```
// apply.js

let obj = {
    name: 'yuva'
}


function greet(param){
    console.log(`Hello ${this.name} ${param}`);
}

Function.prototype.myApply = function(context, args){
    let uniqueKey = Symbol();
    context[uniqueKey] = this;
    context[uniqueKey](args);
    delete context[uniqueKey];
}

greet.apply(obj, [1,2,3]);

greet.myApply(obj, [1,2,3]);


// bind.js

let obj = {
    name: 'yuva'
}


function greet(greetMsg){
    console.log(`Hello ${this.name}, ${greetMsg}`);
}

const greetFunc = greet.bind(obj);

greetFunc('Good Morning!');

Function.prototype.myBind = function(context){
    return function(...args){
        this.call(context, ...args);
    }
}

const greetFunc1 = greet.myBind(obj);

greetFunc('Good Evening!');


// call.js

let obj = {
    name: 'yuva'
}


function greet(){
    console.log(`Hello ${this.name}`);
}
```

```
Function.prototype.myCall = function(context, ...args){
    let uniqueKey = Symbol();
    context[uniqueKey] = this;
    context[uniqueKey](...args);
    delete context[uniqueKey];
}


greet.call(obj);

greet.myCall(obj);


// curried.js
function sum(a,b,c,d){
    return a+b+c+d;
}

function curry(func){
    return function curried(...args){
        if(args.length < func.length){
            return (...nextArgs) => curried(...args, ...nextArgs);
        } else {
            return func(...args);
        }
    }

}

const curriedSum = curry(sum);


console.log(curriedSum(1)(2)(3)(4));
console.log(curriedSum(1,2)(3,4));
console.log(curriedSum(1,2,3)(4));
console.log(curriedSum(1,2,3,4));


// debounce.js
function debounce(cbFunc, delay){
    let timer;
    return function(...args){
        clearTimeout(timer);
        timer = setTimeout(() => cbFunc(...args), delay);
    }
}


// deepEqual.js
let obj = {
    name: 'shiv',
    age: '33',
    hey: {
        hello: {
            name: {
                anc: 'abc'
            }
        }
    }
}
```

```
let obj1 = {
    name: 'shiv',
    hey: {
        hello: {
            name: {
                anc: 'abc'
            }
        }
    },
    age: '33'
}


// Warning: Incomplete solution. Refer to below.
function shouldDeepCompare(type) {
    return type === '[object Object]' || type === '[object Array]';
  }

  function getType(value) {
    return Object.prototype.toString.call(value);
  }

function deepEqual(valueA, valueB) {
    const typeA = getType(valueA);
    const typeB = getType(valueB);

    if (typeA === typeB && shouldDeepCompare(typeA) && shouldDeepCompare(typeB)) {
      // When both props are objects or arrays, we traverse into them by calling
`isEqual` again.
    }

    return Object.is(valueA, valueB);
  }


console.log(JSON.stringify(obj) == JSON.stringify(obj1));

console.log(deepEqual(obj, obj1));


// deepOmit.js
const obj = {
    a: 1,
    b: 2,
    c1: {
      d: 3,
      e: 4,
    },
    f: [5, 6],
  };
  let result = deepOmit(obj, ['b', 'c', 'e']); // { a: 1, f: [5, 6] }

  console.log("result...", result);

  function deepOmit(obj, omitArr){


    function omit(nestedObj){

        if(typeof nestedObj != 'object' || nestedObj == null) return nestedObj;
```

```
        if(Array.isArray(nestedObj)){
            return nestedObj.map(omit);
        }
        let ans = {};
        Object.entries(nestedObj).forEach(([key, value]) => {
            console.log("key = ", key);
            if(!omitArr.includes(key)){
                if(typeof value == 'object' && value != null){
                    ans[key] = omit(value);
                } else {
                    ans[key] = value;
                }

            }
        })
        return ans;
    }

    return omit(obj);
  }



// deepclone.js
let obj = {
    name: 'shiv',
    age: '33',
    hey: {
        hello: {
            name: 'text'
        }
    },
    companies: [
        {
            name: 'sapient'
        },
        {
            name: 'harman'
        }
    ]
}

const newObj = {...obj};

function deepClone(obj){


    function clone(object){
        if(Array.isArray(object)){
            return object.map(item => clone(item));
        } else if(typeof object == 'object' && object != null){
            let result = {};
            Object.entries(object).forEach(([key, value]) => {
                result[key] = clone(value)
            })
            return result;
        }
        return object;
    }
```

```
    return clone(obj);
}

const deepCloned = deepClone(obj);

obj.companies[0].name = 'polaris';



console.log(obj);
console.log(newObj);
console.log(deepCloned);



// filter.js
let arr = [1,2,3];

console.log(arr.filter(ele => ele%2));

Array.prototype.myFilter = function(cb){
    let arr = [];
    for(let i=0;i<this.length;i++){
        if(cb(this[i])){
            arr.push(this[i]);
        }
    }
    return arr;
}

console.log(arr.myFilter(ele => ele%2));


// flat.js

let arr = [1,[2,[3,[4]]],5,[6,7],8];

console.log(arr.flat(2));

Array.prototype.myFlat = function(depth){
    const ans = [];
    function flatten(arr, d){
        for(let i=0;i<arr.length;i++){
            if(Array.isArray(arr[i]) && (d > 0 || d == Infinity)){
                flatten(arr[i], d-1);
            } else {
                ans.push(arr[i]);
            }
        }
    }

    flatten(this, depth);
    return ans;
}

console.log(arr.myFlat(Infinity));


// jsonStringify.js
const obj = {
```

```
    a: 1,
    b: 2,
    c1: {
      d: 3,
      e: 4,
    },
    f: [5, 6],
  };


  function JSONStringify(object){

    function stringify(value){
       if(value == null) return 'null'

       if(typeof value == 'string') return value;

       if(typeof value == 'number' || typeof value == 'boolean') return
String(value);

       if(Array.isArray(value)){
            const items = value.map((item) => stringify(item)).join(',');
            return `[${items}]`;
       }

       if(typeof value == 'object' && value != null){
        const entries = Object.entries(value).map(([key, value]) => `${key}:
${stringify(value)}`);
        return `{${entries.join(',')}}`;
       }

       return 'null';
    }

    return stringify(object)
  }

  let res = JSONStringify(obj);

  console.log(res);


// map.js
let arr = [1,2,3];

console.log(arr.map(ele => ele*3));

Array.prototype.myMap = function(cb){
    let arr = [];
    for(let i=0;i<this.length;i++){
        arr[i] = cb(this[i]);
    }
    return arr;
}

console.log(arr.myMap(ele => ele*3));


// map_size.js
const map = new Map(Object.entries([1,2,3,4]));
```

```
console.log(map.size);

Object.defineProperty(Map.prototype, 'length', {
    get: function(){
        return this.size
    },
    enumerable: false,
    configurable: true
})


console.log(map.length);


// memoize.js
function memoize(){
    let cache = {};

    return function multiply(num){
        if(cache[num]) {
            return cache[num];
        }
        else {
            console.log("Generating...")
            cache[num] = 7 * num;
            return cache[num];
        }
    }
}


const multiplyBy7 = memoize();
const ans = multiplyBy7(4);
console.log("Ans = ", multiplyBy7(4));
console.log("Ans = ", multiplyBy7(9));
console.log("Ans = ", multiplyBy7(8));
console.log("Ans = ", multiplyBy7(4));



// mixin.js
const logging = {
   log(){
    console.log(`Logging ${this.name}`)
   }
}

const user = {name: 'yuva'};

Object.assign(user, logging);

user.log();


// myPromise.js
function MyPromise(executor) {
    let value;
    let onResolve, onReject;

    let isFulfilled = false,
```

```
    isRejected = false,
    isCalled = false;

function resolve(val) {
    if (isFulfilled || isRejected) return;
    isFulfilled = true;
    value = val;

    if (typeof onResolve === 'function' && !isCalled) {
        isCalled = true;
        onResolve(value);
    }
}

function reject(err) {
    if (isFulfilled || isRejected) return;
    isRejected = true;
    value = err;

    if (typeof onReject === 'function' && !isCalled) {
        isCalled = true;
        onReject(value);
    }
}

this.then = function (callback) {
    return new MyPromise((resolve, reject) => {
        onResolve = function (val) {
            try {
                const result = callback(val);
                if (result instanceof MyPromise) {
                    result.then(resolve, reject);
                } else {
                    resolve(result);
                }
            } catch (err) {
                reject(err);
            }
        };

        if (isFulfilled && !isCalled) {
            isCalled = true;
            onResolve(value);
        }
    });
};

this.catch = function (callback) {
    return new MyPromise((resolve, reject) => {
        onReject = function (err) {
            try {
                const result = callback(err);
                resolve(result);
            } catch (e) {
                reject(e);
            }
        };

        if (isRejected && !isCalled) {
            isCalled = true;
```

```
                onReject(value);
            }
        });
    };

    this.finally = function (callback) {
        return this.then(
            (val) => MyPromise.resolve(callback()).then(() => val),
            (err) => MyPromise.resolve(callback()).then(() => { throw err; })
        );
    };

    try {
        executor(resolve, reject);
    } catch (err) {
        reject(err);
    }
}

// '  Static Methods

MyPromise.resolve = function (val) {
    return new MyPromise((resolve) => resolve(val));
};

MyPromise.reject = function (err) {
    return new MyPromise((_, reject) => reject(err));
};

MyPromise.all = function (promises) {
    return new MyPromise((resolve, reject) => {
        const results = [];
        let count = 0;

        promises.forEach((p, i) => {
            MyPromise.resolve(p).then((val) => {
                results[i] = val;
                count++;
                if (count === promises.length) resolve(results);
            }).catch(reject);
        });
    });
};

MyPromise.allSettled = function (promises) {
    return new MyPromise((resolve) => {
        const results = [];
        let count = 0;

        promises.forEach((p, i) => {
            MyPromise.resolve(p).then((val) => {
                results[i] = { status: 'fulfilled', value: val };
            }).catch((err) => {
                results[i] = { status: 'rejected', reason: err };
            }).finally(() => {
                count++;
                if (count === promises.length) resolve(results);
            });
        });
    });
```

```
};

MyPromise.race = function (promises) {
    return new MyPromise((resolve, reject) => {
        promises.forEach((p) => {
            MyPromise.resolve(p).then(resolve).catch(reject);
        });
    });
};

MyPromise.any = function (promises) {
    return new MyPromise((resolve, reject) => {
        let rejections = [];
        let count = 0;

        promises.forEach((p, i) => {
            MyPromise.resolve(p).then(resolve).catch((err) => {
                rejections[i] = err;
                count++;
                if (count === promises.length) {
                    reject(new AggregateError(rejections, 'All promises were
rejected'));
                }
            });
        });
    });
};

const promise = new MyPromise((resolve) => {
    setTimeout(() => {
        resolve("Success...");
    }, 1000);
});

promise
    .then((data) => {
        console.log("1:", data);
        return MyPromise.resolve("Next value");
    })
    .finally(() => {
        console.log("2: In finally");
    })
    .then((data) => {
        console.log("3:", data);
        return MyPromise.reject("Oops");
    })
    .catch((err) => {
        console.log("4: Caught", err);
    });

const one = new MyPromise((res) => setTimeout(() => res("A"), 100));
const two = new MyPromise((res) => setTimeout(() => res("B"), 200));
const fail = new MyPromise((_, rej) => setTimeout(() => rej("Error"), 150));

MyPromise.all([one, two])
    .then(console.log) // ["A", "B"]

MyPromise.any([fail, one])
    .then(console.log) // "A"
```

```
MyPromise.allSettled([one, fail])
    .then(console.log); // [{status: "fulfilled", value: "A"}, {status:
"rejected", reason: "Error"}]

MyPromise.race([two, fail])
    .then(console.log)
    .catch(console.log); // "Error"
```

```
// pipe.js

const add5 = (value) => value +5;
const multiply3 = (value) => value * 3;
const subtract4 = (value) => value - 4;

function pipe(...functions){
    return (value) => {
        return functions.reduce((currParam, currFun) => currFun(currParam),
value);
    }
}

const processNumber = pipe(add5, multiply3, subtract4);

console.log(processNumber(2));
```

```
// privatePropClass.js
class BankAccount{
    #balance;

    constructor(name, balance){
        this.name = name;
        balance = balance;
    }

    withdraw(amount){
        balance = balance - amount;
        consolee
    }

    getDetails(){
        console.log(`Details = ${JSON.stringify(this)}`);
    }
}

const yuva_acc = new BankAccount('yuva', 1000);
const dhrityi_acc = new BankAccount('Dhrityi', 5000);


yuva_acc.getDetails();

dhrityi_acc.getDetails();

console.log(Object.getOwnPropertyNames(yuva_acc));
```

```
// propertyDescriptors.js
let obj = {
    name: 'yuva',
    age: 33
}

console.log(Object.getOwnPropertyDescriptors(obj));


// proxyAndReflect.js
const user = {
    name: 'yuva',
    age: 33
}

const handler = {
    set(target, prop, value){
        console.log("prop", prop, "typeof value ", typeof value, "value = ",
value)
        if(prop == 'age' && typeof value !== 'number'){
            throw new Error('Age must be a number!');
        }
        return Reflect.set(target, prop, value);
    }
}

const ProxyUser = new Proxy(user, handler);


ProxyUser.age = '34';



// reduce.js
let arr = [1,2,3];

console.log(arr.reduce((prev, curr) => prev+curr, 10));

Array.prototype.myReduce = function(cbFunc, initialValue){
   let startIndex, currValue;

  if(initialValue){
    currValue = initialValue;
    startIndex = 0;
  } else {
    currValue = this[0];
    startIndex = 1;
  }

  for(let i=startIndex;i<this.length;i++){
    currValue = cbFunc(currValue, this[i]);
  }
  return currValue;
}

console.log(arr.myReduce((prev, curr) => prev+curr, 10));


// squash_object.js
const object = {
```

```
    a: 5,
    b: 6,
    c: {
      f: 9,
      g: {
        m: 17,
        n: 3,
      },
    },
  };

  function squashObject(object){

    function squash(obj, prefix= '', result = {}){
        // console.log(obj);
        for(let [key,value] of Object.entries(obj)){
          let newKey = prefix ? `${prefix}.${key}` : key;
          if(typeof value == 'object' && value !=null){
            squash(value, newKey, result);
          } else {
            result[newKey] = value;
          }
        }
        return result;
    }

   return  squash(object);
  }

  let result = squashObject(object); // { a: 5, b: 6, 'c.f': 9, 'c.g.m': 17,
'c.g.n': 3 }

  console.log(result);
```

```
// throttle.js
function throtttle(func, delay){
    let flag = false;

    return function(...args){
        if(!flag){
            flag = true;
            func(...args);
            setTimeout(() => {
                flag = false;
            }, delay);
        }

    }
}
```