

HTML5 + CSS3 + JavaScript

로 배우는 웹프로그래밍 기초

개정
2판



제8장 자바 스크립트 객체와 배열

객체 개념

- 현실 세계는 객체들의 집합
 - 사람, 책상, 자동차, TV 등
 - 객체는 자신만의 고유한 구성 속성
 - 자동차: <색상:오렌지, 배기량:3000CC, 제조사:한성, 번호:서울1-1>
 - 사람: <이름:황기태, 나이:20, 성별:남, 주소:서울>
 - 은행계좌: <소유자:황기태, 계좌번호:111, 잔액:35000원>



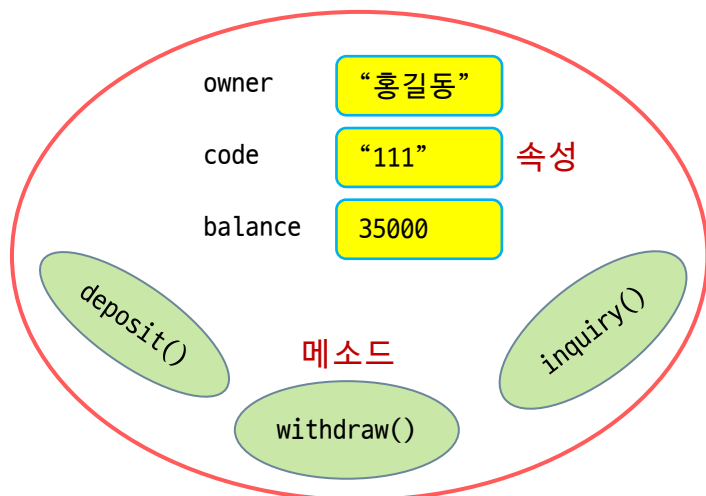
자동차 객체(car)



은행 계좌(account)

자바스크립트 객체

- 자바스크립트 객체 구성
 - 여러 개의 속성(property)과 메소드로 구성
 - 속성 : 객체의 고유한 변수
 - 메소드(method) : 함수



자바스크립트 객체 **account**

```
let account = {  
  owner    : " 홍길동",  
  code     : "111",  
  balance  : 35000,  
  deposit  : function() { ... },  
  withdraw : function() { ... },  
  inquiry  : function() { ... }  
};
```

account 객체를 만드는 자바스크립트 코드

자바스크립트에서의 객체

- 자바스크립트는 객체 기반 언어
 - 자바스크립트는 객체 지향 언어 아님
- 자바스크립트 객체의 유형
 1. **핵심 객체**
 - 자바스크립트 언어가 실행되는 어디서나 사용 가능한 기본 객체
 - 기본 객체로 표준 객체
 - **Array, Date, String, Math** 타입 등
 - 웹 페이지 자바스크립트 코드에서 혹은 서버에서 사용 가능
 2. **HTML DOM(Document Object Model) 객체**
 - HTML 문서에 작성된 각 HTML 태그들을 객체화한 것들
 - HTML 문서의 내용과 모양을 제어하기 위한 목적
 - W3C의 표준 객체
 3. **브라우저 객체**
 - 자바스크립트로 브라우저를 제어하기 위해 제공되는 객체
 - BOM(Browser Object Model)에 따르는 객체들
 - 비표준 객체

코어 객체

- 코어 객체 종류
 - Array, Date, String, Math 등
- 코어 객체 생성
 - new 키워드 이용

```
let today = new Date(); // 현재 날짜가 저장된 Date 객체 생성
let books = new Array("HTML", "Java", "C++"); // 배열이 생성
let msg = new String("Hello"); // 문자열 객체 생성
```

- 객체가 생성되면 객체 내부에 속성과 메소드들 존재
- 객체 접근
 - 객체와 멤버 사이에 점(.) 연산자 이용

```
obj.속성 = 값; // 객체 obj의 프로퍼티 값 변경
변수 = obj.속성; // 객체 obj의 프로퍼티 값 알아내기
obj.메소드(매개변수 값들); // 객체 obj의 메소드 호출
// today.getDate()
```

자바스크립트 객체 생성 및 활용

Js_object1.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title>객체 생성 및 활용</title>
</head>
<body>
<h3>객체 생성 및 활용</h3>
<hr>
<script>
  // Date 객체 생성
  let today = new Date();

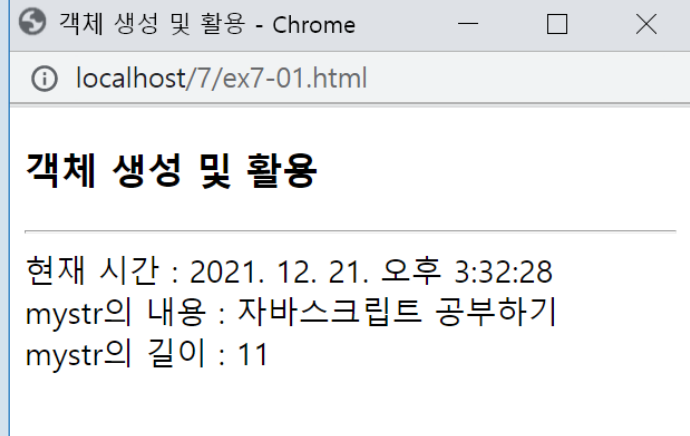
  // Date 객체의 toLocaleString() 메소드 호출
  document.write("현재 시간 : " + today.toLocaleString() + "<br>");

  // String 객체 생성
  let mystr= new String("자바스크립트 공부하기");
  document.write("mystr의 내용 : " + mystr + "<br>");
  document.write("mystr의 길이 : " + mystr.length + "<br>");
  // mystr.length=10; // 이 문장은 오류이다.
</script>
</body>
</html>
```

객체 생성

메소드 호출

프로퍼티 읽기



Date 객체

- Date 객체

- 시간 정보를 담는 객체
- 현재 시간 정보

```
let now = new Date(); // 현재 날짜와 시간(시, 분, 초) 값으로 초기화된 객체 생성
```

- 학기 시작일 2024년 3월 1일의 날짜 기억

```
let startDay = new Date(2024, 2, 1); // 2024년 3월 1일(2는 3월을 뜻함)
```

- Date 객체 활용

```
let now = new Date();           // 현재 2024년 5월 15일 저녁 8시 48분이라면  
let date = now.getDate();       // 오늘 날짜. date = 15  
let hour = now.getHours();      // 지금 시간. hour = 20
```

Date 객체 메소드

- Date 객체에서 getFullYear(), getMonth(), getDate(), getDay() 메소드를 사용하면 연도, 월, 일, 시, 분 값을 모두 알 수 있다.

- getDate() (1-31 반환)
- getDay() (0-6 반환)
- getFullYear() (4개의 숫자로 된 연도 반환)
- getHours() (0-23 반환)
- getMilliseconds() (0-999)
- getMinutes() (0-59)
- getMonth() (0-11)
- getSeconds() (0-59)

- setDate()
- setDay()
- setFullYear()
- setHours()
- setMilliseconds()
- setMinutes()
- setMonth()
- setSeconds()

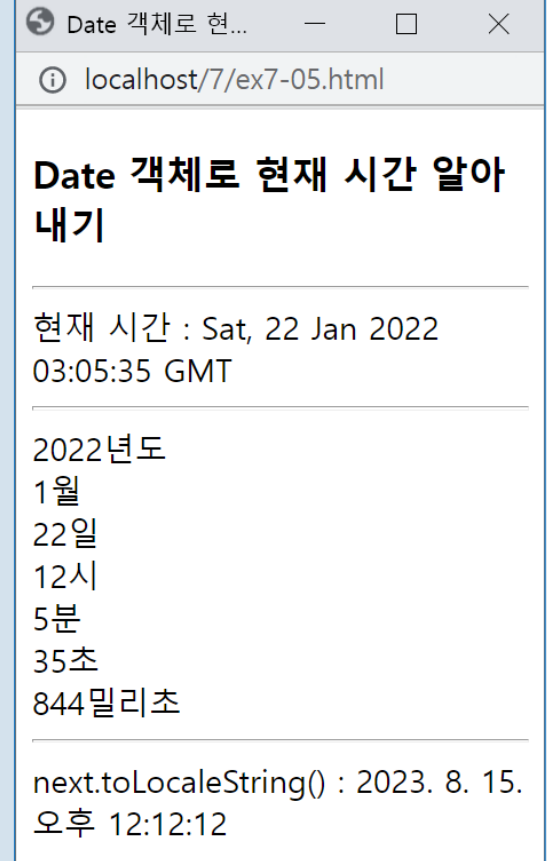
object_date1.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title>Date 객체로 현재 시간 알아내기</title>
</head>
<body>
<h3>Date 객체로 현재 시간 알아내기</h3>
<hr>
<script>
let now = new Date();
document.write("현재 시간 : " + now.toUTCString()
    + "<br><hr>");
document.write(now.getFullYear() + "년도<br>");
document.write(now.getMonth() + 1 + "월<br>");
document.write(now.getDate() + "일<br>");
document.write(now.getHours() + "시<br>");
document.write(now.getMinutes() + "분<br>");
document.write(now.getSeconds() + "초<br>");
document.write(now.getMilliseconds() + "밀리초<br><hr>");

let next = new Date(2023, 7, 15, 12, 12, 12); // 7은 8월
document.write("next.toLocaleString() : "
    + next.toLocaleString() + "<br>");
</script>
</body>
</html>
```

UTC(협정세계시)
문자열로 리턴

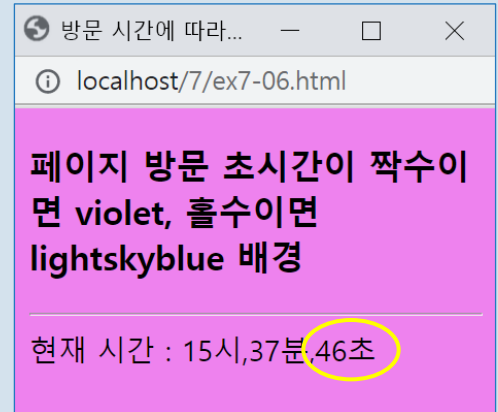
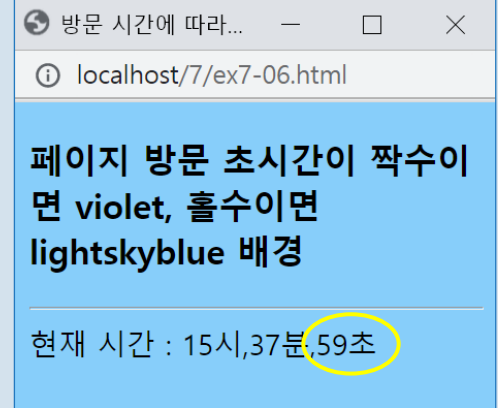
로컬시(국내)를
문자열로 리턴



object_date2.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title>방문 시간에 따라 변하는 배경색</title>
</head>
<body>
<h3>페이지 방문 초시간이 짝수이면 violet, 홀수이면 lightskyblue 배경</h3>
<hr>
<script>
  let current = new Date();
  if(current.getSeconds() % 2 == 0)
    document.body.style.backgroundColor = "violet";
  else
    document.body.style.backgroundColor = "lightskyblue";

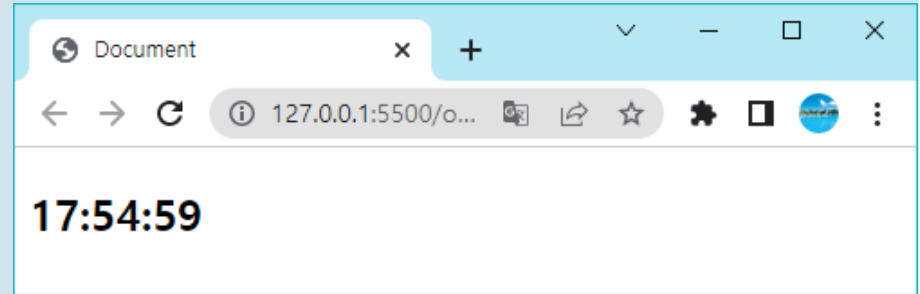
  document.write("현재 시간 : ");
  document.write(current.getHours(), "시,");
  document.write(current.getMinutes(), "분,");
  document.write(current.getSeconds(), "초<br>");
</script>
</body>
</html>
```



object_date3.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title>시계만들기</title>
</head>
<body>
<h3> setTimeout()로 타이머 설정하기 </h3>
<hr>
<h2 id='clock'></h2>
<script>
  function setClock() {
    let now = new Date();
    let s = now.getHours() + ':' + now.getMinutes() + ':' +
      now.getSeconds();
    document.getElementById('clock').innerHTML = s;
    setTimeout('setClock()', 1000);    //1초=1000밀리초
  }
  setClock();
</script>
</body>
</html>
```

특정시간이 지난 다음에
코드를 실행하는 함수



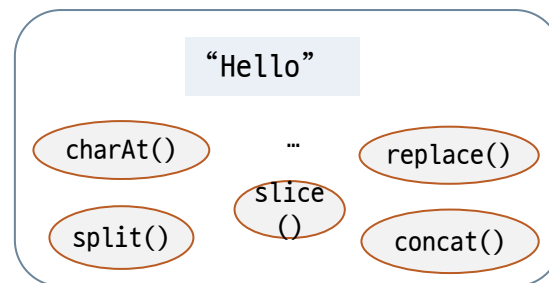
String 객체

- String
 - 문자열을 담기 위한 객체

```
// 2 경우 모두 오른쪽 String 객체 생성
```

```
let hello = new String("Hello");  
let hello = "Hello";
```

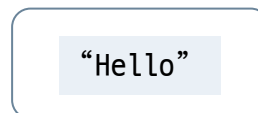
hello 객체



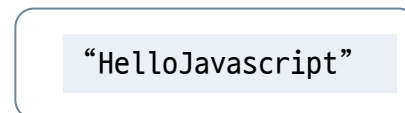
- String 객체는 일단 생성되면 수정 불가능

```
let hello = new String("Hello");  
let res = hello.concat("Javascript");  
  
// concat() 후 hello의 문자열 변화 없음
```

hello 객체



res 객체



String 객체의 특징

- 문자열 길이
 - String 객체의 length 프로퍼티 : 읽기 전용

```
let hello = new String("Hello");  
let every = "Boy and Girl";  
let m = hello.length;           // m은 5  
let n = every.length;          // n은 12
```

```
let n = "Thank you".length; // n은 9
```

- 문자열을 배열처럼 사용
 - [] 연산자를 사용하여 각 문자 접근

```
let hello = new String("Hello");  
let c = hello[0]; // c = "H". 문자 H가 아니라 문자열 "H"
```

String 객체 메소드

방법	설명
charAt(위치)	지정된 위치의 문자를 반환한다.
indexOf(검색 문자열, 위치)	지정된 숫자 인덱스에서 시작하여 지정된 문자열이 처음 나타나는 인덱스를 반환한다. 찾을 수 없으면 -1을 반환한다.
replace(검색값, 대체값)	지정된 값을 검색하고 대체값으로 바꾸고 새 문자열을 반환한다.
match(RegExp)	지정된 정규식을 기반으로 일치하는 항목을 검색한다.
slice(시작 위치, 종료 위치)	지정된 시작 및 끝 인덱스를 기반으로 문자열의 일부를 추출하고 새 문자열을 반환한다.
split(분리자, 제한 길이)	지정된 분리자에 따라 문자열을 문자열 배열로 분할한다.
substr(시작, 길이)	지정된 시작 위치에서 지정된 문자 수(길이)까지 문자열의 일부를 반환한다.
substring(시작, 끝)	시작 인덱스와 끝 인덱스 사이의 문자열에서 부분 문자열을 반환한다.
toLowerCase()	소문자로 변환한다.
toUpperCase()	대문자로 변환한다.
valueOf()	문자열을 수치값으로 변환한다.

object_string1.html

```
<!DOCTYPE html>
<html><head><meta charset="utf-8">
<title>String 객체의 메소드 활용</title></head>
<body>
<h3>String 객체의 메소드 활용</h3>
<hr>
<script>
let a = new String("Boys and Girls");
let b = "!!";
document.write("a : " + a + "<br>");
document.write("b : " + b + "<br><hr>");

document.write(a.charAt(0) + "<br>");
document.write(a.concat(b, "입니다") + "<br>");
document.write(a.indexOf("s") + "<br>");
document.write(a.indexOf("And") + "<br>");
document.write(a.slice(5, 8) + "<br>");
document.write(a.substr(5, 3) + "<br>");
document.write(a.toUpperCase() + "<br>");
document.write(a.replace("and", "or") + "<br>");
document.write("   kitae   ".trim() + "<br><hr>");

let sub = a.split(" ");
document.write("a를 빈칸으로 분리<br>");
for(let i=0; i<sub.length; i++)
    document.write("sub" + i + "=" + sub[i] + "<br>");

document.write("<hr>String 메소드를 실행 후 a와 b 변함 없음<br>");
document.write("a : " + a + "<br>");
document.write("b : " + b + "<br>");
</script>
</body></html>
```

a.charAt(0)

a.indexOf("s")

a.slice(5,8)



object_string2.html

```
<!DOCTYPE html>
```

```
<html>
```

```
<head>
```

```
<script>
```

```
function checkID() {
```

```
    let s = document.getElementById("id").value;
```

```
    if (s.length < 6)
```

```
        alert("아이디는 6글자 이상이어야 합니다.");
```

```
}
```

```
</script>
```

```
</head>
```

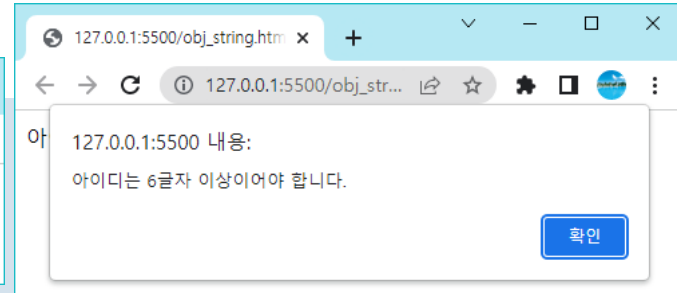
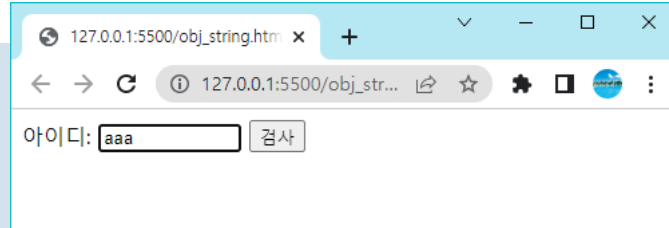
```
<body>
```

```
아이디: <input type="text" id="id" size=10>
```

```
<button onclick="checkID()">검사</button>
```

```
</body>
```

```
</html>
```

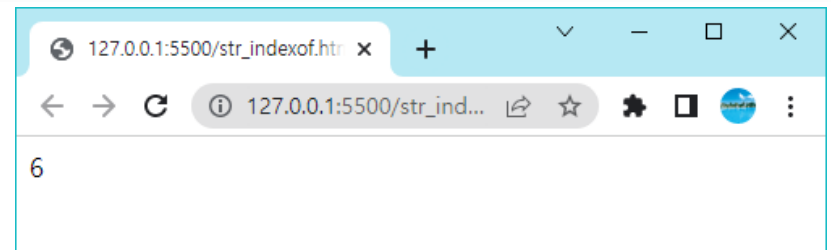


문자열 검색

- indexOf() 메소드는 문자열 안에서 주어진 텍스트가 처음 등장하는 위치를 반환한다. 아주 편리한 메소드이다.

str_indexof.html

```
<script>  
  let s = "A bad workman always blames his tools.";  
  let n = s.indexOf("workman");  
  document.write(n + '<br>'); // 6  
</script>
```



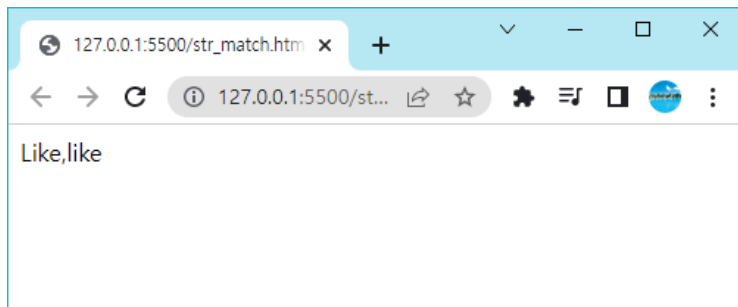
문자열 매칭

- match() 메소드에서는 정규식(regular expression)을 사용할 수 있다.

str_match.html

```
<script>
let str = 'Like father, like son.';
// like를 찾는다. i와 g는 옵션으로 insensitive, globally를 의미한다.
result = str.match(/like/ig);
document.write(result + '<br>');
</script>
```

"/"/ 사이에 정규식 패턴을 넣고 플래그 옵션(i,g)을 추가
i : 대소문자를 구분하지 않고 검색
g : 문자열 내의 모든 패턴을 검색

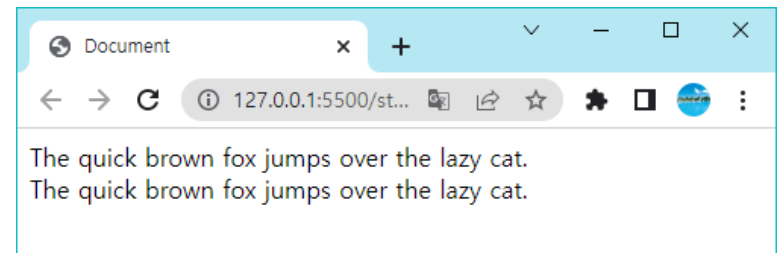


문자열 대체

- `replace()` 메소드는 문자열 안에서 주어진 값을 다른 값으로 대체한다. `replace()`도 정규식을 사용할 수 있어서 강력한 기능 중의 하나이다.

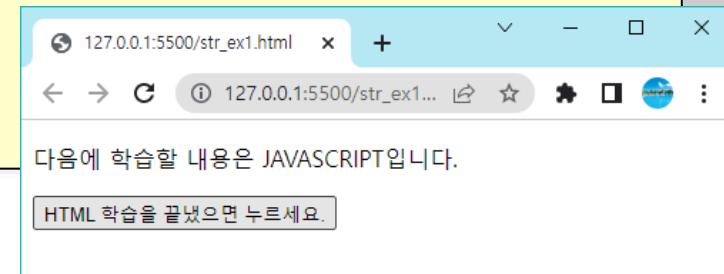
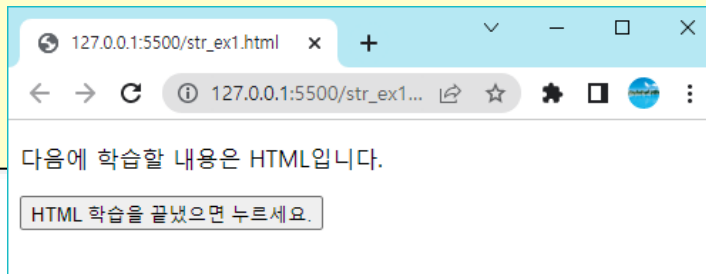
str_replace1.html

```
<script>
  let p = 'The quick brown fox jumps over the lazy dog.';
  document.write(p.replace('dog', 'cat')+"<br>");
  let regex = /Dog/i;
  document.write(p.replace(regex, 'cat')+"<br>");
</script>
```



str_replace2.html

```
<!DOCTYPE html>
<html>
<body>
  <p id="test">다음에 학습할 내용은 HTML입니다. </p>
  <button onclick="myFun()">HTML 학습을 끝냈으면 누르세요. </button>
  <script>
    function myFun() {
      let text = document.getElementById("test").innerHTML;
      document.getElementById("test").innerHTML =
        text.replace("HTML", "JAVASCRIPT");
    }
  </script>
</body>
</html>
```

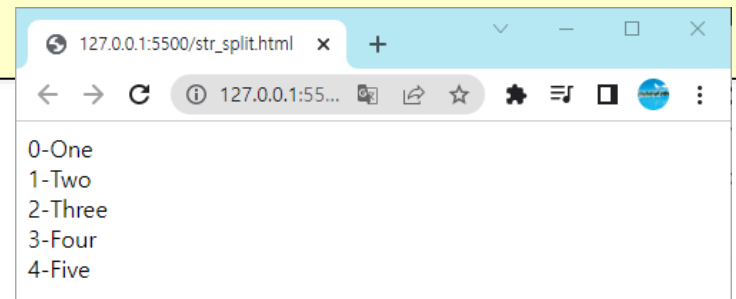


문자열 분할

- `split()`는 첫 번째 인수를 분리자로 하여서 주어진 문자열을 분리한 후에 각 항목들을 가지고 있는 배열을 반환한다

str_split.html

```
<script>
  let s = "One,Two,Three,Four,Five";
  array = s.split(',');
  for (i = 0; i < array.length; i++) {
    document.write(i + '-' + array[i] + '<br>');
  }
</script>
```

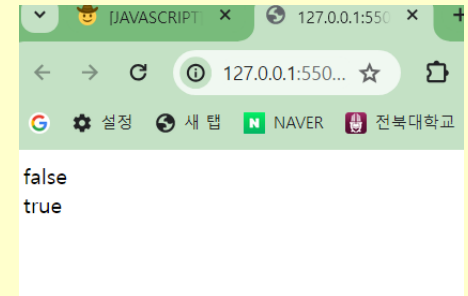


문자열이 특정 문자열로 시작하는지(or 끝나는지) 확인

- `startsWith()`, `endsWith()`는 문자열이 특정 문자열로 시작하는지, 끝나는지에 대해 `true`, `false`값을 반환

[str_startsWith.html](#)

```
<script>
  let s = 'examples.html';
  document.write(s.startsWith('exe') + "<br>");
  document.write(s.endsWith('.html'));
</script>
```

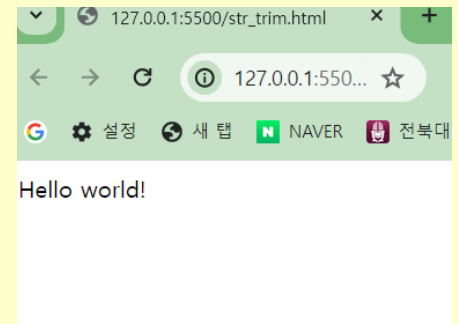


공백 제거

- 문자열 앞뒤에 붙은 공백 문자들을 제거하는데는 trim() 메소드를 사용할 수 있다.
- trim() 메소드가 왜 필요할까? 사용자가 문자열을 입력할 때 실수로 앞뒤에 공백을 추가하는 일도 매우 흔하게 발생하기 때문이다.

str_trim.html

```
<script>
  let s = '  Hello world!  ';
  s = s.trim();
  document.write(s);
</script>
```



Math 객체

- Math

- 수학 계산을 위한 프로퍼티와 메소드 제공
- new Math()로 객체 생성하지 않고 사용

```
let sq = Math.sqrt(4);    // 4의 제곱근을 구하면 2  
let area = Math.PI*2*2;    // 반지름이 2인 원의 면적
```

- 난수 발생

- Math.random() : 0~1보다 작은 랜덤한 실수 리턴
- Math.floor(m)은 m의 소수점 이하를 제거한 정수 리턴

```
// 0~99까지 랜덤한 정수를 10개 만드는 코드  
for(let i=0; i<10; i++) {  
    let m = Math.random()*100; // m은 0~99.999... 보다 작은 실수 난수  
    let n = Math.floor(m);      // m에서 소수점 이하를 제거한 정수(0~99사이)  
    document.write(n + " ");  
}
```


Math 객체

메소드	설명
<code>abs(x)</code>	절대값
<code>acos(x)</code> , <code>asin(x)</code> , <code>atan(x)</code>	아크 삼각함수
<code>ceil(x)</code> , <code>floor(x)</code>	실수를 정수로 올림, 내림 함수
<code>cos(x)</code> , <code>sin(x)</code> , <code>tan(x)</code>	삼각함수
<code>exp(x)</code>	지수함수
<code>log(x)</code>	로그함수
<code>max(x,y,z,...,n)</code>	최대값
<code>min(x,y,z,...,n)</code>	최소값

메소드	설명
<code>pow(x,y)</code>	지수함수 x^y
<code>random()</code>	0과 1 사이의 난수값 반환
<code>round(x)</code>	반올림
<code>sqrt(x)</code>	제곱근

obj_math1.html

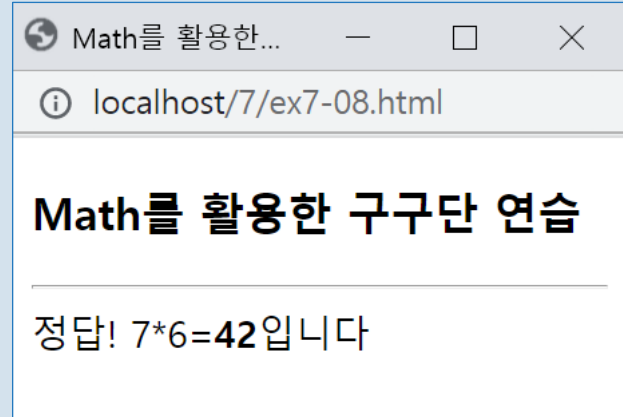
```
<!DOCTYPE html>
<html>
<head><meta charset="utf-8">
<title>Math를 활용한 구구단 연습</title>
<script>
    function randomInt() { // 1~9의 십진 난수 리턴
        return Math.floor(Math.random()*9) + 1;
    }
</script>
</head>
<body>
<h3>Math를 활용한 구구단 연습</h3>
<hr>
<script>
    // 구구단 문제 생성
    let ques = randomInt() + "*" + randomInt();
    // 사용자로부터 답 입력
    let user = prompt(ques + " 값은 얼마입니까?", 0);
    if(user == null) { // 취소 버튼이 클릭된 경우
        document.write("구구단 연습을 종료합니다");
    }
    else {
        let ans = eval(ques); // 구구단 정답 계산
        if(ans == user) // 정답과 사용자 입력 비교
            document.write("정답! ");
        else
            document.write("아니오! ");
        document.write(ques + "=" + "<strong>" + ans
            + "</strong>입니다<br>");
    }
</script>
</body>
</html>
```

localhost 내용:

7*6 값은 얼마입니까?

확인

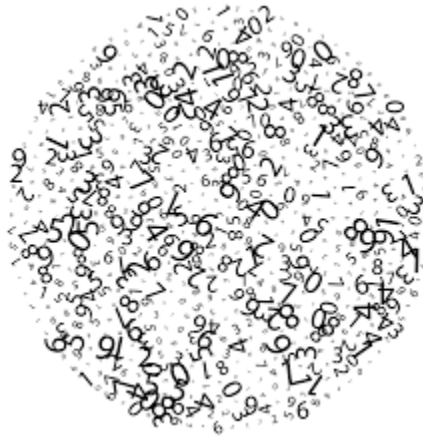
취소



난수 발생

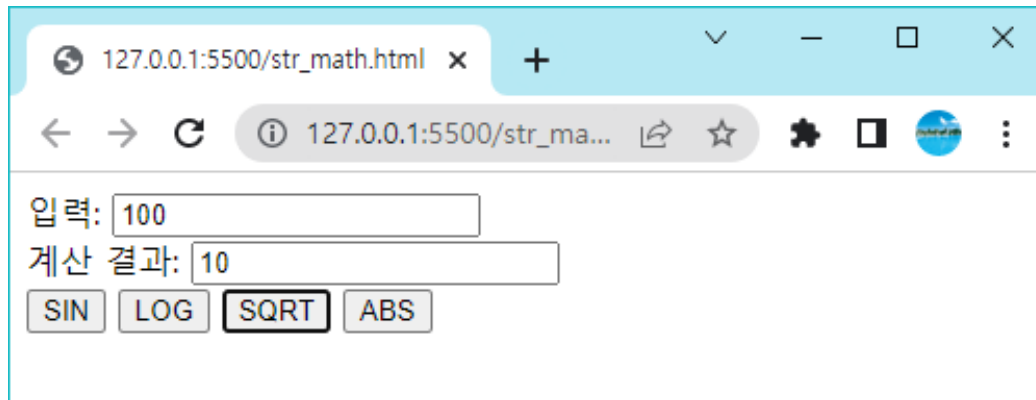
- Math.random()을 호출하면 0에서 1 미만의 난수가 발생된다. 따라서 여기에 적당한 수를 곱해서 사용하면 된다.

```
let f = Math.random() * 100; // f는 0.0~100.0 미만의 실수  
let n = Math.floor(f);        // n은 0~99 사이의 정수
```



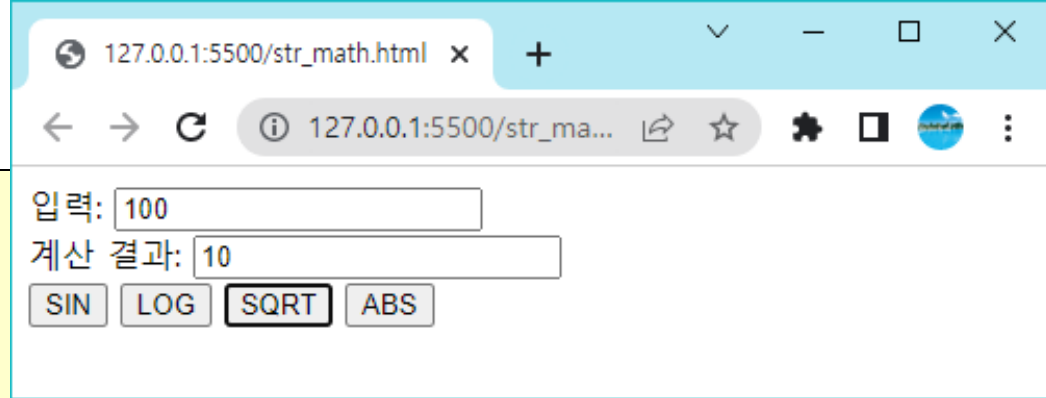
예제: 공학용 계산기

- 몇 가지만 계산하는 공학용 계산기 프로그램을 작성해보자.



obj_math2.html

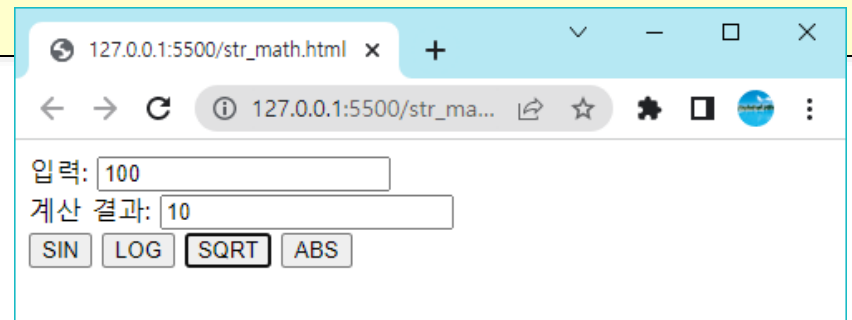
```
<html>
<head>
  <script>
    function calc(type) {
      let x = parseFloat(document.calculator.number1.value);
      if (type == 1)
        y = Math.sin((x * Math.PI) / 180.0);
      else if (type == 2)
        y = Math.log(x);
      else if (type == 3)
        y = Math.sqrt(x);
      else if (type == 4)
        y = Math.abs(x);
      document.calculator.total.value = y;
    }
  </script>
</head>
```



```
<body>
  <form name="calculator">
    입력:      <input type="text" name="number1"><br />
    계산 결과:  <input type="text" name="total"><br />

    <input type="button" value="SIN" onclick="calc(1);">
    <input type="button" value="LOG" onclick="calc(2);">
    <input type="button" value="SQRT" onclick="calc(3);">
    <input type="button" value="ABS" onclick="calc(4);">
  </form>

</body>
</html>
```



자바스크립트 배열

- 배열
 - 여러 개의 원소들을 연속적으로 저장
 - 전체를 하나의 단위로 다루는 데이터 구조
- 배열 생성 사례

```
let cities = ["Seoul", "New York", "Paris"];
```

cities	"Seoul"	<i>cities[0]</i>
	"New York"	<i>cities[1]</i>
	"Paris"	<i>cities[2]</i>

```
let n = [4, 5, -2, 28, 33];
```

n	4	5	-2	28	33
	<i>n[0]</i>	<i>n[1]</i>	<i>n[2]</i>	<i>n[3]</i>	<i>n[4]</i>

- 0에서 시작하는 인덱스를 이용하여 배열의 각 원소 접근

```
let name = cities[0];    // name은 "Seoul"  
cities[1] = "Gainesville"; // "New York" 자리에 "Gainesville" 저장
```

자바스크립트에서 배열을 만드는 방법

- 배열 만드는 2가지 방법
 - []로 배열 만들기
 - Array 객체로 배열 만들기
- []로 배열 만들기
 - [] 안에는 원소들의 초기 값 나열

```
let week = ["월", "화", "수", "목", "금", "토", "일"];  
let plots = [-20, -5, 0, 15, 20];
```

- 배열 크기 : 배열의 크기는 고정되지 않고 원소 추가 시 늘어남
 - 배열의 끝에 원소 추가

```
plots[5] = 33; // plots 배열에 6번째 원소 추가. 배열 크기는 6이 됨  
plots[6] = 22; // plots 배열에 7번째 원소 추가. 배열 크기는 7이 됨
```

- **주의** : 현재 배열보다 큰 인덱스에 원소를 추가하면 값이 비어 있는 중간의 원소들도 생기는 문제 발생

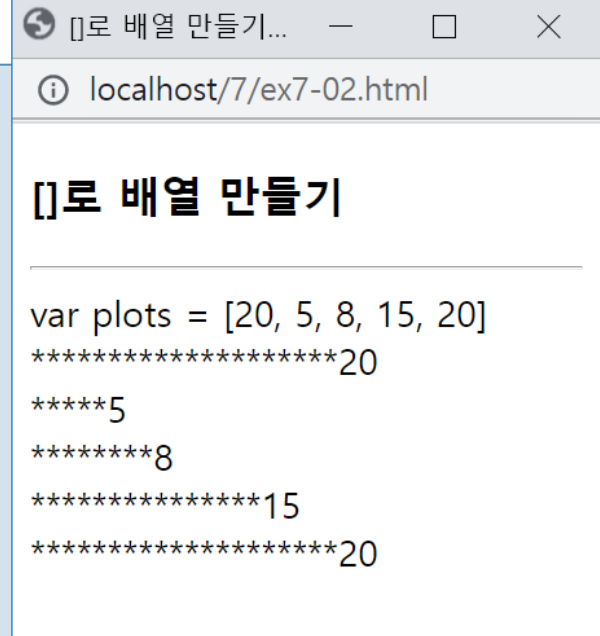
```
plots[10] = 33; // 주의. plots 배열의 크기는 11개가 되고,  
                // plots[7], plots[8], plots[9]의 값은 모두 undefined 값
```


Js_array1.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title>[]로 배열 만들기</title>
</head>
<body>
<h3>[]로 배열 만들기</h3>
<hr>
<script>
  let plots = [20, 5, 8, 15, 20]; // 원소 5개의 배열 생성
  document.write("var plots = [20, 5, 8, 15, 20]<br>");

  for(let i=0; i<5; i++) {
    let size = plots[i]; // plots 배열의 i번째 원소
    while(size>0) {
      document.write("*");
      size--;
    }
    document.write(plots[i] + "<br>");
  }
</script>
</body>
</html>
```

plots.length로해도됨



Array로 배열 만들기

- 초기 값을 가진 배열 생성

```
let week = new Array("월", "화", "수", "목", "금", "토", "일");
```

- 초기화되지 않은 배열 생성

- 일정 크기의 배열 생성 후 나중에 원소 값 저장

```
let week = new Array(7); // 7개의 원소를 가진 배열 생성
```

```
week[0] = "월";  
week[1] = "화";  
...  
week[6] = "일";
```

- 빈 배열 생성

- 원소 개수를 예상할 수 없는 경우

```
let week = new Array(); // 빈 배열 생성
```

```
week[0] = "월"; // 배열 week 크기 자동으로 1  
week[1] = "화"; // 배열 week 크기 자동으로 2
```

배열의 원소 개수, length 프로퍼티

- 배열의 크기 : Array 객체의 length 프로퍼티

```
let plots = [-20, -5, 0, 15, 20];  
let week = new Array("월", "화", "수", "목", "금", "토", "일");  
let m = plots.length;    // m은 5  
let n = week.length;    // n은 7
```

- length 프로퍼티는 사용자가 임의로 값 변경 가능
 - length 프로퍼티는 Array 객체에 의해 자동 관리
 - 사용자가 임의로 값 변경 가능
 - 배열의 크기를 줄이거나 늘일 수 있음

```
plots.length = 10; // plots의 크기는 5에서 10으로 늘어남  
plots.length = 2; // plots의 크기는 2로 줄어 들어,  
                // 처음 2개의 원소 외에는 모두 삭제 됨
```

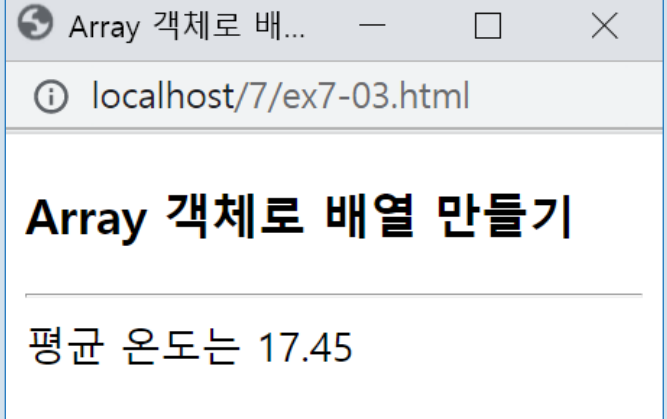
Js_array2.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title>Array 객체로 배열 만들기</title>
</head>
<body>
<h3>Array 객체로 배열 만들기</h3>
<hr>
<script>
  let degrees = new Array(); // 빈 배열 생성
  degrees[0] = 15.1;
  degrees[1] = 15.4;
  degrees[2] = 16.1;
  degrees[3] = 17.5;
  degrees[4] = 19.2;
  degrees[5] = 21.4;
  let sum = 0;
  for(let i=0; i<degrees.length; i++)
    sum += degrees[i];

  document.write("평균 온도는 " + sum/degrees.length + "<br>");
</script>
</body>
</html>
```

배열 크기만큼 루프

배열 degrees의 크기, 6



배열의 특징

- 배열은 Array 객체
 - []로 생성해도 Array 객체로 다루어짐
- 배열에 여러 타입의 데이터 섞여 저장 가능

```
let any = new Array(5);    // 5개의 원소를 가진 배열 생성
any[0] = 0;
any[1] = 5.5;
any[2] = "이미지 벡터";    // 문자열 저장
any[3] = new Date();       // Date 객체 저장
any[4] = convertFunction;  // function convertFunction()의 주소 저장
```

Js_array3.html

```
<!DOCTYPE html>
<html><head><meta charset="utf-8"><title>Array 객체의 메소드 활용</title>
<script>
    function pr(msg, arr) { document.write(msg + arr.toString() + "<br>"); }
</script>
</head>
<body>
<h3>Array 객체의 메소드 활용</h3>
<hr>
<script>
    let a = new Array("황", "김", "이");
    let b = new Array("박");
    let c;

    pr("배열 a = ", a);
    pr("배열 b = ", b);
    document.write("<hr>");

    c = a.concat(b); // c는 a와 b를 연결한 새 배열
    pr("c = a.concat(b)후 c = ", c);
    pr("c = a.concat(b)후 a = ", a);

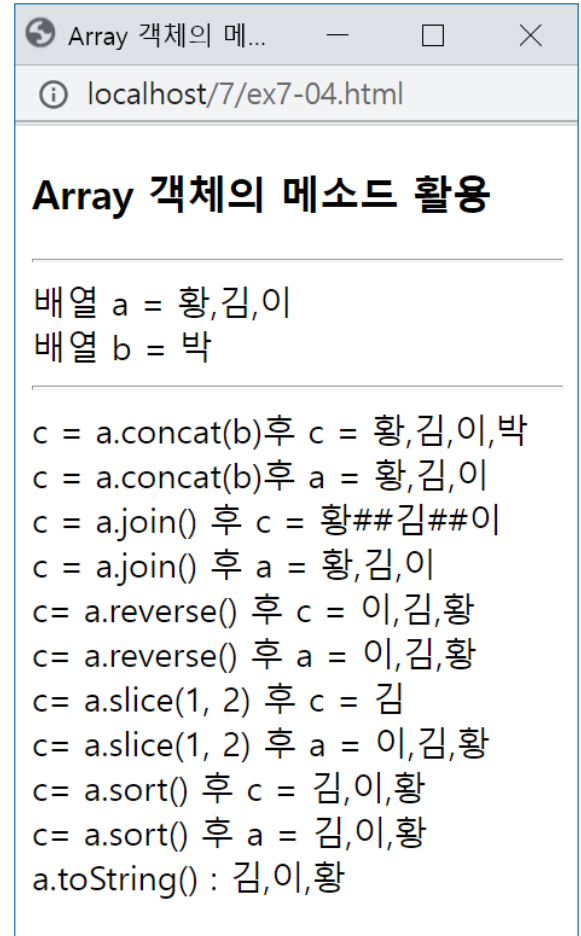
    c = a.join("##"); // c는 배열 a를 연결한 문자열
    pr("c = a.join() 후 c = ", c);
    pr("c = a.join() 후 a = ", a);

    c = a.reverse(); // a.reverse()로 a 자체 변경. c는 배열
    pr("c= a.reverse() 후 c = ", c);
    pr("c= a.reverse() 후 a = ", a);

    c = a.slice(1, 2); // c는 새 배열
    pr("c= a.slice(1, 2) 후 c = ", c);
    pr("c= a.slice(1, 2) 후 a = ", a);

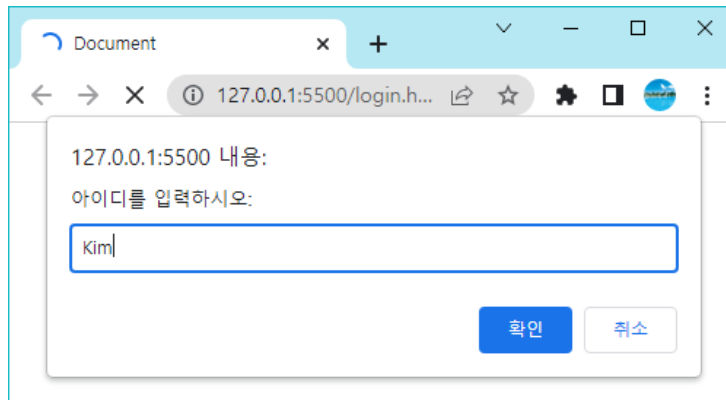
    c = a.sort(); // a.sort()는 a 자체 변경. c는 배열
    pr("c= a.sort() 후 c = ", c);
    pr("c= a.sort() 후 a = ", a);

    c = a.toString(); // toString()은 원소 사이에 ","를 넣어 문자열 생성
    document.write("a.toString() : " + c); // c 는 문자열
</script></body></html>
```



예제: 로그인 시스템 만들기

- 컴퓨터 관리자들의 아이디를 배열에 저장한다. 관리자의 아이디가 입력되면 "로그인 성공"을 출력한다. 관리자의 아이디가 아니면 "로그인 실패"를 화면에 출력한다.



Document

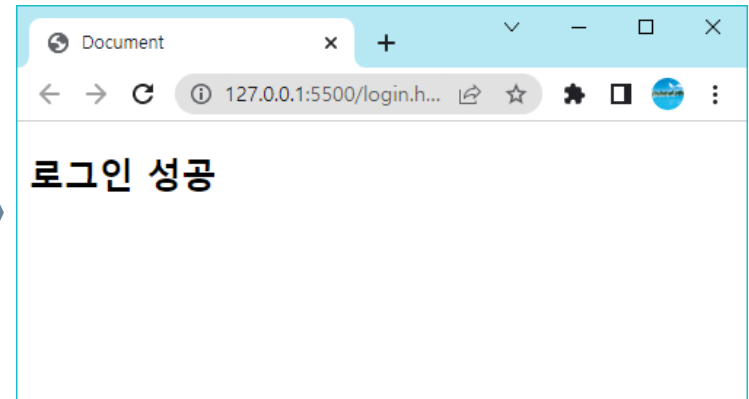
127.0.0.1:5500/login.h...

127.0.0.1:5500 내용:

아이디를 입력하십시오:

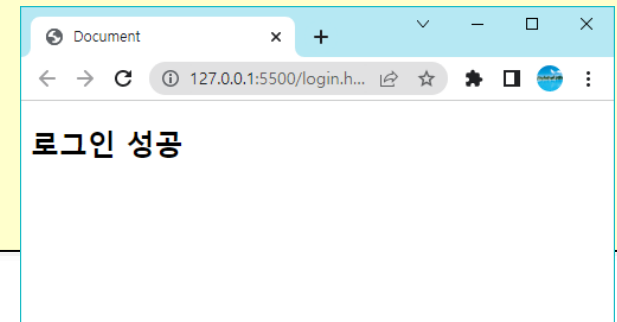
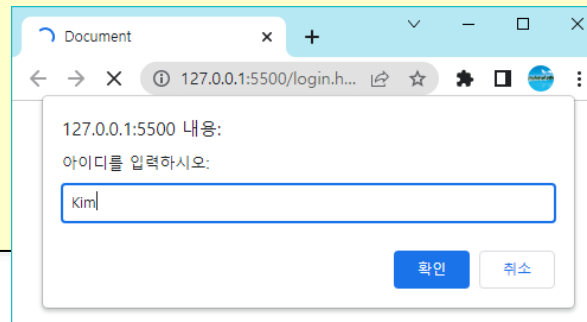
Kim

확인 취소



Js_array4.html

```
<!DOCTYPE html>
<head></head>
<body>
  <h2 id="test"></h2>
  <script>
    let admin = ["Kim", "Lee", "Choi"]
    person = prompt("아이디를 입력하시오: ", "");
    document.getElementById("test").innerHTML = "로그인 실패";
    for (i = 0; i < admin.length; i++) {
      if (person == admin[i]) {
        document.getElementById("test").innerHTML = "로그인 성공";
        break;
      }
    }
  </script>
</body>
</html>
```

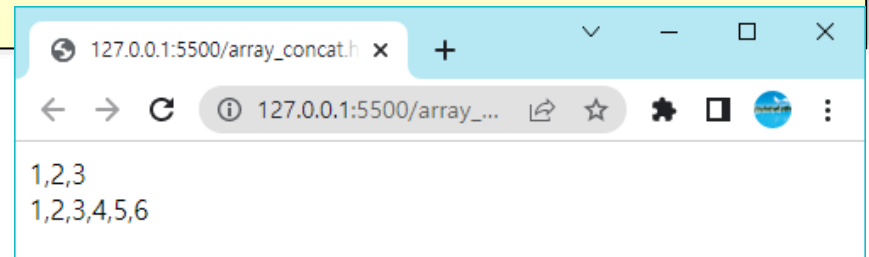


Array 메소드: concat()

- concat() 메소드는 전달된 인수를 배열의 끝에 추가한다. 인수는 배열일 수도 있다.

array_concat.html

```
<script>
  let x = [1, 2, 3];
  let y = [4, 5, 6];
  let joined = x.concat(y);
  document.writeln(x+"<br>");           // 출력: 1,2,3
  document.writeln(joined+"<br>");       // 출력: 1,2,3,4,5,6
</script>
```



Array 메소드: indexOf()

- indexOf() 메소드는 요소의 값을 가지고 요소의 인덱스를 찾을 때, 사용한다.

array_indexof.html

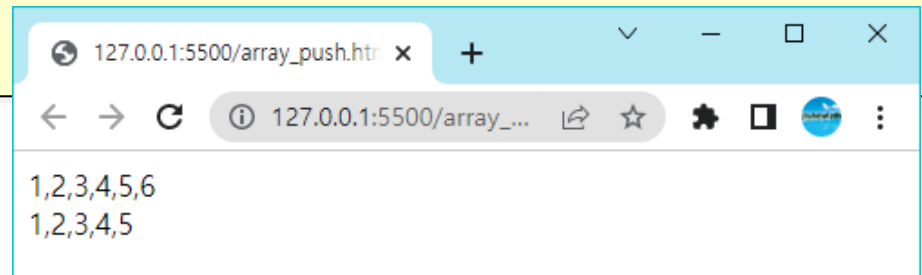
```
<script>  
  let fruits = ["apple", "banana", "grape"];  
  document.writeln(fruits.indexOf("banana"));    // 출력: 1  
</script>
```

Array 메소드: push(), pop()

- push()는 스택(stack)에 데이터를 삽입하는 메소드이다. pop()은 스택에서 데이터를 꺼내는 메소드이다.

array_push.html

```
<script>
  let numbers = [1, 2, 3, 4, 5];
  numbers.push(6);
  document.writeln(numbers + '<br>');      // 출력: 1,2,3,4,5,6
  item = numbers.pop();
  document.writeln(numbers + '<br>');      // 출력: 1,2,3,4,5,
</script>
```



Array 메소드: shift(), unshift()

- shift()는 배열의 첫번째 요소를 반환하고 이 첫번째 요소를 배열에서 제거
- Unshift()는 배열 앞에 요소를 추가하고, 배열의 새로운 길이 값을 반환

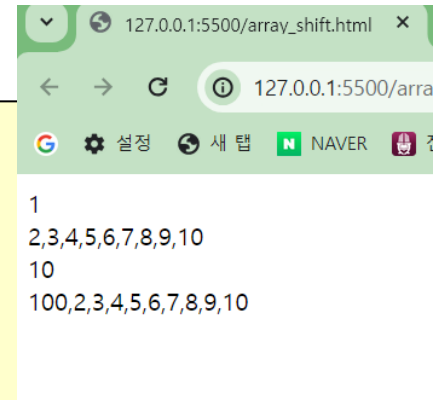
array_shift.html

```
<script>
  let numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];

  let item1 = numbers.shift();
  document.writeln(item1 + '<br>');           // 출력: 1
  document.writeln(numbers + '<br>');         // 출력: 2,3,4,5,6,7,8,9,10

  let item2 = numbers.unshift(100);
  document.writeln(item2 + '<br>');           // 출력: 10
  document.writeln(numbers + '<br>');         // 출력: 100,2,3,4,5,6,7,8,9,10

</script>
```

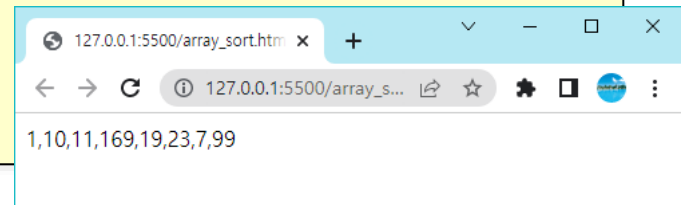


Array 메소드: sort()

- sort() 메소드는 배열을 알파벳 순으로 정렬한다. 숫자를 정렬하면 우리의 예상과 다를 수 있다.

array_sort1.html

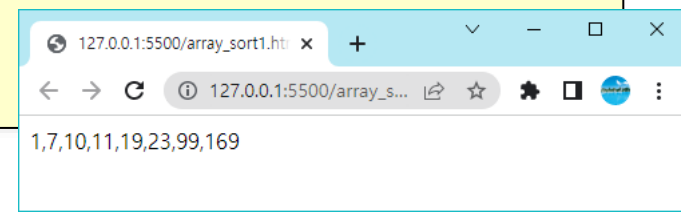
```
<script>
  let myArray = [10, 7, 23, 99, 169, 19, 11, 1];
  myArray.sort() //알파벳 순서로 정렬
  document.writeln(myArray);
</script>
```



array_sort2.html

```
<script>
  let myArray = [10, 7, 23, 99, 169, 19, 11, 1];
  myArray.sort(function (a, b) { return a - b });
  document.writeln(myArray);
</script>
```

콜백함수
a-b가 음수이면 a를 왼쪽으로
b를 오른쪽으로 넘긴다.



Array 메소드: slice()

- slice() 메소드는 배열의 일부를 복사하여 새로운 배열로 반환한다.

array_slice.html

```
<script>
```

```
let numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];
```

```
let newArray = numbers.slice(5);
```

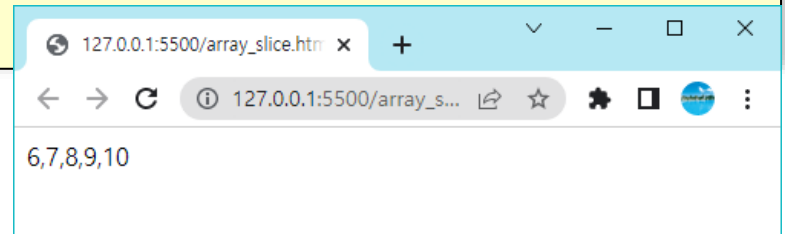
```
document.writeln(newArray + '<br>');
```

```
</script>
```

시작인덱스가 가리키는 값부터
배열의 마지막 값까지 모두
복사해준다.

// 출력: 6,7,8,9,10

numbers.slice(2,5)는
2부터 4까지를 복사한 값
3,4,5

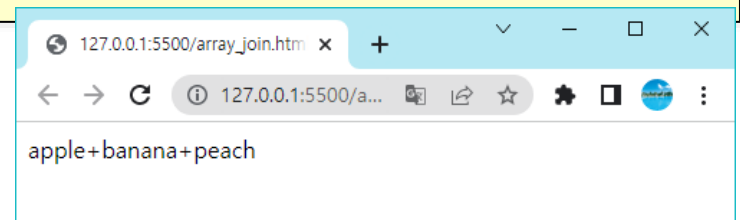


Array 메소드: join()

- join()은 배열의 요소들을 하나의 문자열로 출력한다.

array_join.html

```
<script>
  let fruits = ["apple", "banana", "peach" ];
  let s = fruits.join("+");
  document.writeln(s+'<br>'); // 출력: apple+banana+peach
</script>
```



Array 메소드: filter()

- filter() 메소드는 어떤 조건에 부합하는 요소만을 선택하여 새로운 배열로 만들어서 반환한다.

array_filter.html

```
<script>
```

```
let numbers = [10, 20, 30, 40, 50];
```

```
function isBigEnough(element, index, array) {
```

```
    return (element >= 30);
```

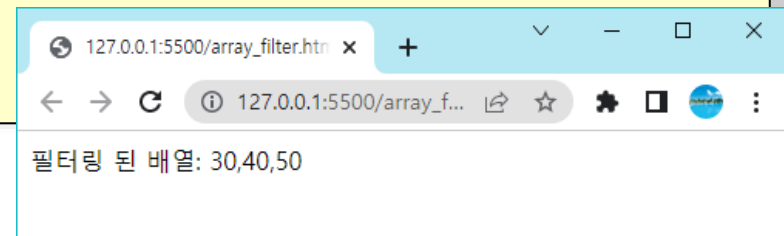
```
}
```

```
let filtered = numbers.filter(isBigEnough);
```

```
document.write("필터링 된 배열: " + filtered);
```

```
</script>
```

element: 현재 처리 중인 배열 요소
index: 배열 요소의 인덱스
array: filter()를 호출한 배열 자체
element % 2 == 0; // 짝수인 경우



2차원 배열

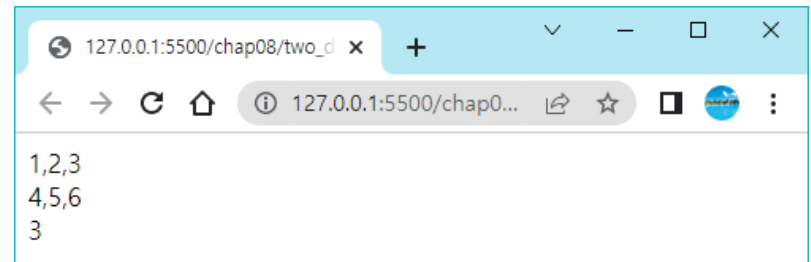
- 자바스크립트에서는 2차원 배열이 가능한가? 가능하다!

```
let z = [ [1, 2, 3], [4, 5, 6] ]
```

```
document.writeln(z[0] + "<br>");    // 출력: 1, 2, 3
```

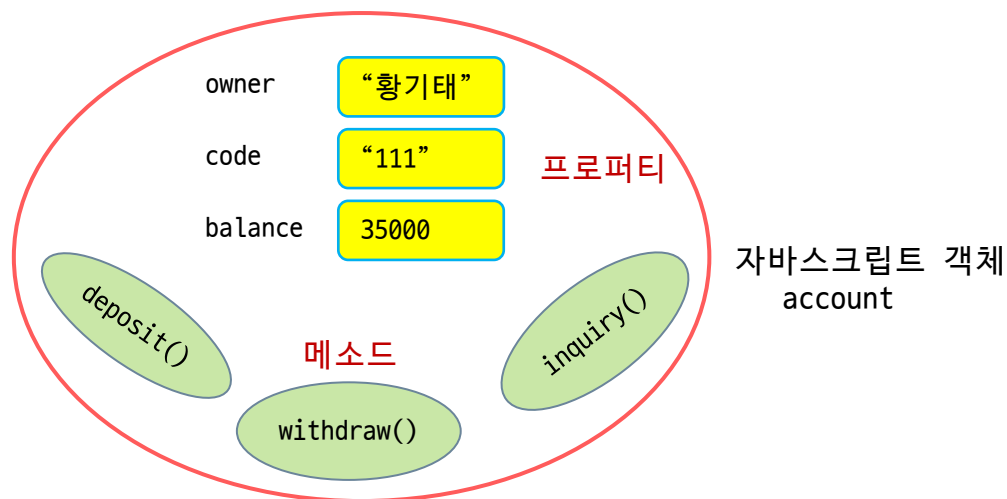
```
document.writeln(z[1] + "<br>");    // 출력: 4, 5, 6
```

```
document.writeln(z[0][2] + "<br>"); // 출력: 3
```



사용자 객체 만들기

- 사용자가 새로운 타입의 객체 작성 가능 : 3 가지 방법
 - 1. 직접 객체 만들기
 - new Object() 이용
 - 리터럴 표기법 이용
 - 2. 객체의 틀(프로토타입)을 만들고 객체 생성하기
- 샘플
 - 은행 계좌를 표현하는 account 객체



new Object()로 객체 만들기

- 과정
 - 1. new Object()로 빈 객체 생성
 - 2. 빈 객체에 프로퍼티 추가
 - 새로운 프로퍼티 추가(프로퍼티 이름과 초기값 지정)
 - 3. 빈 객체에 메소드 추가
 - 메소드로 사용할 함수 미리 작성
 - 새 메소드 추가(메소드 이름에 함수 지정)

```
let account = new Object();  
account.owner = "홍길동"; // 계좌 주인 프로퍼티 생성 및 초기화  
account.code = "111"; // 코드 프로퍼티 생성 및 초기화  
account.balance = 35000; // 잔액 프로퍼티 생성 및 초기화  
account.inquiry = inquiry; // 메소드 작성  
account.deposit = deposit; // 메소드 작성  
account.withdraw = withdraw; // 메소드 작성
```

new_object.html

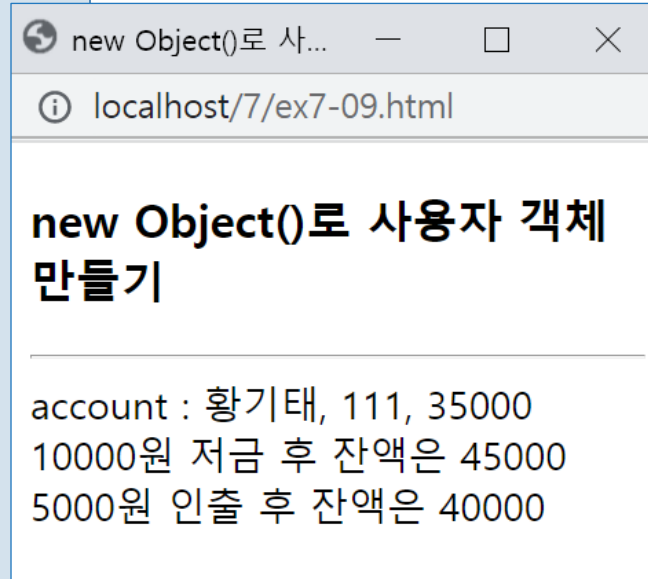
```
<!DOCTYPE html>
<html><head><meta charset="utf-8">
<title>new Object()로 사용자 객체 만들기</title>
<script>
  //메소드로 사용할 3 개의 함수 작성
  function inquiry() { return this.balance; } // 잔금 조회
  function deposit(money) { this.balance += money; } // money 만큼 저금
  function withdraw(money) { // 예금 인출, money는 인출하고자 하는 액수
    // money가 balance보다 작다고 가정

    this.balance -= money;
    return money;
  }

  // 사용자 객체 만들기
  let account = new Object();
  account.owner = "황기태"; // 계좌 주인 프로퍼티 생성 및 초기화
  account.code = "111"; // 코드 프로퍼티 생성 및 초기화
  account.balance = 35000; // 잔액 프로퍼티 생성 및 초기화
  account.inquiry = inquiry; // 메소드 작성
  account.deposit = deposit; // 메소드 작성
  account.withdraw = withdraw; // 메소드 작성
</script></head>
<body>
<h3>new Object()로 사용자 객체 만들기</h3>
<hr>
<script>
  // 객체 활용
  document.write("account : ");
  document.write(account.owner + ", ");
  document.write(account.code + ", ");
  document.write(account.balance + "<br>");

  account.deposit(10000); // 10000원 저금
  document.write("10000원 저금 후 잔액은 " + account.inquiry() + "<br>");
  account.withdraw(5000); // 5000원 인출
  document.write("5000원 인출 후 잔액은 " + account.inquiry() + "<br>");
</script>
</body></html>
```

this.balance는 객체의
balance 프로퍼티



리터럴 표기법으로 만들기

- 과정
 - 중괄호를 이용하여 객체의 프로퍼티와 메소드 지정
 - 가장 많이 사용하는 방법

```
let account = {  
  // 프로퍼티 생성 및 초기화  
  owner : "황기태",    // 계좌 주인 프로퍼티 추가  
  code : "111",        // 계좌 코드 프로퍼티 추가  
  balance : 35000,     // 잔액 프로퍼티 추가  
  
  // 메소드 작성  
  inquiry : function () { return this.balance; }, // 잔금 조회  
  deposit : function(money) { this.balance += money; }, // 저금. money 만큼 저금  
  withdraw : function (money) { // 예금 인출, money는 인출하고자 하는 액수  
                                // money가 balance보다 작다고 가정  
    this.balance -= money;  
    return money;  
  }  
};
```

객체 리터럴 사용

- 객체 리터럴을 사용하면 하나의 명령문에서 객체를 정의하고 생성할 수 있다.

```
let myCar = {  
  model: "520d",  
  speed: 60,  
  color: "red",  
  
  brake: function () { this.speed -= 10; },  
  accel: function () { this.speed += 10; }  
};  
  
myCar.color = "yellow";  
myCar.brake();
```

```
<!DOCTYPE html>
<html>
<head> <meta charset="utf-8">
<title>리터럴 표기법으로 사용자 객체 만들기</title>
<script>
//사용자 객체 만들기
```

```
let account = {
  owner : "황기태", // 계좌 주인
  code : "111",    // 계좌 코드
  balance : 35000, // 잔액 프로퍼티

  inquiry : function () { return this.balance; }, // 잔금 조회
  deposit : function(money) { this.balance += money; }, // 저금. money 만큼 저금
  withdraw : function (money) { // 예금 인출
    this.balance -= money;
    return money;
  }
};
```

```
</script> </head>
<body>
<h3>리터럴 표기법으로 사용자 객체 만들기</h3>
<hr>
<script>
  document.write("account : ");
  document.write(account.owner + ", ");
  document.write(account.code + ", ");
  document.write(account.balance + "<br>");
  account.deposit(10000); // 10000원 저금
  document.write("10000원 저금 후 잔액은 " + account.inquiry() + "<br>");
  account.withdraw(5000); // 5000원 인출
  document.write("5000원 인출 후 잔액은 " + account.inquiry() + "<br>");
</script>
</body> </html>
```

리터럴 표기법으로 사용자 객체 만들기

account : 황기태, 111, 35000
10000원 저금 후 잔액은 45000
5000원 인출 후 잔액은 40000

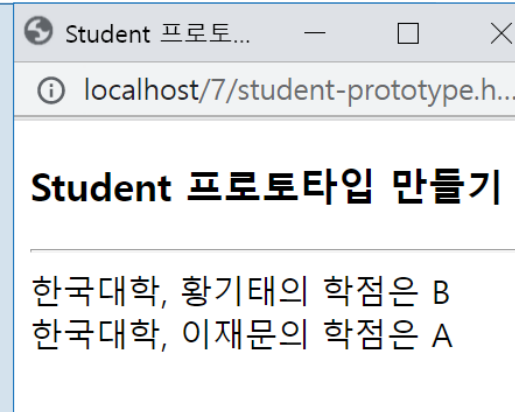
프로토타입

- 프로토타입(prototype)이란?
 - 객체의 모양을 가진 틀
 - 붕어빵은 객체이고, 붕어빵을 찍어내는 틀은 프로토타입
 - C++, Java에서는 프로토타입을 클래스라고 부름
 - Array, Date, String : 자바스크립트에서 제공하는 프로토타입
 - 객체 생성시 'new 프로토타입' 이용
 - let week = **new Array**(7); // Array는 프로토타입임
 - let hello = **new String**("hello"); // String은 프로토타입임

프로토타입 만드는 사례 : Student 프로토타입

- 프로토타입은 함수로 만든다
 - 프로토타입 함수를 **생성자 함수**라고도 함

```
// 프로토타입 Student 작성
function Student(name, score) {
  this.univ = "한국대학";
  this.name = name;
  this.score = score
  this.getGrade = function () { // getGrade() 메소드 작성
    if(this.score > 80) return "A";
    else if(this.score > 60) return "B";
    else return "F";
  }
}
```

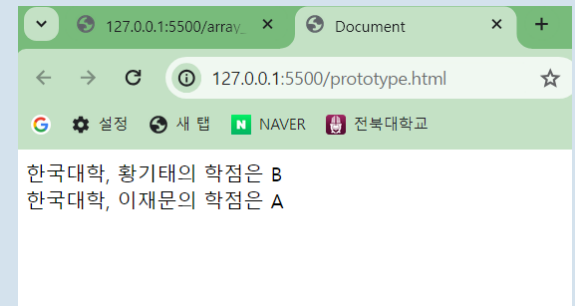


- new 연산자로 객체를 생성한다

```
let kitae = new Student("황기태", 75); // Student 객체 생성
let jaemoon = new Student("이재문", 93); // Student 객체 생성
document.write(kitae.univ + ", " + kitae.name + "의 학점은 " + kitae.getGrade() + "<br>");
document.write(jaemoon.univ + ", " + jaemoon.name + "의 학점은 " + jaemoon.getGrade() + "<br>")
```

prototype.html

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Document</title>
<script>
    function Student(name, score) {
        this.univ = "한국대학";
        this.name = name;
        this.score = score
        this.getGrade = function () {
            if(this.score > 80) return "A";
            else if(this.score > 60) return "B";
            else return "F";
        }
    }
</script>
</head>
<body>
<script>
    let kitae = new Student("황기태", 75);      // Student 객체 생성
    let jaemoon = new Student("이재문", 93);    // Student 객체 생성
    document.write(kitae.univ + ", " + kitae.name + "의 학점은 " + kitae.getGrade() + "<br>");
    document.write(jaemoon.univ + ", " + jaemoon.name + "의 학점은 " + jaemoon.getGrade() + "<br>")
</script>
</body>
</html>
```



생성자 함수를 이용한 객체 생성

- 이 방법은 생성자 함수를 정의하고 이 함수를 여러 번 호출하여서 원하는 만큼의 객체를 생성하는 방법이다.

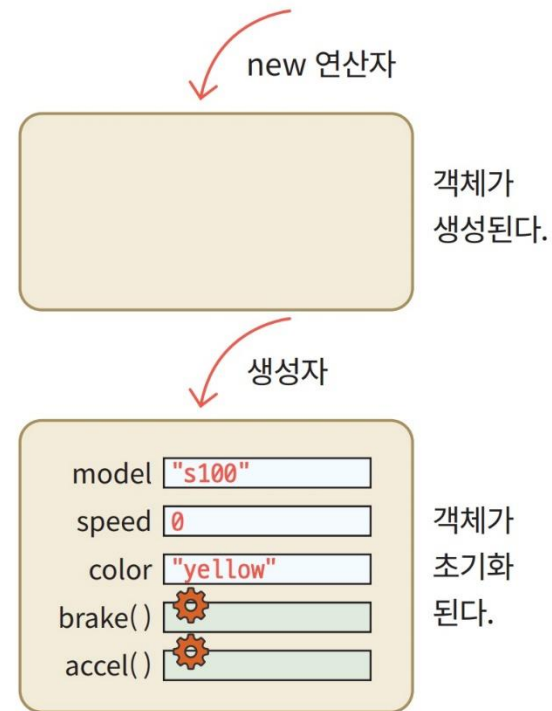
→ 생성자도 함수이다.
→ 생성자 이름은 항상 대문자로 한다.

```
function Car(model, speed, color) {  
  this.model = model;  
  this.speed = speed;  
  this.color = color;  
  this.brake = function () {  
    this.speed -= 10;  
  }  
  this.accel = function () {  
    this.speed += 10;  
  }  
}
```

this 키워드로
일반 변수와
객체 속성을
구별한다.

→ 객체의 속성

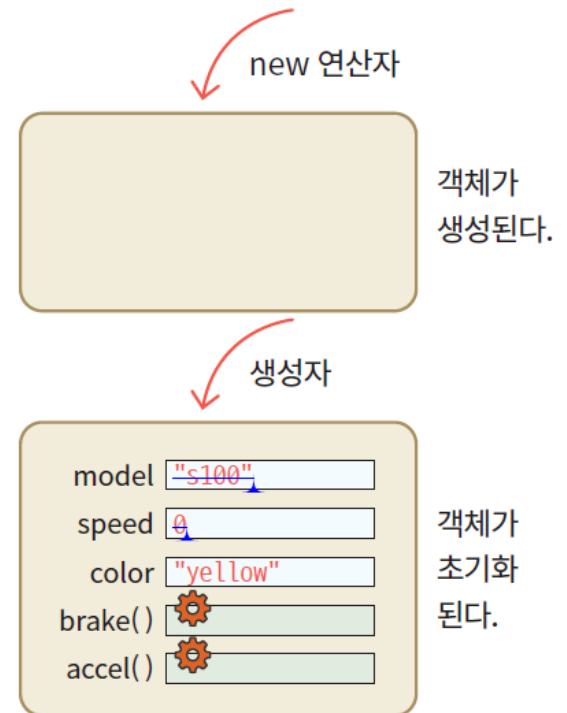
→ 객체의 메소드



생성자 함수를 이용한 객체 생성

- 생성자를 정의하였으면 객체는 다음과 같이 생성할 수 있다

```
let myCar = new Car("520d", 60, "white");  
myCar.color = "yellow";  
myCar.brake();
```

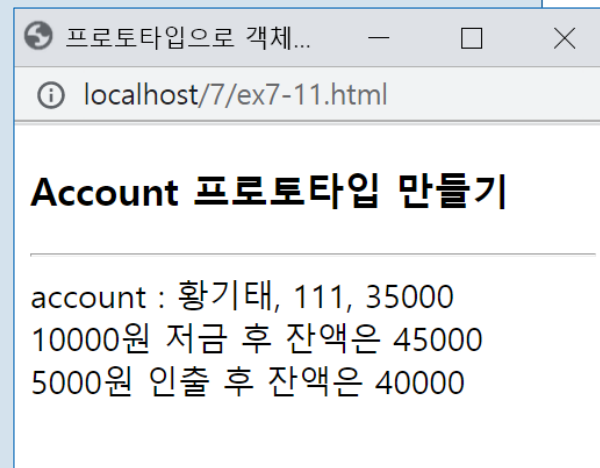


prototype_object1.html

```
<!DOCTYPE html>
<html> <head> <meta charset="utf-8"> <title>프로토타입으로 객체 만들기</title>
<script>
  function Account(owner, code, balance) {
    // 프로퍼티 만들기
    this.owner = owner; // 계좌 주인 프로퍼티 만들기
    this.code = code; // 계좌 코드 프로퍼티 만들기
    this.balance = balance; // 잔액 프로퍼티 만들기

    // 메소드 만들기
    this.inquiry = function () { return this.balance; }
    this.deposit = function (money) { this.balance += money; }
    this.withdraw = function (money) { // 예금 인출, money는 인출하는 액수 money가 balance보다 작다고 가정
      this.balance -= money;
      return money;
    }
  }
</script> </head>
<body>
<h3>Account 프로토타입 만들기</h3>
<hr>
<script>
  let account = new Account("황기태", "111", 35000);
  document.write("account : ");
  document.write(account.owner + ", ");
  document.write(account.code + ", ");
  document.write(account.balance + "<br>");

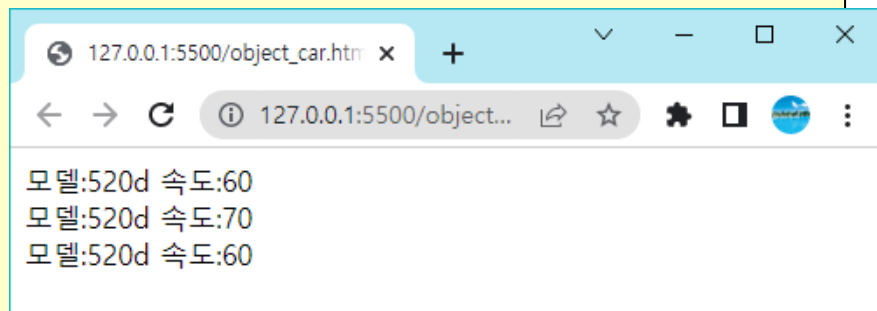
  account.deposit(10000); // 10000원 저금
  document.write("10000원 저금 후 잔액은 " + account.inquiry() + "<br>");
  account.withdraw(5000); // 5000원 인출
  document.write("5000원 인출 후 잔액은 " + account.inquiry() + "<br>");
</script>
</body> </html>
```



```

<!DOCTYPE html>
<html>
<body>
  <script>
    function Car(model, speed, color)
    {
      this.model=model;
      this.speed=speed;
      this.color = color;
      this.brake = function () {
        this.speed -= 10;
      }
      this.accel = function () {
        this.speed += 10;
      }
    }
    myCar = new Car("520d", 60, "red");
    document.write("모델:" + myCar.model + " 속도:" + myCar.speed + "<br />");
    myCar.accel();
    document.write("모델:" + myCar.model + " 속도:" + myCar.speed + "<br />");
    myCar.brake();
    document.write("모델:" + myCar.model + " 속도:" + myCar.speed + "<br />");
  </script>
</body></html>

```

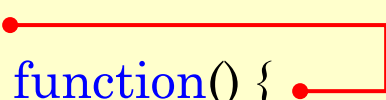


기존의 객체에 속성과 메소드 추가하기

- 자바스크립트에서는 기존에 존재하고 있던 객체에도 속성을 추가할 수 있는데, 생성자를 변경할 필요없이 단순히 값을 대입하는 문장을 적어주면 된다.

// 기존의 myCar 객체가 있다고 가정하자.

```
myCar.turbo = true;
myCar.showModel = function() {
    alert( "모델은 " + this.model + "입니다." )
}
```



새로운 속성과 메소드가 객체에 추가 된다.

prototype_object3.html

```
<!DOCTYPE html>
<html>
<body>
  <script>
    function Car(model, speed, color)  {
      this.model=model;
      this.speed=speed;
      this.color = color;
      this.brake = function () { this.speed -= 10; }
      this.accel = function () { this.speed += 10; }
    }
    myCar = new Car("520d", 60, "red");
    document.write("모델:" + myCar.model + " 속도:" + myCar.speed + "<br />");
    myCar.accel();
    document.write("모델:" + myCar.model + " 속도:" + myCar.speed + "<br />");
    myCar.brake();
    document.write("모델:" + myCar.model + " 속도:" + myCar.speed + "<br />");

    myCar.turbo = true;
    myCar.showModel = function() { alert( "모델은 " + this.model + "입니다." ) }
    document.write("터보:" + myCar.turbo + "<br />");
    myCar.showModel();
  </script>
</body>
</html>
```


for/in 루프

- for/in 루프는 객체 안의 모든 속성들에 대하여 어떤 처리를 반복할 수 있는 구조이다.

```
for ( 변수 in 객체 )
```

```
{
```

```
    문장;
```

```
}
```

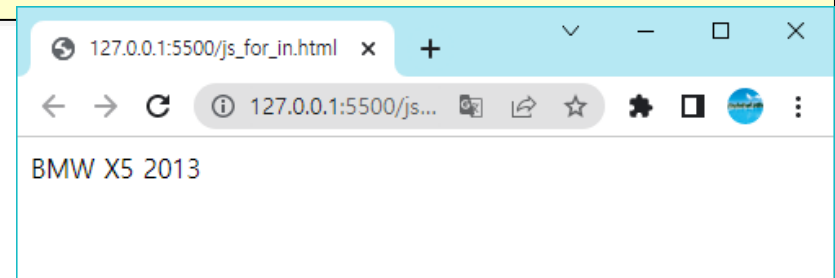
변수에 객체의 속성이
하나씩 대입되면서 반복한다.

예제 소스

js_for_in.html

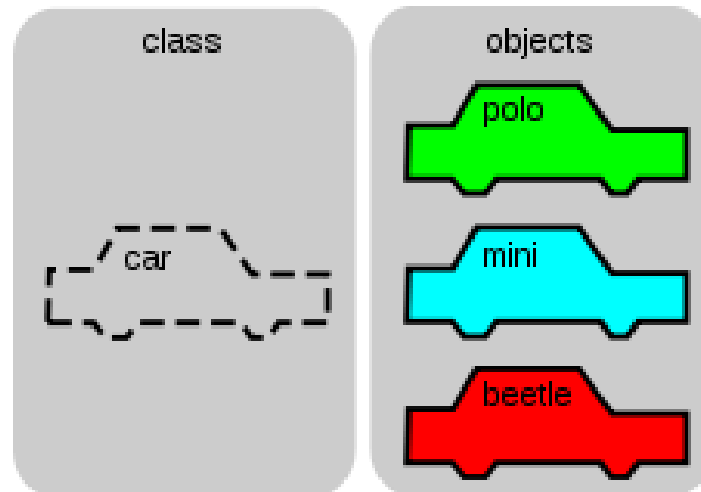
```
<script>
  let myCar = { make: "BMW", model: "X5", year: 2013 };
  let txt="";

  for (let x in myCar) {
    txt += myCar[x] + " ";
  }
  document.write(txt);
</script>
```



사용자 객체 생성하기 #2

- ECMAScript 2015부터 class라고 하는 키워드를 도입하였다. 이것은 파이썬이나 자바 언어의 클래스와 동일한 개념으로 이것을 사용하면 상당히 편리하게 클래스를 정의하고 객체를 찍어낼 수 있다.



```
<!DOCTYPE html>
<html>
<body>
  <h2 id="test"></h2>
```

```
<script>
```

```
class Dog {
  constructor(name, adoptedYear) {
    this.name = name;
    this.adoptedYear = adoptedYear;
  }
```

Constructor()함수는 생성자메소드로
반드시 이름이 constructor이어야한다.

```
  age() {
    let date = new Date();
    return date.getFullYear() - this.adoptedYear;
  }
```

클래스안에는 원하는 만큼의 메소드를
정의할 수 있다.

```
let myDog = new Dog("Mary", 2013);
document.getElementById("test").innerHTML = "나의 강아지의 이름은 " +
  myDog.name + "이고, 나이는 " + myDog.age() + "살입니다.";
```

```
</script>
```

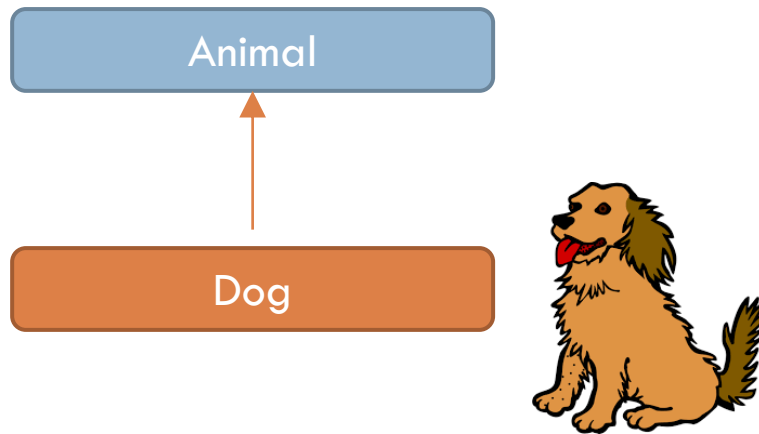
```
</body>
```

```
</html>
```

나의 강아지의 이름은 Mary이고, 나이는 9살입니다.

상속이 가능하다.

- 자바스크립트의 새로운 버전에서는 자바 언어와 동일한 방법으로 상속을 사용할 수 있다. 예를 들어서 Animal을 상속하여서 Dog 클래스를 만들 수 있다.



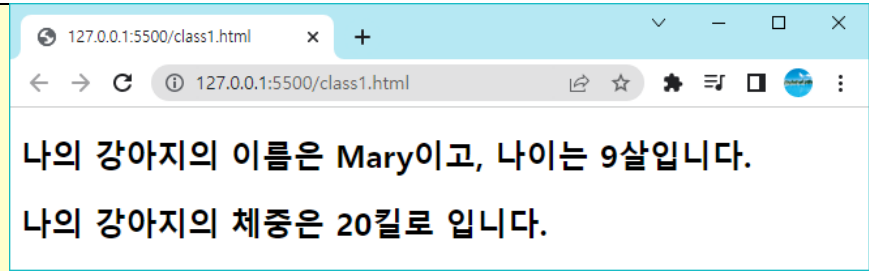
class_inheri.html

```
<!DOCTYPE html>
<html>
<body>
  <h2 id="test"></h2>
  <h2 id="test1"></h2>

  <script>
    class Animal {
      constructor(weight) {
        this.weight = weight;
      }
    }
    class Dog extends Animal {
      constructor(name, adoptedYear, weight) {
        super(weight);
        this.name = name;
        this.adoptedYear = adoptedYear;
      }
      age() {
        let date = new Date();
        return date.getFullYear() - this.adoptedYear;
      }
    }
  </script>
</body>
</html>
```

부모 클래스

Animal클래스를 상속받아서
Dog클래스를 정의한다는 의미



```
let myDog = new Dog("Mary", 2013, 20);
```

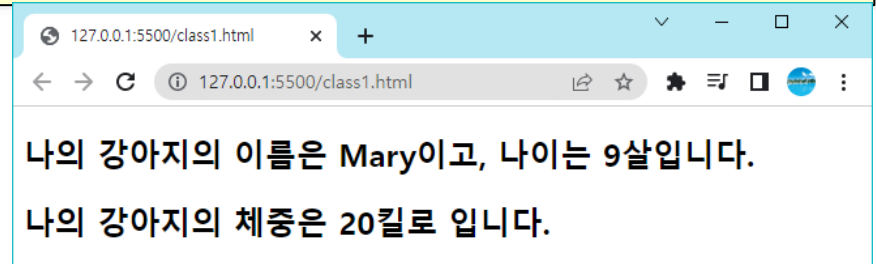
```
document.getElementById("test").innerHTML =  
    "나의 강아지의 이름은 " + myDog.name + "이고, 나이는 "  
    + myDog.age() + "살입니다.";
```

```
document.getElementById("test1").innerHTML =  
    "나의 강아지의 체중은 " + myDog.weight + "킬로 입니다.";
```

```
</script>
```

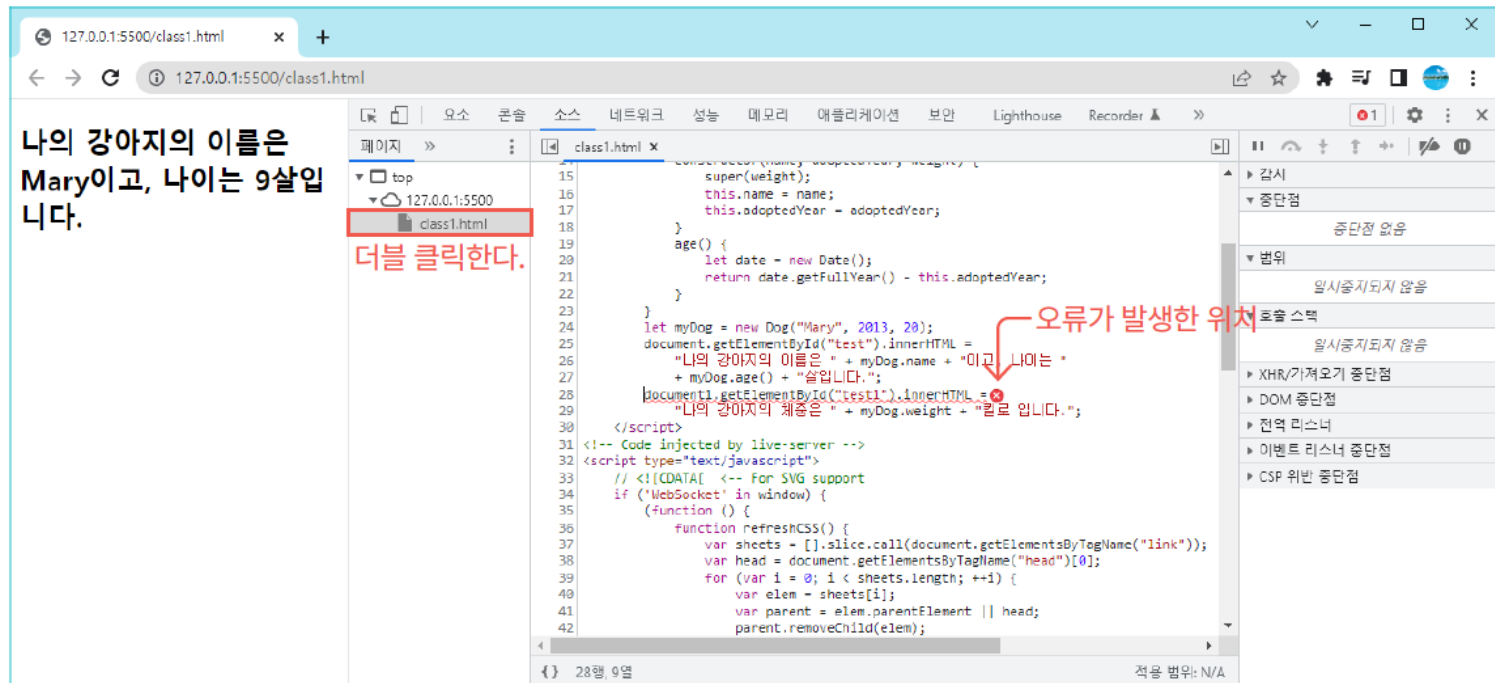
```
</body>
```

```
</html>
```



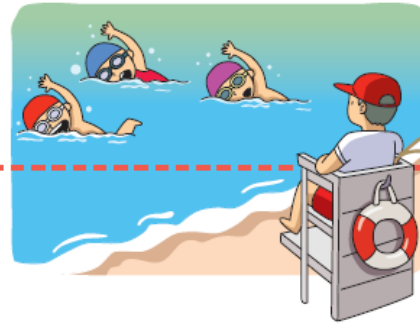
자바스크립트에서의 오류 처리

- 오류가 발생되면 자동적으로 실행이 중단되면서 콘솔을 통하여 다음과 같이 오류가 보고된다.



Try-Catch 구조

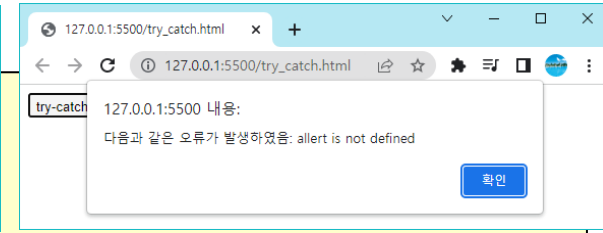
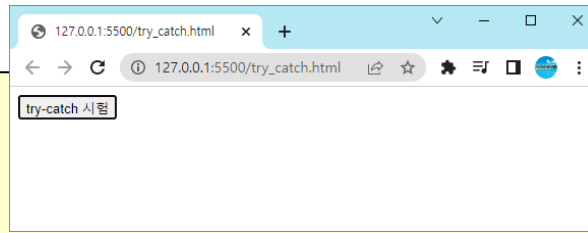
```
try
{
    // 예외가 발생할 수 있는 코드
}
catch (변수)
{
    // 예외를 처리하는 코드
}
```



try 블록에서
오류가 발생하면
처리합니다.

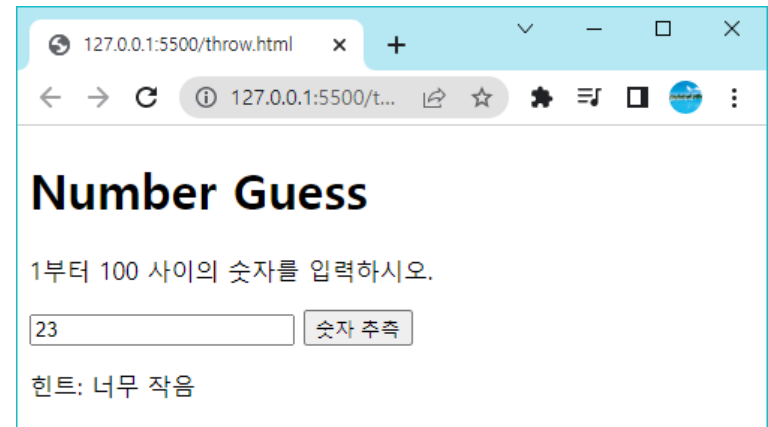
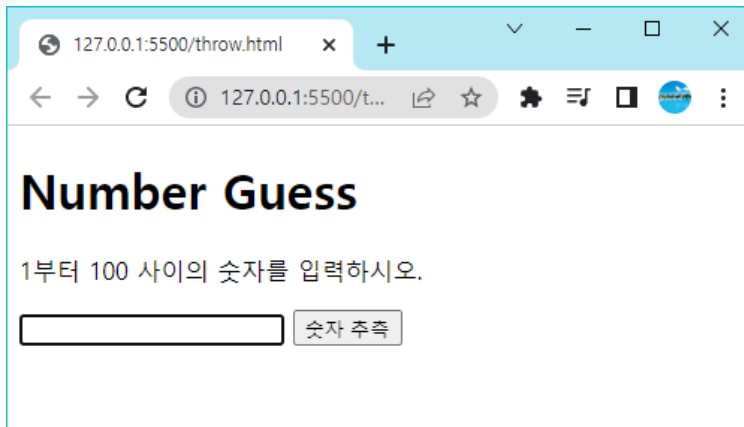
try_catch.html

```
<!DOCTYPE html>
<html>
<head>
  <script>
    let msg = "";
    function test() {
      try {
        alert("Hello World!");
      }
      catch (error) {
        msg = "다음과 같은 오류가 발생하였음: " + error.message;
        alert(msg);
      }
    }
  </script>
</head>
<body>
  <input type="button" value="try-catch 시험" onclick="test()" />
</body>
</html>
```

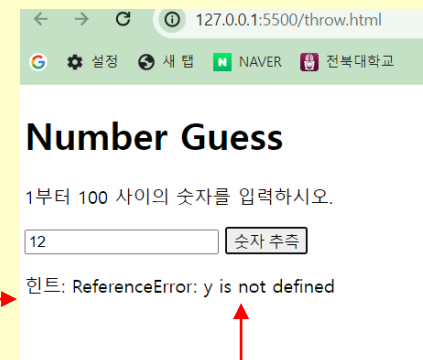


throw 문장

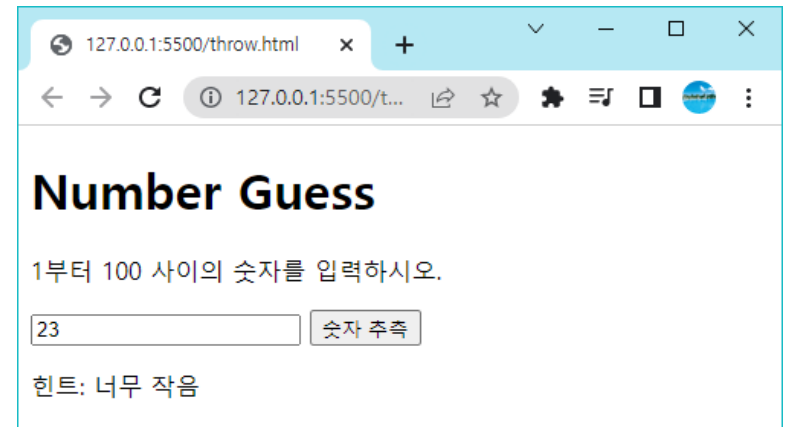
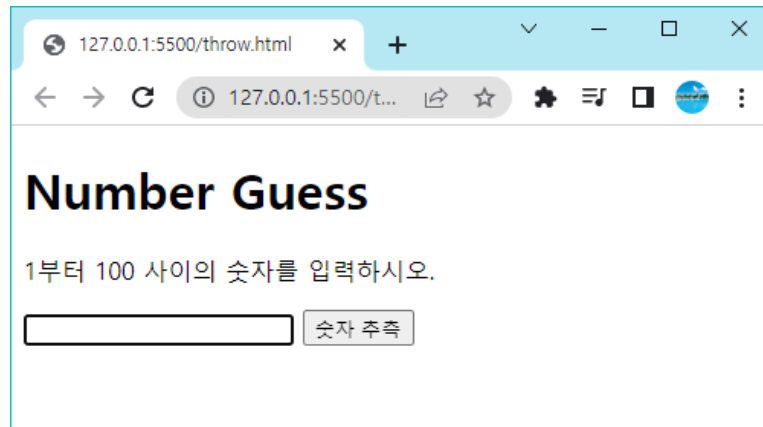
- throw 문장은 개발자가 오류를 생성할 수 있도록 한다. 예외를 발생시키는 것을 예외를 던진다(throw)고 한다고 표현한다.
- 개발자는 자신이 어떤 기준을 정하고 이 기준에 맞지 않으면 사용자에게 어떤 경고 메시지를 줄 수 있다.



```
<!DOCTYPE html>
<html>
<body>
  <script>
    let solution= 53;
    function test() {
      try {
        let x = document.getElementById("number").value;
        if (x == "") throw "입력없음";           // 만약 x 대신 y라면 error
        if (isNaN(x)) throw "숫자가 아님";
        if (x > solution) throw "너무 큼";
        if (x < solution) throw "너무 작음";
        if (x == solution) throw "성공";
      }
      catch (error) {
        let y = document.getElementById("message");
        y.innerHTML = "힌트: " + error;
      }
    }
  }
</script>
</body>
</html>
```

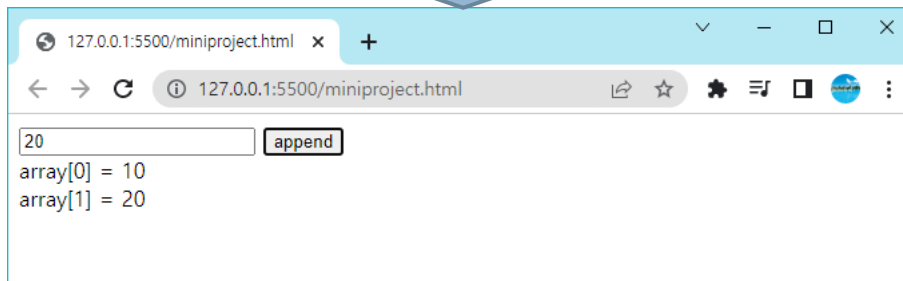
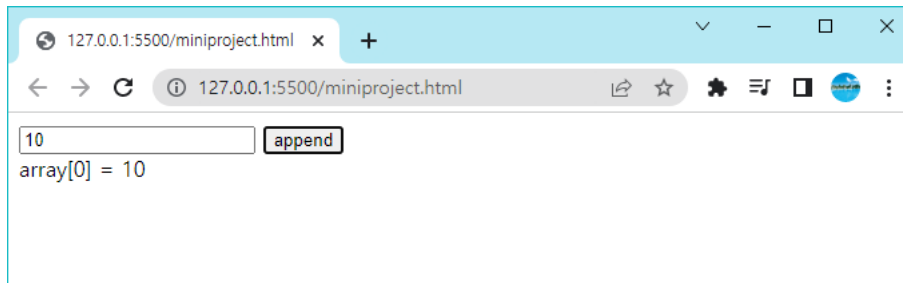


```
</script>
<h1>Number Guess</h1>
<p>1부터 100 사이의 숫자를 입력하십시오.</p>
<input id="number" type="text">
<button type="button" onclick="test()">숫자 추측</button>
<p id="message"></p>
</body>
</html>
```



Mini Project: 배열에 값 저장하기

- 버튼을 누르면 `<input>` 요소의 값을 읽어서 배열에 저장한다. 배열에 저장된 값을 모두 꺼내서 화면의 아래쪽에 표시한다.



miniproject.html

```
<!DOCTYPE html>
<html>
<head>
  <meta charset=utf-8 />
</head>
<body>
  <input type="text" id="text1"> </input>
  <input type="button" id="button1" value="append" onclick="add_array();" > </input>
  <div id="result"> </div>
  <script>
    let index = 0;
    let array = Array();
    function add_array() {
      array[index] = document.getElementById("text1").value;
      index++;
      let str = "";
      for (let i = 0; i < array.length; i++) {
        str += "array[" + i + "] = " + array[i] + "<br/>";
      }
      document.getElementById("result").innerHTML = str;
    }
  </script>
</body>
</html>
```

