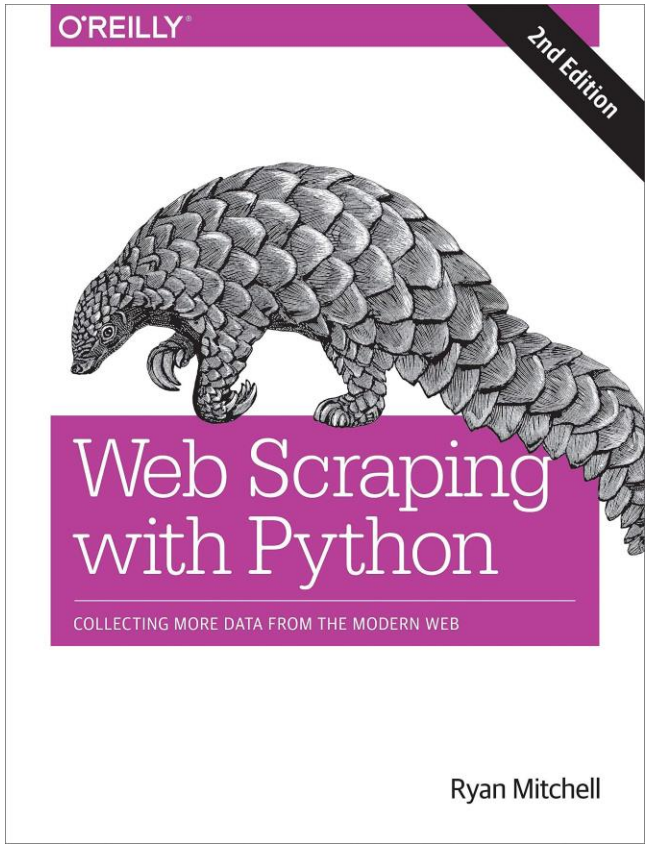
INFO 215: Web Science.
Lecture 01: Web Architecture, Client side technologies: XML, HTML, XPath

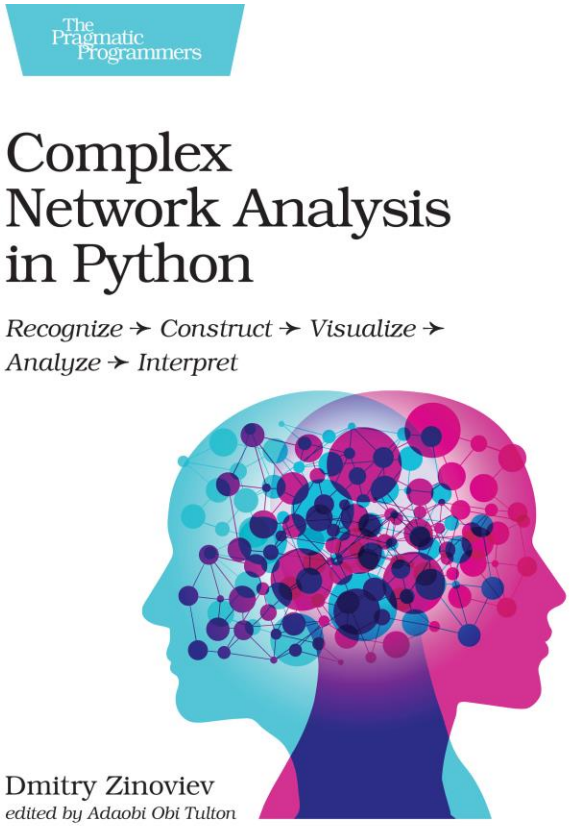
- *Fazle Rabbi, Ph.D*
- *Assoc. Prof in Information Science*
- *University of Bergen*
- *Fazle.Rabbi@uib.no*

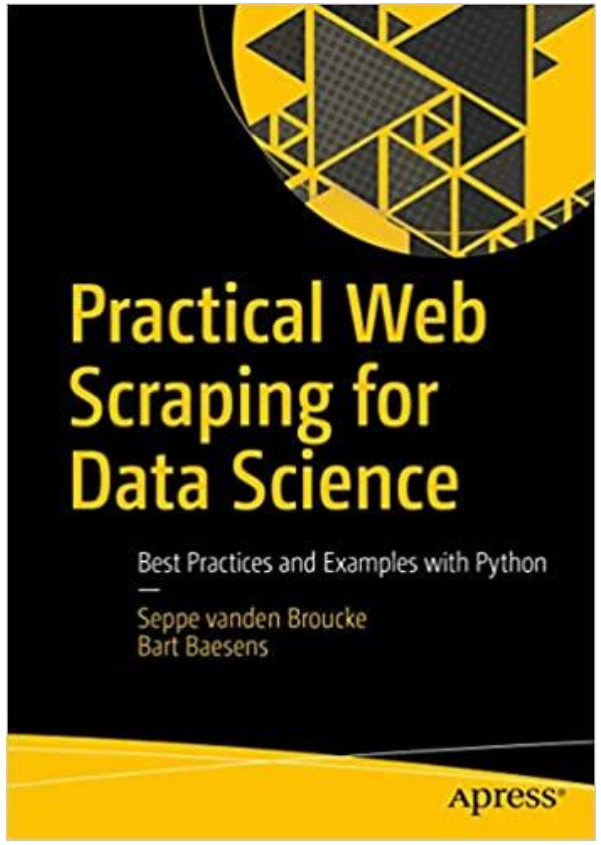
- *January 16, 2021, Bergen, Norway*

Agenda/Topics of INFO215

Web server development (Django)	HTML/XML	XPath/Regex
Web Scraping using Python	Natural Language Processing	Semantic web
Constructing and measuring networks using NetworkX	Visualization of network using Gephi	Mining the Social Web


O'REILLY
2nd Edition
Web Scraping with Python
COLLECTING MORE DATA FROM THE MODERN WEB
Ryan Mitchell


The Pragmatic Programmers
Complex Network Analysis in Python
Recognize → Construct → Visualize → Analyze → Interpret
Dmitry Zinoviev
edited by Adaobi Obi Tulton


Practical Web Scraping for Data Science
Best Practices and Examples with Python
Seppe vanden Broucke
Bart Baesens
Apress

Literature

Programming Environment/Tools/IDEs

- Python 3, XML, HTML, JavaScript
- Jupyter notebook
- Sublime Text
- Gephi
- Google chrome
- Postman
- Command prompt/Terminal

Evaluation

Summative assessments

- 6 mandatory assignments
- Written exam (4 hours) at the end of the semester

Formative assessments

- Lab assignments/Homeworks
- Practice Quiz

In this lecture we cover the following

- Internet basics
  - Packet switching, TCP/IP, DNS
  - OSI model
  - Client-server architecture
- Web based technologies
  - Client-side web technologies
    - HTML Basics
    - XPath

Internet Basics

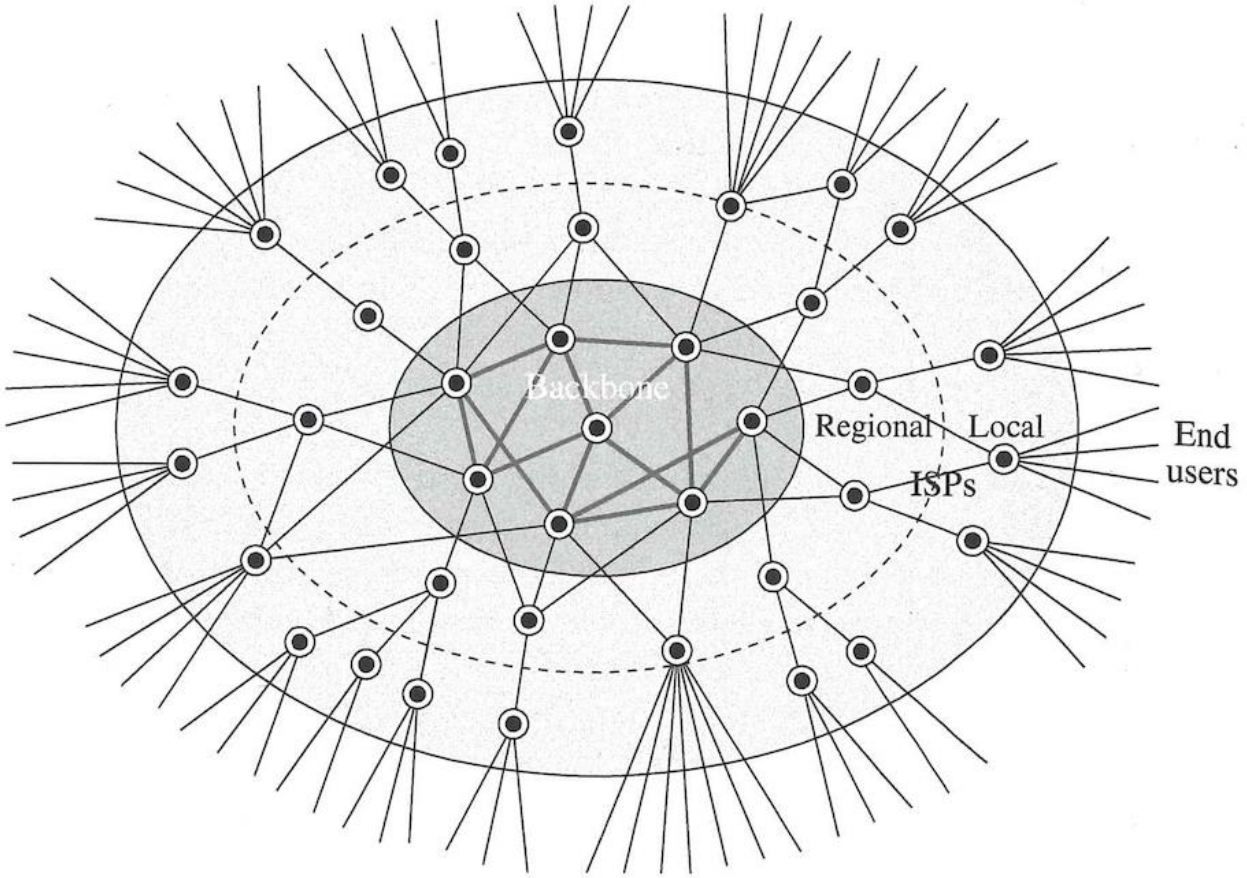


Figure 2.1: A schematic depiction of the structure of the Internet. The nodes and edges of the Internet fall into a number of different classes: the backbone of high-bandwidth long-distance connections; the ISPs, who connect to the backbone and who are divided roughly into regional (larger) and local (smaller) ISPs; and the end users—home users, companies, and so forth—who connect to the ISPs.

The Web Speaks HTTP

Client computer

- Most interactions over the Internet use the *client/server interaction* protocol:
  - When you click a Web link, your computer gets the page for you...beginning the client/server interaction
    - Your computer is the *client* computer and the computer with the Web page is the *server* (**Web server**)
    - The *client,* gets services from the *server*
  - When the page is return, the operation is completed and the client/server relationship ends

Web Server

The Web Speaks HTTP

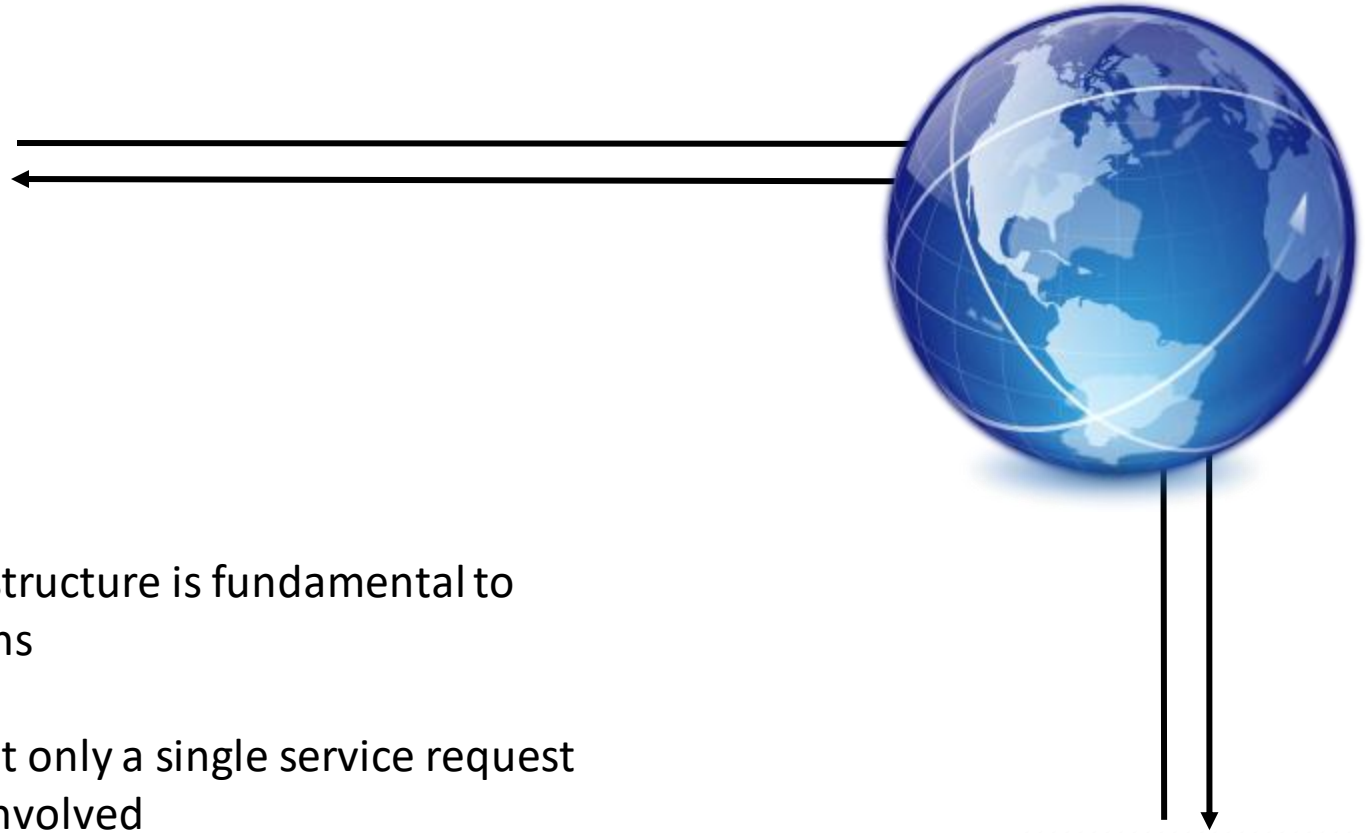
Client computer

Web Server

- The following series of steps that gets executed by your web browser once you navigate to a website, say www.google.com:
- You enter "www.google.com" into your web browser, which needs to figure out the IP address for this site.
- Your browser sets off to figure out the correct IP address behind "www.google.com". To do so, your web browser will use another protocol, called DNS (which stands for Domain Name System)
- Once the IP address behind www.google.com, 172.217.17.68 is found, your browser can now establish a connection to 172.217.17.68, Google's web server.
- The TCP/IP (Transmission Control Protocol) is used to deliver network messages. TCP/IP provides a general, reliable means to deliver network messages, as it includes functionality for error checking and splitting messages up in smaller packets.
- Inside the TCP message, we find another message, formatted according to the HTTP protocol (Hypertext Transfer Protocol)
- Google's web server now sends back HTTP reply, containing the contents of the page we want to visit.

Basic client server interaction

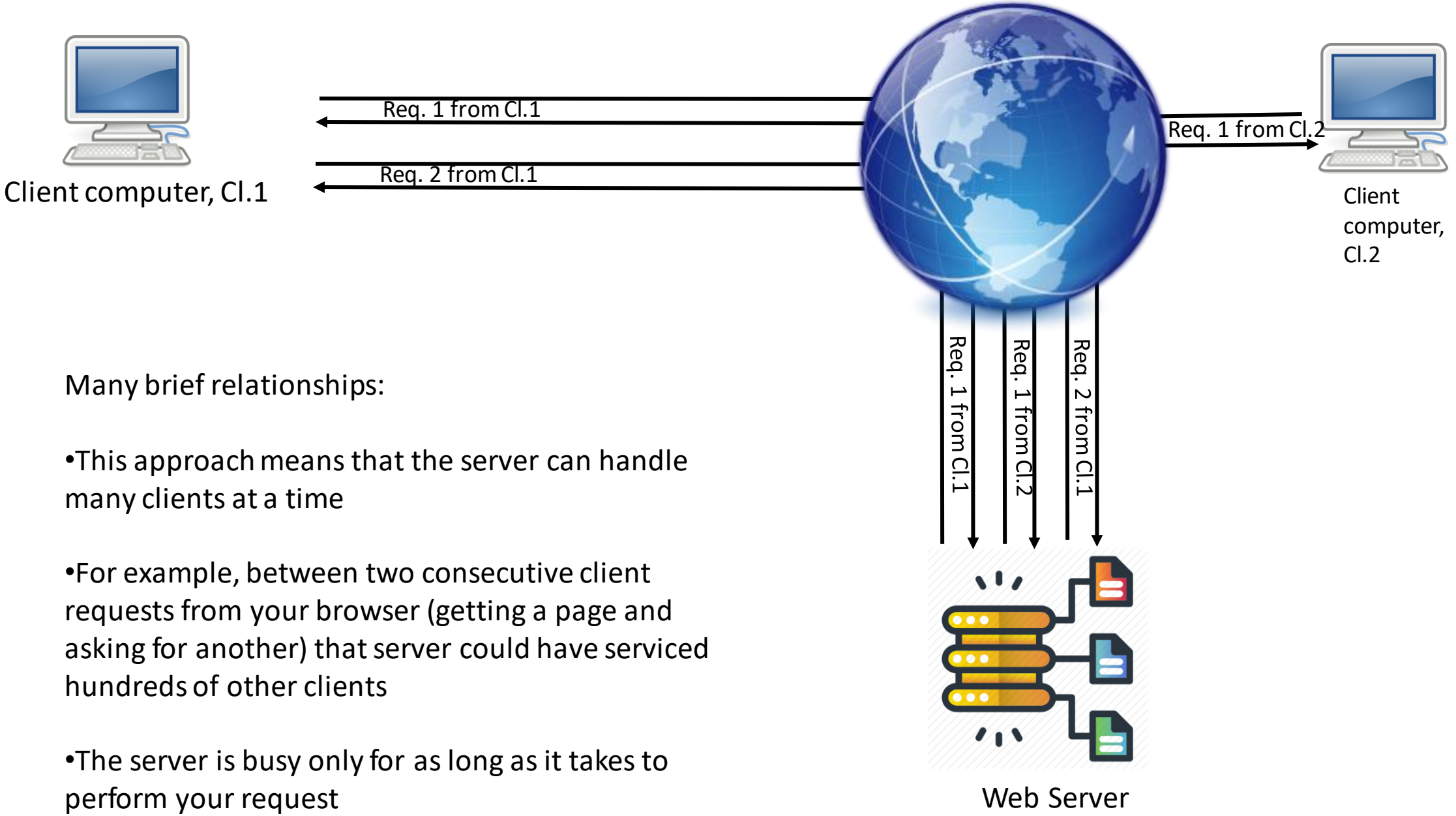
Client computer



- The client/server structure is fundamental to Internet interactions
- A key aspect is that only a single service request and response are involved
- The *relationship* is very brief relationship, lasting from the moment the request is sent to the moment the service is received

Web Server

Basic client server interaction



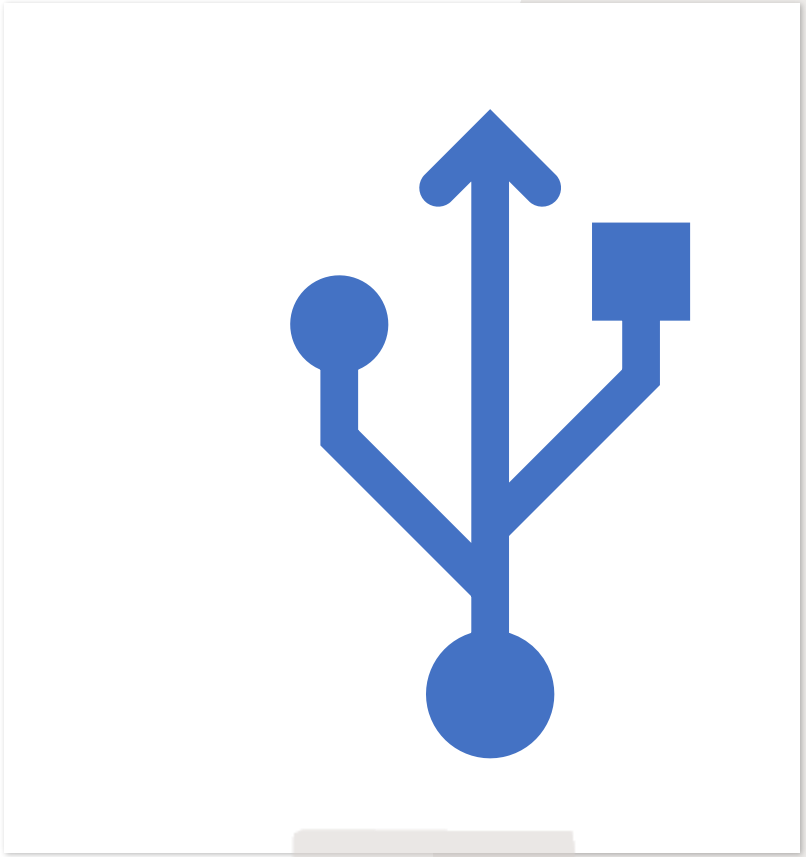
Many brief relationships:

- This approach means that the server can handle many clients at a time
- For example, between two consecutive client requests from your browser (getting a page and asking for another) that server could have serviced hundreds of other clients
- The server is busy only for as long as it takes to perform your request

Open Systems Interconnection (OSI) model

- The Open Systems Interconnection (OSI) model is a conceptual model created by the International Organization for Standardization which enables diverse communication systems to communicate using standard protocols.
- The OSI provides a standard for different computer systems to be able to communicate with each other.
- The OSI model organizes computer communication into seven layers:

- Layer7: Application Layer: HTTP, FTP and DNS, for instance.
- Layer6: Presentation Layer: Includes protocols to format and translate data. ASCII, SSL, for instance
- Layer5: Session Layer: Includes protocols for opening/closing and managing sessions.
- Layer4: Transport Layer: TCP, but also protocols such as UDP, which do not offer the advanced error checking and recovery mechanisms of TCP for sake of speed.
- Layer3: Network Layer: Includes IP (Internet Protocol).
- Layer2: Data link Layer: Includes the Ethernet protocol.
- Layer1: Physical Layer: Includes the Ethernet protocol, but also USB, Bluetooth, and other radio protocols



Internet vs. WWW

- The internet is a vast, international network, made up of computers and the physical connections (wires, routers, etc) allowing them to communicate.
- The World Wide Web (WWW) is a collection of software that spans the internet and enables the interlinking of documents and resources. The WWW provides a way to access information on the internet.

[https://en.wikipedia.org/wiki/Internet]

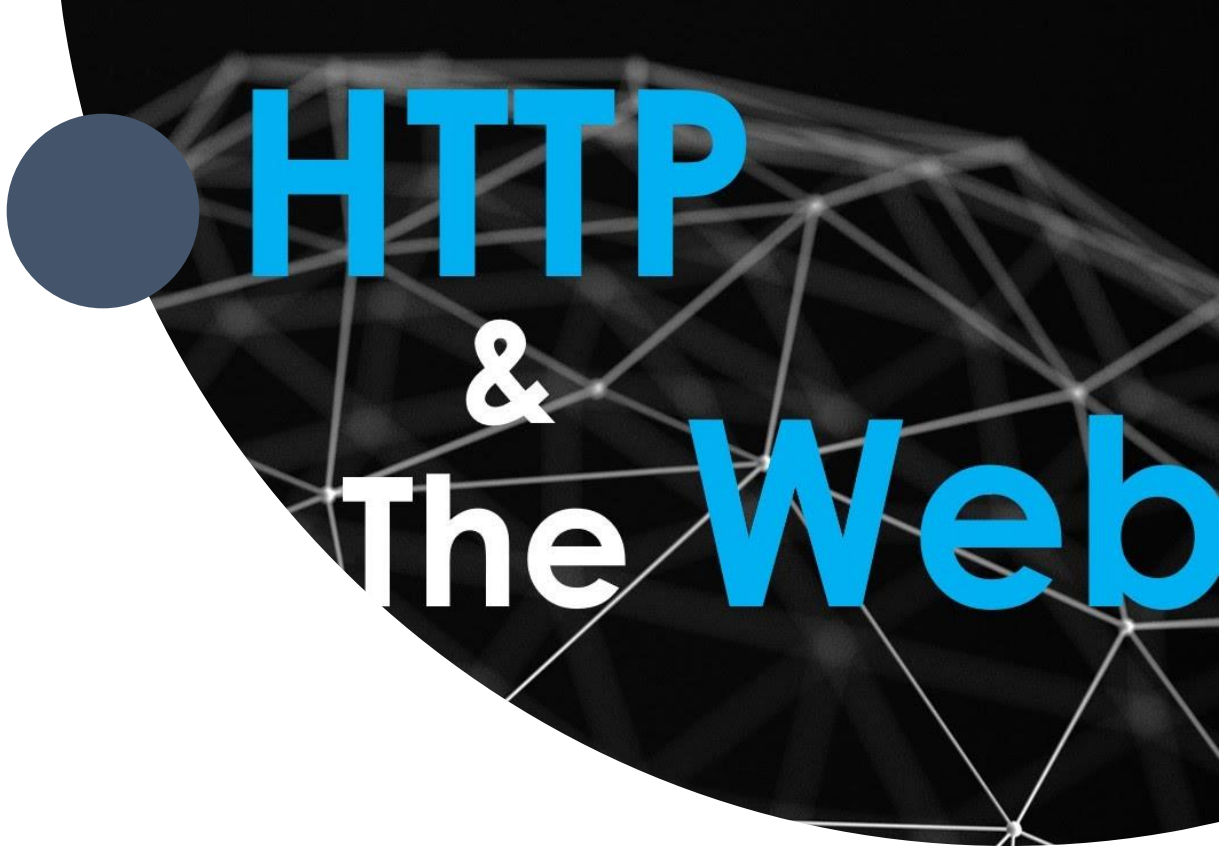
Requesting a web page

http://www.cs.washington.edu/homes/snyder/index.html

The **URL** has three main parts:

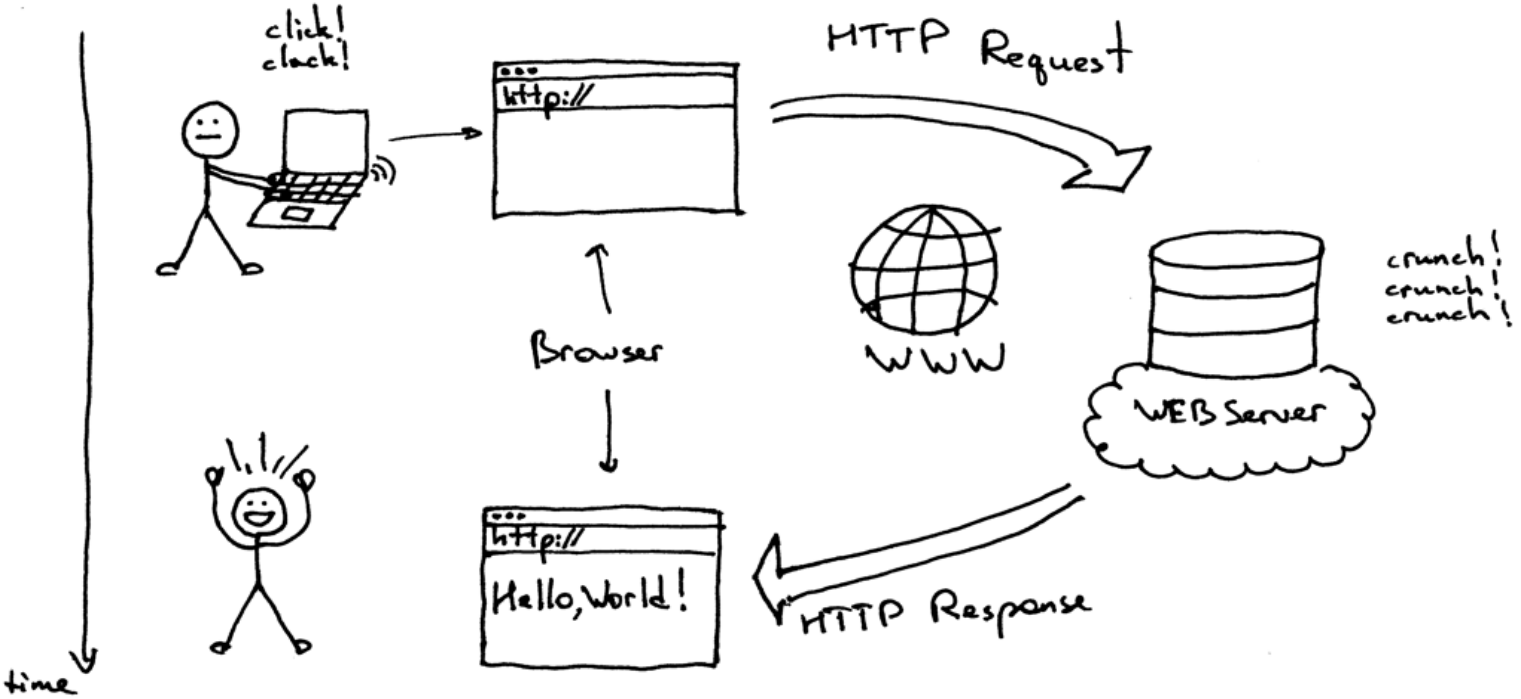
1. **Protocol:** tells the computers how to handle the file
2. **Server computer's name:** or the name given by the domain hierarchy
3. **Page's pathname:** tells the server which file (page) is requested and where to find it

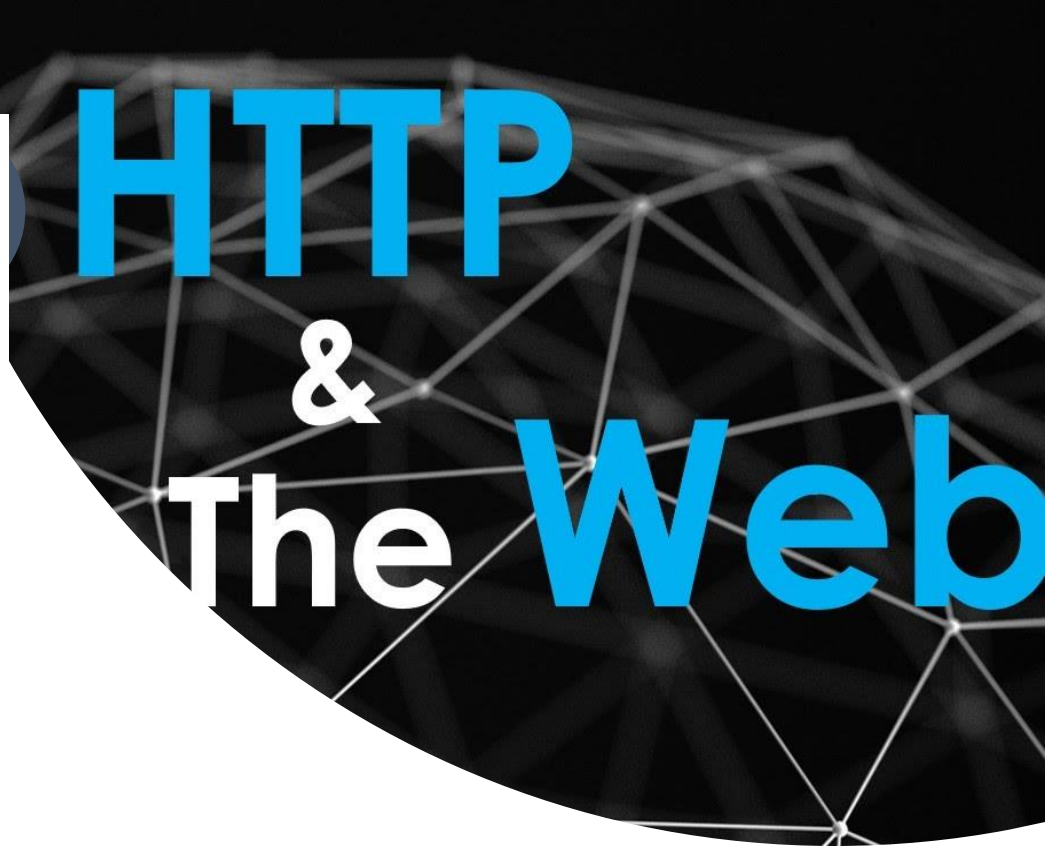
Hypertext Transfer Protocol



A Web page consist of objects
Object can be HTML file, image, audio clip, video clip, java applet
A web page consists of a base HTML file which includes a collection of referenced objects
Each object is accessed by a Uniform Resource Locator (URL)
Example:
https://www.uib.no/sites/w3.uib.no/files/styles/content_main_wide_1x/public/media/naturhistorisk_leopard.jpg

Hypertext Transfer Protocol



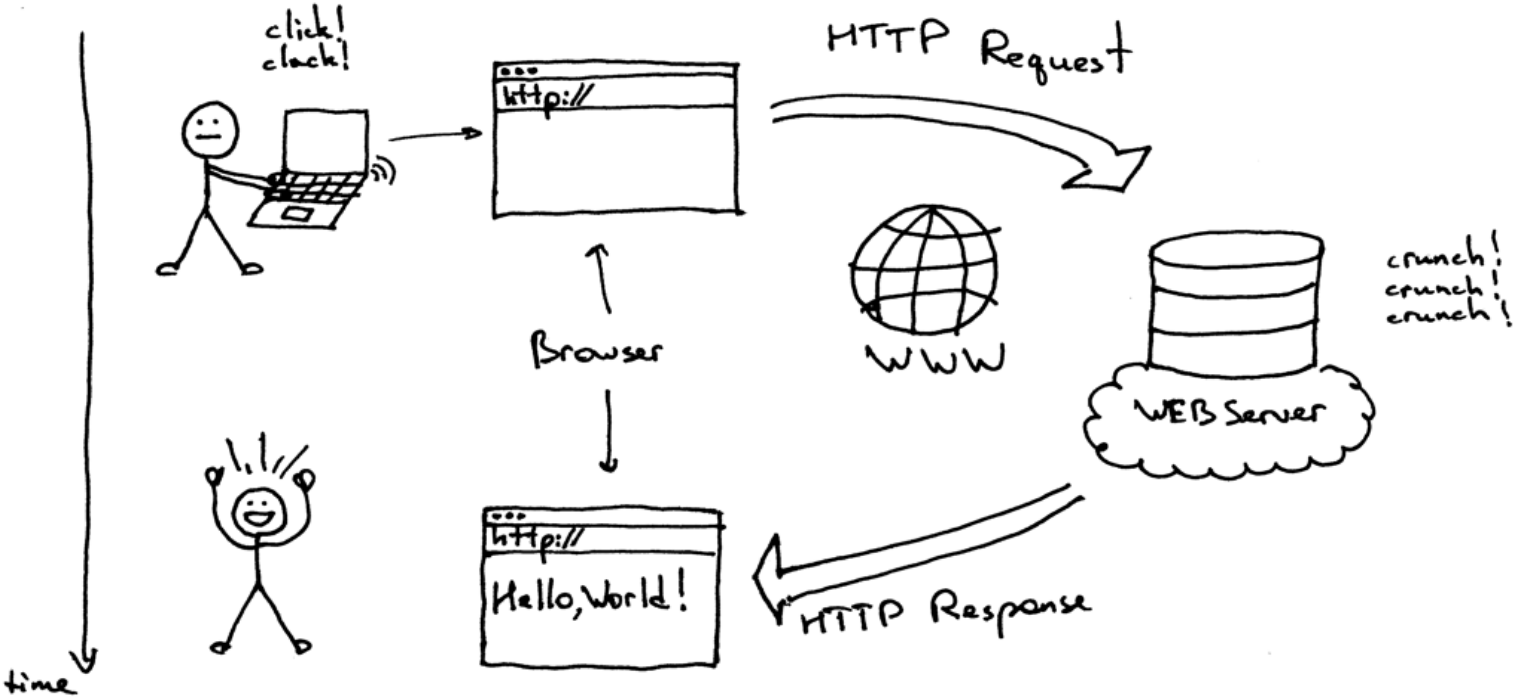


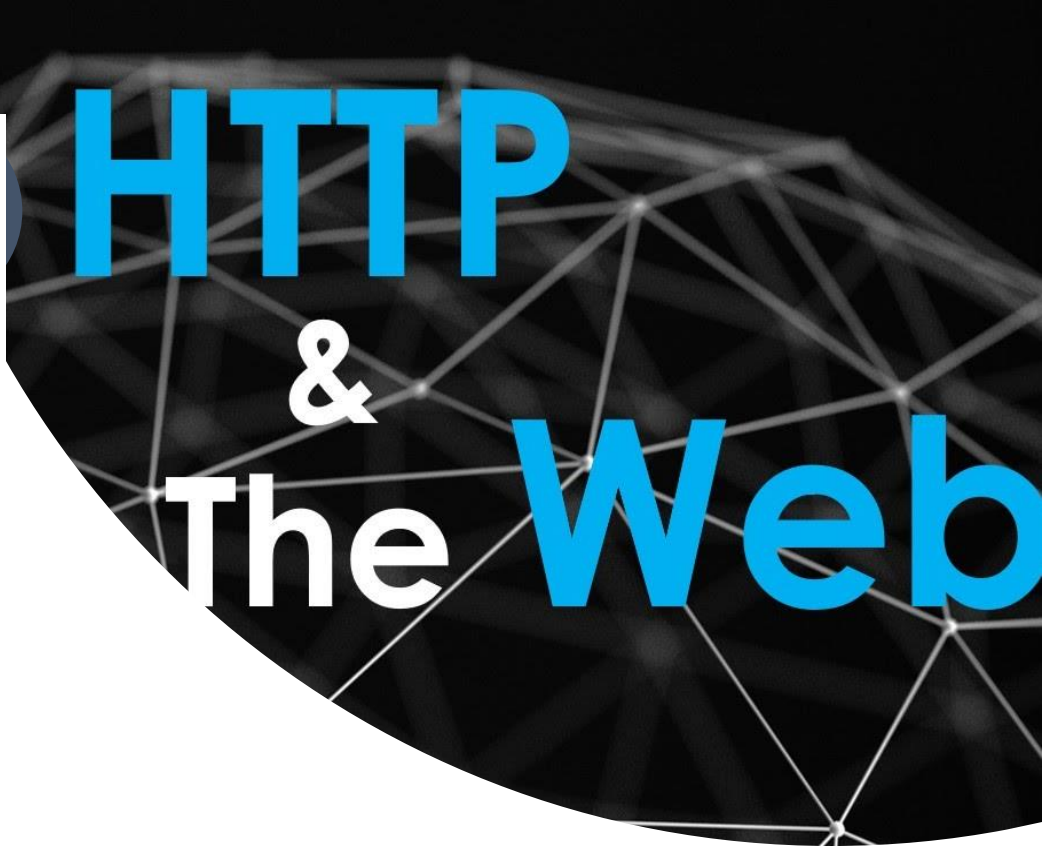
Client / Server model:

Client: A program that requests and receives web objects. Example: Mozilla Firefox, IE, Chrome, Safari browser, etc.

Server: A web server that sends objects in response to requests. Example: Apache web server, Tomcat web server, etc.

Hypertext Transfer Protocol





HTTP uses TCP:
Client initiates TCP connection to server.
Server accepts TCP connection from client.
HTTP messages are exchanged from client to server.
TCP connection is closed.

HTTP is stateless:
The protocol does not keep any information from either end to keep the history of past requests.

HTTP Messages

A request message consists of the following:

- A request line
- A number of request headers, each on their own line
- An empty line
- An optional message body, which can also take up multiple lines.

Each line in an HTTP must end with <CR><LF>

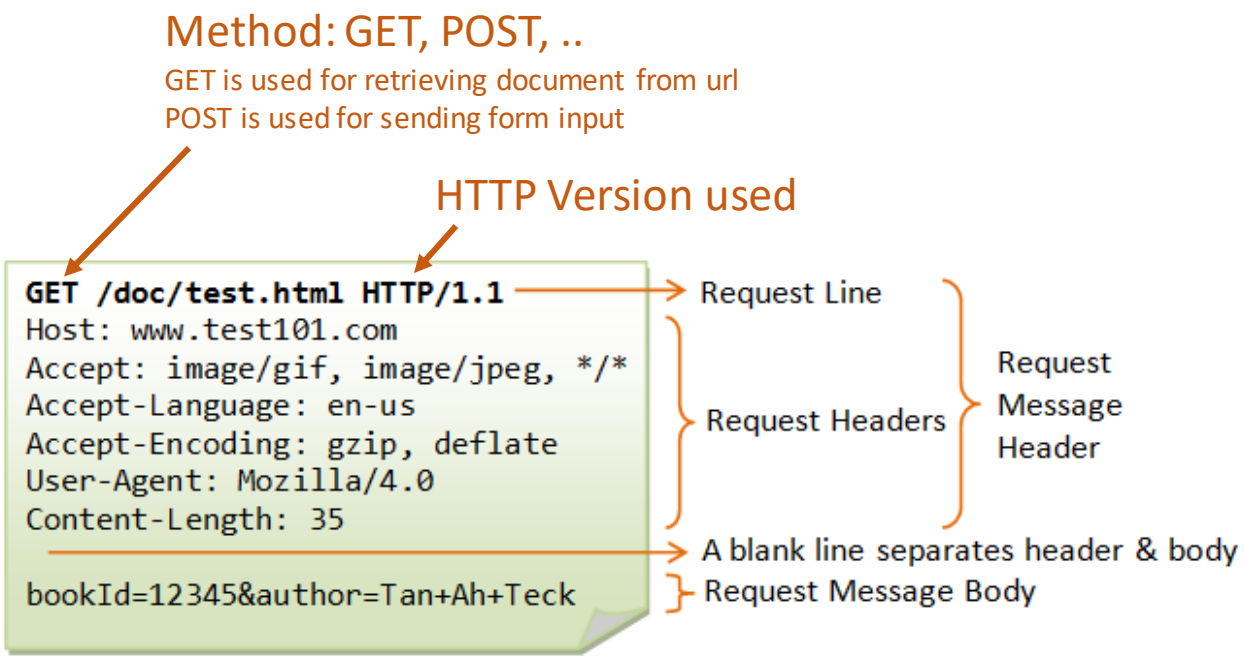
<CR><LF> are two special characters to indicate that a new line should be started.

A response message consists of the following:

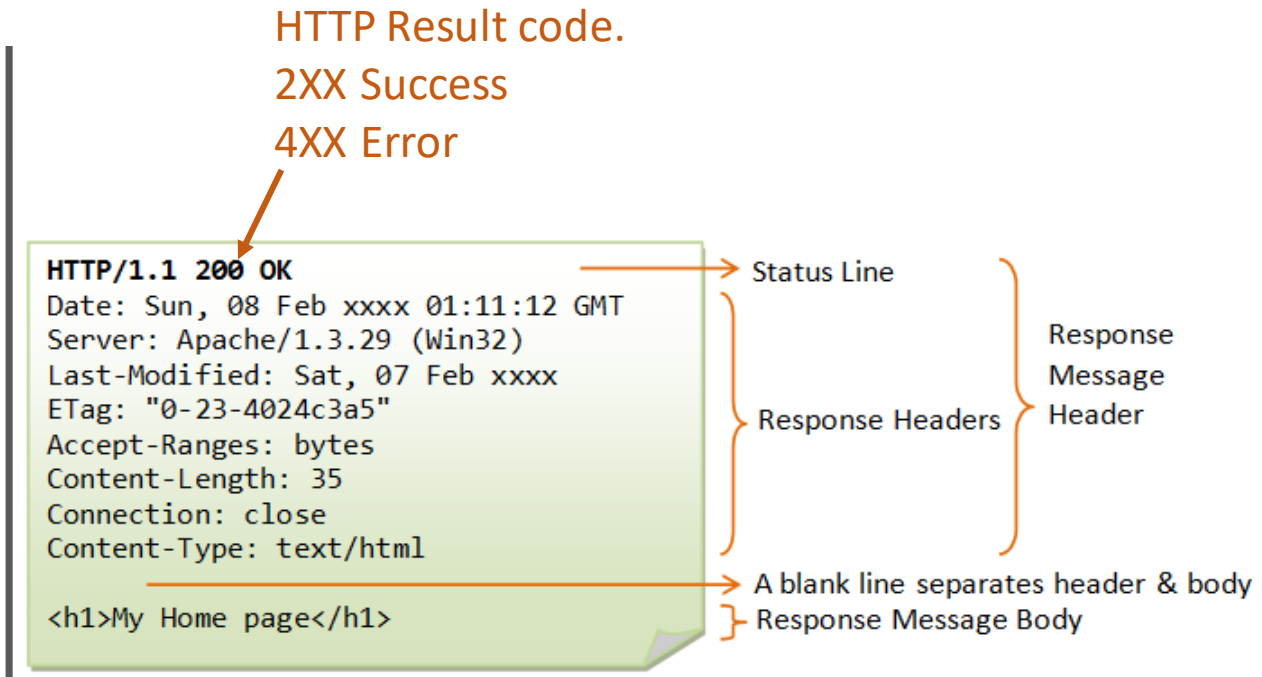
- A status line that includes the status code and a status message
- A number of response headers, again all on the same line;
- An empty line;
- An optional message body.

Two types of HTTP messages

HTTP Messages



HTTP Request message



HTTP Response message

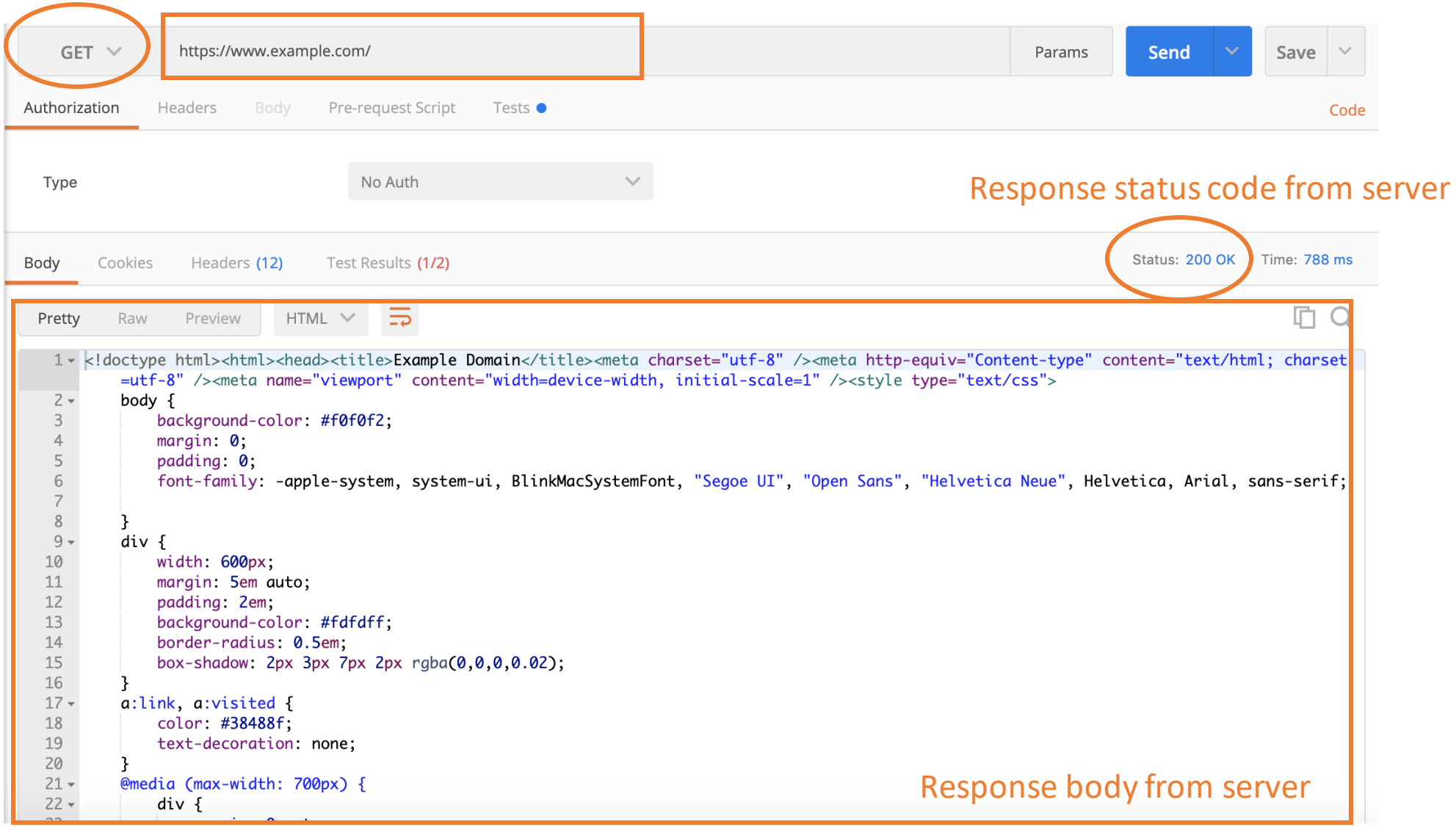
Two types of HTTP messages

[Bilder: ntu.edu.sg]

Anatomy of Postman (https://www.postman.com/)

Request method

Web address



Python code to access html page
(*We will see more in Lecture03*)

```
# urllib is a standard Python library and contains functions for requesting data across the web, handling cookies.
# You can get more information from https://docs.python.org/3/library/urllib.html

from urllib.request import urlopen

html = urlopen('http://pythonscraping.com/pages/page1.html')
print(html.read())
```

In this lecture we cover the following

- Internet basics
  - Packet switching, TCP/IP, DNS
  - OSI model
  - Client-server architecture
- Web based technologies
  - Client-side web technologies
    - HTML Basics
    - XPath

Web Server

- A **web server** is software that can satisfy client requests on the World Wide Web.
- Sometime a dedicated hardware is used to serve client requests which is programmed to send files to browsers on other computers connected to the internet.
- A web server can, in general, contain one or more websites.
- A web server processes incoming network requests over HTTP and several other related protocols.
- The primary function of a web server is to store, process and deliver web pages to clients.
- The communication between client and server takes place using the Hypertext Transfer Protocol (HTTP).
- Pages delivered are most frequently HTML documents, which may include images, style sheets and scripts in addition to the text content.

[https://en.wikipedia.org/wiki/Web_server]

Web Client

- A user agent, commonly a web browser or web crawler, initiates communication by making a request for a specific resource using HTTP and the server responds with the content of that resource or an error message if unable to do so.
- The resource is typically a real file on the server's secondary storage, but this is not necessarily the case and depends on how the web server is implemented.
- While the major function is to serve content, a full implementation of HTTP also includes ways of receiving content from clients.
- This feature is used for submitting web forms, including uploading of files.

[https://en.wikipedia.org/wiki/Web_server]

- Wireshark is a network **packet analyzer**.

A network packet analyzer presents captured packet data in as much detail as possible.

(You could think of a network packet analyzer as a measuring device for examining what's happening inside a network cable, just like an electrician uses a voltmeter for examining what's happening inside an electric cable).

- Wireshark allows you to filter the log before the capture starts or during analysis, so you can narrow down and zoom in on what you're looking for in the network trace. For example, you can set a filter to see TCP traffic between two IP addresses, or you can set it only to show you the packets sent from one computer. The filters in Wireshark are one of the primary reasons it has become the standard tool for packet analysis.

Network Protocol Analysis with Wireshark

- A great example of this is when Varonis security researchers discovered the Norman crypto miner, they received an alert from several machines pointing to suspicious network and file activity. During the analysis of the crypto miner, Varonis researchers used Wireshark to inspect network activities for some of the machines that were misbehaving. Wireshark showed the research team that a new crypto miner, dubbed Norman, was actively communicating to command and control (C&C) servers using DuckDNS. The Varonis team could see all the IP addresses of the C&C servers the attackers used with Wireshark so the company could shut off communication, stopping the attack.

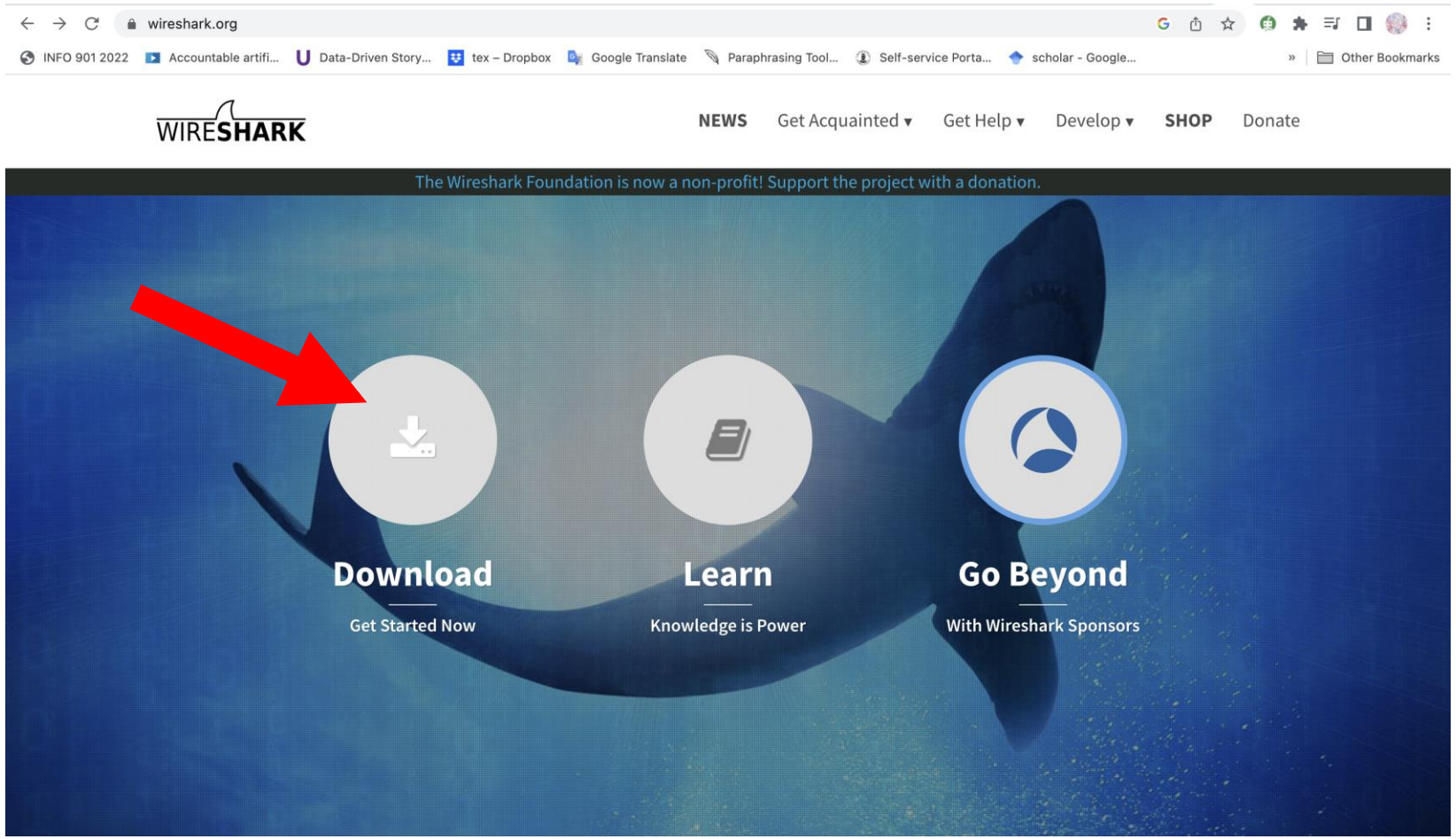
https://www.varonis.com/blog/how-to-use-wireshark

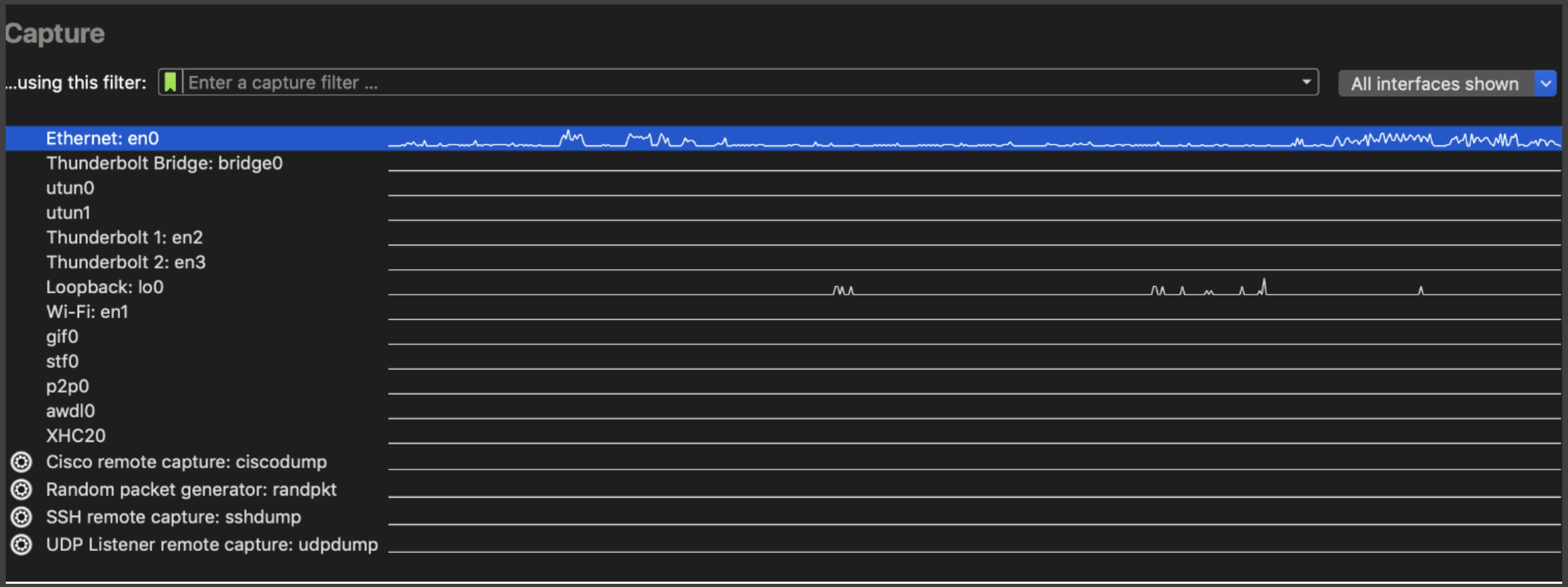
Here are some reasons people use Wireshark:

- Network administrators use it to troubleshoot network problems
- Network security engineers use it to examine security problems
- QA engineers use it to verify network applications
- Developers use it to debug protocol implementations
- People use it to learn network protocol internals

Wireshark can also be helpful in many other situations.

Install from https://www.wireshark.org/





- When you open Wireshark without starting a capture or opening a capture file it will display the “Welcome Screen,” which lists any recently opened capture files and available capture interfaces. Network activity for each interface will be shown in a sparkline next to the interface name. It is possible to select more than one interface and capture from them simultaneously.

Three panes for inspecting packet data:

- The Packet List:

lists all the packets in the capture. When you click on a packet, the other two panes change to show you the details about the selected packet. You can also tell if the packet is part of a conversation (you can right-click the packet and select Follow to see only the packets that are part of that conversation.).

Here are details about each column in the top pane:

No.: This is the number order of the packet captured. The bracket indicates that this packet is part of a conversation.

Time: This column shows how long after you started the capture this particular packet was captured. You can change this value in the Settings menu to display a different option.

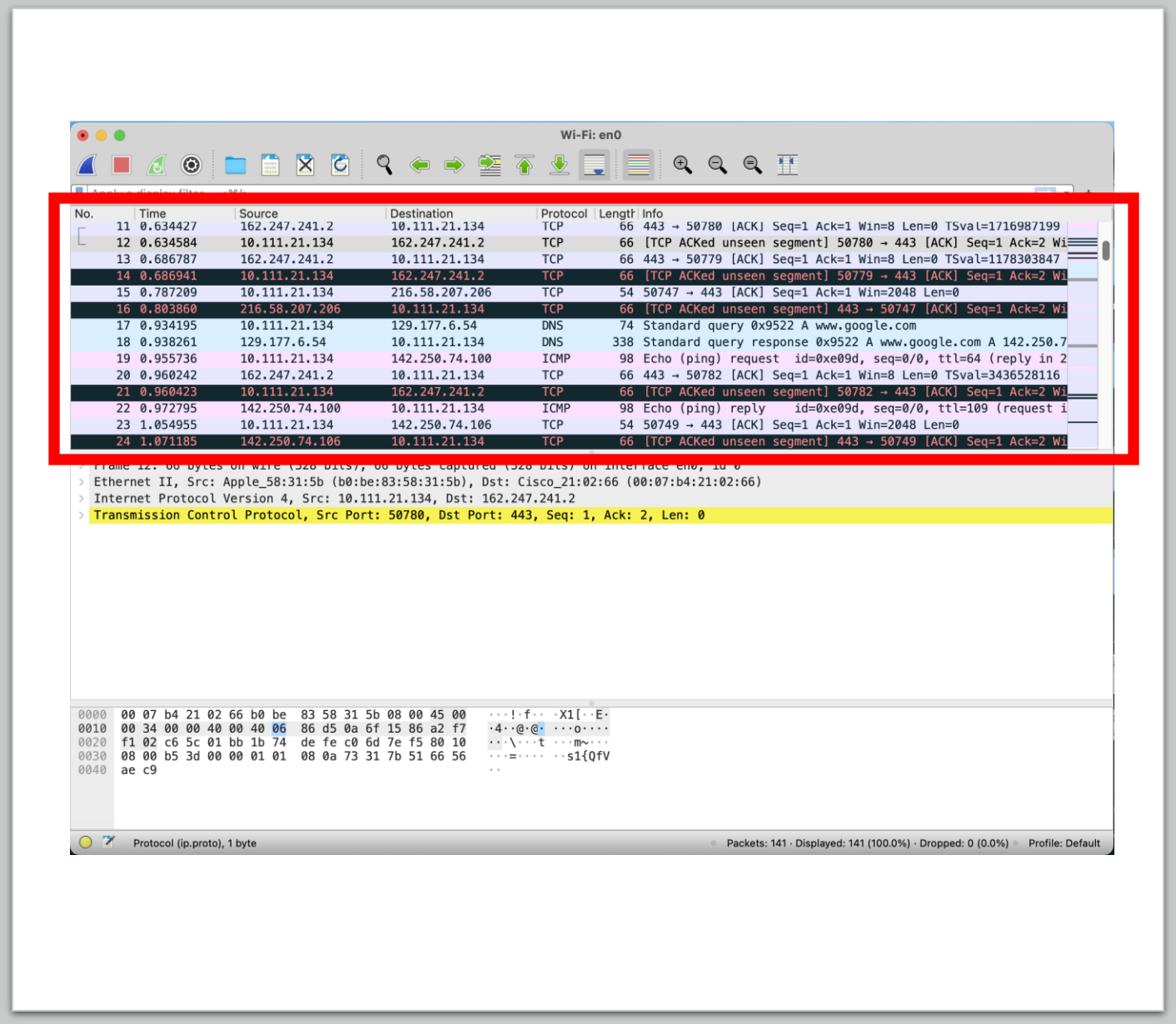
Source: This is the address of the system that sent the packet.

Destination: This is the address of the packet destination.

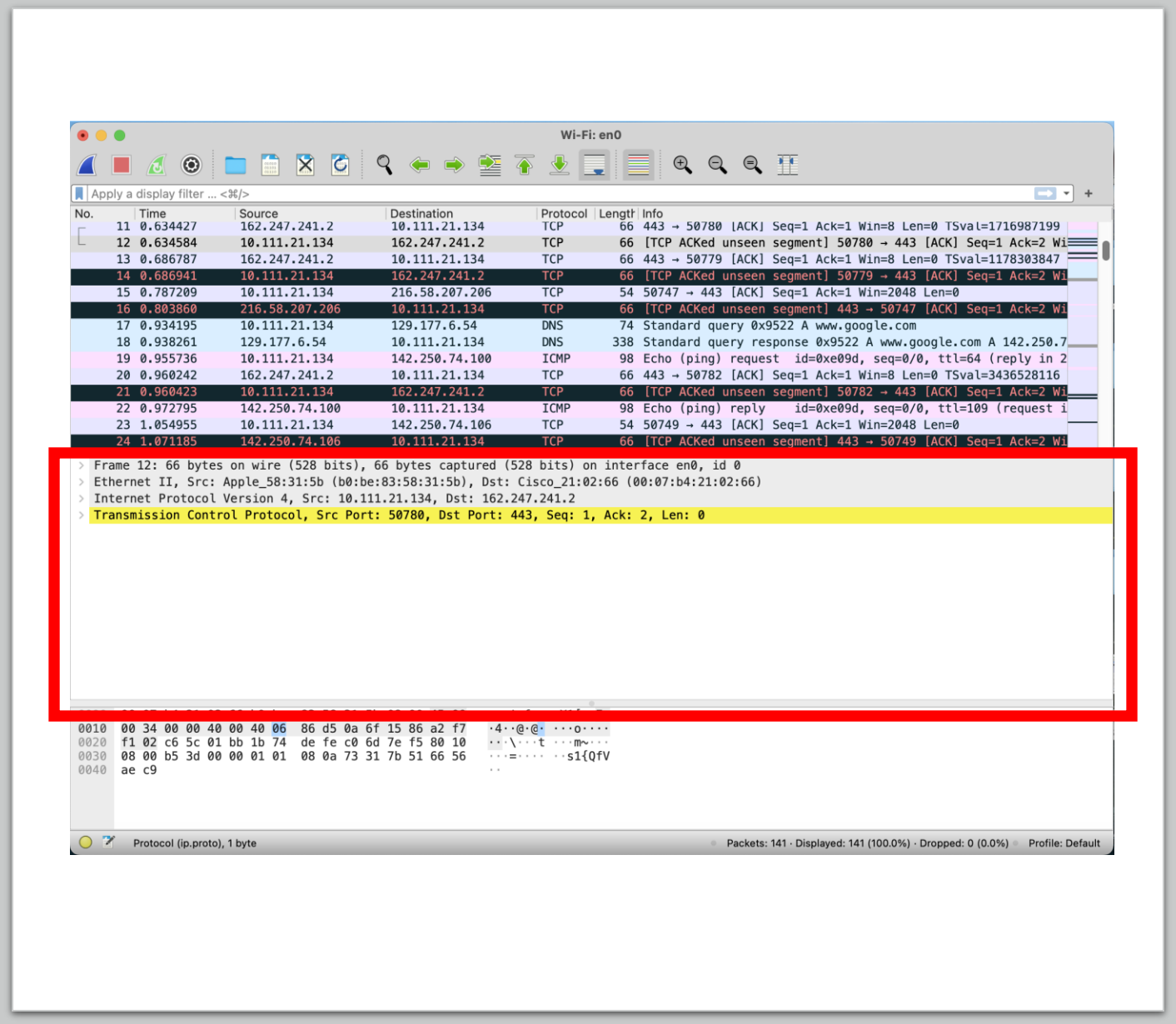
Protocol: This is the type of packet. For example: TCP, DNS, DHCPv6, or ARP.

Length: This column shows you the packet's length, measured in bytes.

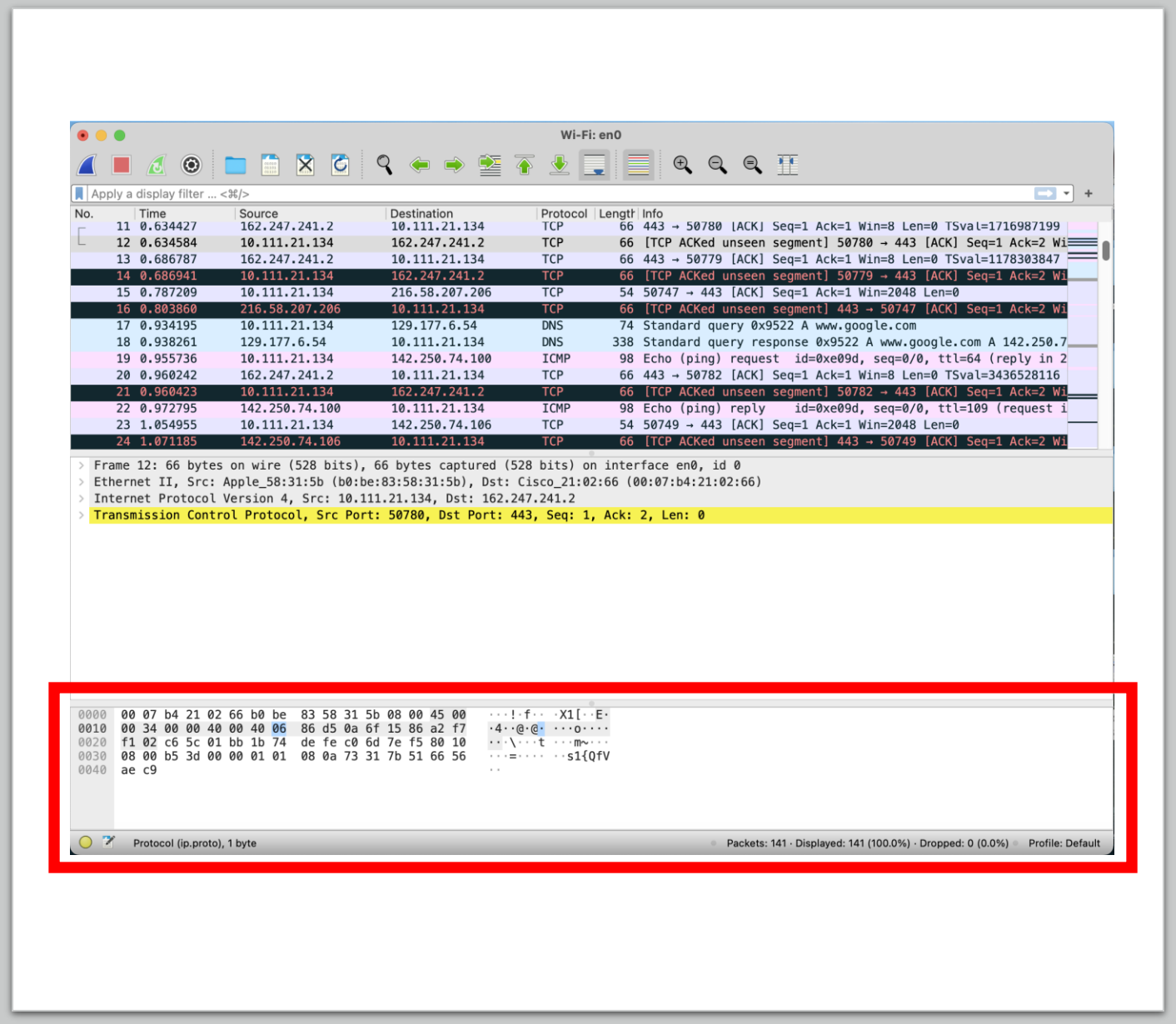
Info: This column shows you more information about the packet contents, which will vary depending on the type of packet.



- Packet Details, the middle pane, shows you as much readable information about the packet as possible, depending on the packet type. You can right-click and create filters based on the highlighted text in this field.



- The bottom pane, Packet Bytes, displays the packet exactly as it was captured in hexadecimal.



- Capture filters limit the captured packets by the chosen filter. If the packets don't match the filter, Wireshark won't save them.

Examples:

❖ ip.dst == 216.239.34.36

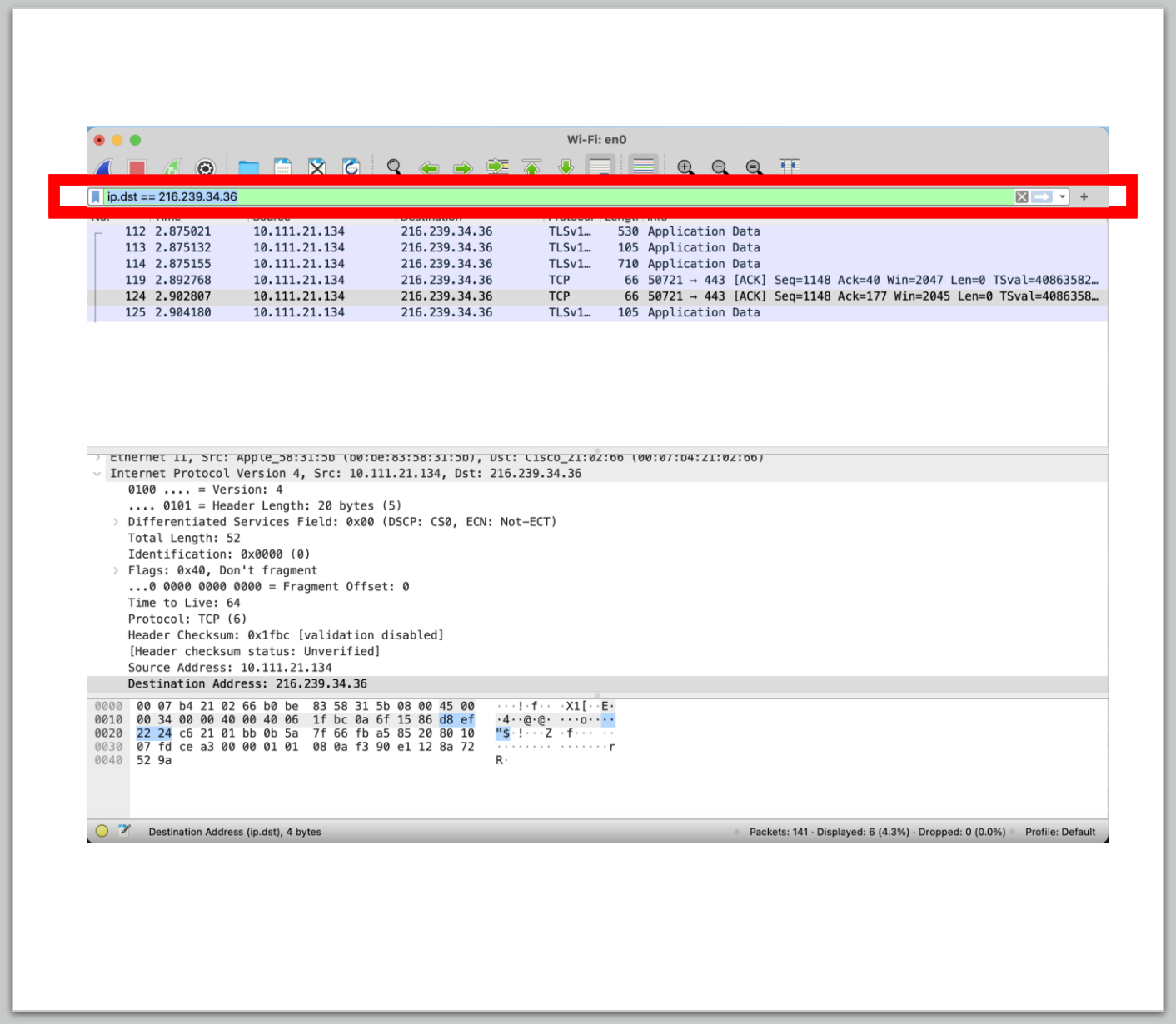
This filter shows packets sent from one computer (ip.src) to another (ip.dst). You can also use ip.addr to show packets to and from that IP.

❖ tcp.port == 25

will show you all traffic on port 25.

❖ Icmp/tcp/udp etc.

filtering based on the protocol.



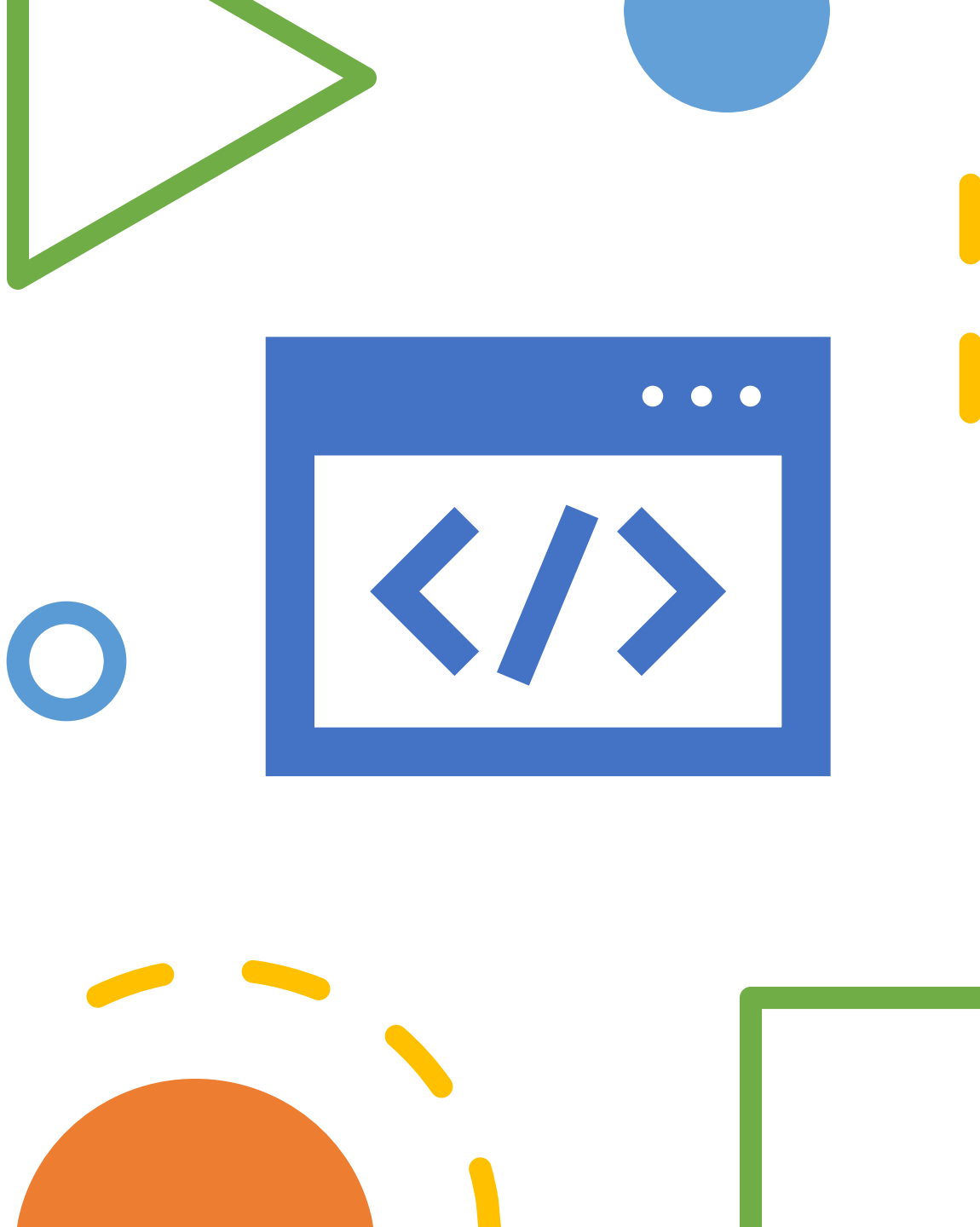
Pick a website you wish (for example www.google.com). Using ping utility (window/macOS) and wireshark find answer to the following questions:

1. Run <ping -n 5 www.yoursite.com> or <ping -c 5 www.yoursite.com> in your cmd/terminal while sniffing the network by wireshark.
2. What is the IP address of the website you pick?
3. How many packets have you captured overall and how many have you captured from this website?
4. What protocol does the ping utility use?
5. Using only wireshark, compute the RTT (Round Trip Time) – how long it took since your ping request was sent and until the ping reply was received?
6. How many TCP packets have your computer sent or received?

- Your answer should contain the screenshot of each stage.

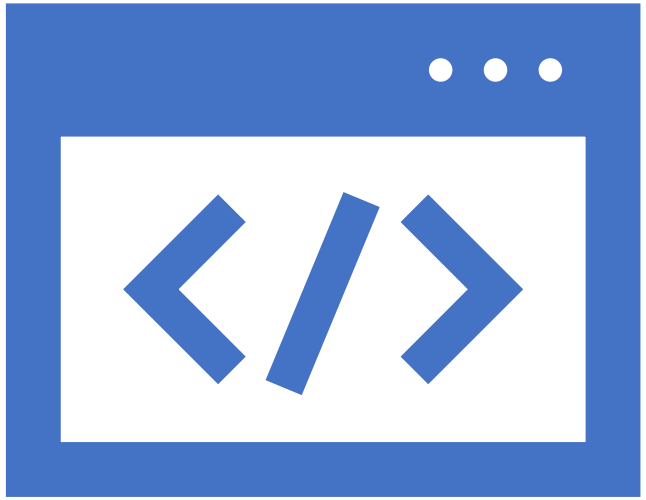
HTML Basics

- HTML Tags
- HTML5 with SVG

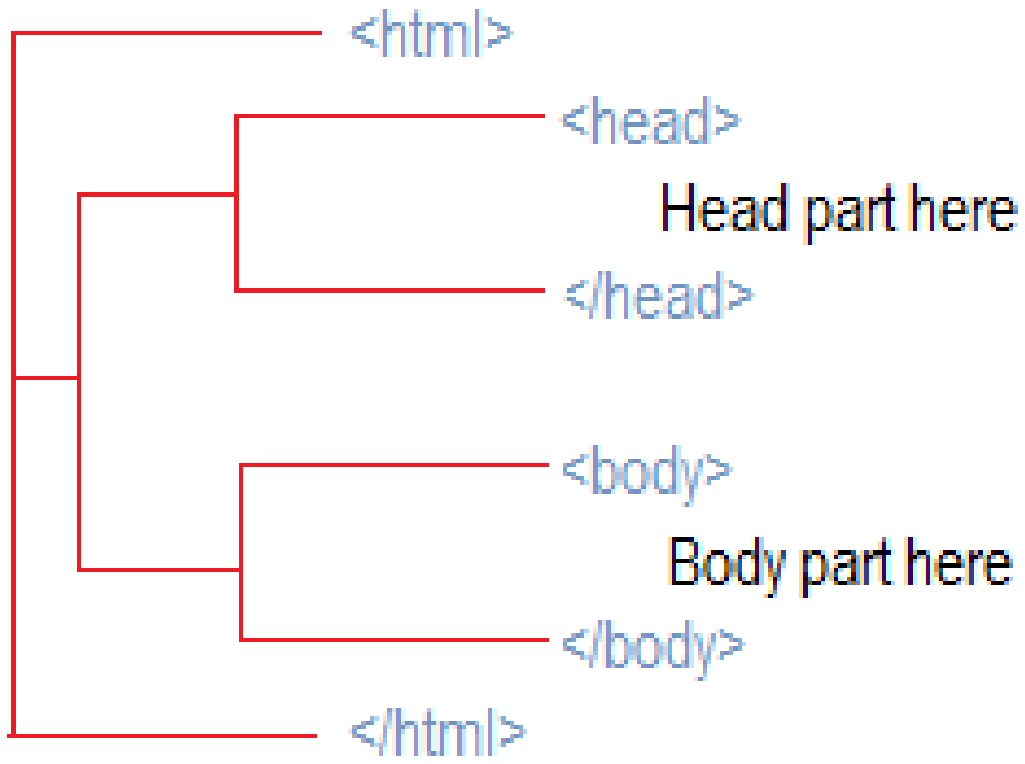


What is HTML?

- HTML is a markup language for describing web pages
- HTML stands for Hyper Text Markup Language
- HTML defines of a set of markup tags.
- The tags are used to describe document contents.
- HTML documents contain HTML tags and plain text



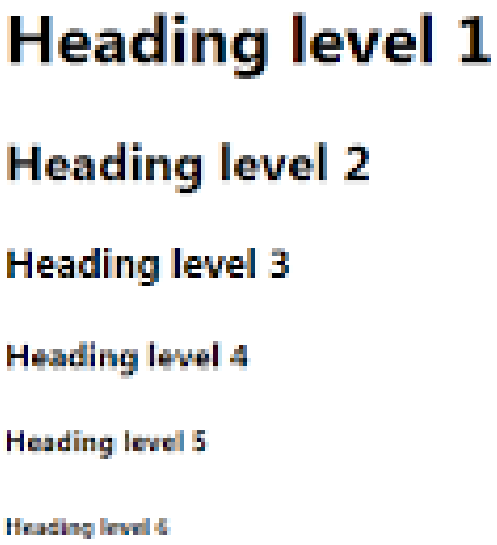
Basic HTML tags



HTML page structure

- HTML tags are keywords surrounded by angle brackets
- HTML tags normally come in pairs e.g., <body> and </body>
- The first tag in a pair is called the start tag and the second tag is called the end tag

HTML heading tags



- HTML headings are defined with the <h1> to <h6> tags.
- Example:

<h1>Heading level 1 </h1>

<h2>Heading level 2 </h2>

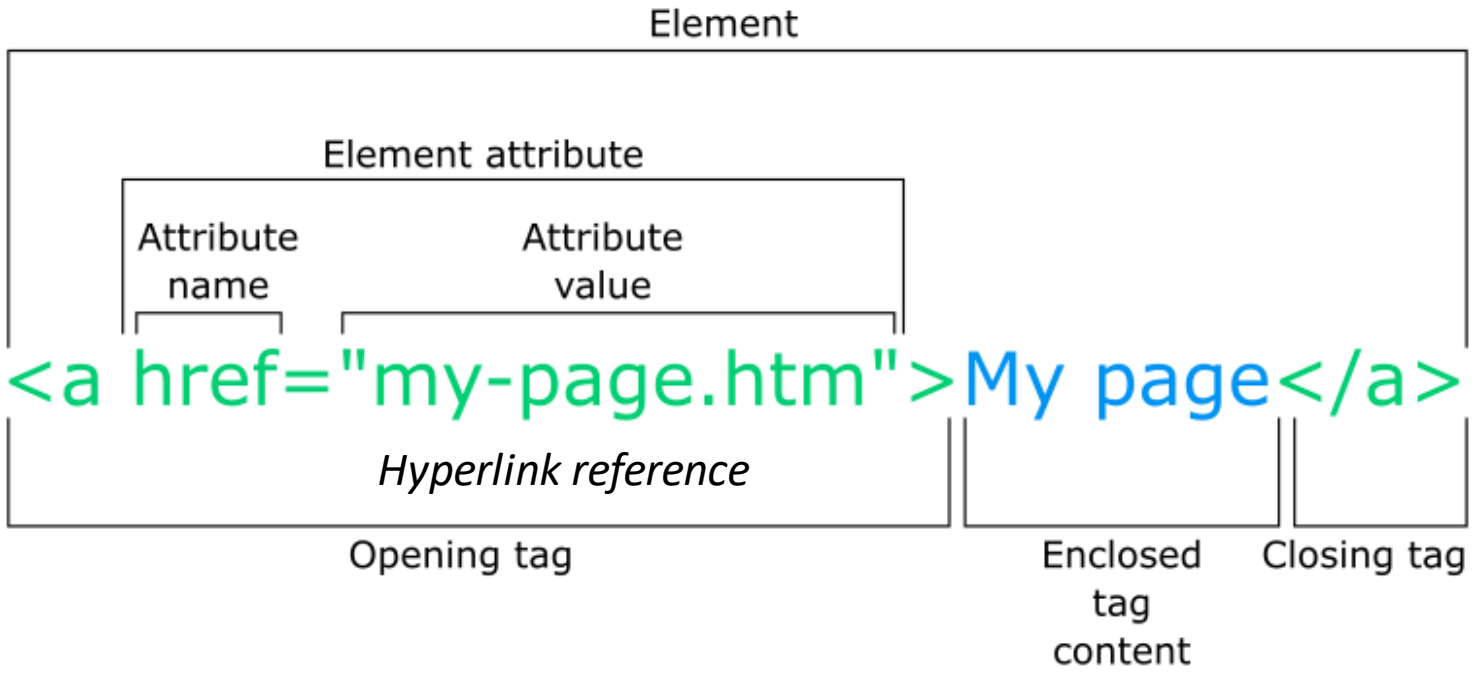
<h3>Heading level 3 </h3>

<h4>Heading level 4 </h4>

<h5>Heading level 5 </h5>

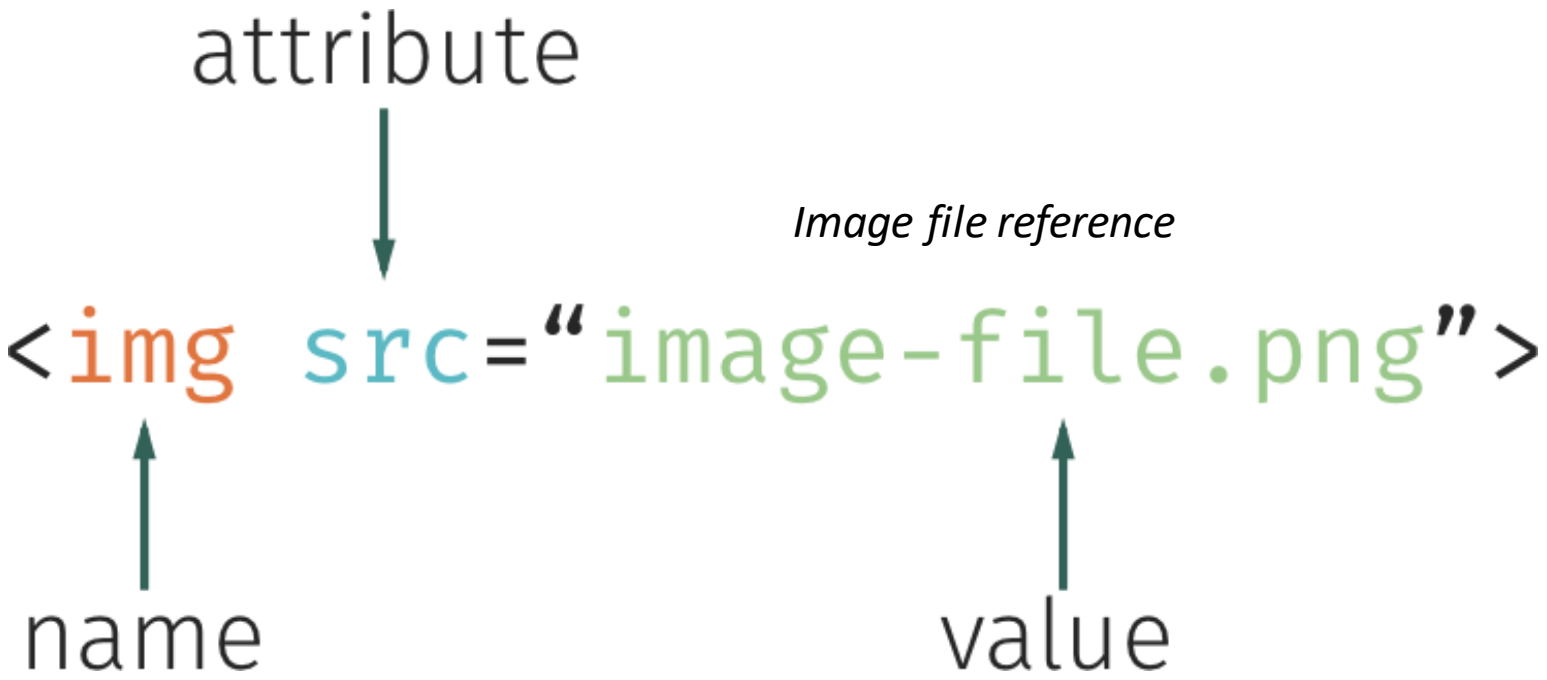
<h6>Heading level 6 </h6>

HTML hyperlink tag



- HTML link tags are defined with the <a>.
- Hyperlinks are used to move to another web page/document.
- The href attribute of an <a> tag indicates the link's destination.

HTML <img> tag



- HTML image tags are defined with the <img> .
- The <img> tag is used to embed an image in a web page.
- Images are not technically inserted into a web page; images are linked to web pages. The <img> tag creates a holding space for the referenced image.

HTML <style> tag

```
<!DOCTYPE html>
<html>
<head>
  <title>Page Title</title>
  <style>
    body {background-color: powderblue;}
    h1 {color: red;}
    p {color: blue;}
  </style>
</head>
<body>

<h1>This is a Heading</h1>
<p>This is a paragraph.</p>

</body>
</html>
```

- The <style> tag is used to define style information for an html document.
- Inside the <style> element we can specify how html elements should be rendered in a browser

HTML tag

```
<!DOCTYPE html>
<html>
<head>
<style>
.myDiv {
  border: 5px outset red;
  background-color: lightblue;
  text-align: center;
}
</style>
</head>
<body>

<h1>The div element</h1>

<div class="myDiv">
  <h2>This is a heading in a div element</h2>
  <p>This is some text in a div element.</p>
</div>

<p>This is some text outside the div element.</p>

</body>
</html>
```

- The tag defines a division or a section in an HTML document.
- The tag is used as a container for HTML elements - which is then styled with CSS or manipulated with JavaScript.
- The tag is easily styled by using the class or id attribute.
- Any sort of content can be put inside the tag!

- Try it yourself:
- https://www.w3schools.com/tags/tag_div.ASP

Other HTML tags

HTML Tag	Description
<b>	Bold
 	Line break
	Emphasize text
<font>	Determines appearance
<h1> to <h6>	Section Heading 1 to 6
<hr>	Horizontal line
<i>	Italics
	Open list
	Close numbered list
<p>	Indented paragraph break
<pre>	Defines preformatted text
<s>	Strikethrough
<strike>	Strikethrough; See also <s>
	Bold; See also <b>
<sub>	Subscript
<sup>	Superscript
<u>	Underlined
	Close bulleted list

HTML Table

Tag	Description
<table>	It defines a table.
<tr>	It defines a row in a table.
<th>	It defines a header cell in a table.
<td>	It defines a cell in a table.
<caption>	It defines the table caption.
<tbody>	It is used to group the body content in a table.
<thead>	It is used to group the header content in a table.

HTML <table> tag

```
<h2>HTML Table</h2>

<table>
  <tr>
    <th>Company</th>
    <th>Contact</th>
    <th>Country</th>
  </tr>
  <tr>
    <td>Alfreds
Futterkiste</td>
    <td>Maria Anders</td>
    <td>Germany</td>
  </tr>
  <tr>
    <td>Centro comercial
Moctezuma</td>
    <td>Francisco Chang</td>
    <td>Mexico</td>
  </tr>
  <tr>
    <td>Ernst Handel</td>
    <td>Roland Mendel</td>
    <td>Austria</td>
  </tr>
```

HTML Table

Company	Contact	Country
Alfreds Futterkiste	Maria Anders	Germany
Centro comercial Moctezuma	Francisco Chang	Mexico
Ernst Handel	Roland Mendel	Austria

Tag	Description
<table>	It defines a table.
<tr>	It defines a row in a table.
<th>	It defines a header cell in a table.
<td>	It defines a cell in a table.
<caption>	It defines the table caption.
<tbody>	It is used to group the body content in a table.
<thead>	It is used to group the header content in a table.

Scalable Vector Graphics (SVG)

SVG tags: (can be used inside html body tag)

```
<svg width="400" height="300">
    <rect fill="green"  width="100" height="100"></rect>
   <line x1="50" y1="50" x2="180" y2="150" style="stroke:#00ff00; fill: none;"></line>
   <ellipse cx="200" cy="100" rx="100" ry="50" style="stroke:#0000ff; fill:#6666ff;"></ellipse>
</svg>
```

Try it out by using the editor below:

https://www.javatpoint.com/oprweb/test.jsp?filename=htmlsvg2

- **SVG is an XML**-based vector image format for two-dimensional graphics with support for interactivity and animation.
- The SVG specification was developed by the World Wide Web Consortium (W3C).
- SVG supports the following drawing features:
  - Basic shapes: Line, Rectangle, Ellipse, Circle, Polygons
  - Advanced shapes using paths e.g., arcs
  - Text with various fonts
  - Images http://tutorials.jenkov.com/svg/svg-examples.html

Scalable Vector Graphics (SVG)

```
<svg xmlns="http://www.w3.org/2000/svg"
   xmlns:xlink="http://www.w3.org/1999/xlink">

   <g>
    <line x1="10" y1="10" x2="85" y2="10"
       style="stroke: #006600;"/>

    <rect x="10" y="20" height="50" width="75"
       style="stroke: #006600; fill: #006600"/>

    <text x="10" y="90" style="stroke: #6600
     Text grouped with shapes</text>
   </g>
```

- The SVG <g> element is used to group SVG shapes together. Once grouped you can transform the whole group of shapes as if it was a single shape.

Scalable Vector Graphics (SVG)

```
<svg xmlns="http://www.w3.org/2000/svg"
   xmlns:xlink="http://www.w3.org/1999/xlink">

   <g transform="rotate(45  50 50)">
    <line x1="10" y1="10" x2="85" y2="10"
       style="stroke:  #006600;"/>

    <rect x="10" y="20" height="50" width="75"
       style="stroke:  #006600; fill: #006600"/>

    <text x="10" y="90" style="stroke:  #660000; fill: #660000">
     Text grouped with shapes</text>
   </g>
```

- The transform attribute specifies what transformations to apply to the shapes.
- You can apply transformation to all SVG shapes. You can also apply transformation to the <g> element, thus effectively transforming a whole group of elements in one go.
- Notice how all shapes within the <g>-element are rotated 45 degrees around the point 50,50.

Scalable Vector Graphics (SVG)

```
<svg width="400" height="300">
    <polygon points="200,10 250,190 160,210" style="fill:lime;stroke:purple;stroke-width:1" />
</svg>
```

- **Drawing with SVG polygon.**
- The points attribute defines the x and y coordinates for each corner of the polygon

Scalable Vector Graphics (SVG)

- **Using svg tags draw the Norwegian flag as shown on the left.**
- **You are allowed to use <polygon>, <rect>, <line>, etc but <img> tag is not allowed.**

Title of your website

Description of the website….

A simple SVG logo (e.g. a circle, square etc.)

Your website’s content goes here.

(your content should include lists, images, links)

A table using table tag.

Scalable Vector Graphics (SVG)

```
<svg width="400" height="300">
    <polyline points="0,40 40,40 40,80 80,80 80,120 120,120 120,160"
 style="fill:white;stroke:red;stroke-width:4" />
</svg>
```

- **Drawing with SVG polygon.**
- The points attribute defines the x and y coordinates for each corner of the polygon

Scalable Vector Graphics (SVG)

Drawing image:

```
<svg width="1000"  height="600">
   <image x="100" y="20"  width="300" height="180" transform="  rotate(25) "
        xlink:href="img/UIB_Campus.png"  />
</svg>
```

- Drawing with SVG image tag

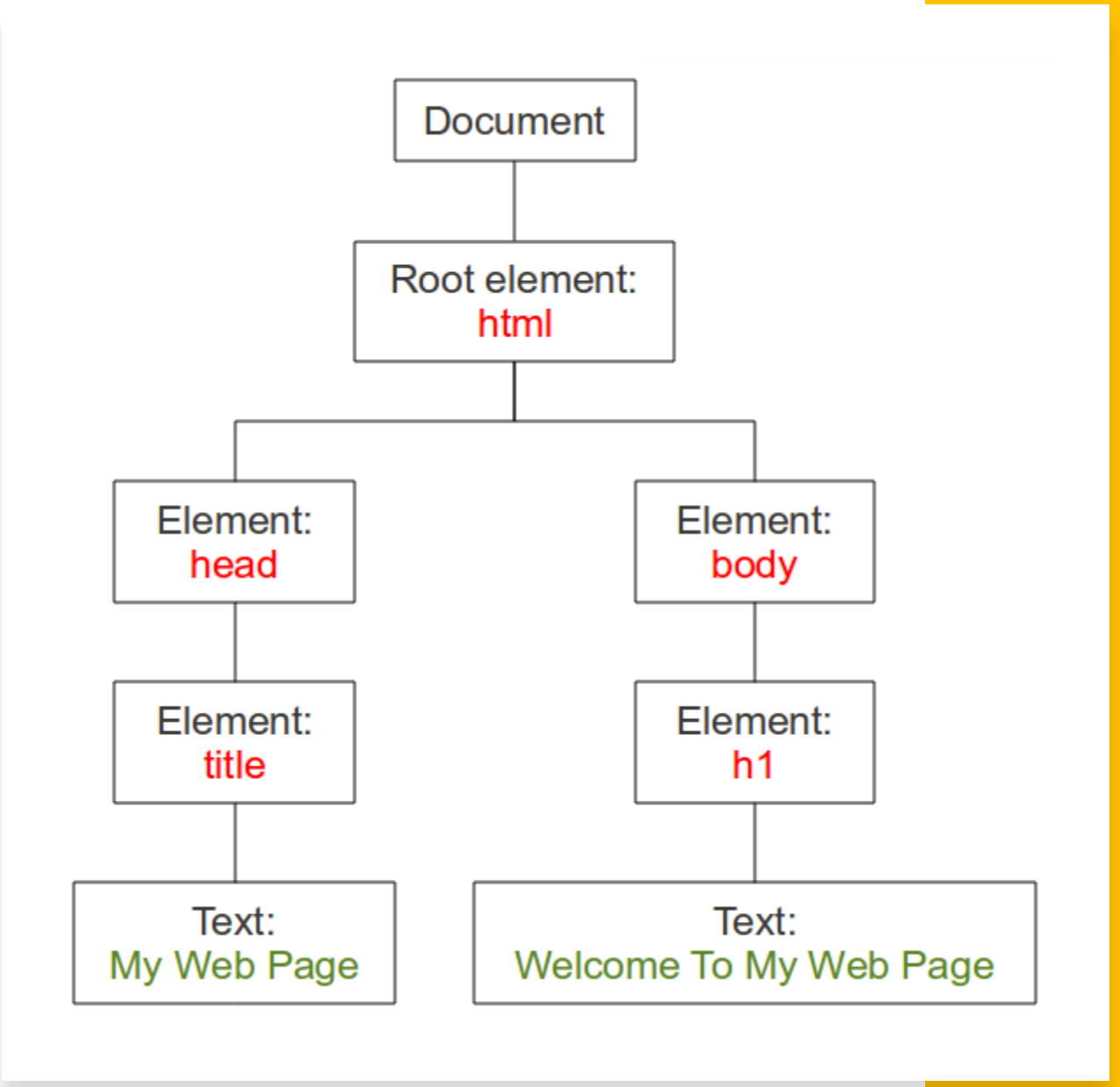
Document Object Model (DOM)

- The Document Object Model (DOM) is a programming interface for HTML and XML documents:
- DOM provides a structured representation of a XML or HTML document
- It defines a way that the structure can be accessed from programs so that they can change the document structure, style and content
- The DOM provides a representation of the document as a structured group of **nodes** and **objects** that have properties and methods.

Document Object Model (DOM)

- The HTML DOM views an HTML document as a tree structure called the **node tree**.

```
<html>
<head>
   <title>My Web Page</title>
</head>
 <body>
   <h1>Welcome To My Web Page</h1>
 </body>
</html>
```



Document Object Model (DOM)

- To view and change DOM using chrome:

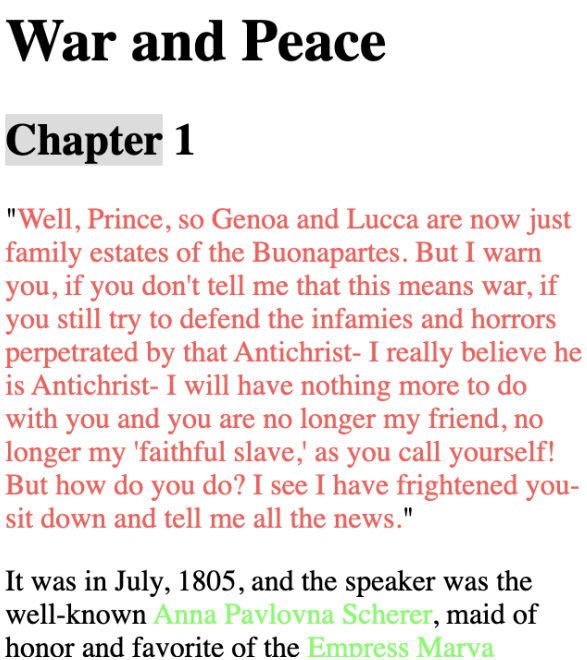
Right click on an element and click on Inspect.

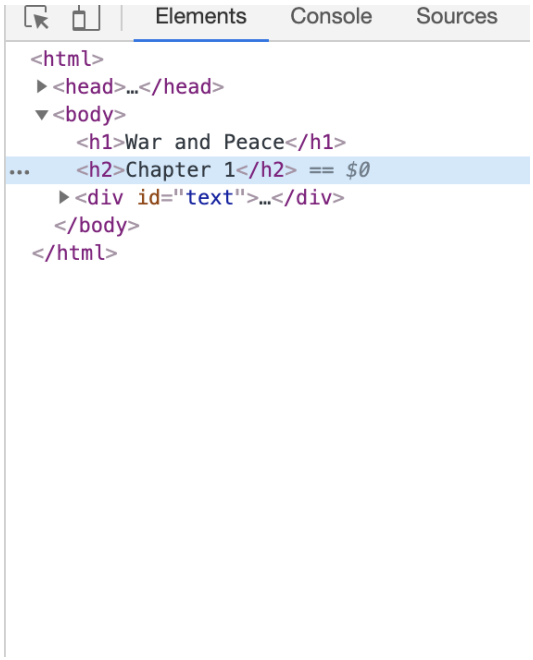
For example:

Open the following webpage in chrome browser.

http://www.pythonscraping.com/pages/warandpeace.html

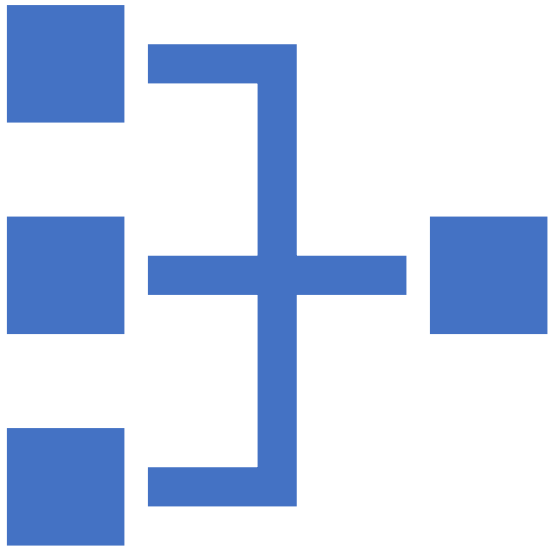
Right click on Chapter 1 and click on Inspect.





XPath

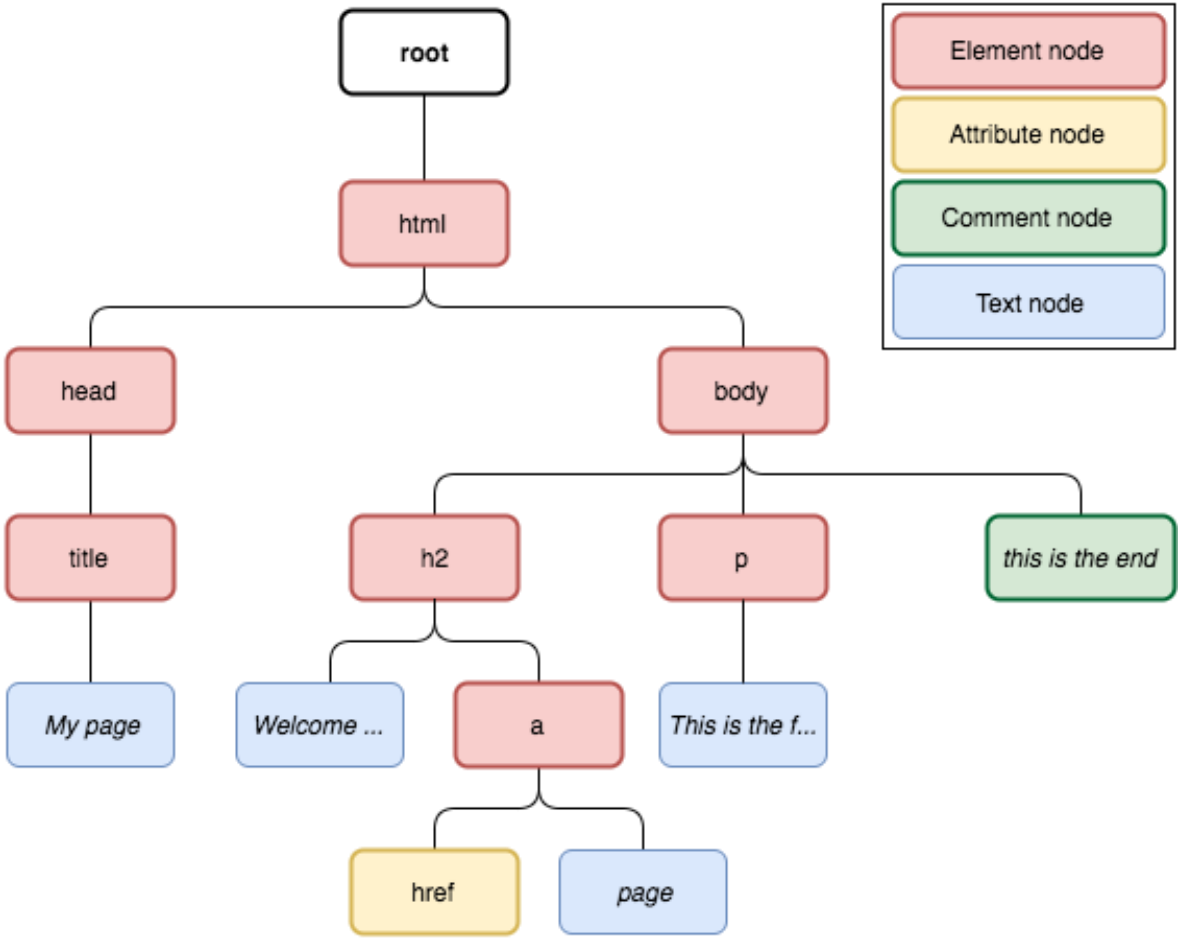
- XPath is a syntax for defining parts of an XML document.
- XPath handles any XML/HTML document as a tree.
- XPath uses path expressions to navigate in XML documents
- XPath contains a library of standard functions



XPath Terminology

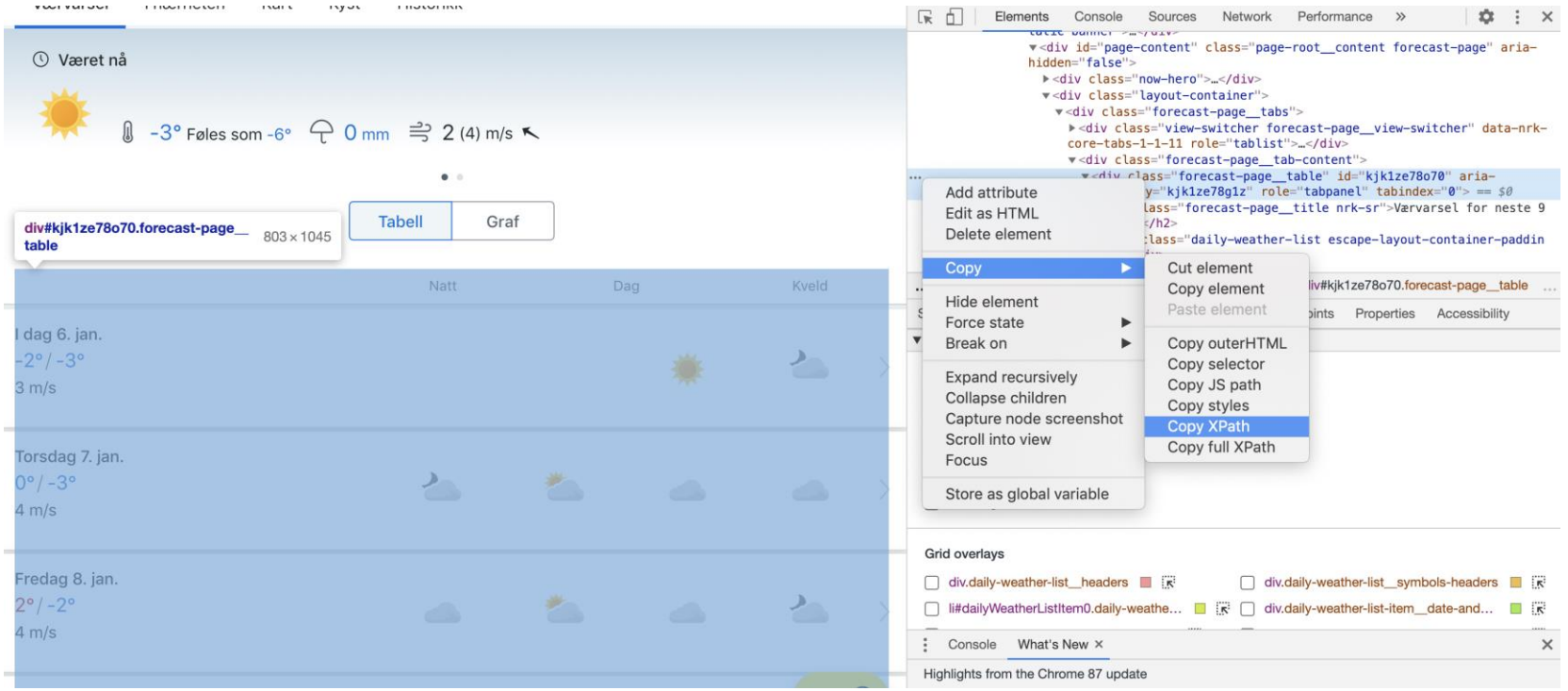
Kinds of Nodes in XPath:

- **Element node:** represents XML/HTML element, e.g., HTML tag.
- **Attribute node:** represents an attribute from an element node, e.g. “href” attribute in the hyperlink tag <a>.
- **Comment node:** represents comments in the document (<!-- … -->).
- **Text node:** represents the text enclosed in an element node (example in <h1>Harry Porter</h1>).



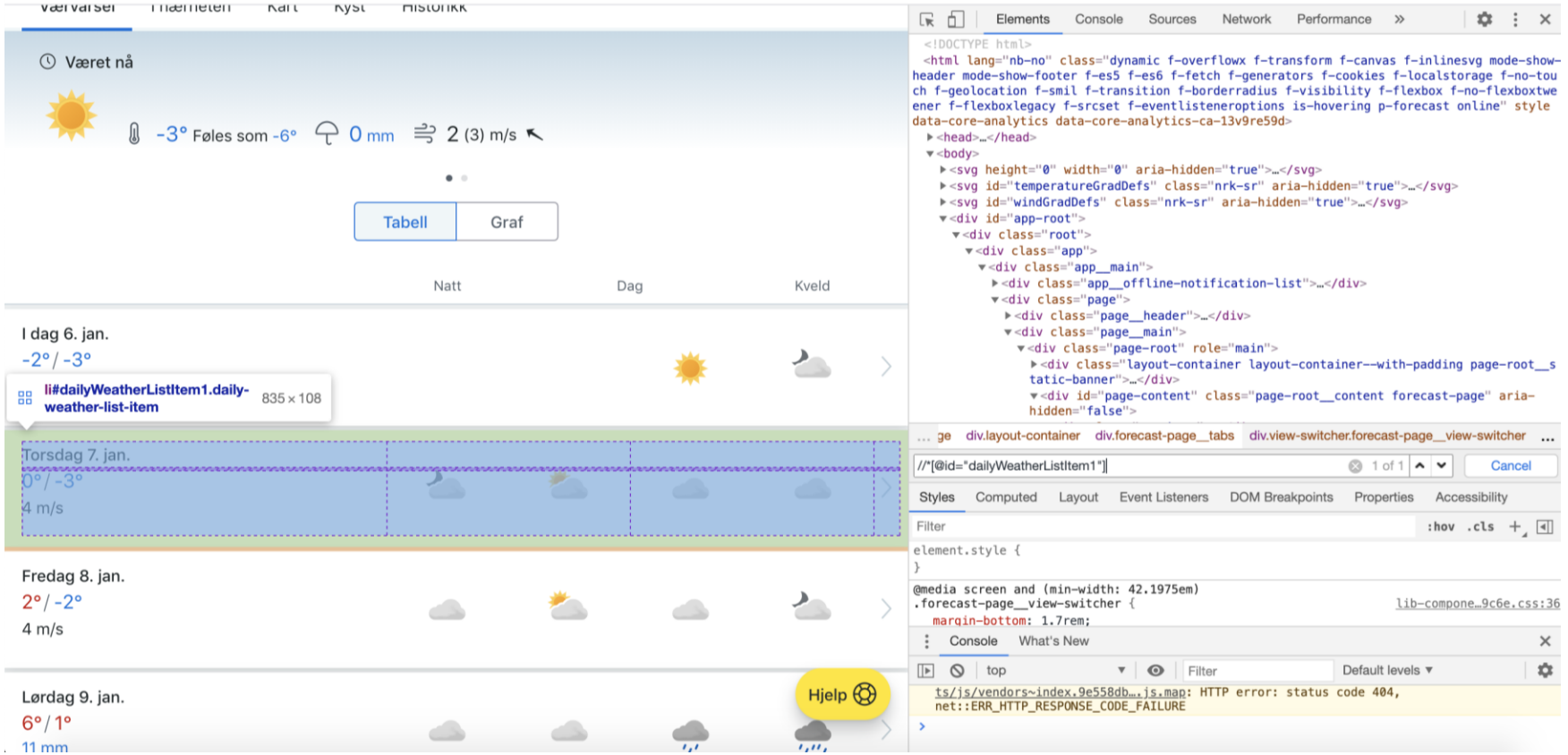
How to get XPath in chrome browser

- **From Chrome :**
- Right click "inspect" on the item you are trying to find the **xpath**.
- Right click on the highlighted area on the console.
- Go to Copy **xpath**.



How to use XPath in chrome browser

- **From Chrome :**
- 1. Right-click "inspect"
- 2. Press CTRL+F
- 3. Type in XPath code



Sample XML

```
<r>
  <e a="1"/>
  <f a="2" b="1">Text 1</f>
  <f/>
  <g>
    <i c="2">Text 2</i>
    Text 3
    <j>Text 4</j>
  </g>
</r>
```

XPath Syntax to select text

Sample XML

```
<r>
  <e a="1"/>
  <f a="2" b="1">Text 1</f>
  <f/>
  <g>
    <i c="2">Text 2</i>
    Text 3
    <j>Text 4</j>
  </g>
</r>
```

Select Text using XPath

Using XPath, we want to select the text node with this string value:
"Text 1"

This XPath,
/r/f/text()
will select the text node with this string value: "Text 1"

XPath Syntax to select text

Sample XML

```
<r>
  <e a="1"/>
  <f a="2" b="1">Text 1</f>
  <f/>
  <g>
    <i c="2">Text 2</i>
    Text 3
    <j>Text 4</j>
  </g>
</r>
```

Select Text using XPath

Using XPath, we want to select the text node with this string value:
"Text 1"

Also, this XPath,
string(/r/f)
will return the string value of f, which is also: "Text 1"

XPath Syntax to find element

Sample XML

```
<r>
  <e a="1"/>
  <f a="2" b="1">Text 1</f>
  <f/>
  <g>
    <i c="2">Text 2</i>
    Text 3
    <j>Text 4</j>
  </g>
</r>
```

Select Element using XPath

This XPath,
/r/e

will select this element:

<e a="1"/>

XPath: Find nodes that have a specific attribute

Syntax

1. Inside a specific node
    - /path to/element[@attribute_name]
2. Anywhere in the document
    - //*[@attribute_name]
3. Inside a specific node with some value
    - /path to/element[@attribute_name='search value']
    - /path to/element[@attribute_name="search value"]
4. Anywhere in the document with some value
    - //*[@attribute_name='search string']
    - //*[@attribute_name="search string"]

XPath Syntax to find nodes that have specific attribute

Sample HTML

```
<html>
    <body>
        <a>link</a>
        <div class='container' id='divone'>
            <p class='common' id='enclosedone'>Element One</p>
            <p class='common' id='enclosedtwo'>Element Two</p>
        </div>
    </body>
</html>
```

Find an element with a specific id in a particular path

This XPath,
/html/body/div/p[@id='enclosedone']

Returns the following element:
Element One

XPath Syntax to find nodes that have specific attribute

Sample HTML

```
<html>
    <body>
        <a>link</a>
        <div class='container' id='divone'>
            <p class='common' id='enclosedone'>Element One</p>
            <p class='common' id='enclosedtwo'>Element Two</p>
        </div>
    </body>
</html>
```

Find an element with a specific id and class in a particular path

This XPath,
//p[@id='enclosedone' and @class='common']

Returns the following element:
Element One

Look into the following webpage:
http://www.pythonscraping.com/pages/warandpeace.html

On this page, the lines spoken by characters in the story are in read, whereas the names of characters are in green.
You can see the span tags, which reference the appropriate CSS classes, in the following sample of the page's source code:

Heavens! What a virulent attack! replied
the prince, not in the least disconcerned
By this reception.

Write XPath expression to select elements that are red.

Absolute vs. Relative path selector

Sample HTML

```
<html>
    <body>
        <a>link</a>
        <div class='container' id='divone'>
            <p class='common' id='enclosedone'>Element One</p>
            <p class='common' id='enclosedtwo'>Element Two</p>
        </div>
    </body>
</html>
```

Absolute XPath

Start from the root element of the document.

/html/body/div/p[@id='enclosedone']

Relative XPath

Start from any point in the document based on the search criteria.

For example:

//p[@id='enclosedone' and @class='common']

XPath Syntax to find nodes that have specific attribute

Sample HTML

```
<html>
    <body>
        <a>link</a>
        <div class='container' id='divone'>
            <p class='common' id='enclosedone'>Element One</p>
            <p class='common' id='enclosedtwo'>Element Two</p>
        </div>
    </body>
</html>
```

Select an element with a specific id in the entire page

This XPath,
//*[@id='divone']

will select this element:

Element One Element Two

XPath Syntax to find an element with a specific attribute value and select the text

Sample HTML

```
<html>
    <body>
        <a>link</a>
        <div class='container' id='divone'>
            <p class='common' id='enclosedone'>Element One</p>
            <p class='common' id='enclosedtwo'>Element Two</p>
        </div>
    </body>
</html>
```

Find an element with a specific id and select the text

This XPath,
//*[@id='enclosedone']/text()

Returns:
Element One

XPath Syntax to find nodes that have specific attribute

Sample XML

```
<Galaxy>
    <name>Milky Way</name>
    <CelestialObject name="Earth" type="planet"/>
    <CelestialObject name="Sun" type="star"/>
</Galaxy>
```

Find nodes with a specific attribute

This XPath,
/Galaxy/*[@name]

or

//*[@name]

Gives the following output:

<CelestialObject name="Earth" type="planet" />
<CelestialObject name="Sun" type="star" />

XPath Syntax to find nodes that have specific attribute value

Sample XML

```
<Galaxy>
    <name>Milky Way</name>
    <CelestialObject name="Earth" type="planet"/>
    <CelestialObject name="Sun" type="star"/>
</Galaxy>
```

Find nodes with a specific attribute value

This XPath,
/Galaxy/*[@name='Sun']

Or

//*[@name='Sun']

Gives the following output:

<CelestialObject name="Sun" type="star" />

XPath Syntax to find nodes that have specific attribute value

Sample XML

```
<Galaxy>
    <name>Milky Way</name>
    <CelestialObject name="Earth" type="planet"/>
    <CelestialObject name="Sun" type="star"/>
</Galaxy>
```

Find nodes by substring matching of an attribute's value

This XPath,
/Galaxy/*[contains(@name,'Ear')]

Or

//*[contains(@name,'Ear')]

Or
/Galaxy/*[contains(@name, "Ear")]

Gives the following output:

<CelestialObject name="Earth" type="planet" />

XPath Syntax to find elements containing specific text

Sample XML

```
<root>
    <element>hello</element>
    <another>
        hello
    </another>
    <example>Hello, <nested> I am an example </nested>.</example>
</root>
```

Find all elements with certain text

This XPath,
//*[text() = 'hello']

Gives the following output:

<element>hello</element>

XPath Syntax to find elements containing specific text

Sample XML

```
<root>
    <element>hello</element>
    <another>
        hello
    </another>
    <example>Hello, <nested> I am an example </nested>.</example>
</root>
```

Find an element containing specific text

This XPath,
//*[contains(text(), 'Hello')]

Gives the following output:

<example>Hello, <nested> I am an example </nested>.</example>

XPath Syntax to find elements containing specific text

Sample XML

```
<root>
    <element>hello</element>
    <another>
        hello
    </another>
    <example>Hello, <nested> I am an example </nested>.</example>
</root>
```

Find text that spans multiple children/text nodes

This XPath,
//*[. = 'Hello, I am an example .']

Gives the following output:

<example>Hello, <nested> I am an example </nested>.</example>

XPath Syntax to get nodes relative to the current node

1. All ancestors of a node
    - /path to the node/ancestor::node()
2. A specific ancestor of a node
    - /path to the node/ancestor::ancestor_name
3. Parent of a node
    - /path to the node/parent::node()
4. Following siblings of a node
    - /path to the node/following-sibling::node()
5. A specific sibling following a node
    - /path to the node/following-sibling::sibling_name
6. Preceding siblings of a node
    - /path to the node/preceding-sibling::node()
7. A specific sibling preceding a node
    - /path to the node/preceding-sibling::sibling_name
8. All immediate child nodes of a node
    - /path to the node/child::node()
9. A specific immediate child node of a node
    - /path to the node/child::chid_name
10. All the descendants of a node
    - /path to the node/descendant::node()
11. All specific descendants of a node
    - /path the to node/descendant::descendant_name

Sample XML

```
<GrandFather name="Bardock" gender="male" spouse="Gine">
    <Dad name="Goku" gender="male" spouse="Chi Chi">
        <Me name="Gohan" gender="male"/>
        <brother name="Goten" gender="male"/>
    </Dad>
</GrandFather>
```

This XPath,
//Me/ancestor::node()

Gives the following output:

Find ancestors of a node

```
<GrandFather name="Bardock" gender="male" spouse="Gine">
  <Dad name="Goku" gender="male" spouse="Chi Chi">
    <Me name="Gohan" gender="male" />
    <brother name="Goten" gender="male" />
  </Dad>
</GrandFather>
<Dad name="Goku" gender="male" spouse="Chi Chi">
  <Me name="Gohan" gender="male" />
  <brother name="Goten" gender="male" />
</Dad>
```

Sample XML

```
<GrandFather name="Bardock" gender="male" spouse="Gine">
    <Dad name="Goku" gender="male" spouse="Chi Chi">
        <Me name="Gohan" gender="male"/>
        <brother name="Goten" gender="male"/>
    </Dad>
</GrandFather>
```

Find a parent node

This XPath,
//Me/ancestor::Dad
Or
//Me/parent::node()

Gives the following output:

```
<Dad name="Goku" gender="male" spouse="Chi Chi">
   <Me name="Gohan" gender="male" />
   <brother name="Goten" gender="male" />
</Dad>
```

Sample XML

```
<GrandFather name="Bardock" gender="male" spouse="Gine">
    <Dad name="Goku" gender="male" spouse="Chi Chi">
        <Me name="Gohan" gender="male"/>
        <brother name="Goten" gender="male"/>
    </Dad>
</GrandFather>
```

Find a specific ancestor node

This XPath,
//Me/ancestor::GrandFather
Or
//Me/parent::node()/parent::node()

Gives the following output:

```
<GrandFather name="Bardock" gender="male" spouse="Gine">
  <Dad name="Goku" gender="male" spouse="Chi Chi">
    <Me name="Gohan" gender="male" />
    <brother name="Goten" gender="male" />
  </Dad>
</GrandFather>
```

Sample XML

```
<GrandFather name="Bardock" gender="male" spouse="Gine">
    <Dad name="Goku" gender="male" spouse="Chi Chi">
        <Me name="Gohan" gender="male"/>
        <brother name="Goten" gender="male"/>
    </Dad>
</GrandFather>
```

Find a sibling (following) node

This XPath,
//Me/following-sibling::brother

Gives the following output:

<brother name="Goten" gender="male" />

Sample XML

```
<Dashavatar>
    <Avatar name="Matsya"/>
    <Avatar name="Kurma"/>
    <Avatar name="Varaha"/>
    <Avatar name="Narasimha"/>
    <Avatar name="Vamana"/>
    <Avatar name="Balabhadra"/>
    <Avatar name="Parashurama"/>
    <Avatar name="Rama"/>
    <Avatar name="Krishna"/>
    <Avatar name="Kalki"/>
</Dashavatar>
```

Find sibling (preceding) nodes

This XPath finds all avatars before Parashurama,
//Avatar[@name='Parashurama']/preceding-sibling::node()

Gives the following output:

<Avatar name="Matsya"/>
<Avatar name="Kurma"/>
<Avatar name="Varaha"/>
<Avatar name="Narasimha"/>
<Avatar name="Vamana"/>
<Avatar name="Balabhadra"/>

Sample XML

```
<Dashavatar>
    <Avatar name="Matsya"/>
    <Avatar name="Kurma"/>
    <Avatar name="Varaha"/>
    <Avatar name="Narasimha"/>
    <Avatar name="Vamana"/>
    <Avatar name="Balabhadra"/>
    <Avatar name="Parashurama"/>
    <Avatar name="Rama"/>
    <Avatar name="Krishna"/>
    <Avatar name="Kalki"/>
</Dashavatar>
```

Find sibling (following) nodes

This XPath finds all avatars after Parashurama,
//Avatar[@name='Parashurama']/following-sibling::node()

Gives the following output:

<Avatar name="Rama" />
<Avatar name="Krishna" />
<Avatar name="Kalki" />

Sample XML

```
<Dashavatar>
    <Avatar name="Matsya"/>
    <Avatar name="Kurma"/>
    <Avatar name="Varaha"/>
    <Avatar name="Narasimha"/>
    <Avatar name="Vamana"/>
    <Avatar name="Balabhadra"/>
    <Avatar name="Parashurama"/>
    <Avatar name="Rama"/>
    <Avatar name="Krishna"/>
    <Avatar name="Kalki"/>
</Dashavatar>
```

Find all nodes except current one

This XPath finds all avatars except the current one (Parashurama),
//Avatar[@name='Parashurama']/following-sibling::Avatar |
//Avatar[@name='Parashurama']/preceding-sibling::Avatar

Gives the following output:

<Avatar name="Matsya" />
<Avatar name="Kurma" />
<Avatar name="Varaha" />
<Avatar name="Narasimha" />
<Avatar name="Vamana" />
<Avatar name="Balabhadra" />
<Avatar name="Rama" />
<Avatar name="Krishna" />
<Avatar name="Kalki" />

Sample XML

```
<House>
   <Rooms>10</Rooms>
   <People>4</People>
   <TVs>4</TVs>
   <Floors>2</Floors>
</House>
```

Find all child nodes

This XPath finds all the details (child nodes) of House,
/House/child::node()

Gives the following output:

```
<Rooms>10</Rooms>
<People>4</People>
<TVs>4</TVs>
<Floors>2</Floors>
```

Sample XML

```
<House>
   <numRooms>4</numRooms>
   <Room name="living"/>
   <Room name="master bedroom"/>
   <Room name="kids' bedroom"/>
   <Room name="kitchen"/>
</House>
```

Find all immadiate children of a node

This XPath finds all rooms (immediate children named Room) in House,
/House/child::Room
Or
/House/*[local-name()='Room']

Gives the following output:

```
<Room name="living" />
<Room name="master bedroom" />
<Room name="kids' bedroom" />
<Room name="kitchen" />
```

Sample XML

```
<House>
    <numRooms>4</numRooms>
    <Floor number="1">
        <Room name="living"/>
        <Room name="kitchen"/>
    </Floor>
    <Floor number="2">
        <Room name="master bedroom"/>
        <Room name="kids' bedroom"/>
    </Floor>
</House>
```

Find all children (regardless of position) of a node

This XPath finds all rooms (irrespective of the position) in House,
/House/descendant::Room

Gives the following output:

```
<Room name="living" />
<Room name="kitchen" />
<Room name="master bedroom" />
<Room name="kids' bedroom" />
```

XPath Syntax to get the count of nodes

Sample XML

```
<Goku>
  <child name="Gohan"/>
  <child name="Goten"/>
</Goku>
```

Count all immediate children of a node

This XPath finds the number of children that Goku have,
count(/Goku/child)

Gives the following output:

2.0

XPath Syntax to get the count of nodes

Sample XML

```
<House>
    <LivingRoom>
        <plant name="rose"/>
    </LivingRoom>
    <TerraceGarden>
        <plant name="passion fruit"/>
        <plant name="lily"/>
        <plant name="golden duranta"/>
    </TerraceGarden>
</House>
```

Count all children of a node

This XPath finds the number of plants that the House node have,
count(/House//plant)

Gives the following output:

4.0

XPath Syntax to select nodes based on their children

Sample XML

```
<Students>
  <Student>
    <Name>
      <First>Ashley</First>
      <Last>Smith</Last>
    </Name>
    <Grades>
      <Exam1>A</Exam1>
      <Exam2>B</Exam2>
      <Final>A</Final>
    </Grades>
  </Student>
  <Student>
    <Name>
      <First>Bill</First>
      <Last>Edwards</Last>
    </Name>
    <Grades>
      <Exam1>A</Exam1>
    </Grades>
  </Student>
</Students>
```

Select nodes based on child count

This XPath finds all students that have at least 2 grades recorded,

```
//Student[count(./Grades/*) > 1]
```

Gives the following output:

```
<Student>
    <Name>
      <First>Ashley</First>
      <Last>Smith</Last>
    </Name>
    <Grades>
      <Exam1>A</Exam1>
      <Exam2>B</Exam2>
      <Final>A</Final>
    </Grades>
</Student>
```

XPath Syntax to select nodes based on their children

Sample XML

```
<Students>
  <Student>
    <Name>
      <First>Ashley</First>
      <Last>Smith</Last>
    </Name>
    <Grades>
      <Exam1>A</Exam1>
      <Exam2>B</Exam2>
      <Final>A</Final>
    </Grades>
  </Student>
  <Student>
    <Name>
      <First>Bill</First>
      <Last>Edwards</Last>
    </Name>
    <Grades>
      <Exam1>A</Exam1>
    </Grades>
  </Student>
</Students>
```

Select nodes based on specific child node

This XPath selects all students that have a score for Exam2 recorded,

```
//Student[./Grades/Exam2]
```

Or

```
//Student[.//Exam2]
```

Gives the following output:

```
  <Student>
    <Name>
      <First>Ashley</First>
      <Last>Smith</Last>
    </Name>
    <Grades>
      <Exam1>A</Exam1>
      <Exam2>B</Exam2>
      <Final>A</Final>
    </Grades>
  </Student>
```

XPath Syntax to select nodes with names equal to or containing some string

Sample XML

```
<Galaxy>
    <Light>sun</Light>
    <Device>satellite</Device>
    <Sensor>human</Sensor>
    <Name>Milky Way</Name>
</Galaxy>
```

Select nodes based on node name

This XPath searches for nodes with name as Light, Device or Sensor,

```
/Galaxy/*[local-name()='Light' or local-name()='Device' or local-name()='Sensor']
```

Or

```
//*[local-name()='Light' or local-name()='Device' or local-name()='Sensor']
```

Gives the following output:

```
<Light>sun</Light>
<Device>satellite</Device>
<Sensor>human</Sensor>
```

XPath Syntax to select nodes with names equal to or containing some string

Sample XML

```
<Galaxy>
    <Light>sun</Light>
    <Device>satellite</Device>
    <Sensor>human</Sensor>
    <Name>Milky Way</Name>
</Galaxy>
```

Select nodes based on node name

This XPath searches for nodes nodes with name light (case insensitive),

```
/Galaxy/*[lower-case(local-name())="light"]
```

Or

```
//*[lower-case(local-name())="light"]
```

Gives the following output:

```
<Light>sun</Light>
```

XPath Syntax to select nodes with names equal to or containing some string

Sample XML

```
<Data>
   <BioLight>
      <name>Firefly</name>
      <model>Insect</model>
   </BioLight>
   <ArtificialLight>
      <name>Fire</name>
      <model>Natural element</model>
      <source>flint</source>
   </ArtificialLight>
   <SolarLight>
      <name>Sun</name>
      <model>Star</model>
      <source>helium</source>
   </SolarLight>
</Data>
```

Select nodes based on node name

This XPath searches for nodes with name as Light, Device or Sensor,

```
/Data/*[contains(local-name(),"Light")]
```

Or

```
//*[contains(local-name(),"Light")]
```

Gives the following output:

```
<BioLight>
   <name>Firefly</name>
   <model>Insect</model>
</BioLight>
<ArtificialLight>
   <name>Fire</name>
   <model>Natural element</model>
   <source>flint</source>
</ArtificialLight>
<SolarLight>
   <name>Sun</name>
   <model>Star</model>
   <source>helium</source>
</SolarLight>
```

XPath Syntax to select nodes with names equal to or containing some string

Sample XML

```
<College>
   <FootBall>
      <Members>20</Members>
      <Coach>Archie  Theron</Coach>
      <Name>Wild cats</Name>
      <StarFootballer>David  Perry</StarFootballer>
   </FootBall>
   <Academics>
      <Members>100</Members>
      <Teacher>Tim  Jose</Teacher>
      <Class>VII</Class>
      <StarPerformer>Lindsay  Rowen</StarPerformer>
   </Academics>
</College>
```

Select nodes based on node name

This XPath searches for nodes that has name that starts with Star,

```
/College/*/*[starts-with(local-name(),"Star")]
```

Or

```
//*[starts-with(local-name(),"Star")]
```

Gives the following output:

```
<StarFootballer>David  Perry</StarFootballer>
<StarPerformer>Lindsay  Rowen</StarPerformer>
```

XPath Syntax to select nodes with names equal to or containing some string

Sample XML

```
<College>
   <FootBall>
      <Members>20</Members>
      <Coach>Archie Theron</Coach>
      <Name>Wild cats</Name>
      <StarPlayer>David Perry</StarPlayer>
   </FootBall>
   <VolleyBall>
      <Members>24</Members>
      <Coach>Tim Jose</Coach>
      <Name>Avengers</Name>
      <StarPlayer>Lindsay Rowen</StarPlayer>
   </VolleyBall>
   <FoosBall>
      <Members>22</Members>
      <Coach>Rahul Mehra</Coach>
      <Name>Playerz</Name>
      <StarPlayer>Amanda Ren</StarPlayer>
   </FoosBall>
</College>
```

Select nodes based on node name

This XPath searches for nodes that has name that ends with Ball,

```
/College/*[ends-with(local-name(),"Ball")]
Or
//*[ends-with(local-name(),"Ball")]
```

Gives the following output:

```
<FootBall>
   <Members>20</Members>
   <Coach>Archie Theron</Coach>
   <Name>Wild cats</Name>
   <StarPlayer>David Perry</StarPlayer>
</FootBall>
<VolleyBall>
   <Members>24</Members>
   <Coach>Tim Jose</Coach>
   <Name>Avengers</Name>
   <StarPlayer>Lindsay Rowen</StarPlayer>
</VolleyBall>
<FoosBall>
   <Members>22</Members>
   <Coach>Rahul Mehra</Coach>
   <Name>Playerz</Name>
   <StarPlayer>Amanda Ren</StarPlayer>
</FoosBall>
```

XPath Syntax to select nodes with names equal to or containing some string

Sample XML

```
<College>
   <FootBall>
      <Members>20</Members>
      <Coach>Archie  Theron</Coach>
      <Name>Wild cats</Name>
      <StarPlayer>David  Perry</StarPlayer>
   </FootBall>
   <VolleyBall>
      <Members>24</Members>
      <Coach>Tim  Jose</Coach>
      <Name>Avengers</Name>
      <StarPlayer>Lindsay  Rowen</StarPlayer>
   </VolleyBall>
   <FoosBall>
      <Members>22</Members>
      <Coach>Rahul  Mehra</Coach>
      <Name>Playerz</Name>
      <StarPlayer>Amanda  Ren</StarPlayer>
   </FoosBall>
</College>
```

Select nodes based on node name

Write an XPath expression that searches for nodes that has name that ends with Player,

Write the output below:

Obligatory Exercise

Useful links:

- https://www.w3schools.com/xml/xpath_examples.asp
- https://topswagcode.com/xpath/
- https://www.wireshark.org/download/docs/Wireshark User%27s Guide.pdf