



# Programming on the web

CSC309

Kianoosh Abbasi

# Why take a web programming course?

- It **revolutionized** the world
  - Every human being's life has changed prior to twenty years ago
- Life is **unimaginable** without internet
- Things are **connected** and **seamless**
- The technology behind is truly fascinating
  - We'll learn it in this course

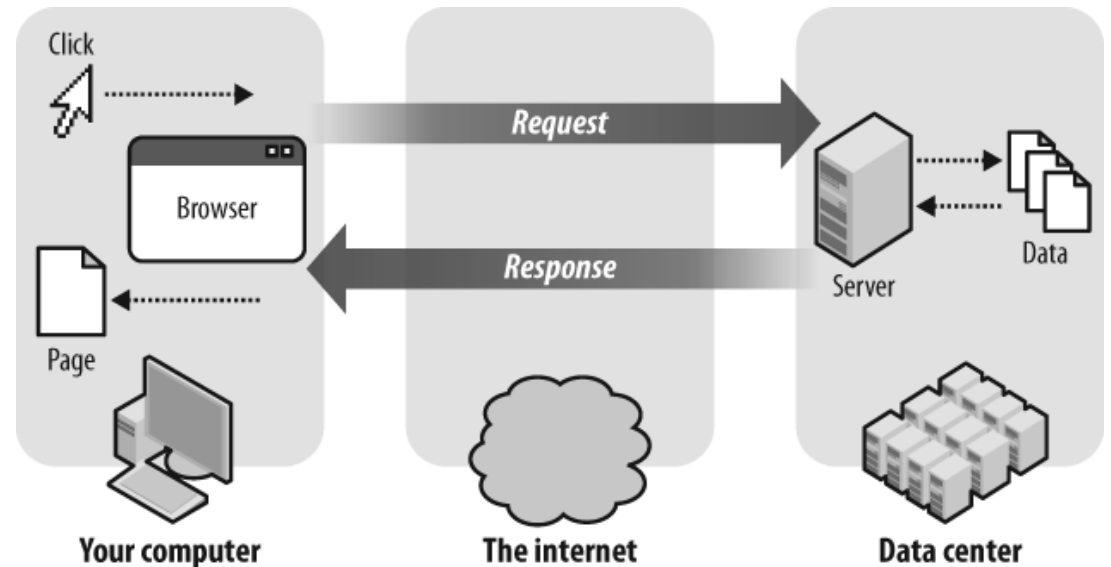


# What is it about?

- How does **web** work?
  - Client/server, browsers, protocols
- Components of a website
  - Server, backend, frontend
- Software **design**
  - Design models, frameworks, data management
- Website **deployment**
  - Make your website accessible to the world!

# How web works

- A lot of things happen when a single **webpage** is loaded!
- Lots of HTML/CSS/JS is fetched
- All in the form of **requests** & **responses**
  - Browser (**client**) sends requests to one or more **servers** and receives responses





# How web works

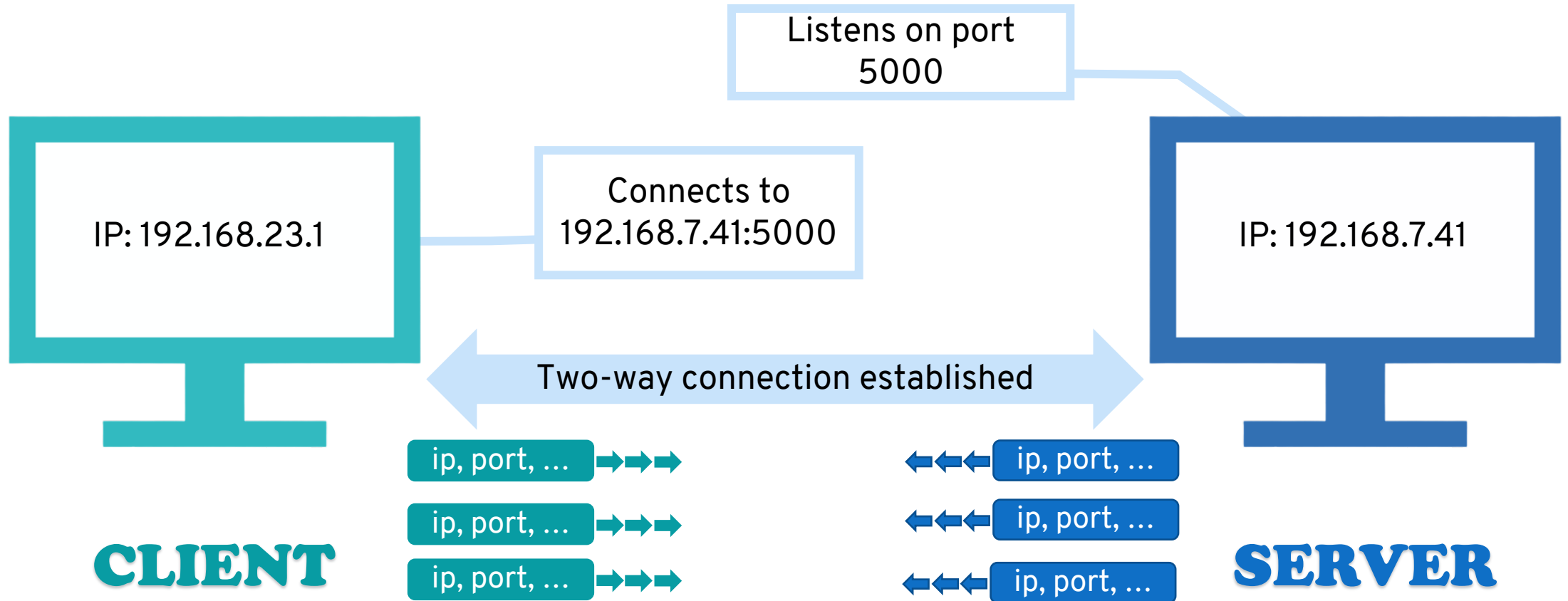
The screenshot shows a web browser displaying the University of Toronto Computer Science website. The address bar shows `https://web.cs.toronto.edu`, which is circled in red. The website header includes the University of Toronto logo and navigation links: Home, About, Undergraduate, Graduate, Research, News & Events, People, and Contact Us. The main content area features a message about administrative staff availability and a link to COVID-19 updates, followed by a large image of the Toronto skyline.

The browser's developer tools are open to the Network tab, showing a list of requests. The 'Domain' column is circled in red, and the '45 requests' summary at the bottom left is also circled in red.

Status	Method	Domain	File	Initiator	Type	Transferred	Size	0 ms	2.56 s	5.12 s	7.68 s
200	GET	web.cs.toronto.edu	/	document	html	20.66 kB	104.90 kB	83 ms			
200	GET	use.typekit.net	WhQmG4e5xihD2s2Be7yvGea3coDHZslayFyJfBfheKvfeCtiffHN4UJLFRbh52jhWDMR5e9ojQkwD9hwewDFe	script	js	7.24 kB	18.34 kB	3 ms			
304	GET	assets.squarespace.com	modern.js		js	cached	80.12 kB	19 ms			
200	GET	assets.squarespace.com	moment-js-vendor-26ddeab7fa5f90b6c8cb3-min.en-US.js	script	js	45.69 kB	251.71 kB	19 ms			
200	GET	assets.squarespace.com	ddr-resource-pack-be81d1ce004cbca505842-min.en-US.js	script	js	24.87 kB	118.94 kB	16 ms			
200	GET	assets.squarespace.com	common-vendors-stable-5f58a0e5b599c258afba7-min.en-US.js	script	js	76.61 kB	243.17 kB	22 ms			
200	GET	assets.squarespace.com	common-vendors-a15d3b6e09e0c8a937ea6-min.en-US.js	script	js	168.76 kB	585.13 kB	34 ms			
200	GET	assets.squarespace.com	common-be3203642cb72770e4c89-min.en-US.js	script	js	188.02 kB	747.99 kB	33 ms			
200	GET	assets.squarespace.com	performance-bc3576d7eca79df62e49-min.en-US.js	script	js	14.27 kB	45.70 kB	17 ms			
304	GET	ajax.googleapis.com	jquery.min.js	script	js	cached	93.54 kB	2 ms			
304	GET	ajax.googleapis.com	jquery.min.js	script	js	cached	82.40 kB	5 ms			
200	GET	www.googletagmanager.com	js?id=G-9CDDN8N95H	script	js	60.96 kB	163.12 kB	40 ms			
304	GET	code.jquery.com	jquery-3.5.0.min.js	script	js	cached	87.40 kB	17 ms			
304	GET	static1.squarespace.com	site-bundle.js	script	js	cached	57.09 kB	11 ms			
200	GET	assets.squarewebsites.org	style.css	stylesheet	css	2.93 kB (raced)	9.29 kB	45 ms			
304	GET	assets.squarewebsites.org	custom-table.js	script	js	cached	14.40 kB	40 ms			
304	GET	cdnjs.cloudflare.com	jquery.min.js	script	js	cached	87.40 kB	16 ms			
302	GET	web.cs.toronto.edu	plugin-accotabs.js	script	js	34.60 kB (raced)	124.39 kB	37 ms			
302	GET	web.cs.toronto.edu	plugin-lightbox.js	script	js	72.93 kB (raced)	228.19 kB	39 ms			

45 requests 4.75 MB / 1.38 MB transferred Finish: 6.31 s DOMContentLoaded: 951 ms load: 1.47 s

# How computers talk to each other?



# Domains

- Mapped to IP addresses
  - www.google.com -> 142.251.41.78
- Stored in Domain Name Servers (DNS)
- Clients first resolve the domain, then connect to the IP address
- Already knows which DNS server to talk to



# Stateful vs Stateless

Two-way open connection is **stateful**

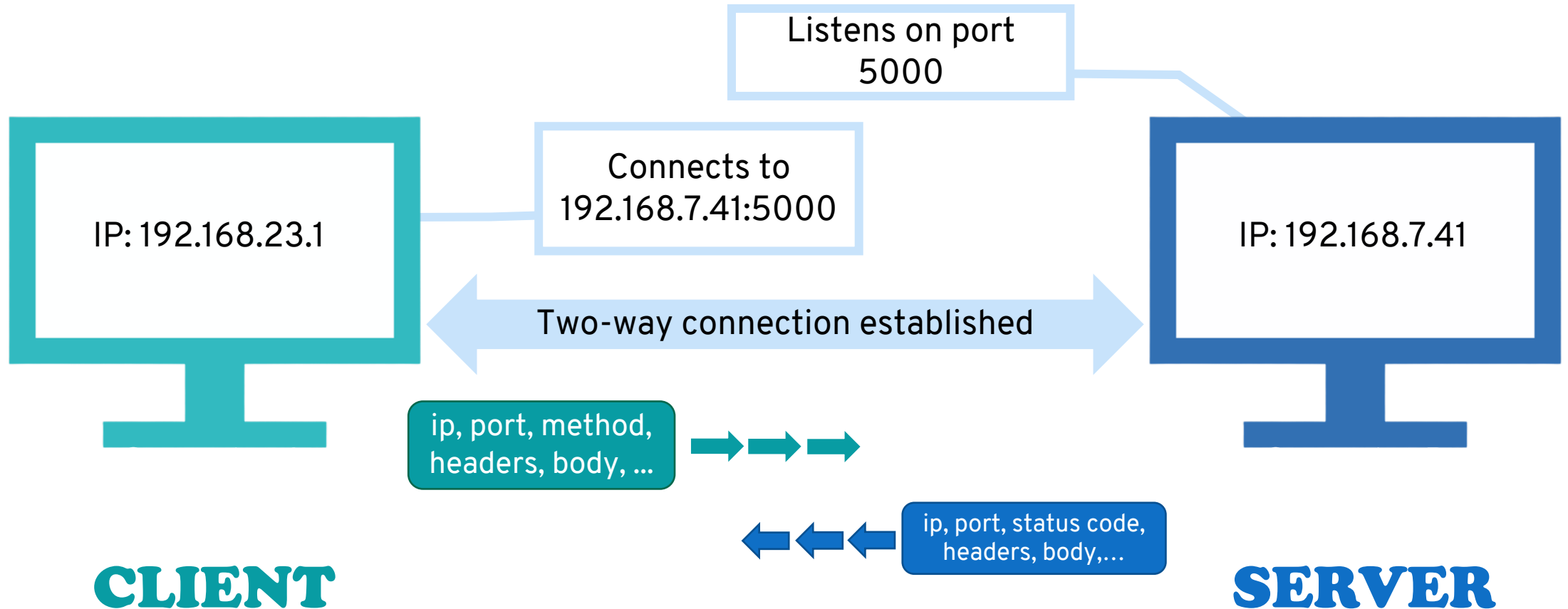
What the server responds depends on **previous messages**

Server should keep track of **thousands** of open connections

If connection breaks, all the state is **lost**

A **stateless** protocol is preferred

# Stateless Protocol



**HyperText Transfer Protocol (HTTP)**

# HTTP Message

- A **string** with a special **format**
- Request a more specific target
  - **Path**: /, /signup, /account/index.html, ...
  - **Method**: GET, POST, PUT, ...
- Headers & Body
- Default port is **80**

# HTTP Message

## Requests

```
POST / HTTP/1.1
Host: localhost:8000
User-Agent: Mozilla/5.0 (Macintosh;... )... Firefox/51.0
Accept: text/html,application/xhtml+xml,..., */*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Upgrade-Insecure-Requests: 1
Content-Type: multipart/form-data; boundary=-12656974
Content-Length: 345
```

```
-12656974
(more data)
```

## Responses

```
HTTP/1.1 403 Forbidden
Server: Apache
Content-Type: text/html; charset=iso-8859-1
Date: Wed, 10 Aug 2016 09:23:25 GMT
Keep-Alive: timeout=5, max=1000
Connection: Keep-Alive
Age: 3464
Date: Wed, 10 Aug 2016 09:46:25 GMT
X-Cache-Info: caching
Content-Length: 220
```

```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML
2.0//EN">
(more data)
```

start-line

HTTP headers

empty line

body

# Response codes

- **Success: 200-299**
  - 200 OK, 201 Created
- **Redirection: 300-399**
  - 301 moved Permanently
- **Client errors: 400-499**
  - 404 Not Found, 400 Bad Request, 403 Permission Denied
- **Server errors: 500-599**
  - 500 Internal Server Error, 502 Bad Gateway

# HTML

A specific form of Extensible Markup Language (XML)

- Data is annotated with nested tags
- HTML has specific tags for a webpage to describe what the page contains
- More on HTML later today

# Web browser

- Upon entering the **Uniform Resource Locator (URL)**
  - Connects, sends requests to server, and receives responses
- **Renders** the response
  - HTML
  - Image
  - PDF
  - Text



# So far...

- Server **listens** on a specific port, client(s) connect to IP and port
- Stateless HTTP protocol: **Request** & **Response**
- HTTP response body can be in **HTML** format
- Browsers understand this format and **renders** accordingly

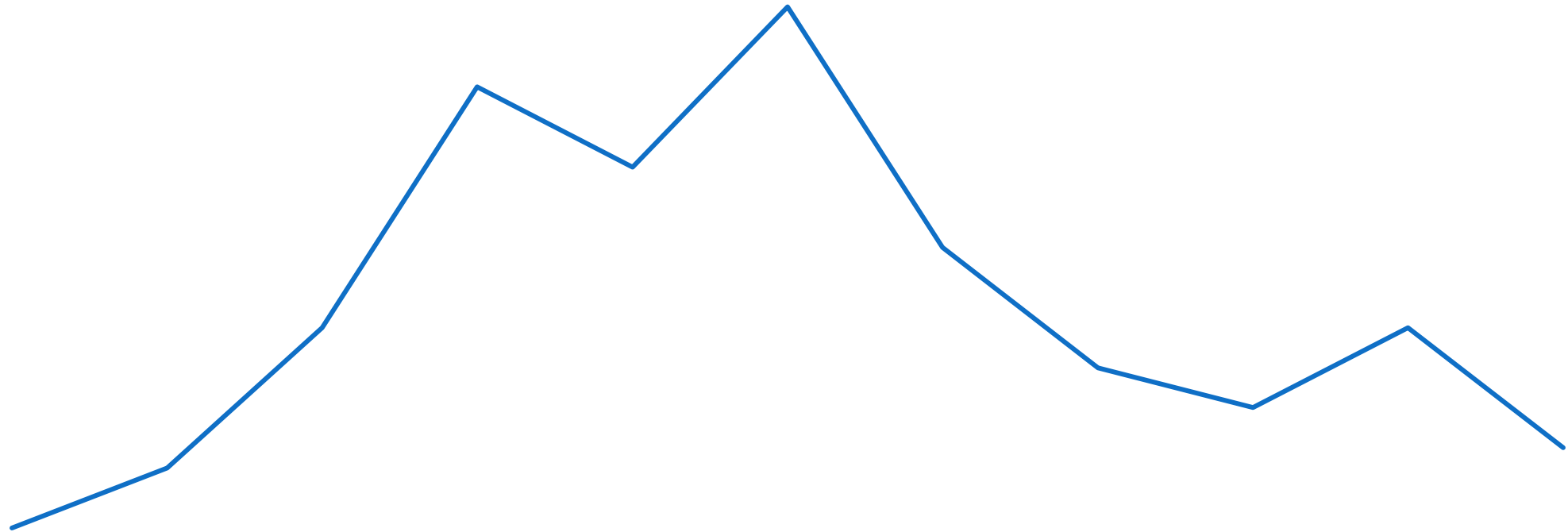
# CSC309

- No **prior** knowledge/experience in web development is assumed
  - We'll start from scratch!
- However, the course is quite **up-to-date**!
  - Web in 2024 is so different than in 2000!
  - We will **cover** latest technologies/frameworks
- It's fast-paced!

# Topics

- Week 1
  - Course intro, web architecture
  - [HTML](#), [CSS](#)
- Week 2
  - [JavaScript](#)
- Week 3-6
  - Backend development
  - [Node.js](#), [Next.js](#)
  - [Database](#) models & ORM
  - [Restful](#) APIs
  - Authentication
- Week 7-9
  - Frontend development
  - Single-page applications
  - [React](#)
  - Advanced styling
  - [Typing](#) in JavaScript (TypeScript)
- Week 10
  - [Docker](#)
- Week 11
  - [Deployment](#) & DevOps

# Difficulty levels throughout the term



# Course delivery

- Tuesdays and Thursdays 5-7
  - 7-8: Office hours, tutorials, lecture continuation, ...
- Two sections are the same
  - Attend either lectures
  - Same TA's, assessments, etc.

# Assessment

- Weekly exercises (10%)
  - Educational questions
  - Supplementing lecture material
  - Best 9 out of 11 will count
- Project (50%)
  - A full, real-world web app!
  - Two parts (PP1 20% - PP2 30%)
- Midterm (10%)
  - Oct 1st and 3rd
  - During lecture time
- Final exam (30%)
  - 40% required for passing the course

# Weekly exercises

- Small, **educational** questions regarding the most recent lecture
  - Designed to provide initial **experience** with tools
- Auto-graded
  - Auto-grader will be **given** to you!
- Deadline: every **Thursday at 3pm.**
  - No extension, tokens, remark request



# Project

- A **full** web app
- In groups of up to 3 people
  - You may team up with people from the either sections
- Two parts
  - **PP1**: Next.js backend, **PP2**: Next.js frontend & Docker
- Mentor sessions with TA's

# Project

- Each part is graded **separately** via TA interviews
  - Overall functionality of the app (**shared** within group)
  - Team member's contribution and participation (**individual**)
- Start looking for teammates now!
- Use best tools out there!
  - Open-source codes, **ChatGPT**, etc.
  - No **code sharing** between groups

# Exams

- Closed-book, paper-based exams
  - One piece of hand-written cheat-sheet allowed!
- Challenges your understanding of concepts
  - Not your memory!
  - No need to memorize tags, classes, syntax, etc.
- Midterm (10%)
  - During lecture time in week 5
- Final exam (30%)

# Contact points

- Course website  
[www.cs.toronto.edu/~kianoosh/courses/csc309/](http://www.cs.toronto.edu/~kianoosh/courses/csc309/)
- Piazza: Announcements + Q&A  
[piazza.com/utoronto.ca/fall2024/csc309](https://piazza.com/utoronto.ca/fall2024/csc309)
- Announcements are **NOT** copied to Quercus  
to avoid duplication
- Email:  
[csc309-2024-09@cs.toronto.edu](mailto:csc309-2024-09@cs.toronto.edu)
- Discord server  
<https://discord.gg/XjY5SnFwhY>  
Informal Q&A and chat
- Office hours
  - After lectures (7-8)

# Course schedule

Lecture Number	Class Dates	Title	Important Events
L01	Sep 3rd, 5th	Intro, HTML, CSS	
L02	Sep 10th, 12th	JavaScript	
L03	Sep 17th, 19th	Node.js, API, Next.js	
L04	Sep 24th, 26th	Async, Models, ORM	
L05	Oct 1st, 3rd	CRUD	Midterm (10%)
L06	Oct 8th, 10th	Auth, migrations	
L07	Oct 15th, 17th	React, JSX	
L08	Oct 22nd, 24th	Monorepo, hooks	PP1 (20%)
Reading week			
L09	Nov 4th, 6th	Typing, advanced CSS	
L10	Nov 11th, 13th	Docker	
L11	Nov 18th, 20st	Deployment	PP2 (30%)
L12	Nov 25th, 27th	Optional topics	

# Academic integrity

- University's policy takes it **very seriously**
  - Violations may result in **failing** the course
- Rules
  - Exams are closed-book and paper-based
  - No interaction with other students during exams
  - No **code sharing** between groups in the project
  - Cite all open-source or **AI-generated** codes

# Questions?



# HTML

- A specific text format understood by browsers
- HTML file surrounded by the `<html>` tag
  - `<body>` and `<head>` tags
- Tags and elements
- Elements can have attributes

# HTML tags

Visit <https://www.w3schools.com/html/>

- Headings: `<h1>` to `<h6>`
- Paragraphs: `<p>`
- Links: `<a>`
  - Stands for anchor
- Images: `<img />`
- Lists: `<ol>` and `<ul>`
- Tables: `<table>`
- Navigation Bar: `<nav>`
- New line: `<br />`

# HTML attributes

- `style` attribute
  - Discussed shortly later
- Identifiers: `id` vs `class`
- Other attributes: `src` for `<img />` and `href` for `<a>`
- You can put any custom attribute you want
  - There's no compilation error!

# Other HTML tags

- `<div>` and `<span>`
- Select part of document to apply **specific attributes**
- `<span>`: inline organization
- `<div>`: block-level organization

# Forms

- Primary way to send user data to server
- On submit, a **request** is often sent
- Comprised of many **inputs**

## Create Binance Account

Register with your email or mobile

Email

Mobile

Email

Password

Referral ID (Optional) ▼

☐ Subscribe to email updates

☒ I have read and agree to Binance's [Terms of Service](#)

Create Account

Already registered? [Log In](#)

# Inputs

- Text field  
`<input type="text" />`
- Passwords, emails, etc.  
`<input type="password" />`
- Radio button  
`<input type="radio" />`
- Checkbox  
`<input type="checkbox" />`
- Textarea  
`<textarea/>`
- Submit button  
`<button type="submit" />`

# Forms

- **Action** attribute defines the URL/path of the HTTP request
- **Method** attribute: HTTP method parameter
- Inputs: **name** and **value** attributes



# GET vs POST

- GET is usually used for **queries** and **retrievals**
  - Google search
- The **query params** are appended to the end of the URL
  - Why?
- POST: sending **private user data** (name, password, etc.)
  - How much private actually?

# CSS styles

- **Style** attribute
  - **Describes** how the element should look like
- Usage:  

```
<h1 style="color: red"> Hello </h1>  
<div style="width: 50px; height: 60px"> ... </div>
```

# Basic CSS properties

- color
- background-color
- font-size
- text-align
- width
- height
- z-index
- font-family
- list-style-type
- opacity

# Stylesheets

- **Style tag**
  - Makes styles **reusable** by defining **selectors**
- **Stylesheets**
  - A separate CSS file with the content of style tags
  - Imported via `<link rel="stylesheet" href="style.css" />`

# Selectors

- Same styles can be applied to an **arbitrary** set of elements
- All elements of a certain **tag**
  - All elements (the `<html>` element)
- A **specific** element: the **id** attribute
- A **set** of elements: the **class** attribute

# Selectors

- CSS content (inside <style> or stylesheet)

```
/* Makes all paragraphs red */  
p {  
    color: red;  
}
```

```
/* Specifies the size of the element with id `big-text` */  
#big-text {  
    font-family: Arial;  
    font-size: 20px;  
}
```

```
/* Aligns all elements from class `center` */  
.center {  
    text-align: center;  
}
```

# Precedence

Visit <https://wattenberger.com/blog/css-cascade>

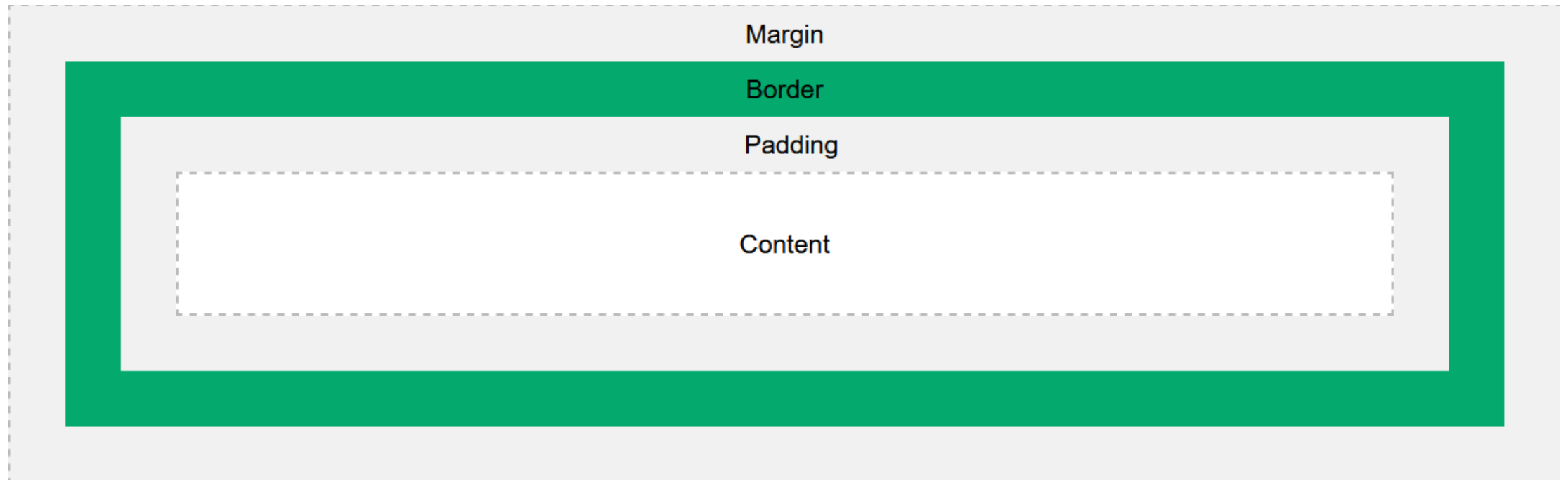
- More **specific** selectors **overrides** less specific ones
  - #ID > .class > tag
  - Inline css > style tag > css file
- If everything is the same
  - **Later** rules **override** earlier ones
- To **evade** these:
  - Use the **!important** keyword

# Units

- For width, margin, font-size, etc.
- **Absolute** lengths
  - cm, in, px
- **Relative** lengths
  - %,
  - vw: 1% of **viewpoint** width
  - em: element's **font-size**,
  - rem: **root** element's font-size (usually for font-size)
  - fr: fraction (of the available space)



# Box Model



# Spacing

- Border

```
border-style: solid/dotted/...  
border-width: 10px/thin/0px  
border-color: red/white/#fa23ca  
border-radius: 5px
```

- The shortcut

```
border: 10px solid red;
```

- Style a specific edge

```
border-top-color, border-left-width
```

- Combine edges

- Top-right-bottom-left

```
border-width: 1px 2px 3px 5px;
```

- Alternative

- Top & bottom - left & right

```
border-width: 1px 2px;
```

# Next session

- JavaScript history and syntax
- Scope and closure
- Dynamic web pages