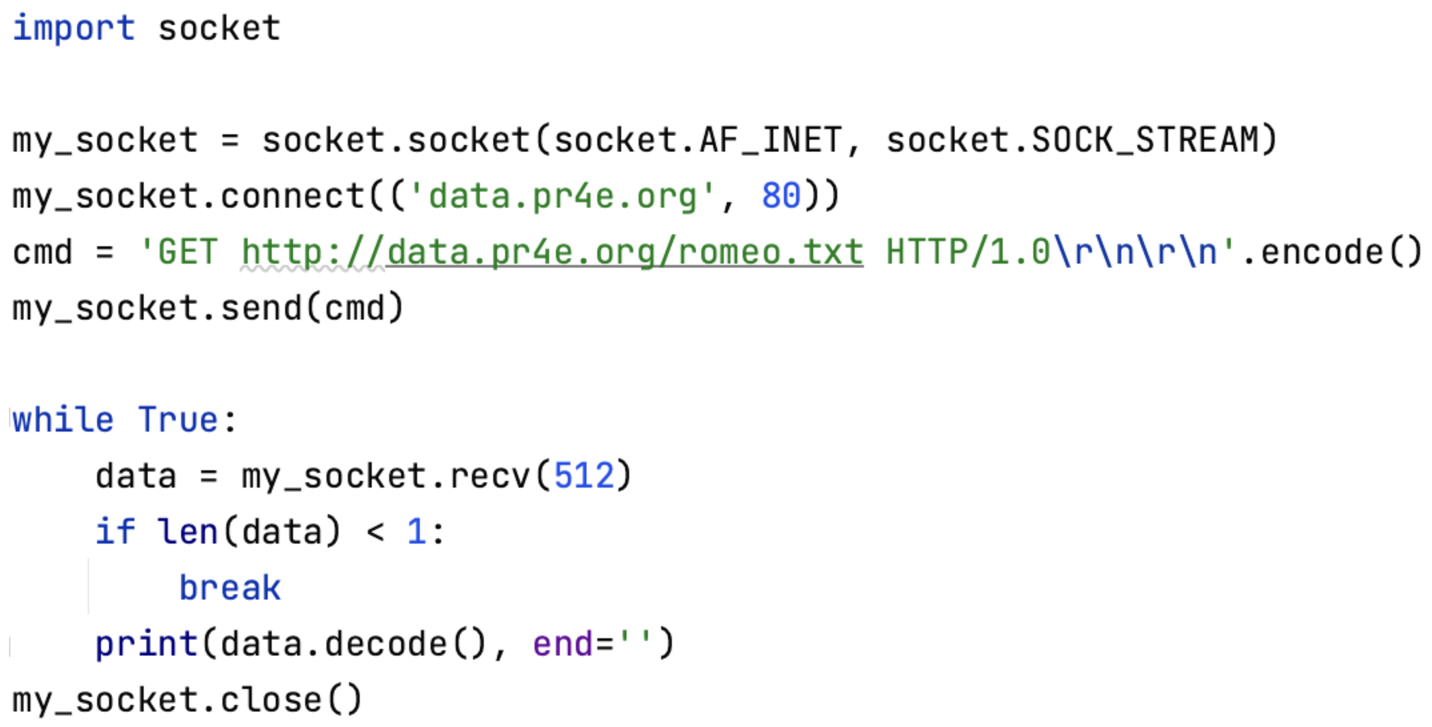
Syntax

```
data = socket.recv(buffer_size)
```

Description

- Reads up to `buffer_size` bytes from the socket.
- Returns the data as a **bytes object**.
- If the connection is closed by the remote host, it returns an empty string (`b''`).

An HTTP Request in Python



```python
import socket

my_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
my_socket.connect(('data.pr4e.org', 80))
cmd = 'GET http://data.pr4e.org/romeo.txt HTTP/1.0\r\n\r\n'.encode()
my_socket.send(cmd)

while True:
    data = my_socket.recv(512)
    if len(data) < 1:
        break
    print(data.decode(), end='')
my_socket.close()
```

a method call that closes the socket connection.

Headers

```
HTTP/1.1 200 OK
Date: Sun, 15 Oct 2023 22:14:45 GMT
Server: Apache/2.4.18 (Ubuntu)
Last-Modified: Sat, 13 May 2017 11:22:22 GMT
ETag: "a7-54f6609245537"
Accept-Ranges: bytes
Content-Length: 167
Cache-Control: max-age=0, no-cache, no-store,
must-revalidate
Pragma: no-cache
Expires: Wed, 11 Jan 1984 05:00:00 GMT
Connection: close
Content-Type: text/plain
```

Data

```
But soft what light through yonder window breaks
It is the east and Juliet is the sun
Arise fair sun and kill the envious moon
Who is already sick and pale with grief
```

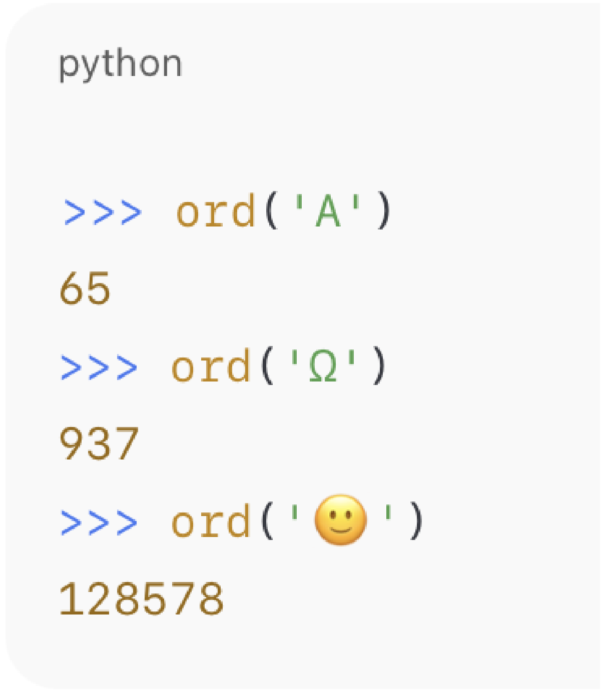
Unicode

- Unicode is a character encoding system that can represent characters from all languages in the world
- Unicode standard explicitly separates the identity of characters from specific byte representations:
  - Code point: is the identity of a character and is a number ranging from 0 to 1,114,111.
  - Code point is shown in the Unicode standard as 4 to 6 hex digits with a "U+" prefix
  - The code point for the letter A is U+0041, whereas the Euro sign is U+20AC
- The actual bytes that represent a character depend on the encoding in use.
- An encoding is an algorithm that converts code points to byte sequences and vice versa.
- A common character encoding system is **UTF-8** (Unicode Transformation Format – 8-bit). It represents all characters in the Unicode standard using 1 to 4 bytes per character.
- E.g., Code point for letter A (U+0041) is encoded as the single byte \x41 in the UTF-8 encoding, or as bytes \x41\x00 in UTF-16LE encoding
- Another example, UTF-8 requires three bytes - \xe2\x82\xac to encode the Euro sign, but in UTF-16LE the same code point is encoded as two bytes - \xac\x20
- ***Converting from code points to bytes is encoding, converting from bytes to code points is decoding.***

Unicode

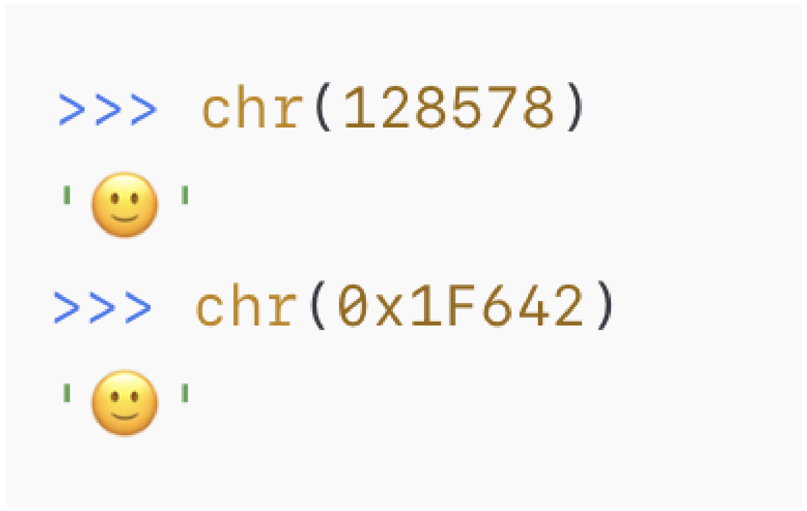
In Python, you can get the **Unicode code point** of a character using the built-in ord() function.

Example:



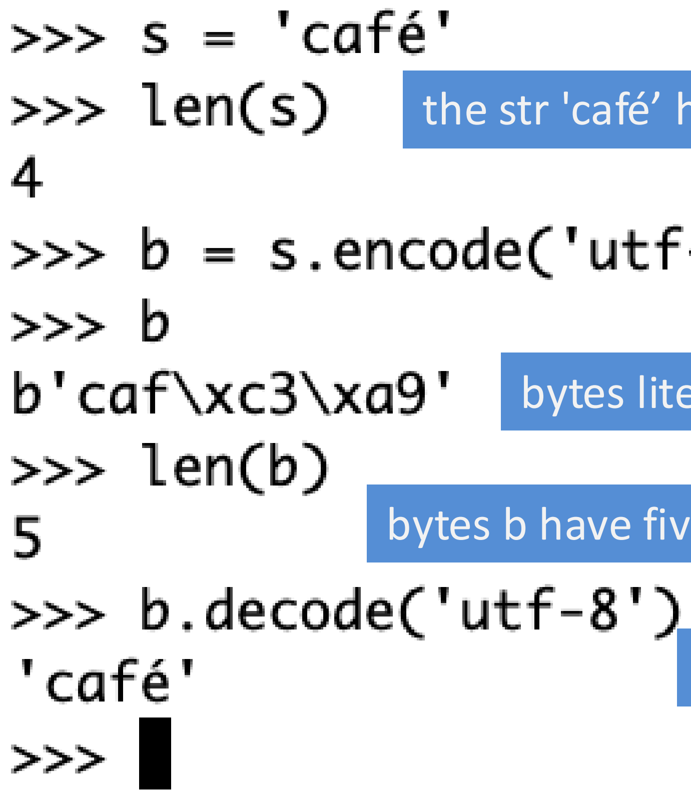
```python
>>> ord('A')
65
>>> ord('Ω')
937
>>> ord('🙂')
128578
```

The other way around is using chr():



```
>>> chr(128578)
'🙂'
>>> chr(0x1F642)
'🙂'
```

Unicode Encoding and Decoding



```
>>> s = 'café'
>>> len(s)
4
>>> b = s.encode('utf-8')
>>> b
b'caf\xc3\xa9'
>>> len(b)
5
>>> b.decode('utf-8')
'café'
>>> █
```

the str 'café’ has 4 Unicode characters

encode str to bytes using UTF-8 encoding

bytes literals have a b prefix

bytes b have five bytes, 1 for c a f and 2 bytes for é

decode bytes to str using UTF-8 encoding

An HTTP Request in Python

```python
import socket

my_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
my_socket.connect(('data.pr4e.org', 80))
cmd = 'GET http://data.pr4e.org/romeo.txt HTTP/1.0\r\n\r\n'.encode()
my_socket.send(cmd)

while True:
    data = my_socket.recv(512)
    if len(data) < 1:
        break
    print(data.decode(), end='')
my_socket.close()
```

When we talk to an external resource like a network socket we send bytes, so we need to encode Python 3 strings into a given character encoding

When we read data from an external resource, we must decode it based on the character set so it is properly represented in Python 3 as a string

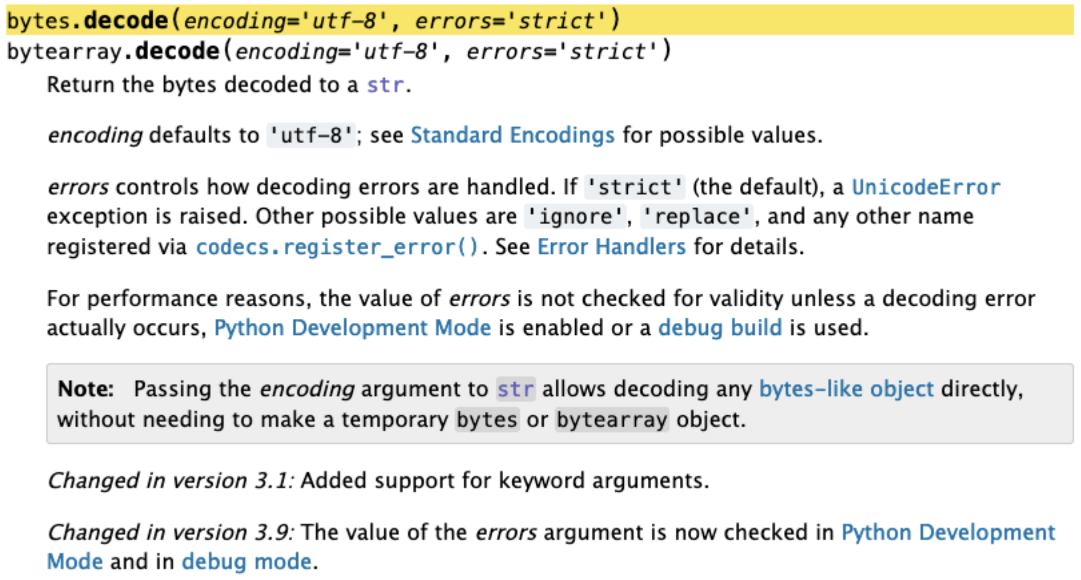
Headers

```
HTTP/1.1 200 OK
Date: Sun, 15 Oct 2023 22:14:45 GMT
Server: Apache/2.4.18 (Ubuntu)
Last-Modified: Sat, 13 May 2017 11:22:22 GMT
ETag: "a7-54f6609245537"
Accept-Ranges: bytes
Content-Length: 167
Cache-Control: max-age=0, no-cache, no-store, must-revalidate
Pragma: no-cache
Expires: Wed, 11 Jan 1984 05:00:00 GMT
Connection: close
Content-Type: text/plain
```

Data

```
But soft what light through yonder window breaks
It is the east and Juliet is the sun
Arise fair sun and kill the envious moon
Who is already sick and pale with grief
```

decode() **and** encode()



`bytes.`**`decode`**`(`*`encoding='utf-8', errors='strict'`*`)`
`bytearray.`**`decode`**`(`*`encoding='utf-8', errors='strict'`*`)`

Return the bytes decoded to a `str`.

encoding defaults to `'utf-8'`; see Standard Encodings for possible values.

errors controls how decoding errors are handled. If `'strict'` (the default), a `UnicodeError` exception is raised. Other possible values are `'ignore'`, `'replace'`, and any other name registered via `codecs.register_error()`. See Error Handlers for details.

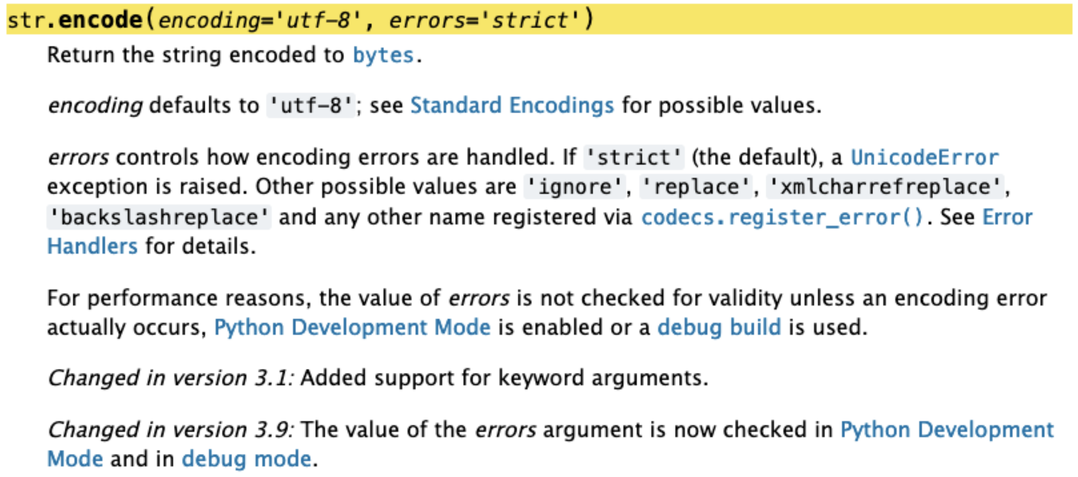
For performance reasons, the value of *errors* is not checked for validity unless a decoding error actually occurs, Python Development Mode is enabled or a debug build is used.

Note: Passing the *encoding* argument to `str` allows decoding any bytes-like object directly, without needing to make a temporary `bytes` or `bytearray` object.

Changed in version 3.1: Added support for keyword arguments.

Changed in version 3.9: The value of the *errors* argument is now checked in Python Development Mode and in debug mode.

https://docs.python.org/3/library/stdtypes.html#bytes.decode



`str.`**`encode`**`(`*`encoding='utf-8', errors='strict'`*`)`

Return the string encoded to `bytes`.

encoding defaults to `'utf-8'`; see Standard Encodings for possible values.

errors controls how encoding errors are handled. If `'strict'` (the default), a `UnicodeError` exception is raised. Other possible values are `'ignore'`, `'replace'`, `'xmlcharrefreplace'`, `'backslashreplace'` and any other name registered via `codecs.register_error()`. See Error Handlers for details.

For performance reasons, the value of *errors* is not checked for validity unless an encoding error actually occurs, Python Development Mode is enabled or a debug build is used.

Changed in version 3.1: Added support for keyword arguments.

Changed in version 3.9: The value of the *errors* argument is now checked in Python Development Mode and in debug mode.

https://docs.python.org/3/library/stdtypes.html#str.encode

Information Sciences Institute

USC Viterbi
School of Engineering

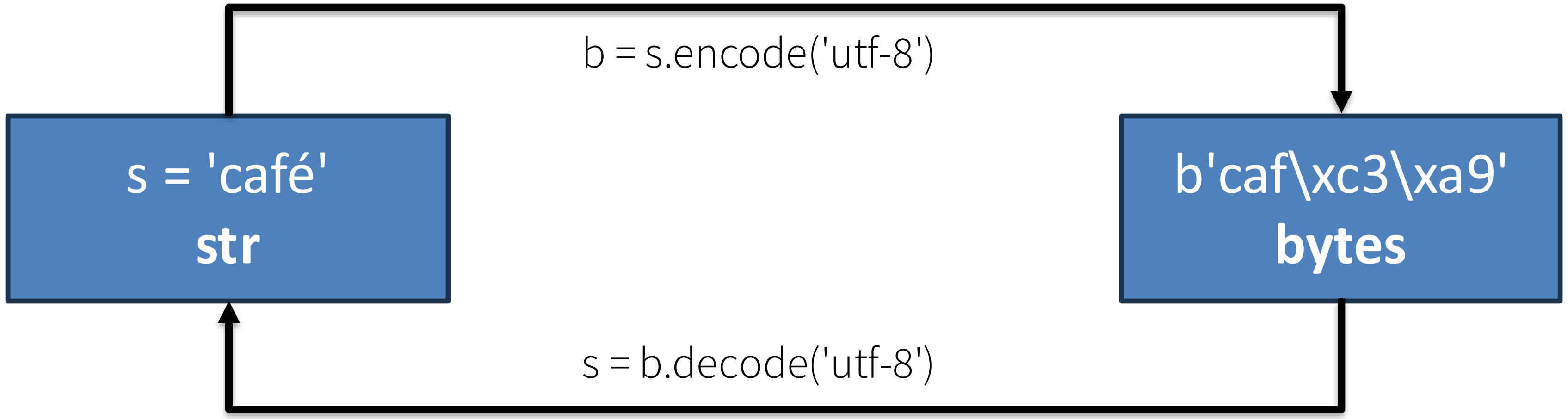
Python 3 and Unicode

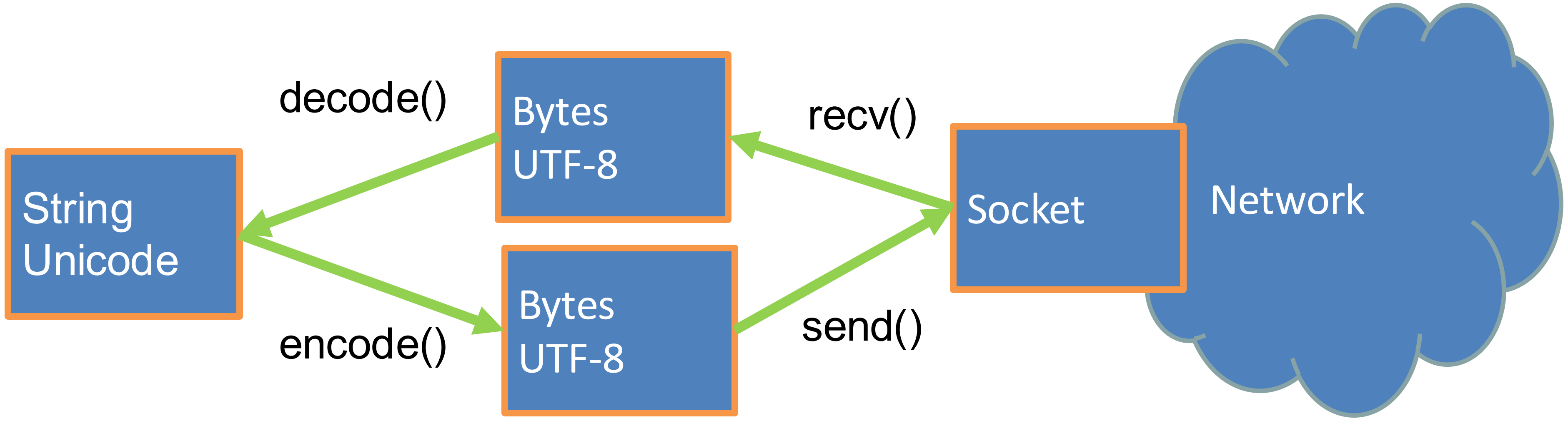
```
>>> x = b'abc'
>>> type(x)
<class 'bytes'>
>>> x = "🐍 : 你好, мир ,नमस्ते ,🌍!"
>>> type(x)
<class 'str'>
```

- In Python 3, all strings internally are Unicode.
- Working with string variables in Python programs and reading data from files usually "just works"
- When we talk to a network resource using sockets or talk to a database we have to encode and decode data (usually to UTF-8)

Python Data Type: bytes

- A **bit** is a binary digit that can be "0" or "1".
- A **byte** is a unit of digital information with 8 bits.
- It can represent $2^8 = 256$ different values.
- Python data type bytes is similar to data type str,
  - but with bytes instead of Unicode characters.





```
import socket

my_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
my_socket.connect(('data.pr4e.org', 80))
cmd = 'GET http://data.pr4e.org/romeo.txt HTTP/1.0\r\n\r\n'.encode()
my_socket.send(cmd)

while True:
    data = my_socket.recv(512)
    if len(data) < 1:
        break
    print(data.decode(), end='')
my_socket.close()
```

Requests: HTTP for Humans™

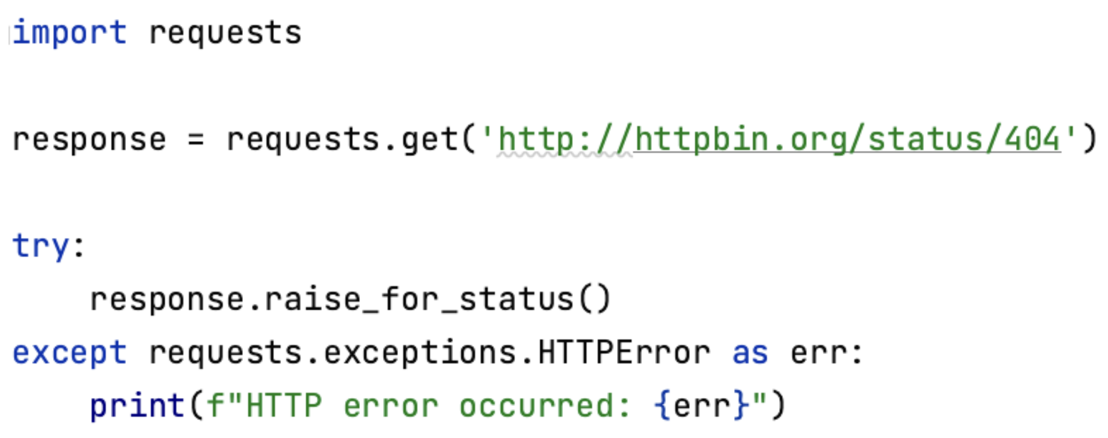
- Requests is an elegant and simple HTTP library for Python, built for human beings.
- Requests allows you to send HTTP/1.1 requests easily.
- No need to manually add query strings to yours URLs, or to form-encode your POST data.
- Keep-Alive & Connection Pooling.
- Automatic Content Decoding
- Basic/Digest Authentication
- And more: https://docs.python-requests.org/en/latest/index.html
- Installation:

  pip install requests

```
>>> r = requests.get('https://api.github.com/user', auth=('user', 'pass'))
>>> r.status_code
200
>>> r.headers['content-type']
'application/json; charset=utf8'
>>> r.encoding
'utf-8'
>>> r.text
'{"type":"User"...'
>>> r.json()
{'private_gists': 419, 'total_private_repos': 77, ...}
```

From now on, we will be using the requests **library for HTTP requests**

requests: raise_for_status()



```
import requests

response = requests.get('http://httpbin.org/status/404')

try:
    response.raise_for_status()
except requests.exceptions.HTTPError as err:
    print(f"HTTP error occurred: {err}")
```

```
HTTP error occurred: 404 Client Error: NOT FOUND for url:
http://httpbin.org/status/404
```

- raise_for_status() **is used to raise an exception if an HTTP request made using the requests library receives a non-successful (non-2xx) HTTP response code**
- **After making an HTTP request using** requests.get(), requests.post(), **etc., you can call** raise_for_status() **on the response object.**
- **If the status code is not in the 2xx range (indicating a successful request), it will raise an** HTTPError**.**

HTTP Error codes: 3-digit numbers grouped into 5 classes

- 1xx - Informational
  - Request received, continuing process
- 2xx - Success
  - The action was successfully received, understood, and accepted
- 3xx – Redirection
  - Further action must be taken to complete the request
- 4xx – Client Error
  - The request contains bad syntax or cannot be fulfilled
- 5xx – Server Error
  - The server failed to fulfill a valid request

Common Error codes

200 – OK : The request succeeded

201 – Created: The request succeeded, and a new resource was created as a result

204 – No Content: There is no content to send for this request, but the headers may be useful

301 – Moved Permanently: The URL of the requested resource has been changed permanently

400 - Bad Request: Client error, malformed request syntax, invalid request framing etc

403 –Forbidden: The client does not have access rights to the content

404 – Not Found: The server cannot find the requested resource

405 – Method Not Allowed: The requested method is known by the server but is not supported by target resource.

418 - I'm a teapot: The server refuses the attempt to brew coffee with a teapot

500 – Internal Server Error: The server has encountered an error

503– Service Unavailable: Server is not ready to handle the request

HTML & CSS: Languages used to structure and style webpages for display in web browsers.”

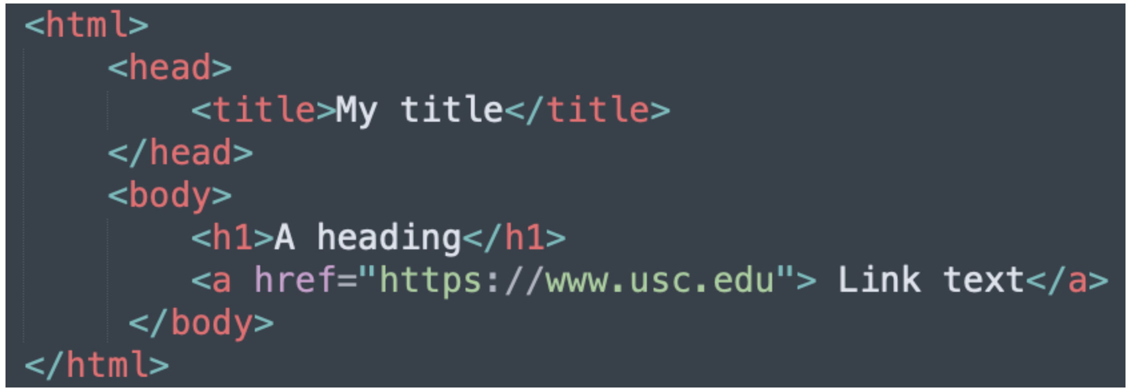
HTML (Hypertext Markup Language): language for describing structure of Web pages. You can:

- Publish online documents with headings, text, tables, lists, photos, etc.
- Retrieve online information via hypertext links, at the click of a button.
- Design forms for conducting transactions with remote services, for use in searching for information, making reservations, ordering products, etc.
- Include spread-sheets, video, sound clips, and other applications in their documents.
- With HTML, authors describe the structure of pages using *markup.* The *elements* of the language label pieces of content such as “paragraph,” “list,” “table,” and so on.

CSS (Cascading Style Sheets): language for describing the presentation of Web pages, including colors, layout, and fonts.

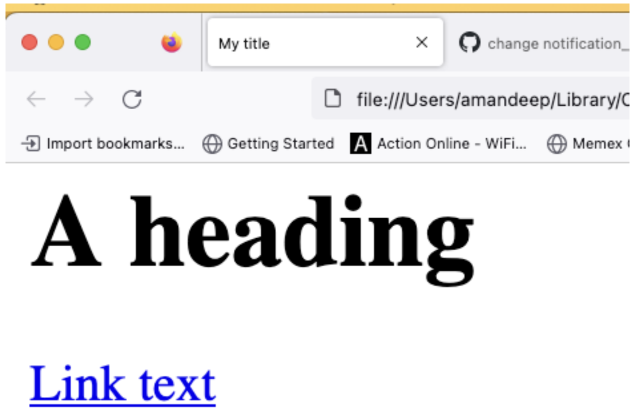
- Can adapt presentation to different devices: large screens, small screens, printers.
- Separation of HTML from CSS makes it easier to maintain sites, share style sheets.

Document Object Model (DOM)



```
<html>
    <head>
        <title>My title</title>
    </head>
    <body>
        <h1>A heading</h1>
        <a href="https://www.usc.edu"> Link text</a>
     </body>
</html>
```

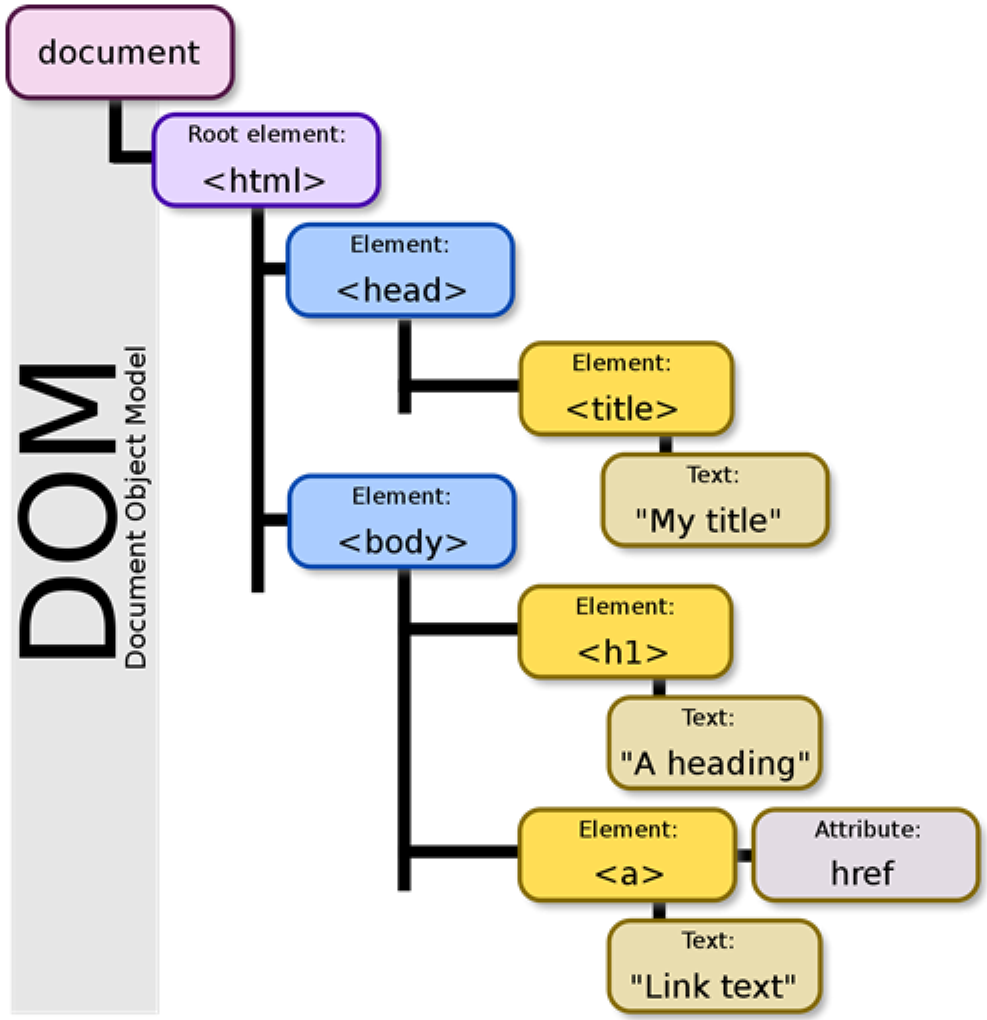
HTML



A heading

Link text

Browser View



HTML vs DOM

HTML	The *source code* of the page (text).
DOM	The *browser’s internal data structure* (object model) created from that HTML.

1. **HTML** is static — what the server sends.
2. **DOM** is dynamic — it changes as the browser interprets JavaScript or user actions.

Common HTML Tags

Document

<html>   Creates an HTML document

<head> Sets off the title & other info that isn't displayed

<body> Sets off the visible portion of the document

<title> Puts name of the document in the title bar; when bookmarking pages, this is what is bookmarked

Structure

<h1> … <h6> : Headings -- H1=largest, H6=smallest

<p> Creates a new paragraph

<div> Used to format block content with CSS

<span> Used to format inline content with CSS

<br> Inserts a line break

Lists

<ul> Creates an unordered list

<ol> Creates an ordered list

<li> each list item

Links

<a href="URL">clickable text</a> Creates a hyperlink to a Uniform Resource Locator

<a href="mailto:EMAIL_ADDRESS">clickable text</a> Creates a hyperlink to an email address

<a name="NAME"> Creates a target location within a document

<a href="#NAME">clickable text</a> Creates a link to that target location

Text Tags

<strong> Emphasizes a word (usually processed in bold)
<b> Creates bold text (should use <strong> instead)

<em> Emphasizes a word (usually processed in italics),
<I> Creates italicized text (should use <em> instead)

<pre> Creates preformatted text
<code> Used to define source code, usually monospace,
<tt> Creates typewriter-style text

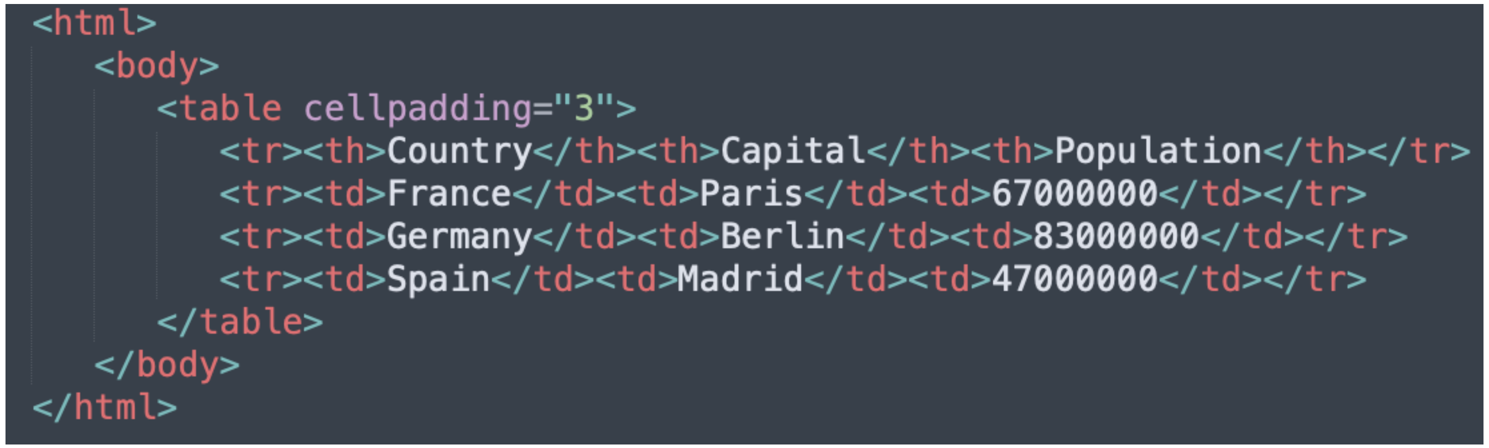
Graphical elements

<img src="URL" /> Adds image located at URL

<hr> Inserts a horizontal rule

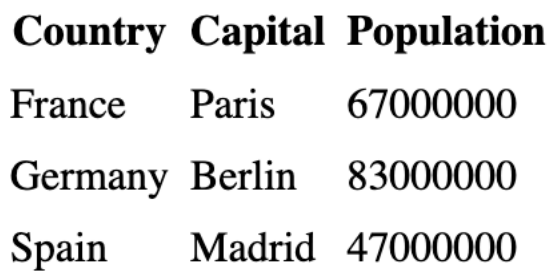
HTML Table Example

HTML



```
<html>
    <body>
        <table cellpadding="3">
            <tr><th>Country</th><th>Capital</th><th>Population</th></tr>
            <tr><td>France</td><td>Paris</td><td>67000000</td></tr>
            <tr><td>Germany</td><td>Berlin</td><td>83000000</td></tr>
            <tr><td>Spain</td><td>Madrid</td><td>47000000</td></tr>
        </table>
    </body>
</html>
```

Browser View



Country	Capital	Population
France	Paris	67000000
Germany	Berlin	83000000
Spain	Madrid	47000000

<tr> Table Row
<th> Table Header
<td> Table Data