

02. 변수와 자료형

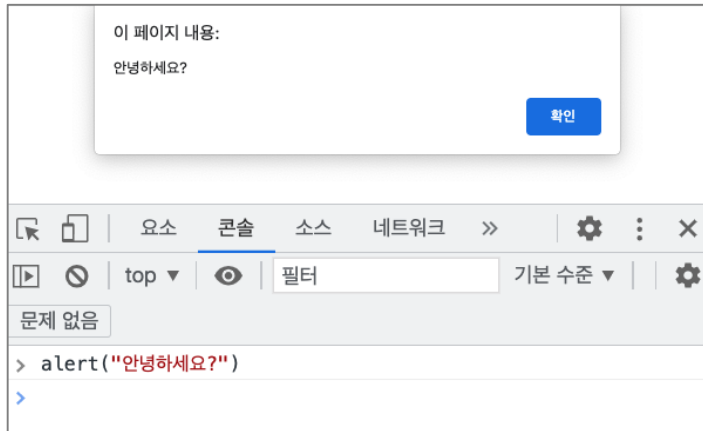
입력과 출력 방법

alert() 함수

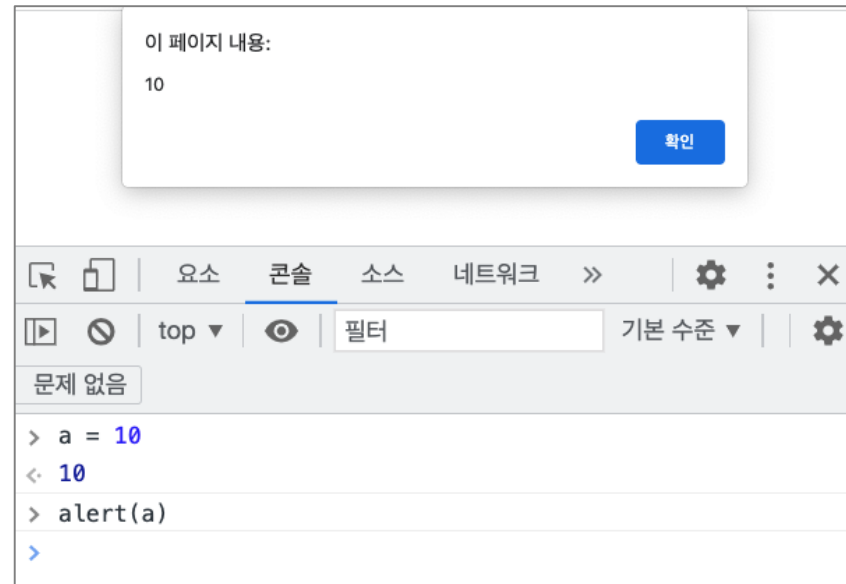
- 알림 창 표시 (앨럿 창이라고도 함)
- alert() 함수의 괄호 안에 메시지를 입력하거나 변수를 사용 → 알림 창에 텍스트나 변수값 표시

alert(내용)

alert("안녕하세요?")



a = 10
alert(a)

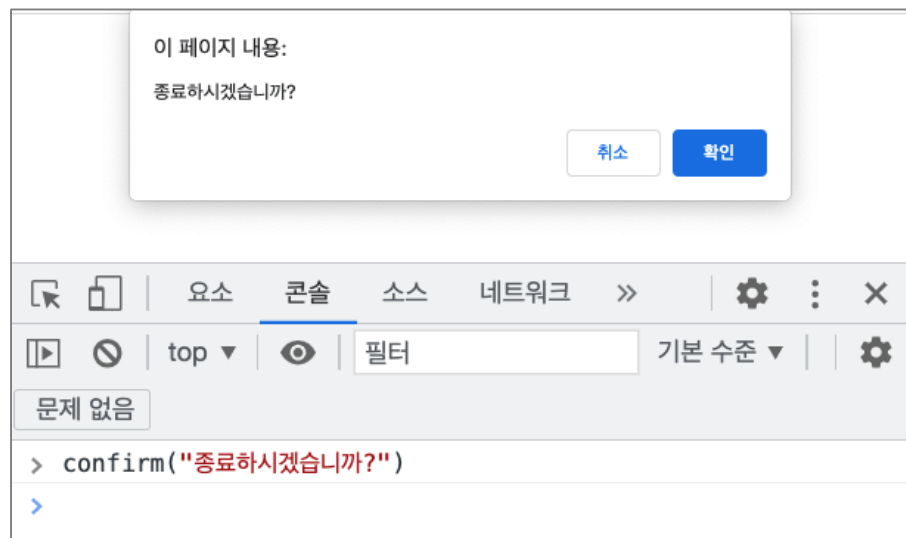


confirm() 함수

- 확인 창 표시 (컨펌 창이라고도 함)
- [확인] 버튼과 [취소] 버튼이 있어서 사용자가 어떤 버튼을 클릭했는가에 따라 다르게 동작하도록 할 수 있다

```
confirm(내용)
```

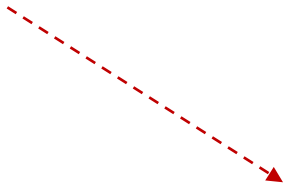
```
confirm("종료하시겠습니까?")
```



확인 창에서 [확인] 버튼을 누르면

```
> confirm("종료하시겠습니까?")  
< true  
>
```

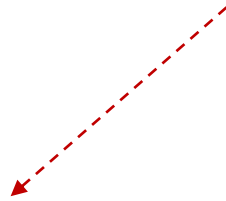
결괏값 true



확인 창에서 [취소] 버튼을 누르면


```
> confirm("종료하시겠습니까?")  
< false  
>
```

결괏값 false



결괏값을 확인하면 사용자가 [확인]을 눌렀는지 [취소]를 눌렀는지 알 수 있음
[확인]인지 [취소]인지에 따라 프로그램이 다르게 동작하도록 소스 작성할 수 있음

콘솔 창 팁

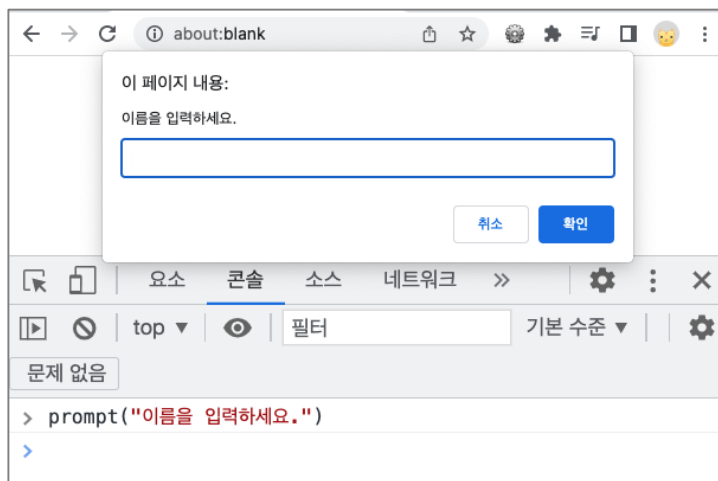
- 콘솔 창에 소스가 가득했을 때 내용을 지우려면 콘솔 창 위에 있는  클릭
- 이전에 입력했던 소스를 똑같이 입력하려면 콘솔 창에서 위로 화살표 또는 아래로 화살표 클릭
- 콘솔 창이 지워져도 이전에 입력했던 내용이 사라진 것은 아니므로 이전 소스를 찾아서 입력할 수 있음
- 콘솔 창에 나타나는 undefined는 오류가 아님.
 - 콘솔 창에서는 한번에 한 줄씩 명령을 실행한 후 그 결과를 콘솔 창에 표시함
 - 딱히 결괏값이 없는 명령을 실행했을 경우에는 결괏값 대신 undefined라고 표시함.
 - 예를 들어, alert() 함수는 화면에 창을 표시하고 나면 따로 결괏값이 없기 때문에 undefined라고 나타남.

prompt() 함수

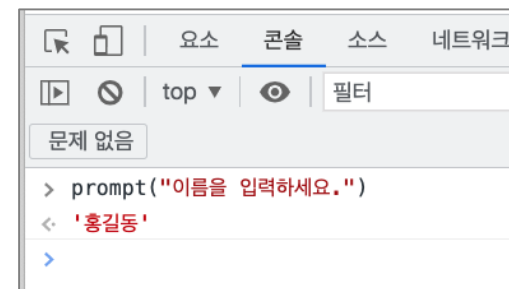
- 프롬프트 창 - 사용자가 간단한 값을 입력할 수 있는 창 표시
- 프로그램 실행에 필요한 값을 받을 때 자주 사용
- 기본 값을 지정하지 않으면 텍스트 필드가 빈 상태로 표시됨

prompt(내용) 또는 prompt(내용, 기본값)

`prompt("이름을 입력하세요.")`

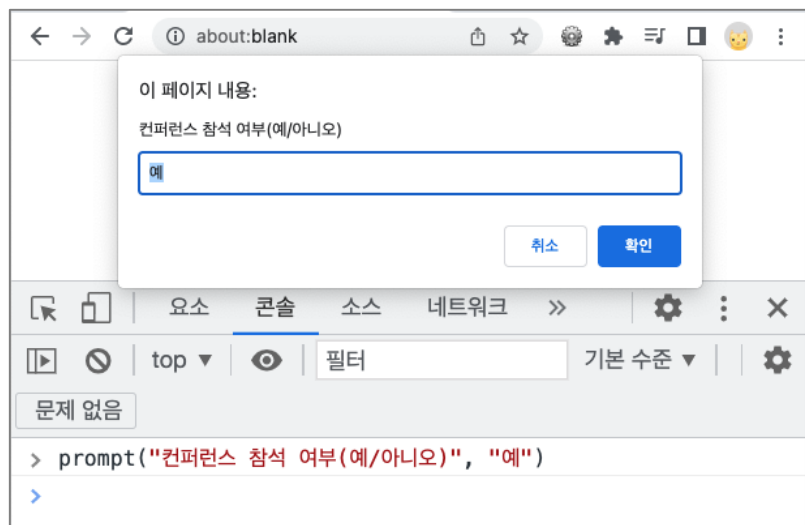


이름을 입력하고 [확인]을 누르면
입력한 내용이 결과값이 됨



사용자가 많이 입력할 것 같은 값을 기본 값으로

```
prompt("컨퍼런스 참석 여부(예/아니오)", "예")
```

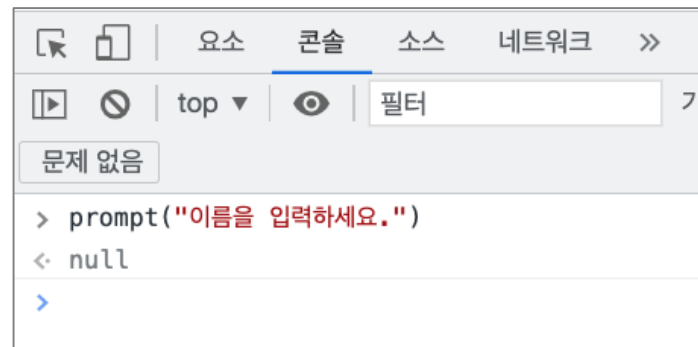


- 기본값을 사용한다면 [Enter]만 누르면 되기 때문에 편리함.
- 기본값을 지운 후 다른 내용을 입력해도 됨

```
prompt("이름을 입력하세요.")
```

프롬프트 창에서 입력하지 않고 [취소]를 누르면?

→ 결괏값 null



프로그램을 작성하면서 prompt() 함수를 사용할 때 사용자가 값을 입력했는지 확인하려면

prompt() 반환값이 null인지의 여부를 확인한다.

console.log() 함수

- 콘솔 창에 괄호 안의 내용을 표시함
- 자바스크립트 소스를 작성하면서 중간중간에 프로그램이 제대로 동작하는지 확인하는 용도로 자주 사용
- 콘솔 창에 결과를 표시하는 함수는 많지만 주로 console.log()를 많이 사용함
- 괄호 안에 텍스트나 변수를 사용할 수 있음

```
console.log(내용)
```

document.write() 함수

- 괄호 안의 내용을 웹 브라우저 화면에 표시함
- 실제 웹 브라우저 화면에 내용을 표시할 때에는 DOM을 이용하지만, 아직 DOM을 공부하지 않았기 때문에 일단 document.write() 함수 사용.
- document.write()문에서 연결 연산자(+)를 사용할 수도 있고, 템플릿 리터럴을 사용할 수도 있다.

```
document.write(내용)
```

변수부터 익히자

변수란 무엇일까

- 변수 : 프로그램에서 사용하기 위해 값을 담아놓는 바구니
(예) 날씨 정보를 알려 주는 프로그램이라면 지역이나 날짜 같은 값
- 일반적으로 변수는 프로그램 안에서 값이 달라질 수 있는 데이터를 가리킴.
- 하지만 프로그램 안에서 계속 값이 바뀌지 않더라도 변수로 만들어서 사용함(상수 변수라고 함)

변수 이름 지정하기

- 변수 이름을 지정하는 것은 값을 저장해 놓은 메모리 공간에 문패를 붙이는 것과 같다.
- 프로그램 안에서 사용할 값이 메모리의 어느 위치에 저장되어 있는지 신경쓰지 않고
문패 이름, 즉 값을 넣어놓은 변수 이름만 기억해 놓으면 됨
- 변수 이름을 쉽게 가져와서 그 안의 값을 사용할 수도 있고, 같은 위치에 바뀐 값을 저장할 수도 있음
- ➔ 따라서 변수 이름은 서로 다르게 만들어야 함

변수 이름 정하는 규칙

- 1) 변수 이름은 숫자로 시작할 수 없고
이름 안에 공백이 포함되어 있으면 안 된다

```
current, _current, $current // 사용 가능
```

- 2) 자바스크립트는 영문자의 대소문자를 구별한다

```
current, Current, CURRENT // 모두 다른 변수
```

- 3) 한 단어로 이루어진 변수를 사용할 때에는 주로 소문자를 사용

```
current, age, sum
```

- 4) 두 단어 이상으로 이루어진 변수는 언더바(_)로 연결하거나 중간에 대문자를 섞어 사용

```
current_year, total_area // 스네이크 표기법
```

```
currentYear, totalArea // 카멜 표기법
```

- 5) 자바스크립트에서 미리 정해 놓은 예약어(예: let 등)는 변수 이름으로 사용할 수 없음
- 6) 무의미한 변수 이름은 피한다

변수 선언 및 할당

1) 변수 선언: 키워드 `let`이나 `const` 다음에 변수 이름을 적어서 변수를 선언한다.

```
let 변수명;  
const 변수명;
```

`const`는 상수를 위한 예약어.

프로그램 안에서 바뀌지 않는 값(상수)을 변수에 담아놓고 사용함.

2) 변수에 값 할당 : 변수 오른쪽에 `=` 기호를 붙이고 오른쪽에 저장할 값이나 식을 작성한다.

```
변수명 = 값 또는 식;
```

3) 변수 선언과 값 할당을 동시에 할 수도 있다

```
let 변수명 = 값 또는 식;
```

1) let 키워드를 써서 result 변수를 선언하고 10이라는 값을 할당하기

```
let result;  
result = 10;
```

```
> let result = 10  
< undefined  
> result  
< 10  
>
```

2) 이미 만들어 놓은 result 변수에 20을 할당하기

```
> result = 20  
< 20  
> result  
< 20  
>
```

← let 변수에는 다른 값을 할당할 수 있다

1) const 키워드를 써서 number 변수를 선언하고 10이라는 값 할당하기

```
const number;  
number = 10;
```

또는

```
const number = 10;
```

```
> const number = 10  
< undefined  
> number  
< 10  
>
```

2) number 변수에 20 할당하기

```
number = 20;
```

오류 발생!!

```
> number = 20  
✖ ▶ Uncaught TypeError: Assignment to  
   constant variable.  
   at <anonymous>:1:8  
>
```

상수 변수에는 다른 값을 할당할 수 없다

상수 변수가 왜 필요할까?

값이 바뀌지 않는게 상수인데 프로그래밍에서 왜 상수를 변수로 만들어서 사용할까?

(예) 나이 계산 프로그램

올해 연도를 따로 `currentYear` 상수로 저장해 두면

프로그램 안에서 올댓값을 사용해야 할 때 `currentYear` 값 사용.

내년에 다시 이 프로그램을 사용한다면, `currentYear` 값만 바꿔주면 됨.

```
let age;
```

```
const currentYear = 2022;
```

```
age = currentYear - birthYear + 1;
```

(예) 웹 문서의 요소를 가져와 요소를 변형해야 할 경우

웹 문서 요소를 가져와서 상수 변수로 저장한 후 사용함

```
const button = document.querySelector("button");
```

```
button.onclick = hello;
```

버튼 클릭했을 때 `hello()` 함수 실행하는 소스

var와 let, const

ECMAScript 2015(ES6) 이전까지는 var를 사용해 변수를 선언했음

지금은 자바스크립트 역할이 커지면서 var로는 부족해서 let과 const를 사용하게 됨

키워드	선언하지 않고 사용하면?	재선언	재할당
var	오류없음	O	O
let	오류 발생	X	O
const	오류 발생	X	X

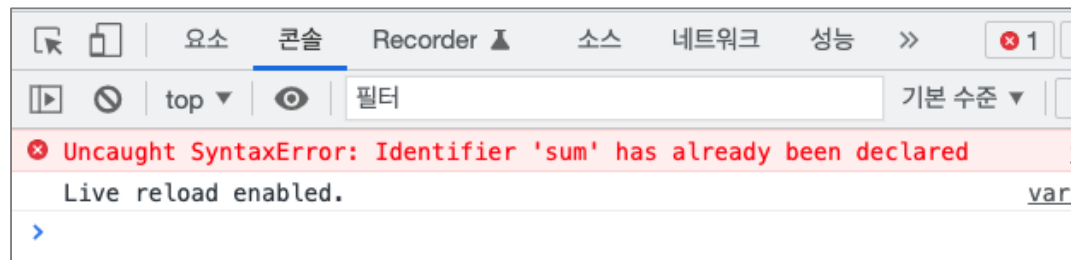
변수의 재선언과 재할당

var 변수는 재선언, 재할당 가능

```
function add(a, b) {  
  return a + b;  
}  
  
var sum = add(10, 20);  
console.log(sum);    // 30  
  
var sum = 100;  
console.log(sum);    // 100
```

let 변수는 재선언 안되고 재할당 가능

```
function add(a, b) {  
  return a + b;  
}  
  
let sum = add(10, 20);  
let sum = 100;  
console.log(sum);
```



변수의 재선언과 재할당

const 변수는 재선언도 재할당도 안됨

```
const myNumber = 10;  
  
myNumber = 50;  
console.log(myNumber);
```

✖ Uncaught TypeError: Assignment to variable-3.js:3
constant variable.
at variable-3.js:3:10

Live reload enabled. variable-3.html:39

> |

자바스크립트의 자료형

자료형(data type)이란

- 프로그램에서 처리하는 자료의 형태 (예) 3을 숫자로 처리할지, 문자로 처리할지
- 자료형, 자료 유형, 데이터 타입 등으로 부름
- 자바스크립트 자료형은 크게 '원시형'과 '객체'로 나눔
 - 원시형(primitive type) : 하나의 값만 가지고 있는 자료형.
 - 객체(object): 원시형 외의 모든 자료

자료형		설명
기본형	number(숫자)	따옴표 없이 표기한 숫자를 나타냅니다.
	string(문자열)	작은따옴표(')나 큰따옴표(")로 묶어 나타냅니다.
	boolean(논리형)	참(true)과 거짓(false)이란 두 가지 값만 가지고 있는 유형입니다.
	undefined	자료형을 지정하지 않았을 때의 유형입니다. 예를 들어 변수를 선언만 하고 값을 정의하지 않으면 undefined가 됩니다.
	null	값이 유효하지 않을 때의 유형입니다.
복합형	array(배열)	하나의 변수에 여러 값을 저장하는 유형입니다.
	object(객체)	함수와 속성이 함께 포함된 유형입니다.

typeof() 함수

자바스크립트 안에 미리 만들어져 있는 함수로,
괄호 안에 값이나 변수를 넣으면 어떤 자료형인지 알려준다.

```
typeof(값 또는 변수)
```

```
typeof("안녕하세요?")
```

```
let data = 5  
typeof(data)
```

숫자형(number)

- C나 자바 같은 프로그래밍 언어에서는 정수와 실수를 명확히 구별하고 정수도 크기에 따라 다른 자료형을 사용함.
- 하지만 자바스크립트에서는 정수와 실수를 함께 묶어서 숫자형이라고 함.
- (최근에 BigInt라는 자료형이 추가됨. 기존의 자바스크립트 숫자형의 한계를 넘는 큰 정수를 다루기 위한 자료형)
- 숫자라고 해도 따옴표(' ' 또는 " ")로 묶으면 숫자가 아닌 문자열로 인식함

```
typeof(10)      // 'number'  
typeof("10")    // 'string'  
typeof(3.145)   // 'number'
```

문자열(string)

- 작은따옴표(')나 큰따옴표(")로 묶은 데이터
- 큰따옴표이든, 작은따옴표이든 문자열의 앞뒤에 붙이는 따옴표는 같아야 함
- 최근에는 작은 따옴표(' ')를 많이 사용함

```
typeof("안녕하세요?")    // 'string'  
typeof("10")              // 'string'  
typeof("")               // 'string', 빈 문자열
```


특수 기호 표시하기

- 특수 기호를 표시하려면 백슬래시(\) 다음에 기호 사용
(예) 문자열이 아니라 순수하게 따옴표를 표시하고 싶다면 \" 처럼 써야 함

\\ddd (여기서 d는 숫자)	8진수 문자
\\xddd	16진수 문자
\\\\	백슬래시 문자
\\'	작은따옴표 문자
\\''	큰따옴표 문자

\\b	백스페이스 문자
\\f	폼 피드 문자
\\n	줄 바꿈 문자
\\r	캐리지 리턴 문자
\\t	탭 문자

```
console.log('I\\'m studying now  
.')  
console.log('탭 \\t 포함 ')
```

```
> console.log('I\\'m studying now');  
I'm studying now VM760:1  
< undefined  
> console.log('탭\\t 포함')  
탭 포함 VM783:1  
< undefined  
> |
```

템플릿 리터럴

- 문자열과 변수, 식을 섞어서 하나의 문자열을 만드는 표현 형식
- ES6 이전에는 +를 사용해서 식이나 변수와 연결했음 → 변수나 식이 많아질수록 오타가 나올 확률이 높다.
- 백틱(`)기호 사용 (백틱을 눌렀는데 #로 표시된다면 영문 상태로 바꾸고 백틱 입력)
- 변수나 식이 들어간다면 \${ } 로 묶고, 태그나 띄어쓰기, 이스케이프 문자를 그대로 표시할 수 있기 때문에 사용이 편리하다.

```
name = "백두산"  
classroom = 205  
console.log(`${name}님, ${classroom}호 강의실로 입장하세요.`)
```

변수 부분만 \${ } 로 묶어주고 원하는 결과 문자열을 그대로 사용하면 됨

논리형

- 참^{true}이나 거짓^{false} 값을 표현하기 위한 데이터 유형. 불린^{boolean} 유형이라고도 함.
- 사용할 수 있는 값은 true와 false
- 논리형 값은 지정한 조건을 체크하는 조건식에서 많이 사용한다.

truthy와 falsy

- true와 false 라는 명확한 값 외에 참과 거짓을 판별하는 방법
- truthy: true로 인정할 수 있는 값, falsy: false로 인정할 수 있는 값
- falsy 값을 제외한 모든 값은 truthy하다. 즉 true로 친다.

falsy 값

```
0      // 숫자 0
""     // 빈 문자열
NaN
undefined
null
```

- NaN은 숫자가 아님(Not a Number)을 나타낸다.
- 변수를 선언만 하고 값이 할당되지 않은 상태에서 그 변수를
더하거나 빼는 연산에 사용하면 NaN이 됨

undefined

- 변수를 선언하기만 하고
값을 할당할지 않을 때 변수의 초기값.
- undefined는 값이면서 동시에 자료형

```
let userName  
userName // undefined
```

d

null

- 유효하지 않은 값
- null 역시 값이면서 동시에 자료형

```
let age = null
```

undefined과 null

```
let first, second;  
second = null;  
console.log(first)  
console.log(second)
```

undefined	null
선언만 하고 할당하지 않음	null 값을 할당함
주로 사용자의 실수에 의해 발생	주로 사용자가 의도적으로 null을 할당

배열

- 하나의 변수에 여러 값을 할당할 수 있는 형태
- 대괄호([])로 묶고, 그 안에 값을 나열함. 각 값은 쉼표(,)로 구분
- 대괄호 안에 아무 값도 없으면 '빈 배열'이라고 하고, 이것 역시 배열

```
배열명 = [값1, 값2, ...  
]
```

```
emptyArr = []           // 빈 배열  
colors = ["red", "blue", "green"] // 문자열 배열  
arr = [10, "banana", true] // 여러 자료형으로 구성된 배열
```

배열과 인덱스

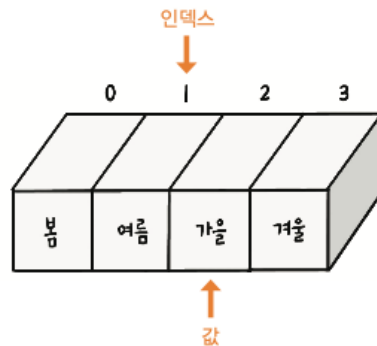
```
season = ["봄", "여름", "가을", "겨울"]
```

인덱스

```
> season = ["봄", "여름", "가을", "겨울"]  
< ▼ (4) ['봄', '여름', '가을', '겨울'] ⓘ  
0: "봄"  
1: "여름"  
2: "가을"  
3: "겨울"  
length: 4  
▶ [[Prototype]]: Array(0)
```

배열의 크기

변수 이름 → Season



- 인덱스 : 배열에 있는 여러 값을 저장하는 방 번호
- 인덱스는 0부터 시작!

- 두번째 값을 알고 싶다면 `season[1]`
- 배열에 있는 요소의 갯수를 알고 싶다면 `season.length`
- 배열의 마지막 값을 알고 싶다면 `season[season.length - 1]`

심볼

- ES6에 새롭게 추가된 원시 유형의 자료형
- 심볼의 가장 큰 특징은 유일성을 보장한다는 것
- 심볼은 객체 프로퍼티의 키로 사용할 수 있다

(예) 자바스크립트 프로그램에서 오픈 소스를 가져와 사용하거나
다른 팀원이 만든 객체들을 함께 사용할 경우 객체의 키 이름이 중복될 수도 있다.
➔ 키 이름을 심볼로 지정하면 서로 충돌이 발생하지 않는다.

심볼

- 심볼을 만들 때는 Symbol() 함수 사용
- 심볼은 한 번 만들면 변경할 수도 없고, 같은 값을 가진 심볼을 만들 수도 없다.

```
Symbol()
```

```
let var1 = Symbol()  
let var2 = Symbol()
```

var1과 var2는 똑같아 보이지만, 심볼은 유일한 값이기 때문에 두 변수는 같지 않다!

```
var1 === var2    // fa  
lse
```


- 심볼을 키로 사용할 때에는 [키]처럼 대괄호로 묶어서 표현
- 키에 접근할 때도 마침표가 아닌 대괄호 사용

예) member 객체를 만들면서 id 키를 고유하게 만들기

```
let id = Symbol()
const member = {
  name : "Kim",
  [id] : 12345
}
```

→

```
member          // {name: "Kim", Symbol(): 12345}
: 12345
member[id]      // 12345
```

다시 id 키를 지정하면?

```
member.id = 678
9
```

```
> member
< {name: 'Kim', id: 6789, Symbol(): 1235}
  id: 6789
  name: "Kim"
  Symbol(): 1235
  ▶ [[Prototype]]: Object
```

다시 심볼형 id 키를 지정하면?

```
id = Symbol();
member[id] = 555;
```

```
> member
< {name: 'Kim', id: 6789, Symbol(): 1235, Symbol(): 555}
  id: 6789
  name: "Kim"
  Symbol(): 1235
  Symbol(): 555
  ▶ [[Prototype]]: Object
```

자료형 변환

자바스크립트의 형 변환

- 자바스크립트는 다른 언어와 다르게 프로그램 실행 중에 자료형이 변환되는 언어
- 자동으로 형이 변환될 때에도 있다 → 이런 상황을 미리 알아 두지 않으면 오류를 발생시키기도 하고, 처음에 예상했던 것과 다른 결과가 나올 수도 있습니다.

C 언어나 자바 등 일반 프로그래밍 언어

- 변수를 선언할 때 변수의 자료형을 결정
- 자료형에 맞는 값만 변수에 저장 가능
- 자료형으로 인한 프로그램의 오류 방지 가능

```
int num = 20           // 정수형 변수
num
char *name = "John"    // 문자형 변수
name
```

자바스크립트

- 변수를 선언할 때 자료형 지정하지 않음
- 변수에 값을 저장할 때 자료형 결정
- 편리하긴 하지만 변수를 일관성 있게 유지하기 힘들다

```
num = 20               // 숫자형
num = "John"           // 문자열
```

자동 형 변환

- 변수에 값을 저장할 때 자료형이 결정되기도 하지만
- 연산을 할 때 자료형이 자동으로 변환된다. 주의해야 함!
- 문자열을 사칙 연산에 사용하면 자동으로 숫자형으로 변환됨
- 숫자와 문자열을 연결하면 숫자가 문자열로 변환됨

```
one = "20"    // 문자열  
two = 10      // 숫자형
```

```
one + two     // "2010"  
one - two     // 10
```

+ 연산자

- + 기호 앞이나 뒤에 문자열이 있으면 " 연결 연산자"
- + 기호 앞뒤에 숫자가 있으면 "더하기 연산자"

-, *, / 연산자

- 기호 앞이나 뒤에 문자열이 있으면 숫자로 인식함

프롬프트 창에서 값을 입력 받으면 그 값은 문자열

```
let userInput = prompt("아무 숫자나 입력하세요.")
```

```
typeof(userInput)           // 'string'
```

```
result = userInput * 10     // 1000
```

```
userInput                   // '10'
```

- userInput에 10을 곱하면 자동으로 숫자형으로 변환되면서 계산값이 result에 저장됨.
- 하지만 userInput 값은 계속 문자열인 상태.
- userInput이 숫자로 바뀌었다고 착각할 수 있음

프롬프트 창에서 숫자를 입력 받을 경우 직접 숫자로 변환한 후 연산에 사용하는 것이 좋다.

숫자형으로 변환하기 - Number()

문자열 뿐만 아니라 null과 undefined를 포함해서 모든 자료형을 숫자로 변환할 수 있다

Number() 함수의 변환 규칙

기존 유형	변환 결과
true	1
false	0
숫자	숫자
null	0
undefined	NaN
정수 문자열	정수(맨 앞에 0이 있으면 제거)
실수 문자열	실수(맨 앞에 0이 있으면 제거)
16진수 문자열	10진수
빈 문자열	0
위 상황 외	NaN

```
Number(true)    // 1
Number("20")    // 20
Number("Hi?")   // NaN
```

숫자형으로 변환하기 – parseInt(), parseFloat()

- parseInt() 함수: 괄호 안의 값을 정수로 변환
- parseFloat() 함수 : 괄호 안의 값을 실수로 변환

```
let userInput = parseInt(prompt("아무 숫자나 입력하세요."
));
```

```
let bodyHeat = prompt("현재 체온은?")
parseFloat(bodyHeat) // 36.4
```

문자열로 변환하기 – toString() 함수

null 데이터형과 undefined 데이터형을 제외한 데이터형을 문자열 데이터로 변환

원래값 뒤에 마침표를 붙이고 함수를 작성

숫자를 문자열로 변환할 때는 basis 옵션을 사용해 숫자가 10진수인지, 2진수인지 같이 지정.

```
값.toString()
```

```
값.toString(basis)
```

```
num = 10 // 원래값 숫자형
```

```
isEmpty = true // 원래값 논리형
```

```
num.toString() // '10', 10진수 문자열
```

```
num.toString(2) // '1010', 2진수 문자열
```

```
isEmpty.toString() // 'true'
```


문자열로 변환하기 – String() 함수

null 데이터형과 undefined 데이터형을 포함해서 문자열 데이터로 변환

String() 함수의 괄호 안에 값을 넣어서 변환

null이면 'null'로, undefined이면 'undefined'로 변환. 그 외에는 toString() 함수와 같다.

String(값)

```
isFull = false           // 원래값 논리형  
initValue = null         // 원래값 null형  
String(isFull)           // 'false'  
String(initValue)        // 'null'
```

논리형으로 변환하기 – Boolean() 함수

다른 유형의 데이터를 논리형 데이터로 변환
함수의 괄호 안에 원랫값을 넣는다.

```
Boolean(값)
```

논리형으로 변환할 때의 규칙

	true 값이 되는 데이터	false 값이 되는 데이터
숫자형	0이 아닌 값	0
문자열	빈 문자열이 아닌 모든 문자열	빈 문자열
undefined	-	undefined

```
Boolean(5 * 4)      // true
Boolean("Hi?")      // true
Boolean(undefined)  // false
```

[실습] 화씨 온도 -> 섭씨 온도 변환기

먼저 생각해 보기

- 화씨 온도를 섭씨 온도로 변환하는 공식은 무엇일까?
- 프롬프트 창에서 받은 값을 정수로 변환할까, 실수로 변환할까?

구글링해서 공식을 찾아보자

$$\text{섭씨 온도} = (\text{화씨 온도} - 32) / 1.8$$

실수로 나누는 부분이 있으니 결괏값도 실수일 수 있다.

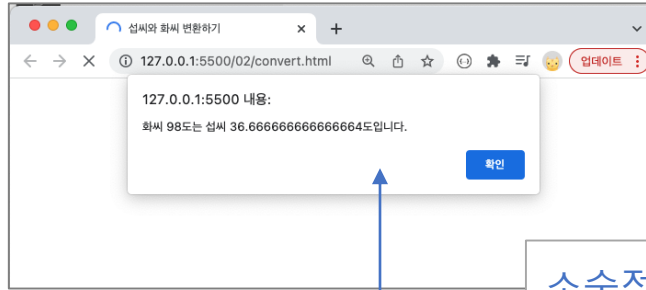
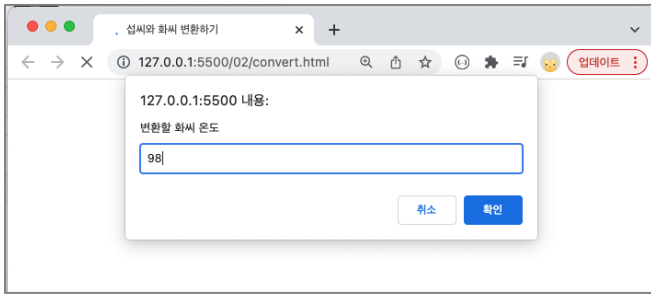
```
// 섭씨: c, 화씨: f
```

```
let f = parseFloat(prompt("변환할 화씨 온도"))  
);
```

```
let c;
```

```
c = (f - 32) / 1.8;
```

```
alert(`화씨 ${f}도는 섭씨 ${c}도입니다.`);
```



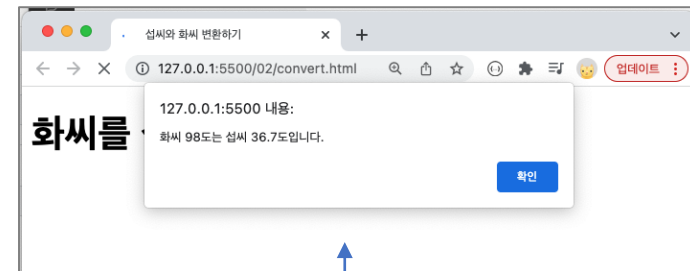
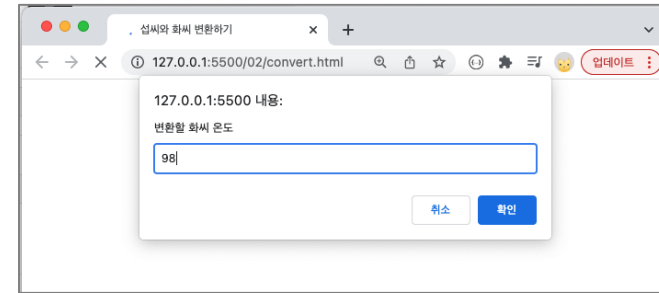
소숫점 이하 자리가 너무 많이 나타나는 경우가 생김
→ 소숫점 이하 자릿수를 지정하자

실수의 소수점 자릿수를 고정하려면 → toFixed() 함수 사용
값.toFixed(1)이나 값.toFixed(2)처럼 괄호 안에 자릿수만 지정하면 됨.

```
// 섭씨: c, 화씨: f

let f = parseFloat(prompt("변환할 화씨 온도"));
;
let c;

c = ((f - 32) / 1.8).toFixed(1);
alert(`화씨 ${f}도는 섭씨 ${c}도입니다.`);
```



소숫점 이하 한자리만 표시됨