Functions in JavaScript

Defining Functions

Dan Geabunea

Passionate Software Developer

@romaniancoder | www.dangeabunea.com

Functions in JavaScript

Version Check

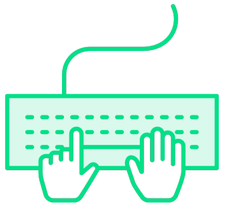
Version Check

This course was created by using:

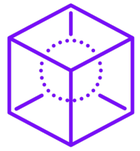
- ECMAScript 2022 Specification
- Node 16.x LTX

Without functions, any computer program would be useless

Deep Dive into Functions



Define and invoke functions



Understand scope and closure

Pass data to functions

Demystify “this”

Use methods, getters & setters

Use asynchronous functions

Understand How to Use Functions in JS Effectively

Become a better JavaScript developer

Become a better TypeScript developer

Understanding JS functions can be leveraged on all JS projects: Node, Angular, React, Vue, etc.

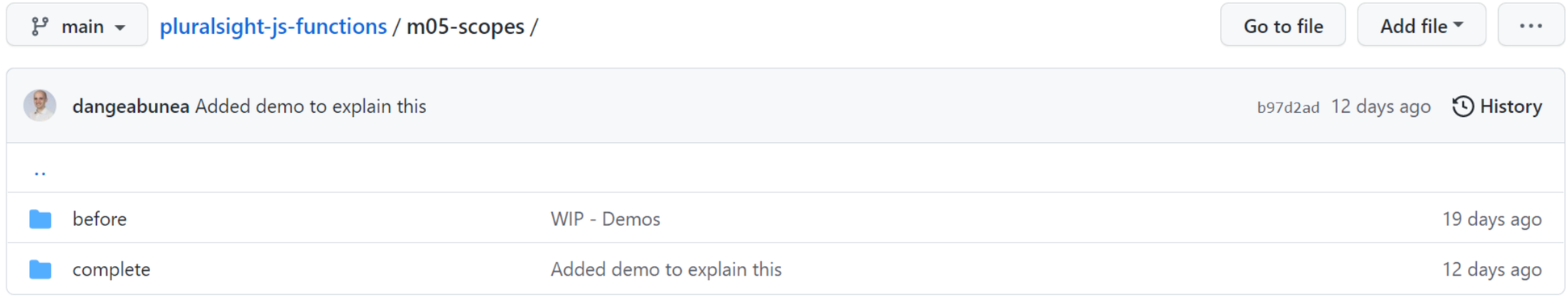
Get the most out of this course by trying out the demos

Source Code

dangeabunea Added demo to explain this `b97d2ad` 12 days ago **13** commits

m02-defining-functions	Fixed main.js	15 days ago
m03-function-parameters	WIP m03	27 days ago
m04-methods	All demos now have html	15 days ago
m05-scopes	Added demo to explain this	12 days ago
m06-this	Added demo to explain this	12 days ago
m07-asynchronous-functions	Added demo to explain this	12 days ago
.gitignore	Initial commit	28 days ago
readme.md	Modified readme	15 days ago

Source Code


main
pluralsight-js-functions / m05-scopes /
Go to file
Add file
dangeabunea Added demo to explain this
b97d2ad 12 days ago
History
..
before
WIP - Demos
19 days ago
complete
Added demo to explain this
12 days ago

Requirements: Text Editor

Web Storm

VS Code, Sublime

stackblitz.com

Defining Functions

Overview

Understanding what a function is made of

Exploring multiple ways to define functions:

- Function declaration
- Function expression
- Arrow function
- Function constructor

IIFE (Immediately invoked function expression)

Function

A block of code that executes a specific task

Function Anatomy

- [x] Name
- [x] Parameters
- [x] Body
- [x] Return statement

Declaring Functions in JavaScript

Function declaration

Function expression

Arrow function

Function constructor

Function Declaration

```
// Function that returns a value

function convertToLiters(gallons) {

    return gallons * 3.785;

}


// Invoke/Call the function

let result = convertToLiters(10);
```

Function Declaration

1. The “function” keyword
2. Function name
3. Parameters
4. Body
5. Return statement

```
// Function that does not return any value

function printTime() {

  console.log(new Date().toLocaleTimeString());

}


printTime();
```

Function Declaration

1. The “function” keyword
2. Function name
3. Body

Defining Functions - Function Declaration

Function Expression

```
function convertToLiters(gallons) {
    return gallons * 3.785;
}
const convertToLiters = function(gallons){
    return gallons * 3.785;
}
// Invoke/Call the function
let result = convertToLiters(10);
```

Function Expression

Can be assigned to a variable
Can be anonymous
It can be invoked by using the variable that it was assigned to

```
let result = convertToLiters(10); // result = 37.84


function convertToLiters(gallons) {
  return gallons * 3.785;
}
```

Hoisting

Function declarations are hoisted, meaning functions can be invoked before they are declared

```
let result = convertToLiters(10);


const convertToLiters = function(gallons) {
  return gallons * 3.785;
}


// Error
```

Hoisting

Function expressions are not hoisted; they need to be declared before they are invoked

You

“Which function definition should I use?”

Version

v8.27.0

Search

USER GUIDE

Getting Started

Core Concepts

Configuring

Configuration Files (New)

Configuration Files

Configuring Language Options

Configuring Rules

Configuring Plugins

Ignoring Code

Command Line Interface

Rules

Formatters

Integrations

Migrating to v8.x

no-use-before-define

Disallow the use of variables before they are defined

In JavaScript, prior to ES6, variable and function declarations are hoisted to the top of a scope, so it's possible to use identifiers before their formal declarations in code. This can be confusing and some believe it is best to always declare variables and functions before using them.

In ES6, block-level bindings (`let` and `const`) introduce a "temporal dead zone" where a `ReferenceError` will be thrown with any attempt to access the variable before its declaration.

Rule Details

This rule will warn when it encounters a reference to an identifier that has not yet been declared.

Examples of **incorrect** code for this rule:

```
1   /*eslint no-use-before-define: "error"*/
2
3   alert(a);
4   var a = 10;
5
6   f();
7   function f() {}
8
9   function g() {
10      return b;
11  }
```

ESLint: tinyurl.com/2nuc9hjr

Defining Functions - Function Expression

Arrow Function

Arrow Function

A compact alternative to a function expression with some limitations

MDN Web Docs: https://tinyurl.com/sb36nhv8

Function Expression to Arrow Function I

```
const printSum = function (a, b) {
  const sum = a + b;
  console.log(sum);
};
```

```
const printSum = (a, b) => {
  const sum = a + b;
  console.log(sum);
};
```

Function Expression to Arrow Function II

```
const convertToLiters = function(gallons) {
  return gallons * 3.785;
}

// Step 1: Remove the function keyword and add an arrow
const convertToLiters = (gallons) => {
  return gallons * 3.785;
}

// Step 2: Get rid of parentheses
const convertToLiters = gallons => {
  return gallons * 3.785;
}

// Step 3: Remove return and braces because we are dealing with a single line
const convertToLiters = gallons => gallons * 3.785;
```

Limitations of Arrow Functions

Arrow functions are not hoisted

Can not be used as constructor functions

Can not use the arguments object

They do not have their own “this” binding

Arrow Functions Can Not Be Used as Constructor Functions

```
const Aircraft = (model, capacity) => {
  this.model = model;
  this.capacity = capacity;
};

const boeing737 = new Aircraft('Boeing 737', 200);

// ERROR: Aircraft is not a constructor
```

Arrow Functions Can Not Access the Arguments Object

```
const sum = () => {
  let sum = 0;

  for (let i = 0; i < arguments.length; i++) {
    sum += arguments[i];
  }

  return sum;
};

console.log(sum(1, 2, 3));

// Error
```

Defining Functions - Arrow Function

Function Constructor

Function Constructor

Creates a new Function object from a list of arguments and a body provided as a string

Function Constructor

```
// Function constructor
const convertToLiters = new Function('gallons', 'return gallons * 3.785;');


// Equivalent to
const convertToLiters = gallons => gallons * 3.785;


// Function constructor without new
const convertToLiters = Function('gallons', 'return gallons * 3.785;');
```

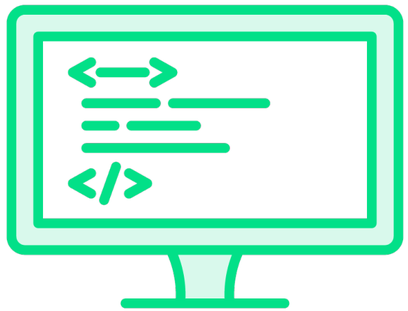
Signature

```
new Function(body)
new Function(arg0, body)
new Function(arg0, arg1, body)
new Function(arg0, arg1, /* … ,*/ argN, body)
```

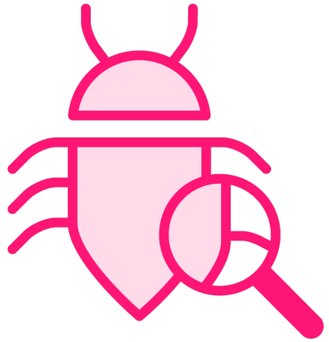
You

“Why on Earth would I ever use Function constructors?”

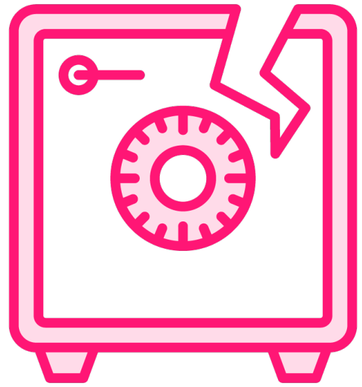
Pros and Cons of Function Constructor



Create functions on the fly



Code becomes prone to errors



Security issues

Defining Functions - Function Constructor

Recursion

A function in JavaScript can call itself – this is called recursion

Recursive Calls

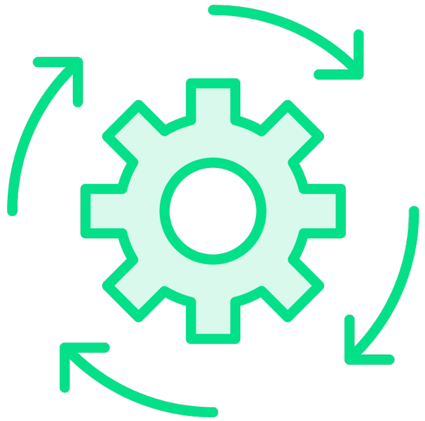
```
function decrementAltitude(alt) {
  console.log(`${alt} feet`);

  if (alt > 0) {
    decrementAltitude(alt - 100);
  }
}

decrementAltitude(500);
```

```
500 feet
400 feet
300 feet
200 feet
100 feet
0 feet
```

Structure of Recursive Functions



Call itself

Stop condition

Recursive Calls

```
function decrementAltitude(alt) {
  console.log(`${alt} feet`);

  if (alt > 0) {
    decrementAltitude(alt - 100);
  }
}

decrementAltitude(500);
```

Infinite Loop

```
function decrementAltitude(alt) {
  console.log(`${alt} feet`);
  decrementAltitude(alt - 100);
}

decrementAltitude(500);
```

```
Infinite loop

The program will eventually
break: Maximum call stack
size exceeded
```

IIFE (Immediately Invoked Function Expression)

IIFE

A function that automatically executes after its declaration

How to Create an IIFE

Declare a normal function

Wrap it in parentheses

Invoke it by using ();

IIFE Example

Normal function

```
function printDate() {
  let date = new Date().toLocaleDateString();
  console.log(date);
}

// To execute the function, you need to call it
printDate();
```

IIFE

```
(function printDate() {
  let date = new Date().toLocaleDateString();
  console.log(date);
})();


// Will print date on the screen
```

Immediately Invoked Function Expression

Summary

Functions are the building blocks of JS

JS is very flexible when declaring functions

- Function declaration
- Function expressions
- Arrow function
- Function constructor

The way you declare functions can impact their capabilities

Functions are not executed until they are invoked/called

If you need a function to execute immediately after creation, use an IIFE

Up Next:

Passing Data to Functions