JS

1. JavaScript?

Web Browser 위에서 동작하는 언어

변수
(Variable)

값을 저장할 수 있는 메모리 공간에 붙은 이름
즉, 값을 담는 그릇(컨테이너)

```
name = '손흥민';
birthYear = 1992;
```

변수는 어떤 정보에 이름을 붙여 저장하고 싶을 때 사용하며, 위와 같은 형태로 선언한다.

name이라는 변수명에는 손흥민이라는 값이 들어가 있고, birthYear에는 1992라는 값이 들어가 있다.

```
name = '손흥민';
Name = '박지성';
NAME = ‘안정환';

birthYear = 1992;
```

변수명은 대소문자를 구분하기 때문에
Name, NAME 등으로 새로운 변수를 생성할 수 있지만
혼란스럽기 때문에 이렇게 쓰는 것은 좋지 않다.

가독성을 부여하기 위해 이름을 짓는 규칙(네이밍 컨벤션)이 있다.

대표적인 것이 camelCase와 snake_case가 있다.

camelCase

합성어를 대문자로 구분
(각 단어의 첫글자를 대문자로 작성, 첫 글자 제외)

snake_case

합성어를 _로 구분

camelCase birthYear, myName, userInfoList

snake_case place_of_birth, my_age, address_code_list

변수를 만들 때 주의할 사항 중 하나는 예약어이다.

예약어는 사용 불가

```
name = '손흥민';
birthYear = 1992;
class = 'Top of The World';
```

이 코드를 실행하면 에러가 발생한다.

```
name = '손흥민';
birthYear = 1992;
class = 'Top of The World';
```

왜냐하면 class는 자바스크립트에서 이미 예약해 둔 단어이기 때문이다.

예약어는 자바스크립트에서 특정 목적을 위해 사용되기 때문에
변수 이름은 물론 나중에 배우게 될 함수 이름으로도 사용될 수 없다.

abstract	arguments	await	boolean
break	byte	case	catch
char	class	const	continue
debugger	default	delete	do
double	else	enum	eval
export	extends	false	final
finally	float	for	function
goto	if	implements	import
in	instanceof	int	interface
let	long	native	new
null	package	private	protected
public	return	short	static
super	switch	synchronized	this
throw	throws	transient	true
try	typeof	var	void
volatile	while	with	yield

alert()

경고창을 띄우는 함수 alert()

console.log()

로그를 찍는 함수 console.log()

```
name = '손흥민';
birthYear = 1992;

alert(name);
alert(birthYear);
alert('축구선수');
```

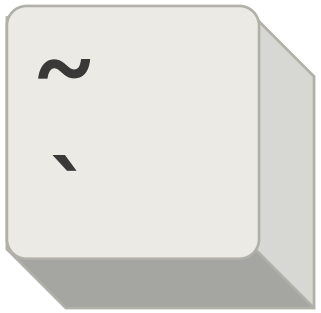
```
name = '손흥민';
birthYear = 1992;

console.log(name);
console.log(birthYear);
console.log('축구선수');
console.log('축구선수:', name, '태어난 년도:', birthYear);
```

```
name = '손흥민';
birthYear = 1992;
```

```
name = "손흥민";
birthYear = 1992;
```

```
name = `손흥민`;
birthYear = 1992;
```


ESC
F1
~
`
!
1
@
2
#
3
Tab
Q
W
Caps Lock
A
S


~
`

여기서 궁금한 것이 생길 수 있다. 따옴표가 포함된 문자열은 어떻게 작성할까?

프로그래밍 입문자를 위한 '자바스크립트'

```
message = ' 프로그래밍 입문자를 위한 '자바스크립트' ';
message = ' 프로그래밍 입문자를 위한 \'자바스크립트 \' ';
message = " 프로그래밍 입문자를 위한 '자바스크립트' ";
message = `프로그래밍 입문자를 위한 '자바스크립트'`;
```


Backspace
Ins
15ms
}
]
|
\ ₩
Del
Enter


|
\ ₩

사실 이렇게 변수를 선언하는 것은 매우 위험하다.

혼자 만드는 작은 프로젝트라면 괜찮을지 모르지만
여럿이 협업해야 하는 프로젝트라면 위험도는 더 증가한다.

```
name = '손흥민';
birthYear = 1992;
```

왜냐하면 다른 개발자가 name이라는 변수를 사용할 수도 있기 때문이다.

```
name = '손흥민';
birthYear = 1992;
.
.
.
name = '방탄소년단';
console.log(name);
```

이를 방지하기 위해 변수를 선언할 때,
let과 const 라는 2가지의 키워드가 사용된다.

let과 const를 사용한 변수 선언

이렇게 코드를 작성하면 'name'이 이미 선언되어 있다는 구문 오류가 발생한다.

즉, 다른 개발자는 name이 사용 중이라는 것을 알게 되어,
다른 변수명을 선택할 수 있게 되는 것이다.

```
let name = '손흥민';
birthYear = 1992;
.
.
.
let name = '방탄소년단';
console.log(name);
```

반대로 한 번 선언된 변수명을 반드시 변경해줘야하는 경우가 생길 수도 있다.
그럴 때는 이와 같이 let을 의도적으로 생략해주면 변경이 가능하다.

```
let name = '손흥민';
birthYear = 1992;
.
.
.
name = '방탄소년단';
console.log(name);
```

최초로 선언되는 변수에 let을 붙이는 습관을 들인다면,
이미 선언된 변수가 변경되는 것을 방지할 수 있다.

const는 절대로 바뀌지 않는 값인 상수를 취급할 때 사용한다.

변수명을 대문자로 하는 것이 규칙은 아니지만
상수임을 전달하기에 좋은 방법이 될 수 있다.

```
let name = '손흥민';
const BIRTH_YEAR = 1992;
```

변수명에 대한 규칙

1. 문자, 숫자, $, _ 만 사용 가능합니다.
2. 절대 첫 글자에는 숫자가 올 수 없습니다.
3. 절대 예약어는 사용할 수 없습니다.
4. 변수명은 읽기 쉽고 이해할 수 있게 선언하는 것이 좋습니다.

변수명으로 사용될 수 없는 것을 모두 고르세요.

A. `let 토트넘 = "England Football Club";`
B. `let name = '손흥민';`
C. `let 1th_position = 'Forward';`
D. `let number = 7;`
E. `let $ = '€ 80,000,000';`
F. `let _birth_Year = 1992;`
G. `let true = 'The best football player';`

변수명으로 사용될 수 없는 것을 모두 고르세요.

A. `let 토트넘 = "England Football Club";`
B. `let name = '손흥민';`
C. ~~`let 1th_position = 'Forward';`~~ // 숫자는 변수명 맨 앞에 올 수 없습니다.
D. `let number = 7;`
E. `let $ = '€ 80,000,000';`
F. `let _birth_Year = 1992;`
G. ~~`let true = 'The best football player';`~~ // true는 예약어입니다.

자료형

(Data Types)

기본형

- number
- undefined
- string
- null
- boolean

객체형 - object

- function
- array
- 정규 표현식
- • • •

숫자 자료형

```
const BIRTH_YEAR = 1992;
const PI = 3.14;
let age = 30;
let avgScore = 95.2;
```

숫자형 데이터는 숫자이기 때문에 연산이 가능하다.

```
let add = 2+7;
let subtract = 2-7;
let multiply = 2*7;
let square = 2**7;
let divide = 7/2;
let remainder = 7%2;
```

숫자 연산을 하다보면, Infinity와 NaN이라는 결과값이 나타나는 경우가 있다.

Infinity? NaN?

Infinity는 0으로 나누었을 때 발생하는 무한대의 값이다.

```
let unlimitedNumber = 5/0;

console.log(unlimitedNumber);
```

```
let unlimit = Infinity;

console.log(unlimit+3);
console.log(unlimit-3);
console.log(3-unlimit);
console.log(unlimit*3);
console.log(unlimit**3);
console.log(3**unlimit);
console.log(unlimit/3);
console.log(unlimit%3);
console.log(3/unlimit);
console.log(3%unlimit);
```

```
let unlimit = Infinity;


console.log(3-unlimit);



console.log(unlimit%3);
console.log(3/unlimit);
console.log(3%unlimit);
```

```
let unlimit = Infinity;


console.log(3-unlimit);          // -infinity



console.log(unlimit%3);          // NaN
console.log(3/unlimit);          // 0
console.log(3%unlimit);          // 3
```

NaN은 Not a Number의 줄임말로,
숫자가 아닌 값과 연산을 실행하는 경우에 나타나는 결과이다.

```
console.log('two'/2);      // NaN
```

문자 자료형

```
let name = '손흥민';
let team = "토트넘";
let position = `윙백`;
```

백틱(`)은 문자열 내부에서 변수를 표현할 수 있다는 특징이 있다.

```
let name = '손흥민';
const BIRTH_YEAR = 1992;
let message = `제가 응원하는 선수는 ${name}입니다.`;

console.log(message);
```

게다가 표현식을 통해 연산까지도 가능하다.

```
let name = '손흥민';
const BIRTH_YEAR = 1992;
let message = `제가 응원하는 선수는 만 ${2021-BIRTH_YEAR}살의 ${name}입니다.`;

console.log(message);
```

물론 문자열의 연산을 이용하는 방법도 있다.
문자열 사이 + 기호를 사용한 연산은 문자를 서로 연결해주는 역할을 한다.
(백틱 방식이 좀 더 쓰기 편하다)

```
let name = '손흥민';
const BIRTH_YEAR = 1992;
let message = '제가 응원하는 선수는 만 ' + 2021-BIRTH_YEAR + '살의 ' + name + '입니다.';

console.log(message);
```

불린 자료형

```
let isEasy = true;
let isHard = false;
```

```
let name = '손흥민';
const BIRTH_YEAR = 1992;

console.log(name == '손흥민');
console.log(BIRTH_YEAR < 1990);
```

undefined, null

선언된 변수에 아무런 값도 할당하지 않았을 때,
변수에 할당된 값이 없다는 의미로 undefined가 출력된다.

```
let noAssign;

console.log(noAssign);
```

Null은 빈 값이라는 의미로,
변수에 비어있는 값을 직접 할당하는 것이다.

```
let nullValue = null;

console.log(nullValue);
```

typeof() 함수를 이용하면 변수가 어떤 자료형인지를 확인할 수 있다.

```
let message = '손흥민의 등 번호는 ';
let backNumber = 7;

let newMessage = message + backNumber
console.log(typeof(newMessage));
```

```
let textTen = '10';
let five = 5;
console.log(textTen - five);
console.log(textTen * five);

let textFive = '5';
console.log(textTen - textFive);
console.log(textTen * textFive);
```

```
let textTen = '10';
let five = 5;
console.log(textTen + five);

let textFive = '5';
console.log(textTen + textFive);
```

형 변환

(Data Type Conversion)

```
console.log(typeof(true));
console.log(typeof(3.14));
console.log(typeof('손흥민'));
console.log(typeof('300'));
```

prompt()는 사용자로부터 어떤 값을 입력 받을 때 사용하는 함수이다.

(만약 입력을 하지 않고, 취소 버튼을 누르게 된다면 null 값이 저장되는 것을 확인할 수 있다.)

```
let name = prompt('이름을 입력하세요');
console.log(`${name} 입장`);
```

```
let phoneNumber = prompt('휴대폰 번호를 입력하세요', '010-****-****');

console.log(`${phoneNumber} 저장`);
```

prompt()를 이용해,
세 과목 [국어(kor), 영어(eng), 수학(math)]의 점수를 입력받고,

console.log()를 이용해,
세 과목 점수의 평균(avg)를 콘솔 창에 출력해보세요.

만약 이와 같이 적었다면, 평균 연산이 잘못된 것이다.

```
let kor = prompt('국어 점수를 입력하세요');
let eng = prompt('영어 점수를 입력하세요');
let math = prompt('수학 점수를 입력하세요');

let avg = kor + eng + math / 3;
console.log(avg);
```

연산에는 계산기처럼 우선순위가 정해져있다.
따라서 덧셈 뺄셈보다 곱셈 나눗셈을 우선 시 하는 경향이 있고,

그렇기 때문에 괄호를 통해 우선순위를 지정해줄 필요가 있다.

```
let kor = prompt('국어 점수를 입력하세요');
let eng = prompt('영어 점수를 입력하세요');
let math = prompt('수학 점수를 입력하세요');

let avg = (kor + eng + math) / 3;
console.log(avg);
```

하지만 괄호를 쳐서 계산을 했음에도 불구하고
결과값이 이상한 것을 알 수 있다.

```
let kor = prompt('국어 점수를 입력하세요');
let eng = prompt('영어 점수를 입력하세요');
let math = prompt('수학 점수를 입력하세요');

let avg = (kor + eng + math) / 3;
console.log(avg);
```

prompt로 입력받은 값은 무조건 문자 자료형이다.

따라서 입력받은 점수가 문자열로 연결되어 결과값이 나타나게 된 것이다.

```
let kor = prompt('국어 점수를 입력하세요');
let eng = prompt('영어 점수를 입력하세요');
let math = prompt('수학 점수를 입력하세요');

console.log(
      typeof(kor),
      typeof(eng),
      typeof(math)
);
```

문자 자료형이 숫자로 구성된 값일 경우에
숫자 연산이 자동으로 이루어지는 것을 '자동형변환'이라고 부른다.

```
let promotion = "708090" / 3;

console.log(typeof(promotion));
```

이렇게 자동으로 이루어지는 형변환은 편리하기도 하지만
입력한 국영수 세 과목의 평균 계산이 제대로 작동하지 않았듯이,
원인을 찾기 힘든 문제를 발생시키기도 한다.

Number 형 변환을 명시적으로 작성해줌으로써
처음에 시도했던 평균 점수가 이제야 제대로 동작할 수 있게 할 수 있다.

```
let kor = Number(prompt('국어 점수를 입력하세요'));
let eng = Number(prompt('영어 점수를 입력하세요'));
let math = Number(prompt('수학 점수를 입력하세요'));

let avg = (kor + eng + math) / 3;
console.log(avg);
```

```
console.log(
    Number('7'),
    Number('Two'),
    Number(true),
    Number(false),
    Number(null),
    Number(undefined)
);
```

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

```
console.log(
    Boolean(2),
    Boolean('0'),
    Boolean(' '),
    Boolean('true'),
    Boolean('false'),

    Boolean(0),
    Boolean(''),
    Boolean(null),
    Boolean(NaN),
    Boolean(undefined)
);
```

```
console.log(

    Boolean('0'),
    Boolean(' '),



    Boolean(0),
    Boolean('')



);
```

연산자

(Operator)

+	- 문자열의 연결, 더하기
-	- 빼기
*	- 곱하기
**	- 제곱
/	- 나누기
%	- 나머지

```
let isOdd = Number(prompt('숫자를 입력하세요'));

console.log(Number(isOdd%2));
```

```
let age = 30;

// 일년 후...
age = age + 1;

console.log(age);
```

```
let age = 30;

// 일년 후...
age += 1;

console.log(age);
```

```
let age = 30;

age += 1;
age -= 1;
age *= 2;
age **= 2;
age /= 3;
age %= 3;
```

```
let age = 30;

age -= 1;
age += 1;
```

```
let age = 30;

age -- 1;
age ++ 1;
```

증감 연산자는 변수명 앞뒤로 사용이 가능하다.
하지만 앞과 뒤에 놓인 위치에 따라 결과값이 달라진다.

```
let age = 10;
let exAge = --age;
console.log(age);
console.log(exAge);

age = 10;
let newAge = ++age;
console.log(age);
console.log(newAge);
```

```
let age = 10;
let exAge = age--;
console.log(age);
console.log(exAge);

age = 10;
let newAge = age++;
console.log(age);
console.log(newAge);
```

앞에 놓이면 1을 먼저 빼거나 더하는 연산을 진행한 뒤에,
exAge 또는 newAge 변수에 증가된 age 값을 저장하는 반면,

```
let age = 10;
let exAge = --age;
console.log(age);
console.log(exAge);

age = 10;
let newAge = ++age;
console.log(age);
console.log(newAge);
```

뒤에 놓이면 exAge 또는 newAge에 age 값을 저장한 뒤에,
age에 1을 빼거나 더하는 형태가 된다.

```
let age = 10;
let exAge = age--;
console.log(age);
console.log(exAge);

age = 10;
let newAge = age++;
console.log(age);
console.log(newAge);
```

<, >, <=, >=, ==, !=

```
console.log(typeof(1>2));
console.log(typeof(1<2));
console.log(typeof(3==3));
console.log(typeof(3!=3));
```

```
let name = '손흥민';
const BIRTH_YEAR = 1992;

console.log(name='박지성');
```

```
let name = '손흥민';
const BIRTH_YEAR = 1992;

name='박지성'
console.log(name);
```

```
let name = '손흥민';
const BIRTH_YEAR = 1992;

console.log(name=='박지성');
```

```
let name = '손흥민';
const BIRTH_YEAR = 1992;

console.log(name!='박지성');
```

```
let name = '손흥민';
const BIRTH_YEAR = 1992;

console.log(BIRTH_YEAR=='1992');
```

```
let name = '손흥민';
const BIRTH_YEAR = 1992;

console.log(BIRTH_YEAR==='1992');
```

조건문

(Conditional Statements)

if

switch

```
if (조건) {
    조건이 참이 경우, 실행될 코드
}
```

prompt()를 이용해,
국어(kor) 점수를 입력받고,

60점이 넘는 경우,
‘합격 하셨습니다’ 메시지가
콘솔 창에 출력(console.log)되도록 해보세요.

```
let kor = Number(prompt('국어점수를 입력하세요'));

if (kor >= 60) {
    console.log('합격하셨습니다.');
}
```

```
let kor = Number(prompt('국어점수를 입력하세요'));

if (kor >= 60) {
    console.log('합격하셨습니다.');
}
if (kor < 60) {
    console.log('불합격하셨습니다.');
}
```

```
if (조건) {
    조건이 참이 경우, 실행될 코드
} else {
    조건이 거짓일 경우, 실행될 코드
}
```

```
let kor = Number(prompt('국어점수를 입력하세요'));

if (kor >= 60) {
    console.log('합격하셨습니다.');
} else {
    console.log('불합격하셨습니다.');
}
```

```
if (조건1) {
    조건1이 참일 경우, 실행될 코드
} else if (조건2) {
    조건2이 참일 경우, 실행될 코드
}
else {
    모든 조건이 거짓일 경우, 실행될 코드
}
```

```
let kor = Number(prompt('국어점수를 입력하세요'));

if (kor >= 60) {
    console.log('합격하셨습니다.');
} else if (kor === 100) {
    console.log('축하합니다.');
} else {
    console.log('불합격하셨습니다.');
}
```

```
let kor = Number(prompt('국어점수를 입력하세요'));

if (kor === 100) {
    console.log('축하합니다.');
} else if (kor >= 60) {
    console.log('합격하셨습니다.');
} else {
    console.log('불합격하셨습니다.');
}
```

콩쥐팥쥐를 읽었니?	흥부놀부를 읽었니?	'콩쥐팥쥐' 와 '흥부놀부' 를 읽었니?
예	예	
예	아니오	
아니오	예	
아니오	아니오	

콩쥐팥쥐를 읽었니?	흥부놀부를 읽었니?	'콩쥐팥쥐' 나 '흥부놀부' 를 읽었니?
예	예	
예	아니오	
아니오	예	
아니오	아니오	

콩쥐팥쥐를 읽었니?	흥부놀부를 읽었니?	'콩쥐팥쥐' 와 '흥부놀부' 를 읽었니?
예	예	예
예	아니오	아니오
아니오	예	아니오
아니오	아니오	아니오

콩쥐팥쥐를 읽었니?	흥부놀부를 읽었니?	'콩쥐팥쥐' 나 '흥부놀부' 를 읽었니?
예	예	예
예	아니오	예
아니오	예	예
아니오	아니오	아니오

&&	||
AND	OR

```
console.log (true && true)

console.log (true && false)
console.log (false && true)
console.log (false && false)
```

```
console.log (true && true)
console.log (true && false)
console.log (false && true)

console.log (false && false)
```

사용자로부터 숫자를 입력 받아 콘솔창에 메시지를 출력합니다.

메시지 내용은 아래와 같습니다.

해당 숫자가 5의 배수 또는 7의 배수이면 '통과'
그 외의 숫자는 '통과 실패'
(단, 숫자를 입력하지 않거나 0인 경우에는 '에러')

```
let num = Number(prompt('숫자를 입력하세요'));

if (!num) {
    console.log('에러');
} else if (num % 5 === 0 || num % 7 === 0) {
    console.log('통과');
} else {
    console.log('통과 실패');
}
```

```
switch (변수) {
    case x:
            // 변수 값이 x인 경우, 실행될 코드
            break;
    case y:
            // 변수 값이 y인 경우, 실행될 코드
            break;
}
```

```
let decHoliday = prompt('12월 날짜를 입력하세요', '12월 01일');

switch (decHoliday) {
        case '12월 03일':
                console.log('소비자의 날');
        case '12월 05일':
                console.log('무역의 날');
        case '12월 25일':
                console.log('크리스마스');
        case '12월 27일':
                console.log('원자력의 날');
}
```

```
let decHoliday = prompt('12월 날짜를 입력하세요', '12월 01일');

switch (decHoliday) {
        case '12월 03일':
                console.log('소비자의 날');
                break;
        case '12월 05일':
                console.log('무역의 날');
                break;
        case '12월 25일':
                console.log('크리스마스');
                break;
        case '12월 27일':
                console.log('원자력의 날');
                break;
}
```

```
let decHoliday = prompt('12월 날짜를 입력하세요', '12월 01일');

switch (decHoliday) {
        case '12월 3일':
        case '12월 03일':
                console.log('소비자의 날');
                break;
        case '12월 5일':
        case '12월 05일':
                console.log('무역의 날');
                break;
        case '12월 25일':
                console.log('크리스마스');
                break;
        case '12월 27일':
                console.log('원자력의 날');
                break;
}
```

```
switch (변수) {
    case x:
            // 변수 값이 x인 경우, 실행될 코드
            break;
    case y:
            // 변수 값이 y인 경우, 실행될 코드
            break;
    default :
            // 조건에 해당하지 않는 경우, 실행될 코드
}
```

```
let decHoliday = prompt('12월 날짜를 입력하세요', '12월 01일');

switch (decHoliday) {
        case '12월 3일':
        case '12월 03일':
                console.log('소비자의 날');
                break;
        case '12월 5일':
        case '12월 05일':
                console.log('무역의 날');
                break;
        case '12월 25일':
                console.log('크리스마스');
                break;
        case '12월 27일':
                console.log('원자력의 날');
                break;
        default:
                 console.log('입력값 오류 또는 보통날');
}
```

```
let decHoliday = prompt('12월 날짜를 입력하세요', '12월 01일');

if (decHoliday === '12월 3일'  || decHoliday === '12월 03일') {
            console.log('소비자의 날');
} else if (decHoliday === '12월 5일' || decHoliday === '12월 05일') {
            console.log('무역의 날');
} else if (decHoliday === '12월 25일') {
            console.log('크리스마스');
} else if (decHoliday === '12월 27일') {
            console.log('원자력의 날');
} else {
            console.log('입력값 오류 또는 보통날');
}
```

반복문

(Looping Statements)

```
let Num = Number(prompt('점수를 입력하세요'));
if (Num >= 60) {
        console.log('합격입니다.');
} else {
        console.log('불합격입니다.');
}

Num = Number(prompt('점수를 입력하세요'));
if (Num >= 60) {
        console.log('합격입니다.');
} else {
        console.log('불합격입니다.');
}

Num = Number(prompt('점수를 입력하세요'));
if (Num >= 60) {
        console.log('합격입니다.');
} else {
        console.log('불합격입니다.');
}
```

반복문이 필요한 이유

효율적인 코드 작성!

for

while

do while

```
for (시작값; 조건; 주기별작업) {
    반복되는 코드
}
```

```
let score;

for (let i = 0; i < 3; i++) {
        score = Number(prompt('점수를 입력하세요'));
       if (score >= 60) {
          console.log('합격입니다.');
       } else {
          console.log('불합격입니다.');
       }
}
```

prompt()를 이용해,
숫자를 입력받고,

for 반복문을 이용해,
입력받은 숫자만큼
‘Hello JavaScript’가 콘솔창에 출력되도록 작성하세요.

```
let num = Number(prompt('숫자를 입력하세요'));

for (let i = 0; i < num; i++) {
      console.log('Hello JavaScript');
}
```

```
while (조건) {
    반복되는 코드
    주기별작업
}
```

```
let score;
let i = 0;

while (i < 3) {
        score = Number(prompt('점수를 입력하세요'));
       if (score >= 60) {
          console.log('합격입니다.');
       } else {
          console.log('불합격입니다.');
       }
       i++;
}
```

prompt()를 이용해,
숫자를 입력받고,

while 반복문을 이용해,
1부터 입력받은 숫자까지
콘솔창에 하나씩 출력되도록 작성하세요.

```
let num = Number(prompt('숫자를 입력하세요'));
let i = 1;

while (i <= num) {
      console.log(i);
      i++
}
```

```
do {
    반복되는 코드
    주기별작업
} while (조건)
```

```
let score;
let i = 0;

do {
        score = Number(prompt('점수를 입력하세요'));
       if (score >= 60) {
          console.log('합격입니다.');
       } else {
          console.log('불합격입니다.');
       }
       i++;
} while (i < 3)
```

prompt()를 이용해,
숫자를 입력받고,

do while 반복문을 이용해,
1부터 입력받은 숫자까지
콘솔창에 하나씩 출력되도록 작성하세요.

```
let num = Number(prompt('숫자를 입력하세요'));
let i = 1;

do {
      console.log(i);
      i++
} while (i <= num)
```

```
let num = Number(prompt('숫자를 입력하세요'));
let i = 1;

while (i <= num) {
      console.log(i);
      i++
}
```

```
let num = Number(prompt('숫자를 입력하세요'));
let i = 1;

do {
      console.log(i);
      i++
} while (i <= num)
```

break

continue

break	continue
반복문을 빠져나옵니다.	다음 주기로 이동합니다.

```
while (true) {
    let num = Number(prompt('손흥민 선수의 등 번호는 몇 번일까요?'));
    if (num === 7) {
        console.log('정답입니다.');
        break;
    } else {
        console.log('틀렸습니다');
    }
}
```

```
let sum = 0;
for (let num=1; num<=10; num++) {
      if (num == 5) {
                  continue;
      }
      sum += num;
}
console.log(sum);
```

객체
(Object)

객체는 중괄호로 작성하고, 키(key)와 값(value)을 한 쌍으로 하는 프로퍼티가 들어간다.

키와 값은 콜론으로 구분하며, 프로퍼티끼리는 쉼표를 사용하여 구분한다.

마지막 프로퍼티의 쉼표는 안써도 무방하나,
쉼표를 쓰는 것이 나중에 수정 추가 삭제가 간편해지는 이점이 있다.

```
let 손흥민 = {
    직업 : '축구선수',
    국적 : '대한민국',
    키 : 188,
    등번호 : 7,
    생년월일 : '1992-07-08',
}
```

```
// 객체에 접근

let 손흥민 = {
    직업 : '축구선수';
    국적 : '대한민국',
    키 : 188,
    등번호 : 7,
    생년월일 : '1992-07-08',
}

console.log(손흥민);
console.log(손흥민.직업);
console.log(손흥민['직업']);
```

```
// 객체의 추가

let 손흥민 = {
    직업 : '축구선수';
    국적 : '대한민국',
    키 : 188,
    등번호 : 7,
    생년월일 : '1992-07-08',
}

손흥민.팀 = '토트넘 홋스퍼';
손흥민['혈액형'] = 'AB';

console.log(손흥민);
```

```
// 객체의 수정

let 손흥민 = {
    직업 : '축구선수';
    국적 : '대한민국',
    키 : 188,
    등번호 : 7,
    생년월일 : '1992-07-08',
}

손흥민.직업 = 'Athlete';
손흥민['국적'] = 'South Korea';

console.log(손흥민);
```

```
// 객체의 삭제

let 손흥민 = {
    직업 : '축구선수';
    국적 : '대한민국',
    키 : 188,
    등번호 : 7,
    생년월일 : '1992-07-08',
}

delete 손흥민.등번호;
delete 손흥민['생년월일'];

console.log(손흥민);
```

```
let name = '손흥민';
let age = 30;

let son = {
    name : name,
    age : age,
    backnumber : 7,
}
```

```
let name = '손흥민';
let age = 30;

let son = {
    name,
    age,
    backnumber : 7,
}
```

```
let son = {
    name : '손흥민',
    age : 30,
    backnumber : 7,
}

console.log(son.height);
```

```
let son = {
    name : '손흥민',
    age : 30,
    backnumber : 7,
}

console.log('height' in son);
console.log('name' in son);
```

```
for (let 키 in 객체) {
    객체와 키를 이용한 반복 코드
}
```

```
let son = {
      name : '손흥민',
      age : 30,
      backnumber : 7,
}

for (let key in son) {
   console.log(key, son[key]);
}
```

배열

(Array)

위와 같이 변수로 만든다면, 생성도 힘들뿐더러 일일이 변수명을 기억해야 하고, 순서도 일정하지 않기 때문에 몇 명인지 파악하는 것도 어려울 것이다.

```
let jungkook = 'Jungkook';
let v = 'V';
let jimin = 'Jimin';
let jin = 'Jin';
let suga = 'Suga';
let rm = 'RM';
let jHope = 'JHope';
```

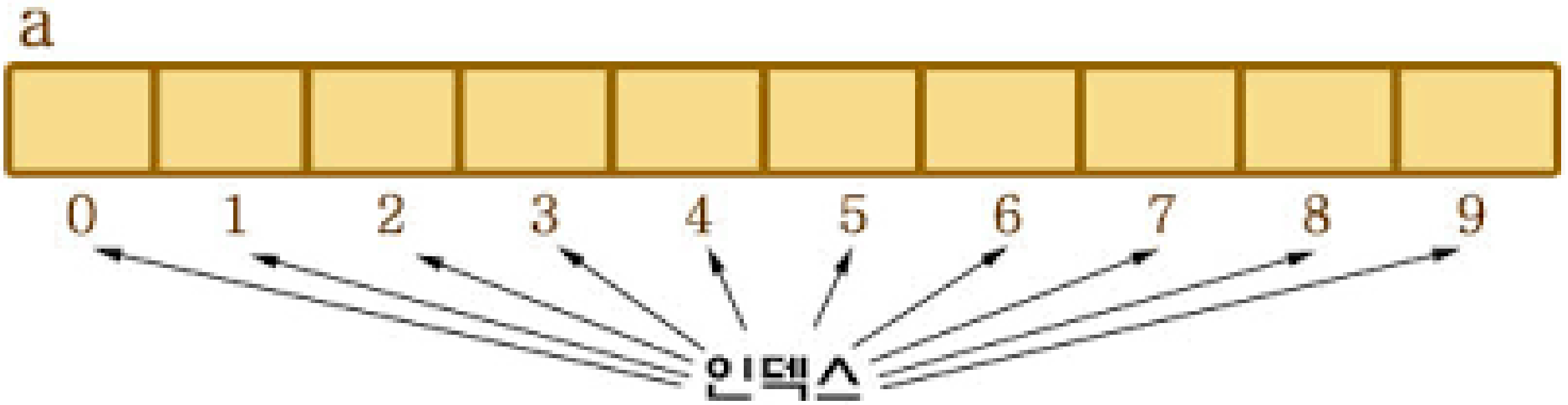
bts라는 변수에 방탄소년단 멤버 이름을 순서대로 넣어주었다.
배열은 대괄호로 감싸주고 쉼표로 구분한다.

```
let bts = ['Jin', 'Suga', 'JHope', 'RM', 'Jimin', 'V', 'JungKook'];
```

```
let bts = ['Jin', 'Suga', 'JHope', 'RM', 'Jimin', 'V', 'JungKook'];
console.log(bts.length);
console.log(`방탄소년단의 멤버의 수는 총 ${bts.length}명 입니다.`);
```

배열도 객체의 일부이기 때문에
객체와 같이 문자 뿐만 아니라, 숫자, 불린, 객체, 함수 등 다양한 자료형을 요소로 사용할 수 있다.

```
let arrayExample = [
        '손흥민',
        7,
        true,
        {
                name : 'Son',
                age : 30,
                shooting: function() {
                        console.log('슈팅');
                }
        },
        function() {
                console.log('배열에는 문자, 숫자, 불린, 객체, 함수 등이 다양하게 넣을 수 있다.');
        }
]
```


a
0
1
2
3
4
5
6
7
8
9
인덱스

```
let bts = ['Jin', 'Suga', 'JHope', 'RM', 'Jimin', 'V', 'JungKook'];

console.log(bts);
console.log(bts[0]);
console.log(bts[2]);
console.log(bts[4]);
console.log(bts[6]);
```

```
let bts = ['Jin', 'Suga', 'JHope', 'RM', 'Jimin', 'V', 'JungKook'];

bts[3] = 'RapMonster';

console.log(bts[3]);
```

추가 시에는 배열의 크기가 증가하며, 만약 중간의 번호를 건너뛴다면 공백이 생기게 된다.
삭제 시에는 배열의 길이가 줄어들지 않는다.

따라서 배열의 추가와 삭제는 다른 방법을 이용하게 된다.

```
let bts = ['Jin', 'Suga', 'JHope', 'RM', 'Jimin', 'V', 'JungKook'];
bts[9] = 'someone';

console.log(bts);

delete bts[9];
console.log(bts);
```

```
let rainbow = ['빨', '주', '노', '초', '파', '남'];

rainbow.push('보');

console.log(rainbow);
```

```
let rainbow = ['빨', '주', '노', '초', '파', '남'];

rainbow.push('보');
console.log(rainbow);

rainbow.pop();
console.log(rainbow);
```

```
let days = ['화', '수', '목', '금', '토', '일']

days.unshift('월');
console.log(days);

days.shift();
console.log(days);
```

```
let raindow = ['빨', '주', '노', '초'];

rainbow.push('파', '남', '보');
console.log(rainbow);
```

```
let days = ['수', '목', '금', '토']

days.unshift('일', '월', '화');
console.log(days);
```

```
console.log('무지개 색 출력하기');

let rainbow = ['빨강', '주황', '노랑', '초록', '파랑', '남색', '보라'];

for (let i = 0; i < rainbow.length; i++) {
    console.log(`${i+1}번째 : ${rainbow[i]}`);
}
```

배열에서 반복문을 더 간편하게 쓰는 방법은 for of문을 사용하는 것이다.

단, index 번호를 얻지 못한다는 단점이 있다.

```
let days = ['월', '화', '수', '목', '금', '토', '일'];

for (let day of days) {
    console.log(`${day}요일`);
}
```

함수

(Function)

console.log()

alert()

prompt()

함수
(Function)

typeof()

Number()

String()

함수 정의	특정 행동을 하도록 만드는 명령어 생성
함수 호출	명령어를 이용해, 특정 행동 유도

함수 정의

```
function 함수명 (매개변수) {
    // 함수 호출 시 실행되는 코드
}
```

함수 호출

```
함수명 (인수);
```

```
function errMsg() {
    console.log('에러 발생');
}

errMsg();
```

```
function errMsg (errCode) {
      let newErrCode = errCode || '알 수 없음' ;
      if (errCode) {
                  let msg = `에러발생! 에러코드: ${newErrCode}`;
      }
      console.log(msg);
}

errMsg('Login_001');
errMsg('Join_002');
errMsg();
```

```
function errMsg (errCode = '알 수 없음') {
    let msg = `에러코드: ${errCode}`;
    console.log(msg);
}

errMsg('Login_001');
errMsg('Join_002');
errMsg();
```

전역변수

전체에서 접근 가능한 변수

지역변수

함수 내부에서만 접근이 가능한 변수

함수 안에서 정의되고 사용된 msg라는 변수는 함수 바깥에서 출력되지 못하고 있다.

이러한 변수는 지역변수라고 불리며 함수 바깥에서는 사용이 불가능하다.

```
function errMsg (errCode = '알 수 없음') {
    let msg = `에러코드: ${errCode}`;
    console.log(msg);
}

errMsg();
console.log(msg);    // 에러 발생 (정의된 변수가 없음)
```

하지만 함수 바깥에서 정의된 변수는
전역 변수라고 불리면서 어디서든 사용이 가능하다.

```
let var_G = '함수의 바깥';

function errMsg (errCode = '알 수 없음') {
    let msg = `에러코드: ${errCode}`;
    console.log(var_G);
}

errMsg();
console.log(var_G);
```

처음에 변수에 대해서 배울 때 let으로 선언된 변수에 값을 재할당하기 위해서는 let를 붙이지 않아야 하며,
이를 통해 값이 할당된 변수인지 아닌지 확인할 수 있다고 배웠다.

하지만 variable이라는 변수를 각각 전역변수와 지역변수로 선언하게 된다면,
서로 간섭받지 않고 독립적으로 사용이 가능하다.

```
let variable = '전역변수';

function variableTest() {
    let variable = '지역변수';
    console.log(`함수를 통해 호출되는 변수는 ${variable}입니다.`);
}

variableTest();
console.log(`바깥에서 호출되는 변수는 ${variable}입니다.`);
```

공통적으로 사용되는 변수가 아니라면 지역변수로 선언하는 것이 변수 관리에 편리하다.

```
let variable = '전역변수';
console.log(`함수 호출 전, 변수는 ${variable}입니다.`);

function variableTest() {
        variable = '변경된 전역변수';
        console.log(`함수를 통해 호출되는 변수는 ${variable}입니다.`);
}

variableTest();

console.log(`함수 호출 후, 변수는 ${variable}입니다.`);
```

어떤 값을 반환하는 함수를 만들 때에는 return이라는 구문을 사용한다.

return

return은 함수 실행을 종료하고, 주어진 값을 함수 호출 지점으로 반환하는 역할을 한다.

```
function rectangle ( width, height = null ) {
    if (height === null) {
        return width * width;
    } else {
        return width * height;
    }
}

let rectangleArea = rectangle(3);
console.log(rectangleArea);

let squareArea = rectangle(3, 4);
console.log(squareArea);
```

return문이 없는 함수는 항상 undefined를 반환한다.

```
function errMsg (errCode = '알 수 없음') {
    let msg = `에러코드: ${errCode}`;
    alert(msg);
}

let result = errMsg();

console.log(result);
```

비어있는 return문도 항상 undefined를 반환한다.

```
function errMsg (errCode = '알 수 없음') {
    let msg = `에러코드: ${errCode}`;
    alert(msg);
    return;
}

let result = errMsg();

console.log(result);
```

return에 도달하면 함수의 실행이 중단된다.

```
function errMsg (errCode = '알 수 없음') {
    let msg = `에러코드: ${errCode}`;
    console.log(msg);
    return;
    console.log('함수가 종료되었기 때문에 절대 실행되지 않는 내용입니다.');
}

let result = errMsg();
```

함수를 작성하는 또 다른 방법, 함수 표현식에 대해 알아보자.

함수 표현식

함수 표현식은 식으로 나타나기 때문에 등호가 생긴다.

```
let errMsg = function (errCode = '알 수 없음') {
    let msg = `에러코드: ${errCode}`;
    console.log(msg);
}

errMsg();
```

함수 선언문과 함수 표현식을 비교해보자.

함수 선언문

```
function errMsg (errCode = '알 수 없음') {
    let msg = `에러코드: ${errCode}`;
    console.log(msg);
}

errMsg();
```

함수 표현식

```
let errMsg = function (errCode = '알 수 없음') {
    let msg = `에러코드: ${errCode}`;
    console.log(msg);
}

errMsg();
```

콘솔에 찍히지 않는다는 것을 이미 알 수 있다.

왜냐하면 javascript는 인터프리터 언어이기 때문이다.

```
console.log(num);

let num = '콘솔에 찍힐까?';
```

위에서부터 아래로 순서대로 실행되는 언어를 인터프리터 언어라고 한다.

분명 같은 상황임에도 불구하고,
해당 코드는 문제없이 작동이 되고 있는 것을 알 수 있다.

```
errMsg();

function errMsg (errCode = '알 수 없음') {
    let msg = `에러코드: ${errCode}`;
    console.log(msg);
}
```

바로 함수선언문과 함수표현식에서의 차이가 여기에 있다.

```
errMsg();

let errMsg = function (errCode = '알 수 없음') {
    let msg = `에러코드: ${errCode}`;
    console.log(msg);
}
```

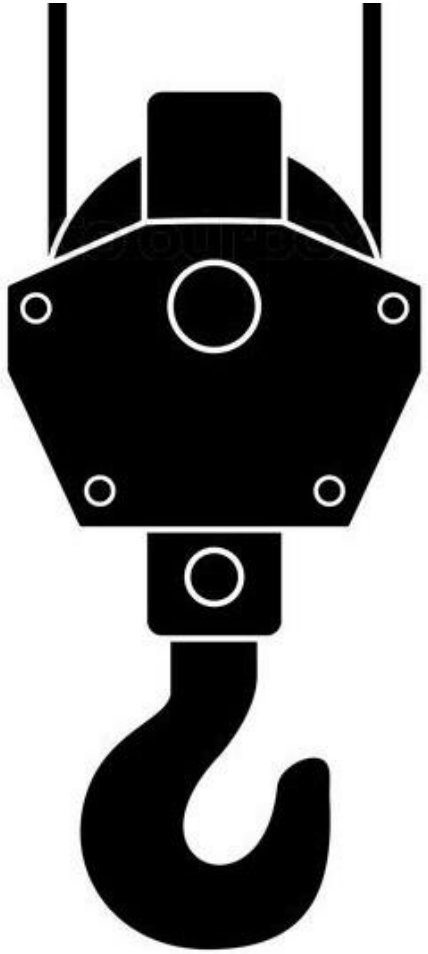
자바스크립트는 위에서부터 아래 순서로 실행되는 인터프리터 언어이지만
컴파일 언어의 성격을 갖는다는 특징이 있다.

자바스크립트를 실행하면 제일 먼저 parser가 빌드를 진행하면서 문법을 검사한다.
문법 검사 시에 자바스크립트는 코드 내에 선언할 수 있는 것들을 모두 선언하게 된다.

이로써 함수 선언문의 사용범위가 위쪽까지 확대되는 것이다.

반면에 함수표현식은 위에서부터 아래로 순서대로 실행되기 때문에
해당 코드에 도달되어야 비로소 생성되는 것이다.

이것을 호이스팅이라고 한다.



JAVASCRIPT HOISTING