코딩앙마 – JS 중급

1. 변수, 호이스팅 TDZ

|  |  |  |  |
| --- | --- | --- | --- |
|  | var | let | const |
| scope | only function | func, if, while, for, try 등등 {}가 있는 모든 곳 | |
| hoisting | 선언+초기화까지 호이스팅 | 선언만 호이스팅 | 얘는 무조건 선언, 초기화, 할당이 동시에 일어나야댐 |
|  | 즉 if{} 문 안에서 선언한 친구도 밖에서 쓸 수 있음 |  |  |
|  | 선언을 나중에해도 호이스팅 되서 오류가 안나고 undefined가 뜸 |  |  |

Temporal Dead Zone: 스코프 내에서 호이스팅으로 선언은 되어 있지만, 할당하기 전에는 사용할 수 없는 영역. 아래쪽에는 호이스팅을 위한 선언 코드가 있을 것이다.

2. 생성자 함수

함수인데 안에 변수와 메쏘드를 담고 있음. 관용적으로 함수이름 첫 문자는 대문자로 해준다. 그리고 인스턴스를 만들 때는 new를 붙여준다. 그럼 this = {} 라는 객체 생성 후 이 객체 안에 변수와 메쏘드를 넣어주고, return this 하는 두 과정이 생략되어 있다.

3. Computed Property,

객체의 변수 명이 고정 된 값이 아니라, 생성 시에 정하고 싶을 때 [ ] 직각 가로를 이용해서 변수 명을 사용하게 한다.

그리고 객체 사용시 같이 사용 가능한 함수로 Object.assign( { 합칠 함수 }, 추가할 함수 )

Object.keys( 객체 ), Object.values( 객체 ), Object.entries( 객체 ), Object.fromEntries( 객체 )가 있다.

4. const a = Symbol(); // new 를 붙이지 않는다.

심볼은 유일한 식별자를 만들 때 쓰임. Symbol( 설명 전달 가능 ), 설명 란에 포함 된 문자열은 잘 작성해 놓으면 디버깅 시에 편리함. 설명이 똑같아도 되고 아무 문자열이나 포함 가능. 심볼을 객체 안에 넣으면 Object.keys or 다른 함수들 or for문 등에서 검색이 안됨! 나만 쓰기 위해서 만드는 듯.

5. Math 관련 함수!

숫자.toString( 몇 진수로 바꿀 것인지 ) => 해당 진수로 바꾼 str 값을 리턴한다.

숫자.toFixed( 소수점 아래 몇 번째 자리 수 까지 남길 것 인지 ) \*문자열을 반환함

Math 내장 객체

Math.PI = 원주율

Math.ceil( 수 ) / Math.floor( 수 ) / Math.round( 수 )

Math.random() : 0~1 사이의 임의 숫자를 반환한다.

1~100 사이의 임의 정수를 반환하는 법 : Math.floor(Math.random() \* 100) + 1

Math.abs() / Math.pow(n, m) : n의 m승 / Math.sqrt(16) : 4 /

Math.max( ) / Math.min( ) : ( ) 안에 배열이 못들어가네!?! 그럼 배열의 최대 최소는 어떡하지 ㅜ

NaN은 NaN == NaN 과 같이 등호로는 판별 불가능! isNaN( 비교할 값 ) 이라는 함수를 사용해야 함.

parseInt() 는 Number() 와 다르다!

Number()는 문자열이 껴 있으면 NaN을 반환하지만, ParseInt()는 문자열이 있어도 읽을 수 있는 데까지 (숫자 영역까지는) 읽고 문자가 나오면 읽었던 숫자 부분을 리턴한다.

또한 ParseInt( 바꿀 수 , 두번째 인자 ) 두번째 인자로 몇 진수로 읽을 것인지도 정할 수 있음

ex) ParseInt( ‘11’, 2 ) => 3 임 (2진수로 읽었으니까)

parseFloat() 위랑 똑같지만 소수점까지 반환

6. 문자열 메소드

‘’, “” 는 큰 차이가 없다. 다만 html 코드는 “” 이 많이 쓰이니까 ‘’로 감싸는게 좋고,

영어 문장은 ‘를 많이 쓰니까 “”로 감싸는게 좋다.

백틱(``)은 엔터까지 입력이 가능해서 좋다.

문자열.length : 배열처럼 길이 알 수 있음.

문자열[ index ] 로 인덱싱도 가능

문자열.slice(n, m) 으로 슬라이싱도 가능

문자열.substr(n, m) : n부터 시작해서 m개의 문자를 가져옴

문자열.trim() : 문자열 앞 뒤의 공백을 모두 제거

문자열.repeat(n) : 문자열을 n번 반복함!

문자열.includes(‘Hi’) : ‘Hi’가 포함 되어 있으면 true, 아니면 false 리턴

문자열.toUpperCase() & .toLowerCase() 로 모든 문자를 대문자 & 소문자로 변형 가능

문자열.indexOf(‘to’) : ‘to’가 문자 열 내에 존재하면 시작 index를 반환하고 없으면 -1을 반환

따라서 if 문을 쓸 때 보통 이렇게 씀

if ( desc.indexOf(‘Hi’) > -1) {

7. 배열 메소드

배열.push() : 뒤에 삽입

배열.pop() : 뒤에 삭제

배열.unshift() : 앞에 삽입

배열.shift() : 앞에 삭제

배열.splice(n, m, x) : n번째 index부터 m개의 요소를 지워라 그리고 지운 후 x 부터 나오는 요소들을 집어 넣어라. 만약 m에 0을 넣으면 아무 것도 지우지 않고 x 부터 값만 넣어라. splic는 삭제된 요소의 배열을 반환한다.

배열.slice(n, m) : 문자열처럼 n부터 m까지 slicing 해서 반환. 만약 아무 값도 안 넣으면 배열이 복사되어 반환된다.

배열.concat(arr2, arr3, …) : concat은 배열과 가로 안에 배열 들을 합쳐서 하나의 배열로 반환한다.

let arr = [1, 2];

arr.concat([3,4], 5, 6) 이렇게 해도 [1, 2, 3, 4, 5, 6]이 반환되는 것은 동일하다.

배열은 for문이나 for ( of ) 문으로 반복했는데, forEach(fn) 을 이용해서도 가능!

arr.forEach ( ( item, index, arr ) => { } )

arr.indexOf( 3 ) : 3이 배열 내에 존재하면 해당 index를 리턴

arr.indexOf( 3, 4 ) : 3이 배열 내에 4번째 index 이후에 존재하면 해당 index를 리턴

arr.lastIndexOf(3) : 끝에서 가장 가까운 3의 index를 리턴

arr.includes(2) : 포함하고 있는지만 알고 있으면 이게 나음 true, false 를 리턴하니까

arr.find( func ) : func 의 리턴 값이 트루일 때 멈추고 func 이 트루가 되도록 했던 인풋 값을 리턴

* 막 객체가 들어 있는 배열 같은 경우는 indexOf 를 쓰기 힘드니까 그럴 때 씀
* 근데 하나 찾으면 멈추는게 좀….

그래서 딱 하나만 찾는게 아니면 filter를 쓰면 됨!

arr.filter( ( item ) => { if ~~ return true else false } )

arr.map ( func ) : 함수를 받아 특정 로직을 실행하고 새로운 배열을 리턴

arr.reverse() : 역순으로 재정렬

arr.join( 기호 ) : 배열을 문자열로 합쳐주는데 중간에 기호를 넣어 줌

arr.split( 기호 ) : 문자열을 기호를 기준으로 나눠서 배열로 만들어 줌 / 기호 없으면 다 잘라줌

Array.isArray( arr ) 넣으면 arr가 배열일 경우 true, 아닐 경우 false 를 리턴!

* 보통 typeof 를 쓰는데 배열과 객체 모두 Object type으로 인식되서 구별할 때 씀!

arr.sort() : 파이썬 처럼, 정렬된 배열이 리턴 되는게 아니라 배열 자체가 정렬 되버린단다!

* sort는 메모리의 배열 구조를 바꾸기 때문인건가!?

근데 문제는 모두 str 으로 인식해서 정렬해 버리기 때문에, 숫자를 정렬하고 싶은 경우 인자로 정렬하는 함수를 전달해야 함.

>> 배열 sort 관련해서는 더 공부해야할 듯 lodash + 유툽 !

arr.reduce( (prev, cur) => {}, 초기값 ) : 배열을 돌면서 원하는 작업을 해주고 최종 값을 반환!

함수의 첫 인자로 이전 값 prev를 받고, 두번째 인자로 현재 값 cur 를 받는다!

초기값은 안쓰면 배열의 첫번째 값이 들어간다.

요주의 메소드! filter, map, sort, reduce, forEach 중 뭐를 쓸지 구별하는 것이 실력

8. 구조 분해 할당

let [a, b, c] = [1, 2, 3] 이런건데! 내가 원했던거 ㅋㅋㅋㅋ 근데 문제는 개수가 안맞으면 undefined가 발생할 수 있어서 기본 값을 미리 세팅해주는 것도 좋은 방법임!

let [a=3, b=4, c=5] = [1, 2] 이렇게 하면 c 에 undefined가 들어가는 것을 방지할 수 있음

공백과 쉼표로 할당하지 않을 변수를 생략할 수도 있음

ex) let [user1, , user3] = [1, 2, 3] 이런 식으로하면 2는 할당x

값 바꿔치기할 때 편함! [a,b] = [b,a] 이런식으로!

객체도 구조 분해 할당이 가능하다

let user = { name : ‘woongil’, age : 30 }

let {name, age} = user 이런식으로! 사실 순서가 바껴도 됨 let {age, name} = user 이렇게 써도 key값과 객체 내부 값이 같으면 알아서 매칭함! 근데 만약 새로 배정하는 변수 이름을 바꾸고 싶다면 let {name : userName, age : userAge} = user 이런식으로 하면 매칭은 잘 되는데 userName, userAge 변수로 값을 사용가능!

9. Rest parameters, Spread syntax

JS는 꼭 변수 개수와 사전 함수에 정의된 변수의 개수가 같아야 하는 것은 아니다. 달라도 오류가 안난다. 다만 누락되는 값이 있을 뿐이지. 그래서 변수가 몇 개가 들어올 지 모르는 상황에서 쓰는 것이 arguments와 Rest parameters 인데, Rest parameters가 권장된다.

… 세개 찍고 name 변수이름 적으면 댐 function showName( …Name ) { }

Spread syntax : 전개 구문은 원래 배열 안에 있는 값을 빼서 원하는 위치에 넣는 작업이 귀찮은 작업인데 (push, splice, concat 등등) 그걸 한 번에 가능하게 함.

let arr1 = [1, 2, 3]

let arr2 = [4, 5, 6]

let result = [0, …arr1, …arr2, 99, 100] 하면 하나의 list 완성!

객체도 가능하고 매우 간편해서 Object.assign( ) 쓸 필요가 없어짐!

let user = {name : ‘Mike’, age: 30};

let user2 = {…user};

10. 함수 실행을 시간으로 제어하는 setTimeout, clearTimeout, setInterval, clearInterval

리턴값 = setTimeout( 실행할 함수, 1000ms (실행하는데 걸릴 시간), 함수의 변수로 넘겨 줄 초기값 )

clearTimeout(리턴값) 하면 위의 함수 실행을 없앤다.

리턴 값 = setInterval ( 실행할 함수, ms 걸릴 시간, 초기값 ) : 얘는 시간 단위로 계속해서 호출함 무한 반복

clearInterval( 리턴 값 ) : 위의 반복을 멈추게 됨

1. call, apply, bind 함수

함수.call( 객체 ) 하면 함수의 this가 객체를 가르키게 된다.

함수.call( 객체, 매개변수 ) 이렇게 하면 함수의 this가 객체를 가르키고 함수가 갖고 있는 매개변수에 맞게 값들이 들어가서 객체가 수정 된다.

즉 함수를 입력한 객체의 메소드인 것처럼 사용 가능하다.