**2장. 알고 있으면 유용한 자바스크립트 문법**

**<<01. 삼항 연산자>>**

let text = array.length === 0 ? '배열이 비어있습니다' : '배열이 비어있지 않습니다.';

- 조건 ? true일 때 : false 일 때

**<<02. Truthy and Falsy>>**

- Truthy: 1, 유효한 값

- Falsy: 0, null, undefined, empty

function print(person) {

if (!person) { //person으로 넣으면 에러가 뜸.

console.log('person이 없네요');

return;

}

console.log(person.name);

}

const person = null;

print(person);

- undefined 와 null 은 Falsy 한 값입니다. Falsy 한 값 앞에 느낌표를 붙여주면 true로 전환됩니다.

console.log(!undefined);

console.log(!null);

const value = { a: 1 };

if (value) {

console.log('value 가 Truthy 하네요.');

}

- value 가 Truthy 한 값이기 때문에, 콘솔에 메시지가 출력 될 것입니다. 반면, value 가 null, undefined, 0, '', NaN 중 하나라면, 나타나지 않을 것입니다.

Truthy한 값과 Falsy한 값을 잘 알아놓으면 조건문을 작성할 때 편할 것임

**\* 특정 값이 Truthy 한 값이라면 true, 그렇지 않다면 false 로 값을 표현**

const value = { a: 1 };

const truthy = value ? true : false;

const value = { a: 1 };

const truthy = !!value;

- 이전에 배운 삼항연산자를 사용하면 쉽게 value 값의 존재 유무에 따라 쉽게 true 및 false로 전환이 가능함.

**<<03. 단축 평가(short-circuit evaluation) 논리 계산법**

**\* 논리 연산자를 유용하게 사용하는 방법**

\* animal 값이 제대로 주어졌을 때만 name을 조회하고, 그렇지 않을 때는 그냥 undefined를 반환하게 하고 싶을 때

const dog = {

name: '멍멍이'

};

function getName(animal) {

if (animal) {

return animal.name;

}

return undefined; //이렇게 처리 안 하면 에러 뜸

}

const name = getName();

console.log(name);

**\* && 연산자로 코드 단축시키기**

- Falsy한 값을 만나면 그것을 console에 찍어줌

- 계속 Truthy한 값을 만나면 마지막 값을 찍어줌

const dog = {

name: '멍멍이'

};

function getName(animal) {

return animal && animal.name;

}

const name = getName(dog);

console.log(name); // 멍멍이

console.log(true && 'hello'); // hello

console.log(false && 'hello'); // false

console.log('hello' && 'bye'); // bye

console.log(null && 'hello'); // null

console.log(undefined && 'hello'); // undefined

console.log('' && 'hello'); // ‘’

console.log(0 && 'hello'); // 0

console.log(1 && 'hello'); // hello

console.log(1 && 1); // 1

**\* || 연산자로 코드 단축시키기**

- 어떤 값이 Falsy 하다면 대체로 사용할 값을 지정해줄 때 매우 유용하게 사용할 수 있음

- 둘 중 Truthy한(유효한) 값을 반환함

const namelessDog = {

name: ''

};

function getName(animal) {

const name = animal && animal.name;

return name || '이름이 없는 동물입니다.';

}

const name = getName(namelessDog);

console.log(name); // 이름이 없는 동물입니다.

**<<04. 함수의 기본 파라미터>>**

**\* NaN:** Not a Number라는 의미로, 숫자가 아닌 값을 연산해서 이상한 결과가 나옴

**\* ES6 이전**

function calculateCircleArea(r) {

const radius = r || 1;

return Math.PI \* radius \* radius;

}

const area = calculateCircleArea();

console.log(area); // 3.141592653589793

**\* ES6**

function calculateCircleArea(r = 1) {

return Math.PI \* r \* r;

}

const area = calculateCircleArea();

console.log(area); // 3.141592653589793

**<<05. 조건문 더 스마트하게 쓰기>>**

**\* 특정 값이 여러 값 중 하나인지 확인해야 할 때**

- 배열을 만들고 배열의 includes 함수를 사용

function isAnimal(name) {

const animals = ['고양이', '개', '거북이', '너구리'];

return animals.includes(name);

}

console.log(isAnimal('개')); // true

console.log(isAnimal('노트북')); // false

- animals배열 생략, 화살표 함수로 작성

const isAnimal = name => ['고양이', '개', '거북이', '너구리'].includes(name);

console.log(isAnimal('개')); // true

console.log(isAnimal('노트북')); // false

**\* 값에 따라 다른 결과물을 반환해야 할 때**

function getSound(animal) {

const sounds = {

개: '멍멍!',

고양이: '야옹~',

참새: '짹짹',

비둘기: '구구 구 구'

};

return sounds[animal] || '...?';

}

console.log(getSound('개')); // 멍멍!

console.log(getSound('비둘기')); // 구구 구 구

**\* 값에 따라 실행해야 하는 코드 구문이 다를 때**

- 객체에 함수를 넣으면 됨

function makeSound(animal) {

const tasks = {

개() {

console.log('멍멍');

},

고양이() {

console.log('고양이');

},

비둘기() {

console.log('구구 구 구');

}

};

if (!tasks[animal]) {

console.log('...?');

return;

}

tasks[animal]();

}

getSound('개');

getSound('비둘기');

**<<06. 비구조화 할당(구조 분해) 문법>>**

**\*복습**

const object = { a: 1, b: 2 };

const { a, b } = object;

console.log(a); // 1

console.log(b); // 2

**\* 복습:** 함수의 파라미터에서 비구조화 할당 사용

const object = { a: 1, b: 2 };

function print({ a, b }) {

console.log(a);

console.log(b);

}

print(object);

- 만약 b의 값이 주어지지 않았다면

const object = { a: 1 };

function print({ a, b }) {

console.log(a);

console.log(b);

}

print(object);

// 1

// undefined

**\* 비구조화 할당시 기본값 설정**

const object = { a: 1 };

const { a, b = 2 } = object;

console.log(a); // 1

console.log(b); // 2

**\* 비구조화 할당시 이름 바꾸기**

- animal 객체 안에 있는 name을 nickname이라고 선언하겠음

const animal = {

name: '멍멍이',

type: '개'

};

const { name: nickname } = animal

console.log(nickname);

**\* 배열 비구조화 할당**

const array = [1, 2];

const [one, two] = array;

console.log(one);

console.log(two);

- 기본값 지정 가능

const array = [1];

const [one, two = 2] = array;

console.log(one);

console.log(two);

**\* 깊은 값 비구조화 할당**

const deepObject = {

state: {

information: {

name: 'velopert',

languages: ['korean', 'english', 'chinese']

}

},

value: 5

};

const {

state: {

information: { name, languages }

},

value

} = deepObject;

const extracted = {

name,

languages,

value

};

console.log(extracted);

**<<07.spread와 rest>>**

**\* spread와 rest 문법:** ES6에 도입됨

**\* spread:** 객체 또는 배열을 펼칠 수 있음 (+ 개념)

- 기존 객체를 건들이지 않고 추가하여 새로운 객체를 만듦

const slime = {

name: '슬라임'

};

const cuteSlime = {

name: '슬라임',

attribute: 'cute'

};

const purpleCuteSlime = {

name: '슬라임',

attribute: 'cute',

color: 'purple'

};

console.log(slime);

console.log(cuteSlime);

console.log(purpleCuteSlime);

- spread 문법 사용

const slime = {

name: '슬라임'

};

const cuteSlime = {

...slime, //…문자가 바로 spread 문법

attribute: 'cute'

};

const purpleCuteSlime = {

...cuteSlime,

color: 'purple'

};

console.log(slime);

console.log(cuteSlime);

console.log(purpleCuteSlime);

- 배열에서 spread 문법 사용

const animals = ['개', '고양이', '참새'];

const anotherAnimals = [...animals, '비둘기'];

console.log(animals);

console.log(anotherAnimals);

- 배열에서 spread 문법 여러 번 사용

const numbers = [1, 2, 3, 4, 5];

const spreadNumbers = [...numbers, 1000, ...numbers];

console.log(spreadNumbers); // [1, 2, 3, 4, 5, 1000, 1, 2, 3, 4, 5]

**\* rest 문법:** 객체, 배열, 그리고 함수의 파라미터에서 사용 가능(- 개념)

const purpleCuteSlime = {

name: '슬라임',

attribute: 'cute',

color: 'purple'

};

const { color, ...rest } = purpleCuteSlime; //color를 제외한 속성이 들어감

console.log(color); //purple

console.log(rest); //슬라임, cute

- attribute까지 없앤 새로운 객체를 만들고 싶다면

const purpleCuteSlime = {

name: '슬라임',

attribute: 'cute',

color: 'purple'

};

const { color, ...cuteSlime } = purpleCuteSlime; //purpleCuteSlime 에서 color 속성을 빼고 cuteslime 만듦

console.log(color); //purple

console.log(cuteSlime); //슬라임, cute

const { attribute, ...slime } = cuteSlime; //cuteslime에서 attribute 속성 빼고 slime 만듦

console.log(attribute); //attribute

console.log(slime); //슬라임

- 배열에서의 rest

const numbers = [0, 1, 2, 3, 4, 5, 6];

const [one, ...rest] = numbers;

console.log(one); //0

console.log(rest); //[1, 2, 3, 4, 5, 6]

- 하지만 spread처럼 뒤에서는 못함.

const numbers = [0, 1, 2, 3, 4, 5, 6];

const [..rest, last] = numbers; //에러

- 함수 파라미터에서의 rest

function sum(...rest) {

return rest;

}

const result = sum(1, 2, 3, 4, 5, 6);

console.log(result); //[1, 2, 3, 4, 5, 6]

function sum(...rest) {

return rest.reduce((acc, current) => acc + current, 0);

}

const result = sum(1, 2, 3, 4, 5, 6);

console.log(result); // 21

**\* 함수 인자와 spread**

- 함수에서 값을 읽을 때, 그 값들을 파라미터라고 함

- 함수에서 값을 넣어줄 때, 그 값들을 인자라고 함

const myFunction(a) { // 여기서 a 는 파라미터

console.log(a); // 여기서 a 는 인자

}

myFunction('hello world'); // 여기서 'hello world' 는 인자

function sum(...rest) {

return rest.reduce((acc, current) => acc + current, 0);

}

const numbers = [1, 2, 3, 4, 5, 6];

const result = sum(...numbers);

console.log(result);

**\* 퀴즈**

- 에 n 개의 숫자들이 파라미터로 주어졌을 때, 그 중 가장 큰 값을 알아내세요.

function max() {

return 0;

}

const result = max(1, 2, 3, 4, 10, 5, 6, 7);

console.log(result);

<script>

      function max(...array) {

        let cnt = 0;

*//if를 쓰려면 함수 형식으로 처리해야 하는데, 그렇게 되면 변수 num이 계속 undefined로 초기화 돼서 cnt 추가 시킴*

*//앞으로는 강사님 소스처럼 함수 쓰지 않고 처리*

        array.reduce((num, current) => {

          if (current > cnt) {

            cnt = current;

          }

        }, 0);

        return cnt;

*//강사님 소스*

*/\**

*return array.reduce(*

*(acc, current) => (current > acc ? current : acc),*

*array[0]*

*);*

*\*/*

      }

      const result = max(1, 2, 3, 4, 10, 5, 6, 7);

      console.log(result);

    </script>

**<<08. 자바스크립트의 Scope에 대한 이해>>**

**\* Scope:** 변수 혹은 함수를 선언하게 될 때 해당 변수 또는 함수가 유효한 범위를 의미함

**\* Scope의 종류:**

- Global(전역) Scope: 코드의 모든 범위에서 사용이 가능함

- Function(함수) Scope: 함수 안에서만 사용이 가능함

- Block(블록) Scope: if, for, switch 등 특정 블록 내부에서만 사용이 가능함

**\* 예시를 통한 Scope 이해**

const value = 'hello!';

function myFunction() {

console.log('myFunction: '); //1

console.log(value); //2. hello!

}

function otherFunction() {

console.log('otherFunction: ');//3

const value = 'bye!';

console.log(value); //4.bye!

}

myFunction();

otherFunction();

console.log('global scope: '); //5.

console.log(value); //6. Hello!

const value = 'hello!';

function myFunction() {

const value = 'bye!';

if (true) {

const value = 'world';

console.log('block scope: '); //1.

console.log(value); //2. world

}

console.log('function scope: '); //3.

console.log(value); //4. Bye!

}

myFunction();

console.log('global scope: '); //5.

console.log(value); //6.hello!

- 하지만 var는 Function Scope로 선언 되므로, if문 블록 내부에서 선언한 value 값이 블록 밖의 value에도 영향을 미치게 됨

var value = 'hello!';

function myFunction() {

var value = 'bye!';

if (true) {

var value = 'world';

console.log('block scope: '); //1.

console.log(value); //2. world

}

console.log('function scope: '); //3.

console.log(value); //4. world

}

myFunction();

console.log('global scope: '); //5.

console.log(value); //6. hello!

**\* Hoisting:**

- 자바스크립트에서 아직 선언되지 않은 함수/변수를 “끌어올려서” 사용할 수 있는 자바스크립트의 작동 방식임

function myFunction() {

console.log('hello world!');

}

myFunction(); //hello world!

console.log(number); //에러

- var 사용하여 Hoisting

console.log(number); //undefined

var number = 2;

- const, let 사용하여 Hoisting

function fn() {

console.log(a);

let a = 2;

}

fn(); //에러

\* 나중에 개발을 본격적으로 하게 될 때에는 ESLint 라는 것을 사용하여 Hoisting이 발생하는 코드를 에디터상에서 쉽게 발견할 수 있음