JavaScript

자바스크립트(JavaScript)

1. **동적인 정보 표현**
    - HTML/CSS만으로는 정적인 웹 페이지를 만들 수 있음
    - JS를 사용하면 **버튼 클릭, 입력값 처리, 실시간 데이터 업데이트 등** 동적인 기능 구현 가능
2. **애니메이션, 상호작용**
    - 화면 요소 이동, 색상 변화, 슬라이드 쇼 등 **애니메이션 추가** 가능
    - 이벤트(클릭, 마우스 이동, 키보드 입력 등)에 따라 반응하는 **인터랙티브 웹** 제작 가능
3. **HTML/CSS 조작**
    - DOM(Document Object Model)을 통해 **HTML 요소 생성, 삭제, 수정** 가능
    - CSS 스타일을 변경하여 **실시간 디자인 변경** 가능
4. **스크립팅 언어**
    - 웹 브라우저에서 실행되는 **프로그래밍 언어**
    - 서버(Node.js)에서도 실행 가능, 즉 **클라이언트 + 서버** 양쪽에서 사용 가능

document와 DOM

- document
    - 자바스크립트에서 브라우저가 제공하는 전역 객체
    - 현재 웹페이지(HTML 문서)를 자바스크립트에서 접근할 수 있는 객체
- DOM(Document Object Model)
    - HTML 문서를 객체 트리(Tree)구조로 표현한 것
    - 자바스크립트가 HTML 요소를 조작, 읽기, 변경할 수 있게 해주는 모델
- 즉, document는 DOM을 조작할 수 있는 대표 객체

변수

- 변수(Variable)는 값을 담기 위한 공간이다. 예를 들어 사용자 이름, 주소, 쇼핑몰 사이트의 상품 항목 등등의 값을 저장하기 위해 변수를 사용한다.
- 사용 방법
    - let username = "Alberto";
    - const username = "Alberto";
    - 키워드
        - const - 상수를 저장하기 위한 공간. 즉 값의 변경이 불가능하다. 값의 변경이 불가하기 때문에 저장된 값은 변조되지 않았다는 것을 보장받는다
        - let - const와 저장된 내용은 변경이 가능하다.
    - 기본적으로 const를 사용. 재할당이 필요한 경우에만 let사용

- 변수 작명 규칙
  - 변수 이름은 숫자로 시작할 수 없다(let 3apple = "apple")
  - 변수 이름 중 공백은 들어갈 수 없다(let app le = "apple")
  - 변수 이름은 의미를 부여해서 만드는게 좋다(let username = "Alberto")
  - 합성어가 있는 경우 캐멀 케이스방법을 사용한다(let firstName = "Alberto")

자료형

- string : 문자열. 큰따옴표" 또는 작은 따옴표' 로 묶여 있는 값(예 : let name = "Alberto")
- number : 숫자. 따옴표 없이 들어오는 수(예 : let age = 20, let num = 1.123)
- boolean : 불리언. 참 또는 거짓의 값(예 : let bool = true, let bool = false)
- null : 널. "값이 없다"를 의도적으로 표시한 값. (예 : let a = null)
- undefined : 정의되지 않음 변수를 선언만 하고 값을 넣지 않았을 때 자동으로 들어가는 값(예 : let number;)

객체

- 위에서 설명한 자료형은 하나의 값만 저장할 수 있지만, 객체(object)는 여러 값을 저장할 수 있다
- Key(name, age, address)와 value("alberto", 30, "seoul")으로 쌍을 이루고 있다
- 객체 생성 예

```
const user = {
        name : "alberto",
        age : 30,
        address : "seoul"
}
```

- 사용 법 : user[name] 지정하면 값인 "alberto"가 표현된다.(user[key])

배열

- 순서대로 값을 저장하는 객체. Key대신 index로 접근한다
- index는 0부터 시작한다
- 배열 생성 예 : const fruitBasket = ['apple', 'banana', 'orange'];
- 사용 법 : fruitBasket[0] 표현하면 apple를 사용하게 된다. fruitBasket[1] 표현하면 banana를 사용.

연산자

- 산술 연산자
  - 연산된 결과를 수 또는 값으로 표현

연산자	의미	예시
+	더하기	2 + 3 → 5
-	빼기	5 - 2 → 3
*	곱하기	4 * 2 → 8
/	나누기	6 / 3 → 2
%	나머지	7 % 3 → 1
**	거듭제곱	2 ** 3 → 8

연산자	의미	예시
+	더하기	2 + 3 → 5
-	빼기	5 - 2 → 3
*	곱하기	4 * 2 → 8
/	나누기	6 / 3 → 2
%	나머지	7 % 3 → 1
**	거듭제곱	2 ** 3 → 8
++	1 증가	let a = 1; a++ → 2
--	1 감소	let a = 2; a-- → 1

- 예제

```
let a = 10, b = 3;
console.log(a + b); // 13
console.log(a - b); // 7
console.log(a * b); // 30
console.log(a / b); // 3.333...
console.log(a % b); // 1 (나머지)
console.log(a ** b); // 1000 (거듭제곱)
```

- 예제(증감 연산자)

```
let n = 5;
console.log(++n); // 6 (전위 증가)
console.log(n++); // 6 (후위 증가, 출력 후 증가)
console.log(n);   // 7
console.log(--n); // 6 (전위 감소)
console.log(n--); // 6 (후위 감소)
console.log(n);   // 5
```

- 할당 연산자
  - 오른쪽의 연산된 결과 값을 왼쪽 공간에 저장해 준다

연산자	의미	예시
=	대입	let a = 5;
+=	더해서 대입	a += 3 → a = a + 3
-=	빼서 대입	a -= 2 → a = a - 2
*=	곱해서 대입	a *= 2 → a = a * 2
/=	나누어서 대입	a /= 2 → a = a / 2
%=	나머지 대입	a %= 3 → a = a % 3
**=	거듭제곱 대입	a **= 2 → a = a ** 2

-

- 예제

```
let x = 5;
x += 3; // x = x + 3
console.log(x); // 8
x -= 2; // x = x - 2
console.log(x); // 6
x *= 2;
console.log(x); // 12
x /= 3;
console.log(x); // 4
x %= 3;
```

```
console.log(x); // 1
```

- 비교 연산자
  - 결과적으로 true(참) 또는 false(거짓)의 결과가 나온다

연산자	의미	예시
==	값이 같음	'5' == 5 → true
===	값과 타입이 같음	'5' === 5 → false
!=	값이 다름	5 != 3 → true
!==	값 또는 타입이 다름	'5' !== 5 → true
>	크다	5 > 3 → true
<	작다	3 < 5 → true
>=	크거나 같다	5 >= 5 → true
<=	작거나 같다	3 <= 5 → true

  - **예제**

```
console.log(5 == '5'); // true (값만 비교)
console.log(5 === '5'); // false (값 + 타입 비교)
console.log(5 != 3); // true
console.log(5 !== '5'); // true
console.log(5 > 3); // true
console.log(5 < 3); // false
console.log(5 >= 5); // true
console.log(5 <= 4); // false
```

- 논리 연산자
  - 여러 개의 식을 한줄로 표현하고자 하는 경우 유용하게 사용할 수 있다
  - 결과적으로 true(참) 또는 false(거짓)의 결과가 나온다

연산자	의미	예시
&&	AND, 모두 참이면 참	true && false → false
||	OR, 하나라도 참이면 참	true && false → true
!	NOT, 반대로	!true → false

  - **예제**

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

- 조건(삼항) 연산자
  - 조건 ? 참일 때 값 : 거짓일 때 값
  - (특적 식) ? 참인 경우 실행 : 거짓인 경우 실행;

```
let age = 18;
let result = age >= 18 ? "성인" : "미성년자";
console.log(result); // "성인"
```

- 타입 연산자

연산자	의미	예시
typeof	자료형 확인	typeof 123 → "number"
instanceof	객체 타입 확인	[] instanceof Array → true

- **예제(typeof)**

```
console.log(typeof 123);        // "number"
console.log(typeof "Hello");    // "string"
console.log(typeof true);       // "boolean"
console.log(typeof undefined);  // "undefined"
console.log(typeof null);       // "object"
console.log(typeof {a: 1});     // "object"
console.log(typeof [1, 2, 3]);  // "object" ( 배열도 object )
console.log(typeof function(){}); // "function"
```

- **예제(instanceof)**

```
let arr = [1,2,3];
let date = new Date();

console.log(arr instanceof Array);    // true
console.log(arr instanceof Object);   // true
console.log(date instanceof Date);    // true
console.log(date instanceof Object);  // true
console.log(date instanceof Array);   // false
```

제어문

- 프로그램의 흐름(순서)을 제어하는 문법
- 조건문(if / else if / else)
  - 조건 만족 시(true) 특정 코드 실행

```
let age = 20;

if (age >= 18) {
  console.log("성인입니다.");
} else if (age >= 13) {
  console.log("청소년입니다.");
} else {
  console.log("어린이입니다.");
}
// 출력: 성인입니다.
```

- switch문
  - 여러 조건을 한 번에 비교할 때 사용

```
let day = 3;
switch(day) {
  case 1:
    console.log("월요일");
    break;
  case 2:
    console.log("화요일");
    break;
  case 3:
    console.log("수요일");
    break;
```

```
    default:
      console.log("기타 요일");
  }
  // 출력: 수요일
```

- 반복문(for)
  - 정해진 횟수만큼 반복할 때 사용
  - for(초기값 ; 조건식 ; 증감식){
    종속문장들 실행;
    }
    - 초기값 : 변수를 초기화 하는 경우 사용(생략 가능)
    - 조건식 : 특정 조건이 참인 경우 종속문장 실행
    - 증감식 : 특정 변수의 값을 증가 또는 감소시켜주는 역할
      종속문장 실행 후 증감식 동작
    - 동작 순서 : 1(초기값) -> 2(조건식) -> 3(증감식) -> 조건식
    - 초기값은 처음 딱 한번만 보며 2,3번과 종속문장을 반복한다

```
for (let i = 1; i <= 5; i++) {
  console.log(i);
}
// 출력: 1 2 3 4 5
```

  - for of(배열의 값이 num변수에 하나씩 저장 됨)

```
const numbers = [1,2,3]
for ( const num of numbers ) {
  console.log(i);
}
// 출력: 1 2 3
```

  - for in(키를 표현해 준다)

```
const numbers = [1,2,3]
for ( const num in numbers ) {
  console.log(i);
}
// 출력: 0 1 2( 인덱스 출력 )
const user = { age:30, addr:"seoul",}
for ( const u in user ) {
  console.log(u);
}
// 출력: age, addr( key 출력 )
```

- 반복문 (while)
  - 조건이 참일 동안 반복
  - for문과 다르게 조건식만 존재

```
let i = 1;
while (i <= 5) {
  console.log(i);
  i++;
}
```

```
// 출력: 1 2 3 4 5
```

- 제어 (break / continue)
  - break : 반복문을 즉시 종료. 또는 switch문 종료
  - continue : 현재 반복만 건너뛰고 다음 반복으로 이동

```
for (let i = 1; i <= 5; i++) {
  if (i === 3) continue; // 3은 건너뜀
  if (i === 5) break;   // 5가 되면 종료
  console.log(i);
}
// 출력: 1 2 4
```

함수

- 어떤 동작(작업)을 모아둔 코드 덩어리
- 필요할 때 호출해서 사용할 수 있다
- 같은 코드를 여러 번 쓰지 않고 재사용할 수 있다
- 함수 선언식 예

```
function sayHello() { //함수 선언
 console.log("안녕하세요!");
}
sayHello(); // 호출 → "안녕하세요!" 출력
```

- 화살표 함수 선언 예

```
const sayHello = () => {
    console.log("안녕하세요!");
}
```

- 함수 선언식과 화살표 함수 차이
  - 함수 선언식 - 코드 순서와 상관 없이 어디서든 호출해서 사용 가능

```
sayHello(); // 정상 실행됨
function sayHello() {
 console.log("안녕하세요!");
}
```

  - 화살표 함수 - 순차적으로 실행해야 한다

```
sayHelloArrow(); // 에러
const sayHelloArrow = () => {
 console.log("안녕하세요!");
};
sayHelloArrow(); // 정상실행
```

템플릿 리터럴

- 일반 문자열 삽입 및 사용

```
let name ="alberto"
console.log("my name is " +name) //출력 결과 : my name is alberto
```

- 백틱(` (키보드 물결))

```
let name ="alberto"
console.log('my name is ${name}') //출력 결과 : my name is alberto
```

- 백틱 사용시 문자열 안에 변수를 바로 표현할 수 있다

디스트럭처링

- 객체 디스트럭처링

```
const user = {
        name : "alberto",
        age : 30
}
const { name, age } = user;
```

    - user 객체에 할당된 key를 변수로 지정 시 각 값이 저장된다
    - name = "alberto" , age = 30 저장된다
    - 만약 특정 key만 지정하면 해당 값만 들어온다. const { name } = user;
- 배열 디스트럭처링

```
const user = [ "Alberto", 30, "Seoul" ]
const [ name, age, addr ] = user;
```

    - 결과 값이 인덱스 순서대로 변수에 저장된다
    - name="alberto", age=30, addr="seoul"
    - const [name, age] = user;
    - 만약 변수가 부족하면 부족한 값을 뺀 나머지 값만 할당된다.
    - name = "Alberto", age=30, Seoul은 저장되지 않는다

스프레드 연산자

- 배열

```
const user = [ "Alberto", 30, "Seoul" ]
const newUser = [ "id", ...user ]
```

    - newUser에는 id을 포함한 user의 모든 값이 저장된다([id, alberto, 30, seoul])
- 객체

```
const user = {
        name : "alberto",
        age : 30
}
const newUser = { ...user }
```

    - newUser에는 user가 가지고 있는 모든 값이 저장된다

동기

- 코드가 위에서 아래로 차례대로 실행되는 것
- 앞의 코드가 끝날 때까지 뒤의 코드는 실행되지 않고 기다리고 있음

```
1.console.log( "서버에 접속합니다" )
2.console.log("서버에서 연산을 하고 있습니다. 3초 후 완료")
3.console.log("연산 결과를 출력합니다")
```

    - 동기 방식으로 처리된다고 하면 1번 실행 후 2번 실행, 3번 실행 된다. 이때 2번에

서 연산이 오래 걸리게 된다고 하면 연산이 끝날 때 까지 3번은 실행되지 않고 연산이 끝날 때 까지 기다린 후 실행된다

비동기

- 동기와 마찬가지로 코드는 순차적으로 아래로 실행된다. 하지만 연산이 오래 걸리는 값을 비동기 처리하게 되면 해당 코드를 기다리지 않고 다음 순서를 처리한다

  1.console.log("서버에 접속합니다")

  2.console.log("서버에서 연산을 하고 있습니다. 3초 후 완료")

  3.console.log("연산 결과를 출력합니다")

  - 2번을 비동기로 처리했다고 가정하면, 동기와 마찬가지로 1번 실행, 2번 실행, 3번 실행된다. 이때 2번은 개별적으로 실행되고 연산이 끝날 때까지 기다리지 않고 바로 3번이 실행된다. 그리고 나서 연산이 끝나게 되면 나중에 2번이 출력된다

    ```
    console.log("서버에 접속합니다");
    setTimeout(() => { // 비동기 처리
     console.log("서버에서 연산을 하고 있습니다. 3초 후 완료"); // 3초 후 실행
    }, 3000);
    console.log("연산 결과를 출력합니다");
    ```

  - 실행 결과
    - "서버에 접속합니다"
    - "연산 결과를 출력합니다"
    - "서버에서 연산을 하고 있습니다. 3초 후 완료" (3초 후 결과)

제네레이터

- 일반 함수와 달리 중간에 멈췄다가 다시 시작할 수 있는 함수
- Function* 키워드 선언
- Yield 키워드로 값을 순차적으로 내보낸다
- next()로 다음 값을 가져온다

```
function* myGenerator() {
  yield 1;
  yield 2;
  yield 3;
}
const gen = myGenerator();
console.log(gen.next()); // { value: 1, done: false }
console.log(gen.next()); // { value: 2, done: false }
console.log(gen.next()); // { value: 3, done: false }
console.log(gen.next()); // { value: undefined, done: true }
```

요약본

1. JavaScript 특징

- HTML/CSS만으로는 정적 페이지 → JS로 동적 기능 구현
- 버튼 클릭, 입력 처리, 애니메이션, 실시간 데이터 가능

- DOM을 통해 HTML/CSS 조작
- 브라우저(클라이언트), Node.js(서버) 모두 실행 가능

2. document & DOM

- **document** : 현재 웹페이지를 JS에서 접근할 수 있는 전역 객체
- **DOM** : HTML 문서를 트리 구조로 표현, JS가 요소를 읽고 수정 가능

3. 변수

- const: 값 변경 불가 (상수)
- let: 값 변경 가능 (변수)
- 작명 규칙: 숫자 시작 불가, 공백 불가, 의미 있는 이름, camelCase 권장

4. 자료형

- string, number, boolean, null, undefined
- **객체(Object)**: {key: value}
- **배열(Array)**: 순서(index)로 값 저장

5. 연산자

- **산술**: + - * / % **
- **할당**: += -= *= /= %=
- **비교**: == != === !== > < >= <=
- **논리**: && || !
- **삼항**: 조건 ? 참 : 거짓
- **타입 확인**: typeof, instanceof

6. 제어문

- **조건문**: if / else if / else
- **switch**
- **반복문**
  - for, for of(값 반복), for in(key 반복)
  - while, do while
- **제어**: break, continue

7. 함수

- **선언식**: 어디서든 호출 가능
- **화살표 함수**: 간결, 순서대로 실행
- **재사용**을 위해 코드 묶음

8. ES6+ 문법

- **템플릿 리터럴**: 백틱() → ${변수}`

- **디스트럭처링**: 객체/배열 값 분해 후 변수에 할당
- **스프레드 연산자**: ... → 배열/객체 복사 및 병합

9. 동기 vs 비동기

- **동기**: 순차 실행 (앞 코드 끝날 때까지 대기)
- **비동기**: 오래 걸리는 작업은 기다리지 않고 나중에 실행 (setTimeout, fetch 등)

10. 제네레이터

- function* 로 선언
- yield 로 값 순차 반환
- next() 호출 시 중단점부터 이어 실행