Front-end Essentials

Higher-order Functions

Table of Contents

Section 1

Function

Function

➢ What is function?

- A function is a block of code designed to perform a particular task.
- Values can be passed into functions and used within the function.
- It allows code-reuse.
- Its give us the ability to break down complex requirement into smallers, easy to solve then combine back together.

Function Declaration

- A function declaration defines a named function.
- To create a function declaration you use the function keyword followed by the name of the function
- Syntax:

```
function name([param[, param,[..., param]]]) {
   [statements]
}
```

Function Expression

- A function declaration defines a named or anonymous function.
- Syntax:

```
var myFunction = function [name]([param1[, param2[, ..., paramN]]]) {
   statements
};
```

```
function sayHi() {
  alert( "Hello" );
}
```

=>

```
let sayHi = function() {
  alert( "Hello" );
};
```

Function

- IIFE(Immediately Invoked Function Expression)
  - An IIFE is a JavaScript function that runs as soon as it is defined.
  - Syntax:

```
(function () {
    statements
})();
```

  - Example:

```
(function () {
    var aName = "Barry";
})();
// Variable aName is not accessible from the outside scope
aName // throws "Uncaught ReferenceError: aName is not defined"
```

Function

- **Parameters vs Arguments**
  - *Parameters* are variables listed as a part of the function definition.
  - *Argument* are the values the function receives from each parameter when the function is executed (invoked)

Parameters are the 'slots', the placeholders for inputs that the function should receive

```
function bottleCapper(bottle, cap) {
  return bottle + cap;
}

bottleCapper("green bottle", "white cap");
```

Arguments are the specific values of JS data types that we give those slots when we run a function

Section 2

Abstraction

Abstraction

➢ Take a look to 2 paragraph

"Put 1 cup of dried peas per person into a container. Add water until the peas are well covered. Leave the peas in water for at least 12 hours. Take the peas out of the water and put them in a cooking pan. Add 4 cups of water per person. Cover the pan and keep the peas simmering for two hours. Take half an onion per person. Cut it into pieces with a knife. Add it to the peas. Take a stalk of celery per person. Cut it into pieces with a knife. Add it to the peas. Take a carrot per person. Cut it into pieces. With a knife! Add it to the peas. Cook for 10 more minutes."

"Per person: 1 cup dried split peas, half a chopped onion, a stalk of celery, and a carrot.

Soak peas for 12 hours. Simmer for 2 hours in 4 cups of water (per person). Chop and add vegetables. Cook for 10 more minutes."

Abstraction

➢ What is abstraction?

- Abstractions hide details and give us the ability to talk about problems at a higher (or more abstract) level.
- When programming, we can’t rely on all the words we need to be waiting for us in the dictionary
- We might fall into the pattern of the first recipe—work out the precise steps the computer has to perform, one by one, blind to the higher-level concepts that they express.

Abstract repetition

- Abstract repetition

```
for (let i = 0; i < 10; i++) {
  console.log(i);
}
```

```
function repeatLog(n) {
  for (let i = 0; i < n; i++) {
    console.log(i);
  }
}
```

```
function repeat(n, action) {
  for (let i = 0; i < n; i++) {
    action(i);
  }
}
```

Summary

- Abstractions hide details and give us the ability to talk about problems at a higher level
- Abstractions allow us to **focus** on the main part of a problem we need to solve.
- After we done with the main part, we can work on other details later on

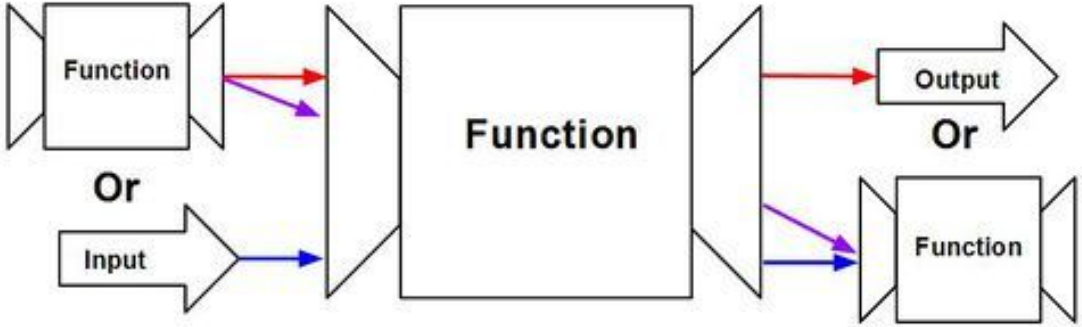
Section 3

Higher-order Functions

Higher-order Functions

➢ What is the Higher-order Functions:

- A higher-order function is a function that can take another function as an argument, **or** that returns a function as a result **or** both.



Higher-order Functions

- **Syntax**

```
function higherOrder(fn) {
    // logic
    // typeof fn === 'function'
}

function higherOrder2() {
    return function() {
        // logic
    }
}

function higherOrder3(fn) {
    // typeof fn === 'function'
    return function() {
        // logic
        fn();
    }
}
```

Usage with map

- **Example**

```
var array = [1, 2, 3, 4];

var newArray = array.map(function(x) {
   return x * 2;
});

console.log(newArray);
```

Usage with map

- **array.map():**
  - The map() method creates a new array with the results of calling a function for every array element.
  - ***Syntax:*** array.map(function(**currentElement**, index, arr), thisValue)

```
var array = [1, 2, 3, 4];

var newArray = array.map(function(x, index, arr) {
   return x * 2;
});

console.log(newArray);
```

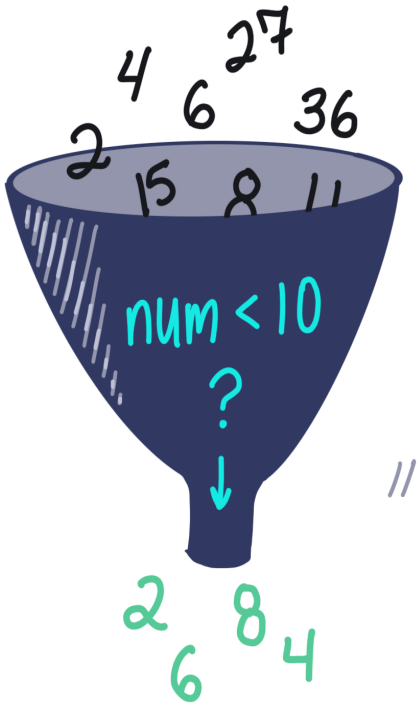
Usage with filter

- **array.filter()**
  - The filter() method creates an array filled with all array elements that pass a test (provided as a function).
  - **Syntax:**

    array.filter(function(**currentValue**, index, arr), thisValue)

Usage with filter

- **array.filter()**




```
const arr =
[15, 2, 8, 36, 11, 4, 6, 27]

const smallNums =
  arr.filter (num => {
    return num < 10
  })

// smallNums =
[2, 8, 4, 6]
```

Usage with reduce

- **array.reduce()**
  - The reduce() method reduces the array to a value.
  - The reduce() method executes a provided function for each value of the array (from left-to-right).
  - **Syntax:**

    array.reduce(function(**total**, **currentValue**, currentIndex, arr), initialValue)

Usage with reduce

- **Use reduce() to calculate sum of array**

```
var array = [1, 2, 3, 4];

var value = array.reduce(function(acc, ele) {
    return acc + ele;
}, 0);

console.log(value); // 10
```

Usage with reduce

- **array.reduce()**

```
const ingredients = ["wine", "onion", "mushrooms"]
```

let's reduce this array to a single output

```
ingredients.reduce((sauce, item) => {
    return (sauce + cook(item))
}
```

returns a sauce full of cooked items

Summary

- Higher-order functions is a function and take a function as parameter or return a function or both
- Use map() to **transform** an array to a new array
- Use filter() to **filter** out element
- Use reduce() to **combine**
- There is also array.sort()

Thank you