HTML

Basic Tags: <…> </…>
html, head, title, body, br, hr

Test Tags: sub, sup, pre, h1, h6, span
i # Italic strong, b # Bold, u # Underline

Links: href="xxx.com/#section1"

Svg image:
<svg width="200" height="200">
<circle cx="100" cy="100" r="50" fill="red" />
</svg>

Forms:
<form> </form>
<textarea rows=""> </textarea>

Nav Bar:
```
<ul>
  <li><a href="default.asp">Home</a></li>
  <li><a href="news.asp">News</a></li>
  <li><a href="contact.asp">Contact</a></li>
  <li><a href="about.asp">About</a></li>
</ul>
```

Graphical: # alt states text
<img src="" alt= height= weight= >

Lists: <ol start="5">
<ul type="circle/square">
<ol type="1/A/a/I/i">

Special character:
< >   # No line break

Tables: <table> </table> <tr> </tr>
<td> </td> # cell/data <th> </th> # header

Favicon:
<link rel="icon"> type="image/x-icon"
href="…">

Forms Input:
<fieldset> <legend> Choose your player name </legend> </fieldset>
<label for="typeName"> Enter your email/name: </label>
<input type="checkbox/radio/text/email/password/date/color/url/tel/search" />
<input type="submit" value="Subscribe!" required/ readonly/ disabled/ autofocus />
<button type="submit/reset"> # value: initial value of the input box

---

CSS Internal: <style> … </style> / Importance: !important>inline>internal>external

<link rel="stylesheet" href=""> # link to External CSS
<meta name="viewport" content="width=device-width, initial-scale=1">
<class="btn btn-basic/default/primary/success/info/warning/danger">
<class="col-sm-1 col-xl-4"># class for div, 12 columns, sm/md/lg/xl/xxl

Responsive Layout:
Show contents differently based on device or screen size. (mobile-first)

display: inline/inline-block/block/none
text-align: left/right/center
font: font-style: /font-family: / font-size:
Visibility: none
position: static/relative/fixed/absolute/sticky
Color: color/background-color:
border-style: dotted/dashed/solid/double/none/hidden

Applying style in .css file:
.classname{} #idname{}
div.classname{}
div p {} # Select <p> in
p:hover{} a:link{}
p: nth-child(3) / p::first-letter

JS External: <script src="xx.js"> </script>, Internal: put before </body>, Index starts with 0

Data types: string, number, bigint, boolean, undefined, symbol, null
Conversion: Number(…) ; String(…) ; Boolean(…)

HTML output: document.write(…)
Popup boxes: window.alert/confirm/prompt(…)
Not CSS skinnable

String methods:
trim() # delete space
split(…) # Separator
Output formatting:
console.log(`I am ${name}`)

Regular Expression:

- Create a RegEx between / /
- ^ asserts the start of the string
- [^\s@]+ matches one or more characters that are not whitespace or @ symbol
  - [^abc] matches any single characters that is not a, b, or c.
  - [^abc]+ matches one or more characters that is not a, b, or c.
- @ matches the @ symbol
- [^\s@]+ again
- \. matches the dot character
- [^\s@]+ again
- $ asserts the end of the string

```
const regex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;
              username domain  com
```

JS (Cont'd)

Loops: for(x of cars) {} – arrays/strings
for(x in person) {} – key-value pairs
while(){} / do{} while() / continue;

Arrow Function: No this & arguments
hello = () => {return "Hello World"} /
hello = () => "Hello World"
# If only return statement

Scheduling calls:
const *x* = setTimeout(function(){…},2000)
clearTimeout(*x*)
const *x* = setInterval(function, sec)
clearInterval(*x*)

Conditional:
if(){} else{} / ? … : …
switch(express){
case 1: … break;
case 2: … break;
default: … break; }

GET: Data is visible in URL / Request stays in browser history
POST: Data is embedded in HTTP request body

Events:
<button event="hello()"></button>
onclick/onload/onunload/
onchange/onmouseover/onfocus

Input Validation (Form):
var input = document.getElementByID(…)
if(regex.test(input)) {alert("Email is Valid")}

Fetch:

```
fetch('https://www.google.com')  fetch() create a Promise object automatically
  .then((response) => { if success
    console.log(response.status);
  })
  .catch((error) => { if reject
    console.log(error);
  })
  .finally(() => { execute it anyway
    document.querySelector('#spinner').style.display='none';
  });
```

Accessing HTML elements:
querySelector() # Return 1st matc
querySelectorAll().style.color
getElementByID().innerHTML
getElementByClassName/TagName()

Promise:

```
let p = new Promise((resolve, reject) => {
      let a = 1 + 1;
      if (a == 2) { resolve('success');}
      else { reject('failed'); }
});
p.then((message) => {
      console.log('this is the then ' + message);
}).catch((message) => {
      console.log('this is the catch ' + message); });
```

Objects & Object Method:
let car = {brand: "Honda", model: "Civic Type R"}
car.brand / car['brand']

Asynchronous function:

```
async function load(){
 const response = await fetch("…");
 const content = await response.json();
 console.log(content); }
```

text() / formData(), arrayBuffer()

DOM (Document Object Model):
Objects, properties of elements
Methods, events

JSON: Object Notation, ***QUOTES***
let jtext = '{"name" : "John", "age": 20}';
let data = JSON.parse(jtext); # Decode
let jtext = JSON.stringify(data); # Encode

Invoking Function:
xxx.onlick = () => alert("Clicked")

Rest Operator: object, array

```
function(x, … more){
   console.log(x);
   console.log(more); }
```

Array methods:
Array.isArray()
Array.form("x") # Split string to array
array.slice(start, end) # end exclusive
array.splice(start, delCount, itemsAdd)
# The original array is updated / changed (M)
array.pop() # Delete last element / M
array.push(…) # Add to the end / M
array.shift() # Delete first element / M
array.unshift(…) # Add to the start / M
array.map(n => n*2) , sort(compareNumbers)
let c = […a, 0, …b] # Combine arrays
[10,20,30] = [a, …b] # b = [20,30]

Callback Function:

```
function f0(cb1, cb2){
  let x = prompt("Num");
  if (x%2) cb1();
  else cb2(); }

f0( ()=> alert("Odd"),
     ()=> alert("Even") };
```

Generator Function:

```
function* idMaker(){
  let id = 0;
  while (true){yield id++; }}

const gen = idMaker();
console.log(gen.next().value);
```

Object Methods:

```
let man = {
  word: "Hello",
  shout() {alert(this.word)}}
```

Array-like objects:
Lower memory usage

Change Items in Array: a = [1,2,3]
for (let i=0; i < a.length; i++){ a[i] = a[i] +1 } /
a.forEach((item,i,a) => a[i] +=1);

- **array.indexOf(item, start)**
- **array.lastIndexOf(item, start)**
  - Return the index if found (with === comparison) from start, or -1 if not found
- **array.includes(value)**
  - True if the array has the value
- **array.find(function(item, index, array))**
  - The way to match can be defined in the function
  - The first item returning true in function will be returned
- **array.filter(function(item,index.array))**
  - An array of matching items will be returned

```
function compareNumbers(a, b) {
  return a - b;
}
```

- **array.reverse()**
  - Reversing order of elements in array
- **array.split() / array.join()**
  - Converting a string to character array, or vice versa
- **array.map(function(item, index, array))**
  - a new array is returned with the transformation defined in function
- **sort([function(a,b)])**
  - Without the function, default sorting is comparing as string (e.g., 2>1000)
  - The function can decide how comparison should be done

```
// [20,24,26,30,40]
```

ReactJS

DOM Control (End of code)

const root = ReactDOM.createRoot(document.querySelector("#app"));
root.render(element);

CSS w/React & wo/React

```
const myStyle = {
      color: 'blue',
      fontSize: '24px',
    };
const element = <h1 style={myStyle}>Hello, I am created by JSX!</h1>;
const element = React.createElement("h1", {style: myStyle}, "Hello, I am from React with CSS.");
```

JSX produces React elements, neither HTML nor string, JSX: <script type="text/babel"> </script>
Virtual DOM: find and update the difference between original & updated version.

Class Components:

```
class App extends React.Component {
    render() {
        return (
        <div className="container">
        <Item subject="CSCI" />
        <Item subject="CENG" />
        <Item subject="AIST" />
        </div>
        );
    }
}
class Item extends React.Component {
    render() { return <div className="box">{this.props.subject}</div>; }
}
```

define props

```
class ActionLink extends React.Component {
  handleClick(e) {
    e.preventDefault();
    console.log('The link was clicked.');
  }

  render() {
    return (
      <a href="#" onClick={this.handleClick}>
        Click me
      </a>
    );
  }
}
```

Functional Components:

```
function Welcome(props) {
    return <h1>Hello, {props.name}</h1>;
}
function App() {
    return (
    <div>
    <Welcome name="Colin" />
    <Welcome name="all students" />
    </div>
    );
}
const root = ReactDOM.createRoot(document.querySelector('#app'));
root.render(<App />);
```

```
function ActionLink() {
       function handleClick(e) {
       e.preventDefault();
       console.log('The link was clicked.');
       }
    return (
    <a href="#" onClick={handleClick}>
    Click me
    </a>
    );
```

Defining States:

```
class App extends React.Component {
    constructor() {
        super();
        this.state = { s1:"CSCI", s2:"CENG", s3:"AIST" };
    }
    render() {
        return (
        <div class="container">
            <Item subject={this.state.s1} />
            <Item subject={this.state.s2} />
            <Item subject={this.state.s3} />
        </div>
        );
        }
}
class Item extends React.Component {
render() { return <div class="box">{this.props.subject}</div>; }
}
const root = ReactDOM.createRoot(document.querySelector('#app'));
root.render(<App/>);
```

Defin

Lifecycle

Mounting -> Updating -> Unmounting.

Static Routing:

Routing before rendering during initialization.

Dynamic Routing:

Routing when the app is rendering.

File-system Routing

Serve all directories, files
Permission 711: Only execute, no read, write

States are mutable.
Props are immutable.

Using setState()

```
class MyButton extends React.Component {
    constructor(props) {
        super(props);

        this.state = {value: 0}; // initialization
        this.buttonClicked = this.buttonClicked.bind(this);
    }
    buttonClicked(event) {
        this.setState({value: this.state.value+1});
    }
    render() {
        return (
            <div>
                <div>{this.state.value}</div>
                <button onClick={this.buttonClicked}>Click</button>
            </div>
        );
    }
```

Rendering Lists

```
const numbers = [1, 2, 3, 4, 5];
const listItems = numbers.map((number) =>
   <li key={number.toString()}>{number}</li>
);


ReactDOM.render(
   <ul>{listItems}</ul>,
   document.getElementById('root')
);
```

Conditional Rendering

```
render() {
  const isLoggedIn = this.state.isLoggedIn;
  return (
    <div>
    User is <b>{isLoggedIn ? 'currently' : 'not'}</b> logged in.
    </div>
  );
}
```

handleChange(event){ # Event Handling
this.setState({value: event.target.value}) } # In Class
onChange = {this.handleChange} # In render{return(..)}

React Router:

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import { BrowserRouter, Routes, Route, Link } from 'react-router-dom';

class App extends React.Component {
  render() {
    return (
      <BrowserRouter>
        <div>
          <ul>
            <li>
              <Link to="/">Home</Link>
            </li>
            <li>
              <Link to="/about">About</Link>
            </li>
          </ul>
          <hr />

          <Routes>
            <Route path="/" element={<Home />} />
            <Route path="/about" element={<About />} />
          </Routes>
        </div>
      </BrowserRouter>
    );
  }
}
```

```
class Home extends React.Component {
  render() {
    return <h2>Home</h2>;
  }
}

class About extends React.Component {
  render() {
    return <h2>About</h2>;
  }
}
```

Single Page Application (SPA):

Pagination (page number) / Infinite scrolling / better UX / Easier maintenance / Efficient use of bandwidth / no footer / cannot bookmark browser history / better for mobile devices / Not friendly for search engine optimization

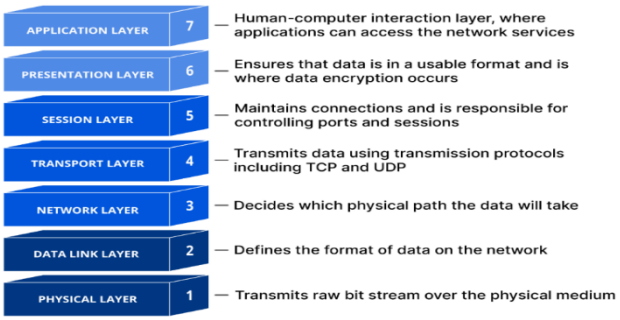
Page-based navigation: Reload for every page

HTTP

Socket = MAC + IP + TCP

Client-Server: OSI model
1 on 1 interaction, 1 Request -> 1 Response
Open Systems Interconnection
Communications are divided into multiple layers.
Reduce complexity.

MAC Address (2):
Local Network
IP Address (3):
Network Interface
Transport (4): Reliability
TCP (reliable) vs
UDP (fast, streaming)



NodeJS

Express Basics:
Routing, Retrieving data from GET & POST, Generate content of response, Retrieving and setting header of request, cookie and sessions

HTTP Response Body: Static or dynamically generated file, data is encoded / Set-Cookie / Content-Type: text/(plain/html/csv)

Process of Web Server:
1. Routing (Deciding GET/PUT method)
2. Retrieve data from HTTP request.
3. Process data (Validation)
4. Generate an HTTP response.

Route Request:

```
const express = require('express');
const app = express();

// To handle a GET request for /path1
app.get('/path1', (req, res) => res.send("You made a GET request"));

// To handle a POST request for /path2
app.post('/path2', (req, res) => res.send("You made a POST request"));

// To handle all requests (regardless of request method)
app.all('/*', (req, res) => res.send("You made a request"));

// The order in which routes are set up is important!
app.get('/path3', (req, res) => res.send("You will not see this"));
// In this example, a GET request for /path3 will be intercepted by app.all('/*', ...)
```

res.send() can only run once (request, response is 1on1)
When it's called, the following code will not run

```
// Regular expression matching: e.g., any path that ends with .jpg
// Note: The expression is not enclosed by any quotes
app.all(/.*\.jpg$/, (req, res) => res.send("You requested a JPG file"));

// Route parameters matching
// e.g., http://hostname/course/2720/lecture/6
app.all('/course/:cID/lecture/:lID', (req, res) => res.send(req.params));
  // Output: {"cID":"2720", lID":"6"}

// hyphen and dot (- and .) are interpreted literally
// e.g., http://hostname/csci2720-t2
app.all('/:course-:tutorial', (req, res) => res.send(req.params));
   // Output: {"course":"csci2720", "tutorial":"t2"}
```

Serve Static Files:
app.get('/content.html', (req,res) => {
res.sendFile(_dirname + '/index.html') });

Generate File Dynamically:
app.get('/content.html', (req,res) => {
var buf = '…' ;
res.send(buf); });

Express Routing:
app.METHOD(path, callback);
GET, POST, PUT, DELETE, ALL

Syntax for getting POST:
const parser = require('body-parser')
app.use(parser.urlencoded({extened: false}))

HTTP Request Header:
User-agent, Referrer, Cookie, Content-Type (html, img)
No body info but still header when there is error (eg 404)

Serve folder & img:
app.use(express.static('public'));
app.use('/img', express.static('images'))

Retrieve Request Header:
req.header / req.get("user-agent")

Extract ?key-value:

```
app.post('/login', (req, res) => {
 // Parameters are made available as properties of req.body
 let id = req.body['loginid'], pwd = req.body['passwd'];
 res.send('Your login is ' + id + ' and password is ' + pwd)
});
```

req.query['key'] / req.params[] # path

Middleware & routing:
function (req, res, next) {…}

Query operators
- **$eq** equal
- **$gt** greater than
- **$gte** greater than or equal to
- **$in** in an arrray
- **$lt** less than
- **$lte** less than or equal to
- **$ne** not equal
- **$nin** not in an array

Cookies
- Can be changed / deleted.
- HTTP is stateless
  --> Can't identify request, dk who user is.
- Stores in local browser.
- Both Client, Server have control on Cookies.
- Client request for Cookies first.
- Limited number of cookies per server.
- Limited data size per cookie.
- Not safe for sensitive data.
- Set-Cookie:
  expires = date, Max-Age: (precede),
  domain: "…" , path="…"

Session
- Keep temporary data at server side

Local Storage
- Store arbitrary data at client side

Security
Validation -> Check string format
Escaping -> Use special char
Sanitization -> Create a whitelist
Encryption: Transfer msg in ciphertext
Authentication: Digital certificate
Domain, organization, extended verification