Passing Data to Functions

Dan Geabunea

Passionate Software Developer

@romaniancoder | www.dangeabunea.com

Overview

- Understanding parameters and arguments
- Pass by value / Pass by reference
- Default parameters
- Rest parameters
- The arguments object
- Passing functions as arguments

Function Parameters

Names given to pieces of data that are provided to a function

Function Arguments

The actual values that are transmitted when calling a function. Inside a function, the value of arguments is read by using the names in the parameter list

```
function convertToLiters(gallons) {
    return gallons * 3.785;
}
let result = convertToLiters(10);
```

Parameters and Arguments

- “gallons” is a parameter for the function
- The value 10 is the argument provided to the function

Things to Consider

1

JavaScript functions do not have any data type information in their definition

2

JavaScript functions do not check the number of received arguments

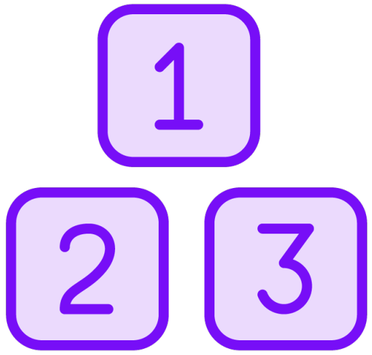
Parameters and Arguments

```
function drawPoint(x,y,color){
     // Implementation…
}


drawPoint(10,20); // color=undefined

drawPoint();      // x,y,z=undefined

drawPoint(10,20,'green',1.0);
```

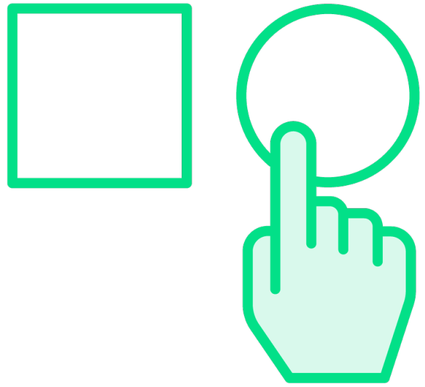
Passing Data to Functions

Pass by Value / Pass by Reference

Primitive & Reference Types



Primitive Types
Pass by value



Objects, Arrays & Functions
Pass by reference

```
let increment = function (num) {
    num = num + 1;
    return num;
};


let val = 10;
let newVal = increment(val);
console.log(val);         // Output is 10
console.log(newVal);      // Output is 11
```

Pass by Value

A copy of the value is passed as an argument
The original value remains unchanged

```
let flight = {id: 'RO3456', durationInMin: 120};


let delayFlight = function (flight, additionalDuration) {
  flight.durationInMin += additionalDuration;
};


delayFlight(flight, 20);
console.log(flight.durationInMin);     // Output is 140
```

Pass by Reference

A copy of the reference is passed as an argument
Changing the properties of an object or elements of an array will also be reflected outside

Default Parameters

Default Parameter

A mechanism that enables parameters to have a value different than undefined when no value is provided on function invocation

The Old Way

```
function offsetCoord(point, offset) {
  return {
    x: (point.x += offset),
    y: (point.y += offset),
  };
}

let res = offsetCoord({ x: 0, y: 0 });
// res = {x: NaN, y: NaN}
```

The Old Way

```
function offsetCoord(point, offset) {
  if (offset === undefined) {
    offset = 0;
  }

  return {
    x: (point.x += offset),
    y: (point.y += offset),
  };
}

let res = offsetCoord({ x: 0, y: 0 });
```

Using Default Parameters

```
function offsetCoord(point, offset = 0) {
  return {
    x: (point.x += offset),
    y: (point.y += offset),
  };
}
```

In JavaScript, every parameter can have e default value, regardless of its position in the parameter list

Pay Attention to Default Parameters

```
function calculateSum(a = 0, b, c = 0) {
  return a + b + c;
}

let sum1 = calculateSum();      // NaN
let sum1 = calculateSum(2);     // NaN
let sum1 = calculateSum(1,1);   // 2
```

Using Default Parameters

The “arguments” Object

The “arguments” Object

An array-like object present in all non-arrow functions that contains the values of the arguments passed to a function

MDN Web Docs: https://tinyurl.com/ppjyztw5

The “arguments” Object Is Array-like

```
function calculateSum() {
  console.log(arguments); // {0: 11, 1: 22, 2: 33}
}

calculateSum(11, 22, 33);
```

The “arguments” Object Is Array-like

```
function calculateSum() {
  console.log(arguments); // {0:11, 1:22, 2:33}

  // Array-like => we can use indexes & length
  let sum = 0;
  for (let i = 0; i < arguments.length; i++) {
    sum += arguments[i];
  }

  return sum;
}

calculateSum(11, 22, 33); // Returns 66
```

Not Usable in Arrow Functions

```
let calculateSum = () => {

  // Error: Will not work
  console.log(arguments); // {0:11, 1:22, 2:33}
  let sum = 0;
  for (let i = 0; i < arguments.length; i++) {
    sum += arguments[i];
  }

  return sum;
};
```

Avoid the “arguments” object and prefer rest parameters

Using the “arguments” Object

Rest Parameters

Rest Parameter

Allows functions to accept an indefinite number of arguments

MDN Web Docs: https://tinyurl.com/yeumxub8

Why Rest Parameters Are Important

```
// Two parameters
function calculateSum(a, b) {
  return a + b;
}

// We need to calculate sums for three, four, n parameters
```

The “Bad” Alternatives to Rest Parameters

1

Create multiple functions for each parameter list variation

2

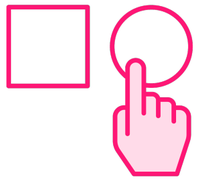
Don’t declare parameters and use the arguments object

Using Rest Parameters

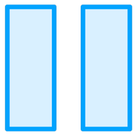
```
// Three full stop characters syntax
function calculateSum(...numbers) {
  let sum = 0;
  numbers.forEach((nb) => (sum += nb));

  return sum;
}

calculateSum(1, 1);
calculateSum(1, 2, 3);
calculateSum(1, 2, 3, 4);
```

Understanding Rest Parameter Rules

A rest parameter is introduced using the “...” notation and is transformed into an array for use inside the function



Only one parameter can be defined as a rest parameter



Only the last parameter can be defined as a rest parameter

Using Rest Parameters

Passing Functions as Arguments

Callback

A function passed as an argument to another function; They usually handle something when an operation is completed

Using “setTimeout”

```
setTimeout(() => console.log('Callback’), 1000);
```

Function That Accepts a Callback Parameter

```
let fp = { id: 'RO1234', dep: 'Paris', dst: '' };

function validateFlightPlan(fp, onErrorHandler) {
  if (fp.dst === '' || fp.dep === '') {
    // Invoke callback
    onErrorHandler();
  } else {
    console.log('VALID');
  }
}


validateFlightPlan(fp, () => console.log('EROOR'));
```

Function That Accepts a Callback Parameter

```
let fp = { id: 'RO1234', dep: 'Paris', dst: '' };

function validateFlightPlan(fp, onErrorHandler) {
  if (fp.dst === '' || fp.dep === '') {
    // Invoke callback
    onErrorHandler();
  } else {
    console.log('VALID');
  }
}

// Error handler
function logValidationError() {
  console.log('Validation Error');
}

validateFlightPlan(fp, logValidationError);
```

You

“When should I use callbacks?”

Callbacks

Summary

Primitive types are passed by value, objects, arrays & functions are passed by reference

Parameters have a default value of undefined

Default parameters allow you to specify a different initial value for parameters

The “arguments” object contains all the arguments passed into a non-arrow function

Rest parameters are a great way to pass an indefinite number of arguments to a function using a single parameter

Functions can accept other functions as arguments

Up Next:

Methods, Getters & Setters