[ITEC] JS Study

> 1 week

> 브라우저로서 JS, DOM의 이해, defer (js in html), 데이터 타입, 변수 선언 (호이스팅, 스코프)

Contents

0. IDE 셋업

- VSCode, WebStorm, Pycharm(pro)
- prettier 잘 동작??

1. 브라우저 언어로서 JS

- JS란??
- 크롬 개발자 도구 사용법
- JS in HTML

JS in HTML

```
<!-- 1. html 안에 JS 코드 바로 작성 -->
<html>
  <head>
    <meta charset="UTF-8" />
    <title>Hello JS</title>
  </head>
  <body>
    <h1>Hello JS</h1>
    <script>
      alert("hello world!");
    </script>
  </body>
</html>
```

- 폴더 구조
  - src
    - index.js
  - index.html

```
<!-- 2. JS 파일을 따로 분리해서 html에 링크 -->
<html>
  <head>
    <meta charset="UTF-8" />
    <title>Hello JS</title>
  </head>
  <body>
    <h1>Hello JS</h1>
    <script src="src/index.js"></script>
  </body>
</html>
```

```
// index.js
alert("hello world!");
```

```
<!-- 3. defer 속성을 추가해서 head 태그 안에 관리하기 -->
<html>
  <head>
    <meta charset="UTF-8" />
    <title>Hello JS</title>
    <script defer src="src/index.js"></script>
  </head>
  <body>
    <h1>Hello JS</h1>
  </body>
</html>
```

2. CodeSandBox 사용법

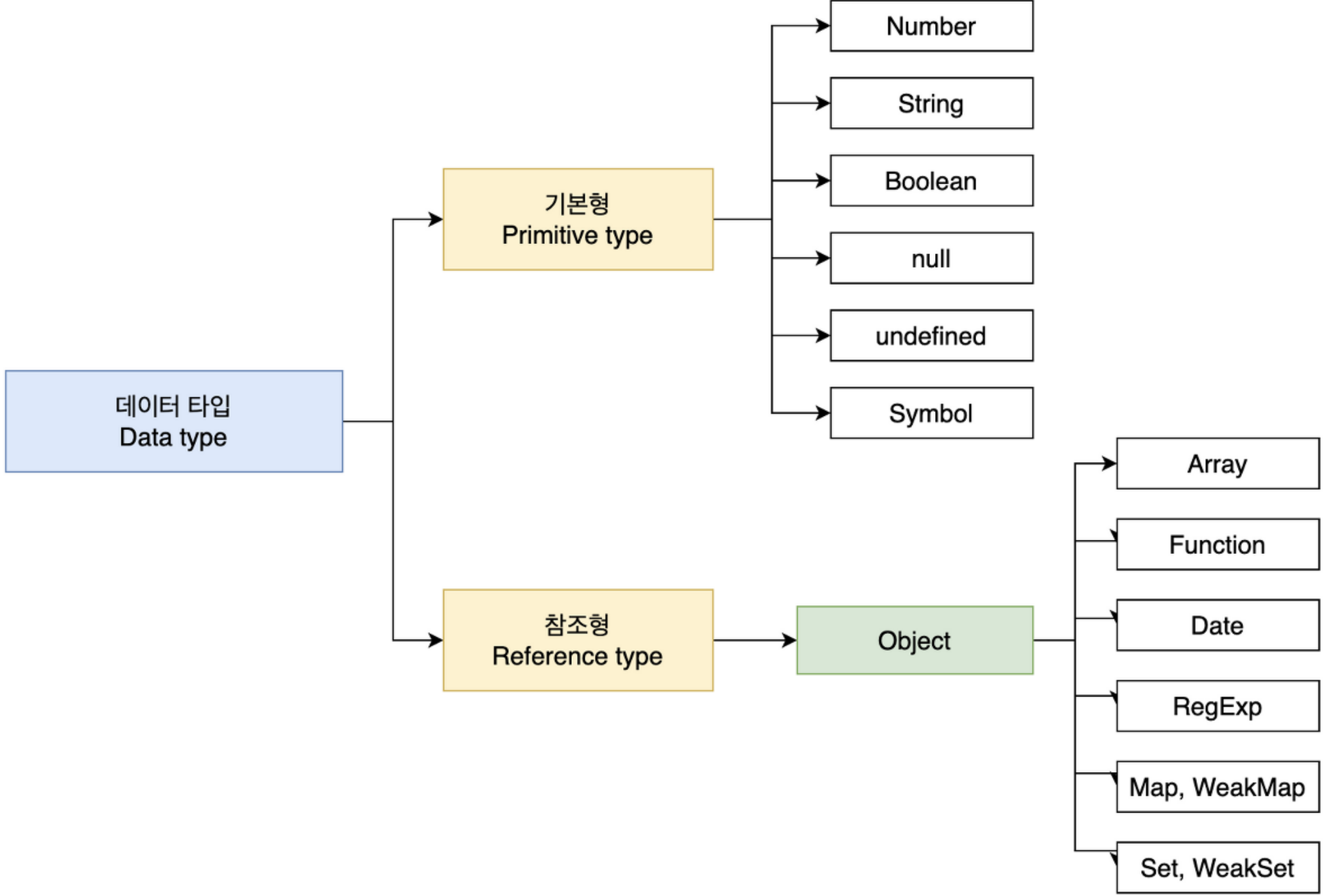
- CodeSandBox IDE를 사용하는 이유
- CodeSandBox에서 로컬 IDE(VSCode, WebStorm, ...)로 작업물 옮기기
- CodeSandBox와 Github 연동하기
- CodeSandBox에서 패키지 추가하기

3. DOM의 이해

- DOM == Documents-Object-Model
- DOM diagram
- DOM 핸들링 예시

4. Data Type

- repo


데이터 타입
Data type
기본형
Primitive type
Number
String
Boolean
null
undefined
Symbol
참조형
Reference type
Object
Array
Function
Date
RegExp
Map, WeakMap
Set, WeakSet

undefined와 null

undefined

JS 엔진은 사용자가 어떤 값을 지정할 것이라고 예상되는 상황임에도 실제로는 그렇게 하지 않았을 때 `undefined`를 반환합니다.

- 값을 대입하지 않는 변수에 접근할 때
- 객체 내부의 존재하지 않는 프로퍼티(속성)에 접근하려고 할 때
- return 문이 없거나 호출되지 않는 함수의 실행 결과

null

'비어있음' (NULL check)을 명시적으로 나타내고 싶을 때는 `undefined`가 아닌 `null`을 쓰면 됩니다.

> typeof null === 'object' (JS 자체 버그)

5. Variables

- var, let, const
- 호이스팅
- 스코프
- 스코프 체인
- 결론

var

- `var` : 변수 생성 키워드
  - 변수 재할당, 재**선언**이 가능하다.
  - `var` 의 변수는 **호이스팅**이 일어난다.
  - `var` 의 변수는 **함수 레벨 스코프**를 갖는다.

```
var name = "amamov";
var name = "wow! :)";
console.log(name);
// wow! :)
```

```
var name;
name = "amamov";
name = "wow! :)";

console.log(name);
// "wow! :)"
```

let

- `let` : 변수 생성 키워드
  - 변수 재할당이 가능하지만 변수 재**선언**이 불가능하다.
  - `let` 의 변수는 **호이스팅**이 일어나지 않는다.
  - `let` 의 변수는 **블록 레벨 스코프**를 갖는다.

```
let name = "amamov";
// let name = "wow! :)"; Error 재선언 불가능

console.log(name);
// "amamov"
```

```
let name;
name = "amamov";
name = "wow! :)";

console.log(name);
// "wow! :)"
```

var & 호이스팅

- **호이스팅**이란 함수 안에 있는 선언들을 모두 끌어올려서 해당 **함수 유효 범위**의 최상단에 선언하는 것을 말합니다.
- 실제로 코드가 끌어올려지는 것은 아니고, 내부적으로 끌어올려서 처리하는 것입니다. 따라서 실제 메모리에는 변화가 없습니다.
- 여기서 **함수의 유효 범위**는 함수 블록 `function { }` 안에서 유효한 범위를 의미합니다.

```
console.log("potato");
var name1 = "yoon"; // var 변수 선언과 동시에 할당
let name2 = "kim"; // let 변수 선언과 동시에 할당
```

```
/* JS 내부의 호이스팅의 결과 */
var name1; // [호이스팅] 변수 "선언"

console.log("potato");
name1 = "yoon"; // 변수 할당
let name2 = "kim";
```

> 코드의 가독성과 유지보수를 위해 호이스팅이 일어나지 않도록 해야 합니다.

스코프

- **스코프 Scope**란 '변수에 접근할 수 있는 범위(영역)'라고 할 수 있습니다.
- JS에서 스코프는 **전역 스코프**, **지역 스코프** 2가지 타입이 있습니다.
- 지역 스코프는 **함수 스코프**, **블록 스코프** 2가지 타입이 있습니다.
- **함수 스코프**는 `function { }` 안에서의 영역을 의미합니다.
- **블록 스코프**는 `{ }` 안에서의 영역을 의미합니다.

var 변수는 함수 스코프를 갖는다.

```
var number = 1;
function test() {
  // 함수 스코프
  var number = 3;
  console.log(number);
}
test(); // 3
console.log(number); // 1
```

> var 변수는 함수 스코프를 갖기 때문에 함수 스코프 내에서 var로 선언된 변수는 그 안에서만 유효하고 함수 외부에서는 유효하지 않다. (참조할 수 없다.)

var은 함수 스코프가 아닌 블록 스코프에서는 마음껏 참조할 수 있다.

```
var number = 1;
if (true) {
  // 블록 스코프
  var number = 3;
  console.log(number); // 3
}
console.log(number); // 3
```

> var 변수는 블록 스코프를 갖지 않기 때문에 함수 스코프가 아닌 블록 스코프에서 자유롭다.

let은 블록 스코프를 갖는다.

```
let number = 1;

if (true) {
  // 블록 스코프
  let number = 3;
  console.log(number); // 3
}

console.log(number); // 1
```

> let 변수는 블록 스코프를 갖기 때문에 블록 스코프 내에서 let로 선언된 변수는 그 안에서만 유효하고 블록 외부에서는 유효하지 않다. (참조할 수 없다.)

함수 스코프도 블록 스코프이다.

```
let number = 1;

function test() {
  let number = 3;
  console.log(number);
}

test(); // 3
console.log(number); // 1
```

```
var x = 0; // var은 함수 레벨 스코프
{
  var x = 1;
  console.log(x); // 1
}
console.log(x); // 1
```

```
let y = 0; // let는 블록 레벨 스코프
{
  let y = 1;
  console.log(y); // 1
}
console.log(y); // 0
```

```
{
  var x = 7;
  console.log(x); // 7
}

console.log(x); // 7
```

```
{
  let x = 7;
  console.log(x); // 7
}

console.log(x); // Error, x is not defined.
```

대부분의 C-family language는 블록 레벨 스코프(block-level scope)를 따릅니다.

복습하자면, 블록 레벨 스코프란 코드 블록({...})내에서 유효한 스코프를 의미합니다.

여기서 “유효하다”라는 것은 “참조(접근)할 수 있다”라는 뜻입니다.

dart 언어 코드를 보면 if문 내에서 선언된 변수 number는 if문 코드 블록 내에서만 유효합니다. 즉, if 문 코드 블록 밖에서는 참조가 불가능합니다.

dart 언어

```dart
void main() {
  int number = 1;
  if (true) {
    int number = 3;
    print(number); // 3
  };
  print(number); // 1
}
```

JS의 let

```javascript
let number = 1;
if (true) {
  let number = 3;
  console.log(number); // 3
}
console.log(number); // 1
```

파이썬

```python
number = 1

if True:
    number = 3
    print(number) # 3

print(number) # 3
```

JS의 var

```javascript
var number = 1;
if (true) {
  var number = 3;
  console.log(number); // 3
}
console.log(number); // 3
```

스코프 체인

> 현재 자신의 스코프에서 사용하고자 하는 변수가 없다면
> 스코프 체인을 통해 해당 변수를 찾게 됩니다.

```
var number = 1;

function test() {
  console.log(number);
}

test(); // 1
console.log(number); // 1
```

```javascript
// let도 동일하게 동작
let number = 1;

function test() {
  console.log(number);
}

test(); // 1
console.log(number); // 1
```

```python
# 파이썬도 동일하게 동작
number = 1

def test():
    print(number)

test() # 1
print(number) # 1
```

```dart
// dart 언어도 동일하게 동작

void main() {
  int number = 1;

  void test() {
    print(number);
  };

  test(); // 1
  print(number); // 1
}
```

> 상위 스코프의 변수를 스코프 체인을 통해 읽을 수 있습니다.

주의!! 1

```
// JS
let number = 1;

function test() {
  console.log(number); // undefined
  let number = 3;
  console.log(number);
}

test(); // 3
console.log(number); // 1
```

```python
# Python
number = 1

def test():
    print(number)
    number = 3 # IndentationError: unindent does not match any outer indentation level
    print(number)

test()
print(number)
```

```dart
// dart lang
void main() {
  int number = 1;

  void test() {
    print(number);
    int number = 3;
    // Error: Can't declare 'number' because it was already used in this scope.
    print(number); // Context: Previous use of 'number'.
  };

  test();
  print(number);
}
```

주의!! 2

```
// JS
let number = 1;
function test() {
  let number = number + 3;
  console.log(number);
}
test(); // NaN
console.log(number); // 1
```

```
// JS
let number = 1;
function test() {
  // 재할당은 가능합니다.
  number = number + 3;
  console.log(number);
}
test(); // 4
console.log(number); // 4
```

```dart
// dart lang
void main() {
  int number = 1;
  void test() {
    // 재할당은 가능합니다.
    number = number + 3;
    print(number);
  };
  test(); // 4
  print(number); // 4
}
```

```python
# Python
number = 1

def solution():
    number = number + 3
    # UnboundLocalError: local variable 'number' referenced before assignment
    print(number)

solution()
print(number)
```

> 상위 스코프의 변수를 스코프 체인을 통해 읽을 수 있습니다.
> 하지만 파이썬일 경우 수정은 불가능합니다. (global 키워드로 가능하긴 합니다.)

const

- `const` : 변수 생성 키워드
  - 변수 재할당, 재선언이 **불가능하다.**
  - `const` 의 변수는 **호이스팅**이 일어나지 않는다.
  - `const` 의 변수는 **블록 레벨 스코프**를 갖는다.

```
const name = "amamov";
// name = "hello"; Error
console.log(name);
```

```
const name; // Error : Const declarations require an initialization value

console.log(name);
```

> 변수 재할당이 불가능하다는 것은
> 변수의 주소값이 한번 할당 되면 변경이 불가능하다는 의미입니다.

- `var`
  - 변수 재할당, 재**선언**이 가능하다.
  - **호이스팅**이 일어난다.
  - **함수 레벨 스코프**를 갖는다.
- `let`
  - 변수 재할당이 가능하지만 변수 재**선언**이 불가능하다.
  - **호이스팅**이 일어나지 않는다.
  - **블록 레벨 스코프**를 갖는다. (블록 안에서 선언을 했다면, 블록 밖에서는 사용 불가능)
  - 블록 밖에서 선언을 해서 블록 안에서 값 변경 가능 (스코프 체인)
- `const`
  - 변수 재할당, 재선언이 **불가능하다.**
  - **호이스팅**이 일어나지 않는다.
  - **블록 레벨 스코프**를 갖는다. (블록 안에서 선언을 했다면, 블록 밖에서는 사용 불가능)

> var을 사용하면 변수 선언의 경우 할당되는 값이 유동적으로 변경될 수 있고 호이스팅이 일어나기 때문에 코드의 가독성과 유지보수가 어려워집니다. 또한 함수 레벨 스코프이므로 strict하지 않는 단점을 가졌습니다.

결론

var 사용은 피하고 let / const를 사용하되,

let은 변수(주소값) 재할당이 필요할 때만 사용하자.

결론적으로 일반적으로 변수를 선언할 때는 const를 사용하자.

> ??? : 왜 API나 여러 공식문서엔 var를 많이 쓸까?
> -> 호환성 문제 때문입니다. 하지만 babel을 사용하여 최신 JS 문법을 바닐라 JS로 문제 없이 트랜스컴파일링을 할 수 있기 때문에 결론에 문제될 것은 없습니다.

첼린지

1. var, let, const 차이점 이해와 결론 명심
2. 모던 JS - 2.7까지 정독
3. 간단한 알고리즘 문제 해결 후 CodeSandBox URL을 보내주세요.

> 정수 배열 numbers가 주어집니다. numbers에서 서로 다른 인덱스에 있는 두 개의 수를 뽑아 더해서 만들 수 있는 모든 수를 배열에 오름차순으로 담아 return 하도록 solution 함수를 완성해 주세요. (numbers의 길이는 2 이상 100 이하이고 numbers의 모든 수는 0 이상 100 이하입니다.)

```
function solution(numbers) {
  const answer = [];
  // write the code.
  return answer;
}
```