Heuristics

Lab Section 7

For students with Programming background

- Scrape the site
- https://data.cityofnewyork.us/Health/DOHMH-New-York-City-Restaurant-Inspection-Results/43nn-pn8j
- and find all the restaurants in Brooklyn that have a critical problem (CRITICAL FLAG == Critical)
- If you don’t know about python-requests or BeautifulSoup, You can follow the next slides for some help.
- You can not use the selenium tool
- You need to also output the request string that you modify/used to get all rows.
- You can not download the data.

Today's Lab

- We will explore these libraries:
- A brief intro to client-server architecture
- Briefly discuss HTTP
- Web Scraping
- HTML
- Requests library
- Beautiful soup library
- Simple exercise

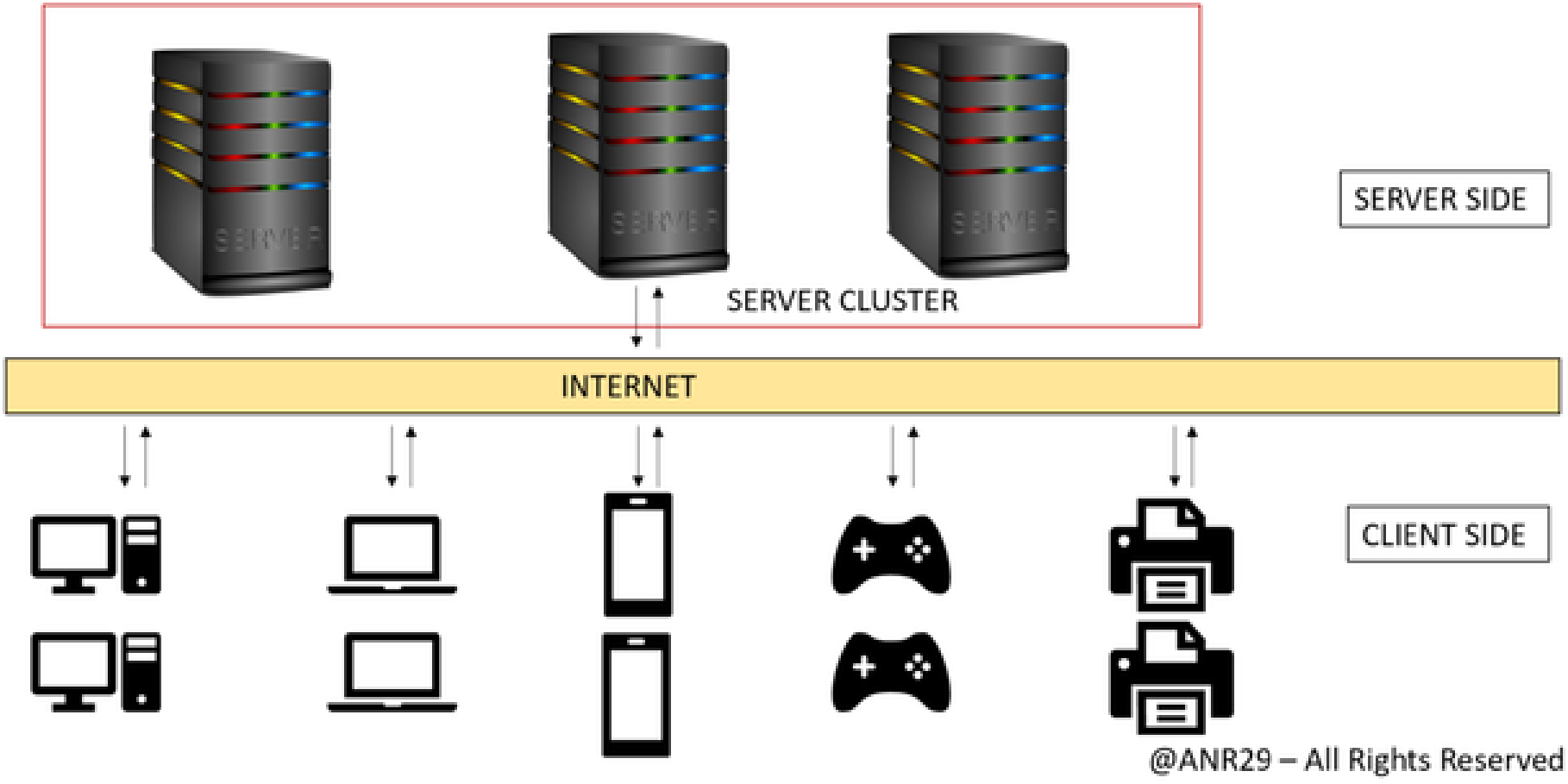
Introduction – Client-Server

- Client — Server architecture forms the basis of all of the internet and its associated services.
- Cloud service providers such as AWS, Azure etc and IoT devices are modern-day examples that primarily utilize the client-server architecture.
- Consists of two entities — Servers and Clients.

Client-Server

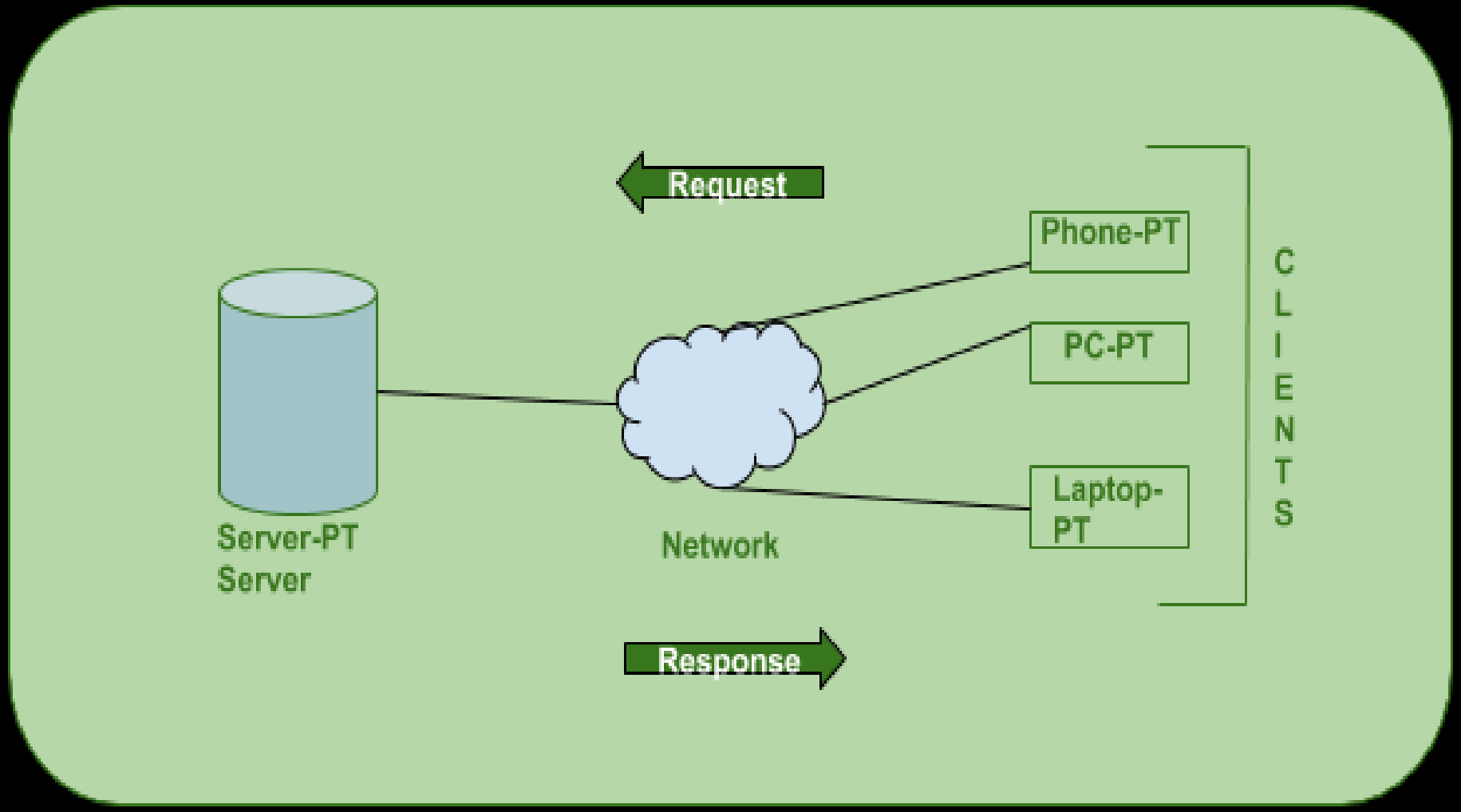
- **SERVER**
- A server is the entity offering the service.
- It could be any service — ranging in web hosting, processing, storage etc.
- **CLIENT**
- A client is the one receiving the service.
- A client is usually a recipient connected to the service over the internet.

Client – Server Architecture



Client-Server – Requests/Response

- Different protocols to do requests/response such as text HTTP or file storage FTP



What is HTTP?

- The Hypertext Transfer Protocol (HTTP) is a request/response protocol based on the client-server architecture for exchanging request and response messages.
- HTTP clients such as web browsers or mobile applications send requests to an HTTP server, and the server responds to them with messages containing a status line, a header, and a body
  - Status line or HTTP response status codes indicate whether a specific HTTP request has been successfully completed or not.
  - A response header is an HTTP header that can be used in an HTTP response and that doesn't relate to the content of the message. Response headers, like Age, Location or Server are used to give a more detailed context of the response.
  - HTTP Message Body is **the data bytes transmitted in an HTTP transaction message.**

Web Scraping

- The internet is an absolutely massive source of data — data that we can access using web scraping and Python!
- In fact, web scraping is often the *only* way we can access data.
- There is a lot of information out there that isn't available in convenient CSV exports or easy-to-connect APIs. And websites themselves are often valuable sources of data — consider, for example, the kinds of analysis you could do if you could download every post on a web forum.
- To access those sorts of on-page datasets, we'll have to use web scraping.
- Today's assignment is also on Web scraping.

Web Scraping – Web page components

- When we visit a web page, our web browser makes a request (HTTP request) to a web server.
  - This request is called a GET request (HTTP GET request), since we're getting page/files from the server.
- The server then sends back files that tell our browser how to render the page for us.
- These files will typically include:
- HTML — the main content of the page.
- CSS — used to add styling to make the page look nicer.
- JS — Javascript files add interactivity to web pages.
- Images — image formats, such as JPG and PNG, allow web pages to show pictures.

After our browser receives all the files, it renders the page and displays it to us.

When we perform web scraping, we're interested in the main content of the web page, so we look primarily at the HTML.

HTML

- HyperText Markup Language (HTML) is the language in which web pages are created in. HTML isn't a programming language, like Python, though. It's a **markup language** that tells a browser how to display content.
- HTML has many functions that are similar to what you might find in a word processor like Microsoft Word — it can make text bold, create paragraphs, and so on.
- HTML consists of elements called tags (opening <tag> and closing </tag>). The most basic tag is the <html> tag. This tag tells the web browser that everything inside of it is HTML.
- Right inside an HTML tag, we have two other tags: the head tag, and the body tag.
- The main content of the web page goes into the body tag. The head tag contains data about the title of the web page, and other information that generally isn't useful in web scraping:

Typical HTML structure of a Web page

- <html>
- <head>
- </head>
- <body>
- Some tags related to content:
  - p tag defines a paragraph.
  - a tag defines a hyperlink text.
    - The href property of the tag determines where the link goes.
  - div — indicates a division, or area, of the page.
  - b — bolds any text inside.
  - i — italicizes any text inside.
  - table — creates a table.
  - form — creates an input form.
- </body>
- </html>
- Here is the full list of tags: https://developer.mozilla.org/en-US/docs/Web/HTML/Element
- The tags elements also have relations (p arent,child sibling). with each other, html is the parent of body, body and head are siblings and so on.

HTML structure – Classes and IDs

- Before we move into actual web scraping, let’s learn about the class and id properties. These special properties give HTML elements names, and make them easier to interact with when we’re scraping.
- One element can have multiple classes, and a class can be shared between elements. Each element can only have one id, and an id can only be used once on a page. Classes and ids are optional, and not all elements will have them.
- Here is a paragraph with class and link with ID:
-
- Learn Data Science Online
-

Python requests library

- The first thing we'll need to do to scrape a web page is to download the page. We can download pages using the Python requests library.
- The Requests library provides a simple API for interacting with HTTP operations such as GET.
- The requests library will make a GET request to a web server, which will download the HTML contents of a given web page for us
  - The methods implemented in the Requests library execute HTTP operations(GET) against a specific web server specified by its URL
- To install the requests library:
  - pip install requests -- in windows
  - pip3 install requests -- in mac

Using GET Request

- We use the get method to request data from a specific web server. Let's try it out:
  - import requests
  - url = "https://dataquestio.github.io/web-scraping-pages/simple.html"
  - page = requests.get(url)
  - print('Response Code:', page.status_code)
  - print('Response Headers:\n', page.headers)
  - print('Response Content:\n', page.content)
- Running the code above outputs a status code of 200, which indicates that the URL is reachable. Then it returns the page's header data followed by the page's content (page HTML).
  - We won't fully dive into status codes here, but a status code starting with a 2 generally indicates success and a code starting with a 4 or a 5 indicates an error.
  - We will also not go into headers as they are mostly unrelated to the content

BeautifulSoup Library

- We can parse the content ourselves, but then we have to:
  - Create a structure that will enable the parsing of content in a systematic fashion, such as:
    - finding all paragraph
    - Finding the parent of the second paragraph
    - Finding all tags with specific class or id
  - To save us from this hassle, we have BeautifulSoup library – which creates a parse tree for parsed pages that can be used to extract data from HTML in a systematic fashion.
  - To install it, use:
    - pip install beautifulsoup4
    - pip3 install beautifulsoup4

BeautifulSoup Library – methods/use

- To use the BeautifulSoup, we first have to import the library, and create an instance of the BeautifulSoup class (object) to parse our document:
  - from bs4 import BeautifulSoup
  - soup = BeautifulSoup(page.content, 'html.parser’)
- Now you can perform actions on soup object by calling methods on it
- print out the HTML content of the page, formatted nicely, using the prettify method on the BeautifulSoup object.
  - print(soup.prettify())
- As all the tags are nested, we can move through the structure one level at a time. We can first select all the elements at the top level of the page using the **children** property of soup.

BeautifulSoup Library – methods/use

- Get the paragraph text in this page: https://dataquestio.github.io/web-scraping-pages/simple.html using children
- children returns a list generator, so we need to call the list function on it
- html = list(soup.children)[2]
  - Ignore the first two elements as they are not tag objects
  - The third is a Tag object, which contains other nested tags.
- To select the item with the relevant tag, we could print the children of the object at hand and find the location of the tag
- body = list(html.children)[3]
- p = list(body.children)[1]
  - Once we've isolated the tag, we can use the get_text method to extract all of the text inside the tag: print(p.get_text())

BeautifulSoup Library – methods advanced

- What we did above was useful for figuring out how to navigate a page, but it took a lot of commands to do something fairly simple.
- Finding all instances of a tag at once.
  - If we want to extract a single tag, we can instead use the find_all method, which will find all the instances of a tag on a page.
  - Note that find_all returns a list, so we'll have to loop through, or use list indexing, it to extract text.
  - soup.find_all('p')[0].get_text()
  - If we only want to find the first instance of a tag, we can use the find method, which will return a single BeautifulSoup object:
    - soup.find('p')

BeautifulSoup Library – methods advanced

- Searching for tags by class and id
- soup.find_all('p', class_='outer-text’)
  - Find all p tag elements with class of outer-text
- soup.find_all(class_="outer-text")
  - Find all elements with class of outer-text
- soup.find_all(id="first")
  - Find all elements with id of first

BeautifulSoup Library – select

- **Using CSS Selectors (pattern of elements and other terms)**

We can also search for items using CSS selectors. These selectors are how the CSS language allows developers to specify HTML tags to style. Here are some examples:

  - p a — finds all a tags inside of a p tag.
  - body p a — finds all a tags inside of a p tag inside of a body tag.
  - html body — finds all body tags inside of an html tag.
  - p.outer-text — finds all p tags with a class of outer-text.
  - p#first — finds all p tags with an id of first.
  - body p.outer-text — finds any p tags with a class of outer-text inside of a body tag.

- soup.select("div p")
- Note that the select method above returns a list of BeautifulSoup objects, just like find and find_all. What does it mean?
  - Mean on each item we can call all beautiful soup methods.

Exercise for Programming Novice

- **Parse this website:** https://weather.com/weather/tenday/l/New+York+NY+USNY0996:1:US
- And print Today's day or night temperature.
- Also, print the weather describing text.

End