Advanced Javascript

https://github.com/uqutub

Background

As developers, we gravitate toward frameworks that help us create extensible and clean application architectures. Yet the complexity of our codebase still gets out of control, and we're challenged to reexamine the basic design principles of our code.

Object-oriented design helps solve part of the problem; but because JavaScript is such a dynamic language with lots of shared state. It isn't long before we accumulate enough complexity to make our code unwieldy and hard to maintain

Because functional programming isn't a framework or a tool, but a way of writing code, thinking functionally is radically different from thinking in object-oriented terms. But how do you become functional? How do you begin to think functionally?

Before you can learn to think functionally, first you must learn what FP is.

1.1 Functional Programing

- Functional programming is a programming paradigm, meaning that it is a way of thinking about software construction based on some fundamental.
- In simple terms, functional programming is a software development style that places a major emphasis on the use of functions. You might say, “Well, I already use functions on a day-to-day basis at work; what’s the difference?” FP requires you to think a bit differently about how to approach the tasks you’re facing. It’s not a matter of just applying functions to come up with a result; the goal, rather, is to *abstract control flows and operations* on data with functions in order to *avoid side effects* and *reduce mutation of state* in your application

1.2 Why Functional Programing?

- Extensibility—Do I constantly refactor my code to support additional functionality?
- Easy to modularize—If I change one file, is another file affected?
- Reusability—Is there a lot of duplication?
- Testability—Do I struggle to unit test my functions?
- Easy to reason about—Is my code unstructured and hard to follow?

What is a Function?

- A function is a process which takes some input, called arguments, and produces some output called a return value.

- A JavaScript function is a block of code designed to perform a particular task. A JavaScript function is executed when "something" invokes it (calls it).

Syxtax:

```
function fncName() {
```

1.2 Example: Old Way

In this example the program is simple, but because everything is hardcoded, you can't use it to display messages dynamically.

Say you wanted to change the formatting, the content, or perhaps the target element; you'd need to rewrite this entire expression.

```
document.querySelector('#mydiv').innerHTML = '<h1>Hello World</h1>';
```

1.2 Example: Somewhat Functional

An improvement, indeed, but still not a completely reusable piece of code

```
function printMessage(elementId,  format,  message){
                    document.querySelector('#'+elementId).innerHTML  =
'<'+format+'>'+message+'</'+format+'>'
}

printMessage("mydiv1","h1","Hello")
printMessage("mydiv","h2","Hello  World")
```

1.2 Fundamental Concepts of FP

In order to fully understand functional programming,
first you must learn the fundamental concepts on which it’s based:

- Declarative programming
- Pure functions
- Referential transparency
- Immutability

1.2.1 Programming Paradigms

A programming paradigm is a fundamental style of computer programming. There are four main paradigms: imperative, declarative, functional (which is considered a subset of the declarative paradigm) and object-oriented.

Declarative programming : (Declarative -> `what` you want done) is a programming paradigm that expresses the logic of a computation(What do) without describing its control flow(How do). Some wel-known examples of declarative domain specific languages (DSLs) include CSS, regular expressions, and a subset of SQL (SELECT queries, for example)

Imperative programming : (Imperative -> `how` you want it done) is a programming paradigm that describes computation in terms of statements that change a program state. Imperative programming tells the computer, in great detail, how to perform a certain task.

Functional programming : is a programming paradigm that treats computation as the evaluation of mathematical functions and avoids state and mutable data. It emphasizes the application of functions, in contrast to the imperative programming style, which emphasizes changes in state. In a pure functional language, such as Haskell, all functions are without side effects, and state changes are only represented as functions that transform the state.

1.2.1 Programming Paradigms

    - With **declarative** programming, you write code that describes what you want, but not necessarily how to get it
    - You should prefer declarative programming over the imperative programming

- **Imperative** programming tells the machine how to do something (resulting in what you want to happen)
- Declarative programming tells the machine what you would like to happen (and the computer figures out how to do it)
- **Declarative** - specify **what** to do, **not how** to do it
  E.g.: HTML describes what should appear on a web page, not how it should be drawn on the screen
- **Imperative** - specify both **what** and **how**
  a. `int x;` - what (declarative)
  b. `x=x+1;` - how

1.2.1 Programming Paradigms

In any program, you will always have both imperative and declarative codes, what we should aim for is to hide all **imperative** codes behind the abstractions, so that other parts of the program can use them **declarativel**y.

Imperative Programming

```
var array = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9];
alert(array);
for(let i = 0; i < array.length; i++) {
	array[i] = Math.pow(array[i], 2);
}
alert(array);
```

Declarative Programming

```
var arr = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9];
var newArr = arr.map(
function(num) {
	return Math.pow(num,5);
});
alert(arr);
alert(newArr);
```

Lambda Functions

Lambda expressions provide a succinct alternative to anonymous functions that can be passed in as a function argument

Function:

```
function (num) {
    return Math.pow(num, 2);
}
```

Lambda Function:

```
num => Math.pow(num,2)
```

1.2.2 Pure Functions

- A **pure function** doesn't depend on and doesn't modify the states of variables out of its scope. Concretely, that means a **pure function** always returns the same result given same parameters. Its execution doesn't depend on the state of the system. **Pure functions** are a pillar of functional programming.

- A pure function is a function which:
  - Given the same input, will always return the same output.
  - Produces no side effects.
  - Relies on no external mutable state.

Characteristics of Pure Functions

- The return value of the pure functions solely depends on its arguments
    - Hence, if you call the pure functions with the same set of arguments, you will always get the same return values
- They do not have any side effects like network or database calls
- They do not modify the arguments which are passed to them

For example,

```
function calculateSquareArea(x) {
    return x * x;
}
```

Characteristics of Impure Functions

- The return value of the impure functions does not solely depend on its arguments
    - Hence, if you call the impure functions with the same set of arguments, you might get the different return values
    - For example, Math.random(), Date.now()
- They may have any side effects like network or database calls
- They may modify the arguments which are passed to them

For example,

```
function squareAll(items) {
  var len = items.length;
```

Example: Impure Functions

```
var values = { a: 1 };

function impureFunction ( items ) {
  var b = 1;

  items.a = items.a * b + 2;

  return items.a;
}

var c = impureFunction( values );
// Now `values.a` is 3, the impure function modifies it.
```

Here we modify the attributes of the given object. Hence we modify the object which lies outside of the scope of our function: the function is impure.

Example: Pure Functions

```
var values = { a: 1 };

 function pureFunction ( a ) {
   var b = 1;

   a = a * b + 2;

   return a;
 }

 var c = pureFunction( values.a );
 // `values.a` has not been modified, it's still 1
```

Now we simply modify the parameter which is in the scope of the function, nothing is modified outside

Url: http://www.nicoespeon.com/en/2015/01/pure-functions-javascript/

Example: Impure Functions

```
var values = { a: 1 };
var b = 1;

function impureFunction ( a ) {
  a = a * b + 2;

  return a;
}

var c = impureFunction( values.a );
// Actually, the value of `c` will depend on the value of `b`.
// In a bigger codebase, you may forget about that, which may
// surprise you because the result can vary implicitly.

```

Here, **b** is not in the scope of the function. The result will depend on the context: surprises expected!

Example: Pure Functions

```
var values = { a: 1 };
 var b = 1;

 function pureFunction ( a, c ) {
   a = a * c + 2;

   return a;
 }

 var c = pureFunction( values.a, b );
 // Here it's made clear that the value of `c` will depend on
 // the value of `b`. No sneaky surprise behind your back.
```

1.2.3 Referential transparency

Referential transparency, a term commonly used in functional programming, means that given a function and an input value, you will always receive the same output. That is to say there is no external state used in the function

Here is an example of a referential transparent function:

```
function plusOne(int x) { return x+1; }
```

- A referentially transparent function is one which acts like a mathematical function; given the same inputs, it will always produce the same outputs. It implies that the state passed in is not modified, and that the function has no state of its own.

https://stackoverflow.com/questions/210835/what-is-referential-transparency

1.2.4 Preserving Immutable data

- Immutable data is data that can’t be changed after it’s been created.

What is Mutable?

A mutable object is an object whose state can be modified after it is created.

Examples of native JavaScript values that are mutable include objects, arrays, functions, classes, sets, and maps.

Finally, its worth noting that it's still possible to treat JavaScript objects as immutable. This can first be done through Object.freeze, which shallowly renders a JavaScript object immutable. But it can also be done with programmer discipline. If we want to rely on object's being immutable, it's possible to enforce that all object updates are done

What is Immutable?

An immutable object is an object whose state cannot be modified after it is created.

Examples of native JavaScript values that are immutable are numbers and strings.

—

It's no surprise Marcos uses Google Translate in his shop regularly.

There are 23 officially recognized languages in the EU.

Tip

Don't let data stand alone. Always relate it back to a story you've already told, in this case, Marco's shop.

Source: theguardian.com

Milestones

October 2014

Translate web pages with Chrome extension

October 2015

Translate text within an app

2014

2015

August 2015

Translate conversations through your Android watch

November 2015

Translate written text from English or German to Arabic with the click of a camera

What people are saying

With this app, I'm confident to plan a trip to rural Vietnam

Wendy Writer, CA

Visual translation feels like magic

Ronny Reader, NYC

Translate has officially inspired me to learn French

Abby Author, NYC

Quotes for illustration purposes only

URLS

- https://alistapart.com/article/making-your-javascript-pure
- https://medium.com/javascript-scene/master-the-javascript-interview-what-is-a-pure-function-d1c076bec976