JavaScript 기초문법

❖ 학습해야 할 내용
- ✓ 변수 (var, let, const)
- ✓ 타입과 연산자
- ✓ 조건문과 반복문
- ✓ 함수
- ✓ 자료구조 (Object와 Array)

1. 변수

- **let :** 값을 재할당 할 수 있는 변수를 선언하는 키워드
  - 한 번만 선언할 수 있으며 여러 번 할당할 수 있다.

```
let x = 1
x = 3
```

  - 재선언 할 경우 아래의 에러가 발생한다.

```
let x = 1
let x = 3

SyntaxError: Identifier 'x' has already been declared
```

  - 블록 유효범위를 갖는다. (if, for 그리고 함수 등의 중괄호{ } 내부를 의미)

```
let x = 1

if (x === 1) {
  let x = 2
  console.log(x)  // 2
}

console.log(x)    // 1
```

JavaScript 기초문법

- const : 값이 변하지 않는 상수를 선언하는 키워드
  - 선언 시 반드시 초기값을 설정해주어야 한다.

```
const myFav = 7
```

  - 초기값을 설정하지 않으면 아래의 에러가 발생한다.

```
const myFav

Uncaught SyntaxError: Missing initializer in const
declaration
```

  - 상수의 값은 재할당 될 수 없고 재선언 될 수 없다.

```
myFav = 5

Uncaught TypeError: Assignment to constant variable.
```

```
const myFav = 10
var myFav = 10
let myFav = 10

Uncaught SyntaxError: Identifier 'myFav' has already
been declared
```

  - let과 동일하게 블록 유효 범위(block scope)를 가지고 있다.

1.var : ES6 이전 변수를 선언하는 키워드
  - var 키워드로 선언된 변수는 같은 var 키워드로 재선언 할 수 있다.

```
var num = 1
var num = 2
console.log(num)  // 2
```

JavaScript 기초문법

- 함수 유효 범위(function scope)를 가지고 있다.
- var 키워드로 선언된 변수의 범위는 현재 실행 문맥이며, 그 문맥은 둘러싼 함수 혹은 전역일 수 있다.

```
var a = 1
let b = 2

if (true) {
  var a = 11        // 전역변수 a 덮어쓰기
  let b = 22        // if 내 지역변수
  console.log(a)    // 11
  console.log(b)    // 22
}

console.log(a)      // 11
console.log(b)      // 2
```

- var는 예기치 않은 문제를 많이 발생시키는 키워드로 **절대 사용하지 않는다.**

❖ 정리

var : 재할당, 재선언, 함수 스코프
let : 재할당, 블록 스코프
const : 블록 스코프

JavaScript 기초문법

2. 타입과 연산자

2.1 타입

- Number

```
const a = 13
const b = -5
const c = 3.14
const d = 2.998e8    // 2.998 * 10^8 = 299,800,000
const e = Infinity   // -Infinity 도 가능
const f = NaN        // Not a Number
```

- Boolean

```
const isTrue = true
const isFalse = false
```

- Empty Value
  - 값이 존재하지 않음을 표현하는 값으로 null과 undefined가 있다. 큰 차이를 두지 않고 interchangeable하게 사용하도록 권장한다.
  - undefined는 값이 없을 경우 JavaScript가 할당 하는 값, null은 값이 없음을 표현하기 위해서 개발자가 인위적으로 사용하는 값으로 여길 수 있다.

```
// 선언만 하고 초기화하지 않음
let a
console.log(a)  // undefined


// 의도적으로 값이 없음을 표현
let b = null
console.log(b)  // null
```

JavaScript 기초문법

- undefined와 null은 typeof 연산자를 통해 서로 다른 값이 반환된다.

```
typeof null         // object
typeof undefined    // undefined
```

2.2 연산자

- **할당 연산자**

```
let c = 0
c += 10    // 10 - c에 10을 더한다.
c -= 3     // 7  - c에 3을 뺀다.
c *= 10    // 70 - c에 10을 곱한다.
c++        // 71 - c에 1을 더한다. (증감식)
c--        // 70 - c에 1을 뺀다. (증감식)
```

- **비교 연산자**
  - 문자열 비교는 영어 소문자가 대문자보다 큰 값을 가진다. 알파벳은 오름차순으로 순서로 비교한다.

```
3 > 2       // true
3 < 2       // false

'A' < 'B'   // true
'Z' < 'a'   // true
'가' < '나'  // true
```

- **동등 연산자**
  - 비교 대상이 서로 다른 타입일 경우, 비교하기 전에 가능하다면 같은 자료형으로 형변환하여 비교한다.
  - 이러한 형변환은 예기치 못한 결과를 야기할 수 있기 때문에 동등 연산자의 사용은 지양한다.

JavaScript 기초문법

```
const a = 1
const b = '1'

console.log(a == b)           // true
console.log(a == Number(b))   // true - Number를 통해 숫자로 형변환
```

- **일치 연산자**
  - 타입과 값이 모두 같은지 비교한다. 동등 연산자와 다르게 엄격한 비교를 하기 때문에 일치 연산자를 사용하는 것을 권장한다.

```
const a = 1
const b = '1'

console.log(a === b)           // false
console.log(a === Number(b))   // true
```

- **논리 연산자**
  - boolean 타입을 연산할 수 있는 연산자로 다음과 같이 세가지 연산을 지원한다 : `and`, `or`, `not`
  - and 연산은 `&&` 연산자를 통해 연산한다. 모두 참일 경우 `true`를 반환한다.

```
true && false // false
true && true  // true

1 && 0  // 0
0 && 1  // 0
4 && 7  // 7
```

JavaScript 기초문법

- or 연산은 || 연산자를 통해 연산한다. 둘 중 하나라도 참일 경우 true를 반환한다.

```
false || true   // true
false || false  // false

1 || 0  // 1
0 || 1  // 1
4 || 7  // 4
```

- not 연산은 ! 연산자를 통해 연산하며, 단일 값에 사용하는 단항 연산자로 해당 논리 값을 반대로 뒤집는다.

```
!true  // false
```

- **삼항 연산자**
  - 조건식이 참이면 : 앞의 값이 반환되며 그 반대일 경우 : 뒤의 값이 반환되는 연산자다.
  - 삼항 연산자의 중첩 사용은 지양하며, 일반적으로 한 줄에 표현한다.

```
true ? 1 : 2    // 1
false ? 1 : 2   // 2

const result = Math.PI > 4 ? 'Yep' : 'Nope'
console.log(result)  // Nope
```

JavaScript 기초문법

3. 조건문과 반복문

3.1 조건문

- **if, else if, else**

```
const name = 'manager'

if (name === 'admin') {
  console.log('관리자님 환영합니다.')
} else if (name === 'manager') {
  console.log('매니저님 환영합니다.')
} else {
  console.log(`${name}님 환영합니다.`)
}
```

- **switch**
  - switch 문은 하나의 표현식을 평가하여, 일치하는 항목의 case 절을 실행하는 조건문이다. 일치하는 항목이 없다면 default 절을 실행한다.
  - break 키워드를 통해 switch 문을 벗어난다는 것을 명시한다.

```
const name = '홍길동'

switch(name) {
  case 'admin': {
    console.log('관리자님 환영합니다.')
    break
  }
  case 'manager': {
    console.log('매니저님 환영합니다.')
    break
  }
  default: {
    console.log(`${name}님 환영합니다.`)
  }
}
```

JavaScript 기초문법

- break 키워드가 명시되지 않을 경우 switch 문을 벗어나지 못하고 아래의 case와 default 절까지 실행하게 된다.

```
> const name = 'admin'

  switch(name) {
      case 'admin': {   // 실행
          console.log('관리자님 환영합니다.')
      }
      case 'manager': { // 실행
          console.log('매니저님 환영합니다.')
      }
      default: {        // 실행
          console.log(`${name}님 환영합니다.`)
      }
  }
  관리자님 환영합니다.
  매니저님 환영합니다.
  admin님 환영합니다.
```

3.2 반복문

- **while**

```
let i = 0

while (i < 6) {
  console.log(i)  // 0 1 2 3 4 5
  i++
}
```

- **for**
  - 사용할 변수 하나를 정의하고, 변수가 특정 조건에 대해 false가 될 때까지 연산하며 반복하는 반복문이다.

```
for (let i = 0; i < 6; i++) {
  console.log(i)  // 0 1 2 3 4 5
}
```

JavaScript 기초문법

- **for of**
  - 배열에서 요소를 하나씩 순회하며 반복하는 반복문이다.
  - 매 요소는 블럭 내에서 새롭게 선언되기 때문에 반드시 변수 선언 키워드를 작성한다.

```
const numbers = [0, 1, 2, 3]

for (const number of numbers) {
  console.log(number)  // 0, 1, 2, 3
}
```

- **for in**
  - Object의 key를 순회하는 반복하는 반복문이다. Array의 경우 index를 순회한다.

```
const fruits = { a: 'apple', b: 'banana' }

for (const key in fruits) {
  console.log(key)          // a, b
  console.log(fruits[key])  // apple, banana
}
```

```
const fruits = ['apple', 'banana']

for (const idx in fruits) {
  console.log(idx)          // 0, 1
  console.log(fruits[idx])  // apple, banana
}
```

JavaScript 기초문법

4. 함수

- **함수 선언식**

```
function add (num1, num2) {
  return num1 + num2
}

add(2, 7)  // 9
```

- **함수 표현식**
  - 위 처럼 이름이 없는 함수를 익명 함수(anonymous function)라고 한다. 익명 함수는 함수 표현식에서만 사용할 수 있다.

```
const sub = function (num1, num2) {
  return num1 - num2
}

sub(7, 2)  // 5
```

  - 기명 함수도 함수 표현식이 가능하다.

```
const mysub = function sub (num1, num2) {
  return num1 - num2
}

mysub(7, 2)  // 5
```

- **기본 인자(Default Arguments)**

```
const greeting = function (name = '홍길동') {
  console.log(`안녕하세요 ${name}님!`)
}
```

JavaScript 기초문법

- **화살표 함수**
  - 함수 선언 시 function 키워드와 중괄호를 생략하기 위해 고안된 단축 문법이다.

```
const arrow = function (name) {
  return `hello! ${name}`
}

// 1. function 키워드 삭제, 화살표 추가
const arrow = (name) => { return `hello, ${name}` }

// 2. 매개변수가 하나일 경우 `( )` 생략
const arrow = name => { return `hello, ${name}` }

// 3. 함수 바디가 하나의 표현식일 경우 `{ }` & return 생략
const arrow = name => `hello, ${name}`

// 4. 단, 표현식이 object 객체일 경우 `( )` 안쪽에 객체 표현
const arrow = name => ({ message: `hello, ${name}` })
```

JavaScript 기초문법

5. 자료구조 (Array와 Object)

5.1 Array (배열)

- 기본 사용법

```
const numbers = [1, 2, 3, 4]

numbers[0]      // 1
numbers[-1]     // undefined => 정확한 양의 정수 index 만 가능
numbers.length  // 4
```

- reverse

```
numbers.reverse()  // [4,3,2,1]
numbers            // [4,3,2,1]
numbers.reverse()  // [1,2,3,4]
numbers            // [1,2,3,4]
```

- push & pop

```
numbers.push('a')  // 5 => 새로운 배열의 길이
numbers            // [1,2,3,4,'a']

numbers.pop()      // 'a' => 가장 마지막 요소
numbers            // [1,2,3,4]
```

- unshift & shift

```
numbers.unshift('a')  // 5 => 새로운 배열의 길이
numbers               // ['a',1,2,3,4]

numbers.shift()       // 'a' => 가장 처음 요소
numbers               // [1,2,3,4]
```

JavaScript 기초문법

- **includes**

```
numbers.includes(1)  // true
numbers.includes(0)  // false
```

- **indexOf**

```
numbers.push('a', 'a')
numbers                 // [1,2,3,4,'a','a']
numbers.indexOf('a')  // 4
numbers.indexOf('b')  // -1
```

- **join**

```
numbers.join()     // '1,2,3,4,a,a'
numbers.join('')   // '1234aa'
numbers.join('-')  // '1-2-3-4-a-a'
```

5.2 Object (객체/오브젝트)

- 선언

```
const me = {
  name: '홍길동',  // key가 한 단어일 때
  'phone number': '01012345678',  // key가 여러 단어일 때
  appleProducts: {
    ipad: '2018pro',
    iphone: '7+',
    macbook: '2019pro',
  },
}
```

JavaScript 기초문법

- **요소 접근**
  - Key를 식별자로 활용할 수 없는 경우 반드시 [ ]로 접근해야 한다.

```
me.name      // 홍길동
me['name']   // 홍길동
me['phone number']      // '01012345678'
me.appleProducts        // { ipad: '2018pro', ... }
me.appleProducts.ipad   // '2018pro'
```

- **Object 축약 문법**
  - 객체를 정의할 때 key와 할당하는 변수의 이름이 같으면 아래와 같이 축약이 가능하다.

```
const name = '김싸피'
const score = '80'

const student = {
  // name: name,
  // score: score,
  name,
  score,
}

console.log(student) // { name: '김싸피', score: '80' }
```

JavaScript 기초문법

5.3 Array Helper Method (ES6+ Features)

- **forEach**
  - 주어진 callback 함수를 배열의 각 요소에 대해 한번씩 실행한다.
  - 문법

```
arr.forEach(callback(element, index, array))
```

  - 사용 예시

```
const colors = ['red', 'blue', 'green']

colors.forEach(function (color) {
  console.log(color)  // 'red', 'blue', 'green'
})
```

- **map**
  - 배열 내 모든 요소에 대해 주어진 callback 함수를 실행하며, 함수의 반환값을 요소로 하는 새로운 배열 반환한다. 배열을 다른 모습으로 바꿀 때 사용한다.
  - 문법

```
arr.map(callback(element, index, array))
```

  - 사용 예시

```
const nums = [1, 2, 3]

const doubleNums = nums.map(function (num) {
  return num * 2
})
console.log(doubleNums)  // [2, 4, 6]
```

JavaScript 기초문법

- **filter**
  - 주어진 callback 함수의 테스트를 만족하는 요소만으로 만든 새로운 배열을 반환한다. callback 함수를 통해 원하는 요소만 추릴 수 있다.
  - 문법

```
arr.filter(callback(element, index, array))
```

  - 사용 예시

```
const products = [
  { name: 'cucumber', type: 'vegetable' },
  { name: 'banana', type: 'fruit' },
  { name: 'carrot', type: 'vegetable' },
  { name: 'apple', type: 'fruit' },
]

const fruits = products.filter(function (product) {
  return product.type === 'fruit'
})
console.log(fruits)  // ['banana', 'apple']
```

- **find**
  - 주어진 callback 함수의 테스트를 만족하는 첫번째 요소를 반환한다. 값이 없다면 undefined를 반환한다.
  - 문법

```
arr.find(callback(element, index, array))
```

JavaScript 기초문법

- 사용 예시

```
const avengers = [
  { name: 'Tony Stark', age: 45 },
  { name: 'Steve Rogers', age: 32 },
  { name: 'Thor', age: 40 },
]

const avenger = avengers.find(function (avenger) {
  return avenger.name === 'Tony Stark'
})
console.log(avenger)  // {name: 'Tony Stark', age: 45}
```

- **some**
  - 배열 안의 하나의 요소라도 callback 함수의 테스트를 만족하면 `true`를 반환, 아닐 경우 `false`를 반환한다. 단, 빈 배열에서 호출 시 `false`를 반환한다.
  - 문법

```
arr.some(callback(element, index, array))
```

  - 사용 예시

```
const requests = [
  { url: '/photos', status: 'complete' },
  { url: '/albums', status: 'pending' },
  { url: '/users', status: 'failed' },
]

const inProgress = requests.some(function (request) {
    return request.status === 'pending'
})
console.log(inProgress)  // true
```

JavaScript 기초문법

- **every**
  - 배열 안의 모든 요소가 callback 함수의 테스트를 만족하면 `true`를 반환, 아닐 경우 `false`를 반환한다. 단, 빈 배열에서 호출 시 `true`를 반환한다.
  - 문법

```
arr.every(callback(element, index, array))
```

  - 사용 예시

```
const users = [
  { id: 21, submmited: true },
  { id: 33, submmited: false },
  { id: 712, submmited: true},
]

const hasSubmitted = users.every(function (user) {
  return user.submitted
})
console.log(hasSubmitted)  // false
```

- **reduce**
  - 배열의 각 요소에 대해 주어진 callback 함수를 실행하고, 하나의 결과 값을 반환한다. reduce는 배열 내의 숫자 총합, 평균 계산 등 배열의 값을 하나로 줄이는 동작을 한다.
  - 문법

```
arr.reduce(callback(acc, element, index, array), initialValue)
```

    - callback 함수의 첫번째 매개변수(acc)는 누적 값(전 단계의 결과)이다.
    - `initialValue`는 반환할 누적 값의 초기 값이다. (생략 시 첫번째 요소가 누적 값이 된다.)
    - callback 함수에서 반환하는 값이 누적 값이 된다.

JavaScript 기초문법

- 사용 예시

```
const scores = [90, 90, 80, 77]

const totalScore = scores.reduce(function (sum, score) {
  return sum + score
}, 0)  // 0 생략 가능 => 첫번째 아이템이 누적값(total)이 된다.
console.log(totalScore)  // 337
```