

# Javascript 기초 (변수 for if 문)

📎 자료	<u>Javascript</u>
☰ 구분	Javascript

1. `const` `let` `var`
2. data type
3. backtick
4. `if`, `else`, `===`, `>=`, `||`, `&&`
5. array 기초
6. object 기초

오늘은 javascript 기초 문법을 살펴 보는 시간으로  
변수/데이터타입/연산자/조건문/반복문 그리고  
라이브 진도보다 조금 더 보기 위해서  
배열과 객체도 조금만 살펴 보겠다.

배열과 객체는 내일 함수 관련해서 수업 진도를 나간 후  
조금 더 살펴 볼 예정이다.

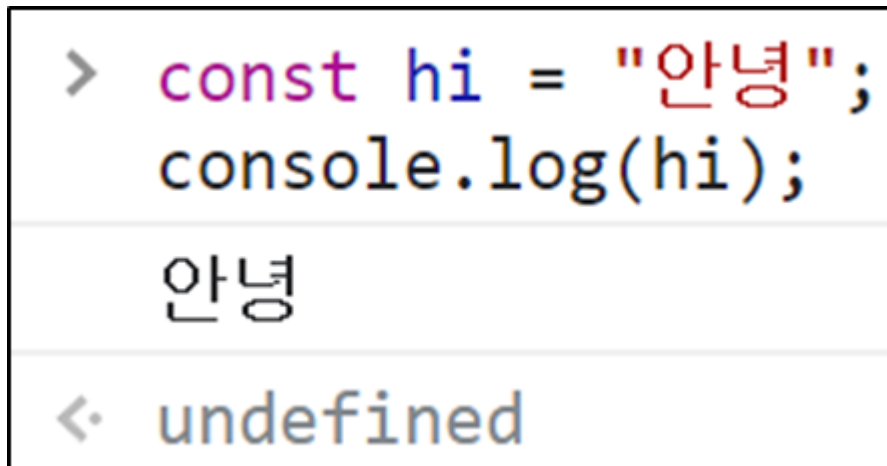
## 1. `const` `let` `var`

변수 선언법은 총 세 종류가 있다.

1. 8~90% 의 상황은 `const`
2. 일반 변수 변경할 일이 있거나 for문 작성시 `let`  
ex) flag, cnt ...
3. `var` 사용은 권장하지 않음 (사용 안 함)

```
const hi = "안녕";  
console.log(hi);
```

크롬창 열고, **F12** 눌러서 개발자도구 열고,  
콘솔 창 열어서 복사 붙여넣기로 실행



```
> const hi = "안녕";  
console.log(hi);  
안녕  
⏪ undefined
```

`console.log` 는 프린트문이다.

`const` 를 선언시 변수의 값을 변경 불가

```
const hi = "안녕";  
hi = "하이";  
console.log(hi); // 에러
```

그러나, 배열의 요소 또는 객체의 속성을 바꾸는 것은 가능하다! (추후 학습 예정 (뒤에나옴))

`let` 은 변수의 값 변경 가능

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

만약 변수 `i` 를 선언 시 `let` 이 아니라 `const` 라면 에러가 난다. 변수를 `const` 로 선언 시 값을 바꿀 수 없기 때문이다.

`var` 는 옛날 문법이며, 사용이 권장되지 않는다.

```
var hi = "하이";  
var hi = "안녕";  
console.log(hi); // 에러 아님
```

동일한 변수명으로 재 선언이 가능하기 때문에, 위험하다.

- `const` , `let` , `var` 같은 변수 선언 키워드를 안 붙여도 파이썬처럼 변수 선언 가능 하던데?

```
hi = "하이";  
console.log(hi);
```

절대 안된다. JavaScript 에서 작동이 되는는 문법 이지만, 원칙상 반드시 선언자를 붙이고 변수를 선언해야 한다.

## 2. data type

선언 시 타입은 붙이지 않지만, 타입은 존재한다.

number - 숫자 (정수 실수 구분없음)

`0` , `1` , `2` , `-1` , `-2.58` , ...

string - 문자열

`"a"` , `'a'` , `"hello"` , `"0"`

큰따옴표를 쓰던, 작은따옴표를 쓰던 똑같이 string 이며, 스타일 가이드에 따르거나 큰따옴표로 통일해서 사용 하는 것을 권장한다.

boolean - `true` , `false`

파이썬과는 다르게 소문자로 시작

array, object, function - 역시 자료형이나, 각각의 조금 이따가 따로 다룬다. (내일 수업내용 이긴 하나 미리 예습하는 것도 나쁘지 않다.)

```
const n1 = 0;
const n2 = 0.123;
const s1 = "a";
const s2 = "abc";
const b1 = true;
const b2 = false;

console.log(typeof n1); // number
console.log(typeof n2); // number
console.log(typeof s1); // string
console.log(typeof s2); // string
console.log(typeof b1); // boolean
console.log(typeof b2); // boolean
```

문자열에 대해서 javascript 문법을 추가로 조금만 더 보자.

```
let str = "aasdf";

const t=str.includes('n'); // include - 존재여부 확인시 사용
console.log(t)             // 있으면 T 없으면 F 반환

const t2=str.split('');    // s를 기준으로 가르기인데 없는문자면 통으로
console.log(t2)            // '' 넣으면 각각 한글자씩 배열에 담아 반환

const t3=str.replace('s','z');
console.log(t3)

const t4=str.replace('a','gg');
console.log(t4)

const t5=str.replaceAll('a','zxcv');
// 모든 a를 zxcv로 바꾸기
console.log(t5)

let str2="  d  abab  d  ";

let b=str2.trim(' ') // 앞뒤공백제거
let c=str2.replaceAll(' ','') // 모든공백제거
let d=str2.trimStart(' ') // 앞공백 제거
let e=str2.trimEnd(' ')   // 뒤 공백 제거

console.log(str2)
```

```
console.log(b)
console.log(c)
console.log(d)
console.log(e)
```

### 3. backtick

` ` 을 백틱이라고 한다. 키보드에선 1 왼쪽에 위치한다.

```
const myName = "jony";
const age = 28;

// 둘 다 jony는 28살
console.log(myName + "는 " + age + "살");
console.log(`${myName}는 ${age}살`);
```

백틱은 변수 뿐만 아니라 함수의 리턴값도 받을 수 있다.

```
const a = 1;
const b = 2;
const addNum = (n, m) => n + m;
console.log(`${a} + ${b} = ${addNum(a, b)}`);
```

(“⇒” 이것은 arrow function 으로 내일 함수 파트에서 살펴 볼 것이다. 지금은 pass)

백틱은 엔터, 탭을 사용시 그대로 적용해서 출력을 한다.

```
console.log(`
이름: jony
나이: 30
사는 곳: 경기도
`);
```

```
이름: jony
나이: 30
사는 곳: 경기도
```

## 4. `if` , `else` , `===` , `>=` , `||` , `&&`

C/C++ 언어와 완벽하게 동일하다.

그러나, 다른 것이 있는데, 비교연산자 `==` 가 아니라 `===` , `!=` 가 아니라 `!==` 사용한다.

`||` 는 or 연산자 이며

`&&` 는 그리고 를 의미하는 and 연산자 이다.

그리고, 조건을 `false` 로 인식하는 것은 다음과 같다.

`false` , `0` , `""` (빈 문자열) , `undefined` , `null`

그러나, 빈 배열 `[]` 과 빈 객체 `{}` 는 `true` 이다.

반드시 외워두자.

## 5. array 기초

배열은 대괄호 `[]` 를 사용한다.

```
const nums = [1, 2, 3, 4];
console.log(nums[0]); // 1
console.log(nums[4]); // undefined
console.log(nums.length); // 4
```

여기서, `undefined` 는 정의되지 않았다는 뜻이며, 에러가 아니다.

배열 선언 시, 배열의 크기는 선언하지 않는다.

`null` VS `undefined`

`null` : 개발자가 의도적으로 비워둔 값

`undefined` : 개발자의 의도와 상관없이 비워진 값. 흔히 에러라고 인식을 하나, 에러가 아니다.

`.length` 는 문자열 길이 구할때도 쓰인다.

```
const str = "총몇글자일까";  
console.log(str.length); // 6
```

배열의 각각을 요소 (엘리먼트 element) 라고 하며, 배열의 요소는 배열을 `const` 로 선언 하였어도 변경이 가능하다.

```
const nums = [1, 2, 3, 4];  
nums[1] = 6;  
console.log(nums); // [1, 6, 3, 4]
```

위와 같이, 배열 출력 시 반복문은 불필요하다.

배열의 요소를 추가할 땐 `push` 를 사용한다.

```
const nums = [1, 2, 3, 4];  
nums.push(5);  
console.log(nums); // [1, 2, 3, 4, 5]
```

그러나, `const` 로 선언된 배열을 다시 선언할 수는 없다

```
const nums = [1, 2, 3, 4];  
nums = [5, 6, 7, 8]; // error
```

이차원 배열도 가능하다.

```
const vect = [  
  [1, 2, 3, 4],  
  [5, 6, 7, 8],  
];  
  
console.log(vect);
```

JavaScript 에서 배열의 각 요소는 서로 다른 타입을 가지는 것을 허용하나, 개발자 입장에선 절대 추천하지 않는다.

```
const myInfo = ["Jony", 28]; // 권장되지않음
```

배열 관련 함수까지 몇 가지만 추가로 살펴보자.

```
const a=[1,2,3,4,5];

a.reverse();

a.push(6,7,8,[1,2,3]); // 맨 뒤 추가
console.log(a);

a.pop(); // 맨 뒤 제거
console.log(a);

a.unshift(9); // 맨 앞에 값 추가
console.log(a);

a.shift(); // 맨 앞 값 제거
console.log(a);

const result=a.indexOf(3); // 값 3이 있는 인덱스를 출력
console.log(result) // 없으면 -1 반환

const ret2=a.includes(3) // 원하는 값이 있으면 true 반환
console.log(ret2)

const ret=a.join('/')// 구분자(/)를 이용해서 문자열로 연결
console.log(ret)

//const ret=a.join('') 하면 배열값을 하나의 문자열로 연결가능

추가적인 연습은 함수를 학습 한 후에 조금 더 진행 해 보겠다.
```

## 6. object 기초

object 는 객체라고 하며, 중괄호 `{ }` 로 감싼 자료형이다. 파이썬의 딕셔너리와 비슷하나, 함수를 담을 수 있고, 함수를 담을 경우 메서드라고 부른다.

반드시 다음 문장을 암기하자.



객체(object)의 정확한 정의:

키(key) 와 값(value) 으로 이루어진 속성(property) 의 모음

예를 보자.

```
const myInfo = {  
  name: "kevin",  
  age: 39,  
  isMarried: true,  
  family: ["아빠", "엄마", "댕댕이"],  
};
```

객체는 순서가 존재하지 않는다. 즉, 위와 아래의 예시는 같은 코드이다.

```
const myInfo = {  
  family: ["아빠", "엄마", "댕댕이"],  
  isMarried: true,  
  name: "kevin",  
  age: 39,  
};
```

객체의 값은 인덱스가 존재하지 않기 때문에 키로 접근한다.

```
const myInfo = {  
  family: ["아빠", "엄마", "댕댕이"],  
  isMarried: true,  
  name: "kevin",  
  age: 39,  
};  
  
console.log(myInfo.family[0]); // 아빠  
console.log(myInfo.skills); // undefined
```

접근할 땐 마침표 `.` 로 접근하며, 없는 값에 접근하면 `undefined` 이다. 에러가 아니다.

객체의 프로퍼티는(속성) `const` 로 선언 되었어도 변경 가능하다.

```
const myInfo = {
  family: ["아빠", "엄마", "댕댕이"],
  isMarried: true,
  name: "kevin",
  age: 39,
};

myInfo.name = "kevin";
console.log(myInfo);
```

그러나, `const` 로 선언된 객체를 재선언을 할 수는 없다.

```
const myInfo = {
  name: "jony",
  age: 28,
  isMarried: true,
  family: ["아빠", "엄마", "멍멍이"],
};

myInfo = {
  name: "David",
  age: 30,
  isMarried: false,
  family: ["고양이"],
};
```

객체는 나중에 JSON Format 으로 서버와 통신할 때 사용되니 매우 중요하다.

객체에 대해서 조금만 더 살펴보자.

```
const friends=['kevin','bob','kate']
const age=[26,25,27]

// 객체문법1: key값을 동적으로 가능하다.
// 객체를 하나 만들어 보자.

let index=1

const school={
  friends:friends,
  age:age,
  [friends[index]]:age[index]
  // 객체의 key값을 동적으로 생성이 가능하다
}
```

```

console.log(school)

// 객체문법2: 속성명 축약

// key와 할당하는 변수의 이름이 같다면 축약 가능하다.
// 이를 <속성명 축약> 이라고 한다.

// 예를 살펴보자.

const friends=['kevin','bob','kate']
const age=[26,25,27]

const school_2={
    friends,
    age,
}
console.log(school_2)

// 많이 쓰는 객체관련 문법을 몇개만 더 살펴보자.

// 객체문법3 : destructuring

// destructuring 이란?
// 배열 또는 객체를 분해한 후
// 변수하나 만들어서 할당하는것을 말한다.

// 하나씩 살펴보자.

const user={
    name1:"minho",
    name2:"bbq",
    gender: "male",
}

// minho랑 male 출력해보자
console.log(user.name1,user.gender)

// 변수를 선언 후에 객체 value를 넣어서 출력해보자
const name1=user.name1;
const gender=user.gender;
console.log(name1,gender)

// 그런데 위와 같이 변수명과 객체의 키값이 같다면
// 아래와 같이 사용이 가능 하다.
const{name1}=user;
const{gender}=user;
console.log(name1,gender)

// 그리고 아래와 같이 묶어서 쓸 수도 있다
const{name1,gender}=user;

console.log(name1,gender)

// =====

```

```

// 객체문법4 : spread

const user=['kevin','bob','kate']
console.log(...user)

// spread syntax를 배열에 사용 할 수 있고
// 객체에도 사용 가능하다!

const ban={
  name3:"kfc",
  name4:"mc",
}

const user={
  name1:"Minho",
  name2:"bbq",

  //ban:ban,  // 통으로 들어감
  ...ban
}

console.log(user)

// json 객체관련 문법도 살펴보자. 그 중에서도
// 자바스크립트 객체를 제이슨 형식으로 변환하는 것을 보겠다.

// json 은 자바스크립트와 다르게
// key에는 반드시 따옴표를 넣는데.. 특히 쌍따옴표를 넣어야 한다!

// 그래서 자바스크립트 객체를 제이슨 형식으로 변환할때
// key 값에 쌍따옴표 들어간다.

// 이때 JSON.stringify 를 사용해서
// javascript 객체를 string화 시킨다. (json 형식으로 변환)

const user={
  name1:"minho",
  name2:"bbq",
  gender: "male",
  test:[1,2,3,4],
}

const Obj_to_Json=JSON.stringify(user)
console.log(Obj_to_Json)
console.log(typeof Obj_to_Json)  // 타입이 string

// 이번에는 json파일을 반대로 객체화 시켜보자.

const Json_to_Obj=JSON.parse(Obj_to_Json)
console.log(Json_to_Obj)
console.log(typeof Json_to_Obj)

// 객체문법5: 메서드명 축약 ( 객체 내 메서드에는 function 을 안써도 된다)

```

```
// 객체의 추가적인 내용은 함수를 공부하고 조금 더 살펴 볼 것이다.  
// 오늘은 여기까지 만 하자 :)
```