

JS 자료형

1) Primitive (기본형) → 더이상 작은 단위로 나눌 수 없는 하위자의 아이템들

- Boolean : true, false * false = 0, null, undefined, NaN, "", ' ',

- Null : 빈값을 표현

- Undefined : 값을 할당하지 않은 변수가 가지는 값 → 스트링 값을 숫자로 나눌 경우.

- Number : 숫자형으로 정수와 부동 소수점, 무한대, NaN (숫자가 아님) (+ BigInt)

- String : 문자열... ' ', " ", ` `
 $\frac{1}{0} = \text{Infinity}$
 $-\frac{1}{0} = \text{Negative Infinity}$
 $\{ \}$ ←

- Symbol : 식별자 (무선호위 위해 사용) Java의 Package 느낌

⇒ 같은 값은 사용해도 다르게 표현 (Symbol.for() 는 동일한 값 만들어줌)

* 종래시 .description 사용해야 출력 가능

2) Object → 위 Primitive 들은 묶어주는 아이템.

- Reference type → First Class Function 도 포함.

- 클래스, 배열, 함수, 사용과 정의 클래스.

- JSON (Java Script Object Notation)의 기본구조.

변수 선언 (let/const/var) 새로운 변수를 생성 keyword

- Var/let : mutable

- Const : immutable

(변수 이름 : 대소문자 구분
여러번 한 번에 선언 가능
지역변수 / 전역변수

변수 선언시 데이터 타입 미리 지정 X

JSON (JavaScript Object Notation)

· 데이터 형식

· 데이터를 저장하거나 전송할 때 많이 사용되는 경량의 DATA 교환 형식

· 객체를 만들어 사용.

· 용량 ↓, 사람/기기 모두 이해 수월.

· JSON 데이터 포맷 / 단순히 데이터를 표시하는 방법.

여기까지의 API에서 사용이 가능

Dynamic typing (dynamically typed language)

- 값은 문맥을 넘나들기 따라 데이터 타입 변환.
 - $\text{String} + \text{number} = \text{String}$ ex) $\text{text} = '7' + 5 \Rightarrow 15 (\text{string})$
 - $\text{String} / \text{string} = \text{number}$ ex) $'8' / '2' \Rightarrow 4 (\text{number})$
- ※ 주의 사항 : Runtime 에서 type 미정해 지기 때문에 error가 Runtime으로 발생.
→ TypeScript 의 탄생

favor immutable data type always

- security : 보안 문제 방지
- ★ - thread safety : 동시화 문제 발생성 예방.
- reduce human mistake : 실입, 동조 실수 방지.

Hoisting

화물 운반 장비, 화물을 들어올려서 사용 → '들어 올린다'라는 의미'

코드에 선언된 변수 / 함수만 코드 상단을 끌어 올려라

함수 내 변수: 해당 함수의 최상위

함수 밖 전역변수: 스크립트 당 최상위 * 함수 선언식 인 경우만 해당

```
noDefine();

function noDefine() {
  // 변수 선언 및 할당 이전에 호출 테스트
  console.log("not defined : " + name);
  var name = "ojava";
  // 변수 초기화 이후 값 확인
  console.log("defined : " + name);
}
```

```
not defined : undefined
defined : ojava
```

```
not defined : ojava
defined : ojava
```

변수 * → 코드에서 오류가 나야 함!

But! JS 엔진에서

var name 과 name = "ojava"로 분리.

변수를 스코프 내 최상위 부분으로 옮긴다.

⇒ Console에서 undefined를 반환

* 주의사항

변수 선언 * → 전역 변수의 형태로 사용됨

* Temporal Dead Zone

let/const는 TDZ에 의해 제막

⇒ var처럼 undefined 반환 * 에러 발생

⇒ 코드 예측 가능, 잠재적으로 쉽게 찾을 수 있음. 실행 시 오류 발생.

↓
* let/const는 Hoisting이 되나 않는다? NO!

```
var name = 'seo' // 전역변수
{
  console.log(name) // 에러 발생
  is not defined
  let name = 'seo' / 지역변수
}
```

에러가 발생.

Var는 선언과 동시에 초기화
→ undefined

let은 분리 → 호이스팅 되면 선언 단계로
초기화 단계 실행 코드에서 실행

⇒ reference error

Const

```
var name;  
name = 'seo'
```

```
const age = 29; // 선언과 동시에  
초기화 필요
```

```
//const age; // const 변수를  
선언만 할 경우 에러가 발생  
//age = '29'; error
```

let과 유사하나 But 선언되면 값이 상수화
⇒ 변경이 불가능.

선언과 동시에 초기화가 필요.
⇒ 값을 변경 X

let/const = { } 내부에서 생명주기 유지

var는 함수 스코프 가정

⇒ { } 내부에 선언되어도
외부에서 접근 가능!

SCOPE
{
 let name = 'seo'
}

```
console.log(name) //  
ReferenceError: name is not  
defined
```

```
{  
  var age = 29;  
}
```

```
console.log(age) // 29 , var는  
함수 레벨 스코프
```

```
1 function DataTypeBasic() {
2   // var는 변수 선언할 때 사용한다.
3   var length = 7 // 정수형도 var
4   var lastName = "Gogosing" // 문자열도 var
5   var x = {firstName: "Rust", lastName: "Golang"} // 묶음 형식도 var
```

```
6
7   // console.log()를 통해서
8   // 웹 페이지에서 F12를 누르고 콘솔을 눌러 mac은 ⌘ + Cmd C
9   // 콘솔 메시지를 확인할 수 있다.
10  // 즉 브라우저용 디버깅에 유용하게 활용할 수 있다.
11  console.log("DataTypeBasic: " + length)
12  console.log("DataTypeBasic: " + lastName)
13  console.log("DataTypeBasic: " + x.firstName)
14  console.log("DataTypeBasic: " + x.lastName)
```

```
15
16  // div의 약자는 Division으로
17  // 웹 사이트의 레이아웃(전체적인 틀)을 만들때 사용함
```

```
18
19  // p는 paragraph 태그로 하나의 문단을 만들때 사용된다.
```

```
20
21  // div에 className이라는 것이 보이는데
22  // 이 부분은 향후 CSS를 할 때 좀 더 자세히 알아볼 것임
```

```
23  return (
24    <div className="DataTypeBasic">
25      <p>
26        {length}, {lastName}, {x.toString()}. 문자열로 나뉜
27      </p> 7 Gogosing
28    </div>
29  )
30 }
```

```
31
32 export default DataTypeBasic
```

```
1  function DataType() {
2      // 숫자 + 문자열은 숫자를 강제로 문자열로 바꿈
3      var test1 = 7 + "Test"
4      var test2 = "7" + "test"
5      // 숫자 + 숫자는 숫자로 계산하고
6      // 이후에 숫자 + 문자열은 문자열로 인해 숫자를 문자열로 바꿈
7      var numTest = 3 + 7 + "Test"
8
9      console.log("DataType: " + test1)
10     console.log("DataType: " + test2)
11     console.log("DataType: " + numTest)
12
13     return (
14         <div className="DataType">
15             <p>
16                 {test1}, {test2}, {numTest}.
17             </p>
18         </div>
19     )
20 }
21
22 export default DataType
```

```

1 function BoolDataType() {
2     var num1 = 3, num2 = 3, num3 = 7
3
4     var boolRes1 = (num1 === num2)
5     var boolRes2 = (num1 === num3)
6
7     console.log("BoolDataType: " + boolRes1)
8     console.log("BoolDataType: " + boolRes2)
9
10    var testNum = 0
11    var testStr = "0"
12
13    // '==='의 경우 데이터만 확인한다.
14    // '===의 경우 데이터와 해당 데이터의 타입을 함께 확인한다.
15    // 즉 데이터타입이 다르다면 false가 된다.
16
17    // 결론: 결국 자바스크립트는 데이터타입이 없는 것이 아니라
18    // 실행중에 동적으로 데이터타입이 생성되는 것이다.
19    // 데이터타입이 없다고 얘기하는 것은 매우 큰 오개념이다.
20    var boolRes3 = (testNum == testStr) true
21    var boolRes4 = (testNum === testStr) false
22
23    console.log("BoolDataType: " + boolRes3)
24    console.log("BoolDataType: " + boolRes4)
25
26    return (
27        <div className="BoolDataType">
28            <p>
29                {boolRes1.toString()}, {boolRes2.toString()}<br/>
30                {boolRes3.toString()}, {boolRes4.toString()}<br/>
31            </p>
32        </div>
33    )
34 }
35
36 export default BoolDataType

```

```
1  function ExpDataType() {
2      // 123 x 10^5
3      var expNum1 = 123e5
4      // 123 x 10^-5
5      var expNum2 = 123e-5
6
7      console.log("ExpDataType: " + expNum1)
8      console.log("ExpDataType: " + expNum2)
9
10     return (
11         <div className="ExpDataType">
12             <p>
13                 {expNum1}, {expNum2}
14             </p>
15         </div>
16     )
17 }
18
19 export default ExpDataType
```



```
1  function ArrayDataType() {  
2      var cars = ["BMW", "Volvo", "Audi", "Toyota", "Tesla"]  
3  
4      console.log("ArrayDataType: " + cars)  
5  
6      return (  
7          <div className="ArrayDataType">  
8              <p>  
9                  {cars[0]}, {cars[1]}, {cars[2]}, {cars[3]}, {cars[4]}  
10             </p>  
11         </div>  
12     )  
13 }  
14  
15 export default ArrayDataType
```

The diagram illustrates the mapping of array elements to JSX elements. Red arrows point from each element in the 'cars' array to its corresponding index in the JSX output. The first and last elements of the array and the closing bracket of the array are circled in red.

```
1  function JsonDataType() {
2      var person = {
3          name: "Sanghoon",
4          major: "Electronics",
5          minor: "Computer Science",
6          numOfRepo: 200
7      }
8
9      console.log("JsonDataType: " + person)
10
11     return (
12         <div className="JsonDataType">
13             <p>
14                 {person.name}, {person.major}<br/>
15                 {person.minor}, {person.numOfRepo}<br/>
16             </p>
17         </div>
18     )
19 }
20
21 export default JsonDataType
```

```
1 function StringDataType() {
2     // 자바스크립트는 아래와 같이
3     // 문자열 자체에 줄 따옴표, 쌍따옴표 작성이 문제가 되지 않는다.
4     // 적고 싶은 형태로 마음대로 작성해도 무방하다.
5     var strTest1 = "It's OK"
6     var strTest2 = "I can 'use' this"
7     var strTest3 = 'I can do "this" too'
8
9     console.log("StringDataType: " + strTest1)
10    console.log("StringDataType: " + strTest2)
11    console.log("StringDataType: " + strTest3)
12
13    // <br/> 태그는 엔터와 유사한 역할을 수행한다 보면 되겠다.
14    return (
15        <div className="StringDataType">
16            <p>
17                {strTest1}<br/>
18                {strTest2}<br/>
19                {strTest3}<br/>
20            </p>
21        </div>
22    )
23 }
24
25 export default StringDataType
```

```
1 function RealDataType() {
2     // 소수점의 경우 .0의 경우 출력시 정수형과 별 다른 차이가 없음
3     var floatNum1 = 33.0
4     var intNum = 33
5     // 명확하게 0.1 형태의 숫자가 있을때는 출력한다.
6     var floatNum2 = 37.1
7     // .00 의 경우에는 소수점이라는 것을 알려주기 위해 . 을 표현해준다.
8     var floatNum3 = 33.00
9
10    console.log("RealDataType: " + floatNum1)
11    console.log("RealDataType: " + intNum)
12    console.log("RealDataType: " + floatNum2)
13    console.log("RealDataType: " + floatNum3)
14
15    return (
16        <div className="StringDataType">
17            <p>
18                {floatNum1}, {intNum}, {floatNum2}, {floatNum3}.
19            </p> 33      33      37.1      33.
20        </div>
21    )
22 }
23
24 export default RealDataType
```

```
1  function TypeOf() {
2      return (
3          <div className="IfTest">
4              <p>
5                  TypeOf Works Fine!
6              </p>
7              <p>
8                  {typeof ""}, {typeof "test"}, {typeof 0}, {typeof 3.14}, <br/>
9                  {typeof true}, {typeof undef}, {typeof undefined}, {typeof null}, <br/>
10                 {typeof {name: 'Test', major: 'EE'}}, {typeof [1, 2, 3, 4]},
11                 {typeof function test(){} }<br/>
12             </p>
13         </div>
14     )
15 }
16
17 export default TypeOf
```

String int float

Boolean

Json

method

array

```
1  function ForTest() {
2      for(var i = 0; i < 3; i++) {
3          console.log("ForTest: " + i)
4      }
5
6      return (
7          <div className="IfTest">
8              <p>
9                  ForTest Works Fine!
10             </p>
11         </div>
12     )
13 }
14
15 export default ForTest
```

3개 연산자 + 데이터 타입 확인


```
1  function IfTest() {
2      var num1 = 3, num2 = 10
3
4      if (num1 !== 10) {
5          console.log("IfTest: num1 !== 10")
6      } else if (num2 !== 10) {
7          console.log("IfTest: num2 !== 10")
8      } else {
9          console.log("IfTest: works fine!")
10     }
11
12     return (
13         <div className="IfTest">
14             <p>
15                 IfTest Works Fine!
16             </p>
17         </div>
18     )
19 }
20
21 export default IfTest
```

```
1 function LetVar() {
2     let num = 77
3
4     console.log("num = " + num)
5     console.log("let은 Hoisting 되지 않으며 let 위에서 해당 변수를 사용할 수 없다.")
6     console.log("var는 되나요 ? " + varTest + " 는 되네 ?!")
7
8     var varTest = "이거 되는거 맞냐 ?"
9
10    const conNum = 33
11
12    varTest = "정말 되는거야 ?"
13
14    console.log("conNum = " + conNum)
15    console.log("const는 반드시 선언과 함께 값의 초기화가 이루어져야 한다.")
16
17    return (
18        <div className="LetVar">
19            <p>
20                let {num}<br/>
21                const {conNum}<br/>
22                var {varTest}<br/>
23            </p>
24        </div>
25    )
26 }
27
28 export default LetVar
```

Handwritten annotations in orange:

- Line 4: `num` is annotated with `77`.
- Line 8: `var` is annotated with `호이스팅` (Hoisting) and an arrow pointing to the `varTest` variable.
- Line 10: `const` is annotated with `33`.
- Line 12: `varTest` is circled with an arrow pointing to the `var {varTest}` line in the JSX element.


```
1  function LetVar2() {
2      let num = 77
3
4      console.log("let test: " + num)
5
6      num = "뭐야 ?"
7
8      return (
9          <div className="LetVar2">
10             <p>
11                 let {num}<br/>
12             </p>
13         </div>
14     )
15 }
16
17 export default LetVar2
```



Handwritten orange text "hη" is located next to the console.log statement on line 4.

```
1  function LetVar3() {
2      var varNum = 33
3
4      console.log("var test: " + varNum)
5
6      var varNum = "뭐야 ?"
7
8      console.log("var test: " + varNum)
9
10     return (
11         <div className="LetVar3">
12             <p>
13                 var {varNum}<br/>
14             </p>
15         </div>
16     )
17 }
18
19 export default LetVar3
```

```

1 function LetVar4() {
2   let letNum = 33
3   ① / ②
4   console.log("let test: " + letNum)
5
6   // LetVar3과 LetVar4의 차이점은 뭘까 ?
7   // let 과 var의 차이점은 뭘지 ?
8   // 호이스팅(Hoisting)
9   var letNum = "뭐야 ?"
10
11  // 호이스팅은 모든 변수가 프로그램 시작시
12  // 최선두에 선언하는 형식과 같이 동작하는 것을 의미한다.
13  // 좀 더 쉽게 말하자면 var는 변수 선언과 값의 할당이 통합된다.
14  // 반면 let은 변수 선언과 값의 할당이 분리되어 있다.
15
16  // 그러다보니 var는 변수 생성시 이름이 같은게 있으면
17  // Mangling 이라는 방식을 통해서 변수의 이름을 바꿔버린다.
18  // 반면 let은 선언과 할당이 분리되어 있다보니
19  // 왜 같은것을 선언하면서 문법 오류가 나는 것이다.
20
21  // var 대신 let을 도입하여 좋아진점
22  // 신입 개발자들이 var로 같은 변수를 도배하다보니
23  // 이전에 중요한 정보가 있었는데 그 정보가 덮어쓰기 되는 일이 많았다.
24  // 반면 중요 정보를 let 으로 만들어둬으로써
25  // 신입 개발자들의 이와 같은 실수를 원천 차단할 수 있게 되었다.
26
27  console.log("let test: " + letNum)
28
29  return (
30    <div className="LetVar4">
31      <p>
32        let {letNum}<br/>
33      </p>
34    </div>
35  )
36 }
37
38 export default LetVar4

```

① / ②
 같다. → 티끌
 동시