Third Pillar of JS

Higher Order Function :

- A function which take another function as argument.

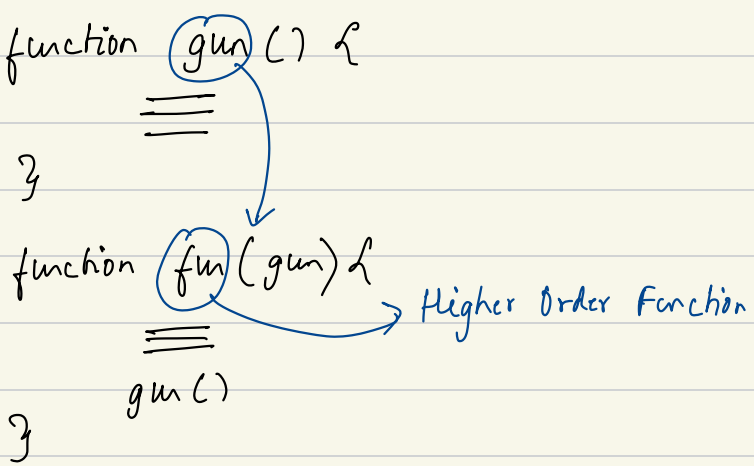
07-Third Pillar of JS > 01.hof.js > ...

```
// Higher Order Function

function f(x, fn) {
  /**
   * x -> number
   * fn -> function
   */
  console.log(x);
  fn();
}

f(10, function exec() {
  console.log(" I am an expression passed to a HOF");
});
```

- These are functions that depends and operate on other function. HOF take another function as argumat or return a function and then execute the logic.



Custom HOF functions: Array function

```
/*

    arrays are also custom object in JS
    index of the element is the key and the element it self is the value
    ["abc", "def", "ghi"] -> {0: "abc", 1: "def", 2: "ghi"}

*/
```

Map Functions:

```
/**
 * map function
 *  map is a higher order function available with arrays
 *  it takes a function as a argument -> f
 *  it return an array in which every value is actually populated by calling
 *  function f with original array element as argument
 */
```

Code:

```
function square(element) {
  return element * element;
}

const arr = [1, 2, 3, 4, 5];

const result = arr.map(square); // square is function passed as an argument.
console.log(result); // 1, 4, 9, 16, 25
```

So basically, every element of the original array is passed one by one in the argument function "f" whatever is the output for each individual elemat, we populate that output in an array.

Map internally iterates/loops over every element of the given original array pass that element in the argument function f and then store the returned value inside an array.

When to use Map?

04.whenToUseMap.md

07-Third Pillar of JS > 04.whenToUseMap.md

```
When to use maps?

In any situation when we have to do an operation on every element of the array and store
the result of each operation map can be a good option

For example:
Array of product objects
```

Index in array:

07-Third Pillar of JS > 05.indexwithMap.js > ...

```js
const newArr = [9, 8, 7, 6, 5];

/**
 *  if the function that we are passing takes two argument
 *  then first argument will be accesssing the actual value
 *  second argumennt will be accesssing index of the value
 */

function print(element, index) {
  return `Element at index ${index} is ${element}`;
}

/**
 *  here map is looping over every element
 *  and then passing element, index in the function print
 */
const result = newArr.map(print);
console.log(result);
```

```
[
  'Element at index 0 is 9',
  'Element at index 1 is 8',
  'Element at index 2 is 7',
  'Element at index 3 is 6',
  'Element at index 4 is 5'
]
```

Custom map:

06.customMapper.js U

07-Third Pillar of JS > 06.customMapper.js > ...

```js
const newArr = [9, 8, 7, 6, 5];
function customMapper(arr, func) {
  let result = [];
  for (let i = 0; i < arr.length; i++) {
    result.push(func(arr[i], i));
  }
  return result;
}

function print(element, index) {
  return `Element at index ${index} is ${element}`;
}

const value = customMapper(newArr, print);
console.log(value);
```

```
[
  'Element at index 0 is 9',
  'Element at index 1 is 8',
  'Element at index 2 is 7',
  'Element at index 3 is 6',
  'Element at index 4 is 5'
]
```

More about sort() :

```
let b = [1, 10, 9, 100, 1000, 11, 12, 13, 14, 2, 3];

b.sort(function (a, b) {
  // if a < b -> a - b is -ve -> if cmp function gives -ve then a is placed before b (a < b)
  // if a > b -> a - b is +ve -> if cmp function gives +ve then b is placed before a (a > b)
  return a - b;
}); // sort is a HOF, the sort function takes a comparator function as argument

console.log(b);
```

```
[
   1,   2,   3,  9,
  10,  11,  12, 13,
  14, 100, 1000
]
```

Filter function

```
/**
 *   Filter Function
 *      Filter is also HOF.
 *      Filter also loops over the array element
 *      there is one special thing about filter, i.e. the argument function f which
 *      we have to pass inside filter should always return a boolean, otherwise output will
 *      be converted to a boolean
 *
 *  Filter loops over every element, passes that element in the argument function and then if the
 *  output of the this function call is true, then it stores the original element in a new array otherwise
 *  doesn't add this add element to the array.
 */
```

Code :

```
function oddEven(x) {
  return x % 2 == 0; // returning a boolean
}

let arr = [1, 2, 3, 4, 5, 6, 7, 8, 9];

const res = arr.filter(oddEven);
console.log(res); // [ 2, 4, 6, 8 ]
```

Reduce f() :

09.reduceFunction.js U

07-Third Pillar of JS > 09.reduceFunction.js > ...

```
/**
 *
 * Reduce:
 *    It's a HOF, available for arrays.
 *    It also takes a function "f" as an argument.
 *    What reduce does is, it one by one goes to every element of the arr,
 *    say the current is arr[0]
 *    reduce will pass the element to the function f, and accumlate the result of further
 *    function calls with this particular result.
 */
```

Code :

```
const arr = [1, 2, 3, 4, 5, 6];
function sum(prevResult, currValue) {
  console.log(prevResult, currValue);
  return prevResult + currValue;
}

const result = arr.reduce(sum);
console.log(result); // 21
```