1주차 교육 자료

목표

- 웹의 정의를 학습한다.
- http, https를 학습한다.
- DOM과 REST의 개념을 학습힌다.
- 자바스크립트 심화 개념과 응용 방법을 학습한다.

시작하기에 앞서

저희는 처음부터 끝 까지 다 가르치지 않을거에요.

if문 작성하는 방법, for문 작성하는 방법… 이런건 여러분들이 직접 배워야해요

우리가 직접 알려주는 것도 있지만 이걸 이해하기 위해선 '사전 지식'이 꼭 필요해요. 그래서 우리가 과제를 끊임없이 내는 이유이기도 해요!

지금은 첫 세션이고 열정이 가득하지만 시간이 지날수록 과목 하나 더 듣는 것 마냥 재미없이질 수 있어요. 그러다 결국 따라오지 못하게 되고.. 이제와서 그만두긴 싫고.. 연합 동아리 활동했다고 자소서에는 한 줄 더 쓰고싶고.. 이런식으로 허송세월 보내다 얻어가는 건 없이 끝날 수도 있어요.

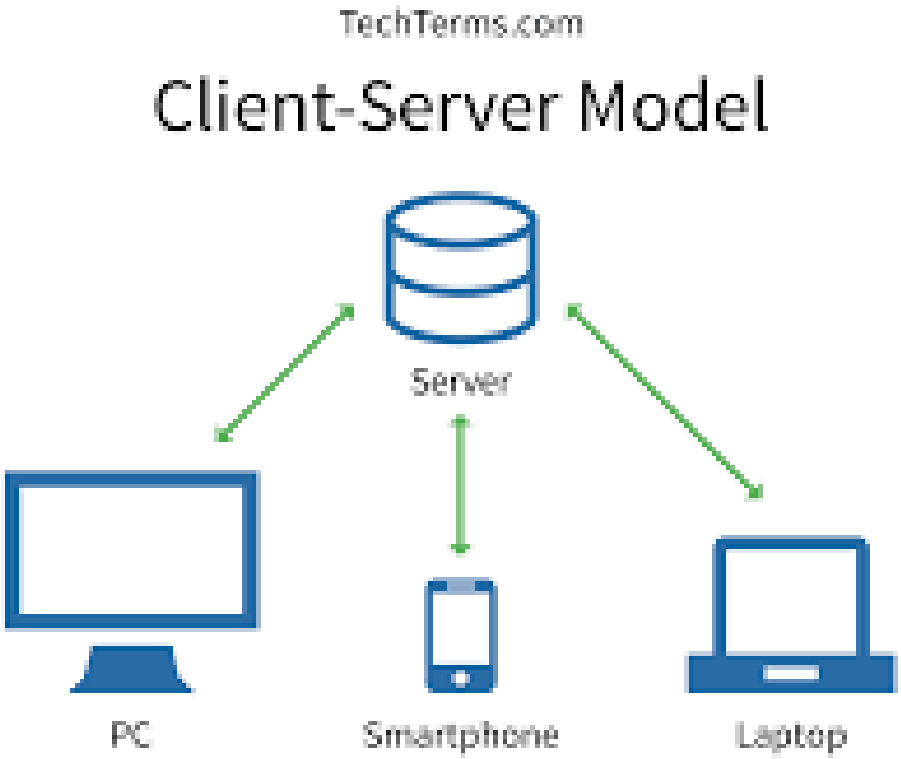
이런 대참사를 막기위해 꼭 스스로 공부하는 것과 질문을 적극적으로 했으면 합니다! 특히 질문을 너무 많이하면 귀찮아 할 것 같아.. 싶을텐데, 저희는 아는걸 자랑하고 싶어하는 사람들이라 질문 받으면 좋아라 하고 도와줄거에요!

만약에 너무 따라오기 힘들다 싶을 땐 아무 운영진에게 가서 말해주세요! 혹시몰라요 특별 세션을 열어줄지도..?

웹 이란?

World Wide Web의 줄임말로 우리가 자주보는 www가 이걸 뜻해요.

웹의 동작



웹에 연결된 컴퓨터는 두 가지 종류가 있어요

사용자의 요청을 처리하는 '서버'와 사용자의 환경인 '클라이언트'가 있어요.

그럼 우리가 네이버 페이지를 요청했을 때를 가정해볼게요

```
1. 사용자는 naver.com을 입력해요
2. 브라우저는 DNS 서버로 가서 naver.com의 진짜 주소(ip 주소)를 찾아요
3. 브라우저는 진짜 주소를 가지고 서버에 가서 웹 사이트를 만들어 달라 HTTP 요청을 보내요
4. 요청을 받는 서버는 클라이언트의 요청을 승인하고 200 OK 메시지를 클라이언트에게 전송해요
5. 그 후 서버는 웹사이트 파일을 패킷이라 불리는 작은 단위로 나눠 클라이언트에 전송해요
6. 클라이언트(브라우저)는 이 단위를 조립해 웹사이트를 만들고 우리에게 보여줘요
```

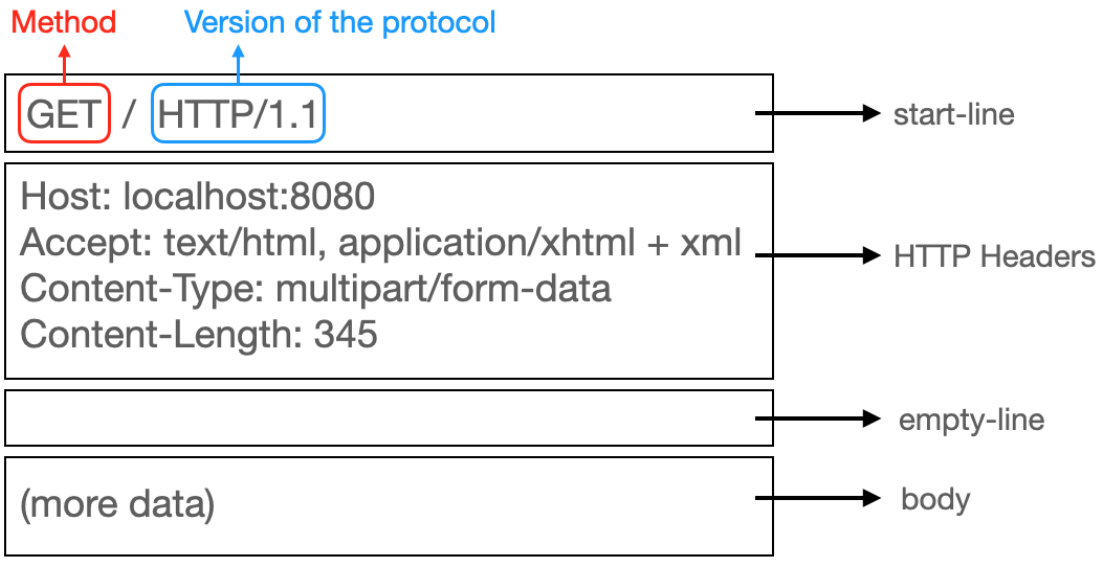
좀 더 자세한 설명은 MDN을 읽어보세요

HTTP

위에서 HTTP 라는 개념이 나왔네요 HTTP에 대해 다뤄볼게요.

HTTP란?

- Hypertext transfer Protocol의 약자로 Client-Server간 데이터를 주고받기 위한 프로토콜이에요.즉, 데이터를 주고받기 위한 일종의 약속이라 생각하면 돼요
- HTTP는 상태를 가지지 않는 Stateless 프로토콜이에요
- HTTP는 Method, Path, Version, Headers, Body로 구성되어있어요



HTTP를 어디서 쓸까?

웹에서 클라이언트가 서버에 특정 작업을 요청한다면 대부분 HTTP를 사용해요.

웹 페이지를 요청할 수도 있고 특정 데이터를 요청할 수도 있어요.

HTTP의 단점

HTTP의 가장 큰 단점은 데이터가 암호화 되지 않아요. 그래서 마음만 먹으면 http 패킷을 들여다봐서 사용자의 민감한 정보를 다 빼낼 수 있어요.

그럼 우리가 사용하는 웹 페이지는 다 http니까 위험한거 아닌가요??

HTTPS

위에서 설명한 HTTP의 단점을 해결하기 위해 나왔어요.

아까전에 우리가 사용하는 웹 페이지는 다 http인데 위험한거 아니냐고 제가 질문했는데 이에 대한 답으로는 'NO'입니다

최근 우리가 사용하는 대부분의 사이트는 https로 되어있어요. 물론 마이너한 사이트들은 http로 되어있긴 해요. 이땐 브라우저가 경고 표시를 해줍니다

지금 주소창을 보면 자물쇠 모양이 보일거에요. 이게 바로 https로 보호받고 있다는 것을 뜻해요.

HTTPS가 왜 안전한가

HTTPS는 SSL 인증서를 사용해서 HTTP 프로토콜을 암호화 하고 있어요.

서버파트 회원분들은 어차피 나중에 배워야 하니까 읽어놓으면 도움 될거에요

HTTP 메서드

`HTTP란?` 설명을 보면 Method를 언급한 적이 있어요.

서버와 클라이언트가 통신할 때 통신 방법을 명시하는 것을 말해요

대표적으로 아래의 메서드가 있어요

method		의미
GET	Read	데이터 **조회**
POST	Create	요청 데이터 처리, 주로 데이터 **등록**에 사용
PUT	Update	데이터 **수정(전체 수정)**
DELETE	Delete	데이터 **삭제**
PATCH	Update	데이터 **수정(부분 수정)**

HTTP 메서드는 `웹 서버 모두 사용해야하니 필수적으로 알고있어야` 해요

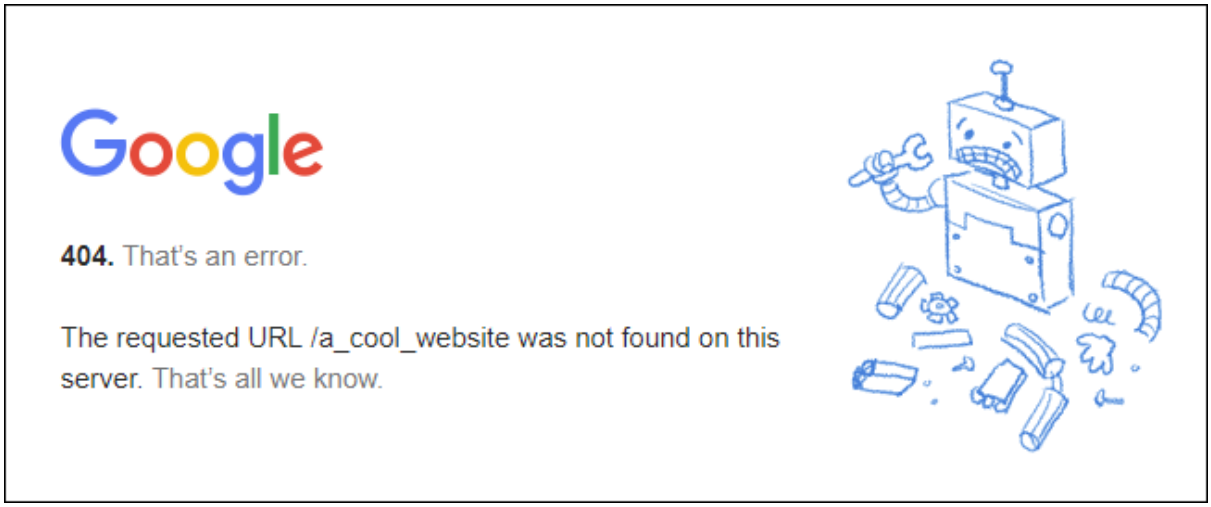
```
김멋사는 likelion.com 서버에 title='멋사'고 author='배교수'인
포스팅 목록을 조회하는 GET 요청을 보낼거에요
(요청 path는 postlist라고 가정할게요)

그럼 다음과 같이 메서드를 구성할 수 있어요

host:[port]/resoucePath?queryString
likelion.com:[8080]/postlist?title='멋사'&author='배교수' // port는 옵셔널 값이에요
```

HTTP 응답 코드

가끔 웹 서핑을 하다보면 이런 화면이 보일거에요



HTTP 요청을 보내면 다양한 응답 코드를 보내는데 목록을 정리하자면 다음과 같아요

코드	설명
200	GET 요청에 대한 성공
400	잘못된 요청
401	권한 없이 요청
403	해당 자원 접근 금지
501	요청한 동작에 대해 서버가 수행할 수 없음
503	서버가 과부하거나 내려간 경우

외에도 엄청나게 많은 응답이 있지만 그것들은 필요할 때 MDN에 찾아보면 될거에요

이걸 왜 알아야 하냐면 나중에 웹과 서버가 통신할 일이 생겼을 때 에러가 발생한다면 어느 파트에 빠따를 들어야 하는지 알 수 있어요

요약해서 설명하자면

`200번대` 라면 안심하시면 되고

`400번대` 라면 웹 파트에 빠따를 (이때, 401번과 404번 처럼 의도적인 에러도 있어요)

`500번대` 라면 서버 파트에 빠따를 들면 돼요

DOM

우리는 웹 프로그래밍을 배우는 만큼 DOM에 대해 알아야겠죠?

DOM은 Document Object Model이라는 뜻이에요.

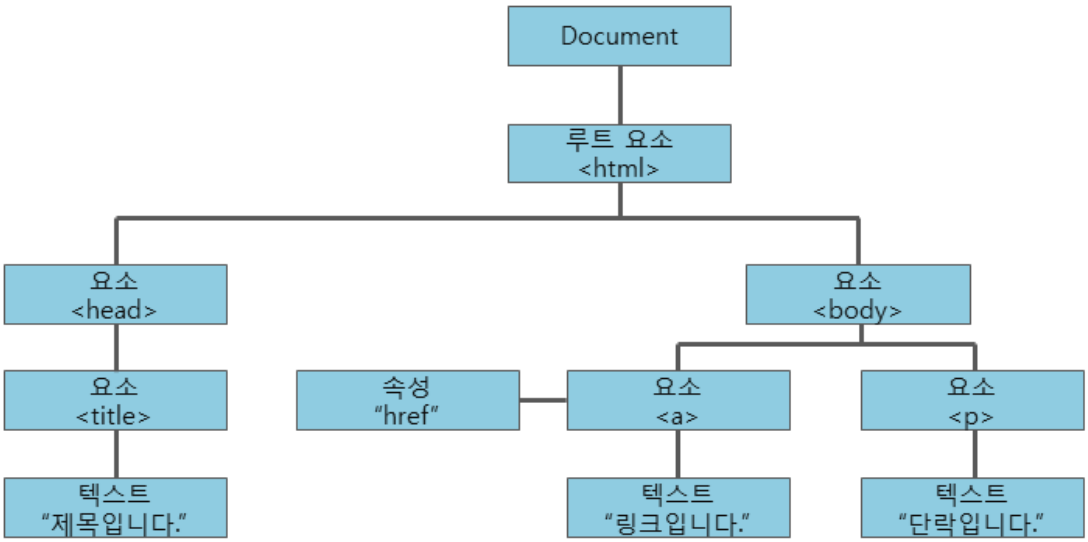
그럼 여기에 Document, 문서가 나왔네요. 왜 갑자기 문서가 나왔을까요?

원래 HTML의 기원은 어느 연구소에서 문서를 쉽게 공유하기 위해 만들어졌어요.

이제 그럼 Document가 왜 나왔는지 알겠죠?

즉, DOM은 이런 문서 객체 구조를 나타내는 것이 돼요

웹 페이지를 수정하거나 생성하는데 사용되는 모든 프로퍼티나 메서드, 이벤트들은 모두 object로 구성되는데 특정 문서 Object가 다른 문서 Object를 물고있는 Object-by-Object Reference를 제공해요



DOM을 도식화 하면 이렇게 돼요

위 내용이 이해하기 어렵다면 아직은 이 사진만 기억해주세요!

그럼 JS는 이 DOM을 조작할 수 있게 되는데 어떻게 DOM에 접근할 수 있는지는 여기서 설명하기엔 양이 매우 기니까 링크로 대체할게요!

지루한 개념 설명은 여기까지하고,

JavaScript로 DOM에 접근하는 대표적인 메서드는 다음과 같아요

```
// 괄호 안에 선택자를 넣어 일치하는 요소 중 첫 번째 Element를 반환
document.querySelector('.container') // #은 id를 선택, .은 class를 선택합니다

// 괄호 안에 id 값을 넣어 일치하는 Element를 반환
document.getElementById()

// 괄호 안에 className을 넣어 일치하는 Element's'를 반환
document.getElementsByClassName()
```

```
// 특정 태그를 생성합니다. 괄호 안에 생성하고자 하는 Element를 넣으세요
document.createElement('div');
```

특정 DOM을 선택한 뒤에 스타일을 추가하거나 text를 삽입하거나, HTML 코드를 삽입하는 등 다양한 동작이 가능해요

```
const divEl = document.createElement('div');
divEl.innerHTML = '<p>안녕하세요!</p>'

const pEl = document.createElement('p');
pEl.textContent = '저는 아기사자입니다!';

divEl.append(pEl);

<div><p>저는 ~입니다~</p> </div>
```

자세한 내용은 잠시 후 진행할 실습에서 다뤄봐요!

REST API

서버 파트 회원분들이 앞으로 질리도록 보게될 개념이에요.

우선, API는 Application Programming Interface로, 통신 관점으로 두 SW 구성 요소가 서로 통신할 수 있게 하는 메커니즘입니다.

더욱 간단하게 말하자면 특정 기능을 하는 SW를 요청하고 응답하기 위한 계약이라 말할 수 있어요

롤을 하시는 분들이라면 OP.GG를 한 번 쯤 들어가보셨을건데, OP.GG에서 유저 이름을 검색하면 ‘이 유저를 찾아줘’ 라고 클라이언트가 요청을 보내고 서버는 이 요청을 받아 ‘이 유저에 대한 정보는 여깄어’ 하고 주는데 이게 API의 한 예시에요

그럼 앞에 ‘REST’는 뭘까요?

REST는 Representational State Transfer로 API 작동 방식에 대한 조건을 부과하는 아키텍처 구조에요.

최근, 대부분의 웹 API 호출 방식은 이 REST API 방식을 써요.

REST API의 가장 큰 특징은 Stateless입니다. 또 나왔네요! Stateless. 즉, 서버가 클라이언트 데이터를 저장하지 않아요. 아래는 REST API의 특징이에요

```
- Client-Server 구조를 가져요
- Stateless입니다
- 캐시 처리가 가능해요
- 계층화를 이뤄요
- 인터페이스의 일관성을 유지할 수 있어요
```

그럼 REST API 설계의 기초 포인트를 알려드릴게요

1. 데이터를 보내거나 받을 때는 JSON 형식을 이용해야돼요 (최근엔 JSON이 거의 표준이긴 해요)
2. 엔드포인트는 명사를 써야돼요

   ```
   // 나쁜 예시, 동사가 들어가있다.
   https://myapi.com/get-posts

   // 좋은 예시, 명사로 이루어져있다.
   https://myapi.com/post
   ```

   그럼 명사로만 이루어져있는데 어떻게 이게 삭제인지, 등록인지, 조회인지 알 수 있을까요?

   이러한 '행동'은 우리가 위에서 배웠던 HTTP 메서드가 처리해요.

   GET=조회, POST=등록, DELETE=삭제가 되겠죠?
3. 다음 URI 규칙을 지켜야해요

   ```
   // 마지막에 '/'를 포함하면 안돼요
   http://myapi.com/user/ (X)
   http://myapi.com/user  (O)

   // 언더바(_) 대신 하이픈(-)을 사용해요
   http://myapi.com/user_group (X)
   http://myapi.com/user-group (O)

   // 파일 확정자를 포함하면 안돼요
   http://myapi.com/photo.png (X)
   http://myapi.com/photo     (O)
   ```

마지막으로, REST API를 설계하다보면 '이 코드는 RESETful 하지 않다' 라는 피드백을 많이 듣게될거에요. RESTful은 간단하게 REST API 아키텍처를 잘 지켜졌는지를 뜻해요.

Class

자바스크립트는 ES6 버전 이후로 `Class`가 추가되어 객체지향 프로그래밍을 좀 더 쉽게 사용할 수 있게 되었어요! ES6 전에는 OOP를 `Prototype`으로 흉내낼 수 있었지만 쓰기가 어려웠어요

(Tip: 자바스크립트는 Prototype 기반 언어입니다)

여기서는 매우 기본적인 문법만 배울게요. Class를 배우려면 세션하나를 통째로 써도 모자랄 만큼 배울게 많기 때문이에요

```
class 붕어빵 {
  // private 프로퍼티, #을 앞에 붙이면 외부에서 접근할 수 없습니다!
  #속재료 = '';
  #가격 = 0;

  // 생성자
  constructor(ingredient, price) {
    this.#속재료 = ingredient;
    this.#가격 = price;
  }

  // getter, setter
  getIngredient() {
    // get ingredient도 가능합니다
    return this.#속재료;
  }
  setIngredient(value) {
    // set ingredident도 가능합니다
    this.#속재료 = value;
  }

  getPrice() {
    return this.#가격;
  }
  setPrice(value) {
    this.#가격 = value;
  }

  // 메서드
  printInfo() {
    return `이 붕어빵은 ${this.getIngredient()} 붕어빵이고 가격은 ${this.getPrice()}입니다`;
  }
}

const 슈붕 = new 붕어빵('슈크림', 100);
console.log(슈붕.getIngredient(), 슈붕.getPrice());
console.log(슈붕.printInfo());
슈붕.setPrice(500);
```

```
console.log(슈붕.getIngredient(), 슈붕.getPrice());
console.log(슈붕.printInfo());

const 팥붕 = new 붕어빵('팥', 300);
console.log(팥붕.getIngredient(), 팥붕.getPrice());
console.log(팥붕.printInfo());
팥붕.setPrice(700);
console.log(팥붕.getIngredient(), 팥붕.getPrice());
console.log(팥붕.printInfo());
```

비동기

아래 코드를 한번 생각해볼까요?

```
const 세탁 = () => {
  console.log('세탁을 시작합니다!');

  // 1.5초 뒤에 세탁이 끝납니다.
  setTimeout(() => console.log('세탁이 끝났습니다!!'), 1500);

  console.log('세탁물을 널어요');
};

세탁();
```

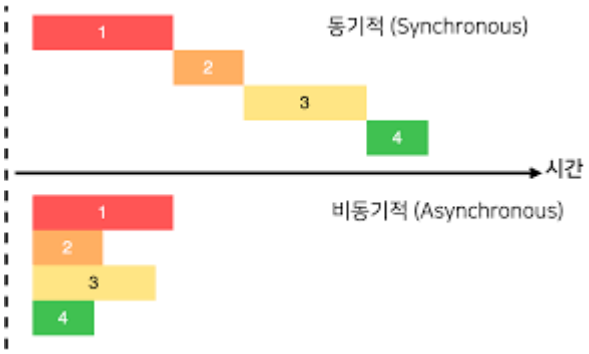
이 코드의 실행 결과를 볼까요??

▼ 실행 결과

https://s3-us-west-2.amazonaws.com/secure.notion-static.com/a2e84469-c52e-45cb-a647-f56113667492/%EB%B9%84%EB%8F%99%EA%B8%B0.js_-_mju-likelion-11th-member-practice_-_Visual_Studio_Code_2023-03-22_23-18-42.mp4

왜 이런 결과가 나올까요?? 이유는 JS는 비동기적으로 작동하기 때문입니다

그럼 비동기는 또 무슨 소리일까요? 아래는 동기 비동기를 표현한 그림입니다



그림을 보면 바로 이해할 수 있겠죠? 그럼 다시 JS는 비동기적으로 작동한다는 것을 고려하고 다시 동작 과정을 생각해볼게요

1. 세탁을 시작합니다
2. 타이머가 동작합니다
3. 비동기에 의해 2번 동작을 완료를 기다리지 않고 세탁물을 널기 시작합니다
4. 타이머가 다 종료되어 세탁이 끝났습니다

어떄요 이해 됐나요?

그럼 이제 의문이 생길거에요 `“그럼 타이머를 기다린 후 실행하는 방법은 없어요?”` 왜 없겠어요 당연히 우리 멋진 JS는 다 준비 되어있어요

```
const 세탁 = () => {
  console.log('세탁을 시작합니다!');

  new Promise((resolve, reject) => {
    setTimeout(() => resolve(console.log('세탁이 끝났습니다!!'), 1500));
  }).then((res) => console.log('세탁물을 널어요'));
};

세탁();
```

갑자기 뭔 Promise라는게 튀어나왔어요 아직 자바스크립트도 이해하기 힘든데 이상한 문법이 튀어나와서 미칠 것 같아요

여기서는 어어엄청 간단하게 요약을 하자면 `‘나중에 결과 줄게 그때까지 기다려 약속!’` 이라 생각하면 돼요. 이렇게 약속을 했으면 지켜야 하기 때문에 결과가 나올 때까지 기다리게 됩니다

그 후 `then` 내부에 약속한 결과를 받은 후 실행할 동작을 넣어주면 돼요

그럼 또 의문이 들거에요 저기 저 resolve, reject는 뭘까요? 간단하게 설명하자면 resolve는 Promise가 성공했을 때의 동작을, reject는 실패했을 때의 동작을 넣으면 돼요. 그래서 현재 코드를 잘 보면 resolve 안에 세탁이 끝났다는 결과를 넣었어요. 만약 실패됐다면? reject 안에 실패했을 때의 동작을 넣으면 돼요.

자 이제 마지막으로 Promise의 세 가지 상태에 대해 배워볼게요

Promise는 총 세 가지 상태가 있어요

- `Pending` ⇒ 프로미스를 처리 중입니다.
- `Fulfiled` ⇒ 프로미스를 정상적으로 완료됐습니다.
- `Rejected` ⇒ 프로미스가 비정상적으로 완료됐습니다.

아직은 비동기가 정확히 와닿지 않을거에요. 아직은 완벽하게 이해하려 하지 말고 추후 세션을 진행하면서 웹-서버와 통신할 때 이 Promise를 적극적으로 활용할 순간이 올거에요. 그 때 되면 '아 이걸 말했던 거구나' 라고 떠올릴 정도면 됩니다!

Tip) Promise가 어렵나요? `Async-Await` 이 있습니다! 이건 각자 공부해보세요!

일급 객체

자바스크립트의 핵심 중 하나는 자바스크립트의 함수는 `일급 객체`라는 것입니다. 일급 객체의 정의는 아래와 같습니다.

> 일급 객체는 다른 객체들에 일반적으로 적용 가능한 연산을 모두 지원하는 객체입니다.

이렇게 말하면 무슨 뜻인지 잘 모르겠죠? 그래서 조금 더 풀어서 말하자면 다음과 같습니다.

> 함수를 데이터 (number, string) 처럼 다룰 수 있습니다.

그래도 모르겠다고요? 그럼 일급 객체의 조건을 말해볼게요

> 1. 일급 객체는 변수나 데이터에 담을 수 있습니다.
> 2. 일급 객체는 함수의 파라미터로 전달할 수 있습니다.
> 3. 일급 객체는 함수의 리턴값으로 사용할 수 있습니다.

그래도 어렵죠? 그래서 예시를 들어볼게요

변수나 데이터에 담을 수 있습니다.

```
// 변수에 함수를 담고 있습니다.
let printHello = function () {
  console.log('안녕하세요!!');
};

printHello();

// c언어로 하면 에러가 발생합니다.
int result = () { return 1+1 }
```

함수의 파라미터로 전달할 수 있습니다.

```
function hello () {
  console.log('저는 11기 아기사자입니다!');
}

function printFunction(someFunction) {
  someFunction() // someFunction이라는 파라미터를 받아 실행합니다
}

printFunction(hello); // printFunction에 hello 함수를 파라미터로 전달합니다.
```

함수의 리턴값으로 사용할 수 있습니다.

```
function returnFunction() {
  console.log('1. 저는 returnFunction이 할당 될 때 실행돼요!!');

  // 함수를 리턴하고 있습니다.
  return function () {
```

```
    console.log('2. 저는 저를 할당한 곳에서 실행될 때마다 실행돼요!!');
  };
}

const closure = returnFunction();
closure();
closure();
```

클로저는 JS 개발자 취업판에서 단골 면접 질문이고 웹 파트원분들의 경우 React를 배울텐데 React의 Hook의 동작 원리가 바로 이 Closure를 이용해 동작합니다! 아직은 ‘아 그렇구나’ 정도로만 넘어가고 나중에 본격적으로 JS를 공부한다면 꼭 공부해보세요!