Lecture4- HTTP API and Programming

NET 445 – Internet Programming

Web and HTTP

First, a review…

- *web page* consists of *objects*
- object can be HTML file, JPEG image, Java applet, audio file,…
- web page consists of *base HTML-file* which includes *several referenced objects*
- each object is addressable by a *URL,* e.g.,

`www.someschool.edu/someDept/pic.gif`

host name | path name

HTTP overview

HTTP: hypertext transfer protocol

- Web's application layer protocol
- client/server model
  - *client:* browser that requests, receives, (using HTTP protocol) and "displays" Web objects
  - *server:* Web server sends (using HTTP protocol) objects in response to requests

HTTP overview (continued)

uses TCP:

- client initiates TCP connection (creates socket) to server, port 80
- server accepts TCP connection from client
- HTTP messages (application-layer protocol messages) exchanged between browser (HTTP client) and Web server (HTTP server)
- TCP connection closed

HTTP is "stateless"

- server maintains no information about past client requests

aside

protocols that maintain "state" are complex!

- past history (state) must be maintained
- if server/client crashes, their views of "state" may be inconsistent, must be reconciled

HTTP connections

non-persistent HTTP

- at most one object sent over TCP connection
  - connection then closed
- downloading multiple objects required multiple connections

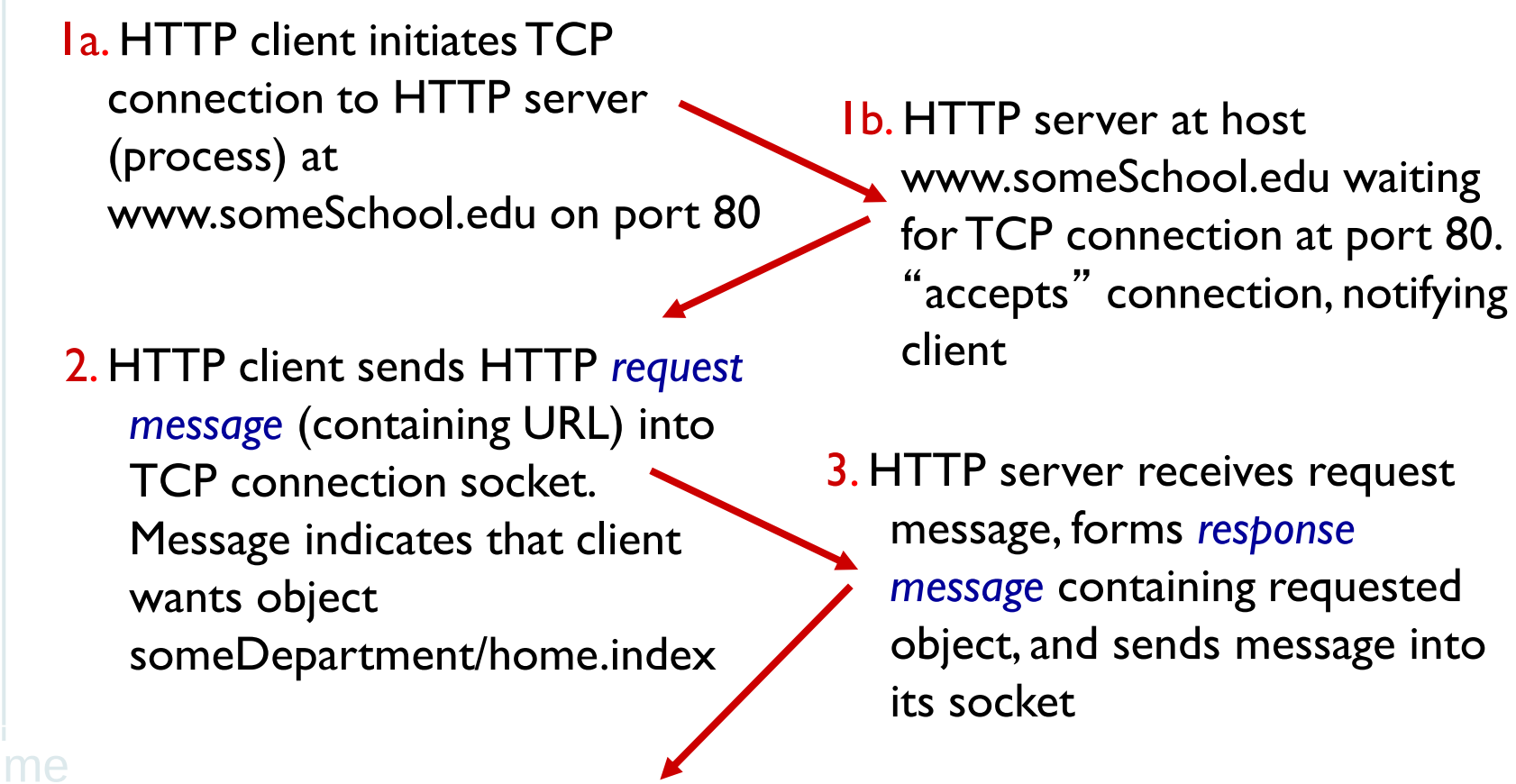
persistent HTTP

- multiple objects can be sent over single TCP connection between client, server

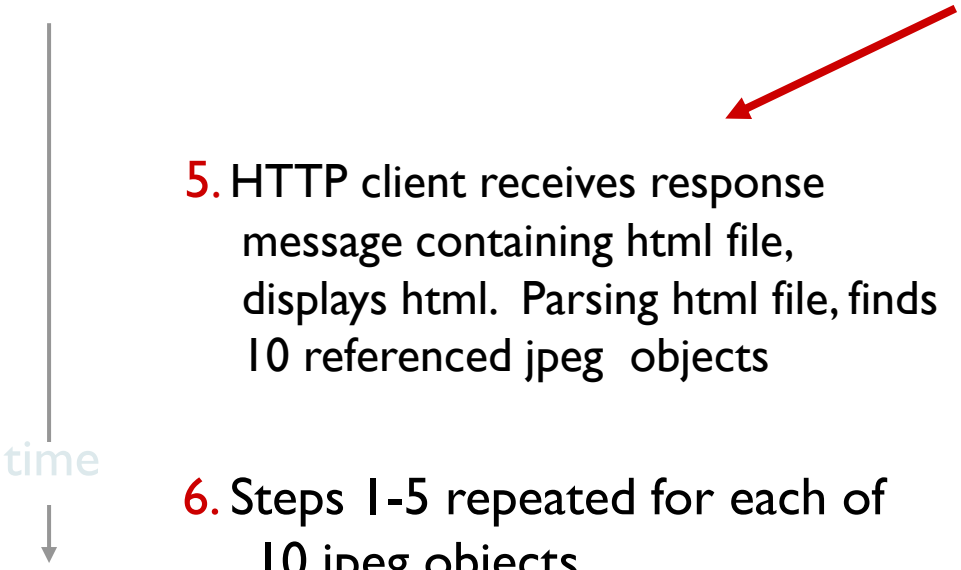
Non-persistent HTTP

suppose user enters URL:
`www.someSchool.edu/someDepartment/home.index`

(contains text, references to 10 jpeg images)



Non-persistent HTTP (cont.)



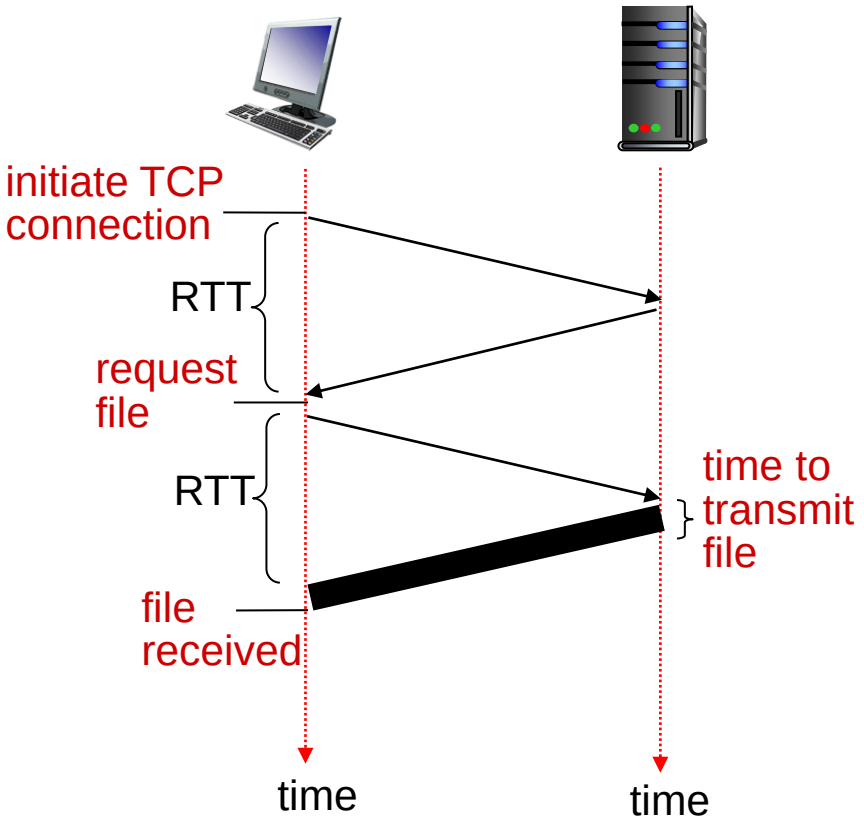
Non-persistent HTTP: response time

RTT (definition): time for a small packet to travel from client to server and back

HTTP response time:

- one RTT to initiate TCP connection
- one RTT for HTTP request and first few bytes of HTTP response to return
- file transmission time
- non-persistent HTTP response time =

  2RTT+ file transmission time



Persistent HTTP

non-persistent HTTP issues:

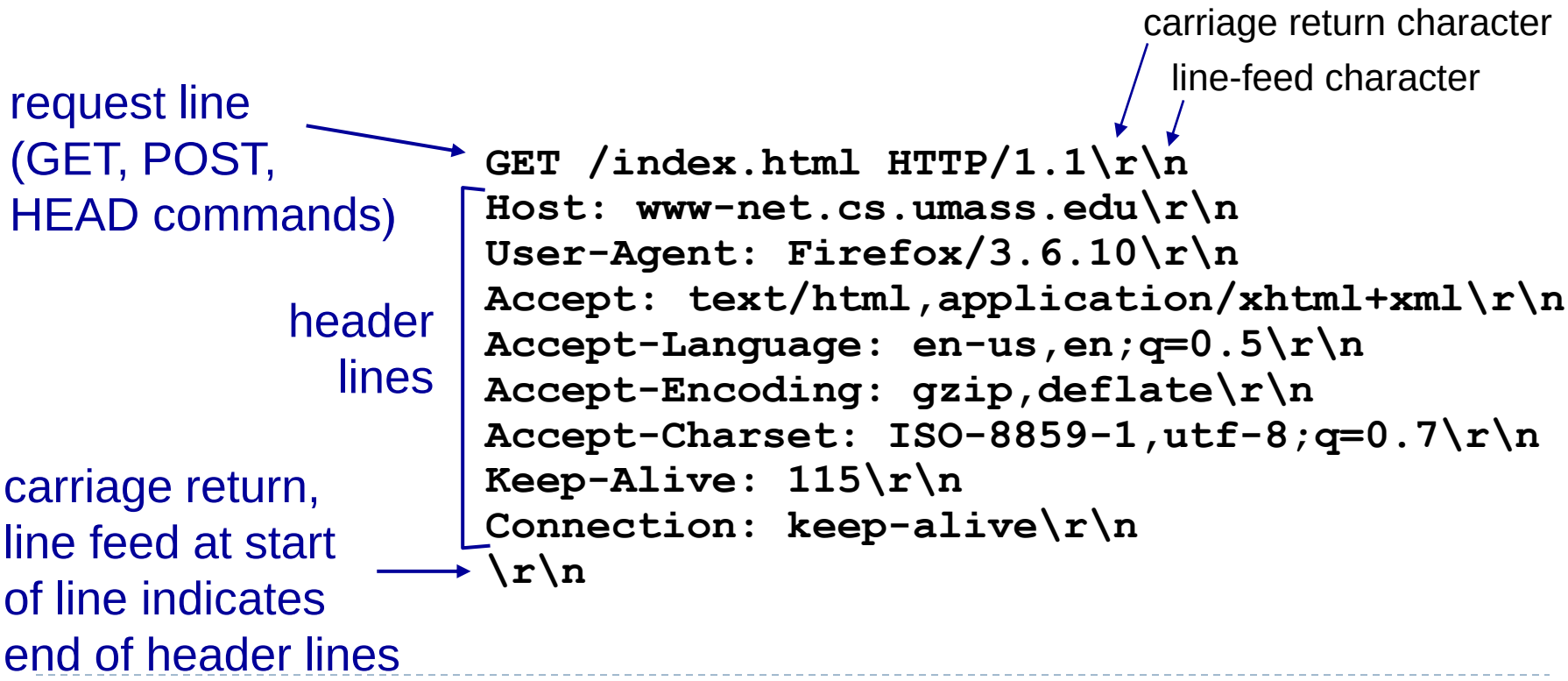
- requires 2 RTTs per object
- OS overhead for *each* TCP connection
- browsers often open parallel TCP connections to fetch referenced objects

persistent HTTP:

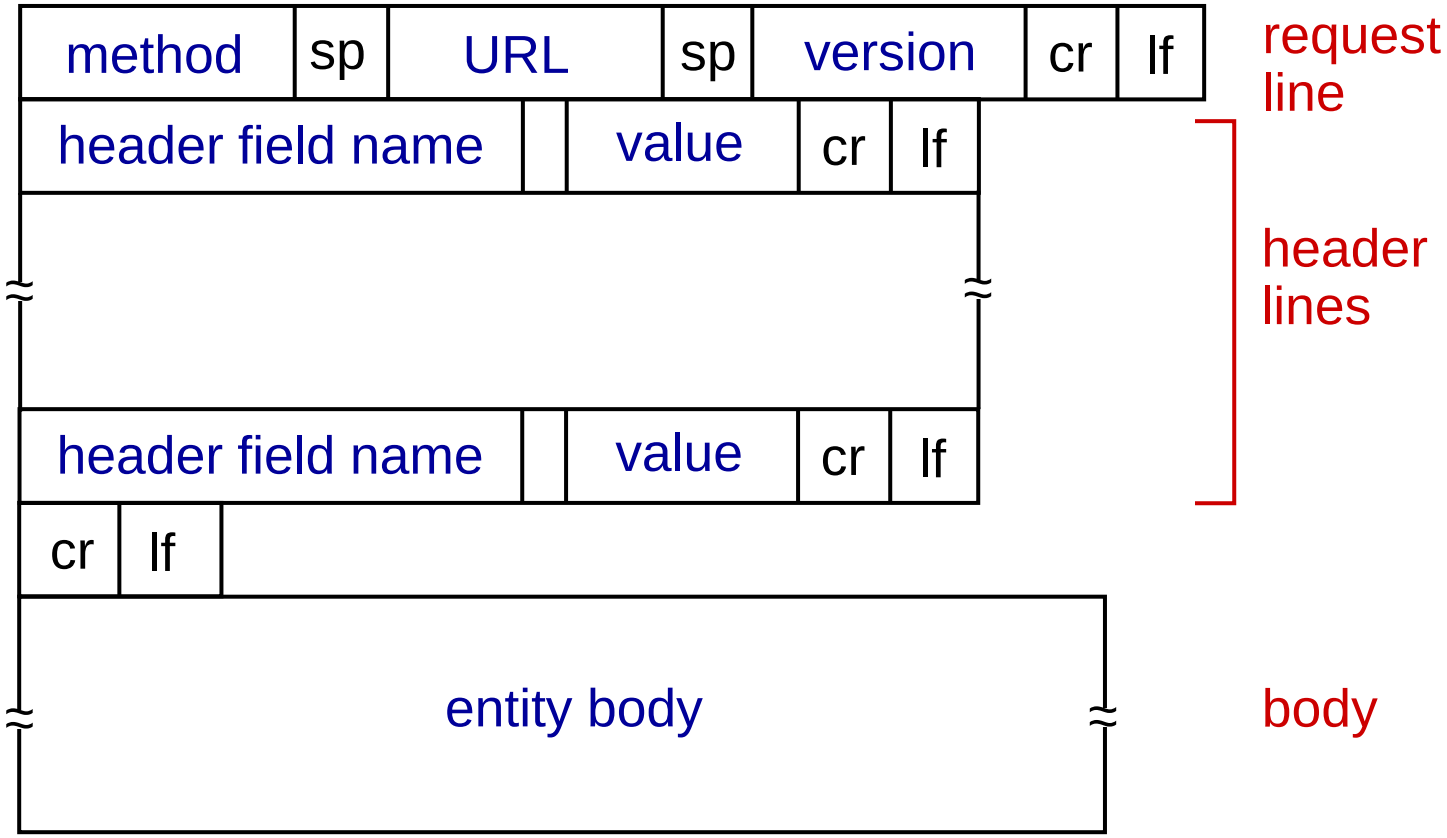
- server leaves connection open after sending response
- subsequent HTTP messages between same client/server sent over open connection
- client sends requests as soon as it encounters a referenced object
- as little as one RTT for all the referenced objects

HTTP request message

- two types of HTTP messages: *request, response*
- HTTP request message:
  - ASCII (human-readable format)



HTTP request message: general format



Uploading form input

POST method:

- web page often includes form input
- input is uploaded to server in entity body

URL method:

- uses GET method
- input is uploaded in URL field of request line:

```
www.somesite.com/animalsearch?monkeys&banana
```

Method types

HTTP/1.0:

- GET
- POST
- HEAD
  - asks server to leave requested object out of response

HTTP/1.1:

- GET, POST, HEAD
- PUT
  - uploads file in entity body to path specified in URL field
- DELETE
  - deletes file specified in the URL field

HTTP response message

status line (protocol status code status phrase)

header lines

data, e.g., requested HTML file

```
HTTP/1.1 200 OK\r\n
Date: Sun, 26 Sep 2010 20:09:20 GMT\r\n
Server: Apache/2.0.52 (CentOS)\r\n
Last-Modified: Tue, 30 Oct 2007 17:00:02
   GMT\r\n
ETag: "17dc6-a5c-bf716880"\r\n
Accept-Ranges: bytes\r\n
Content-Length: 2652\r\n
Keep-Alive: timeout=10, max=100\r\n
Connection: Keep-Alive\r\n
Content-Type: text/html; charset=ISO-8859-
   1\r\n
\r\n
data data data data data ...
```

HTTP response status codes

- status code appears in 1st line in server-to-client response message.
- some sample codes:

200 OK

- request succeeded, requested object later in this msg

301 Moved Permanently

- requested object moved, new location specified later in this msg (Location:)

400 Bad Request

- request msg not understood by server

404 Not Found

- requested document not found on this server

505 HTTP Version Not Supported

Trying out HTTP (client side) for yourself

1. Telnet to your favorite Web server:

```
telnet cis.poly.edu 80
```

opens TCP connection to port 80 (default HTTP server port) at cis.poly.edu. anything typed in sent to port 80 at cis.poly.edu

2. type in a GET HTTP request:

```
GET /~ross/ HTTP/1.1
Host: cis.poly.edu
```

by typing this in (hit carriage return twice), you send this minimal (but complete) GET request to HTTP server

3. look at response message sent by HTTP server!

(or use Wireshark to look at captured HTTP request/response)

Install HTTP requests python library

- Requests is a Python module that you can use to send all kinds of HTTP requests.
- It is an easy-to-use library with a lot of features ranging from passing parameters in URLs to sending custom headers and SSL Verification.
- Requests allow you to send HTTP/1.1 requests.
- You can add headers, form data, multi-part files, and parameters with simple Python dictionaries, and access the response data in the same way
- To install requests

```
pip3 install requests
```

The GET Request

- One of the most common HTTP methods is GET.
- The GET method indicates that you're trying to get or retrieve data from a specified resource.
- To make a GET request, invoke requests.get().
- To test this out, you can make a GET request to GitHub's Root REST API by calling get() with the following URL:

```
import requests
response = requests.get('https://api.github.com')
print (response.status_code )
```

Status Code

- The Status-Code element is a 3-digit integer where first digit of the Status-Code defines the class of response and the last two digits do not have any categorization role. There are 5 values for the first digit:

S.N.	Code and Description
1	**1xx: Informational** It means the request was received and the process is continuing.
2	**2xx: Success** It means the action was successfully received, understood, and accepted.
3	**3xx: Redirection** It means further action must be taken in order to complete the request.
4	**4xx: Client Error** It means the request contains incorrect syntax or cannot be fulfilled.
5	**5xx: Server Error** It means the server failed to fulfill an apparently valid request.

```
import requests
response = requests.get('https://api.github.com')

if response.status_code == 200:
    print('Success!')
elif response.status_code == 404:
    print('Not Found.')
```

Content

- The response of a GET request often has some valuable information, known as a payload, in the message body.
- Using the attributes and methods of Response, you can view the payload in a variety of different formats.
- To see the response's content in bytes, you use .content:

```
import requests
response = requests.get('https://api.github.com')
print (response.content)
```

Request in Text

- While .content gives you access to the raw bytes of the response payload, you will often want to convert them into a string using a character encoding such as UTF-8.
- response will do that for you when you access .text:

```
import requests
response = requests.get('https://api.github.com')
print (response.text)
```

Request in JSON

- f you take a look at the response, you'll see that it is actually serialized JSON content. To get a dictionary, you could take the str you retrieved from .text and deserialize
- A simple way to accomplish this task is to use .json():

```
import requests
response = requests.get('https://api.github.com')
print (response.json())
```

HTTP Headers

- The response headers can give you useful information, such as the content type of the response payload and a time limit on how long to cache the response.
- To view these headers, access .headers:

```
import requests
response = requests.get('https://api.github.com')
print (response.headers)
```

Query String Parameters

- One common way to customize a GET request is to pass values through query string parameters in the URL.
- To do this using get(), you pass data to params.
- For example, you can use GitHub's Search API to look for the requests library:

```
import requests
response = requests.get(
    'https://api.github.com/search/repositories',
    params={'q': 'requests+language:python'},
)

# Inspect some attributes of the `requests` repository
json_response = response.json()
repository = json_response['items'][0]
print(f'Repository name: {repository["name"]}’)
print(f'Repository description: {repository["description"]}')
```

Downloading images using HTTP requests

- You can download an image using HTTP request

```
import requests
r =
requests.get('https://identity.ksu.edu.sa/sites/identity.ksu.edu.sa/files/imce
_images/ksu_masterlogo_colour_rgb.png')
print(r.content)

f = open('logo.png', 'wb')
f.write(r.content)
f.close()
```

References:

- Foundations of Python Network Programming Third Edition by Brandon Rhodes (2014)
- James F. Kurose, and Keith W Ross, Computer Networking: A Top-Down Approach, 6th Edition
- Python 3 documentation
- https://wiki.python.org/moin/UdpCommunication
- https://www.w3schools.com/python/
- https://www.tutorialspoint.com/python/