HOF

```
function filter(elems, predicate) {
    let results = [];
    for (let i = 0; i < elems.length; i++) {
        if (predicate(elems[i])) {
            results.push(elems[i]);
        }
    }
    return results;
}
```

```
let nos = [5, 1, 3, 4, 2, 10, 13];

let evens = filter(nos, function(n) {
    return n % 2 === 0;
});
```

HOF

```
function filter(elems, predicate) {
    let results = [];
    for (let i = 0; i < elems.length; i++) {
        if (predicate(elems[i])) {
            results.push(elems[i]);
        }
    }
    return results;
}
```

```
let products = [
    {"id":`1`, "name":`iPhone`, "price":`90000`, category: 'mobile'},
    {"id":`2`, "name":`Sony Bravia`, "price":`290000`, category: 'tv'},
    {"id":`3`, "name":`Logitech `, "price":`5000`, category: 'computer'},
    {"id":`4`, "name":`Samsung Fold`, "price":`210000`, category: 'mobile'},
    {"id":`5`, "name":`LG OLED`, "price":`210000`, category: 'tv'},
    {"id":`6`, "name":`Wacom`, "price":`4500`, category: 'computer'},

]
```

```
let mobiles = filter(products, function(product) {
    return product.category === 'mobile';
});
```

HOF

```
function adder(base) {
    return function(arg) {
        return base + arg;
    }
}

let fiveAdder = adder(5);
```

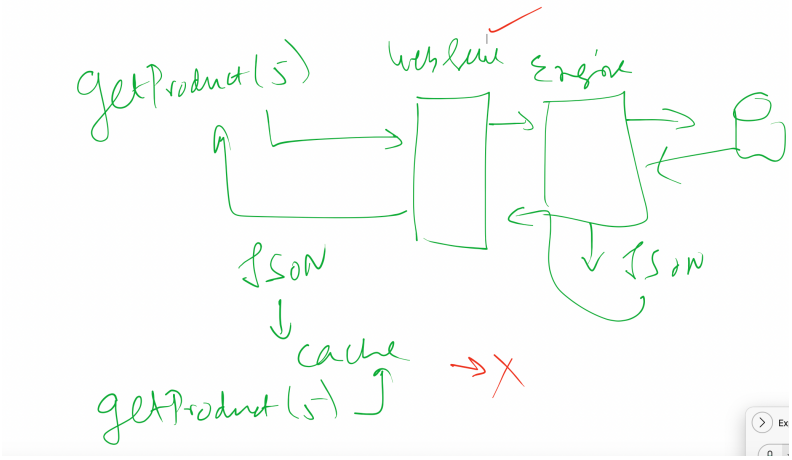
Closure

HEAP

5

base

```
> fiveAdder.prototype
< {} i
    constructor: ƒ (arg)
        arguments: null
        caller: null
        length: 1
        name: ""
    prototype:
        constructor: ƒ (arg)
            arguments: null
            caller: null
            length: 1
            name: ""
            prototype: {}
            [[FunctionLocation]]: VM98:3
            [[Prototype]]: ƒ ()
            [[Scopes]]: Scopes[3]
                0: Closure (adder) {base: 5}
                1: Script {fiveAdder: ƒ}
                2: Global {window: Window, self: Window, document: document, name: '', location: Location, …}
        [[Prototype]]: Object
```




```
function fibanocci(no) {
    if (no <= 1) {
        return no;
    }
    return fibanocci(no - 1) + fibanocci(no - 2);
}

function memoize(fn) {
    let cache = {};
    return function (no) {
        if (cache[no] === undefined) {
            cache[no] = fn(no);
        }
        return cache[no];
    }
}

let memFib =  memoize(fibanocci);

console.time("first");
console.log(memFib(34));
console.timeEnd("first");

console.time("second");
console.log(memFib(34)); /
console.timeEnd("second");
```