자바스크립트 기초

2021.12.14

목차

■ 변수

예약어로는 변수를 선언할 수 없다. 이미 **지정된 값**이기 때문이다.

alert()	경고창을 띄우는 함수
console.log()	콘솔창에 로그를 띄우는 함수

```
name = "명찬";
age = "27";

console.log(age);

--------------------
30
```

위와 같은 함수는 좋은 함수 사용방법이 아니다.

변수명이 유일하다는 보장이 없기 때문이다.

따라서 **let** 과 **const** 를 사용한다.

let	바뀔 수 있는 함수
const	절때로 바뀌지 않는 상소. 수정하려 할 시 에러가 난다. 대문자 사용을 권장한다.

```
const PI = 3.14;
const SPEED_LIMIT = 60;
const BIRTH_NUMBER = '19950408';
```

■ 정리

자바스크립트에서 변수를 선언할 때는 **변하지 않는 값을 const** 으로,

변할 수 있는 값은 let 으로 선언한다.

왜냐하면 의도치 않은 동작을 방지할 수 있기 때문이다.

let변수상 내용이 중복되면 최종 내용으로 간다.

> 💡 모든 변수를 const로 선언하고 변경될 여지가 있는 변수들만 나중에 let으로 변경한다.

■ 변수명을 정할 대 주의사항

▼ 변수는 문자와 숫자, $ 와 _ 만 사용할 수 있다.

const **MY_NAME** = "..."; **O**

let _ = 1; **O**

let **$** = 3; **O**

▼ 첫글자는 숫자가 될 수 없다.

let 1stGrade = 'a='; **X**

▼ 예약어는 사용할 수 없다.

let let = 99; **X**

▼ 가급적 상수는 대문자로 알려준다.

const **MAX_SIZE** = 9; **O**

▼ 변수명은 읽기 쉽고 이해할 수 있게 선언한다.

let **a** = 1; **X**

let **userNumber** = 3; **O**

▼ 2. 자료형

■ 문자형

```
const name = "조명찬"; //문자형 string
const a = "나는";
const b = "이다";
const age = 27;

const message = "I'm boy.";
```

```
const message2 = "I|'m boy"; // 작은 따옴표를 써야겠다 할 때 사용한다.
const message3 = `내 이름은 ${name}이다`; // 내 이름은 조명찬이다.
// 달러를 사용할때는 큰 따옴표를 사용하지 않는다.
const message4 = `나는 ${26+1}살 이다.`; //나는 27살이다.

console.log(a+name+b); // 나는 조명찬이다.
```

■ 숫자형

숫자형 String	const PI = 3.14;
더하기	console.log(1+2)
빼기	console.log(10-3)
곱하기	console.log(3*2)
나누기	console.log(6/3)
나머지	console.log(6%4)

```
const x = 1/0 //infinity
const y = name/2; //NaN - Number a Number이란 의미로 숫자와 관련된 작업때 염두해 두자.
```

■ 불린형

```
const a = true;
const b = false;

const name = "조명찬";
const age = "27";

console.log(name == '최명찬'); //false
console.log(age > 20); //true
```

■ typeof 연산자 (변수의 자료형을 알아볼 수 있다.)

```
console.log(typeof 3); //숫자형 Number
console.log(typeof name); //문자형 String
console.log(typeof true); //불린형 Boolean
console.log(typeof "xx"); //문자형 String
console.log(typeof null); //object
console.log(typeof undefined); //undefined
```

변수를 사용하는 개발자가 직접 하면 typeof 를 쓸일이 많이 없다.

하지만 다른사람이 작성한 변수의 타입을 알아야 하거나 API 통신 등을 통해 받아온 데이터를 타입에 따라 다르게 처리해야 할 때 typeof 를 사용하게 된다.

> 💡 typeof null; //object 가 나왔지만 null은 객체가 아니다. 초기 자바스크립트 오류이다.

null 은 존재하지 않는 값을 의미하며, undefined는 값이 할당되지 않음을 의미한다.

▼ 3. alert, prompt, confirm (대화상자)

사용자와 상호작용을 할 수 있는 대화상자이다.

■ alert

무언가를 알려준다. 메세지를 띄우고 확인버튼을 누르기 전까지 계속 떠 있는다. 일방적 알림

```
const name = prompt("이름은?"); //이름은? 하는 질문창이 나온다. 조명찬을 입력해보자.
alert("환영한다." + name + "님"); // 환영한다. 조명찬님 이라는 alert 알림창이 나온다.
alert(`환영한다, ${name}님`); //``를 이용한 간편한 방법

// 만약 창에서 취소를 누르게 되면 null(내용없음)이 출력된다.
```

■ prompt

사용자에게 값을 입력받는다. default 값을 입력 가능하다.

```
const name = "prompt("예약일을 입력해" , "2021-12-)";
// 예약일을 입력해라는 질문과 함께 답장란에 2021-12- 이 출력되어 있다.
```

■ confirm

확인을 받는다. 결제하시겠습니까? , 삭제하시겠습니까? 등에서 자유 사용한다.

```
const isAdult = confirm("당신은 성인입니까?");
console.log(isAdult); // '네' 를 누르면 true,  '취소' 를 누르면 false 이다.
```

■정리

기본제공이라 간단히 사용할 수 있다는 장점이 있다.

하지만 **단점**도 명확히 존재한다.

1. 창이 떠있는 동안 스크립트가 일시정지 된다. 뜰 때마다 상호 동작을 받아 창을 닫아야 한다.
2. 스타일링이 불가해 위치와 모양을 정할 수 없다. 때문에 html, css를 이용해 만들기도 한다.

그럼에도 불구하고 기본제공이라 빠르고 쉽게 적용가능하여 많은 부분에서 사용되고 있다.

▼ 4. 형변환

String()	문자형으로 변환한다.
Number()	숫자형으로 변환한다. Number("문자") // NaN
Boolean()	불린형으로 변환한다. // 0 , '' , null , undefined , NaN 이 5개가 false로 반환 된다.

■ 형변환이 필요한 이유

문자형과 문자형을 더하면 이어서 보여준다.

"hello" + "명찬" = "hello 명찬"

숫자와 숫자를 더하면 두 수의 합을 보여준다.

100 + 100 = 200

만약 자료형이 다르면 의도치 않은 동작이 발생할 수 있다.

"100" + 100 = ???

```
const mathScore = prompt("수학 몇점?);
const histScore = prompt("역사 몇점?);

const result = (mathScore + histScore) / 2; //괄호를 묶는 이유는 우선권이 있기 때문!

console.log(result); //점수는 각각 90, 80 을 입력해본다.

-----------------------------------------------------
4050... ???? 왜 이런 값이 ....
```

원인은 prompt 입력의 기본값이 항상 문자형인데서 온다.

입력한 숫자는 사실 문자형으로 입력된 것이기에 90 + 80 = 170 이 아닌 9080 이 된 것이다.

그런데 나누기는 왜 된것일까?

문자형으로 숫자가 나와도 자동적으로 나누기가 된다. 이를 자동형변환 이라 한다.

자동이라 편할 것 같지만 때때로 원인을 찾기 힘든 상황을 만들어 낸다.

이를 보완하기 위해 명시적형변환(앞글자는 항상 대문자를 쓴다.)을 사용한다.

■ String

괄호안의 타입을 문자로 바꾸어 준다.

```
console.log(
  String(3), //3
  String(true), //true
  String(false), //false
  String(null), //null
  String(undefined) //undefined
};
```

■ Number

괄호안의 타입을 숫자로 바꾸어 준다.

```
console.log(
  Number("1234"), //1234
  Number("1234aaaa"), //NaN
  Number(true), //1
  Number(false), //0
}
```

■Boolean

false 만 기억하면 된다. 나머지느 전부 true 가 되기 때문이다.

false 가 되는 경우

숫자 0 , 빈분자열 '' , null , undefined , NaN

```
// true 가 되는 경우
```

```
console.log(
  Boolean(1),
  Boolean(123),
  Boolean("javascript")
}

-------------------------------------------------

//false 가 되는 경우

console.log(
  Boolean(0),
  Boolean(""),
  Boolean(null),
  Boolean(undefined),
  Boolean(NaN)
}
```

■ 정리

String()	문자형으로 변환
Number()	숫자형으로 변환 Number("문자") // NaN
Boolean()	불린형으로 변환 (0 , '' , null , undefined , NaN) = false

💡 버그 없는 코드를 만들기 위해서는 null 과 undefined 를 잘 파악하고 있어야 한다.

Number(null) = 0

Number(undefined) = NaN

이 된다. 외웁니다....

사용자에게 나이를 입력 받을 때 취소를 누르면 그냥 0이 된다.

```
Number(0) //flase
Number('0') // true

Number('') //flase
Number(' ') //true
```

▼ 5. 기본 연산자

■ 기본적인 연산

+	-	/	%
더하기	빼기	곱하기	나머지

나머지(%)를 어디에 사용할까?

나머지 값은 생각보다 유용하며 자주 사용되는 연산자이다.

1.홀수인지 짝수인지를 구분해야 할때

홀수: X % 2 = 1

짝수: Y % 2 = 0

2.어떤 값이 들어와도 5를 넘기면 안 될 경우

X % 5 = 0~4 사이의 값만 변환한다.

■ 거듭제곱(**)

```
const num = 2**3;
console.log(num); //8 (2의3승)
```

■ 우선순위

(* 과 /) > (+ 와 -)

■ 줄여서 쓸 수 있는 연산자

```
let num = 10;
num = num + 5;

//위의 식을 줄여서 표현한다면?

num += 5;
console.log(num);

// += , -= , *= , /= 사용 가능하다.
```

■ ++ , - - (증가연산자와 감소연산자)

값을 1만큼 증가시키거나 1만큼 감소시킨다.

```
let num = 10;
let++; //11
let--; //9

console.log(num);
```

++ , - - 는 앞에쓰는 지 뒤에 쓰는지에 따른 차이가 있다.

```
let num = 10;
let result = num++;

console.log(result); //10이 나온다.
```

뒤에 ++, - -를 입력하면 증가시키기 전의 값을 상수로 잡는다.

그래서 값이 변경되지 않는 것이다.

앞에 ++, - -를 입력하면 증가시킨 값을 함수로 잡는다.

▼ 6. 비교 연산자, 조건문 (if, else

<	>	≤	≥	≠
<	>	<=	>=	!=

비교 연산자의 값은 항상 true or false 값만 반환한다.

```
console.log(10 > 5); //true
console.log(10 == 5); //flase
console.log(10 != 5); //true
```

== 를 동등연산자라 하는데 이 연산자가 오류가 날 때가 있다.

```
const a = 1;
const b = "1";

console.log(a == b); //true

//숫자형과 문자형을 비교한 것인데 같다고 나와버린다.
```

=== 는 일치연산자라 하며 타입(type)까지 비교를 한다. 버그 방지를 위해 사용한다.

```
const a = 1;
const b = "1";

console.log(a === b); //false
```

■ 조건문

어떤 조건에 따라 이후 행동을 다르게 만들어주는 역할을 한다.

if 문

괄호 안 조건이 트루면 실행한다.

```
const age = 27;

if(age > 19){
  console.log('hello')
}

if(age <= 19){
  console.log('bye')
}
```

---

else 문

if 조건이 false 일 때 실행한다.

```
const age = 27;

if(age > 19){
  console.log('hello')
} else {
  console.log('bye')
}
```

---

else if 문

또 다른 조건이나 같은 경우에 사용한다.

```
const age = 19;

if(age > 19){
  console.log('hello')
} else if(age === 19){
```

```
  console.log('oh~! 19 !!')
} else {
  console.log('bye')
}

//쟈기 안녕? 안녕쟈기? 잘하고있어? 보여?
```

▼ **7. 논리 연산자 (AND, OR, NOT)**

|| (or)	&& (and)	! (not)
여러개 중 하나라도 true 이면 true 이다. 모든 값이 false 일때만 false 를 반환하며 a 와 b 중 true 가 있으면 true 이다.	모든 값이 true 일때만 true 이다. 하나라도 false 이면 false 를 반환한다. a 와 b 둘다 true 이면 true 이다.	true 와 false 를 반대값으로 바꾼다. true 이면 false 로 바꾼다. a가 false 이면 true 이다.

■ 예시

스티브잡스가 **한국인 이거나(or)** , 남자이다. = true

스티브잡스는 **한국인 이고(and)** , 남자이다. = false

■ 평가를 하는 경우

OR

첫번째 true 를 발견하는 즉시 평가를 멈춘다.

스티브잡스는 남자 **이거나**(or) , 한국인 **이거나**, 군인 **이거나**.....등

아무리 길어도 **남자라는 정보가 true 이기에 true 로 평가**한다.

```
// OR : 이름이 조명찬 이거나, 성인이면 통과하는 경우

const name = "조찬명";
const age = 27;

if(name == '조명찬' || age > 19){
  console.log('통과')
}

----------------------------------------
통과한다. 이름은 다르지만 성인이기 때문이다.
```

AND

첫번째 false 를 발견하는 즉시 평가를 멈춘다.

스티브잡스는 남자 이고(and) , 한국인 이며, 미국인인 동시에 ... 등

아무리 뒤에 참이 있어도 한국이닝 아니기에 false로 평가한다.

처음 발견된 한국인 까지만 평가하고 뒤는 무시해버린다.

> 💡 효과적으로 사용하기 위해서는 아래와 같이 적용한다.

운전면허가 있고(전체 군인의 80%) ,

시력이 좋은(천체 군인의 60%)

여군(전체 군인의 7%)

이와 같은 경우 여군인데, 시력이 좋고, 운전면허가 있는 사람으로 코딩을 하는 것이 효과적이다. 첫번째 평가에서 93%를 걸러내기 때문이다.

이를 통해 성능최적화를 이끌어 낼 수 있다.

```
// AND : 이름이 조명찬 이고, 성인이면 통과하는 경우

const name = "조명찬";
const age = 27;

if(name == '조명찬' && age > 19){
  console.log('통과')
} else {
  console.log('돌아가')
}
-----------------------------------------
돌아가. (성인이 아니기 때문이다.)
```

NOT

나이를 입력받아 성인이 아니면 돌아가라.

```
const age = prompt('나이가?');
const isAdult = age > 19;

if(!isAdult){
  console.log('돌아가')
};
-------------------------------------
돌아가. (prompt에서 10을 입력햇을 경우)
```

우선순위

코드는 순서대로 실행해서 상황에 따라서 원하지 않았던 결과가 나올 수 있다. 이를 괄호를 통해 해결할 수 있다.

1. 괄호를 사용하지 않은 경우

```
const gender = "F";
const name = '조명찬';
const isAdult = true;

if(gender === 'M' && name === '조명찬' || isAdult){
  console.log('통과')
} else {
  console.log('돌아가')
}

----------------------------------------
통과 (우선순위가 있어 통과를 하는 것이다.)
```

2. 괄호를 사용한 경우

```
const gender = "F"
const name = '조명찬';
const isAdult = true;

if(gender === 'M' && (name === '조명찬' || isAdult)){
  console.log('통과')
} else {
  console.log('돌아가')
}

-----------------------------------------
돌아가 (괄호를 사용하여 정확하게 표시)
```

▼ 8. 반복문 (for, while, do while)

반복문(loop)은 동일한 작업을 여러번 반복할 때 사용한다.

1 부터 10 까지 로그를 찍어야 할때 console.log(1); .... console.log(10)

으로 적을 수 있다.

하지만...

1부터 10000까지 로그를 찍어야 한다면?

이럴때 반복문을 사용한다. for 을 가장 많이 사용한다.

for

세미콜론으로 구분하며 세 부분으로 나눌 수 있다.

```
for(let i = 0 //초기값
        i < 10; //조건. false가되면 멈춘다.
        i++ //코드 실행 후 작업
){
  console.log(i+1);
}

---------------------------------------
1,2,3,4,5,6,7,8,9,10
```

---

while

```
let i = 0;
while (i < 10){
  console.log(i);
  i++; //코드를 지정해주지 않으면 부한반복으로 서버가 뻗는다.
}

-----------------------------------------
0,1,2,3,4,5,6,7,8,9
```

---

do while

while 과 비슷하지만 조건을 아래로 옮길 수 있다.

적어도 한번은 실행한다는 차이점을 가지고 있다.

```
let i = 0;
do{
  //코드
  i++;
} while (i < 10)

//do 를 먼저 실행하고 while 를 실행한다.
```

■ 반목문을 빠져나가는 기능

break

만나는 순간 코드 실행을 멈추고 해당 반복문을 빠져 나온다.

```
while(true){ //while(true) 는 무한반복을 의미한다.
  let answer = confirm('계속할까요?');
if(!answer){
  break;
  }
}
----------------------------------------
확인을 누르면 창이 계속 뜨며 취소를 누르면 닫힌다.
```

continue

코드 실행을 멈추는 것 까지는 동일하지만 빠져나오지 않고

다음 반복으로 점프한다.

```
for(let i ==0; i < 10; i++){
  if(i % 2){
    continue;
  }
  console.log(i);
}
----------------------------------------
0,2,4,6,8
```

▼ 9. switch문

모든 문장은 사실 if else 로도 사용 가능하다.

하지만 케이스가 다양할 때 가결히 쓸 수 있다는 장점이 있다.

아래의 코드는 서로 같은 코드이다.

```
switdh(평가){
  case A;
  //A 일때 코드 들어가는 부분
  case B;
  //B 일때 코드 들어가는 부분
}
```

```
if(평가 == A){
  //A 일때 코드 들어가는 부분
} else if(평가 == B){
```

```
  //B 일때 코드 들어가는 부분
}
```

■ 예제

사고싶은 과일을 물어보고 가격 알려주기

사과: 100원/ 바나나: 200원/ 키위: 300원/ 멜론: 500원/ 수박: 500원

```
let fruit = prompt('무슨 과일을 사고 싶나요?_사과,바나나,키위,멜론,수박');

switch(fruit){
  case '사과':
    console.log('100원 입니다.');
    break;
  case '바나나':
    console.log('200원 입니다.');
    break;
  case '키위':
    console.log('300원 입니다.');
    break;
  case '멜론':
  case '수박':
    console.log('500원 입니다.');
    break;
   default :
    console.log('그런 과일은 없습니다.');
}
```

▼ 10. 함수 (function)의 기초

한번에 한 작업에만 집중한다.

읽기 쉽고 어떤 동작인지 알 수 있게 이름을 짓는 것이 중요하다.

showError	에러를 보여준다.
getName	이름을 얻어온다.
createUserData	유저데이터를 생성한다.
checkLogin	로그인 여부를 체크한다.

■ 장점

서비스를 만들 때 닫거나 비슷한 동작이 생길 수 있다.

자주 사용하거나 비슷한게 많으면 하나를 만들어

여러군데에 사용하는 것이 좋다.

```
// 기본 공식
function[함수] sayHello[함수명] (name[매개변수]){
  console.log(`Hello, ${name}`);
}

sayHello('Mike'); 로 호출 가능하다.

-------------------------------------------------

// 적용해보기
function showError(){
  alert('에러가 발생했습니다. 다시 시도해주세요.');
}

showError()
```

💡 적재적소에 사용한게 되면 그만큼 유지보수가 쉬워진다

■ 매개변수가 있는 함수

다양한 대응을 가능하게 한다.

■ 일반적으로 이름을 입력했을 경우

```
function sayHello(name){
  const msg = `Hello, ${name}`;
  console.log(msg);
}

sayHello(`mike`);
sayHello(`Tom`);

-------------------------------------
"Hello, Mike"
"Hello, Tom"
```

■ 이름을 입력하지 않았을 경우

```
function sayHello(name){
  let msg = `Hello`;
  if(name){
    msg += ` , ${name}`;
    }
```

```
    console.log(msg);
}

sayHello();

----------------------------
"Hello"
```

전역변수(global varable)	어디서나 접근할 수 있는 함수
지역변수(locak varable)	함수 내부에서만 접근할 수 있는 함수

■ or

```
function sayHello(name){
  let newName = name || 'friend';
  let msg += `Hello, ${newName}`
  console.log(msg)
}

sayHello();
sayHello('jane');

-------------------------------------
“Hello, friend”
“Hello, jane”
```

■ default

name 이 없을 때 할당

```
function sayHello(name = 'friend'){
  let msg = `Hello, ${name}`
  console.log(msg)
}

sayHello();
sayHello('jane');

-----------------------------------------
“Hello, friend”
“Hello, jane”
```

■ return

오른쪽 값 반환 후 종료

```
function showError(){
  alert('에러발생!');
  return;
  alert('이 코드는 절대 실행 불가다!');
}

const result = showError();
console.log(result);

----------------------------------------
“에러발생!”
undefined
```

▼ 11. 함수 표현식, 화살표 함수 (arrow function)

함수선언문 vs 함수 표현식

```
// 함수선언문
function sayHello(){
  console.log('Hello');
}

// 함수표현식
let sayHello = function(){
  console.log('Hello');
}
```

이 둘의 차이점은 무엇일까?

차이점은 호출할 수 있는 타이밍에서 차이가 있다.

■ 함수선언문

어디서든 호출이 가능하다.

```
function sayHello(){
  console.log('Hello');
}

sayHello();

-----------------위아래 둘 다 가능한 호출법-------------

sayHello();

function sayHello(){
```

```
  console.log('Hello');
}
```

■ 함수표현식

코드에 도달하면 생성한다.

```
...
...
let sayHello = function(){  -> 생성
  console.log(‘Hello’);  -> 사용가능
}
sayHello();
```

■ 화살표 함수(arrow function)

```
let add = function(num1, num2){
  return num1 + num2;
}

-------------위의 함수를 아래와 같이 변경가능하다-----------

let add = (num1, num2) => {
  return num1 + num2;
}
```

■ 예제

```
showError();

function showError(){
  console.log('error');
}

-----------------------
error
```

```
let showError = () => {
  console.log("error");
}

---------------------------
error
```

```
const sayHello = function(name) {
  const msg = `Hello, ${name}`;
  console.log(msg);
};

---------------------------------
error
```

```
const sayHello = (name) => {
  const msg = `Hello, ${name}`;
  console.log(msg);
};

-----------------------------
error
```

■ 인수가 두 개이고 return 문이 있는 예

```
const add = function (num1, num2) {
  const result = num1 + num2;
  return result;
};

----------아래와 같이 변경 가능하다 2가지로 ----------

const add = (num1, num2) => {
  return num1 + num2;
};

----혹은----

const add = (num1, num2) => num1 + num2;

// 이것은 리턴문이 하나이기에 최대한 간단히 표현한 방식이다.
```

▼ 12. 객체 (object)

마지막에 , 를 붙여주는 것이 좋다.

```
// Adult
// 이름: 조명찬
// 나이: 27

const Adult = {
  name : '조명찬', //key
  age : 27, //value
}
```

■ object

접근, 추가, 삭제

```
const Adult = {
  name : ‘조명찬’, (키 key)
  age : 27, (값 value)
}

-----------------------------------------------------

접근
Adult.name // 조명찬
Adult[‘age’] //27

추가
Adult.gender = ‘male’;
Adult[‘hariColor’] = ‘black’;

삭제
delete Adult.hairColor;
```

단축프로퍼티 (아래 세개 다 같은 표현이다.)

```
const name = ‘조명찬’
const age = 27;
```

```
const Adult = {
  name : name,
  age : age,
  gender : ‘male’,
}
```

```
const Adult = {
  name,
  age,
  gender : ‘male’,
}
```

프로퍼티 존재의 여부를 확인할 경우

```
const Adult = {
  name = ‘조명찬’,
```

```
  age = ‘27’,
}

Adult.birthDay; //undefined

‘birthDay’ in Adult; //false

‘age’ in Adult // true
```

어떤 값이 나올지 확신할 수 없을 때 in을 사용한다.

함수의 인자를 받거나 API 통신등을 통해 데이터를 받을때!

for ... in 반복문

객체를 순회하면서 값을 얻을 수 있다.

```
for(let key in Adult){
  console.log(key)
  console.log(Adult[key])
}
```

예

```
const superman = {
  name : 'clark',
  age : 30,
}

console.log(superman.name)
console.log(superman.age)

----------------------------------
“clark”
30
```

```
const superman = {
  name : 'clark',
  age : 30,
}

console.log(superman)

----------------------------------
object {
  name: “clark”
age:30,
}
```

```
const superman = {
  name : 'clark',
  age : 30,
}

superman.hariColor = 'black';
superman.['hobby'] = 'football';
console.log(superman)

---------------------------------
object {
  age: 30,
  hairColor: “black”,
  hobby: “football”,
  name: “clark”
}
```

```
const superman = {
  name : 'clark',
  age : 30,
}

delete superman.age;
console.log(superman)

---------------------------------
object {
  name: “clark”
}
```

이름과 나이를 받아서 객체를 변환하는 함수

```
function makeObject(name, age){
  return {
    name,
    age,
    hobby: "football"
  };
}

const Mike = makeObject("Mike", 30);
console.log(Mike);

//in을 통해 확인해보기
console.log("age" in Mike);
console.log("birthday" in Mike);

-----------------------------------------------------

object{
```

```
  age:30,
  hobby: “football”,
  name: “Mike”
}

true

false
```

in 을 활용하여 나이를 확인하고 성인인지 아닌지 구분

```
function isAdult(user){
  if(!('age' in user) || user.age < 20){
    return false;
  } else {
    return true;
  }
}

const Mike = {
  name : 'Mike',
  age : 30
}

const Jane = {
  name: "Jane"
};

console.log(isAdult(Mike))
console.log(isAdult(Jane))

--------------------------------------------------
true
```

💡 기본적 의미를 파악합시다.
!(‘age’ in user) || // user에 age 가 없거나
user.age <20 // 20살 미만이거나

false = if에 age in user 이라는 조건을 넣어주었기 때문에
false 가 나왔다. 보통 나이를 입력하지 않았을 경우에는
성인이 아니다 라고 판단하는 것이 좋다.

for in 을 활용한 예

```
const Mike = {
  name: "Mike",
```

```
  age: 30
};

for(x in Mike){
  console.log(x)
}

---------------------
name
age
```

💡 마이크가 가지고 있는 키값들을 알려줌

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

for(x in Mike){
  console.log(Mike[x])
}

--------------------------
“Mike”
30
```

💡 console.log(Mike[x]) // Mike[‘name’] 한번 돌고
console.log(Mike[x]) // Mike[‘age’] 실행

▼ 13. 객체 (object) - method, this

method : 객체 프로퍼티로 할당 된 함수

```
const superman = {
  name : ‘clark’,
  age : 30,
  fly : function(){
    console.log(‘날아갑니다’)
  }
}

superman.fly();

---------------------같은 의미 입니다. ------------------
```

```
const superman = {
  name : ‘clark’,
  age : 30,
  fly(){ //function 키워드 생략
    console.log(‘날아갑니다’)
  }
}

superman.fly();
```

object 와 method 의 관계

```
const user = {
  name : 'Mike',
  sayHello : function(){
    console.log(`Hello, I'm ${user.name}`);
  }
}
```

위의 방법은 문제가 발생할 수 있다.

```
const user = {
  name : 'Mike',
  sayHello : function(){
    console.log(`Hello, I'm ${this.name}`);
  }
}

user.sayHello();
----------------------------
Hello, I’m Mike
```

예

```
let boy = {
  name : ‘Mike’,
}

sayHello : function(){
  console.log(`Hello, I’m ${this.name}`);
}

boy.sayHello();
---------------------------------
I’m Mike
```

```
let girl = {
  name : ‘Jane’,
}

sayHello : function(){
  console.log(`Hello, I’m ${this.name}`);
}

girl.sayHello();
-----------------------------------
I’m Jane
```

💡 ※ this 는 값이 정해지지 않았다. 어떠한 함수를 호출하는가에 따라 값이 달라진다.

화살표 함수를 사용하게 되면 어떻게 될까?

```
sayHello : () => {
  console.log(`Hello, I’m ${this.name}`);
}
= 전혀 다른 결과가 나온다.
```

💡 왜냐하면 화살표 함수는 일반 함수와는 달리 자신만의 this를 가지고 있지 않기 때문이다.
화살표 함수 내부에서 this 를 사용하면, 그 this 는 외부에서 값을 가져온다.

화살표 함수시 주의사항에 대한 예시

```
let boy = {
  name : 'Mike',
  sayHello : () => {
    console.log(this);
   //전역객체 브라우저에서는 window Node js에서는 global
  }
}

boy.sayHello(); //this !=boy
```

Javascript에서의 this 는 상당히 복잡한 함수이다. 가장 어려운 부분 중 하나이다.

예

```
//매서드

let boy = {
  name : "Mike",
  showName : function () {
    console.log(boy.name)
  }
};

boy.showName();
= “Mike”
```

```
// 객체가 하나 있는데 boy로도 접근 가능하고
//  man 으로도 접근 가능하다.
let boy = {
  name : "Mike",
  showName : function () {
    console.log(boy.name)
  }
};

let man = boy;
man.name = "Tom"

console.log(boy.name)
= “Tom”
```

```
let boy = {
  name : "Mike",
  showName : function () {
    console.log(boy.name)
  }
};
let man = boy;
boy = null;

man.showName();

= 값 사라짐. 왜냐하면 boy를 null로 만들어
  내용이 사라졌기 때문이다.
```

```
let boy = {
  name : "Mike",
  showName : function () {
    console.log(this.name)
  }
};
```

```
let man = boy;
boy = null;

man.showName();
=“Mike”
```

▼ 14. 배열 (array)

순서가 있는 리스트

※배열은 대괄호로 묶어주고 쉼표로 구분한다.

하나하나 변수를 만들면 힘들다. 몇 개인지도 파악하기 힘들 것이고, 하나 하나 기억하는것도 어렵다. 때문에 사용함
배열을 탐색할 때는 index 라는 고유번호를 사용한다.

0부터 시작한다.

```
//1번 철수, 2번 영희 ... 30번 영수 를 입력할때

let students = [‘철수’ , ‘영희’ , ... ‘영수’];

console.log(students[0]); //철수
console.log(stundents[1]); //영희
----------------------------------------
수정하고 싶을 때
student[0] = ‘민정’;
console.log(students) // ‘민정’ , ‘영희’  , ...
```

배열의 특징

배열은 문자 뿐만 아니라, 숫자, 객체, 함수 등도 포함할 수 있다.

```
let arr = [
  '민수', //문자
  3, //숫자
  false, //불린
  {
    name: 'Mike', //객체
    age: 30, //객체
  },
  function(){ //함수
    console.log('TEST'); //함수
  }
];

이렇게 다양하게 넣을 수 있다.
```

배열의 매서드 (토글이니 눌러보시오)

▼ length : 배열의 길이를 구하고자 할 때 사용

students.length //30 << 정원이 30명이라는 가정 하에 나온 결과 값

▼ push() : 배열의 끝에 추가

```
let days = ['월','화','수'];
days.push('목')
console.log(days) //  ['월','화','수','목']
```

▼ pop() : 배열 끝 요소 제거

```
let days = ['월','화','수'];
days.pop()
console.log(days) //  ['월','화']
```

▼ shift, unshift : 배열 앞에 제거/추가

※ 여러 요소를 한번에 추가, 제거할 수 있다.

```
추가
day.unshift('일');
console.log(days) // ['일','월','화','수'];
제거
days.shift();
console.log(days) // ['월','화','수'];
```

▼ for : 반복문

```
let days = ['월','화','수'];
for(let index = 0; index < days.length; index++){
console.log(days[index])
}
```

▼ for ... of : 반복문

※for..in 과 헷갈리지 말 것!
for 문보다 간단하지만 index 값을 얻을 수 없다.

```
let days = ['월','화','수'];
for(let day of days){
console.log(day)
}
```

예

```
let days = ['mon', 'tue', 'wed'];

console.log(days[1]);

-----------------------------------
“tue”
```

```
let days = ['mon', 'tue', 'wed'];

days[1] = '화요일'

console.log(days);

-----------------------------------
["mon","화요일","wed"]
```

```
let days = ['mon', 'tue', 'wed'];

days[1] = '화요일'

console.log(days.length);

-----------------------------------
3
```

```
let days = ['mon', 'tue', 'wed'];

days.push('thu')

console.log(days.length);

-----------------------------------
["mon","tue","wed","thu"]
```

```
let days = ['mon', 'tue', 'wed'];

days.push('thu')
days.unshift("sun")
console.log(days);

-----------------------------------
["sun","mon","tue","wed","thu"]
```

```
let days = ['mon', 'tue', 'wed'];

days.push('thu')
days.unshift("sun")

for(let index = 0; index < days.length; index++){
  console.log(days[index]);
}

-----------------------------------
일월화수목 순서대로 연속 출력
```

```
let days = ['mon', 'tue', 'wed'];

days.push('thu')
days.unshift("sun")

for (let day of days){
  console.log(day)
}

-----------------------------------
일월화수목 순서대로 출력
```