HTML, CSS, JavaScript

#18011936 황윤하 #디자인이노베이션학과
#20011612 이승주 #컴퓨터공학과
#20011803 류지승 #데이터사이언스학과
#20011818 정소윤 #데이터사이언스학과

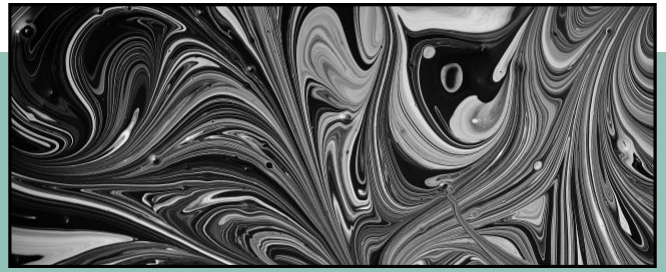
Open Year Round

#OYR #1조

#01

HTML

발표자: 20011612 이승주



#01

HTML?

- 웹사이트를 만들기 위한 마크업 언어
- 모두 태그들로 구성
- 태그는 <>로 표현

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
        웹사이트에 대한 설명,
        외부자료에 대한 참조 내용
    <title>Document</title>
</head>
<body>
    우리가 보는 화면의 내용
</body>
</html>
```

#01

HTML-text

<h1~h6>

제목 등에 많이 이용, 뒤에붙는 숫자별로 글씨의 크기가 다름 (1이 가장 큼)

<p>

문단을 넣을때 사용

**
**

줄바꿈에 이용

** & <b>**

굵은 글씨 표현

은 강조하는 의미가 더해짐

**<i> & **

이탤릭체 표현

은 강조하는 의미가 더해짐

** & **

영역 구분

: 박스단위 / : 줄단위

```
<div>
    <h1>html 소개</h1>
    <p>
        <strong>웹페이지</strong>를 만들기 위한 <em>마크업
        언어</em>
        <br>줄바꿈해서 다음줄
    </p>
</div>
```

html 소개

웹페이지를 만들기 위한 *마크업 언어*
줄바꿈해서 다음줄

#01
HTML-목록

<ol>

순서가 부여
자동으로 앞에 숫자가 붙음

<ul>

순서가 없음

<li>

ol, ul 태그 안에 하나의 목록 마다 쓰는 태그, list item

<dl>

단어를 정의하고 설명할때 사용
<dt><dd>와 함께 쓰임
<dt> 단어
<dd> 단어설명

```
<ol>
    <li>1번항목</li>
    <li>2번항목</li>
    <li>3번항목</li>
</ol>
<ul>
    <li>항목1</li>
    <li>항목2</li>
    <li>항목3</li>
</ul>
<dl>
    <dt>단어</dt>
    <dd>단어 설명해주세요</dd>
</dl>
```

1. 1번항목
2. 2번항목
3. 3번항목

- 항목1
- 항목2
- 항목3

단어
단어 설명해주세요

#01
HTML-테이블

<table>
테이블을 만듦
<caption>
테이블 이름 표시
<th>
테이블의 헤드부분
(가운데정렬, 굵은글씨)
<tr>
한 행 삽입
<td>
테이블의 한열

```
<table border="1">
    <caption>테이블 설명</caption>
    <tr>
        <td></td>
        <th>이름1</th>
        <th>이름2</th>
        <th>이름3</th>
    </tr>
    <tr>
        <th>성적</th>
        <td>4.5</td>
        <td>4.5</td>
        <td>4.5</td>
    </tr>
    <tr>
        <th>월급</th>
        <td>100</td>
        <td>100</td>
        <td>100</td>
    </tr>
</table>
```

테이블 설명

	이름1	이름2	이름3
성적	4.5	4.5	4.5
월급	100	100	100

HTML-링크&이미지

링크

<a href="원하는 주소">

다른 웹사이트로 가는 링크를 걸어주는 태그

이미지

<img src="원하는 이미지 주소" alt="이미지 불러오기 실패했을때 대체할 텍스트">

원하는 이미지를 넣을 수 있는 태그

#01 HTML

#폼

<form>

특정정보들을 묶어서 하나의 폼으로 제출할때 사용

ex) 로그인 정보, 회원가입 정보

<input type=???>

text: 일반문자

password: 비밀번호

email: 이메일

radio: 한 개만 선택할 수 있는 컴포넌트(name을 같게 설정해서 하나로 묶어주기)

checkbox: 다수 선택 하는 컴포넌트

file : 파일 업로드

submit: 제출

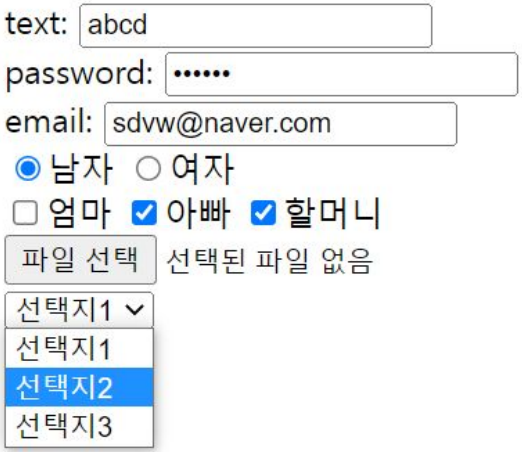
<select> 여러 개의 선택지중 하나를 선택

<option> 선택지를 만드는 태그

<label> 폼의 양식에 이름 붙이는 태그

#01 HTML #폼

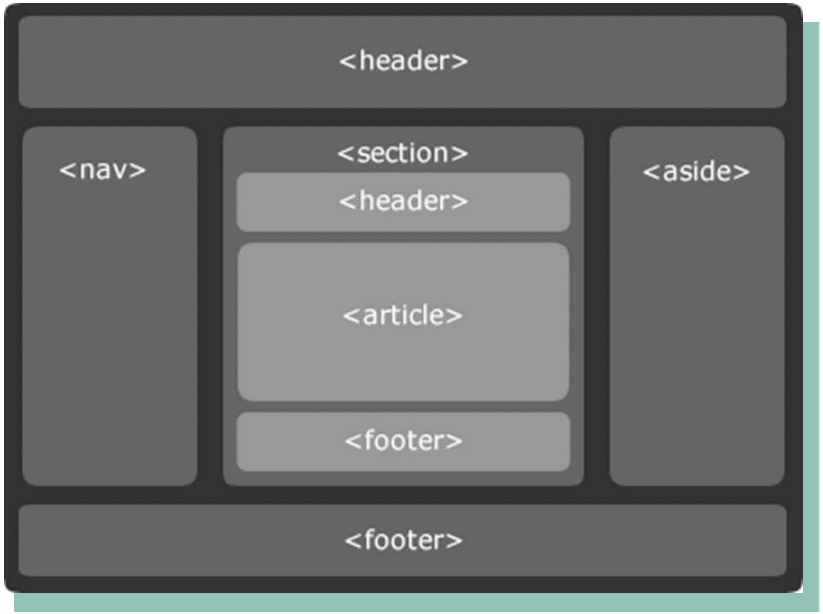
```
<form>
    <label>text: <input type="text"></label><br />
    <label>password: <input type="password"></label><br />
    <label>email: <input type="email"></label><br />

    <label><input type="radio" name="sex">남자</label>
    <label><input type="radio" name="sex">여자</label><br />

    <label><input type="checkbox" name="family">엄마</label>
    <label><input type="checkbox" name="family">아빠</label>
    <label><input type="checkbox" name="family">할머니</label>
    <br />

    <input type="file">

    <select>
        <option>선택지1</option>
        <option>선택지2</option>
        <option>선택지3</option>
    </select>
</form>
```



#01 HTML

#시멘틱 태그

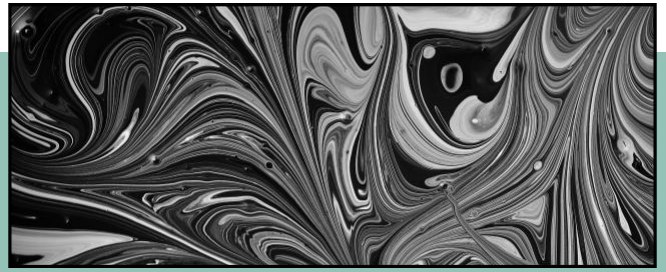
- 의미가 부여됨, div태그와 같은 기능
- **<header>** 제목, 헤더
- **<nav>** 네비게이션, 목차, 리스트 등 다른 페이지로의 이동을 위한 링크공간을 위주로 표현
- **<main>** 문서의 중심주제 주요 내용
- **<aside>** 좌측 또는 우측 사이드 공간, 본문 외에 부수적인 내용 표현
- **<article>** 기사, 블로그 포스트 등 텍스트 페이지 구성에 이용
- **<section>** 카테고리 별로 섹션을 구분하는 용도의 태그, 같은 테마를 하나의 콘텐츠로 그룹화
- **<footer>** 바닥글, 문서하단에 들어가는 공간



#02

CSS

발표자 : 20011803 류지승



CSS(Cascading Style Sheet) 정의

문서의 콘텐츠와 레이아웃, 글꼴, 및 시각적 요소들로 표현되는
문서의 외관(디자인)을 분리하기 위한 목적으로 만들어졌다.

CSS 속성

- 여러 속성 값은 반드시 공백으로 구분되어야 한다.
- 축약 표현 속성은 여러 속성 값을 하나의 간소화된 선언으로 적용할 수 있다.
- 속성이 명시되지 않으면 해당 속성의 기본 값이 적용된다.

#01

CSS 정의 & 속성

css 속성 위치

link 요소를 이용한 외부 css 파일

```
<head>
    <link href="style.css" rel="stylesheet" type="text/css">
</head>
```

Import 를 이용한 외부 css 파일

```
<head>
    <style type="text/css">
        @import url(style.css);
    </style>
</head>
```

css 속성 위치

HTML문서의 head 태그

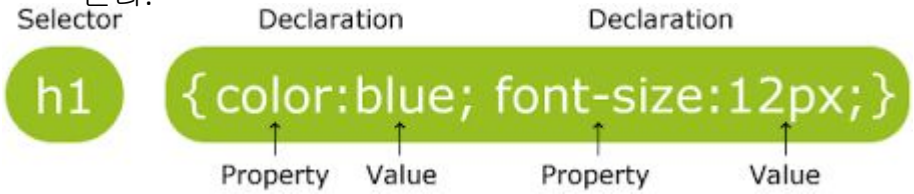
```
<head>
    <style type="text/css">
        /* */
    </style>
</head>
```

HTML요소에 인라인 스타일로
삽입

```
<body>
    <div style=color: blue;></div>
</body>
```

CSS 기본문법

css 규칙은 선택자(selector)와 선언부(declaration)로 나뉜다.
선언부는 다시 속성(property)과 속성값(value)로 나누게 된다.



ex) 선택자(.className / #id / tag) / 선언부 (선택자를 제외한 나머지)
속성(color / font-size / background-color / display) / 속성값(10px / red)

- 선택자는 보통 스타일링하고 싶은 HTML요소나 부여한 ID 또는 class가 위치한다.
- 선언부에 여러개의 속성과 속성값이 있을 때는 ;(세미콜론)으로 구분한다.
- 각각의 선언은 속성과 속성값을 : (콜론)으로 구분한다.

css 우선순위

1. 상속

```
<style>
  body {
    background-color: black;
    font-family: monospace;
    color: green;
  }
</style>
<body>
  <h1>Hello World!</h1>
</body>
```

css 우선순위

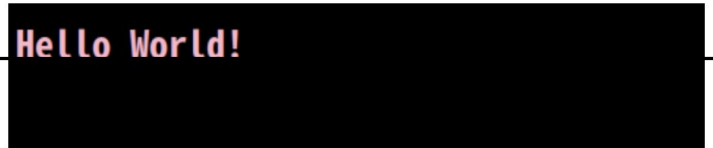
2. 타입선택자(Element)로 스타일 적용

```
<style>
  body {
    background-color: black;
    font-family: monospace;
    color: green;
  }
  h1 {
    color: red;
  }
</style>
<body>
  <h1>Hello World!</h1>
</body>
```

css 우선순위

3. 선택자(class)로 스타일 적용

```
<style>
  body {
    background-color: black;
    font-family: monospace;
    color: green;
  }
  .pink-text {
    color: pink;
  }
  h1 {
    color: red;
  }
</style>
<body>
  <h1 class="pink-text">
    Hello World!
  </h1>
</body>
```



css 우선순위

4. Style 적용 순서에 따라

```
<style>
  body {
    background-color: black;
    font-family: monospace;
    color: green;
  }
  .pink-text {
    color: pink;
  }
  .blue-text {
    color: blue;
  }
  h1 {
    color: red;
  }
</style>
<body>
  <h1 class="blue-text pink-text">
    Hello World!
  </h1>
</body>
```

css 우선순위

5. Id Style 적용

```
<style>
  body {
    background-color: black;
    font-family: monospace;
    color: green;
  }
  #orange-text {
    color: orange;
  }
  .pink-text {
    color: pink;
  }
  .blue-text {
    color: blue;
  }
  h1 {
    color: red;
  }
</style>
<body>
  <h1 id="orange-text" class="blue-text pink-text">
    Hello World!
  </h1>
</body>
```

css 우선순위

6. Inline Style 적용

```
<style>
  body {
    background-color: black;
    font-family: monospace;
    color: green;
  }
  #orange-text {
    color: orange;
  }
  .pink-text {
    color: pink;
  }
  .blue-text {
    color: blue;
  }
  h1 {
    color: red;
  }
</style>
<body>
  <h1 id="orange-text" class="blue-text pink-text" style="color: white">
    Hello World!
  </h1>
</body>
```

css 우선순위

7. 타입선택자(Element)로 스타일 적용

```
<style>
  body {
    background-color: black;
    font-family: monospace;
    color: green;
  }
  #orange-text {
    color: orange;
  }
  .pink-text {
    color: pink;
  }
  .blue-text {
    color: blue;
  }
  h1 {
    color: red !important;
  }
</style>
<body>
  <h1 id="orange-text" class="blue-text pink-text" style="color: white">
    Hello World!
  </h1>
</body>
```

자주 사용하는 CSS 속성

width & height : 가로 / 세로

- 값의 단위

  auto -> 기본값, 브라우저가 계산한 너비

  px -> 픽셀

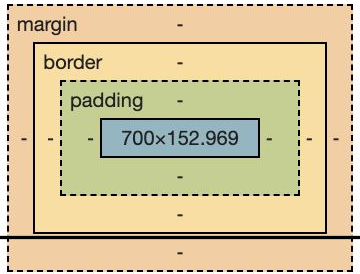
  % -> 부모 요소에 상대적인 너비

  initial -> 기본값으로 초기화

  inherit -> 부모 요소로부터 상속 받은 값

margin & padding : 바깥쪽 여백 / 안쪽 여백

top / bottom / right / left를 이용하여 방향 지정 가능



자주 사용하는 CSS 속성

color : 글자 색상 / background-color : 배경색

값

inherit -> 기본값 , 부모의 색상을 가져온다

red 또는 blue -> 이미 css로 정의된 색상

#000 또는 #FFFFFF -> 16진수의 색상코드

rgb(255,255,255) -> rgb 색상

rgba(200,100,150,0.5) -> 알파(투명도)가 적용된 rgba 색상

font : 글자의 폰트

font 의 속성들

font-style -> 기울기 등의 스타일 지정

font-weight -> 글자 두께

font-variant -> 글꼴 변형 (소문자를 대문자로 바꾸는 등)

font-size -> 글자 크기

line-height -> 줄 간격

font-family -> 글꼴

자주 사용하는 CSS 속성

border : 테두리

border-width -> 테두리의 두께 px 단위사용

border-style -> 테두리의 스타일

border-color -> 테두리의 색상 .. color 값과 동일함

display : 요소 보여주기

속성

none -> 보이지 않음

block -> 블록 박스 (세로)

inline -> 인라인 박스 (가로)

inline-block -> 블록과 인라인의 중간 형태

자주 사용하는 CSS 속성

position : 요소 위치 설정

값

static -> default값, 다른 태그와의 관계에 의해 자동으로 배치됨 . 임의로 설정 불가

absolute -> 절대 좌표와 함께 위치를 지정해 줄 수 있다.

relative -> 원래 있던 위치를 기준으로 좌표를 지정한다.

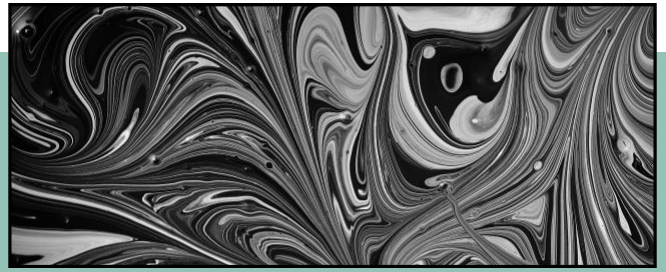
fixed -> 스크롤과 상관없이 항상 문서 최 좌측 상단을 기준으로 좌표를 고정한다.

inherit -> 부모 태그를 상속받음.

#03

Javascript ®

발표자: 18011936 황윤하, 20011818 정소윤



변수

정의 및 선언

데이터를 저장할 때 쓰이는 '이름이 붙은 저장소'

변경 가능한 값

let 키워드 사용해 선언 (한 번만 선언)

```
let num = 4;
num = 'four';

alert(num) //four
```

변수 명명

변수 명명 규칙

1. 문자와 숫자, 기호는 $, _만 들어갈 수 있음
2. 첫 글자에 숫자 X
3. 예약어 사용 X ex) let, class, return, function, break, continue ...

카멜 표기법

여러 단어 조합할 때 사용

단어를 차례대로 나열하면서 첫 단어를 제외한 각 단어의 첫 글자를 대문자로 작성 ex) openYearRound

서술적이고 간결한 변수명을 지어야 한다.

상수

정의 및 선언

변하지 않는 값 → 변경하려고 하면 에러 발생

const 키워드 사용해 선언

대문자 상수

기억하기 힘든 값을 상수에 할당해 별칭으로 사용하는 관습,
주로 대문자와 밑줄로 구성된 이름으로 명명

```
const COLOR_RED = "#F00";
COLOR_RED = "CHANGE"; //ERROR
```

자료형

#1 숫자형

정수 및 부동소수점 숫자

#2 BigInt

길이에 상관없이
정수를 나타낼 수 있음

#3 문자형

문자열
따옴표에 넣어서 표현

#4 boolean형

긍정(true)이나 부정(false)을
나타내는 값을 저장할 때 사용

#5 null

존재하지 않는 값,
비어 있는 값,
알 수 없는 값을
나타내는 데 사용

#6 undefined

값이 할당되지 않은 상태
를 나타낼 때 사용

#7 객체

다양한 데이터를
담을 수 있음

#8 심볼

객체의 고유한 식별자를
만들 때 사용

primitive

(숫자형, 문자형, boolean ...)

```
//원시형 변수는 데이터 자체를 담고 있음
let num1 = 4;
let num2 = num1; //num1의 값인 4가 할당됨

console.log(num1); //4
console.log(num2); //4

num1 = 9;

console.log(num1); //9
console.log(num2); //4
```

데이터 자체를 담고 있음

object

(객체)

```
//객체형 변수는 객체가 저장된 레퍼런스(메모리 주소)를 담고 있음
const obj1 ={
  name:'정소윤',
  age:23
};
const obj2 = obj1;

console.log(obj1.name) //정소윤
console.log(obj2.name) //정소윤

obj1.name = 'jeongsoyoon';

console.log(obj1.name) //jeongsoyoon
console.log(obj2.name) //jeongsoyoon
```

객체가 저장되어 있는 메모리 주소인
객체에 대한 참조 값 담고 있음

산술연산자

이항 산술 연산자

- 덧셈 연산자 +
- 뺄셈 연산자 -,
- 곱셈 연산자 *,
- 나눗셈 연산자 /,
- 나머지 연산자 %,
- 거듭제곱 연산자 **

```
console.log(4+2); //6
console.log(4-2); //2
console.log(4*2); //8
console.log(4/2); //2
console.log(4%2); //0
console.log(4**2); //16
```

단항 산술 연산자

- 증가 연산자 ++
  : 변수를 1 증가시킴
- 감소 연산자 –
  : 변수를 1 감소시킴

```
let num = 1;
num++; //1 증가
console.log(num); //2
num--; //1 감소
console.log(num); //1
```

할당연산자

우항의 값을 좌항에 대입

할당 연산자 =

```
a = 2;
```

복합 할당 연산자
: +=, -=, *=, /=, %=

```
let a = 2;

a += 1; // a = a+1; , 3
a -= 1; // a = a-1; , 2
a *= 2; // a = a*2; , 4
a /= 2; // a = a/2; , 2
a %= 2; // a = a%2; , 0
```

비교연산자

boolean형 반환

대소관계 비교 연산자
: >, <, <=, >=

```
console.log('Z'>'A'); //true
```

동등, 일치 비교 연산자

- 동등 연산자 : **==**
- 부등 연산자 : **!=**
- 일치 연산자 : **===**
- 불일치 연산자 : **!==**

```
console.log(0==false);
// true, false가 0으로 변환됨
console.log(0===false);
// false, 0은 숫자형이고 false는 boolean형
```

일치 연산자, 불일치 연산자는
형변환없이
자료형의 동등 여부까지 검사

논리연산자

|| (OR)

피연산자 중 하나라도 true라면 true 반환

```
console.log(false || false); //false
console.log(false || true); //true
console.log(true || false); //true
console.log(true || true); //true
```

&& (AND)

두 피연산자 모두 true이면 true 반환

```
console.log(false && false); //false
console.log(false && true); //false
console.log(true && false); //false
console.log(true && true); //true
```

! (NOT)

피연산자가 true라면 false,
false라면 true 반환

```
console.log(!false); //true
console.log(!true); //false
```

함수

#1 정의

함수를 정의하는 3가지 방법:
함수 선언문 / 함수 표현식 / 화살표 함수

#2 호출과 호이스팅

호이스팅

#3 익명함수

이름을 지정하지 않는

#4 즉시실행함수

바로 써버리는

#5 콜백함수

필요할 때 호출

#01 함수를 정의하는 3가지 방법

#1 함수 선언문

```
function 함수이름 (매개변수1, 매개변수2) {
  실행문;
  return 반환값;
};
```

매개변수 자료형 명시 X
& 인자로 전달된 값에 대한 자료형 검사 X
→ 정의보다 적은 매개변수가 사용되어도 오류X
→ undefined 값 전달

#2 함수 표현식

```
var/let/const 변수 = function 함수이름 (매개변수1, 매개변수2){실행문};
```

#3 화살표 함수

```
var/let/const 함수이름 = (매개변수1, 매개변수2) => {실행문};
```

function이라는 키워드 대신 =>(fat arrow) 사용

#02
호출과 호이스팅

hoisting

함수 내 모든 변수는 어느 시점에 선언해도 함수의 맨 처음에 선언된 것처럼 동작

전역
```
var 변수1 = 10;

function 함수_A(){
  document.write("변수1의 값은" + 변수1 + "입니다.<br>");
지역 var 변수1 = 20;
  document.write("변수1의 값은" + 변수1 + "입니다.<br>");
}

함수_A();
```

```
var 변수1 = 10;

function 함수_A(){
  var 변수1;
① document.write("변수1의 값은" + 변수1 + "입니다.<br>");
  변수1 = 20;
② document.write("변수1의 값은" + 변수1 + "입니다.<br>");
}

함수_A();
```

① 변수1: 선언 O 초기화 X >> undefined값 반환
② 변수1: 선언 O 초기화 O >> 20 반환

#03

익명함수

함수명 없이 함수표현식으로
변수에 함수 실행코드 저장

```
//이름이 있는 일반 함수
var 변수 = function 함수이름(매개변수1, 매개변수2){실행문};

//익명함수
var 변수 = function (매개변수1, 매개변수2){실행문};
```

#04

즉시실행함수, IIFE

Immediately Invoked Function Expression

정의되자마자 즉시 실행되는
함수표현식

일회용성이 강하므로 익명함수에
주로 쓰임 : 개발자들 사이에서도
의견이 갈리긴 함

```
//익명함수로 IIFE
(function (매개변수1,매개변수2){ 실행문 })(인자1,인자2);

//화살표함수로 IIFE
((매개변수1,매개변수2) => { 실행문 })(인자1, 인자2);
```

#05

콜백함수

다른 함수에 매개변수로 넘겨준 함수

매개변수로 지정되면 일단 넘겨받고,
필요 시에 호출(call back)

```
function 함수A(count, 매개변수2_함수, 매개변수3_함수){
  count < 3 ? 매개변수2_함수() : 매개변수3_함수();
}
function 함수B(){실행문;}
function 함수C(){실행문;}

함수A(2, 함수B, 함수C);
```

조건문

#1 if문

if / if-else / if-else if- else
삼항연산자

#2 switch문

호이스팅

#01 if문

주어진 **표현식**의 결과에 따라 명령을 수행하도록 제어하는 실행문

#1 if문

```
if (표현식) {
  표현식=참 의 실행문;
}
```

#2 if-else문

```
if (표현식) {
  표현식=참 의 실행문;
} else {
  표현식=거짓 의 실행문;
}
```

#3 if-else if-else문

```
if (표현식1) {
  표현식1=참 의 실행문;
} else if (표현식2) {
  표현식1=거짓 && 표현식2=참 의 실행문;
} else {
  표현식1=거짓 && 표현식2=거짓 의 실행문;
}
```

if-else문 ⇒ 삼항연산자

```
if (표현식) {
  return 반환값1;
} else {
  return 반환값2;
}
```

```
표현식 ? 반환값1 : 반환값2
```

#02 switch문

if문보다 가독성이 높아진 문법

```
switch (조건값){
  case 값1:
    조건값 = 값1 의 실행문;
    break;
  case 값1:
    조건값 = 값1 의 실행문;
    break;
  ...
  default:
    조건값이 어느 값에도 해당되지 않을 때의 실행문;
    break;
```

case가 끝나면 switch문 전체를 종료
break가 없으면 다음 case로 넘어가서
각 case와의 값 일치 여부와 상관 없이 이후 모든 실행문 실행

반드시 필요 X

반복문

#1 while문

이름을 지정하지 않는

#2 do while문

바로 써버리는

#3 for문

이곳에 목차에 대한 상세한 설명을 자유롭게 적어 주세요.

#01 while문

표현식을 만족하는 이상
반복실행

```
while (표현식){
  표현식=참인 동안의 실행문;
}
```

표현식 내 값이 바뀌지 않으면
⇒ 무한루프

[표현식 검사 → 참 → 실행문 실행] 반복

#02 do while문

일단 루프 한 번 돌려

```
do {
  표현식=참인 동안의 실행문;
} while (표현식);
```

실행문 1회 실행 →
[표현식 검사 → 참 → 실행문 실행] 반복

#03 for문

자체적으로 초기식, 표현식, 증감식 포함

```
for (초기식; 표현식; 증감식){
  표현식=참인 동안의 실행문;
}
```

```
for (변수 in 객체){
  객체 내 속성의 개수만큼 실행하는 실행문;
}
```

변수에 속성의 이름이 대입되어 각 속성에 순차적으로 접근 가능

객체

다양한 데이터를 담을 수 있음

키와 값 쌍으로 이뤄진 프로퍼티의 집합

키에는 문자형, 심볼, 값에는 모든 자료형이 허용

```
let User = {
  name : '정소윤', //키:"name", 값:"정소윤"
  age : 23, //키:"age", 값:23
  sayHello : function(){
    console.log("Hello my name is "+this.name);
  },
};
```

객체 생성 01:
객체 리터럴

중괄호 안에 키와 값 작성

```
let User = {
  name : '정소윤', //키:"name", 값:"정소윤"
  age : 23, //키:"age", 값:23
  sayHello : function(){
    console.log("Hello my name is "+this.name);
  },
};
```

객체 생성 02:
기본 객체 생성자 함수 (Object())

빈 객체 생성 후
프로퍼티 추가

```
let User = new Object(); //빈 객체 생성
User.name = '정소윤'; //프로퍼티 추가
User.age = 23;
User.sayHello = function(){
  console.log("Hello my name is "+this.name);
};
```

객체 생성 03:
생성자 함수

생성자 함수에 인자 전달

```
function User(name, age){
  this.name = name;
  this.age = age;
  this.sayHello = function(){
    console.log("Hello my name is "+this.name);
  }
};

const user1 = new User("정소윤", 23);
```

상수 객체도 수정될 수 있음
const는 변수가 담고 있는 주소는 고정하지만,
그 객체의 내용은 고정하지 않음

객체 접근

값 읽기,
프로퍼티 추가 및 변경

점 표기법 & 대괄호 표기법

```
console.log(User.name); //점 표기법
console.log(User["name"]); //대괄호 표기법
User.name = 'jeongsoyoon'; // 변경
User["is female"]= true; // 생성
```

객체 탐색

객체의 모든 키 순회
for(key in object)

```
let User = {
  name : '정소윤', //키:"name", 값:"정소윤"
  age : 23, //키:"age", 값:23
  "is female" : true //키:"is female", 값:true
};

for(let key in User){
  console.log(key); //name,age,is female
  console.log(User[key]); //'정소윤', 23, true
}
```

배열

순서가 있는 컬렉션을 저장할 때 쓰는 자료구조

```
let fruits = ["apple","banana","strawberry"];
```

배열 생성 01: 배열 리터럴

대괄호 안에 요소들 작성

```
let arr = [];
let fruits = ["apple","banana","strawberry"];
```

배열 생성 02: 기본 배열 생성자 함수 (Array())

괄호 안에 요소들 작성

```
let arr = new Array();
let fruits = new Array("apple","banana","strawberry");
```

각 배열 요소에는 0부터 시작하는 숫자(인덱스) 매겨짐
대괄호 안에 인덱스 넣어줌

배열 접근

값 읽기,
요소 추가 및 변경

```
let fruits = ["apple", "banana", "strawberry"];
console.log(fruits[1]); //값 읽기, banana
fruits[2] = "blueberry"; //요소 수정
fruits[3] = "lemon"; //요소 추가
//  ["apple", "banana", "strawberry", "lemon"]
```

배열 탐색

배열 모든 요소 순회
for(elem of array)
+) for(elem in array)도 가능

```
let fruits = ["apple", "banana", "strawberry"];
for(let fruit of fruits){
  alert(fruit);
}
```

배열 요소 추가

push & unshift

push

괄호 안 요소를 배열 끝에 추가, 요소 이동 X

```
let fruits = ["apple", "banana", "strawberry"];
fruits.push("lemon","kiwi");
alert(fruits); //["apple", "banana", "strawberry", "lemon", "kiwi"]
```

unshift

괄호 안 요소를 배열 앞에 추가
모든 요소를 오른쪽으로 이동 **=>** 느림

```
let fruits = ["apple", "banana", "strawberry"];
fruits.unshift("lemon","kiwi");
alert(fruits); //["lemon", "kiwi", "apple", "banana", "strawberry"]
```

배열 요소 삭제

pop & shift

pop

배열 끝 요소를 제거하고 제거한 요소를 반환, 요소 이동 X

```
let fruits = ["apple", "banana", "strawberry"];
alert(fruits.pop()); //strawberry
alert(fruits); //["apple", "banana"]
```

shift

배열 앞 요소를 제거하고 제거한 요소를 반환
모든 요소를 왼쪽으로 이동 **=>** 느림

```
let fruits = ["apple", "banana", "strawberry"];
alert(fruits.shift()); //apple
alert(fruits); //["banana", "strawberry"]
```

배열 메소드

splice(pos, deleteCount, ...items)	pos부터 deleteCount개의 요소를 지우고, items 추가하기
slice(start, end)	start부터 end 바로 앞까지의 요소를 복사해 새로운 배열을 만듦
concat(...items)	배열의 모든 요소를 복사하고 items를 추가해 새로운 배열을 만든 후 이를 반환함
indexOf/lastIndexOf(item, pos)	pos부터 원하는 item을 찾음. 찾게 되면 해당 요소의 인덱스를, 아니면 -1을 반환함
find/filter(func)	func의 반환 값을 true로 만드는 첫 번째/전체 요소를 반환함
forEach(func)	모든 요소에 func을 호출함. 결과는 반환되지 않음
map(func)	모든 요소에 func을 호출하고, 반환된 결과를 가지고 새로운 배열을 만듦
sort(func)	배열을 정렬하고 정렬된 배열을 반환함

THANK YOU ®

HTML, CSS, JavaScript

Open Year Round
1조

#18011936 황윤하 #디자인이노베이션학과
#20011612 이승주 #컴퓨터공학과
#20011803 류지승 #데이터사이언스학과
#20011818 정소윤 #데이터사이언스학과