


섹션 1. JavaScript 기본

# ing	12
# goal	12
Σ 수식	✓

Vanilla - CodeSandbox

JavaScript example starter project

 <https://codesandbox.io/s/vanilla-vanilla>

```
console.log("텍스트"); // 텍스트 내의 내용을 콘솔에 출력  
// mac에서 백틱 입력: option + `
```

변수와 상수

자료형

형 변환

연산자

조건문

함수

객체

배열

반복문

배열 내장 함수

▼ 변수와 상수

1. 변수

```
let 변수 이름 = 변수 값; // 변수 중복 선언을 허용하지 않음  
var 변수 이름 = 변수 값; // 변수 중복 선언을 허용
```

- 변수명에는 _와 \$를 제외한 기호 사용 불가
- 변수명은 반드시 문자로 시작해야 함
 - age6 가능, 6age 불가

2. 상수

```
const 상수 이름 = 상수 값;
```

- 상수는 변수와 달리 값을 변경할 수 없음
- 선언할 때 무조건 초기화 필요

▼ 자료형

1. Primitive Type(원시 타입)

- 한번에 하나의 값만 가질 수 있음
- 하나의 고정된 저장 공간 이용

```
// 숫자형
let number = 12;
let inf = Infinity; // 무한대
let minusInf = -Infinity; // 음의 무한대
let nan = NaN;

// 문자열
let name = "eunjung";
let name2 = 'eunjung';
let name3 = `은정 ${name2}`; // eunjung 은정

// boolean
let isSwitchOff = false;

// NULL
let a = null;

// undefined
let variable; // 값 할당 x하면 자동으로 undefined 할당받음
```

2. Non-Primitive Type(비 원시 타입)

```
let array = [1, 2, 3, 4];
```

- 한번에 여러 개의 값을 가질 수 있음
- 여러 개의 고정되지 않은 동적 공간 사용

▼ 형 변환

```
let numberA = 12;
let numberB = "2";

// 묵시적 형 변환: js가 자동으로 자료형을 변환해 줌
console.log(numberA*numberB); // 24

// 명시적 형 변환(parseInt가 문자열을 받아 숫자형으로 변환해 줌)
console.log(numberA + parseInt(numberB)); // 24
```

▼ 연산자

```
// 연결 연산자
let a = "1";
let b = 2;
console.log(a+b); // 12(묵시적 형 변환)

// 비교 연산자
let compareA = 1 == "1"; // true(타입 비교 X)
let compareB = 1 === "1"; // false(타입 비교 O)

typeof 변수 이름; // 해당 변수의 type을 보여 줌

// NULL 병합 연산자
let a;
a = b ?? 10; // 양쪽 연산자 중 NULL이 아닌 것을 출력
console.log(a); // 10
```

▼ 조건문

```
switch(비교 대상){
  case 'ko':
    console.log("한국");
    break;
  case 'cn':
    console.log("중국");
    break;
  case 'jp':
    console.log("일본");
    break;
  default:
    console.log("미분류");
    break;
}

// 비교 대상을 각각의 case와 비교한 뒤 안의 내용을 출력
```

▼ 함수

```
// 기본 함수 구조
function 함수이름(){
  함수 내용
}
함수이름(); // 함수 호출

// 함수 표현식(함수를 이름 없이 만들어 변수에 담아 사용하는 것)
let hello = function(){
  return 리턴값;
};
```

```
const helloText = hello();

console.log(hello); // f hello() {}
console.log(helloText); // 리턴값

// 화살표 함수(표현식과 동일하게 기능)
let helloA = () => {
  return 리턴값;
}
let helloA = () => "리턴값";
```

- 함수 선언식으로 만들어진 함수는 위치 상관없이 사용이 가능(아래에 만들고 위에서 호출도 O)
- 함수 표현식은 불가

```
// 콜백 함수(함수의 파라미터로 함수를 넘김)
function checkMood(mood, goodCallback, badCallback){
  if(mood === "good"){
    // 기분 좋을 때 하는 행동
    gooeCallback();
  }
  else{
    // 기분 안 좋을 때 하는 행동
    badCallback();
  }
}

function cry(){
  console.log("ACTION :: CRY");
}
function sing(){
  console.log("ACTION :: SING");
}
function dance(){
  console.log("ACTION :: DANCE");
}

checkMood("sad", sing, cry);
// function cry()를 badCallback이라는 매개 변수에 담음
```

▼ 객체

```
/*1. 객체 생성자 이용*/
let person = new Object(); // new라는 키워드 사용

/*2. 객체 리터럴 방식*/
let person = {
  key: "value", // 이런 구조로 데이터 삽입(property)
  key1: "value2" // key에는 문자열만, value에는 어떤 자료형이 들어가도 상관 x
};

console.log(person); // {key: "value", key1: "value2"}
```

```
//점 표기법
console.log(person.key1); // value2
//괄호 표기법: 괄호 안에는 무조건 문자열 형태로, 다른 형태로 넣으려면 해당 이름의 변수 생성 필요
console.log(person["key1"]); // value2

/*key를 통해 value를 받아 오는 함수 생성*/
function getPropertyValue(key) {
    return person[key];
}
console.log(getPropertyValue("key")); // value
```

```
/*객체 수정하기*/
let person = {
    name: "조은정", // 함수가 아닌 property = 멤버
    age: 25,
    say: function(){ // 객체 안에 들어 있는 함수 = 메서드
        console.log(`안녕 나는 ${this["name"]}`); // 여기서 가리키는 this = 자기 자신(person)
    }
};

// property 추가
person.location = "한국";
person["gender"] = "여성";

// property 수정
person.name = "조은정A"; // name property의 값 수정됨
person["age"] = 50; // age property의 값 수정됨
// 객체를 const로 선언하였어도 수정에 오류 발생 x
// person이라는 상수 자체를 수정하는 행위가 아닌, 안의 Object를 수정하는 행위이기 때문
// person = {};로 새 객체를 할당하는 것이 상수 훼손

// property 삭제
delete person.age;
delete person["name"]; // delete 사용은 객체와 property와의 연결을 끊을 뿐, 메모리에서 삭제되진 않음

person.name = null; // 이렇게 해야 메모리에서도 삭제됨

// property의 존재 여부 확인
console.log(`name: ${"name" in person}`); // in 연산자를 통해 확인
```

▼ 배열

```
let arr = []; // 배열 리터럴
// 객체와 동일하게 배열 안에 무슨 자료형을 넣든 상관x

// 0부터 시작하는 인덱스 순서를 통해 배열 각각의 요소에 접근 가능
let arr = {1, 2, 3, 4, 5};
console.log(arr[0]); // 1

arr.push(추가할 원소); // 배열의 가장 마지막에 원소 추가
arr.length; // 배열의 길이(배열 자체도 객체라 동일한 방법으로 접근 가능)
```

▼ 반복문

```
for(초기식; 조건식; 연산식){
  // 반복 수행할 명령
}

let person = {
  name: "조은정",
  age: 25,
  tall: 165
};

const personKeys = Object.keys(person);

for(let i=0; i<personKeys.length; i++){
  console.log(personKeys[i]);
} // key 값만 출력
for(let i=0; i<personKeys.length; i++){
  const curKey = personKeys[i];
  const curValue = person[curKey];

  console.log(`${curKey}: ${curValue}`);
} // key: value 형태로 출력

const personValues = Object.values(person); // value 값을 받아 오는 함수
for(let i=0; i<personValues.length; i++){
  console.log(personValues[i]);
} // value 값만 출력
```

▼ 배열 내장 함수

1. forEach

```
const arr = {1, 2, 3, 4};

arr.forEach((elm) => console.log(elm));

//아래 함수와 동일
arr.forEach(function (elm){
  console.log(elm);
});
```

2. map

```
const arr = {1, 2, 3, 4};
const newArr = arr.map((elm)=>{
  return elm*2;
});
// 배열의 모든 요소에 대해 한 번씩 콜백 함수 실행
```

```
console.log(newArr);  
// [2, 4, 6, 8]
```

3. includes

```
const arr = {1, 2, 3, 4};  
  
let number=3;  
  
// 주어진 배열에서 전달받은 인자와 일치하는 값이 존재하는지 확인  
// === 연산 사용(자료형 다르면 false 출력  
console.log(arr.includes(number));  
  
// 아래 함수와 동일  
arr.forEach((elm) => {  
  if(elm === number){  
    console.log(true);  
  }  
});
```

4. indexOf

```
const arr = {1, 2, 3, 4};  
  
let number="3";  
  
console.log(arr.indexOf(number));  
// 주어진 값과 배열에 일치하는 값이 하나도 없다면 -1 반환  
// "3" 입력 -> -1 반환, 3 입력 -> 2 반환
```

5. findIndex

```
const arr=[  
  {color: "red"},  
  {color: "black"},  
  {color: "blue"},  
  {color: "green"}  
];  
  
let number=3;  
  
// 객체 배열에서 원하는 특성을 가지는 배열의 요소 인덱스 가져옴  
// 배열의 요소가 중복되는 경우 첫 번째로 만나는 값 리턴  
console.log(arr.findIndex((elm) => elm.color==="green")); // 3  
  
// 아래 함수와 동일  
console.log(  
  arr.findIndex((elm) => {  
    return elm.color==="green";  
  })  
);
```

6. find

```
const arr=[
  {color: "red"},
  {color: "black"},
  {color: "blue"},
  {color: "green"}
];

let number=3;

// index가 아닌 배열의 element 그대로를 반환
const element = arr.find((elm) => {
  return elm.color==="blue";
});

console.log(element); // color: "blue"
```

7. filter

```
const arr=[
  {num: 1, color: "red"},
  {num: 2, color: "black"},
  {num: 3, color: "blue"},
  {num: 4, color: "green"},
  {num: 5, color: "blue"}
];

// 전달한 콜백 함수가 true를 반환하는 모든 요소를 배열로 다시 반환
console.log(arr.filter((elm) => elm.color==="blue"));
```

8. slice: 배열 자르기

```
const arr=[
  {num: 1, color: "red"},
  {num: 2, color: "black"},
  {num: 3, color: "blue"},
  {num: 4, color: "green"},
  {num: 5, color: "blue"}
];

console.log(arr.slice(0, 2)); // [{num: 1, color: "red"}, {num: 2, color: "black"}]
// slice(begin, end)
// end - 1 값까지만 반환
```

9. concat: 배열 합치기


```
const arr=[
  {num: 1, color: "red"},
  {num: 2, color: "black"},
  {num: 3, color: "blue"}
];

const arr2=[
  {num: 4, color: "green"},
  {num: 5, color: "blue"}
];

console.log(arr1.concat(arr2));
```

10. sort: 배열 정렬

```
let char=["나", "다", "가"];

chars.sort(); // 원본 배열 자체를 정렬
console.log(chars); // "가", "나", "다"
```

```
let numbers=[0, 1, 3, 2, 10, 30, 20];

numbers.sort();
console.log(numbers); // 0, 1, 10, 2, 20, 3, 30
// sort를 그냥 사용하면 숫자를 문자열로 인식해 정렬함
```

```
let numbers=[0, 1, 3, 2, 10, 30, 20];

// 숫자를 작은 순서대로 정렬하려면 새로운 함수 필요
const compare=(a, b) => {
  // 1. 같다
  // 2. 크다
  // 3. 작다

  if(a>b){
    // 크다
    return 1;
  }

  if(a<b){
    // 작다
    return -1; // 음수가 나오면 a가 앞으로 감
  }

  // 같다
  return 0;
}

numbers.sort(compare);
console.log(numbers); // 0, 1, 2, 3, 10, 20, 30
```

11. join

```
const arr=["조은정", "님", "안녕하세요", "또오셨군요"];

console.log(arr.join());
// 조은정.님.안녕하세요.또오셨군요

console.log(arr.join(" "));
// 조은정 님 안녕하세요 또오셨군요

//join(여기 들어가는 게 간격에 들어가는 내용이 됨)
```