자바스크립트 기초

1. 자바 스크립트란?

웹 페이지를 동적으로, 그리고 상호작용적으로 만들기 위해 사용되는 프로그래밍 언어

- 기존에 배웠던 HTML로는 정적웹(Static Web), 즉 데이터가 갱신되지 않거나, 사용자와 상호작용을 할 수 없는 페이지만 만들 수 있었음.
- 현대적인 웹에서는 정적웹 말고도 웹 브라우저에 표시된 내용을 조작하거나 상호작용하게 할 수 있다. 이러한 웹을 정적웹과 대비해 동적 웹(Dynamic Web)이라고 한다.

⇒ 자바 스크립트는 이러한 동적 웹을 만들 때 사용하는 언어가 볼 수 있다.

Java Script vs JAVA

Java Script 와 Java는 직접적인 연관이 없는 별개의 언어

자바스크립는 1995년 Netscape에서 웹브라우저에 프로그래밍 기능을 추가하기로 하고 새로운 언어 개발하였다. 처음에는 '**Mocha**'라는 이름으로 개발되었고 이후 '**LiveScript**'로 바뀌었다가

Java의 인기에 편승하기 위해 현재의 이름인 '**JavaScript**'로 최종 확정

⇒ 즉 자바와 연관이 있어서 자바스크립트인 것이 아닌, 정말 그저 이름만 비슷한 언어

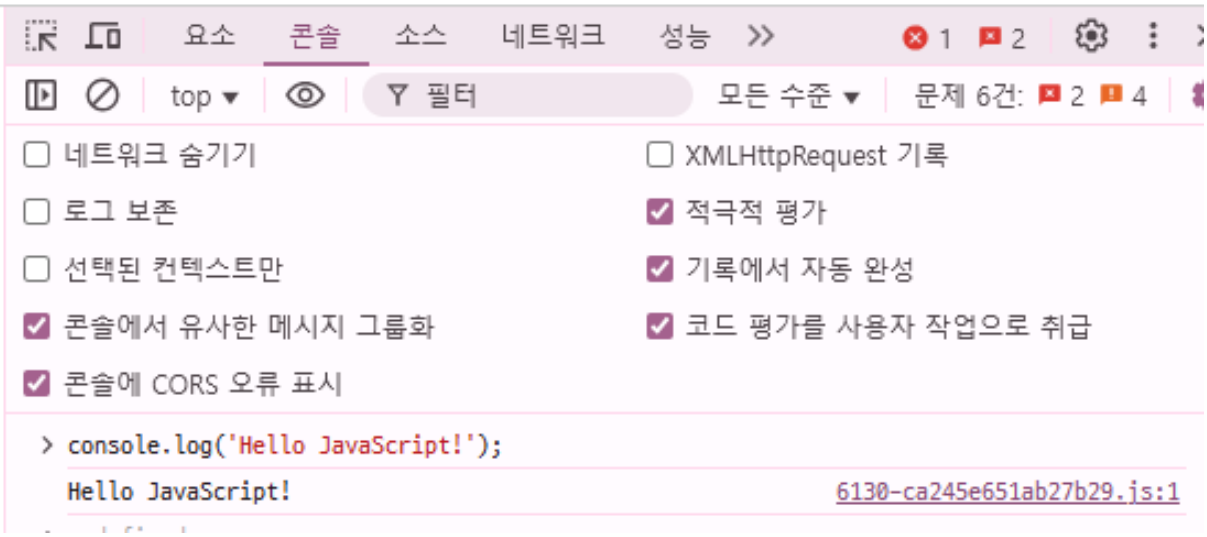
2. 자바 스크립트 사용해보기

개발자 도구에서 자바스크립트 실행해보기

자바스크립트는 여러분의 브라우저에서 언제든 실행해볼 수 있습니다. 개발자 도구를 사용하면 되는데요, 윈도우는 `F12 혹은 Ctrl + Shift + I` 맥에서는 `Cmd + Option + I`를 눌러 개발자 도구로 들어갈 수 있습니다.

개발자 도구의 콘솔 창에서 다음과 같은 코드를 입력해보세요

```
console.log('Hello JavaScript!');
```

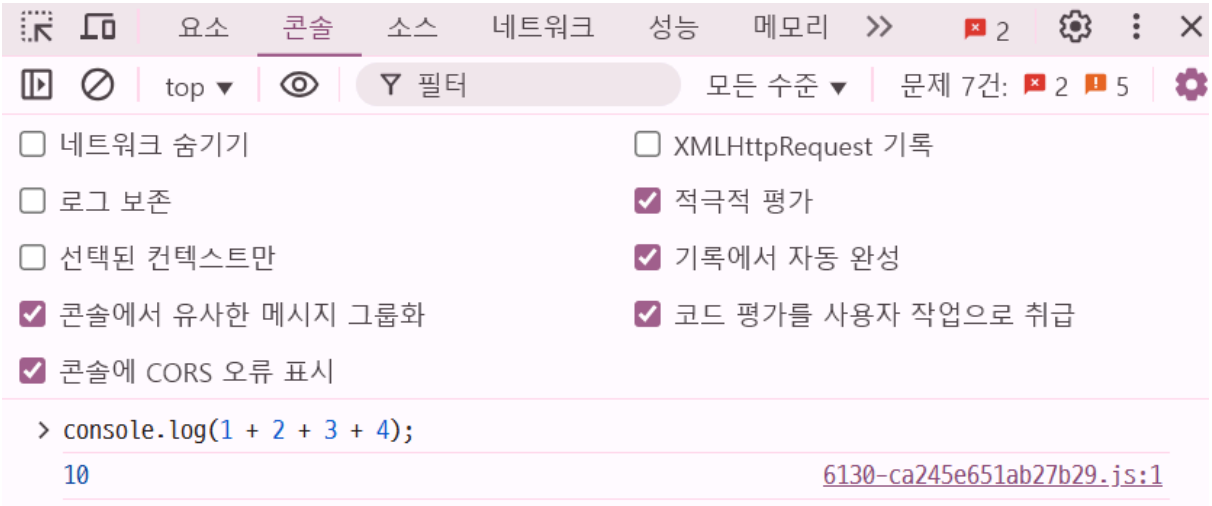


그렇다면 결과로 Hello JavaScript가 출력되는 것을 볼 수 있습니다.

여기서 `console.log` 는 콘솔에 특정 내용을 출력하는 것을 의미합니다,

이번에는 다음과 같은 코드를 한번 입력해보세요

```
console.log(1 + 2 + 3 + 4);
```



이번에는 10이라는 결과가 나타날 겁니다.

자바스크립트는 이렇게 연산도 가능합니다. 그런데 매번 코드를 이런 개발자 도구에서 짜는 건 상당히 불편할겁니다. 그래서 저희는 이전 세션에서 썼던 것과 마찬가지로 VScode를 이용해서 코드를 배워볼 예정입니다.

VScode에서 자바스크립트 실행해보기

JS main.js

vscode에서 `main.js` 파일을 생성합니다.(파일명은 뭐든 상관 없습니다.)

마찬가지로 위에서 했던 다음 두가지 코드를 작성하고 실행해보세요

```
console.log('Hello JavaScript!');
console.log(1 + 2 + 3 + 4);
```

다음과 같이 결과가 잘 나온다면 성공입니다.

3. 변수와 상수

프로그래밍 언어를 공부할 때 가장 먼저 배우는 개념은 데이터 처리인데요, 데이터를 처리하려며 데이터를 저장할 공간이 필요합니다. 자바스크립트에서는 데이터를 저장하는 공간을 **변수**와 **상수**라고 합니다.

변수

변수는 이름에서 볼 수 있듯 변하는 수 입니다. 그래서 자바스트립트에서는 값이 변하는 데이터를 저장하고 관리하기 위한 공간으로 변수를 만듭니다.

```
선언키워드 변수이름 = 값;
```

이런 식으로 변수를 선언하게 됩니다.

상수

상수는 변수와 반대로 변할 수 없는 수입니다. 다시 말해 자바스크립트에서 한번 선언되고 나면 그 데이터를 바꿀 수 없습니다.

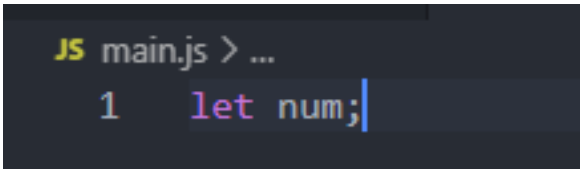
```
선언키워드 상수이름 = 값;
```

상수 선언 형태는 변수와 비슷합니다.

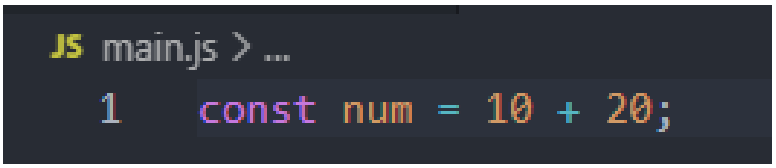
선언, 할당, 초기화

변수를 생성하고 값을 저장할 때 우리는 키워드를 사용해 내가 생성하고자 하는 변수가 어떤 변수(데이터 타입)인지 **선언**합니다. 그리고 할당 연산자인 기호 = 을 이용해 우변에 있는 값을 변수 공간에 대입(저장)하는 것을 **값을 할당한다**고 합니다.

변수 초기에는 아래와 같이 값을 할당하지 않고 선언만 할 수 있습니다.



또한 선언과 할당을 같이 하는 것도 가능한데, 선언과 할당을 한번에 하면 **변수를 초기화한다**고 합니다.



여기서 잠깐 let? const? 이게 뭘까요??

파이썬에서는 변수 앞에 저런게 없었어요. 또한 파이썬 외 다른 언어를 해보신 분들이라면 변수 앞에 변수의 자료형이 붙는게 익숙하실 수도 있는데요!

자바스크립트에서 변수 앞에 붙는 것은 자료형이 아니랍니다 ^_^

자바스크립트에서 변수 또는 상수를 선언할 때는 `let`, `const` , `var` 세 가지 키워드를 사용해요.

let

- **변수**를 사용할 때 쓰는 키워드
- 변수 재선언 불가능, 재할당은 가능

```
let y = 1;
// let y = 2; // 재선언 불가능 => 이거 실행해보면 오류가 납니다~!
y = 3;// 재할당 가능

console.log(y);// 출력값 => 3
```

const

- **상수**를 사용할 때 쓰는 키워드
- 변수 재선언, 재할당 모두 불가능
- 반드시 선언과 동시에 초기화 필요

```
const z = 1;
// const z = 2; // 재선언 불가능// z = 3; // 재할당 불가능


console.log(z);// 1 출력
```

var

- 가장 오래된 변수 선언 키워드로 지금은 잘 사용되지는 않습니다.
- 재선언, 재할당이 모두 가능함. ⇒ 변수명을 실수로 중복 사용이 가능함

```
var x = 1;
var x = 2;// 재선언 가능
x = 3;// 재할당 가능

console.log(x);// 3 출력
```

4. 자료형

자료형이란 자바스크립트에서 사용할 수 있는 데이터의 종류를 의미합니다. 간단히 말하자면 자료형은 데이터가 어떤 모양이고 어떤 종류인지를 정의하는 것입니다.

숫자(Number) 타입

자바스크립트는 정수와 실수를 구분하지 않고 전부 하나의 자료형(숫자형)으로 취급합니다. 예를 들어 10이나 0.1이나 같은 숫자형입니다.

```
let num1 = 10;
let num2 = 0.1;
```

문자열(string) 타입

자바스크립트에서 문자열은 “Hello, JavaScript”나 ‘Hello JavaScript’와 같이 큰따옴표(“”)나 작은 따옴표(‘’)로 둘러싸인 텍스트 데이터를 의미합니다.

주의할 점은 시작과 끝의 따옴표가 같아야 한다는 것입니다. (큰 따옴표로 시작했으면 끝도 큰 따옴표로, 작은 따옴표로 시작했으면 끝도 작은 따옴표로 끝나야함)

```
let string1 = “Hello, JavaScript”;
let string2 = ‘Hello JavaScript’;
```

문자열의 연결 연산자는 우리가 알고 있는 덧셈 기호와 같습니다.

```
let string = '문자열' + '연결 연산자';
```

논리(Boolean) 타입

논리형은 참 또는 거짓에 해당하는 논리값인 true, false를 의미합니다. (오직 이 두개 뿐이에요)

```
let boolean1 = true;
let boolean2 = false;
```

true, false를 직접 명해줘도 되지만 아래 예시처럼 false가 되는 예시를 넣어준다면 그 변수는 false가 됩니다.

```
let boolean = 10 < 5; //10이 5보다 작다는 건 틀렸으므로 변수 boolean은 false를 가질
```

undefined

값이 할당되지 않은 상태를 나타냅니다

변수를 선언했지만 값을 할당하지 않으면 자동으로 `undifined`가 됩니다.

```
let empty;  //선언만 했으니 값이 없음
```

null

null 자료형은 null 값 하나만 있으며 변수나 상수를 선언하고 의도적으로 선언한 공간을 비워둘 때 사용합니다.

```
let empty = null;
```

객체(object)

객체(object)는 자바스크립트의 핵심 자료형입니다. 앞서 언급한 기본 자료형을 제외하고, 자바스크립트에서 거의 모든 데이터와 자료구조는 객체라고 봐야합니다. 객체 자료형에서 파생되는 자료형으로는 배열, 객체 리터럴, 함수가 있습니다. 함수는 뒤에서 따로 다루고 지금은 배열과 객체 리터럴만 다루겠습니다.

배열

배열은 복수의 데이터를 정의할 수 있는 자료형입니다. 지금까지 배운 자료형은 한번에 하나의 데이터만 정의할 수 있었습니다. 예를 들어 학생 A, 학생 B의 의 국어, 영어, 수학, 과학 점

수를 각각 저장하는 변수를 만든다고 합시다. 기존 자료형으로는 다음과 같은 코드만 작성할 수 있었을 겁니다.

```
let A_korean = 80;
let A_english = 70;
let A_math = 90;
let A_science = 75;

let B_korean = 75;
let B_english = 90;
let B_math = 80;
let B_science = 85;
```

그러나 배열을 사용하면 하나의 자료형에 여러개의 데이터를 정의할 수 있습니다.

```
let A_score = [80, 70, 90, 60];
let B_score = [75, 90, 80, 85]; ///국어, 영어, 수학, 과학 순
```

이렇게 배열로 정의한 데이터를 요소라고 합니다. 그리고 배열 요소에 접근하려면 **인덱스**를 이용합니다. 인덱스는 배열에서 각 데이터가 있는 위치를 가리키는 숫자라고 생각하면 됩니다. 인덱스는 다른 언어들과 동일하게 0부터 시작합니다. 따라서 영어 점수에 접근하고 싶다면 다음와 같이 코드를 작성하면 됩니다.

```
let A_score = [80, 70, 90, 60];
let B_score = [75, 90, 80, 85]

console.log(A_score[1]);
```

객체 리터럴

객체 리터럴은 객체를 정의하는 가장 간단한 방법입니다. 객체 리터럴은 객체를 정의할 때 중괄호{}를 사용하며, 중괄호 안에 key와 value의 한쌍으로 이루어진 속성이 들어갑니다.

객체 리터럴은 배열보다 장점을 많이 가지고 있는데, 특히 키로 값을 구분할 수 있다는 점입니다. 배열은 하나의 자료형에 여러 데이터를 담을 수는 있지만 배열에 담긴 값들이 어떤 값인지 명확하게 알 수 없다는 단점이 있습니다. 막상 앞선 코드처럼 어떤 학생의 점수를 국어,

영어, 수학, 과학 순서로 담은 배열이라 해도 이를 명시해놓지 않는다면 다른 이들은 알 수 없겠죠.

하지만 객체 리터럴은 다릅니다. 배열처럼 여러 값을 하나로 묶을 수 있지만 값을 키로 구분해줍니다.

```
let A_score = {
    korean : 80,
    english : 70,
    math : 90,
    science : 60
};

let B_score = {
    korean : 75,
    english : 90,
    math : 80,
    science : 85
};

console.log(B_score.english);
console.log(B_score['english']
console.log('A의 수학 점수는' + A_score.math + '이고' + '과학 점수는' + A_score.s
```

객체 리터럴은 각 값에 접근할 때도 배열처럼 대괄호를 사용하고 마침표도 사용할 수 있습니다.

5. 객체 구조 분해

앞서 객체에 대해 배울 때 항상 A_score.math 처럼 객체 내부의 값을 조회할 때마다 그 `객체명.` 을 입력해줘야했습니다. 한두개일 때는 상관이 없겠지만, 불어와야할 내용이 많을수록 불편할 겁니다. 이때 객체 구조 분해(비구조화 할당)이라는 문법을 사용하면 코드를 더욱 짧고 보기 좋게 작성할 수 있습니다.

객체 구조 분해

```
let A_score = {
    name : 'A',
    korean : 80,
    english : 70,
    math : 90,
    science : 60
};

let B_score = {
    name : 'B',
    korean : 75,
    english : 90,
    math : 80,
    science : 85
};

function print_score(score) {
    const {name, korean, science, math} = score;
    const text = name + '의 수학 점수는 ' + math + '이고 과학 점수는 ' + science +
    console.log(text);
};

print_score(A_score)
print_score(B_score)
```

이렇게 구조분해를 사용한다면 같은 속성을 가지고 있는 여러개의 객체를 출력해야하는 상황에서도 각각 따로 정의하지 않아도 된다는 장점이 있습니다.

6. 연산자

위에서 변수나 상수에 데이터를 할당할 때 = 기호를 사용했죠? 이처럼 어떤 연산을 처리하는데 사용하도록 미리 정의된 기호를 자바스크립트에서는 **연산자(operator)**라고 합니다. = 기호는 데이터를 할당하는 연산을 수행하기 때문에 **할당 연산자**라고 합니다. 이외에도 자바스크립트에는 많은 연산자가 있습니다.

산술 연산자

흔히 수학시간에 배운 사칙연산 그대로입니다.

- `+` 덧셈
- `-` 뺄셈
- `*` 곱셈
- `/` 나눗셈
- `%` 값을 나누고 나머지만
- `**` 거듭제곱

```
const x = 3
const y = 2

console.log(x + y) // 5
console.log(x - y) // 1
console.log(x * y) // 6
console.log(x / y) // 1.5
console.log(x % y) // 1
console.log(x**y)  // 9
```

변수에 1을 더하거나 빼는 연산의 축약형

```
n++
// n = n + 1 과 같은 의미

n--
// n = n - 1 과 같은 의미
```

비교 연산자(Equality Operators)

비교 연산자는 피연산자를 비교한뒤 논리형 값인 true, false를 반환하는 연산입니다.

- `==`: 값이 동일한지를 비교합니다. 데이터 유형 변환을 수행할 수 있습니다.

```
console.log(5 == 5);   // true
```

```
console.log("5" == 5); // true, 문자열 "5"가 숫자 5로 변환됨
console.log(5 == '5'); // true, 숫자 5가 문자열 "5"로 변환됨
console.log(5 == 10);  // false
```

- `!=`: 값이 다른지를 비교합니다. 데이터 유형 변환을 수행할 수 있습니다.

```
console.log(5 != 10);  // true
console.log("5" != 5); // false, 문자열 "5"가 숫자 5로 변환됨
console.log(5 != '5'); // false, 숫자 5가 문자열 "5"로 변환됨
```

- `===`: 값과 데이터 유형이 모두 동일한지를 비교합니다.

```
console.log(5 === 5);   // true
console.log("5" === 5); // false, 데이터 유형이 다름
console.log(5 === '5'); // false, 데이터 유형이 다름
console.log(5 === 10);  // false
```

- `!==`: 값 또는 데이터 유형이 다른지를 비교합니다.

```
console.log(5 !== 10);  // true
console.log("5" !== 5); // true, 데이터 유형이 다름
console.log(5 !== '5'); // true, 데이터 유형이 다름
console.log(5 !== 5);   // false
```

- `>`: 왼쪽 피연산자가 오른쪽 피연산자보다 큰지를 비교합니다.

```
console.log(5 > 3); // true
console.log(3 > 5); // false
```

- `<`: 왼쪽 피연산자가 오른쪽 피연산자보다 작은지를 비교합니다.

```
console.log(5 < 3); // false
console.log(3 < 5); // true
```

- `>=`: 왼쪽 피연산자가 오른쪽 피연산자보다 크거나 같은지를 비교합니다.

```
console.log(5 >= 5); // true
console.log(5 >= 3); // true
console.log(3 >= 5); // false
```

- `<=`: 왼쪽 피연산자가 오른쪽 피연산자보다 작거나 같은지를 비교합니다.

```
console.log(5 <= 5); // true
console.log(5 <= 3); // false
console.log(3 <= 5); // true
```

논리 연산자()

논리 연산자는 피연산자를 논리적으로 평가한 뒤 조건에 맞는 피연산자를 반환하는 연산을 수행합니다.

논리 연산자에는 한 가지 특징이 있는데, 어떤 피연산자든 상관 없이 모두 논리형으로 평가한다는 것입니다. 숫자형, 문자열 모두 논리 값으로 평가합니다. 따라서 자바스크립트 자료형 중 “”(빈 문자열). undefined, 0, null만 거짓이고 **나머지는 참으로** 평가됩니다.

- `&&`: 왼쪽부터 평가해 거짓으로 평가된 피연산자를 즉시 반환, 만약 모든 피연산자가 참이라면 마지막에 평가한 피연산자 반환

```
true && true;  //true
true && false && true;  //false
"" && "cat";   //""
```

```
"cat" && null  //null
"cat" && "dog"  //"dog"
```

- `||` : 왼쪽부터 평가해 참으로 평가된 피연산자를 즉시 반환, 만약 모든 피연산자가 거짓이라면 마지막에 평가된 피연산자 반환

```
false || false;  //false
true || false || true;  //true
false || true || false;  //true
"" || "cat";   //"cat"
"cat" || null   //"cat"
"dog" || "cat"  //"dog"
```

- `!` : 무조건 참인건 거짓으로, 거짓인 건 참으로 반환

```
!false;  //true
!(10 < 20);  //false
!(10 < 20 && 20 < 10);  //true
```

7. 조건문

우리는 인생을 살아가면서 많은 선택을 합니다.

예를 들어 외출할 때는 날씨를 보고 우산을 가져갈지 말지를 결정하죠.

프로그래밍에서도 똑같이 이런 선택의 과정이 필요한데 그것이 바로 **조건문**입니다.

if 문

조건이 참일 때 중괄호 안의 코드를 실행

```
if (조건) {
// 조건이 참일 때 실행되는 코드
}
```

> "만약 비가 온다면, 우산을 가져가라"

```
//예시
if (비옴) {
      우산가져가기();
}
```

else 문

if문 뒤에 추가하여, 조건이 거짓일 때 else 블록 안의 코드를 실행

```
if (조건) {
// 조건이 참일 때 실행되는 코드
} else {
// 조건이 거짓일 때 실행되는 코드
}
```

> "만약 비가 온다면, 우산을 가져가라. 그렇지 않다면, 가벼운 옷차림을 하라."

```
//예시
if (비옴) {
      우산가져가기();
} else {
      가벼운옷차림();
}
```

else if 문

여러 조건을 차례로 검사 가능

```
if (조건1) {
  // 조건1이 참일 때 실행되는 코드
} else if (조건2) {
  // 조건2가 참일 때 실행되는 코드
} else {
```

```
  // 모든 조건이 거짓일 때 실행되는 코드
}
```

> "만약 비가 온다면, 우산을 가져가라. 그렇지 않고 바람이 분다면, 자켓을 입어라. 그 외에는 가벼운 옷차림을 하라."

```
if (비가온다) {
  우산가져가기();
} else if (바람분다) {
  자켓입기();
} else {
  가벼운옷차림();
}
```

8. 입력 받아보기

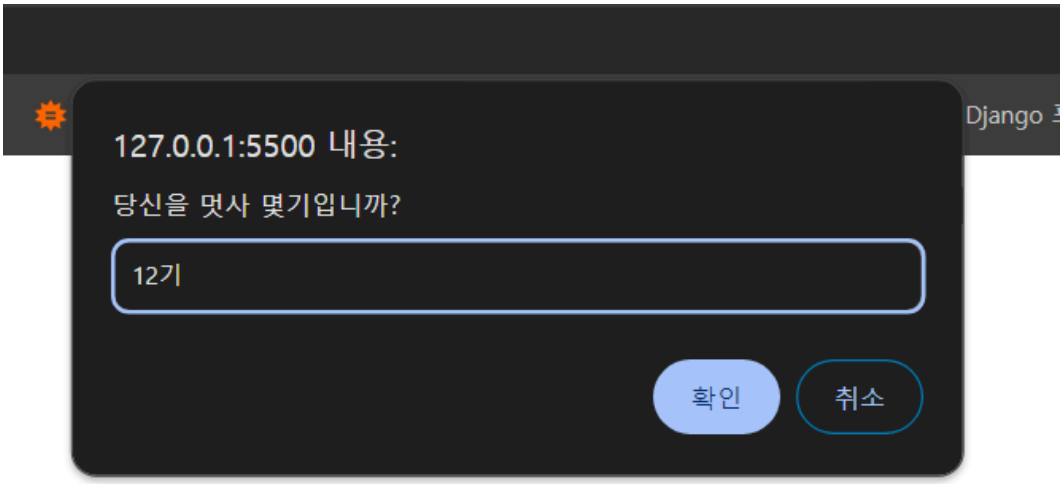
prompt() :

사용자에게 창을 띄워 데이터를 입력받을 수 있는 함수

```
prompt(message, default);
```

- `message` : 사용자에게 표시될 메시지를 나타내는 문자열입니다. 선택 사항입니다.
- `default` : 입력 상자에 표시될 기본값으로, 선택 사항입니다.

```
prompt("당신을 멋사 몇기입니까?", "13기");
```



이렇게 창이 떠요!

여기에 사용자가 입력하게 되면 그 값은 문자열 형식으로 반환돼요.

```
let 기수 = prompt("당신을 멋사 몇기입니까?", "12기");
```

이렇게 반환되는 값을 받으면 됩니다.

조건문과 더불어 자바스크립트의 핵심문법으로는 반복문이 있습니다. 반복문은 지정한 조건이 참으로 평가되는 동안 지정한 블록문을 반복해서 실행하는 문법입니다. 반복문에는 대표적으로 while, for문이 있습니다.

9. 반복문

while 반복문

while 반복문은 조건이 참인 동안에 계속해서 반복을 수행합니다. 반복 횟수를 정확히 모를 때 주로 사용됩니다.

구문:

```
while (조건식) {
  // 조건식이 참이면 실행
}
```

예시:

```
let num = 0;
while (i < 5) {
    console.log(num); // 0, 1, 2, 3, 4
    num ++;
}
```

do ...while 반복문

do ...while 반복문은 특정 조건이 참으로 평가되는 동안 do 다음에 오는 블록문을 반복 실행합니다.

while문은 블록문을 수행하기 전에 조건식을 평가하는 반면 do ...while 조건문은 블록문을 한번 수행한 후에 조건식을 평가한다는 특징이 있습니다.

구문:

```
do{
//블록문
}while(조건식);
```

예시:

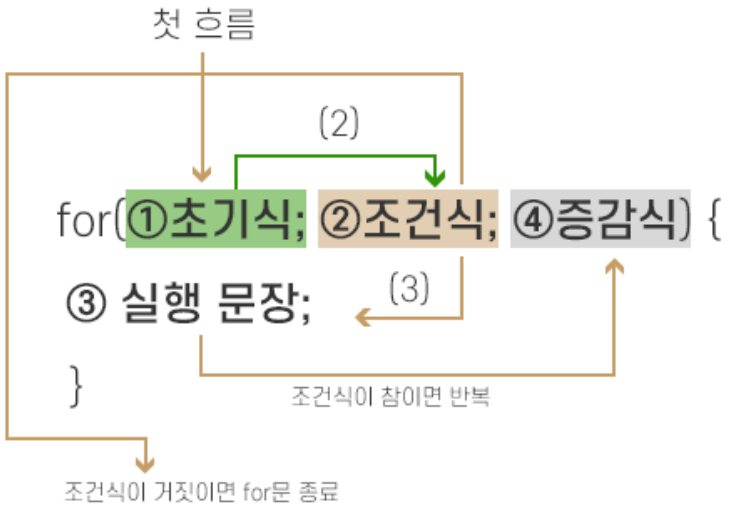
```
do{
console.log("무조건");
console.log("한번은 실행");
}while(false);

// 무조건
// 한번은 실행
```

코드를 실행해보면 while문의 조건이 항상 false 임에도 do 블록문이 한번은 실행되는 것을 확인 하 수 있을 겁니다.

for 반복문

for 문은 횟수를 지정해 지정한 횟수가 끝날 때까지 혹은 배열을 다 순회할 때까지 블록문을 반복실행하는 반복문입니다. 일반적으로 반복 횟수를 알고 있는 경우에 사용됩니다.



for 문은 **초깃값 → 조건식 → 실행문(조건문이 참일 경우) → 증감식 → 조건식** 순서로 실행됩니다.

이때 초깃값은 최초 1회만 실행됩니다.

구문:

```
for (초기화식; 조건식; 증감식) {
   // 실행될 코드
}
```

예시:

```
for (let i = 0; i < 5; i++) {
   console.log(i); // 0, 1, 2, 3, 4
}
```

> 주머니에서 열쇠를 찾는 상황

```
const 주머니 = [휴대폰, 지갑, 열쇠, 시계];

for (let i = 0; i < 4; i++) {
      console.log(주머니[i] + "를 보고 있어요!");
      if(주머니[i] === "열쇠") {
```

```
            console.log("열쇠를 찾았어요!");
        }
    }
```

Break/Continue

반복문 안에서는 `break` 와 `continue` 를 통하여 반복문에서 벗어나거나, 그 다음 루프를 돌게 끔 할 수 있습니다.

```
for (let i = 0; i < 10; i++) {
  if (i === 2) continue; // 다음 루프를 실행
  console.log(i);
  if (i === 8) break; // 반복문을 끝내기
}
```

10. 함수

자바 스크립트에서 함수는 함수 선언문, 함수 표현식, 화살표 함수 등을 사용해 정의합니다.

함수 선언문으로 함수 정의하기

함수 선언문은 function 키워드로 함수를 정의하는 방법입니다. function 키워드 다음에 함수를 식별할 수 있도록 식별자만 붙이면 됩니다. 식별자 뒤에는 소괄호()를 붙입니다.

구문:

```
fuction 식별자(){}
```

예시:

```
fuction gugudan(){
    for(let i = 1;, i<=9, i++){
        console.log(`3 * ${i] = ${3 * i}`);
    }
}
```

함수 표현식으로 함수 정의하기

자바스크립트에서는 함수도 변수에 할당할 수 있는데 이를 이용한 함수 정의 방법을 함수 표현식이라 합니다. 함수 표현식으로 변수에 할당하는 함수에 식별자가 있으면 네이밍 함수(naming function), 없으면 익명함수(anonymous function)로 다시 구분합니다.

구문:

```
const 변수명 = fuction (){};  //익명 함수
const 변수명 = fuction 식별자(){};  //네이밍 함수
```

예시:

```
const gugudan = fuction gugudan(){
    for(let i = 1;, i<=9, i++){
        console.log(`3 * ${i] = ${3 * i}`);
    }
};

gugudan();  //함수 호출
```

단 함수 선언문과는 다르게 fuction 키워드 다음에 오는 식별자로 함수를 호출하지 않고, 할당한 변수명으로 호출하기 때문에 function 키워드 다음에 식별자가 없는 익명함수도 함수를 호출하는데 아무 문제가 없습니다.

```
const gugudan = fuction(){
    for(let i = 1;, i<=9, i++){
        console.log(`3 * ${i] = ${3 * i}`);
    }
};

gugudan();  //함수 호출
```

만약 function 키워드 다음에 오는 함수 식별자로 호출하면 어떻게 될까요?

```
const gugudan = fuction naming(){
    for(let i = 1;, i<=9, i++){
        console.log(`3 * ${i] = ${3 * i}`);
    }
};
```

```
naming();  //함수 호출 => 오류발생
```

naming 변수가 정의되지 않았다는 참조 오류가 납니다. 따라서 함수 표현식으로 함수를 정의할 때는 function 키워드 다음에 오는 식별자가 아니라 **변수명으로** 호출해야 한다는 점을 유의해주세요

함수 표현식으로 함수를 정의할 때는 const 키워드를 주로 사용합니다!

화살표 함수로 함수 정의하기

화살표 함수를 화살표를 사용해 함수를 정의하는 방법으로, 조금 낯설어 보일 수 있지만, 현재는 가장 많이 사용되는 방법입니다 ^_^

화살표 함수는 익명 함수로만 정의할 수 있어서 함수를 호출하려면 정의된 함수를 변수에 할당하는 방법인 함수 표현식을 사용해야 합니다.

구문:

```
() => {};
```

예시:

```
const gugudan = () => {
   for(let i = 1;, i<=9, i++){
      console.log(`3 * ${i] = ${3 * i}`);
   }
};

gugudan();  //함수 호출
```

11. spread / rest

이번에는 ES6에서 도입된 spread와 rest문법에 대해서 알아보겠습니다.

Spread

spread라는 단어 자체에서 볼 수 있듯이 이 문법을 사용하면 객체, 혹은 배열을 펼칠 수 있습니다.

다음과 같은 객체들이 있다고 가정해보겠습니다.

```
const lion = {
   name : '사자'
};

const bravelion = {
   name : '사자',
   attribute : 'brave'
};

const bravelikelion = {
   name : '사자',
   attribute : ' brave',
   color : 'yellow'
};

console.log(lion);
console.log(bravelion);
console.log(bravelikelion);
```

이 코드에서는 먼저 lion이라는 객체를 선언하고, bravelion이라는 객체를 만들었습니다. 기존에 선언한 lion은 건들지 않고 새로운 객체를 만들어서 lion이 가지고 있는 값을 그대로 사용하였습니다.

그리고 bravelikelion이라는 객체를 만들었는데요, 이 객체는 bravelion이 가지고 있는 속성을 그대로 사용하면서 color속성만 추가되었습니다.

이 코드들의 핵심은 기존의 것을 건들이지 않고, 새로운 객체를 만든다는 것인데요. 이런 상황에 사용할 수 있는 유용한 문법이 `spread` 입니다.

```
const lion = {
   name : '사자'
};

const bravelion = {
   ...lion
```

```
    attribute : 'brave'
};

const bravelikelion = {
    ...bravelion
    color : 'yellow'
};

console.log(lion);
console.log(bravelion);
console.log(bravelikelion);
```

위의 코드에서 사용한 `...` 문자가 바로 spread 연산자인데요. `...bravelion` 이라고 한다면 bravelion의 성질을 모두 가진다 라는 의미가 됩니다.

spread 연산자는 배열에서도 사용할 수 있습니다.

```
const animals = ['사자', '고양이', '강아지'];
const anotherAnimals = [...animals, '독수리'];
console.log(animals);
console.log(anotherAnimals);
```

Rest

이제 rest에 대해서 알아봅시다. rest는 spread와 비슷한데, 그 역할은 매우 다릅니다.

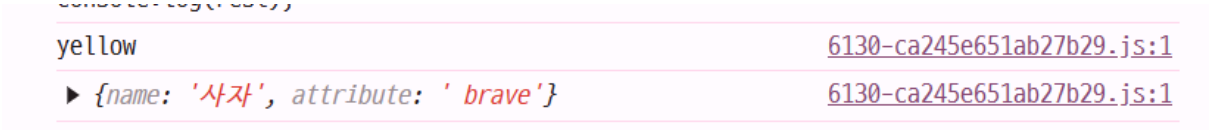
rest는 객체, 배열 그리고 함수의 파라미터에서 사용이 가능합니다.

마찬가지로 다음과 같은 배열이 있다고 가정합시다.

```
const bravelikelion = {
    name : '사자',
    attribute : ' brave',
    color : 'yellow'
};

const {color, ...rest} = bravelikelion;
```

```
console.log(color);
console.log(rest);
```



그렇다면 다음과 같이 color에 해당하는 yellow가 출력되고, rest에는 color을 제외한 나머지 속성들이 들어가 있습니다. 주로 rest라는 키워드로 사용하게 되는데 추출할 값의 이름이 꼭 rest일 필요는 없습니다,

```
const bravelikelion = {
   name : '사자',
   attribute : ' brave',
   color : 'yellow'
};

const {color, ...bravelion} = bravelikelion;
console.log(color);
console.log(bravelion);
```

이렇게 작성해도 무방합니다.

rest는 spread과 마찬가지로 배열에서도 사용 가능합니다.

```
const numbers = [0, 1, 2, 3, 4, 5, 6];

const [one, ...rest] = numbers;

console.log(one);
console.log(rest);
```

🌟 과제