


자바스크립트 중급

https://www.youtube.com/watch?v=4_WLS9Lj6n4&list=LL&index=1

▼ 변수

- var - 한번 선언된 변수 다시 선언 가능 / let과 동일하게 변하는 값 선언 / 선언 전에 사용 가능

```
console.log(name); // undefined  
var name = 'Mike';
```



```
var name;  
console.log(name); // undefined  
name = 'Mike';
```

호이스팅 됨, 쓸 순 있으나 mike 할당은 호이스팅 안돼서 undefined 뜸

- let - 한번 선언되면 사용 불가 / 위에서 let은 안됨
- const - 선언하면서 무조건 할당해줘야

TDZ - 할당 전엔 사용 못하게 만듦, 할당 전 사용한 부분임

변수 생성과정

1. 선언
2. 초기화
3. 할당

var 1 2 3 / let 1+2 3 / const 1+2+3

호이스팅 - 스코프 내에서 어디든 변수 선언은 최상위에 선언된 것 처럼 행동

if나 for문의 스코프에서

var는 내부에서 선언했으면 밖에서도 가능, let const는 불가

var는 함수 내부에서까지 됨

결국 var 대신 let const를 사용하자!

▼ 생성자 함수

이전 선언한게 객체 리터럴

비슷한 객체를 여러개 만들어야할때 생성자 함수

```
      첫 글자는 대문자로
function User(name, age){
  this.name = name;
  this.age = age;
}

let user1 = new User('Mike', 30);
let user2 = new User('Jane', 22);
let user3 = new User('Tom', 17);
      new 연산자를 사용해서 호출
```

붕어방 틀이라 생각하기

생성자 함수는 new를 붙여야

```
function Item(title, price) {
  //this = {};
  this.title = title;
  this.price = price;
  this.showPrice = function(){
    console.log(`가격은 ${price}원`);
  }

  //return this;
}

const item1 = new Item("t11", 200);
const item2 = new Item("t2", 300);

console.log(item1, item2);

item1.showPrice();
```

▼ 객체 메소드, computed property



```
let a = 'age';

const user = {
  name: 'Mike',
  [a]: 30 // age : 30
}
```

computed property (계산된 프로퍼티)

computed property = 이미 할당된 변수 / key에서 []로 감싸주는 특징

[여기서 계산도 가능] / 어떻게 여기에 키가 될지 모를 때 유용

- Object.assign()

```
const cloneUser = user; //안됨, 메모리 주소만 복사된거임
//이러면 user값도 바뀌어지는 문제점

const newUser = Object.assign({}, user); //빈 객체에 원 객체인 user가 들어감
//{name = "mike"} 나 {새로운 키 = "11"} 가능
//name은 덮어씌워지고 새로운 키는 추가
```

- keys

```
//키를 배열 반환
const user = {
  name : "mike",
  age : 30
}

Object.keys(user);
//["name", "age"];
```

- values

똑같이 값만 반환

- entries - 객체의 키와 값을 배열 쌍으로 반환

```
[
  ["name", "mike"],
  ["age", 30]
]
```

- fromEntries - 동일하게 배열 → 객체로 반환

▼ 심볼(자료형임) = 유일한 식별자 - 숨기고 싶을 때 인거같음

유일한 프로퍼티를 사용하고 싶을 때 사용해라

```
const a = Symbol();
```

b 해도 a==b도 false

유일성 보장! 전체 코드 중 하나만 됨 Symbol('id1') 가능 - 설명임

- symbol.for() - 전역심볼

```
const id1 = Symbol.for('id');
const id2 = Symbol.for('id');

id1 === id2; // true
```

for 없을 땐 false 였던거, 하나 생성한 뒤 같은 키를 통해 같은 symbol 공유

```
// 다른 개발자가 만들어 놓은 객체
const user = {
  name: "Mike",
  age: 30,
};

// 내가 작업
// user.showName = function () {};
const showName = Symbol("show name");
user[showName] = function () {
  console.log(this.name);
};

user[showName]();

// 사용자가 접속하면 보는 메세지
for (let key in user) {
  console.log(`His ${key} is ${user[key]}.`);
}
```

//원래 맨 아래 블럭에서 객체에 들어있는 메소드도 한꺼번에 보여지는 기이한 현상 있었는데 심볼 이용해서 메소드를 만들면 안보여지고 user[showName]() 호출 시 mike도 잘 나옴

왜냐면 - 심볼은 object.key나 for in 구문에서 안보이기 때문

▼ Math method

num.toString() - 10진수 → string

num.toString(2) → 2진수

Math.PI

.ceil() 올림

.floor() 내림

round() 반올림

toFixed() 소수점 자리수 컨트롤 - .toFixed(2) - 셋째 자리에서 반올림

0이면 정수만 나옴, 100처럼 커지면 0으로 채워짐

isNaN() 난값인지 체크하는 유일한 함수

parseInt()

parseFloat()

math.random() 0~1

math — max, min, abs, pow(n,m), sqrt

▼ 문자열

- ‘ “ 큰 차이 없음
- ` (~ 위에 있는 따옴표) = 백틱
 - `${name} = mike / ${2+3} = 5` 가능
 - 여러줄이 가능 엔터쳐서, 그냥 " 하려면 \n 해야
- 문자열.length
- 문자열[2] - 배열처럼 가능, 대신 배열처럼 한글자씩 바꾸는건 불가능
- touppercase
- indexOf - 찾는 문자없으면 -1 / 첫번째 문자
 - >-1
 - includes ⇒ true or false 반환
- slice(n,m) n부터 m까지 부분문자열 반환
- substring(n,m) n과 m 바뀌도 동작, 음수는 0으로 인식
- substr(n,m) n부터 시작해서 m개를 가져옴
- trim() 앞뒤 공백 제거
- repeat(n) 문자열 n번 반복
- 문자열 비교 - 아스키 코드 "a" < "c"

- 문자열.codePointAt(0) = 아스키코드

▼ array

- push - 뒤에 삽입
- pop - 뒤에 삭제
- unshift - 앞에 삽입 = queue 인듯
- shift - 앞에 삭제
- splice(n,m) 특정 요소 지움, n부터 m이 개수
 - (1,3,100,200) 2 3 4번째 지우고 100 200 이라는 값 2개만 넣어 대체
 - (1,0, 넣을것들) 뭐 삭제 안하고 넣을것들 추가됨
 - 삭제된 요소 리턴함
- slice 반환
slice()는 복사 의미
- concat
 - 기존배열.concat([1,2],[3,4],5) - 그대로 하나의 배열 리턴
- arr.forEach((item, index) => {});
배열 중 배열값과 인덱스 사용하게 해주는 듯

```
let arr = ["Mike", "Tom", "Jane"];

arr.forEach((name, index) => {
  console.log(`${index + 1}. ${name}`);
});
```

- indexOf(3,3) - 3 인덱스를 찾되 여러 3중에 index가 3 이상인거 부터 찾아
- lastIndexOf - 끝에서부터 탐색
- includes 포함하는지
- find(함수) - index함수와 비슷하지만 함수 인자를 통해 더 복잡한 결과 도출 가능
→ 결과 2임

```
let arr = [1, 2, 3, 4, 5];

const result = arr.find((item) => {
  return item % 2 === 0;
});

console.log(result);
```

- `indexOf()=>{ }`; 함수가 인자면 이런 식임
- `filter(함수)`
 - `find`는 하나만 쫓던거와 달리 만족하는 모든 요소를 배열로 반환
 - 위 사진에서 `filter` 하면 `[2,4]` 리턴
- `reverse()` 역순으로 재정렬
- `map(함수)`
- `join("-")` 배열 합쳐서 문자열로 만듦, 중간엔 -가 들어감, 디폴트는 ,
- `split()` 반대로 문자열 → 배열, 인자 기준으로 나눔
- `isArray()` 배열인지 아닌지, 객체인지
 - `typeof` 치면 둘다 객체로 나와서 구분안됨
- `sort` - 재정렬, 알파벳 순
 - 숫자 순으로 하려면
 - `arr.sort((a,b) => { return a-b });`; - 내부 로직은 b가 작으면 양수여서 앞으로 오게되는..
- `_.sortBy(arr)`
 - 위처럼 `sort`하면 복잡하니 `Lodash`라는 것을 이용하여 로직 상관없이 편히 소팅해주는 거
- `for of`, `forEach`, `map`, `reduce`


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

const newMyArr = myArr.forEach((currentElement, index, array) => {
  console.log(`요소: ${currentElement}`);
  console.log(`index: ${index}`);
  console.log(array);
});

console.log(newMyArr); // undefined
```

map이 이런 형식이고 forEach는 newMyArr없이 그냥 찍으면 출력됨
 reduce() 는 이걸 한번에 해준다함
 (누적 계산값, 현재 값)

▼ 구조 분해 할당

- let [x,y] = [1,2]
- 기본값 - 해당 값이 없으면 undefined라서 let[x=1, y=2] 로 하곤 함
- 바꿔치기 - temp 써야했지만
[a,b] = [b,a] 로 쉽게 가능
- 객체도 구조 분해 가능
객체의 키 값을 그냥 프로퍼티로 쓸 수 있음
기본값 줄 수 있음

▼ 나머지 매개변수, 전개 구문