

다양한 메서드(Data, String, Math 등)



다양한 메서드(Data, String, Math 등)

학습 순서를 확인하고 넘어갑시다!

- 시작하기: 요리 결과 확인하기 (Console)
- 텍스트 조작하기 (String)
- 배열 다루기(Array Basic)
- 배열 순회 문법 (Array ES6+)
- JavaScript Object Notation (JSON)
- 코드 확 줄이기 (Arrow Function)
- JavaScript Object (객체)
- [추가자료] Spread (...) 와 구조 분해 할당



시작하기: 요리 결과 확인하기 (Console)

디버깅(Debugging)

요리사가 음식을 만들면 손님에게 내놓기 전에 맛을 보거나(Testing), 접시에 예쁘게 담아야(Display)겠죠?

'console'이라는 작업대 위에서 우리는 재료의 상태를 확인하고, 문제가 없는지 점검합니다.

이처럼 확인하는 과정을 디버깅(Debugging)이라고 합니다.

The screenshot shows a browser's developer tools with the JavaScript console tab open. On the left, there is a code editor window containing the following JavaScript code:

```
var x = 10;  
var y = 25;  
var z = x + y;
```

To the right of the code editor, a purple box contains the text: "이 코드가 실행되면 터미널 혹은 콘솔에 그 결과가 출력됩니다." (This code will be output to the terminal or console when run).

Below the code editor, a green box highlights the line of code: `console.log("x + y = " + z);`. A large blue arrow points from this line down to the "Output" section on the right.

The "Output" section is a black box displaying the result of the console log statement: `x + y = 35`.

At the top right of the developer tools interface, there are tabs labeled "Input arguments", "Output", and "Generated files".



'console' 메서드

데이터(재료)가 잘 만들어졌는지 눈으로 확인하는 도구 중 하나인 'console' 메서드를 살펴보겠습니다.

console.log(재료)

콤마(,)를 찍어서 여러 재료를 한 번에 보여줄 수도 있습니다.

```
1 // 기본 사용법
2 let name = "철수";
3 console.log(name); // 출력: 철수
4
5 // 여러 재료를 한 번에 보기
6 let age = 25;
7 let city = "서울";
8 console.log(name, age, city); // 출력: 철수 25 서울
9
10 // 디버깅: 코드 실행 지점 확인
11 console.log("여기까지 실행됨!");
12
13 // 라벨과 함께 보기 (추천!)
14 console.log("이름:", name);
15 console.log("나이:", age);
```



'console' 메서드

console.table(재료)

재료를 보기 좋은 표(Table)에 정리해서 보고서로 만드는 것입니다.

복잡한 데이터, 특히 List of Objects (회원 명단 같은 거)를 봐야 하는 상황이라면 엑셀처럼 깔끔하게 정리해 줘서 한눈에 파악하기 좋습니다.

```
1 // [재료 준비] 회원 명단
2 const users = [
3   { name: "철수", age: 20, grade: "A" },
4   { name: "영희", age: 25, grade: "B" },
5   { name: "민수", age: 32, grade: "A" }
6 ];
7
8 // 표로 깔끔하게 출력 (★ 구조 파악에 최고!)
9 console.table(users);
10
11 // 출력 결과 (브라우저 콘솔에서):
12 // ┌─────────┬─────────┬─────────┐
13 // | (index) | name   | age     |
14 // └─────────┼─────────┼─────────┘
15 // | 0       | '철수' | 20      |
16 // | 1       | '영희' | 25      |
17 // | 2       | '민수' | 32      |
18 // └─────────┼─────────┼─────────┘
19
20 // 특정 필드만 선택해서 보기
21 console.table(users, ["name", "age"]);
```



'console' 메서드

console.warn(재료) & console.error(재료)

요리 과정에서 "주의하세요!"(노란 경고장) 혹은 "불이 났어요!"(빨간 비상벨)를 울리는 것과 같습니다.

치명적이진 않지만 조심해야 할 때는 warn, 코드가 멈출 정도로 심각할 때는 error

상황에 맞는 로그를 사용해서
개발자 눈에 띄게 만듭니다.

실제로 코드를 배포한다면
이러한 경고, 에러로그는
고객에게 보이지 않도록 설정한답니다.

```
1 // [재료 준비] 회원 명단
2 const users = [
3   { name: "철수", age: 20, grade: "A" },
4   { name: "영희", age: 25, grade: "B" },
5   { name: "민수", age: 32, grade: "A" }
6 ];
7
8 // 도구 1: 그냥 보여주기 (가장 흔하게 씀)
9 console.log("회원 목록:", users);
10
11 // 도구 2: 표로 정리해서 보여주기 (구조 파악에 최고! ★)
12 console.table(users);
13
14 // 도구 3: 경고 메시지 (노란색 배경으로 출력됨)
15 console.warn("주의: 회원 정보가 아직 저장되지 않았습니다.");
```

점 하나, 괄호 하나에도 다 이유가 있답니다!

본격적으로 도구를 쓰기 전에, 사용법(Syntax)을 확실히 익혀봅시다.

console.log(..) 를 재료.도구(옵션) 의 형태로 바꿔서 이해해 봅시다!

아래 이미지와 같은 형태로 하나씩 살펴보겠습니다!

```
1  console.log ( .. )
2 //    재료 .도구( 옵션 )
```



점(.)의 의미

점(.)의 의미: "너의 주머니를 열어라"

"재료."는 "이 재료가 가지고 있는 기능 목록을 보여줘"라는 뜻

```
1      console.log ( ... )  
2  //      재료 .도구( 옵션 )
```

사과에는 '껍질 깎기' 기능이 있지만, 돌멩이에는 없겠죠?
점을 찍으면 그 재료가 쓸 수 있는 도구들만 나옵니다.

사람을 예시로 들어보겠습니다

```
사람.숨쉬기(); // 기본적으로 사람은 숨 쉬는 게 가능!  
사람.날기(); // 사람이 날수는 없으니 불가능!
```



괄호()의 의미

괄호()의 의미: "작동 버튼"

"도구"라는 이름만 부르면 도구를 쳐다보기만 할 뿐입니다.

도구()라고 괄호를 열고 닫아야 "지금 당장 실행해!(Execute)"라고 버튼을 누르는 것입니다.

주의!

괄호를 빼먹어도

Javascript는 에러를 뱉지 않습니다.

하지만! 아무 일도 일어나지 않는

침묵하는 버그가 발생하게 됩니다.

이 부분은 "객체"를

제대로 다루기 시작하는 시점에

이해할 수 있게 됩니다.

```
1 // 예제: 괄호 있을 때 vs 없을 때의 차이
2 // [재료 준비] 간단한 인사 함수
3 function sayHello() {
4     return "안녕하세요!";
5 }
6
7 // 1. 괄호 없이 - 함수 자체를 보기만 함 (실행 X)
8 console.log(sayHello);
9 // 출력: [Function: sayHello] (함수 정의만 보여줌)
10
11 // 2. 괄호 있음 - 함수를 실행함 (실행 O)
12 console.log(sayHello());
13 // 출력: 안녕하세요! (함수가 실행되어 결과가 나옴)
```



괄호 안의 값(인자/Argument)

괄호 안의 값(인자/Argument): "구체적인 옵션"

믹서기를 돌릴 때 '강하게' 돌릴지 '약하게' 돌릴지 정하는 것과 같습니다.

빈 괄호 (): "기본 설정대로 해."

값이 있는 괄호 ("옵션"): "이 옵션을 넣어서 작동해."

옵션이 여러 개라면 콤마(,)로 구분해서 넣어줍니다.

이렇게 이해하세요

믹서기 : 주인공 (객체)

. (점) : ~의

갈기 : 기능 (메서드)

("강하게") : 옵션 (파라미터)

```
1 // 코드  
2 믹서기.갈기("강하게");  
3  
4 // 결과  
5 // "위아이잉!!! (아주 빠른 속도로 재료가 산산조각 납니다)"
```

Argument 및 Parameter는 엄밀하게는 다르지만 여기선 Argument로 통일해서 설명하겠습니다.
만약 자세히 알고 싶으시다면 [이 링크](#)를 확인해주세요



괄호 안의 값(인자/Argument)

괄호 안의 값(인자/Argument): "구체적인 옵션"

만약 믹서기가 더 똑똑해서 시간까지 설정할 수 있다면 어떨까요?

콤마(,)로 구분해서 두 가지 주문을 한 번에 넣으면 됩니다.

```
1 // "강하게" + "30초 동안" 갈아줘!
2 믹서기.갈기("강하게", 30);
3
4 // 이 부분은 function 을 배울 때 알게 됩니다.
```



반환값(Return Value)

반환값(Return Value): "요리의 결과물"

도구를 쓰고 나면 반드시 결과물이 남습니다.

사과를 깎으면 '껍질 깎인 사과'가 남고, 종이를 자르면 '종이 조각'이 남겠지요?

우리는 이 결과물을 다시 변수에 담아서(let 접시 = ...) 다음 요리에 씁니다.

```
1 // 1. 껍질이 붙어 있는 사과를 준비합니다.  
2 const 사과 = "껍질 붙은 사과";  
3  
4 // 2. 사과에서 "껍질 붙은 " 부분을 찾아서 없앱니다("").  
5 // [중요] 도구가 일을 마친 뒤 남은 결과물("사과")이 '접시'에 담깁니다.  
6 let 접시 = 사과.replace("껍질 붙은 ", "");  
7  
8 // 3. 접시를 확인해보면?  
9 console.log(접시);  
10 // 결과: "사과"
```



[실습 (미니퀘스트)] 01. 01.5

아래 노션 링크로 접속해서 미니퀘스트를 확인해 주세요

<https://kdt-gitlab.elice.io/-/snippets/10>



텍스트 조작하기 (String)

문자열(String)

자동화 테스트를 작성할 때 가장 많이 다루는 데이터 타입은 단연코 문자열(String)입니다.

모든 과정이 문자열 조작과 관련이 있습니다.

화면에 보이는 텍스트를 가져와서 비교하는 경우라든지..

URL에서 특정 ID를 추출한다든지..

가격 데이터에서 화폐 단위를 제거한다든지..

문자열에서 이메일만 쑥 뽑아낸다든지 하는 등!

실무에서 "반드시" 쓰이는 핵심 메서드 6가지를 QA 업무 시나리오와 함께 정리했습니다.



텍스트 조작하기 (String)

1. length (길이 확인)

입력 폼 검증 테스트할 때 이 필드에 몇 글자가 들어왔는지 확인하는 것처럼 문자열의 글자 수를 세는 도구입니다.

```
1 const str = "Hello QA";  
2 console.log(str.length); // 8
```



텍스트 조작하기 (String)

1. length (길이 확인)

실무 활용 예시

회원가입 테스트 시 비밀번호가 최소 8자 이상인지 검증한다든지..

입력 필드의 최대 글자 수 제한(Max Length)을 확인할 때 사용한다든지..

```
1 // 테스트 데이터
2 const inputPassword = "pass";
3 const bioText = "안녕하세요. 자기소개입니다. (1000자 제한 테스트 중...)";
4
5 // 1. 비밀번호 길이 검증 로직
6 if (inputPassword.length < 8) {
7     console.log("Fail: 비밀번호는 8자리 이상이어야 합니다.");
8 }
9
10 // 2. 입력 필드 제한 확인 (Assertion)
11 // DB나 UI에서 가져온 텍스트가 제한을 넘지 않았는지 확인
12 const MAX_LENGTH = 1000;
13 if (bioText.length > MAX_LENGTH) {
14     throw new Error(`Bug Found: 글자 수 제한을 초과했습니다. (현재: ${bioText.length})`);
15 }
```



2. includes (포함 여부 확인)

문자열 안에 특정 텍스트가 포함되어 있는지 확인하여 true / false를 반환합니다.

ES6에서 추가된 기능입니다! 기존 indexOf보다 직관적입니다.

QA시 이런 작업을 많이 합니다.

“로그에 ERROR 메시지가 있는가?” / “응답 값에 success가 포함되어 있는가?

includes는 이런 식으로 문자열을 찾아주는 전용 둘보기라고 생각하시면 되겠습니다.

```
1 const text = "Error: 404 Not Found";
2 console.log(text.includes("Error")); // true
```



2. includes (포함 여부 확인)

실무 활용 예시

API 응답 메시지나 화면의 에러 문구에 특정 키워드가 포함되어 있는지 '빠르게' 확인할 때 가장 많이 씁니다.

```
1 const actualErrorMessage = "로그인 정보가 올바르지 않습니다. 다시 시도해주세요。";
2 const expectedKeyword = "올바르지 않습니다";
3
4 // Assertion (검증)
5 if (actualErrorMessage.includes(expectedKeyword)) {
6     console.log("Pass: 에러 메시지가 올바르게 출력되었습니다。");
7 } else {
8     console.error("Fail: 예상한 에러 문구가 포함되지 않았습니다。");
9 }
```



3. split (문자열 쪼개기)

문자열을 특정 기준(Separator)을 중심으로 잘라 여러 개의 조각(배열)으로 만드는 “데이터 분리 도구”입니다.

```
1 const data = "user,admin,guest";
2 const roles = data.split(",");
3 console.log(roles); // ["user", "admin", "guest"]
```



3. split (문자열 쪼개기)

실무 활용 예시

URL에서 특정 리소스 ID를 추출하거나..

날짜 데이터(YYYY-MM-DD)를 분리하여 검증할 때와 같이 문자열을 쪼개야 하는 상황에서 필수입니다.

```
1 // 시나리오: 회원 가입 후 리다이렉트 된 URL에서 회원 ID(12345)를 추출해야 함
2 const currentUrl = "https://service.com/users/12345/profile";
3
4 // '/'를 기준으로 자릅니다.
5 const urlParts = currentUrl.split("/");
6 // 결과: ["https:", "", "service.com", "users", "12345", "profile"]
7
8 const userId = urlParts[4]; // 인덱스로 접근
9
10 console.log(`추출된 User ID: ${userId}`); // "12345"
11
12 // 이후 이 ID를 사용하여 DB를 조회하거나 다음 API 테스트에 활용
```



4. replace (문자열 교체)

특정 문자열을 다른 문자열로 바꿉니다.

기본적으로 첫 번째 발견된 것만 바꿉니다.

전체를 바꾸려면 replaceAll 혹은 정규식을 사용해야 합니다! 잊지 마세요!

```
1 const text = "I LOVE Python";
2 console.log(text.replace("Python", "JavaScript")); // "I like JavaScript"
```



4. replace (문자열 교체)

실무 활용 예시

화폐 단위(\$, ₩, ,)를 제거하고 숫자로 변환하여 계산 검증을 할 때라든지

테스트 데이터를 정제(Sanitizing) 할 때 등등

특정 문자열을 교체해야 하는 상황에서 무조건 쓰입니다.

```
1 // 화면에 표시된 가격 (문자열)
2 const priceText = "1,500원";
3
4 // '원' 글자와 ',', '(콤마)'를 제거해야 수치 비교가 가능함
5 // 체이닝(Chaining)을 사용하여 연속으로 처리
6 const rawPrice = priceText.replace("원", "").replace(",","");
7
8 const priceNumber = Number(rawPrice); // 문자열을 숫자로 변환
9
10 // 가격이 1000보다 큰지 확인
11 if (priceNumber > 1000) {
12     console.log(`Pass: 가격(${priceNumber})이 정상 범위입니다.`);
13 }
```



텍스트 조작하기 (String)

5. indexOf (위치 찾기)

특정 텍스트가 몇 번째 인덱스에 있는지 숫자로 반환합니다. (없으면 -1을 반환).

전화번호, 우편번호 같은 문자열이 유효한지 검증하는 데에도 쓸 수도 있고

로그 파일에 "ERROR" 문구가 존재하는지 확인하는데 쓸 수도 있고

QA 시 CSV 구조를 검사할 때 사용할 때 쓸 수 있고

아주 다재다능, 다양하게 쓰입니다.

```
1 const email = "test@test.com";
2 console.log(email.indexOf("@")); // 4
```



```
1 const header = "id,name,age,email";
2 console.log(header.indexOf("age"));
3 // 7 → "a"가 시작된 위치
```



5. indexOf (위치 찾기)

실무 활용 예시

단순 포함 여부는 includes를 쓰지만, 특정 문자의 위치를 기준으로 앞뒤 글자를 잘라내야 할 때 사용합니다.

```
1 // 시나리오: 로그 파일의 한 줄에서 "Error:" 뒤에 나오는 내용을 가져오고 싶음
2 const logLine = "[2025-11-20] System Error: Connection Timed Out";
3
4 const target = "Error:";
5 const index = logLine.indexOf(target);
6
7 if (index !== -1) {
8     // "Error:" 가 시작되는 위치 + 글자 길이만큼 더해서 그 뒤부터 끝까지 자름(substring)
9     const errorMessage = logLine.substring(index + target.length).trim();
10    console.log(`파싱된 에러 내용: ${errorMessage}`); // "Connection Timed Out"
11 }
```



6. join (배열을 문자열로 합치기)

split의 반대 개념입니다. 배열의 요소들을 특정 구분자로 연결해 하나의 문자열로 만듭니다.

즉, “여러 개의 데이터 조각을 하나의 문자열로 합쳐서 출력, 전송, 로그 남기기 좋게 만드는 도구”입니다!

```
1 const arr = ["010", "1234", "5678"];
2 console.log(arr.join("-")); // "010-1234-5678"
```



6. join (배열을 문자열로 합치기)

실무 활용 예시

```
1 // 시나리오: 테스트 리포트 파일명을 날짜와 환경 정보를 조합해 자동 생성
2 const testDate = "20251120";
3 const env = "Staging";
4 const testCaseId = "TC_001";
5
6 const fileNameParts = [testDate, env, testCaseId, "Result"];
7
8 // underbar(_)로 연결하여 파일명 생성
9 const reportFileName = fileNameParts.join("_") + ".csv";
10
11 console.log(reportFileName);
12 // 출력: "20251120_Staging_TC_001_Result.csv"
```



[실습 (미니퀘스트)] 02

아래 노션 링크로 접속해서 미니퀘스트를 확인해 주세요

<https://www.notion.so/elice-track/JS-Method-2b42bb984257808ab04df74a6a3e482a>



배열 다루기 (Array 기본)

배열(Array)

리스트의 맨 앞, 맨 뒤, 혹은 중간에 데이터를 넣고 빼는 방법을 다룰 겁니다.

이번 파트에서는 배울 도구는 아래와 같습니다!

맨 앞에서 넣고 빼기 >> `shift()`, `unshift()`

맨 뒤에서 넣고 빼기 >> `push()`, `pop()`

중간 수정하기 >> `splice()`

하나로 합치기 >> `join()`



배열 다루기 – 맨 뒤에서 넣고 빼기 (push, pop)

push(), pop()

가장 직관적이고 자주 쓰이는 쌍입니다.

착착 쌓아나가는 스택(Stack) 구조를 생각하면 됩니다!

특징

처음에 쌓은 건 가장 나중에 꺼낼 수 있습니다.(선입후출)

First-in Last-out (FILO)라고도 부릅니다.

push(), pop() 두 가지가 있으며 하나씩 살펴보겠습니다.





배열 다루기 – 맨 뒤에서 넣고 빼기 (push, pop)

맨 뒤에 추가하기 : push()

배열의 마지막에 새로운 항목을 하나 이상 추가합니다.

그리고 나서 변경된 배열의 길이를 반환합니다.

```
1 const testResults = []; // 빈 배열 생성
2
3 // 테스트 1 수행 후
4 testResults.push("Test Case 1: Pass");
5 console.log("➡ After Test 1:", testResults);
6
7 // 테스트 2 수행 후
8 testResults.push("Test Case 2: Fail");
9 console.log("➡ After Test 2:", testResults);
10
11 // 테스트 3 수행 후 (추가)
12 testResults.push("Test Case 3: Pass");
13 console.log("➡ After Test 3:", testResults);
14
15 console.log("\n==== 최종 테스트 결과 ====");
16 console.log(testResults);
17 // ["Test Case 1: Pass", "Test Case 2: Fail", "Test Case 3: Pass"]
```



배열 다루기 – 맨 뒤에서 넣고 빼기 (push, pop)

맨 뒤에서 제거 : pop()

배열의 마지막 항목을 제거하고 그 제거된 항목을 반환합니다.

그리고 나서 변경된 배열의 길이를 반환합니다.

```
1 const testResults = [
2   'Test Case 1: Pass',
3   'Test Case 2: Fail',
4   'Test Case 3: Pass',
5 ]
6
7 // Pop 1 수행 후
8 const removed1 = testResults.pop()
9 console.log('➤ After Pop 1:', testResults)
10 // ["Test Case 1: Pass", "Test Case 2: Fail"]
11
12 // Pop 2 수행 후
13 const removed2 = testResults.pop()
14 console.log('➤ After Pop 2:', testResults)
15 // ["Test Case 1: Pass"]
16
17 // Pop 3 수행 후
18 const removed3 = testResults.pop()
19 console.log('➤ After Pop 3:', testResults)// []
20
21 console.log('\n==== 최종 테스트 결과 ===')
22 console.log(testResults)
23 // []
24
```



배열 다루기 – 맨 뒤에서 넣고 빼기 (push, pop)

push(), pop() 을 둘 다 사용하면 아래와 같이 리스트가 변하는 걸 볼 수 있습니다.

```
1 let dirtyPlates = ["접시1", "접시2"];
2
3 // push: 맨 위에 접시3을 추가함
4 dirtyPlates.push("접시3");
5 console.log(dirtyPlates); // ["접시1", "접시2", "접시3"]
6
7 // pop: 맨 위에 있는 접시3을 꺼내서 닦음
8 let washing = dirtyPlates.pop();
9 console.log(washing); // "접시3"
10 console.log(dirtyPlates); // ["접시1", "접시2"] (다시 2개만 남음)
```



배열 다루기 – 맨 앞에서 넣고 빼기 (shift, unshift)

```
1 // 1. 현재 김밥 창고 상태 (기본 재고)
2 let kimbapStock = ["참치김밥", "치즈김밥"];
3
4 console.log("--- 📦 현재 김밥 창고 상태 ---");
5 console.log(`재고 개수: ${kimbapStock.length}개`);
6 console.log(`재고 목록: [${kimbapStock.join(", ")}]`);
7 console.log("-----\n");
8
9
10 // 2. 김밥 추가 입고 (push 사용)
11 // 설명: 'push'는 배열의 **맨 뒤에** 새로운 항목을 추가한다.
12 // 입고 → 뒤에 차곡차곡 쌓임
13 console.log("🚚 새 김밥이 입고되었습니다! (push)");
14
15 kimbapStock.push("불고기김밥");
16 console.log(`>> 변경된 재고: [${kimbapStock.join(", ")}]`);
17 // ["참치김밥", "치즈김밥", "불고기김밥"]
18
19 kimbapStock.push("삼겹살김밥");
```



배열 다루기 – 맨 앞에서 넣고 빼기 (shift, unshift)

맨 앞에 추가하기 : unshift()

배열의 가장 앞(0번 인덱스)에 항목을 추가합니다.

나머지 항목들은 뒤로 밀려납니다.

그리고 나서 변경된 배열의 길이를 반환합니다.

아래와 같이 사용할 수도 있습니다.

테스트 리포트 데이터의 맨 윗줄에 '헤더(Header)'를 추가하거나

긴급하게 처리해야 할 작업을 대기열(Queue) 최우선 순위로 넣거나

```
1 const csvData = [
2   "user1, pass",
3   "user2, fail"
4 ];
5
6 // CSV 파일 생성을 위해 맨 윗줄에 컬럼명 추가
7 csvData.unshift("UserID, Result");
8
9 console.log(csvData);
10 // ["UserID, Result", "user1, pass", "user2, fail"]
```



배열 다루기 – 맨 앞에서 넣고 빼기 (shift, unshift)

맨 앞에 제거하기 : shift()

배열의 가장 앞(0번 인덱스) 항목을 제거하고 반환합니다.

아래와 같이 사용할 수도 있습니다.

'선입선출(FIFO)' 구조의 테스트 큐(Queue)를 처리할 때 사용할 수 있습니다.

목록에 있는 테스트 케이스를 순서대로 하나씩 꺼내서 실행하고 목록에서 없앱니다.

```
1 const testQueue = ["LoginTest", "PaymentTest", "LogoutTest"];  
2  
3 while(testQueue.length > 0) {  
4     const currentTest = testQueue.shift(); // 맨 앞 테스트 꺼내기  
5     console.log(`지금 실행 중: ${currentTest}`);  
6     // ... 테스트 실행 로직 ...  
7 }  
8  
9 // 루프 종료 후  
10 console.log(testQueue); // [] (모두 실행되어 비어있음)
```



배열 다루기 – 맨 앞에서 넣고 빼기 (shift, unshift)

전체 코드 -> <https://gist.github.com/ej31/9a7acd65efc6eef5729f24f8457b6f68>

대기 손님 관리 시스템으로 이해해 보기 (Queue)

```
1 // 1. 현재 대기 중인 손님 명단 (기본 대기열)
2 let waitingLine = ["철수", "영희", "민수"];
3
4 console.log("--- 🔈 현재 대기 현황 ---");
5 console.log(`현재 대기 인원: ${waitingLine.length}명`);
6 console.log(`대기 명단: [${waitingLine.join(", ")}]`);
7 console.log("-----\n");
8
9
10 // 2. 상황 발생: 슈퍼 VIP 손님 등장 (unshift 사용)
11 // 설명: 'unshift'는 배열의 맨 앞에 요소를 추가합니다. 기존 손님들은
12 //         뒤로 밀려납니다.
13 console.log("✨ 귀한 VIP 손님이 도착했습니다! (맨 앞으로 모십니다)!");
14
15 waitingLine.unshift("👑VIP장관님");
16
17 console.log(`>> 변경된 대기 명단: [${waitingLine.join(", ")}]`);
18 // 결과: ["👑VIP장관님", "철수", "영희", "민수"]
```



배열 다루기 – 배열 중간 수정하기 (splice)

배열의 중간에 있는 데이터를 삭제 및 교체 혹은 끼워 넣을 때 사용합니다.

1 `array.splice(시작인덱스, 삭제할개수, 추가할아이템...)`

특정 테스트 데이터를 삭제(Clean up)하거나 테스트 시나리오 중간에 단계를 삽입해야 할 때 등 다양하게 쓸 수 있습니다.

```
1 const shoppingCart = ["Apple", "Banana", "ToxicItem", "Orange"];
2
3 // 1. 삭제하기: 인덱스 2번('ToxicItem')부터 1개를 삭제
4 shoppingCart.splice(2, 1);
5 console.log(shoppingCart); // ["Apple", "Banana", "Orange"]
6
7 // 2. 끼워넣기: 'Banana' 뒤(인덱스 2번 자리)에 'Melon' 추가 (삭제 개수 0)
8 shoppingCart.splice(2, 0, "Melon");
9 console.log(shoppingCart); // ["Apple", "Banana", "Melon", "Orange"]
```



배열 다루기 – 배열 중간 수정하기 (splice)

전체 코드 -> <https://gist.github.com/ej31/4d9907d166b79e0a06a9fee4cdfc4351>

```
1 // 0. 초기 김밥 재료 준비
2 let kimbap = ["밥", "햄", "시금치", "단무지"];
3
4 console.log("--- 초기 김밥 상태 ---");
5 console.log("재료:", kimbap);
6 // 인덱스 안내: 0="밥", 1="햄", 2="시금치", 3="단무지"
7 console.log("-----\n");
8
9
10 // 시나리오 1: 재료 교체하기 (삭제 후 추가)
11 // 상황: "시금치(2번)"를 빼버리고, 그 자리에 "오이"를 넣고 싶음.
12 console.log("Mission 1: 시금치를 오이로 교체하라!");
13
14 // 해석: 2번 인덱스에서 시작, 1개 삭제 후 "오이" 추가
15 kimbap.splice(2, 1, "오이");
16
17 console.log("결과:", kimbap);
18 // ["밥", "햄", "오이", "단무지"]
19 console.log("-----\n");
```



배열을 하나의 문자열로 만들기 (join)

join() : 배열을 "한 줄짜리 문자열"로 만드는 함수

join()은 배열 안의 여러 조각들을 하나로 이어 붙입니다.

괄호 안에는 각 조각 사이에 넣고 싶은 구분자(separator)를 적습니다.

구분자가 없으면 쉼표(,) 가 기본값입니다.

```
1 let parts = ["떡", "소세지", "떡", "소세지"];
2 parts.join("-");
3 // "떡-소세지-떡-소세지"
```

배열을 하나의 문자열로 만들기 (join)

1. 기본 상황: 맛있는 꼬치구이 만들기 (구분자 사용)

```
1 let ingredients = ["떡", "파", "오뎅", "소세지"];
2
3 console.log("--- 🍗 꼬치구이 만들기 ---");
4 // join("-") -> 재료들 사이에 하이픈(-)이라는 꼬치를 끼웁니다.
5 let skewer = ingredients.join("-");
6
7 console.log(`재료들: [${ingredients}]`);
8 console.log(`완성품: "${skewer}"`);
9 // 결과: "떡-파-오뎅-소세지" (하나의 문장이 되었습니다!)
10 console.log("-----\n");
```



배열을 하나의 문자열로 만들기 (join)

2. 응용 상황: 자연스러운 문장 만들기 (공백 사용)

```
1 // 단어들이 흩어져 있으면 말이 안 되지만, '띄어쓰기'로 연결하면 문장이 됩니다.  
2 let words = ["안녕하세요", "반갑습니다", "저는", "개발자입니다"];  
3  
4 console.log(`--- 🧑 자기소개 하기 ---`);  
5 // join(" ") -> 단어 사이에 '공백(스페이스)'을 접착제로 사용합니다.  
6 let sentence = words.join(" ");  
7  
8 console.log(`완성된 문장: ${sentence}`);  
9 // 결과: "안녕하세요 반갑습니다 저는 개발자입니다"  
10 console.log(`-----\n`);
```

배열을 하나의 문자열로 만들기 (join)

3. 주소 값을 입력받을 때 여러 입력칸으로 받은 것을 하나로 합치는 경우

단순히 붙이는 것뿐만 아니고 '무엇으로' 붙이느냐에 따라 결과물이 완전히 달라집니다.

```
1 // 여러 칸으로 쪼개진 주소 입력값
2 let addressParts = ["서울시", "강남구", "테헤란로 123", "501호"];
3
4 console.log("--- ↴ 주소 합치기 ---");
5
6 // join(" ") -> 띄어쓰기 하나를 사이에 넣어서 연결
7 let fullAddress = addressParts.join(" ");
8
9 console.log(`최종 주소: "${fullAddress}"`);
10 // 결과: "서울시 강남구 테헤란로 123 501호"
```



배열 순회 문법 (Array Loop, ES 6+)

배열(Array) 복습

배열은 '칸이 나누어진 긴 정리함'이라고 생각하면 됩니다.

우리는 이 정리함에 데이터(숫자, 글자 등)를 순서대로 담아둡니다.

```
1 // 🍎 과일 정리함(배열) 생성  
2 const fruits = ["사과", "바나나", "포도"];
```

그리고 앞서 배운 내용은 "배열 자체를 만지는 도구"였습니다.

지금부터는 "배열을 순회하면서 뭔가를 만들어내는 도구"를 살펴보겠습니다.

ES6+라는 버전에서 제공하는 최신 도구들을 사용해 요리조리 다뤄보겠습니다!



배열 순회 문법 (Array Loop, ES 6+)

1. forEach: "하나씩 출석 부르기"

정리함에 있는 물건을 처음부터 끝까지
하나씩 꺼내서 확인하는 도구입니다.

선생님이 학생들의 이름을 한 명씩 부르는 것과
같습니다.

쉽게 말해서 리스트에 있는 원소를 하나씩
순회하면서 접근하는 겁니다!

```
1 const students = ["철수", "영희", "민수"];
2
3 // 옛날 방식.. 조금 복잡하죠?.. 지금은 몰라도 되용
4 // for (let i = 0; i < students.length; i++) { ... }
5
6 // ✨ ES6+ 방식 (forEach)
7 students.forEach((student) => {
8   console.log(student + " 왔니?");
9 });
10
11 // 결과:
12 // "철수 왔니?"
13 // "영희 왔니?"
14 // "민수 왔니?"
15 // 옛날 방식과 ES6+ 방식 둘 다 결과는 같습니다!
16 // 하지만 코드 가독성이 forEach 가 더 깔끔하죠!
```



2. map: "모두 변신시키기"

정리함의 모든 물건을 규칙에 따라 변신시켜서 새로운 정리함에 담습니다.

빵 공장에서 밀가루 반죽(입력)이 기계를 통과하면 쿠키(출력)가 되어 나오는 것과 같습니다.

```
1 const prices = [1000, 2000, 3000];
2
3 // 모든 가격을 2배로 올려볼까요?
4 const expensivePrices = prices.map((price) => {
5     return price * 2;
6 });
7
8 console.log(expensivePrices);
9 // 결과: [2000, 4000, 6000] -> 원본은 그대로 있고, 새 배열이 생겼어요!
```



3. filter: "조건에 맞는 것만 골라내기"

조건에 맞는(True) 데이터만 남기고 나머지는 버려서 새로운 정리함을 만듭니다.

거름망(필터)으로 모래를 거르고 금만 남기는 것 혹은 클럽 입구에서 성인만 입장시키는 것과 같습니다

```
1 const ages = [15, 22, 13, 29, 19, 35];
2
3 // 성인(20세 이상)만 입장 가능!
4 const adults = ages.filter((age) => {
5     return age >= 20;
6 });
7
8 console.log(adults);
9 // 결과: [22, 29, 35] -> 미성년자는 걸러졌습니다.
```



4. find: "딱 하나만 찾아줘"

조건에 맞는 데이터를 찾는데, 가장 먼저 발견된 하나만 찾아주고 끝납니다.

"월리를 찾아라"에서 월리를 찾으면 게임이 끝나는 것과 같아요!

```
1 const users = [
2   { id: 1, name: "짱구" },
3   { id: 2, name: "맹구" }, // 찾았다!
4   { id: 3, name: "맹구" }, // 얘는 무시됨 (이미 찾았으니까)
5 ];
6
7 // 이름이 '맹구'인 사람을 찾아줘
8 const foundUser = users.find((user) => {
9   return user.name === "맹구";
10 });
11
12 console.log(foundUser);
13 // 결과: { id: 2, name: "맹구" }
```



5. reduce: "하나로 뭉치기" (어려움 주의!)

배열 안의 모든 데이터를 합쳐서 최종적으로 딱 하나의 값으로 만듭니다.

눈덩이를 굴리는 것과 같습니다. 처음엔 작았지만 굴러가면서 눈(데이터)이 계속 불어 커다란 눈사람(결과)이 됩니다.

눈덩이가 이해가 안 되신다면 쇼핑카트 계산하는 것과 같다고 생각하시면 됩니다.

```
1 // 장바구니에 담긴 물건 가격들
2 const cartPrices = [12900, 5900, 32000, 4500];
3
4 // 총 금액 계산하기
5 // acc: 지금까지 더해진 금액 (지갑에서 이미 꺼낸 돈)
6 // cur: 이번에 계산대에 올려진 물건 가격
7 const totalPrice = cartPrices.reduce((acc, cur) => {
8     return acc + cur;
9 }, 0); // 초기값: 0원 (지갑에서 아무 돈도 꺼내지 않은 상태)
10
11 console.log(`총 결제 금액: ${totalPrice}원`);
12 // 결과: 55300원 (12900 + 5900 + 32000 + 4500)
```



JavaScript Object Notation (JSON)

JSON, 프로그래밍 세상의 공용어이자 택배 상자

이번에는 JSON에 대해 아주 쉽게 알아보겠습니다.

웹사이트나 앱을 만들 때, 서버(창고)와 내 컴퓨터(집) 사이에서 데이터를 주고받으려면 반드시 이 JSON을 알아야 합니다.

JSON이 뭔가요?

데이터를 저장하거나 전송할 때 사용하는 텍스트 형식입니다.

자바스크립트 객체(Object)

내 머릿속에 있는 생각이나, 방 안에 펼쳐진 실제 물건들입니다. (바로 쓸 수 있음)

JSON

물건을 다른 사람에게 보내기 위해 택배 상자에 차곡차곡 넣고 테이프로 포장한 상태입니다. (이동하기 좋음)



JavaScript Object Notation (JSON)

1. JSON의 생김새

JSON은 자바스크립트 객체와 아주 비슷하게 생겼지만, 몇 가지 엄격한 규칙이 있습니다.

가장 큰 특징: 이름표(Key)와 내용(Value) 모두 반드시 큰따옴표("")로 감싸야 합니다.

```
1  {
2      "name": "짱구",
3      "age": 5,
4      "isCute": true,
5      "friends": ["철수", "맹구", "훈이"]
6  }
```

주의: 작은따옴표(")는 안 됩니다! 반드시 큰따옴표("")를 써야 합니다. (큰따옴표가 표준입니다!)



2. JSON.stringify(): "포장하기"

자바스크립트 객체(Object)를 JSON 문자열(String)로 바꾸는 과정입니다.

데이터를 서버로 보낼 때 주로 사용합니다.

친구에게 물건을 보내기 위해 택배 상자에 넣고 포장하는 것과 같습니다.

```
1 // 내 컴퓨터에 있는 실제 데이터 (객체)
2 const myInfo = {
3   name: "홍길동",
4   age: 30,
5   hobby: "코딩"
6 };
7
8 // 📦 포장하기 (객체 -> 문자열)
9 const jsonString = JSON.stringify(myInfo);
10
11 console.log(jsonString);
12 // 결과: '{"name": "홍길동", "age": 30, "hobby": "코딩"}'
13 // (이제 이 문자열은 인터넷 선을 타고 날아갈 수 있습니다!)
```



3. JSON.parse(): "언박싱(개봉)하기"

서버에서 받은 JSON 문자열(String)을 다시 자바스크립트 객체(Object)로 바꾸는 과정입니다.

택배가 도착했으니 상자를 뜯고 물건을 꺼내는 것(언박싱)과 같습니다.

언박싱을 했으니 이제 코드에서 데이터를 자유롭게 쓸 수 있습니다.

```
1 // 서버에서 날아온 데이터 (문자열 상태)
2 const receivedData = '{"name": "홍길동", "age": 30, "hobby": "코딩"}';
3
4 // ✕ 언박싱하기 (문자열 -> 객체)
5 const myObject = JSON.parse(receivedData);
6
7 console.log(myObject.name); // "홍길동" (이제 점(.)을 찍어서 꺼낼 수 있어요!)
8 console.log(myObject.age); // 30
```



4. 실수하기 쉬운 포인트

따옴표 실수

JSON 파일 안에서는 무조건 **큰따옴표("")**만 씁니다.

마지막 쉼표

배열이나 객체의 마지막 항목 뒤에는 쉼표(,)를 찍으면 안 됩니다. (엄격해요!)

함수는 안 됩니다

JSON은 데이터만 담을 수 있어서, 함수(function)는 포장할 때 사라집니다.

```
1 const data = {  
2     name: "테스트",  
3     sayHello: function() { console.log("안녕!"); } // 함수  
4 };  
5  
6 const json = JSON.stringify(data);  
7 console.log(json);  
8 // 결과: '{"name": "테스트"}'  
9 // 함수는 사라져 버립니다!
```



JavaScript Object Notation (JSON)

"보낼 땐 묶고(Stringify), 받을 땐 푼다(Parse)"

이것만은 꼭 기억해 주세요!



화살표 함수(Arrow Function)

자바스크립트 코드를 획기적으로 줄여주는 화살표 함수

화살표 함수란?

함수를 만들 때 항상 `function`이라는 긴 단어를 사용하고 있습니다.

`function`이라는 단어 대신 화살표(`=>`) 기호를 사용해 함수를 만드는 새로운 방법입니다.

함수 다이어트 3단계

함수가 어떻게 살을 빼는지 총 세 개의 단계로 하나씩 살펴보겠습니다.



코드 줄이기 (Arrow Function)

[1단계] 기본 변신

function 글자를 지우고 매개변수 뒤에 화살표 =>를 붙입니다.

```
1 // 뚱뚱한 옛날 함수  
2 const add = function(x, y) {  
3     return x + y;  
4 };  
5  
6 // 날씬해진 화살표 함수  
7 const add = (x, y) => {  
8     return x + y;  
9 };
```

[2단계] 중괄호와 return 생략

만약 함수 안의 내용이 "딱 한 줄"이고 그게 반환(return)하는 값이라면?

중괄호 {}와 return 글자마저 생략할 수 있습니다.

```
1 // 더 줄일 수 있다!
2 const add = (x, y) => x + y;
3
4 // 해석: x와 y를 받아서 더한 값을 바로 내놓아라. (직관적이죠?)
```

[3단계] 소괄호 생략

만약 매개변수(재료)가 "딱 하나"라면?

감싸고 있는 소괄호 ()도 벗어버릴 수 있습니다.

```
1 // 괄호조차 귀찮다 !
2 const double = x => x * 2;
3
4 // 해석: x가 들어오면 2배로 뺑튀기 해라 .
```



자바스크립트의 심장, 객체(Object)

자바스크립트의 심장이라고 불리는 객체(Object)에 대해 알아보겠습니다.

앞서 배운 배열(Array)과 형제지만, 성격이 조금 다릅니다.

배열이 번호(순서)로 관리한다면, 객체는 이름(Key)으로 관리합니다.

객체(Object)란?

객체는 '이름표가 붙은 서랍장'입니다. 관련된 정보를 하나의 덩어리로 묶어서 관리할 때 씁니다.

배열: 출석부 (1번, 2번, 3번... 순서가 중요함)

객체: 학생기록부 (이름, 나이, 주소... 항목의 이름이 중요함)

```
1 // '철수'의 학생기록부 (객체)
2 const student = {
3   name: "철수",    // 이름표(Key): 내용(Value)
4   age: 10,
5   isHungry: true
6 };
```



1. 꺼내 보기 (조회)

서랍장에서 물건을 꺼내는 방법은 두 가지가 있습니다.

1-1. 점 찍기 (Dot Notation) ★

가장 많이 쓰는 방법입니다. "누구의.무엇" 처럼 자연스럽게 읽힙니다.

```
1 // '철수'의 학생기록부 (객체)
2 const student = {
3   name: "철수",    // 이름표(Key): 내용(Value)
4   age: 10,
5   isHungry: true
6 };
7 console.log(student.name); // "철수"
8 console.log(student.age); // 10
```



JavaScript Object (객체)

1. 꺼내 보기 (조회)

1-2. 대괄호 쓰기 (Bracket Notation)

이름표를 문자열(" ")로 적어서 찾습니다. 이름표가 변수로 들어오거나, 특수문자가 섞여 있을 때 씁니다.

```
1 // '철수'의 학생기록부 (객체)
2 const student = {
3   name: "철수", // 이름표(Key) : 내용(Value)
4   age: 10,
5   isHungry: true
6 };
7
8 console.log(student["name"]); // "철수"
9
10 // 꿀팁: 변수로 찾을 때 유용해요!
11 const key = "age";
12 console.log(student[key]); // 10 (student.age와 같음)
```



2. 쓰고 고치기 (수정 및 추가)

빈 서랍에 물건을 새로 넣거나 기존 물건을 바꿀 수 있는 것처럼 JS 객체도 똑같이 가능합니다.

```
1 const dog = {  
2   name: "바둑이"  
3 };  
4  
5 // 1. 내용 수정하기  
6 dog.name = "초코";  
7  
8 // 2. 새로운 내용 추가하기 (없으면 새로 생깁니다!)  
9 dog.age = 3;  
10 dog.hobby = "산책";  
11  
12 console.log(dog);  
13 // 결과: { name: "초코", age: 3, hobby: "산책" }
```



3. 객체의 행동 (Method)

객체 안에는 숫자나 글자뿐만 아니라 함수(행동)도 넣을 수 있습니다.

객체 안에 들어있는 함수를 메서드(Method)라고 부릅니다.

```
1 const hero = {  
2     name: "용사",  
3     hp: 100,  
4     // 메서드 (행동)  
5     attack: function() {  
6         console.log("검 휘두르기! ⚔");  
7     }  
8 };  
9  
10 hero.attack(); // 결과: "검 휘두르기! ⚔"
```



4. ES6+ 단축 문법 (Shorthand, 진짜 많이 씁니다.)

최신 자바스크립트에서는 JS 객체를 더 편하게 다룰 수 있습니다.

4-1. 키와 변수명이 같을 때

넣으려는 변수 이름과 객체의 키 이름이 같다면, 한 번만 써도 됩니다.

```
1 const name = "영희";
2 const age = 20;
3
4 // [옛날 방식]
5 // const person = { name: name, age: age };
6
7 // [ES6+ 방식]
8 const person = { name, age };
9 // (알아서 name: "영희", age: 20 으로 들어감)
```



4. ES6+ 단축 문법 (Shorthand, 진짜 많이 씁니다.)

4-2. 객체 구조 분해 할당 (Destructuring)

객체 안의 값을 쑥쑥 뽑아서 변수로 만듭니다.

```
1 const book = {  
2   title: "어린왕자",  
3   price: 10000  
4 };  
5  
6 // title과 price를 꺼내서 변수로 만들기  
7 const { title, price } = book;  
8  
9 console.log(title); // "어린왕자"
```



[추가자료] Spread (...) 와 구조 분해 할당

Spread Operator (...) : "포장 뜯어 펼치기"

배열이라는 박스를 뜯어서 내용물을 차르륵 펼쳐줍니다.

ES6+의 꽃이라고 불리는 문법입니다. 정말 편리합니다! (실제로 엄청 많이 씁니다.)

```
1 const teamA = ["철수", "영희"];
2 const teamB = ["민수", "지수"];
3
4 // 두 팀을 합치고 싶다면?
5 const unitedTeam = [...teamA, ...teamB];
6
7 console.log(unitedTeam);
8 // 결과: ["철수", "영희", "민수", "지수"] -> 아주 쉽게 합쳐졌죠?
```



[추가자료] Spread (...) 와 구조 분해 할당

구조 분해 할당 (Destructuring) : "쏙쏙 뽑아 쓰기"

배열 안에 있는 값을 변수로 바로 꺼내옵니다.

ES6+의 꽃이라고 불리는 문법입니다. 정말 편리합니다! (실제로 엄청 많이 씁니다.)

```
1 const colors = ["빨강", "파랑", "초록"];
2
3 // 옛날 방식
4 // const red = colors[0];
5 // const blue = colors[1];
6
7 // ✨ ES6+ 방식
8 const [red, blue, green] = colors;
9
10 console.log(red); // "빨강"
11 console.log(blue); // "파랑"
```