Written Code

HyperText Markup Language

```
<!DOCTYPE html>
<html len="en">
    <head>

    </head>
    <body>
        <header>

        </header>
        <main>
            => only one element in <body>
        </main>
        <footer>

        </footer>
    </body>
</html>
```

```
<ol> ordered list
    <li> ~ </li>
    <li> ~ </li>
    <ul> unordered list
        <li> ~ </li>
        <li> ~ </li>
    </ul>
    <li> ~ </li>
</ol>
```

nested list

```
<h1> ··· </h1>
<h2> ··· </h2>
<h3> ··· </h3>
<h4> ··· </h4>
<h5> ··· </h5>
<h6> ··· </h6>
```

headings

```
<img src="image path", alt="alt msg", title="title" />
```

=> self-closing tag

```
<a href="https://www.google.com/"> Google </a>
```

- href: Hypertext Reference
- https://www.google.com/: absolute/relative

```
<form>
    <input type="text" placeholder="Full Name" name="name" />
    <input type="submit" />
</form>
```

Cascading Style Sheets

```
⋮
<head>
  ⋯
  <link href="stylefile.css" rel="stylesheet" />
  ⋯
</head>
```

separate .css file

inline

```
<p style="color: blue;"> blue text </p>
```

CSS Rule Syntax

```
selector {
  property : value ;
  property : value ;
    ⋮
  property : value ;
}
```

selector : which element to apply style
property : determine style

```
a, p, h2 {
  color : green ;
  font-size : 14pt ;
} grouping selectors

p {
  text-decorat : underline ;
}
```

Source Order

: later override earlier

Specificity

1. inline
2. id
3. class
4. type

```
⋮
<head>
  <style>
    h1 {
      color : blue ;
    }
  </style>
</head>
```

using <style> in <head>

```
.class { }        [text] { }        ⎫
#id { }           [text="~"] { }    ⎭ attribute selector

ul > li { }   ul의 직계 자식 li      a:hover { }    pseudo-class
div p { }     div 내부의 모든 p      a::before { }  pseudo-element
h1 + p { }    h1 바로 다음 p
```

Flexbox ← containers / items

```
div { ⇒ container
  display : flex ;
  flex-direction : row/column ;
  ⋯
}
```

Grid

```
div {
  display : grid ;
}
```

media queries

```
<head>
  ~
  <meta name="viewport" content="width : device-width, initial scale=1.0" />
  ~
</head>
```

Java Script

```
var name = expression ; // define variable globally
let name = expression ; // define variable with limited scope
const name = expression ; // define constant
```

⇒ "var": not recommended.

variable name : camelCase, case-sensitive

types: not specified.

number	array
string	null
boolean	undefined
object	function

Dynamically-typed language

```
let isValid = true;
isValid = "hello!"; // no error
isValid = 1; // no error
isValid = true; // no error
```

NaN, Not a Number

⇒ undefined numerical result

type: number

`${expression}`, backticks

```
let sumNum = 1+1 ;
```
⇒ number, 2

```
let sumStrNum = "1"+1 ;
```
⇒ string, '11'

```
let foo = null; // null
```
⇒ assigned value. exist but empty

```
let baz ; // undefined
```
⇒ declared but not assigned.

Strict equality ===, !==

1 === "1" ⇒ false

⇒ check type & value

Non-Strict equality ==, !=

1 == "1" ⇒ true.

⇒ type coercion.

Falsy value.

0, 0.0, NaN, "", null, undefined

truthy value

anything else

Define function.

```
function name(params) {
  statement;
  return ;
};
```

Anonymous function.

```
const sum = function(num1, num2) {
  return (num1+num2);
};
```

Arrow function

```
(param1, param2) => {
  statement;
  return ;
};
```

```
let person = {
  name : "Peter Parker",
  age : 23,
  "weight" : 71,
  friends : ["gwen", "black cat"],
};
person.age;
person["weight"];
person.friends[1];
person["name"];
person?.job ; // optional chaining
```

return undefined. no error.

```
let name = [ ];
name[index] = value ;
```
⇒ ["A", "B", undefined, "D"]

```
let name = [value0, value1, ..., valuen]; // prefilled.
name.length ; // property, not function
name[index]; // return value
name.indexOf(value); // return index
name.include(value); // return boolean.
```

```
const numbers = [1,2,3,4,5];
const mapTransform = numbers.map(el => el * 10);
const forEachTransform = [ ];
numbers.forEach(el => {
  forEachTransform.push(el * 10);
});
```

function

mapTransform = [10, 20, 30, 40, 50]

forEachTransform = [10, 20, 30, 40, 50]

Array.forEach(function(element, index, array))

```
<head>
  <script src="file path"></script>
</head>
```

link .js file to HTML page

DOM, JS ⇒ event-driven programming

```
let name = document.getElementById("id");
```
⇒ return DOM element for given id., No exist ⇒ null.

```
let name = document.querySelector(selector);
```
⇒ return first DOM element matched by given CSS selector
No exist ⇒ null

```
let name = document.querySelectorAll(selector);
```
⇒ return all elements matched by given CSS selector (NodeList)
No exist ⇒ empty array

Event Listener

```
element.addEventListener("event", handleFunction);

function handleFunction(event) {
  // event handling code
}

element.removeEventListener("event", handleFunction);
```

handleFunction: function name only / anonymous function / arrow function

removeEventListener: only function name, No anonymous, arrow

Hide DOM elements

```
element.style.display = "none";
```

Show DOM elements

```
element.style.display = "block";
```

Listening load events

```
window.addEventListener("load", init); // every resource loaded
document.addEventListener("DOMContentLoaded", init); // parse HTML DOM

function init() {
}
```

Creating DOM elements

```
let newHeading = document.createElement("h2");
newHeading.textContent = "new Heading is created";
document.body.appendChild(newHeading);
```

Manuplating DOM elements

```
<p id="text"> paragraph </p>

document.getElementById("text").textContent = "modified";
document.querySelector("#text").style.color = "blue";
```

Removing DOM elements

```
document.querySelector(".list-element").remove();

let li = document.querySelector(".list-element");
li.parentElement.removeChild(li);
```

```
<img id="mystery-box" src="question-block.png" />
<button id="box-btn"> Click me! </button>
```

```
let boxBtn = document.querySelector("#box-btn");
boxBtn.addEventListener("click", openBox);

function openBox() {
  let box = document.getElementById("mystery-box");
  box.src = "star.png";
}
```

Asynchronous Programming

```
<button id="btn">Click me!</button>
<p id="output"></p>

document.addEventListener("DOMContentLoaded", init);
function init() {
    id("btn").addEventListener("click", delayedMessage);
}

function delayedMessage() {
    id("output").textContent = "wait for surprise";
    setTimeout(sayHello, 2000);
}

callback
function sayHello() {
    id("output").textContent = "Tada!";
}
```

```
<p id="timer-text"></p>

let timerId = null;
function repeatedMessage() {
    timerId = setInterval(sayHello, 1000);
}

Callback
function sayHello() {
    id("timer-text").textContent = "Hello";
}

repeatedMessage();
```

Error First Callback

```
const doSomething = (callback) => {
    // do something
    callback(error, result);
}
                          Callback
doSomething((error, result) => {
    if (error) {
        console.log("error");
        return;
    }
    console.log("result");
});
```

Promise : pending, fulfilled, rejected

```
fetch(url)
  .then(response => response.json())   // fulfilled
  .then(json => console.log(json))
  .catch(error => console.log(error)); // rejected
```

Creating Promise

```
function setTimeoutPromise(time) {
    function executorFunction(resolve, reject) {
        setTimeout(function () {
            resolve();
        }, time);
    }
    return new Promise(executorFunction);
}
```

```
①console.log("Before setTimeoutPromise");
setTimeoutPromise(1000).then(function () {
    ②console.log("one second later");
});
③console.log("After setTimeoutPromise");
```

print order in console : ① → ③ → ②

async / await ⇒ return Promise

```
async function foo() {
    const result = await asyncFunction();
    console.log(result);
}
```

only in async function

```
async function fetchData() {
    try {
        const response = await fetch(url);
        const data = response.json();
        console.log(data);
    } catch (error) {
        console.error("error");
    }
}
```

Promise	async/await
.then(result ⇒ { ... })	const result = await ...
.catch(error ⇒ { ... })	try { ... } catch (error) { ... }
return Promise	X