Javascript 문법

1. Var와 let의 차이

Var: 중복 선언 가능

Var로 선언한 변수는 동일한 이름으로 여러 번 중복해서 선언이 가능하다.

이와 같은 경우, 마지막에 할당된 값이 변수에 저장된다.

예시)

Var name = ‘jiwoo’;

Console.log(name); // jiwoo

Var name = ‘kimjiwoo’;

Console.log(name); // kimjiwoo

이는 필요할 때마다 변수를 유연하게 사용할 수 있다는 장점이 될 수도 있지만, 기존에 선언해둔

변수의 존재를 잊고 값을 재할당하는 등의 실수가 발생할 가능성이 크다.

Let: 중복 선언 불가능, 재할당 가능

Var와 다르게 let은 중복 선언을 할 경우 해당 변수가 이미 선언되었다는 에러 메시지가 출력된다.

예시)

let name = ‘jiwoo’;

Console.log(name); // Jiwoo;

let name = ‘kimjiwoo’;

Console.log(name);

// Uncaught SyntaxError: Identifier ‘name’ has already been declared

변수 선언 및 초기화 이후 반복해서 다른 값을 재할당 할 수는 있다.

name = ‘kim’;

Console.log(name); // kim

1. Const

Const: 중복 선언 불가능, 재할당 불가능

Let과 const의 차이점은 immutable의 여부이다. Let은 변수에 다른 값을 재할당할 수 있지만,

Const는 재할당 시 에러 메시지가 출력된다.

예시)

const name = ‘jiwoo’;

Console.log(name); // Jiwoo;

const name = ‘kimjiwoo’;

Console.log(name);

// Uncaught SyntaxError: Identifier ‘name’ has already been declared

name = ‘kim’;

Console.log(name);

// Uncaught TypeError: Assignment to constant variable

이처럼 const는 상수를 뜻하기 때문에 한 번만 선언이 가능하며 값을 바꿀 수도 없다.

1. ES5의 Array 관련 메소드

Var testArray = [ “a”, “b”, “c”, “d” ]; 해당 배열을 가지고 예시 코드를 설명하겠다.

1. Array.forEach

배열 전체를 돌며 해당 배열의 요소에 직접 어떤 작업을 수행하고 싶을 때

“배열 자체를 변경”

testArray.forEach( function (item, index, array){

array[index] = item+"\_test";

});

//testArray 출력시 : ["a\_test", "b\_test", "c\_test", "d\_test"]

1. Array.map

배열 전체를 돌며 해당 요소를 사용하여 “다른 새로운 배열”을 만들고 싶을 때

//배열의 모든 요소에 "\_new"를 덧붙인 새로운 배열 생성

var newArray = testArray.map(function (item, index, array) {

return item + "\_new";

});

//newArray 출력시 : ["a\_new", "b\_new", "c\_new", "d\_new

1. Array.some

배열 요소 중 하나라도 특정 조건을 만족하는지 알고 싶을 때

콜백 수행 중 true값을 return하면 바로 중단되며 true를 return한다.

True가 한번도 return 되지 않은 채 끝나면 최종적으로 false를 return한다.

var isTrue = testArray.some(function (item, index, array) {

console.log(index + "번째 요소 : " + item);

return item.search("b")>-1;

});

//출력

//0번째 요소 a

//1번째 요소 b

//isTrue = true

1. Array.every (some과 반대)

배열의 모든 요소가 특정 조건을 만족하는지 알고 싶을 때

콜백 수행 중 한번이라도 false값을 return하면 중단되고 최종적으로 false를 return한다.

False를 한번도 return하지 않으면 최종적으로 true를 return한다.

let isTrue = testArray.every(function (item, index, array) {

console.log(index + "번째 요소 : " + item);

return item.search("a")>-1;

});

//출력

//0번째 요소 : a

//1번째 요소 : b

//isTrue = false

1. Array.filter

배열에서 특정 케이스만 필터링해서 새로운 배열을 만들고 싶을 때

콜백에서 return된 요소로 새로운 배열을 return한다.

//배열의 요소가 "a" 또는 "b"로 이루어져 있으면 추출

var newArray = testArray.filter(function (item, index, array) {

return item.search(/[ab]+/)>-1;

});

//newArray 출력 : ["a", "b"]

1. Array.find

Filter와 비슷하지만 단 하나의 요소만 리턴

콜백의 return이 true인 요소 찾을 때까지 순회

True가 return되면 종료, 모두 false인 경우 최종적으로 undefined를 return

// 정수 배열에서 5의 배수인 정수만 모으기

var arr = [4, 15, 377, 395, 400, 1024, 3000];

var count = 0;

var found = arr.find(function (n) {

count++;

return n % 5 == 0;

});

console.log(count); //출력 2

console.log(found); //출력 15

1. Array.reduce

배열의 요소들을 하나씩 돌며, 이전 콜백의 return 값을 넘겨받아 작업을 수행하고 싶을 때

Promise를 이용해 비동기 작업 순차 실행에 유용

//1.베이직 예제 - initialValue 생략한 경우

var arr = [9, 2, 8, 5, 7];

var count = 0;

var sum = arr.reduce(function (pre, value) {

count++;

return pre + value;

});

console.log(sum); //출력 결과 : 31

console.log(count); //출력 결과 : 4

//2. 베이직 예제 - initialValue있는 경우

var count = 0;

var sum = arr.reduce(function (pre, value) {

count++;

return pre + value;

}, 0);

console.log(sum); // 출력 : 31

console.log(count); // 출력 : 5

1. Rest parameter

나머지 매개변수는 매개변수 이름 앞에 세 개의 점(…)을 붙여서 정의한 매개변수를 의미한다.

나머지 매개변수는 함수에 전달된 인수들의 목록을 배열로 전달받는다.

Function foo( …rest) {

Console.log(Array.isArray(rest)); // true

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

};

Foo(1, 2, 3, 4, 5);

Arguments 객체는 함수 호출 시 전달된 인수들의 정보를 담고 있는 순회가능한 유사 배열 객체이며

함수 내부에서 지역 변수처럼 사용할 수 있다.

Var foo = function () {

Console.log(arguments);

};

Foo(1, 2); // { ‘0’: 1, ‘1’:2 }

Rest 파라미터를 사용하여 가변 인자의 목록을 배열로 전달받을 수 있다. 이를 통해 유사 배열인

Arguments 객체를 배열로 변환하는 번거로움을 피할 수 있다.

Functon sum() {

Var array = Array.prototype.slice.call(arguments);

Return array.reduce(function (pre, cur) {

Return pre + cur;

});

}

Conslole.log(sum(1, 2, 3, 4, 5)); // 15

하지만 화살표 함수에는 함수 객체의 arguments 프로퍼티가 없다. 따라서 화살표 함수로 가변 인자

함수를 구현할 때는 반드시 rest 파라미터를 사용해야 한다.

Function sum(…args) {

Console.log(arguments);

Console.log(Array.isArray(args));

Return args.reduce((pre, cur) => pre + cur);

}

Console.log(sum(1, 2, 3, 4, 5));

1. Arrow function

화살표 함수 표현은 전통적인 함수표현의 간편한 대안이다. 하지만, 화살표 함수는 몇 가지 제한점이 있고 모든 상황에 사용할 수는 없다.

This 나 super에 대한 바인딩이 없고, methos로 사용될 수 없다.

New.target 키워드가 없다.

일반적으로 스코프를 지정할 때 사용하는 call, apply, bind, methods를 이용할 수 없다.

생성자로 사용할 수 없다.

Yield를 화살표 함수 내부에서 사용할 수 없다.

1. 1급 객체

일급객체란 다른 객체들에 일반적으로 적용 가능한 연산을 모두 지원하는 객체를 가리킨다.

일급객체의 조건은 변수에 할당 할 수 있어야 하고, 다른 함수를 인자로 전달 받을 수 있어야 하며

다른 함수의 결과로서 리턴될 수 있어야 한다.

이로 인해 함수를 데이터 다루 듯이 다룰 수 있다는 것을 알 수 있다.

함수가 일급객체이면 고차함수를 만들 수 있고 콜백을 사용할 수 있다.

1. Object의 개념

자바스크립트의 기본 데이터 타입은 객체(object)이다.

객체란 이름(name)과 값(value)으로 구성된 프로퍼티(property)의 정렬되지 않은 집합이다.

프로퍼티의 값으로 함수가 올 수도 있는데, 이러한 프로퍼티를 메소드(method)라고 한다.

자바스크립트에서는 숫자, 문자열, 불리언, undefined 타입을 제외한 모든 것이 객체이다.

하지만 숫자, 문자열, 불리언과 같은 원시 타입은 값이 정해진 객체로 취급되어,

객체로서의 특징도 함께 가진다.

자바스크립트에서 객체의 프로퍼티를 참조하는 방법은 다음과 같다.

객체이름.프로퍼티이름

또는

객체이름[“프로퍼티이름”]

Var person = {

Name: “김지우”, //이름 프로퍼티 정의

Birthday: “970823”, //생년월일 프로퍼티 정의

Id: “1234567”, //개인 id 프로퍼티 정의

Fullid: function() { //생년월일과 개인 id를 합쳐서 주민등록번호를 반환

Return this.birthday + this.id;

}

};

Person.name // 김지우

Person[“name”] // 김지우