JS기본-1

태그	
날짜	@2023년 5월 25일

https://codepen.io

변수

name = “mike”;

age = 30;

문자열(string)은 항상 “”, ‘’로 감싸주어야 한다.

class = “수업” 같은 경우 에러

class 같은 경우 js에서 예약어로 지정되어있기 때문에 에러가 발생한다.

alert()

console.log()

위 처럼 변수를 선언하는 것은 위험하다.

내가 name이라는 변수를 사용하고 다른 누군가 또 name이라는 변수를 사용했을 때 덮어씌워진다.

let 은 선언 후 다른 값으로 변경 할 수 있다. 의도적으로 let을 생략하고 변수명으로 선언을 하면 덮어씌워진다.

const 는 절대로 바뀌지 않는 상수

자바스크립트에서 변수를 선언할 때는, 변하지 않는 값은 const, 변할 수 있는 값은 let으로 선언

팁으론 모든 변수를 const로 선언 후 추후에 변경될 여지가 있는 변수를 let으로 변경

1. 변수는 문자와 숫자, $와 _만 사용

   ex)

   1. const MY_HOME = “…”;
   2. let_ = 1;
   3. let$ = 3;

2. 첫글자는 숫자가 될 수 없다.

   ex)

   1. let 1stGrade = ‘A+’; - 틀린 문법

3. 예약어는 사용할 수 없다.

   ex)

   1. let let = 99; - 틀린 문법

4. 가급적 상수는 대문자로 한다.

   ex)

   1. const MAX_SIZE = 99;

5. 변수명은 읽기 쉽고 이해할 수 있게 선언

   ex)

   1. let a = 1; x
   2. let userNumber = 1; o

자료형

const name = “mike”; // 자료형 String

const name1 = “mike”

const name2 = ‘mike’

const name3 = `mike`

문자형 안에 작은따옴표를 써야하는 경우 큰따옴표로 감싼다.

ex) const message = “I’m a boy.”;

작은따옴표로만 쓸 경우

const message2 = ‘ i\’m a boy.’; // \는 뒤에오는 기능을 삭제

const message3 = `My name is ${name}`;// 백틱은 그 문자 내부의 변수를 표현할 때 사용하면 편하다.

const message4 = `나는 ${30+1} 살 입니다.`;

바로 표현식을 넣을 수도 있다.

숫자형

const age = 30; // 숫자형 number

const PI = 3.14; // 소수점도 표현도 가능

숫자형은 사칙 연산도 가능하다.

console.log(1 + 2);

console.log(10 - 3);

console.log(3 * 2);

console.log(6 / 3);

console.log(6 % 4); // 나머지 값 반환

---

const x = 1/0; // ??
console.log(x) → Infinity(무한대) 반환

---

const name = "mike";
const y = name/2;
console.log(y); → NaN 반환

NaN (not a number) : 숫자가 아니다.

Boolean

논리적인 요소를 나타냄

```
// Boolean

const a = true; // 참
const b = false; // 거짓

const name = "Mike";
const age = 30;

console.log(name=='MIke'); //거짓
console.log(age > 30); //거짓
```

---

null & undefined

null 은 존재하지 않는 값

undefined는 값이 할당 되지 않았다는 것을 의미

```
// null 과 undefined

let age;
console.log(age);

undefined 출력
```

만약 변수에 null을 할당한다면 유저는 존재하지 않는다는 의미

```
// typeof 연산자

const name = 'mike';

console.log(typeof 3); // number
console.log(typeof name); // String
console.log(typeof true); // boolean
console.log(typeof "xxx"); // String
console.log(typeof null); // object
console.log(typeof undefined); // undefined
```

다른 개발자가 작성한 변수의 타입을 알아야 하거나 api 통신 들을 통해 받아온 데이터를 타입에 따라 다른 방식으로 처리해야 할 때 많이 사용된다.

typeof null 이 object가 나왔는데 “object”는 객체형을 의미한다.

사실 null은 객체가 아니다.

이건 자바스크립트 초기 버전의 오류 인데, 하위 호환성을 유지하기 위해 수정하지 않는다고 한다.

```
const name = 'mike';

const a = '나는 ';
const b = " 입니다.";

console.log(a + name + b);
// "나는 mike 입니다."

문자형과 문자형은 + 로 더해주면 하나의 문자로 합칠 수 있다.
숫자형과 문자형도 합칠 수 있는데 이때는 문자형으로 변경된다.

const age = 30;

console.log(a + age + "살"+ b);
// "나는 30살 입니다."
```

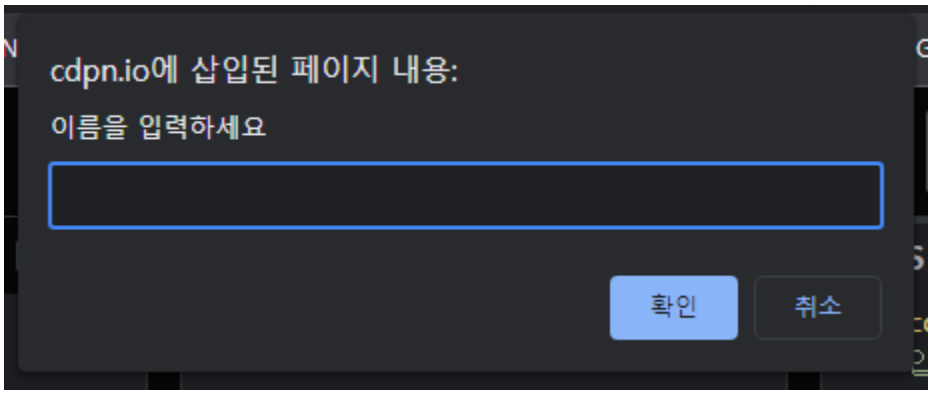
대화상자 alert, prompt, confirm

alert : 알림 메시지를 띄움 (일방적으로 알리는 용도)

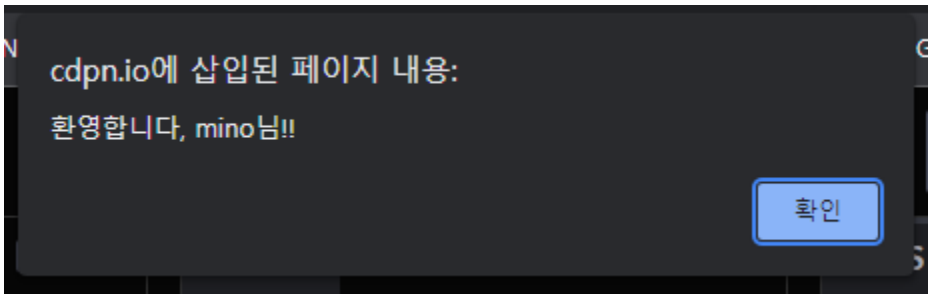
prompt : 입력

confirm : 확인

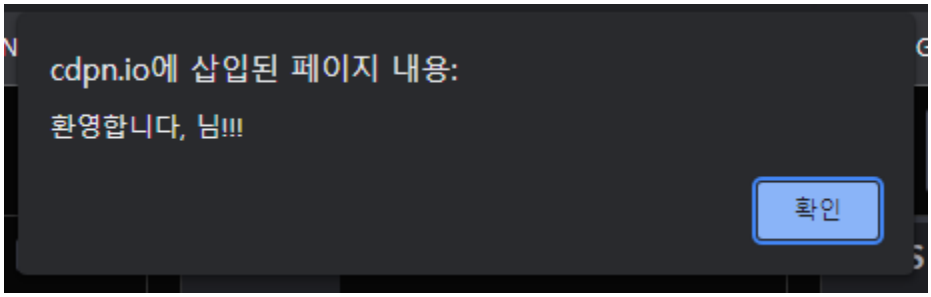
```
const name = prompt("이름을 입력하세요");

alert("환영합니다, " + name + "님!");
```



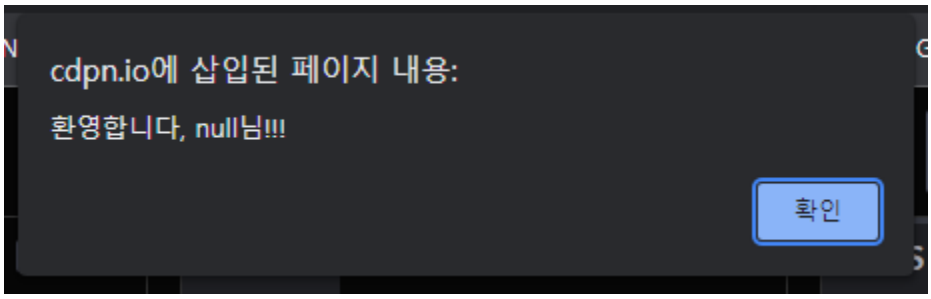
입력 시



빈칸으로 확인 눌렀을 때



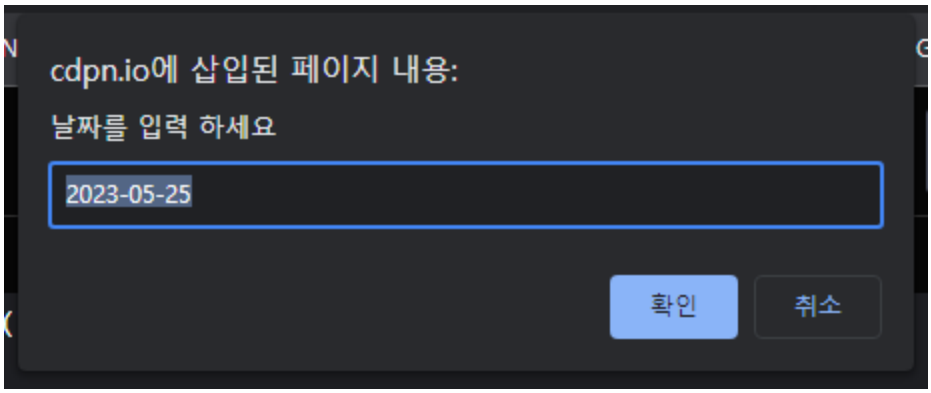
빈칸으로 취소 버튼 눌렀을 때



```
// 백틱으로 표현 시
const name = prompt("이름을 입력하세요");

alert(`환영합니다, ${name}님!!!`);
```

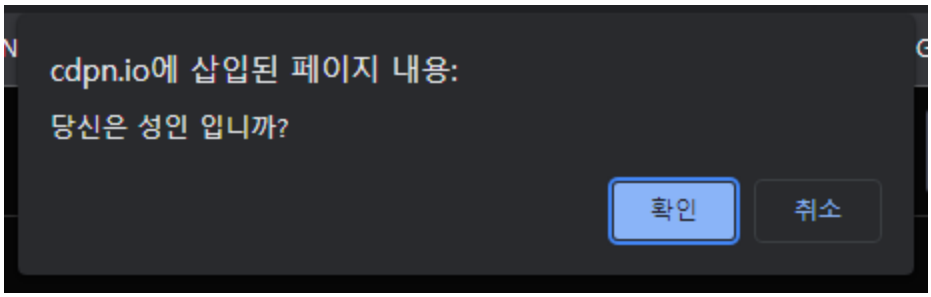
prompt 는 디폴트 값도 설정할 수 있다.

```
const name = prompt("날짜를 입력 하세요", "2023-05-25");

// 위 처럼 두개의 인수를 받을 수 있다.
// 함수를 실행하는 괄호 안에서 들어가는 값을 인수라고 한다.
```



confirm

```
const isAdult = confirm("당신은 성인 입니까?");
console.log(isAdult);

// 확인 - true
// 취소 - false
```



단점

1. 창이 떠있는 동안 스크립트가 일시 정지, 창을 닫기 전에 이후 동작이 제한 된다.
2. 스타일링이 불가능하다. 위치와 모양을 정할 수 있다.

이러한 단점이 있지만 기본 메서드는 빠르고 간단하게 적용 가능한 장점이 있다.

형변환

String() → 문자형으로 변환

Number() → 숫자형으로 변환

Boolean() → 불린형으로 변환

형변환이 왜 필요할까 ?

문자열 + 문자열 = 문자열

숫자열 + 숫자열 = 숫자열

문자열 + 숫자열 = ???

```
const mathScore = prompt("수학 몇점?");
const engScore = prompt("영어 몇점?");
const result = (mathScore + engScore) / 2;

console.log(result)
// 3540  이상한 값이 나왔다.


const mathScore = 80;
const engScore = 70;
const result = (mathScore + engScore) / 2;

console.log(result)
// 75

prompt 입력으로 받은 값은 문자형으로 들어온다.
즉, prompt 명령어로 숫자를 90 , 80 을 입력 받게 되면
result가 170이 아니라 9080이 된다.

그럼 9080은 문자열인데 왜 /2가 될까?
숫자형이 아니더라도 나누기 같은 표현식은 숫자형으로
자동으로 형변환되어 계산된다.
이런 것을 자동 형변환이라고 한다.
편리하다고 생각할 수 도 있지만 원인을 찾기 힘든 에러를
발생할 수 있기 때문에 항상 의도를 가지고 형변환 해주어야
한다. 이런것을 명시적 형변환 이라고 한다.
```

String

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

// "3" "true" "false" "null" "undefined"
// console.log는 , 으로 여러 값을 한번에 수행할 수 있다.
```

Number

```
console.log(
Number("1234"),
  Number("1234asdfa"),
  Number(true),
  Number(false),
  Number("sdfasdf")
)

//1234 NaN 1 0 NaN
```

Boolean

```
false
- 숫자 0
- 빈 문자열 ''
- null
- undefined
- NaN

위 이외에는 모두 true를 반환한다.

console.log(
  Boolean(1),
  Boolean(123),
  Boolean("JAVASCRIPT")
)
// true true true

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

주의사항

```
Number(null) // 0
Number(undefined) // NaN

Boolean 0은 false지만 "0"은 true 이다.
```

```
Boolean(0) //false
Boolean("0") // true

Boolean 빈문자열은 false이지만 공백이 들어가 있으면 true이다.
Boolean('') // false
Boolean(" ") // true
```

연산자(Operators)

- + - / * %

나머지(%)를 어디에 쓸까?

홀수 : X % 2 = 1

짝수 : Y % 2 = 0

5보다 작은 수를 얻고 싶은 경우

X % 5 = 0 ~ 4 사이의 값만 반환

거듭 제곱

const num = 2 ** 3;

console.log(num); // 8

연산자는 우선순위 가 있다.

몇몇 연산자는 줄여서 쓸 수 있다.

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

console.log(num); // 15

이걸 줄여서 쓰면

let num = 10;
num += 5;
// num *=5;
// num -= 5;
// num %= 5;
// num /= 5;
// 다른 연산자도 동일하게 쓸 수 있다.

console.log(num);
```

증감 연산자, 감소 연산자

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

console.log(num); //11

let num = 10;
num --;

console.log(num); // 9
```

변수 앞에 쓰냐 뒤에 쓰냐 차이가 있는데

```
let num = 10;
let result = num++ // 뒤에
console.log(result); // 10
// 증가 되지 않음
-----------------------------------
let num = 10;
let result = ++num; // 앞에
console.log(result); // 10
// 증가됨

뒤에 적으면 증가되기 전값을 result에 넣게 된다.
앞에 적으면 증가시킨 값을 result에 넣게 된다.
```