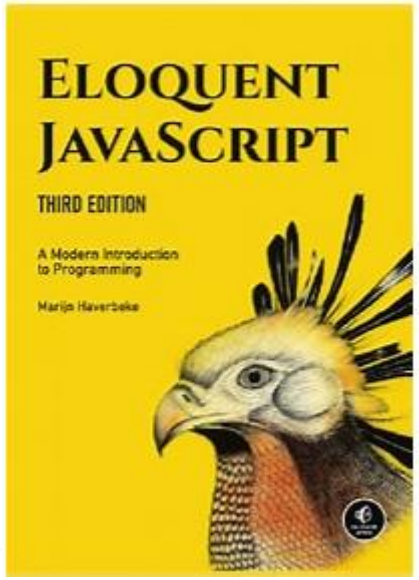
Topic 8 Full Stack
- Program a browser

Full stack

- A full stack web developer is a person who can develop both **client** and **server** software.
- In addition to mastering HTML and CSS, he/she also knows how to:
  - Program a **browser** (like using JavaScript, jQuery, Angular, React or Vue)
  - Program a **server** (like using PHP, ASP, Python, or Node)
  - Program a **database** (like using SQL, SQLite, or MongoDB)



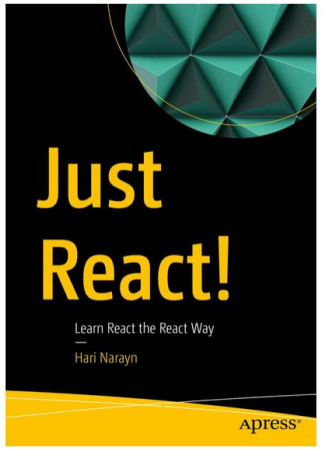
BOOK

[Eloquent JavaScript, 3rd Edition (oreilly.com)]

Just React!: Learn React the React Way

★★★★★ 1 review

By Hari Narayn



TIME TO COMPLETE:
6h 58m

TOPICS:
React

PUBLISHED BY:
Apress

PUBLICATION DATE:
August 2022

PRINT LENGTH:
377 pages

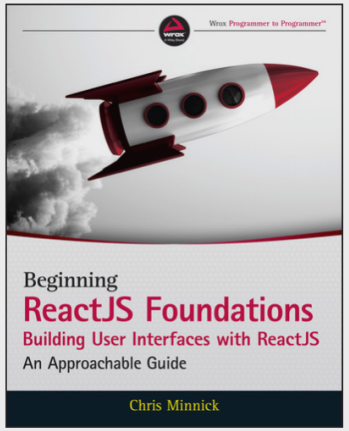
Start

[Just React!: Learn React the React Way (oreilly.com)]

Beginning ReactJS Foundations Building User Interfaces with ReactJS

★★★★★ 2 reviews

By Chris Minnick



TIME TO COMPLETE:
11h 20m

TOPICS:
React

PUBLISHED BY:
Wiley

PUBLICATION DATE:
March 2022

PRINT LENGTH:
512 pages

Start

Quickly learn the most widely used front-end development language with ease and confidence

React JS Foundations: Building User Interfaces with ReactJS - An Approachable Guide walks readers through the fundamental concepts of programming with the explosively popular front-end tool known as React JS.

Written by an accomplished full-stack engineer, speaker, and community organizer, *React JS Foundations* teaches readers how to understand React and how to begin building applications with it. The book:

[Beginning ReactJS Foundations Building User Interfaces with ReactJS (oreilly.com)]

HTML

To structure web pages

CSS

To make them beautiful

JavaScript

To program them

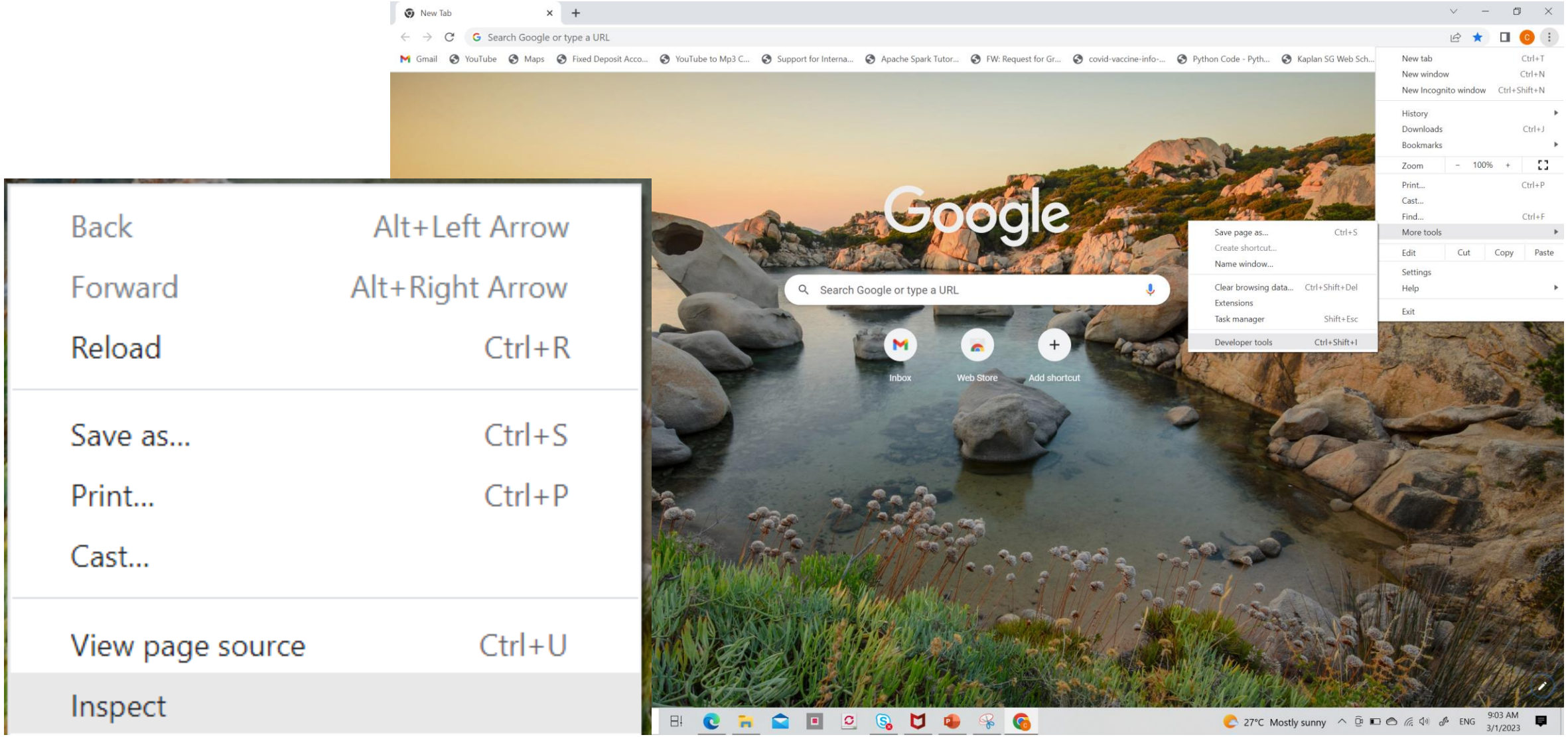
[5 Front-end Development Skills to Land Your First Job - YouTube]

JavaScript

- JavaScript is now everywhere. We can now use JavaScript to make Web applications, Server-side applications, Cross-platform Mobile and Desktop applications, Machine Learning,…..

- JavaScript is different from Java

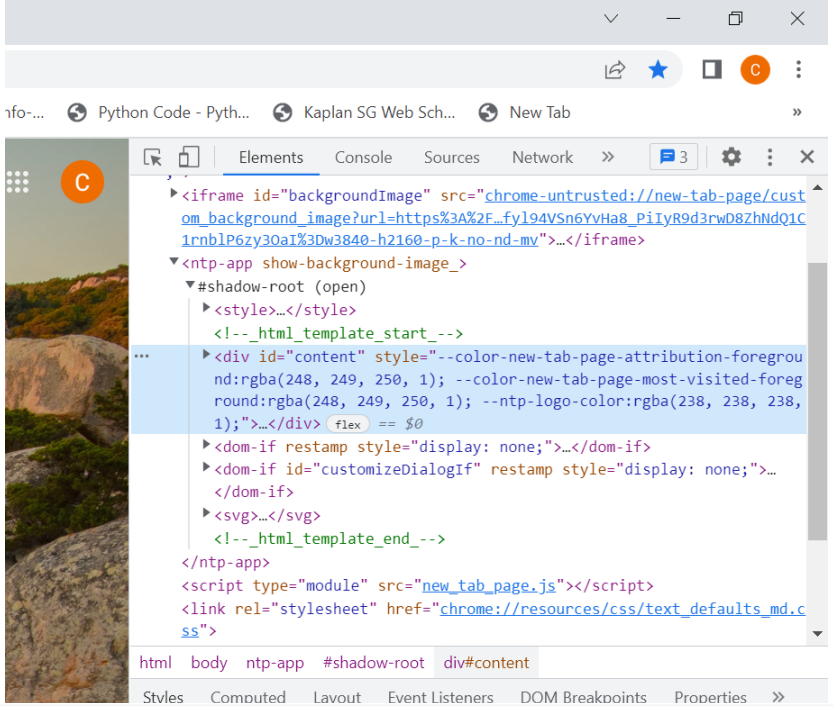
Running JavaScript in the browser

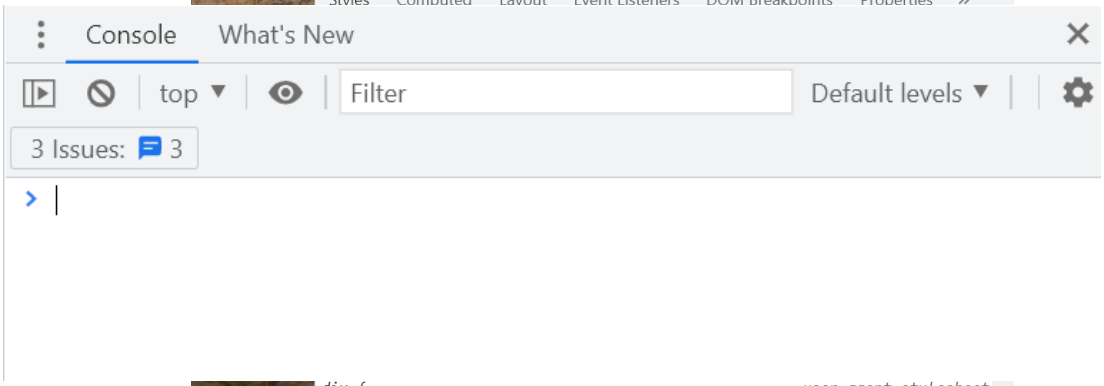


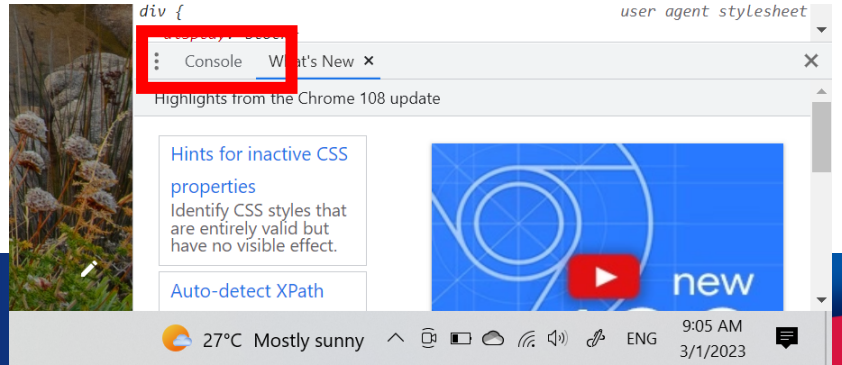
Browser console

- This console is called a **REPL** or **Read Evaluate Print Loop.** Here we can write our JavaScript code, which then will be interpreted by the browser's JavaScript Engine, and the browser will show the output.
- Let's write a code in the console and press Enter:

```
console.log("MsBA");
```







The console.log Function

- Most JavaScript systems (including all modern web browsers and Node.js) provide a console.log function that writes out its arguments to some text output device.

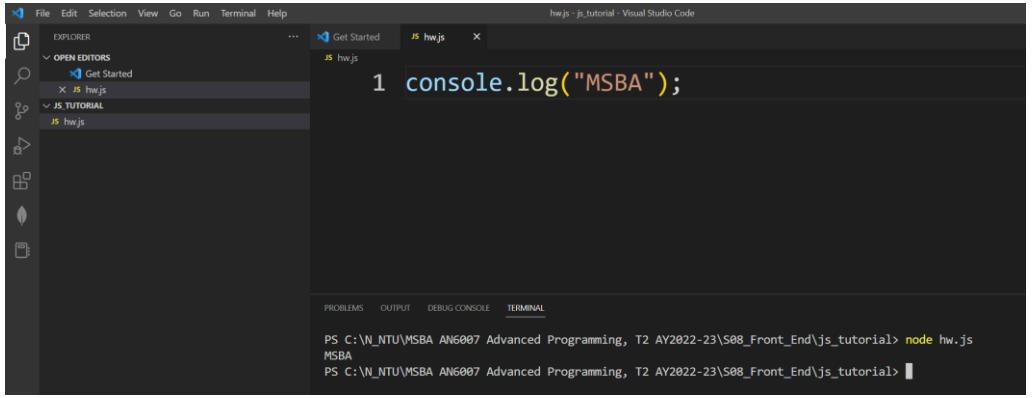
- In browsers, the output lands in the JavaScript console. This part of the browser interface is hidden by default, but most browsers open it when you press F12 or, on a Mac, COMMAND-OPTION-I.

- If that does not work, search through the menus for an item named Developer Tools or similar.

Running JavaScript files in the browser

- JavaScript can run directly from the browser console, it’s only suitable for testing small pieces of code.
- To implement some big and complex code is to write it in a JavaScript file.
- Let’s create a JavaScript file hw.js and open it in VSCode

```
console.log(“MsBA");
```



- run `node hw` after you open a new terminal in VSCode

Declaring Variables in JavaScript - Binding

```
let caught = 5 * 5;
```

- You should imagine bindings as tentacles, rather than boxes. They do not *contain* values; they *grasp* them—two bindings can refer to the same value. A program can access only the values that it still has a reference to. When you need to remember something, you grow a tentacle to hold on to it or you reattach one of your existing tentacles to it.

- When you define a binding without giving it a value, the tentacle has nothing to grasp, so it ends in thin air. If you ask for the value of an empty binding, you'll get the value undefined.

- The words var and const can also be used to create bindings, in a way similar to let.

- Note that var (short for "variable"), is the way bindings were declared in pre-2015 JavaScript.

To enforce that variables must be declared, start your program with

```
'use strict';
```

Variables

```
let a;
console.log(a);

a = 2;
console.log(a);

a = "hello";
console.log(a);

const MAX = 2048;
console.log(MAX);

let count = 0;
console.log(count);

let fname, lname;
console.log(fname);
console.log(lname);
```

[ES6 Tutorial: Learn Modern JavaScript in 1 Hour - YouTube]

[Learn JAVASCRIPT in just 5 MINUTES (2020) - Bing video]

Variables and Arithmetic

- // put this in `prog.js`
  // run `node prog`
  ```
  let a = 4;
  let b = 3;
  console.log(a + b);
  ```

Data types: Strings

```
let uni = "NTU";
let course = "MSBA";
console.log(uni + course);
console.log(uni.indexOf("N"));
console.log(uni.indexOf("Z"));
console.log(uni.includes("N"));
console.log(uni.includes("Z"));
console.log(`I am studing in ${uni} taking ${course}`);
```

strWork.js

Comparisons

false, o,empty string, null, and undefined are false; everything else is true

example values: `true  false`

operators: `&&  ||  !  !=  >  <  ==  ===  !==`

guess the results:

equality	`42=="42"`	`42==="42"`	`0===-0`	`NaN===NaN`
inequality	`"a" > "b"`	`"02" > "1"`		
truthy v falsy	`0  1  -42.3`			
	`""  "a"  "0"`			

Empty values **null** and **undefined**

guess the results:

```
1 console.log(null == undefined);
2 console.log(null === undefined);
3 console.log(!null);
4 console.log(!undefined);
5 console.log(!!null);
6 console.log(!!undefined);
```

null <= no value

undefine <= does not exist

Data Type : Numbers

example values: `0, 1, -1, 42.0, 42.3, 3e4, -3.4e-4`

operators: `+ - * / %`

math library: `Math.round()  Math.floor()  Math.random()` *etc.*

careful: `0.1 + 0.2 != 0.3`

`Infinity  -Infinity  +0  -0  NaN`

`9007199254740991  -9007199254740991` $= 2^{53} - 1$

`0x2a  0o52  0b101010`

bitwise `&  |  ^  <<  >>  >>>` 32-bit-only

Structured values: objects

```
let a = {};

a.uni = "NTU";
a.years = 2;

console.log("University: " + a.uni);
console.log("Years: " + a.years);

console.log(a.course); // outputs
"undefined"
```

```
let a = {};
a.uni = “NTU";
a.year = 2;
```

```
let a = {
  uni: “NTU",
  year: 2,
};
```

```
let uni = “NTU";
let year = 2;

let a = { uni, year };
```

```
let a = {};
a.year = 2;
a[“uni"] = “NTU";
```

Workshop – What is the output?

```
1 let b = {};
2 let puzzle = "msg";
3 b[puzzle] = "where does this go?";
4 b.puzzle = "is this in the same place?";
5 console.log(b);
```

```
PS C:\N_NTU\MSBA AN6007 Advanced Programming, T2 AY2022-23\S08_Front_End\js_tutorial> node workshop1.js
{ msg: 'where does this go?', puzzle: 'is this in the same place?' }
```

JavaScript Object Notation (JSON)

JSON.stringify(x) **JSON.parse(string)**

```
{
  "name": {
    "first": "Jack",
    "last":  "Kopecky"
  },
  "age": 18,
  "units": [ "WebF1", "WebScript", "WebRes", "DBPRIN" ]
}
```

JavaScript – If

```
const date = new Date();

let day = date.getDate();
let month = date.getMonth() + 1;
let year = date.getFullYear();

// This arrangement can be altered based on how we want the date's format to appear.
let currentDate = `${day}-${month}-${year}`;
console.log(currentDate); // "17-6-2022"

if (month == 12){
    console.log("Merry Christmas")
}
else if (month ==2){
    console.log("Happy Lunnar New Year")}
else{
    console.log(`Happy ${year}`)
}
```

JavaScript - looping

```
let number = 0;
while (number <= 12) {
  console.log(number);
  number = number + 2;
}
```

Initial condition

exit condition

Preparation for next iteration

```
let result = 1;
for (let counter = 0; counter < 10; counter = counter + 1) {
  result = result * 2;
}
```

JavaScript - Break

```
for (let current = 20; ; current = current + 1) {
    if (current % 7 == 0) {
        console.log(current);
        break;
    }
    else{
        console.log(`testing ${current} not divided by 7`)
    }
    }
```

```
for (let current = 20; ; current++) {
    if (current % 7 == 0) {
        console.log(current);
        break;
    }
    else{
        console.log(`testing ${current} not divided by 7`)
    }
    }
```

Block Scope

```
let a = 199;

for (let current = 20, a =0; ; current++) {
    if (current % 7 == 0) {
        console.log(current);
        break;
    }
    else{
        console.log(`testing ${current} not divided by 7`)
        a++;
        console.log(`found ${a}`)
    }
}

console.log(`a is ${a}`)
```

```
PS C:\N_NTU\MSBA AN6007 Advanced Programming, T2 AY2022-23\S08_Front_End\js_tutorial> node codeBlock.js
testing 20 not divided by 7
found 1
21
a is 199
```

Anatomy of a simple function

reserved word

function name

parameter / argument

Same comment syntax as Java.
/* */ for block comments
// for lines

```
/* Prints a greeting */
function greet( name ) {
  console.log(`Hello ${name}`);
}
```

Opening block delimiter

Closing block delimiter

Statement(s) executed each time the function is called.

Default parameter values

without:

```
function f (x, y, z) {
    if (y === undefined)
        y = 7;
    if (z === undefined)
        z = 42;
    return x + y + z;
};
f(1) === 50;
```

with:

```
function f (x, y = 7, z = 42) {
    return x + y + z;
}
f(1) === 50;
```

Side Effects

```
let latestGreeting = "";

function greet( name ) {
  console.log( `Hello ${name}` );
  latestGreeting = name;
}

greet( "Skywalker" );
greet( "Vader" );
console.log( latestGreeting );
```

Modules - split code into files

allinone.js

```
function add(a,b) {
  return a+b;
}

console.log(add(3,4));
```

func.js

```
function add(a,b) {
  return a+b;
}
module.exports.add = add;
```

main.js

```
const func = require('./func');

console.log(func.add(3,4));
```

As program grows larger, we will put codes in multiple files

Modules - split code into files and folders

func.js

```
function add(a,b) {
  return a+b;
}
module.exports.add = add;
```

func2/index.js

```
function add(a,b) {
  return a+b;
}
module.exports.add = add;
```

main.js

```
const func = require('./func');

console.log(func.add(3,4));
```

main.js

```
const func = require('./func2');

console.log(func.add(3,4));
```

JavaScript - Recursion

```
function power(base, exponent) {
  if (exponent == 0) {
    return 1;
  } else {
    return base * power(base, exponent - 1);
  }
}
```

- in typical JavaScript implementations, it’s about three times slower than the looping version. Running through a simple loop is generally cheaper than calling a function multiple times.
- The dilemma of speed versus elegance is an interesting one. You can see it as a kind of continuum between human-friendliness and machine-friendliness. Almost any program can be made faster by making it bigger and more convoluted. The programmer has to decide on an appropriate balance.
- In the case of the power function, the inelegant (looping) version is still fairly simple and easy to read. It doesn’t make much sense to replace it with the recursive version. Often, though, a program deals with such complex concepts that giving up some efficiency in order to make the program more straightforward is helpful.
- Worrying about efficiency can be a distraction. It’s yet another factor that complicates program design, and when you’re doing something that’s already difficult, that extra thing to worry about can be paralyzing.
- Therefore, always start by writing something that’s correct and easy to understand. If you’re worried that it’s too slow—which it usually isn’t since most code simply isn’t executed often enough to take any significant amount of time—you can measure afterward and improve it if necessary.
- Recursion is not always just an inefficient alternative to looping. Some problems really are easier to solve with recursion than with loops. Most often these are problems that require exploring or processing several “branches,” each of which might branch out again into even more branches.

Call backs

- JavaScript relies heavily on callback functions.
- Instead of a function giving us a result immediately, we give it another function that tells it what to do next.

```
power = (base, exponent) => {
  let result = 1;
  for (let count = 0; count < exponent; count++) {
    result *= base;
  }
  return result;
};

const anotherName = power
console.log(anotherName(5,3))
```

Anonymous function – a function that is defined where it is used and isn't assigned a name

Anonymous function

```
let a = 3;
console.log( a );

a = function (x,y) { return x+y; };
a = (x,y) => { return x+y; };          // shorthand for the above
a = (x,y) => x+y;                      // shorthand for the shorthand

console.log( a );
console.log( a( 1,2 ) );
```

JavaScript – Arrow Function

- Instead of the function keyword, it uses an arrow (=>) made up of an equal sign and a greater-than character (not to be confused with the greater-than-or-equal operator, which is written >=).
- The arrow comes *after* the list of parameters and is followed by the function's body. It expresses something like "this input (the parameters) produces this result (the body)."
- When there is only one parameter name, you can omit the parentheses around the parameter list. If the body is a single expression, rather than a block in braces, that expression will be returned from the function. So, these two definitions of square do the same thing:

```
const power = (base, exponent) => {
  let result = 1;
  for (let count = 0; count < exponent; count++) {
    result *= base;
  }
  return result;
};
```

```
const square1 = (x) => { return x * x; };
const square2 = x => x * x;
```

Functions of functions

```
const doTwice =(action) +>{
     action()
     action()
}

const hello() =() => {
   console.log(‘hello’)
}

doTwice(hello)
```

Structured values: arrays

```
const DAYS = ["Mon", "Tue", "Wed", "Thu", "Fri", "Sat", "Sun"];
console.log(DAYS);
console.log(DAYS[0]);
DAYS[1]=2;
console.log(DAYS);
console.log(DAYS[1]);
console.log(DAYS.length)
```

Structured values: arrays

```
let a = [];

a[0] = “NTU";
a[1] = 2;


let b = [ “NTU", 2, ];

let c = [
 “NTU",
 2,
];
```

Array API (beware: in-place vs new array)

```
a.sort()
a.pop()
a.shift()
```

```
a.reverse()
a.push('foo')
a.unshift('foo')
```

```
a.concat([1, 2, 3])
```

```
let a = [ "barack", "hillary", "donald" ];

a.indexOf("hillary")
```

Array functions – showcase of Functional programming in javascript

```
a.find(fn)        a.some(fn)        a.every(fn)
a.forEach(fn)     a.map(fn)         a.reduce(fn)
```

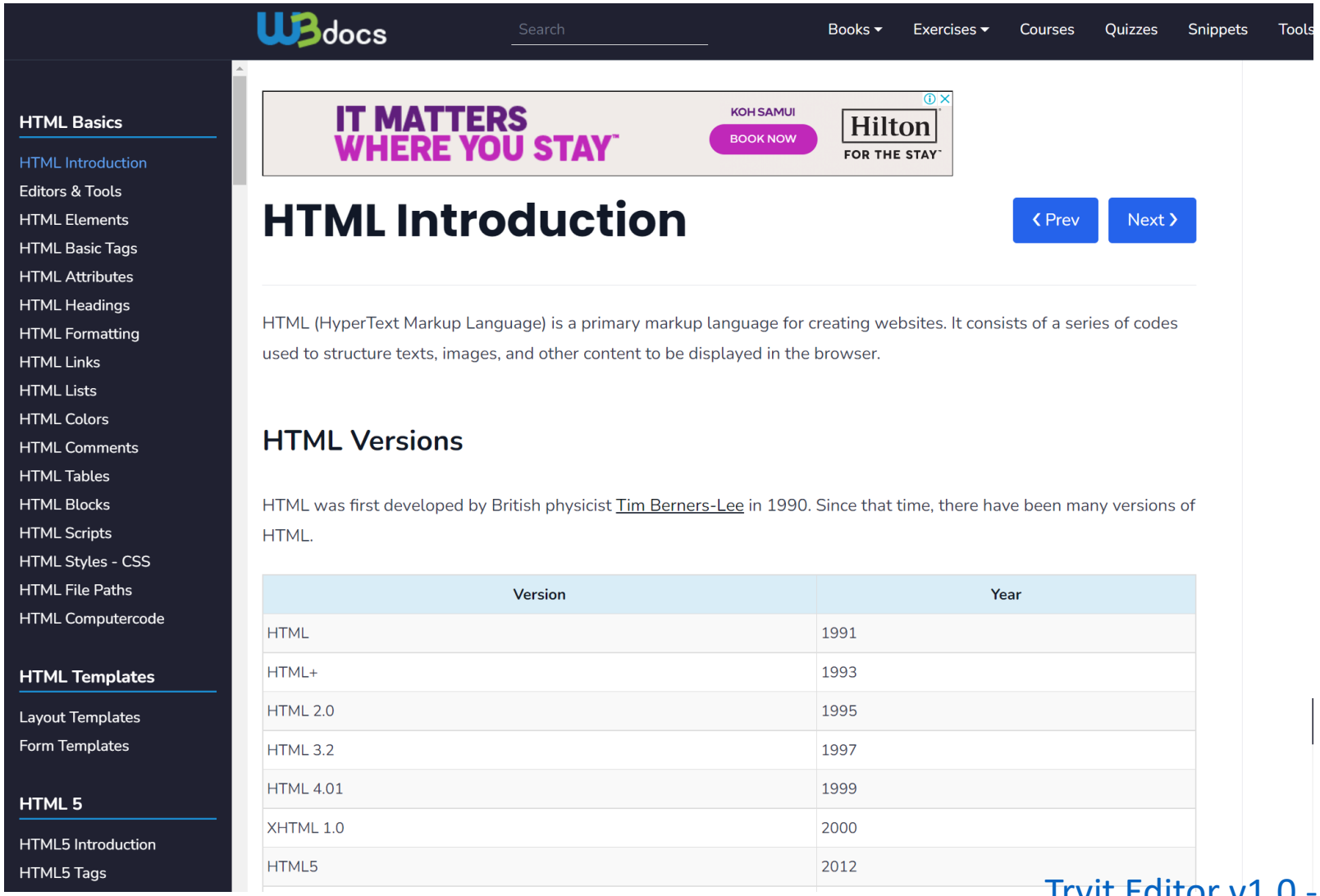
```javascript
const a = [ "barack", "hillary", "donald" ];

a.find( (item) => item.includes('a') );  // try some, every

a.forEach( (item) => { console.log(item); } );

const b = a.map ( (item) => item.toUpperCase() );
```

Loops: For…of

for….of loop allows iteration over values.

```
const quart = [‘Jan', ‘Feb', ‘Mar'];
for (const mth of quart) {
     console.log(mth);
      // outputs Jan then Feb then Mar
}
```

Recommending HTML Introduction | W3Docs Tutorial



Tryit Editor v1.0 - HTML Editors & Tools (w3docs.com)

HyperText Markup Language - HTML

HTML is a language used to define the structure of content.

It comprises a set of codes in a text file called tags.

Each tag is enclosed with <>

Each tag will have a start and an end.

```
<!DOCTYPE html>
<html>
<head>
    <title>My MSBA first html
page</title>
</head>
<body>
    <div id="app">
    <p id="p1">NTU</p>
    <p id="p2">2023</p>
    <button>Change Content</button>
    </div>
</body>
</html>
```

h1.html

WSU-HTML-Cheat-Sheet.pdf

HTML Escapes

Name	Escape Sequence	Character
Less than	<	<
More than	>	>
Ampersand	&	&
Copyright	©	©
Plus/Minus	&plusminus;	±
Mirco	µ	µ

HTML list

- To create an unordered (bulleted) list, we use a ul element, and wrap each item inside the list in li. To create an ordered (numbered) list, we use ol instead of ul, but still use li for the list items.

```
<ul>
  <li>first</li>
  <li>second</li>
  <li>third</li>
</ul>
```

```
<ol>
  <li>first</li>
  <li>second</li>
  <li>third</li>
</ol>
```

- Lists can be nested by putting the inner list's ul or ol inside one of the outer list's li elements

HTML tables

- Each row is a tr (for “table row”), and within rows, column items are shown with td (for “table data”) or th (for “table heading”).

```
<table>
  <tr> <th>Alkali</th>   <th>Noble Gas</th> </tr>
  <tr> <td>Hydrogen</td> <td>Helium</td>    </tr>
  <tr> <td>Lithium</td>  <td>Neon</td>      </tr>
  <tr> <td>Sodium</td>   <td>Argon</td>     </tr>
</table>
```

Alkali	Noble Gas
Hydrogen	Helium
Lithium	Neon
Sodium	Argon

HTML Links

```html
<a href="https://nodejs.org/">Node.js</a>
<br/>
<a href="https://facebook.github.io/react/">React</a>
<br/>
<a href="../index.html">home page (relative path)</a>
```

```html
<img src="./assets/logo.png" title="Book Logo"
     alt="Displays the book logo using a local path" />
<img src="https://js4ds.org/assets/logo.png"
     title="Book Logo"
     alt="Display the book logo using a URL" />
```

- Relative path is usually used, it's interpreted starting from where the web page is located;
- if it's an absolute path, it's interpreted relative to wherever the web browser thinks the root directory of the filesystem is.

Cascading Style Sheets -CSS

- It is a style sheet language which is used to describe the look and formatting of a document written in markup language.
- It provides an additional feature to HTML. It is generally used with HTML to change the style of web pages and user interfaces.
- It can also be used with any kind of XML documents including plain XML, SVG and XUL.

```
<!DOCTYPE html>
<html>
<head>
    <title>My MSBA first html page</title>
</head>
<body>
    <div id="app">
    <p style="color:red;" id="p1">NTU</p>
    <p style="color:green;" id="p2">2023</p>
    <button style="color:blue;">Change
Content</button>
    </div>
</body>
</html>
```

[HTML CSS Tutorial for Beginners | Learn HTML & CSS | Full Stack Training | Edureka - Bing video]
[CSS Tricks: Five Tricks to Enhance Your Web Page (simplilearn.com)]

h2.html

wsu-css-cheat-sheet-gdocs.pdf

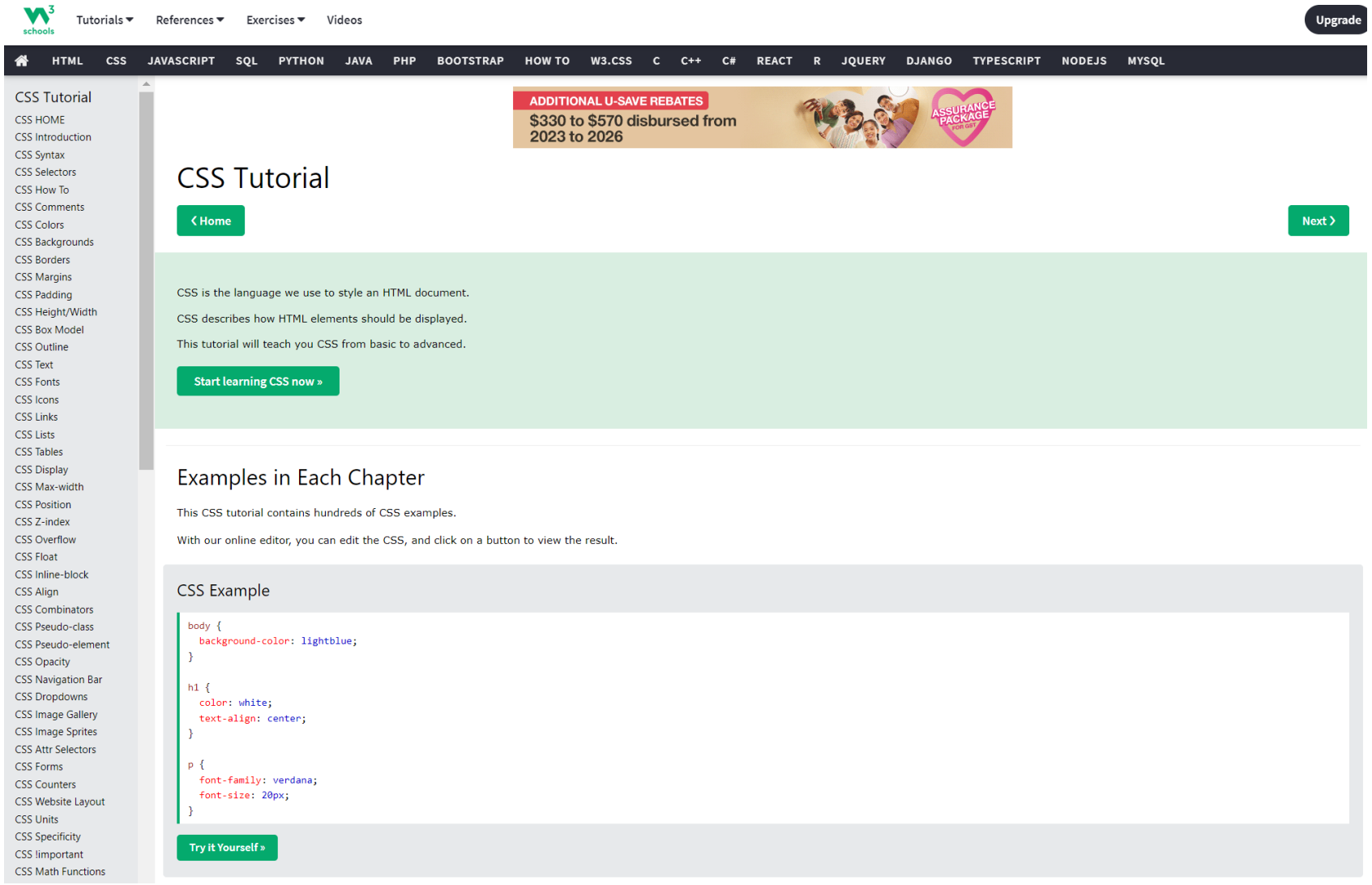
```
<h2 style="color:red;"> Web Development </h2>
```

Internal -> External style sheet

```
<html>
<head>
    <title> Home - dureka </title>
    <link rel="stylesheet" type="text/css" href="css/
    style.css"
</head>
<body>
```

Watch Later

Recommending CSS Tutorial (w3schools.com)

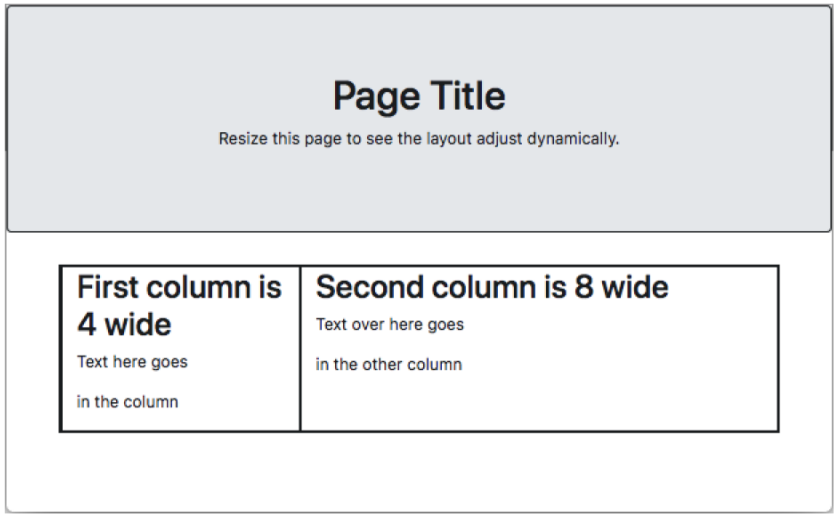


W3Schools Tryit Editor

BOOTSTRAP

- CSS can become very complicated very quickly, so most people use a framework to take care of the details.
- One of the most popular is Bootstrap

```
<html>
  <head>
    <link rel="stylesheet"
          href="https://stackpath.bootstrapcdn.com/bootstrap/
                4.1.3/css/bootstrap.min.css">
    <style>
      div {
        border: solid 1px;
      }
    </style>
  </head>
  <body>

    <div class="jumbotron text-center">
      <h1>Page Title</h1>
      <p>Resize this page to see the layout adjust dynamically.</p>
    </div>

    <div class="container">
      <div class="row">
        <div class="col-sm-4">
          <h2>First column is 4 wide</h2>
          <p>Text here goes</p>
          <p>in the column</p>
        </div>
        <div class="col-sm-8">
          <h2>Second column is 8 wide</h2>
          <p>Text over here goes</p>
          <p>in the other column</p>
        </div>
      </div>
    </div>

  </body>
</html>
```



Multi-page websites vs Single Page Applications

- Multi-page websites, defined by traditional navigation flows, are well-known and trusted by users.

Image credit: Awwwards

- Simple, speedy and responsive single-page websites are preferred for mobile and social media

[Single page vs multi-page design: which is better? - Justinmind]

Benefits Of Multi-Page Website

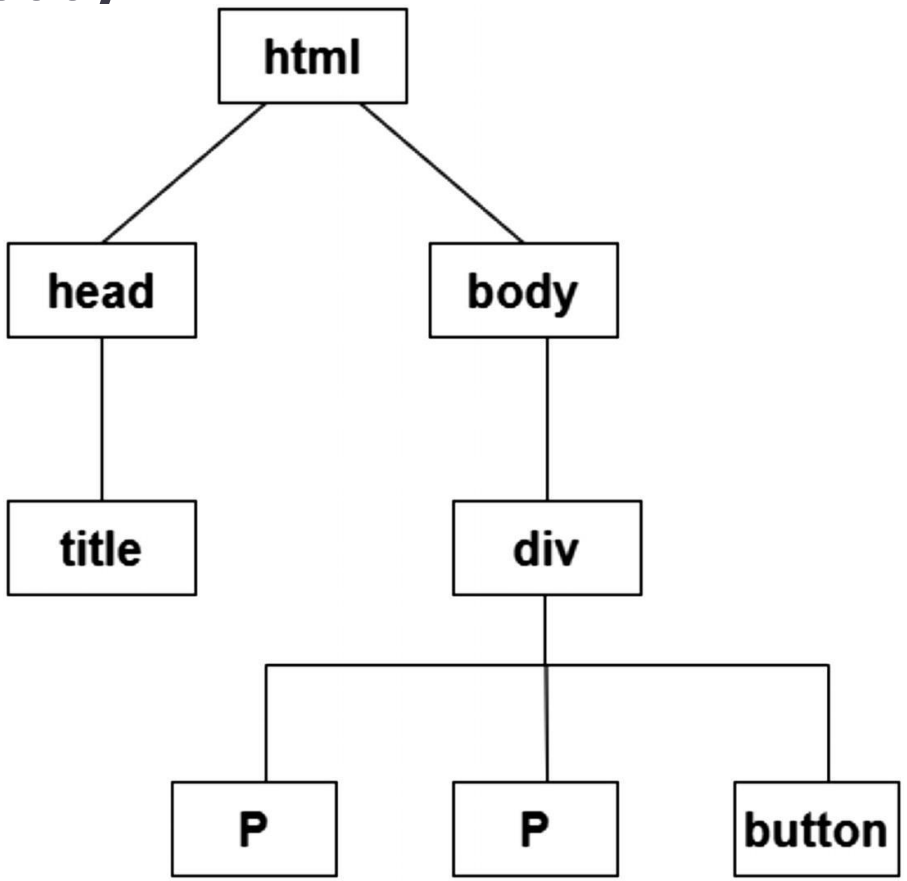
- A multi-page website segregates content on the website better. Rather than adding all the content for the website on a single page using headings, a multi-page website is able to better manage and showcase content on different pages based on the context.
- A multi-page website offers better user flow and user experience. Users use the top navigation to navigate between web pages of a website. The navigation menu makes it clear that such and such content is at such and such page.
- A multi-page website is search engine friendly. When you have a lot of content that differs in context between them, it's a great idea to have a multi-page website. Not only users would be able to better understand the contents of the website, but also search engine bots that crawl your website are able to better segregate the contents based on the web pages

Single Page Applications

- A Single Page Application (SPA) is a single web page, website, or web application that
  - works within a web browser
  - loads just a single document.
  - does not need page reloading during its usage, and most of its content remains the same while only some of it needs updating.
  - When the content needs to be updated, the SPA does it through JavaScript APIs.
- This way, users can view a website without loading the entire new page and data from the server.
  - performance increases, feel like using a native application.
  - offers a more dynamic web experience to the users.
- SPAs help users be in a single, uncomplicated web space in easy, workable, and simple ways

Document Object Model (DOM)

- Every browser has an engine that constructs a Document Object Model (DOM) tree from the HTML content it reads. The DOM views an HTML document as a tree of nodes.
- The browser determines what styles need to apply to elements and creates another tree, called the CSS Object Model (CSSOM) tree. Then it combines the DOM and CSSOM trees to produce the render tree. The difference between a DOM tree and a render tree is that the render tree knows about the styles.
- In contrast to the DOM tree, the render tree has information about the styles. Afterward, the browser computes the width, height, location, size, and position of each node in the render tree. Finally, it paints the elements on the screen. This allows us to view the content on the screen.

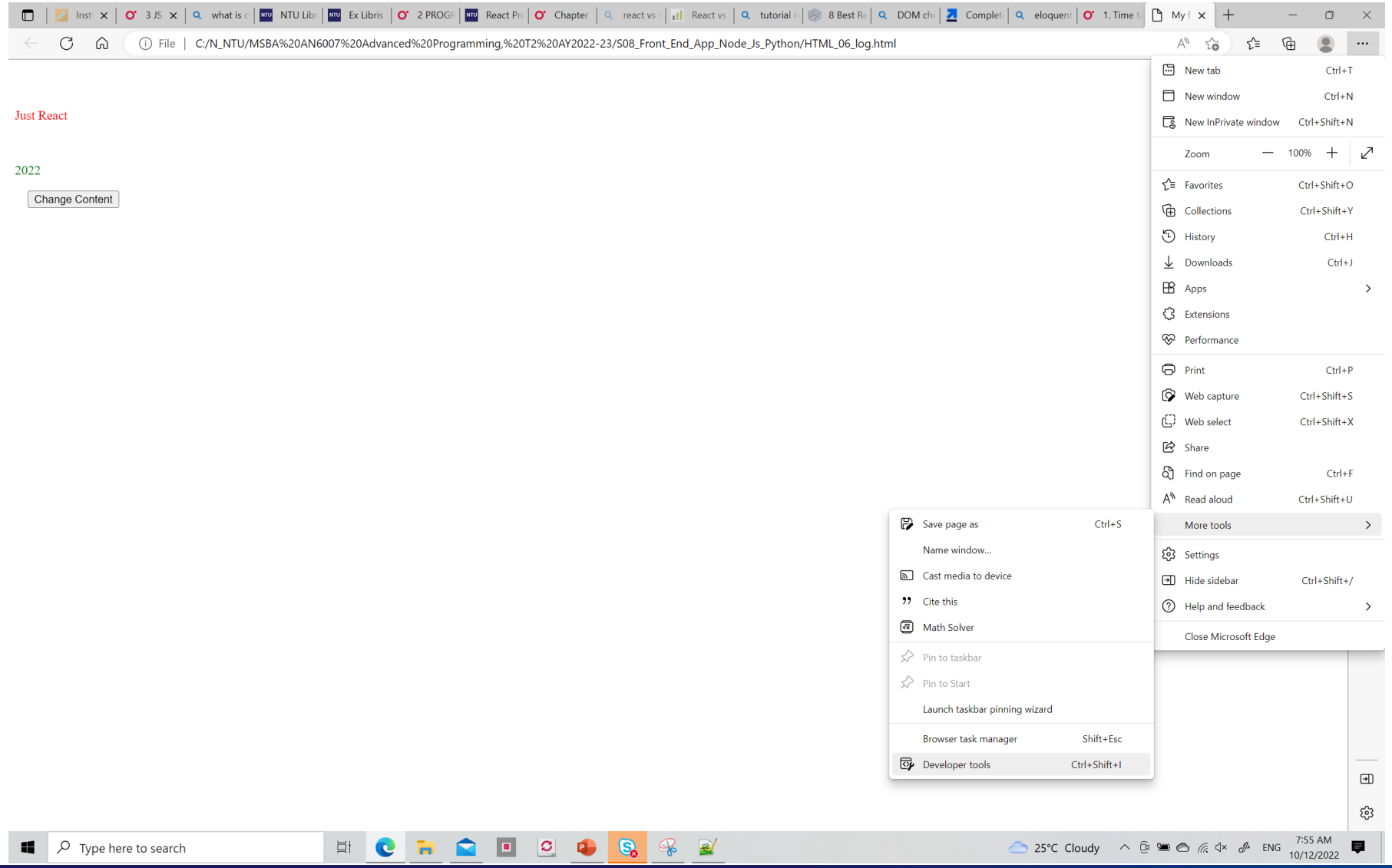


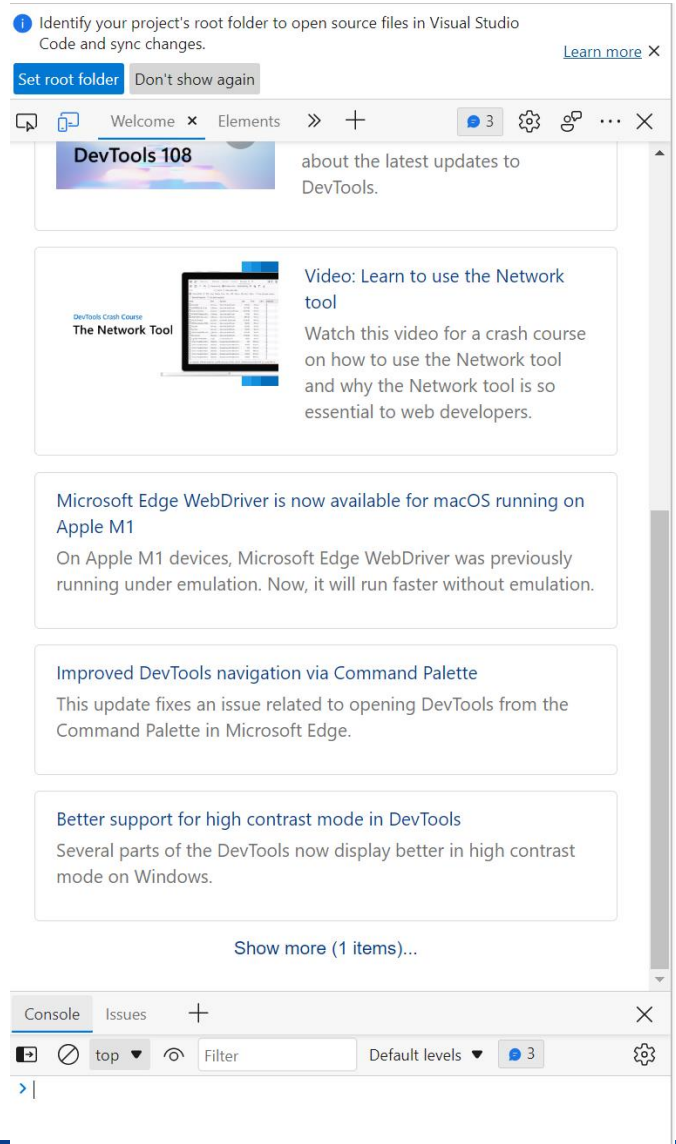
DOM tree

Built-in Events

- **load** (window, body, img)
- **click** (most)
- **keydown, keypress, keyup** (body & form elements)
- **change** (input, select, textarea)
- **submit, reset** (form)
- **focus, blur** (button, input, label, select, textarea, body)
- **mousedown, mousemove, mouseleave, mouseenter, mouseup**
- **unload** (body, frameset)
- **resize** (body, frameset)
- **abort**, error (img)
- **select** (input, textarea)
- **dblclick** (most)

HTML – debugging





File | C:/N_NTU/MSBA%20AN6007%20Advanced%20Programming,%20T2%20AY2022-23/S08_Front_End_App_Node_Js_Python/HTML_06_log.html

Dimensions: Responsive 1088 x 863 100% No throttling

This page says

On which year your graduated from high school?

OK Cancel

Just React

2022

Change Content

Koh

2022

Change Content

Identify your project's root folder to open source files in Visual Studio Code and sync changes.

Learn more

Set root folder Don't show again

Welcome Elements

DevTools 108

about the latest updates to DevTools.

Video: Learn to use the Network tool

Watch this video for a crash course on how to use the Network tool and why the Network tool is so essential to web developers.

Microsoft Edge WebDriver is now available for macOS running on Apple M1

On Apple M1 devices, Microsoft Edge WebDriver was previously running under emulation. Now, it will run faster without emulation.

Improved DevTools navigation via Command Palette

This update fixes an issue related to opening DevTools from the Command Palette in Microsoft Edge.

Better support for high contrast mode in DevTools

Several parts of the DevTools now display better in high contrast mode on Windows.

Show more (1 items)...

Console Issues

top Filter Default levels 3 1 hidden

```
just updated name to Koh                HTML_06_log.html:16
just updated year to 2022               HTML_06_log.html:19
>
```

Type here to search

25°C Cloudy ENG 7:59 AM 10/12/2022

NANYANG TEC

Event-driven Programming

- `addEventListener()` is a method to add an event to an element
- `removeEventListener()` is a method to remove the event

- You can add or remove the click event from the button with id `btnCase` using the following code:
  - `document.getElementById("btnCase").addEventListener("click", swapCase);`
  - `document.getElementById("btnCase").removeEventListener("click", swapCase);`

Multiple event listeners

```
<p id=example>Hello Event World</p>

<script>
  function shout(e) {
        console.log(e.target.textContent);
  }

  function shoutOnce(e) {
        console.log(e.target.textContent, "only once");
        e.target.removeEventListener("click", shoutOnce);
  }

  window.example.addEventListener("click", shout);
  window.example.addEventListener("click", shoutOnce);
</script>
```

Using other types of event: mouseenter

Recap

```
<button id=example>Chase Me!</button>

<script>
  function react(e) {
    e.target.style.transform =
      `translate(${Math.random()*90}vw, ${Math.random()*90}vh)`;
  }

  window.example.addEventListener("mouseenter", react);
</script>
```

UI vs. UX
User Interface vs User Experience

- To the user, the interface IS the system !!
- UX is focused on the user's journey to solve a problem, UI is focused on how a product's surfaces look and function
- A UX designer is concerned with the conceptual aspects of the design process, leaving the UI designer to focus on the more tangible elements

[UI vs UX | Difference Between UI and UX | What is UX or UI (usertesting.com)]

Add React to a Website

- React has been designed from the start for gradual adoption, and **you can use as little or as much React as you need**.
- React components are a great way to to add some “sprinkles of interactivity” to an existing page
- The majority of websites aren’t, and don’t need to be, single-page apps. With **a few lines of code and no build tooling**, try React in a small part of your website. You can then either gradually expand its presence, or keep it contained to a few dynamic widgets.

How To Run React In VSCode (Visual Studio Code React.js Tutorial) - Bing video

React – A JavaScript library for building user interfaces (reactjs.org)

ReactJs Installation Tutorial on Windows 10 - React.js Setup in Visual Studio Code Tutorial 2020 - Bing video

Add React to a Website – React (reactjs.org)