JavaScript Object Types

JavaScript has a variety of object types, each serving different purposes and functionalities. Here's an overview of the different types of objects in JavaScript:

```
// 1. Plain Objects
let obj = { key: 'value', method: function() { return 'hello'; } };
let anotherObj = new Object();
anotherObj.key = 'value';
anotherObj.method = function() { return 'hello'; };
// 2. Arrays
let arr = [1, 2, 3, 'four', { key: 'value' }];
// 3. Functions
function myFunction() {
    return 'hello';
}
let anotherFunction = function() {
    return 'world';
};
// 4. Date Objects
let date = new Date();
// 5. RegExp Objects
let regex = /hello/i;
// 6. Math Objects
let randomNum = Math.random();
let pi = Math.PI;
// 7. JSON Objects
let jsonString = '{"name": "John", "age": 30}';
let jsonObj = JSON.parse(jsonString);
let backToString = JSON.stringify(jsonObj);
// 8. Error Objects
let error = new Error('Something went wrong');
// 9. Promise Objects
let promise = new Promise((resolve, reject) => {
    // some async operation
    let successful = true; // example success condition
    if (successful) {
        resolve('Success!');
    } else {
        reject('Failure.');
    }
});
// 10. Map Objects
let map = new Map();
map.set('key1', 'value1');
map.set('key2', 'value2');
// 11. Set Objects
let set = new Set();
set.add(1);
set.add(2);
set.add(1); // duplicates are not added
// 12. WeakMap Objects
let weakMap = new WeakMap();
let objKey = {};
```

```
weakMap.set(objKey, 'value');
// 13. WeakSet Objects
let weakSet = new WeakSet();
let objValue = {};
weakSet.add(objValue);
// 14. Symbol Objects
let sym = Symbol('description');
let objWithSym = {
    [sym]: 'value'
};
// 15. Typed Arrays
let buffer = new ArrayBuffer(16);
let int32View = new Int32Array(buffer);
// 16. BigInt Objects
let bigInt = BigInt(12345678901234567890123456789012345678901234567890);
```

typeof Operator

You can use the `typeof` operator to determine the type of an object. Here's how you can use it:

```
let obj = { key: 'value' };
let arr = [1, 2, 3];
let func = function() { return 'hello'; };
console.log(typeof obj); // object
console.log(typeof arr); // object
console.log(typeof func); // function
```

we can also use `typeof` operator to check primitive types:

```
let str = 'hello';
let num = 42;
let bool = true;
let nullValue = null;
console.log(typeof str); // string
console.log(typeof num); // number
console.log(typeof bool); // boolean
console.log(typeof nullValue); // object
```

instanceof Operator

You can use the `instanceof` operator to check if an object is an instance of a particular type. Here's how you can use it:

```
let obj = { key: 'value' };
let arr = [1, 2, 3];
let func = function() { return 'hello'; };
let date = new Date();
console.log(obj instanceof Object); // true
console.log(arr instanceof Array); // true
console.log(func instanceof Function); // true
console.log(date instanceof Date); // true
```

we can also use `instanceof` operator to check if an object is an instance of a class:

```
class Person {
```

```
    constructor(name) {
        this.name = name;
    }
}
let person = new Person('John');
console.log(person instanceof Person); // true
```

we can also use primitive types with `instanceof` operator:

```
let str = 'hello';
let num = 42;
let bool = true;
let nullValue = null;
console.log(str instanceof String); // false
console.log(num instanceof Number); // false
console.log(bool instanceof Boolean); // false
console.log(nullValue instanceof Object); // false
let strObj = new String('hello');
let numObj = new Number(42);
let boolObj = new Boolean(true);
console.log(strObj instanceof String); // true
console.log(numObj instanceof Number); // true
console.log(boolObj instanceof Boolean); // true
```

checkIfInstanceOf(obj, classFunction)

```
function checkIfInstanceOf(obj, classFunction) {
    if (obj === null || obj === undefined || typeof classFunction
!== 'function') return false;
    return Object(obj) instanceof classFunction;
}
```

Flatten Array with nested arrys inside it and given depth

```
var flat = function (arr, n) {
    if (n == 0) return arr;
    let answer = [];
    for (let i=0; i<arr.length; i++) {
        if (n > 0 && Array.isArray(arr[i])) {
            answer.push(...flat(arr[i], n-1));
        } else {
            answer.push(arr[i]);
        }
    }
    return answer;
};
Input
arr = [1, 2, 3, [4, 5, 6], [7, 8, [9, 10, 11], 12], [13, 14, 15]]
n = 1
Output
[1, 2, 3, 4, 5, 6, 7, 8, [9, 10, 11], 12, 13, 14, 15]
```

Deep Equals of two objects

```
function areDeepEqual(o1, o2) {
    if (o1 === null || typeof o1 !== 'object') return o1 === o2;
    if (typeof o1 !== typeof o2) return false;
    if (Array.isArray(o1) !== Array.isArray(o2)) return false;
    if (Array.isArray(o1)) {
        if (o1.length !== o2.length) return false;
        for (let i=0; i<o1.length; i++) {
            if (!areDeepEqual(o1[i], o2[i])) return false;
        }
        return true;
    } else {
        const keys1 = Object.keys(o1);
        const keys2 = Object.keys(o2);
        if (keys1.length !== keys2.length) return false;
        for (let key of keys1) {
            if (!areDeepEqual(o1[key], o2[key])) return false;
        }
        return true;
    }
}
o1 = {"x":1,"y":2}, o2 = {"x":1,"y":2}
console.log(areDeepEqual(o1, o2)); // true
o1 = {"y":2,"x":1}, o2 = {"x":1,"y":2}
console.log(areDeepEqual(o1, o2)); // true
o1 = {"x":null,"L":[1,2,3]}, o2 = {"x":null,"L":["1","2","3"]}
console.log(areDeepEqual(o1, o2)); // false
```

Deep Filter of an object

```
function deepFilter(obj, fn) {
    const dfs = (data) => {
        if (Array.isArray(data)) {
            const res = data.map(dfs).filter(item => item !==
undefined);
            return res.length > 0 ? res :undefined;
        }
        if (typeof data === 'object' && data != null) {
            const res = {}
            for (const key in data) {
                if (data.hasOwnProperty(key)) {
                    const filteredValue = dfs(data[key]);
                    if (filteredValue !== undefined) {
                        res[key] = filteredValue;
                    }
                }
            }
            return Object.keys(res).length > 0 ? res : undefined;
        }
        return fn(data) ? data : undefined;
    };
    return dfs(obj);
}
obj = [-5, -4, -3, -2, -1, 0, 1],
fn = (x) => x > 0
```

```
console.log(deepFilter(obj, fn))
obj = {"a": 1, "b": "2", "c": 3, "d": "4", "e": 5, "f": 6, "g":
{"a": 1}},
fn = (x) => typeof x === "string"
console.log(deepFilter(obj, fn))
obj = [[[[5]]]],
fn = (x) => Array.isArray(x)
console.log(deepFilter(obj, fn))
```

Invert an object

```
function invertObject(obj) {
    const ans = {};
    for (const key in obj) {
        if (ans.hasOwnProperty(obj[key])) {
            if (Array.isArray(ans[obj[key]])) {
                ans[obj[key]].push(key);
            } else {
                ans[obj[key]] = [ans[obj[key]], key];
            }
        } else {
            ans[obj[key]] = key
        }
    }
    return ans;
}
obj = ["1", "2", "3", "4", "4"]
console.log(invertObject(obj))
invertedObj = {"1": "0", "2": "1", "3": "2", "4": "3"}
```

JSON.stringify

```
function jsonStringify(object: any): string {
    if (object === null) {
        return 'null';
    }
    if (typeof object === 'string') {
        return `"${object}"`;
    }
    if (typeof object === 'number' || typeof object === 'boolean')
{
        return object.toString();
    }
    if (Array.isArray(object)) {
        return `[${object.map(jsonStringify).join(',')}]`;
    }
    if (typeof object === 'object') {
        return `{${Object.entries(object)
            .map(([key, value]) =>
`${jsonStringify(key)}:${jsonStringify(value)}`)
            .join(',')}}`;
    }
    return '';
}
```

```
Input: object = {"y":1,"x":2}
Output: {"y":1,"x":2}
```