JavaScript

◆ 배열

정수아

Contents

01 배열

배열

❖ 배열

- 데이터 값을 하나의 목록으로 생성
- 여러 개의 원소들을 연속적으로 저장하고, 전체를 하나의 단위로 다루는 데이터 구조

- 예) 가장 좋아하는 과일 3가지 목록

```
var 과일TOP3 = [“딸기”, “사과”, “바나나”]
```

- 변수 3개를 작성하지 않고, 과일TOP3라는 배열 하나만 생성

배열을 배워야 하는 이유

❖ **과일 별로 변수를 만들어 보자.**

```
var 과일1 = “딸기”
var 과일2 = “사과”
var 과일3 = “바나나”
var 과일4 = “포도”
var 과일5 = “수박”
var 과일6 = “참외”
var 과일7 = “키위”
var 과일8 = “망고”
var 과일9 = “체리”
```

❖ **만약 1000가지의 과일 변수를 만들어야 한다면?**

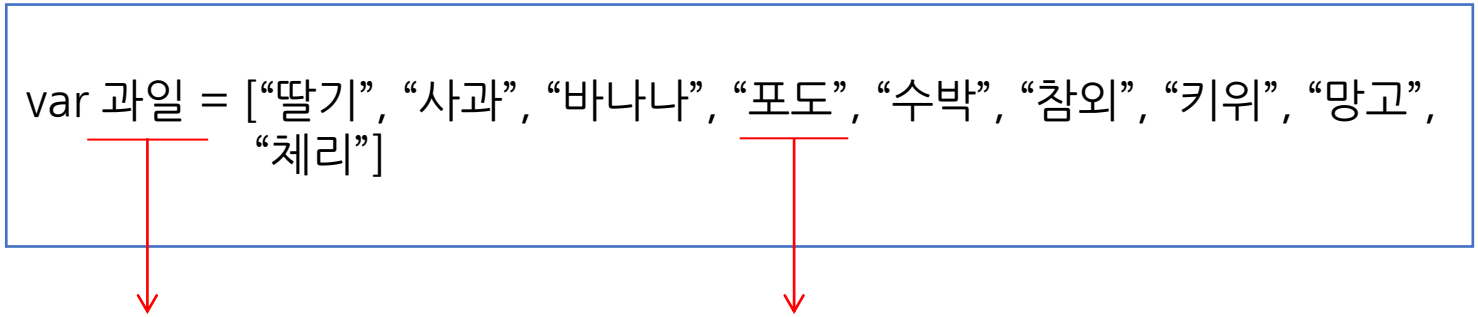
- 변수 1000개를 생성해야 하나?

배열 만들기

❖ 배열 생성

- 대괄호([ ]) 사용
- 대괄호 안에 값을 입력. 여러 개일 경우 쉼표로 구분

```
var 과일 = [“딸기”, “사과”, “바나나”, “포도”, “수박”, “참외”, “키위”, “망고”,
            “체리”]
```



배열 만들기

❖ 배열 생성

과일		
	딸기	과일[0]
	사과	과일[1]
	바나나	과일[2]
	포도	과일[3]
	수박	과일[4]
	참외	과일[5]
	키위	과일[6]
	망고	과일[7]
	체리	과일[8]

배열 만들기

❖ 배열 생성

```
var 점수 = [10, 20, 30, 50, 100];
```

점수	10	20	30	50	100
	점수[0]	**점수[1]**	**점수[2]**	**점수[3]**	**점수[4]**

배열 원소 접근하기

❖ **배열 원소 접근**

- 원하는 원소 색인(index)에 대괄호 사용
- 배열의 첫 번째 원소가 0번, 두 번째 원소가 1번

```
>> 과일[0];
<< “딸기 "
```

```
>> 과일[3];
<< “포도 "
```

배열 원소 설정 및 변경

❖ 대괄호 안의 색인을 변경

- 배열의 원소를 설정 및 변경, 추가 가능

```
과일[0] = “스트로베리”;
```

```
var 과일 = [“스트로베리”, “사과”, “바나나”, “포도”, “수박”, “참외”, “키위”,
            “망고”, “체리”]
```

배열 원소 설정 및 변경

❖ 배열에 새로운 원소 추가

- 빈 배열 생성
  - var 과일 = []
- 0~8번 색인에 각각 과일 이름 삽입

```
> var 과일 = [];
  과일[0] = "딸기"
  과일[1] = "사과"
  과일[2] = "바나나"
  과일[3] = "포도"
  과일[4] = "수박"
  과일[5] = "참외"
  과일[6] = "키위"
  과일[7] = "망고"
  과일[8] = "체리"

  과일;
< ▶ (9) ['딸기', '사과', '바나나', '포도', '수박', '참외', '키위', '망
    고', '체리']
```

배열 원소 설정 및 변경

❖ **배열에 새로운 원소 추가**

- 원소는 어떤 색인이든 추가가 가능하다.
- 33번 색인에 과일 추가

```
> 과일[33] = "메론";
  과일;
< ▶ (34) ['딸기', '사과', '바나나', '포도', '수박', '참외', '키위', '망
  고', '체리', empty × 24, '메른']
```

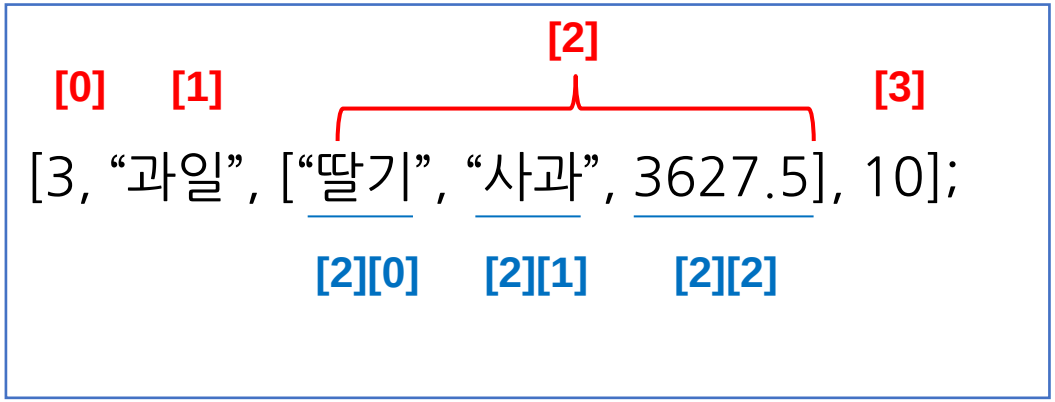
한 배열에서 여러 자료형 사용

❖ **서로 다른 자료형을 원소로 가진 배열 생성**

- 숫자, 문자열, 배열

```
var 과일과숫자 = [3, “과일”, [“딸기”, “사과”], 10];
```

❖ **대괄호를 사용하여 배열 속의 배열의 원소에 접근**



[실습] 내부 배열 원소에 접근

```
>> 과일과숫자[2];
<< [“딸기”, “사과”, 3627.5]
```

```
>> 과일과숫자[2][0];
<< “딸기”
```

```
>> 과일과숫자[2][1];
<< “사과”
```

배열 다루기

❖ **배열은 프로퍼티와 메서드로 다룬다.**

- 프로퍼티 : 배열에 대한 정보
- 메서드 : 배열을 변경하거나 새로운 배열을 반환

배열 다루기

❖ 배열의 길이

- 배열 내 원소의 개수를 확인
  - length 프로퍼티
  - 배열이름.length

```
>> var 엑스맨 = [“울버린”, “미스틱”, “비스트”]
   엑스맨[0];
<< “울버린”
```

```
>> 엑스맨[1];
<< “미스틱”
```

```
>> 엑스맨[2];
<< “비스트”
```

```
>> 엑스맨.length;
<< 3
```

배열 다루기

❖ 배열에 원소 추가

- 배열 끝에 새로운 원소를 추가
  - push 메서드 호출
  - 배열이름.push(추가할_원소_이름)

```
>> var 동물 = [];
   동물.push("고양이");
<< 1
```

```
>> 동물.push("강아지");
<< 2
```

```
>> 동물.push("라마");
<< 3
```

```
>> 동물;
<< ["고양이", "강아지", "라마"]
```

```
>> 동물.length;
<< 3
```

배열 다루기

❖ **메서드 호출이란?**

- 컴퓨터 언어로 어떤 동작을 실행하는 행위
- push 메서드를 호출하면
  - 첫째, 괄호의 한 원소가 배열에 추가
  - 둘째, 배열의 길이가 반환

배열 다루기

❖ 배열에 원소 추가

- 배열 시작에 새로운 원소를 추가
  - unshift 메서드 호출
  - 배열이름.unshift(추가할_원소_이름)

```
>> 동물;
<< [“고양이”, “강아지”, “라마”]
```

```
>> 동물[0];
<< “고양이”
```

```
>> 동물.unshift(“원숭이”);
<< 4
```

```
>> 동물;
<< [“원숭이”, “고양이”, “강아지”, “라마”]
```

```
>> 동물.unshift(“북극곰”);
<< 5
```

```
>> 동물;
<< [“북극곰”, “원숭이”, “고양이”, “강아지”, “라마”]
```

배열 다루기

❖ 배열의 원소 제거

- 배열의 마지막 원소를 제거
  - pop 메서드 호출
  - 배열이름.pop()
  - 마지막 원소를 제거한 후 제거된 원소를 값으로 반환

```
>> 동물;
<< ["북극곰", "원숭이", "고양이", "강아지", "라마"]
```

```
>> 마지막동물 = 동물.pop();
   마지막동물;
<< "라마"
```

```
>> 동물;
<< ["북극곰", "원숭이", "고양이", "강아지"]
```

```
>> 동물.pop();
<< "강아지"
```

```
>> 동물;
<<  ["북극곰", "원숭이", "고양이"]
```

배열 다루기

❖ 배열의 원소 제거

- 배열의 첫 번째 원소를 제거
  - shift 메서드 호출
  - 배열이름.shift()
  - 첫 번째 원소를 제거한 후 제거된 원소를 값으로 반환

```
>> 동물;
<< [“북극곰”, “원숭이”, “고양이”]
```

```
>> var 첫번째동물 = 동물.shift();
   첫번째동물;
<< “북극곰”
```

```
>> 동물;
<< [“원숭이”, “고양이”]
```

배열 다루기

❖ 배열 결합하기

- 배열 두 개를 결합해서 하나의 배열로 생성
- 첫번째배열.concat(두번째배열)

- 예) 포유류와 파충류의 배열을 합쳐보자.

```
>> var 포유류 = [“알파카“, “원숭이”, “설인“];
   var 파충류 = [“보아뱀”, “고질라”];
   var 포유류파충류 = 포유류.concat(파충류);
   포유류파충류;
<< [“알파카”, “원숭이”, “설인”, “보아뱀”, “고질라”]
```

```
>> 포유류;
<< [“알파카”, “원숭이”, “설인”]
```

```
>> 파충류;
<< [“보아뱀”, “고질라”]
```

- 두 배열의 결합되었다고 해서, 원본 배열이 변하지는 않음

배열 다루기

❖ 여러 배열 결합하기

- 두 개 이상의 배열을 결합해서 하나의 배열로 생성
- 결합하려는 배열을 차례로 괄호 안에 넣고 쉼표로 구분

```
>> var 포유류 = [“알파카”, “원숭이”, “설인”];
   var 파충류 = [“보아뱀”, “고질라”];
   var 조류 = [“앵무새”, “도도새”];
   var 모든동물 = 포유류.concat(파충류, 조류);
   모든동물;
<< [“알파카”, “원숭이”, “설인”, “보아뱀”, “고질라”, “앵무새”, “도도새”]
```

배열 다루기

❖ 배열 원소 색인 찾기

- 원하는 배열 원소의 색인을 찾는다.
- 배열이름.indexOf(원소)

- 예) 색상.indexOf(“파랑“)

```
>> var 색상 = [“빨강“, “초록”, “파랑”];
   색상.indexOf(“파랑”);
<< 2
```

```
>> 색상.indexOf(“초록”);
<< 1
```

- 예) 배열에 없는 원소의 값을 물어보면 -1 반환

```
>> 색상.indexOf(“보라”);
<< -1
```

배열 다루기

❖ 배열을 문자열로 만들기

- 배열에 포함된 모든 원소를 하나의 문자열로 생성
- 배열이름.join()
- 모든 원소를 쉼표로 구분해 넣은 문자열 반환

```
>> var 착한동물 = [“원숭이”, “고양이”, “물고기”, “도마뱀”]
   착한동물.join();
<< “원숭이, 고양이, 물고기, 도마뱀 "
```

- 쉼표를 구분자로 사용하고 싶지 않다면?
  - 배열이름.join(separator)

```
>> 착한동물.join(“ - ”);
<< “원숭이 - 고양이 - 물고기 - 도마뱀 "
```

```
>> 착한동물.join(“ * ”);
<< “원숭이*고양이*물고기*도마뱀 "
```

```
>> 착한동물.join(“ 보다 ”);
<< “원숭이 보다 고양이 보다 물고기 보다 도마뱀 "
```

배열 다루기

❖ 배열을 문자열로 만들기

- 문자열로 바꾸고 싶은 배열이 있을 때 유용

```
>> var 내주소 = [“서울시”, “종로구”, “세종로”, “경복궁”];
   내주소.join(“ ”);
<< 서울시 종로구 세종로 경복궁
```

- 만약, join을 모른다면 다음과 같이 코드를 작성해야 함
  - 주소가 더 길다면 더 많은 코드가 필요

```
>> 내주소[0] + “ ” + 내주소[1] + “ ” + 내주소[2] + “ ” + 내주소[3];
<< 서울시 종로구 세종로 경복궁
```

배열 활용하기

❖ 집으로 오는 길 찾기

- push로 배열 만들기

```
>> var 건물 = [];
   건물.push("우리집");
   건물.push("집 앞 도로");
   건물.push("깜박이는 가로등");
   건물.push("물 새는 소화전");
   건물.push("소방서");
   건물.push("고양이 구출 센터");
   건물.push("우리 모교");
   건물.push("친구네 집");
<< 8
```

배열 활용하기

❖ 집으로 오는 길 찾기

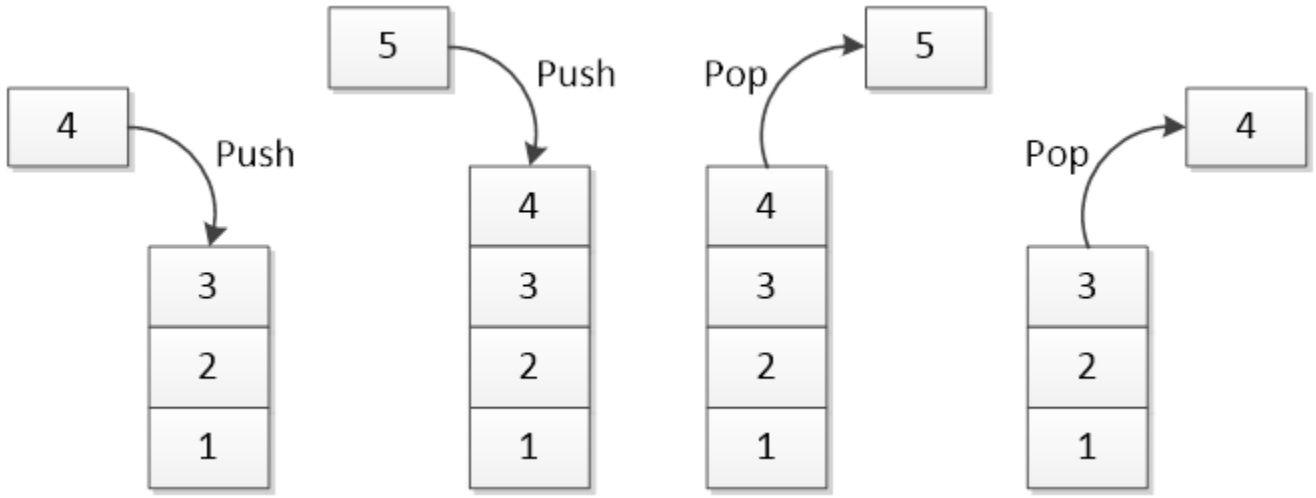
- pop으로 하나씩 지우기

```
>> 건물.pop();
<< "친구네 집"

>> 건물.pop();
<< "우리 모교"

>> 건물.pop();
<< "고양이 구출 센터"

>> 건물.pop();
<< "소방서"

>> 건물.pop();
<< "물 새는 소화전"

>> 건물.pop();
<< "깜박이는 가로등"

>> 건물.pop();
<< "집 앞 도로"

>> 건물.pop();
<< "우리집"
```

스택

❖ **스택(stack)**

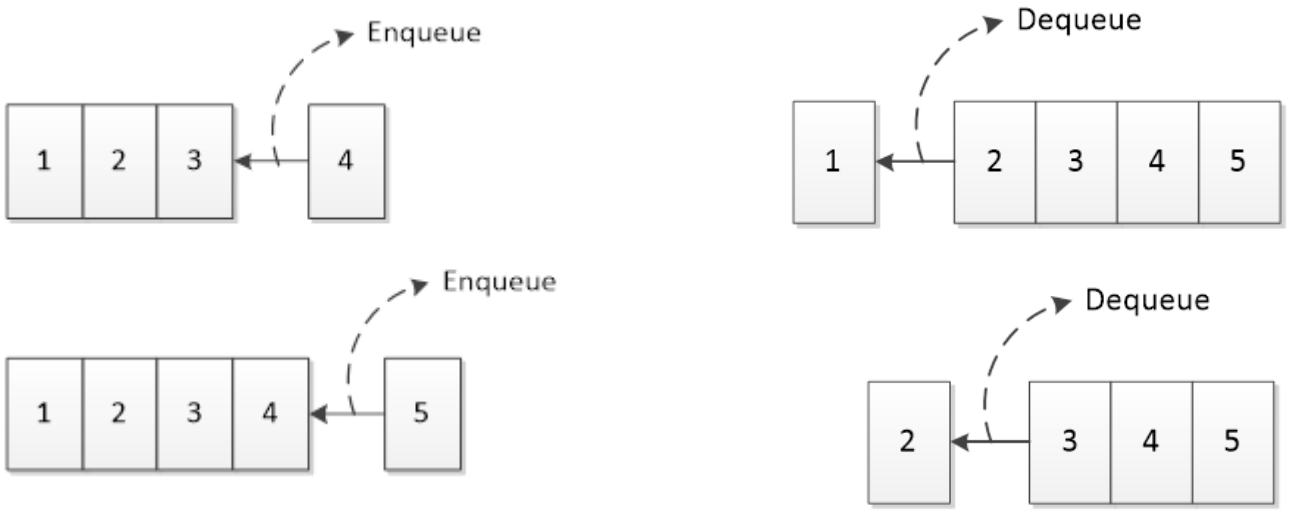
- 맨 처음에 추가한 원소가 맨 마지막에 제거
- LIFO(Last In First Out)



큐

❖ 큐(queue)

- 데이터를 차례대로 입력하고, 입력된 순서대로 하나씩 꺼내서 처리하기 위해 사용
- 맨 처음에 추가된 원소를 가장 먼저 제거
- FIFO(First In First Out)



Math.random()

❖ **Math.random()**

- 0과 1 사이의 숫자 하나를 무작위로 선택해서 반환
- 1 미만의 숫자만 반환
- 더 큰 숫자를 원하면 Math.random()이 반환한 값에 곱셈

```
> Math.random();
< 0.8218628243829418
> Math.random();
< 0.43788566669741624
> Math.random();
< 0.04803900946344086

> Math.random() * 10;
< 6.114258754984563
> Math.random() * 10;
< 1.178696775319612
> Math.random() * 10;
< 4.53824155592166
```

Math.floor()

❖ **Math.floor()**

- 소수점 이하의 숫자를 버리고 정수를 생성

```
>> Math.floor(3.123456);
<< 3
```

```
>> Math.floor(9.99999);
<< 9
```

```
>> Math.floor(0.123456);
<< 0
```

```
>> Math.floor(Math.random()*10);
<< 5
```

Math.floor()

❖ **Math.random()과 Math.floor() 활용**

- Math.random()을 통해 나온 값에 배열의 길이를 곱한 후,
- Math.floor()를 호출
- 0, 1, 2, 3 중 하나의 숫자가 무작위로 반환

```
>> Math.floor(Math.random() * 4);
<< 2
```

- 배열의 원소를 랜덤으로 선택

```
>> var 무작위단어 = [“폭발”, “동굴”, “공주”, “펜”]
   var 무작위색인 = Math.floor(Math.random() * 4);
   무작위단어[무작위색인];
<< “펜”
```

[실습] 의사 결정기

〈답변〉
“좋은 생각이에요.”
“네, 꼭 해보세요.”
“별로 좋은 생각 같지 않아요.”
“오늘은 안 해도 되지 않을까요?”
“컴퓨터는 하지 말라고 하네요.”

〈질문〉
숙제를 해야 할까요?

[실습] 의사 결정기

❖ 입력한 문장들을 랜덤으로 선택해주는 프로그램

```
> var 답변 = [
      "좋은 생각이에요.",
      "네, 꼭 해보세요.",
      "별로 좋은 생각 같지 않아요.",
      "오늘은 안 해도 되지 않을까요?",
      "컴퓨터는 하지 말라고 하네요."
  ];
  // 밀크쉐이크 한 잔 더 마셔도 될까요?
  답변[Math.floor(Math.random() * 5)];
< "별로 좋은 생각 같지 않아요."
> // 숙제를 해야 할까요?
  답변[Math.floor(Math.random() * 5)];
< "컴퓨터는 하지 말라고 하네요."
```

[실습] 인디언 이름 제조기 만들기

〈색상〉
“푸른”, “붉은”, “검은”, ”하얀

〈자연〉
“늑대”, “태양”, “독수리”, “바람”

〈단어〉
“눈물”, “환생”, “기상”, “일격”, “유령”

문제!
색상, 자연, 단어 배열에서 무작위로 단어를 하나씩 골라
인디언의 이름을 만드세요.

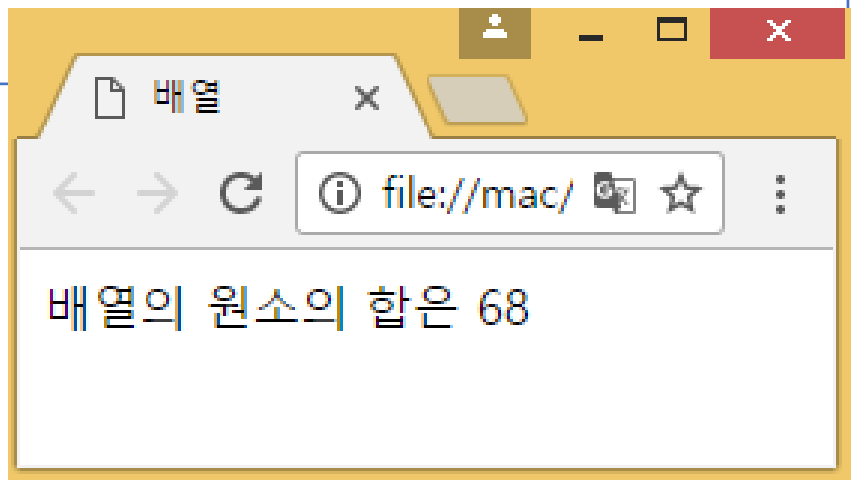
결과 예시!
검은 늑대의 일격

[실습] 인디언 이름 제조기 만들기

```
> var 색상목록 = ["푸른", "붉은", "검은", "하얀"];
  var 자연목록 = ["늑대", "태양", "독수리", "바람"];
  var 단어목록 = ["눈물", "환생", "기상", "일격", "유령"];

  // 색상목록 배열에서 무작윈 단어를 하나 고릅니다.
  var 색상 = 색상목록[Math.floor(Math.random() * 4)];

  // 자연목록 배열에서 무작위 단어를 하나 고릅니다.
  var 자연 = 자연목록[Math.floor(Math.random() * 4)];

  // 단어목록 배열에서 무작위 단어를 하나 고릅니다.
  var 단어 = 단어목록[Math.floor(Math.random() * 4)];

  // 무작위로 고른 문자열을 한 문장으로 조합합니다.
  var 인디언이름 = 색상 + " " + 자연 + "의 " + 단어 + "!!";
  인디언이름;
< "검은 늑대의 일격!!"
```

[실습] for문을 이용하여 배열의 합 구하기

```
var sum = 0;
var n = [4, 5, -2, 28, 33];

for(var i = 0; i < 5; i++)
{
    sum = sum + n[i];
}

document.write("합은 " + sum);
```




THANK YOU