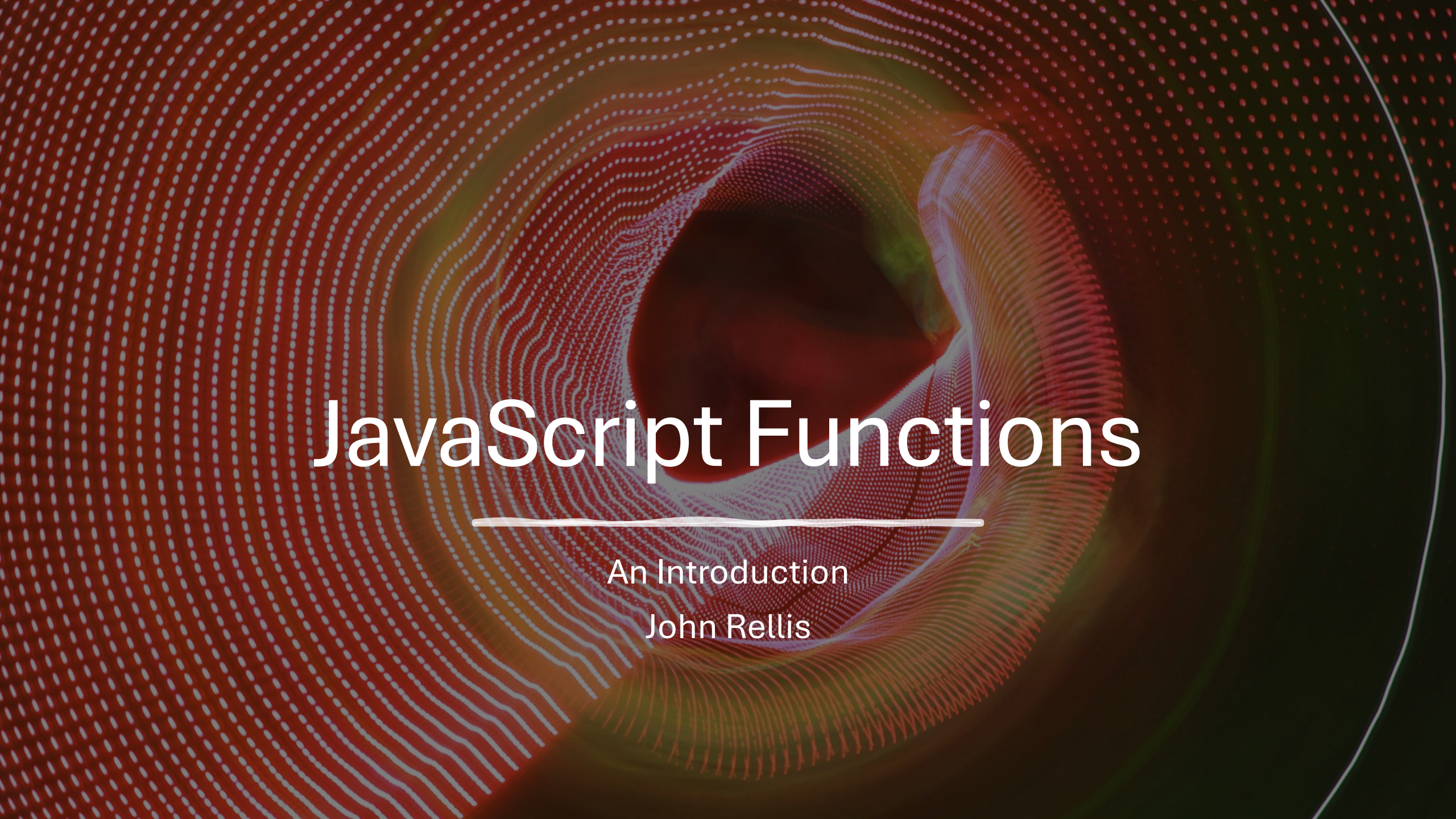


JavaScript Functions

An Introduction

John Rellis

What is a function?

- In mathematics, a function is a relation between a set of inputs and a set of possible outputs.
- $f(x) = 2x + 3$
  - x is the input, and the function returns an output value based on the input
  - You could say this as “the function f of x equals two times x plus three”
  - $f(0) = 2(0) + 3 = 3$
  - $f(1) = 2(1) + 3 = 5$

But we aren’t doing maths?

- In programming languages like JavaScript, functions serve a similar purpose.
- Functions encapsulate a set of instructions that can be executed multiple times with different inputs, just like mathematical functions.
- The term "function" was adopted in programming to reflect this concept of performing a specific task based on input values.

$$f(x) = 2x + 3$$

```
function f(x) {
  return 2 * x + 3;
}
```

```
console.log(f(0)); // Output: 3
console.log(f(1)); // Output: 5
console.log(f(-2)); // Output: -1
```

Or more simply...

JavaScript functions are reusable blocks of code that perform specific tasks when called.

```javascript
function add(a, b) {
  return a + b;
}

// Example usage:
console.log(add(3, 5)); // Output: 8
console.log(add(-2, 7)); // Output: 5
```

```javascript
function square(x) {
  return x * x;
}

// Example usage:
console.log(square(3)); // Output: 9
console.log(square(5)); // Output: 25
```

You can’t have functions without fun!

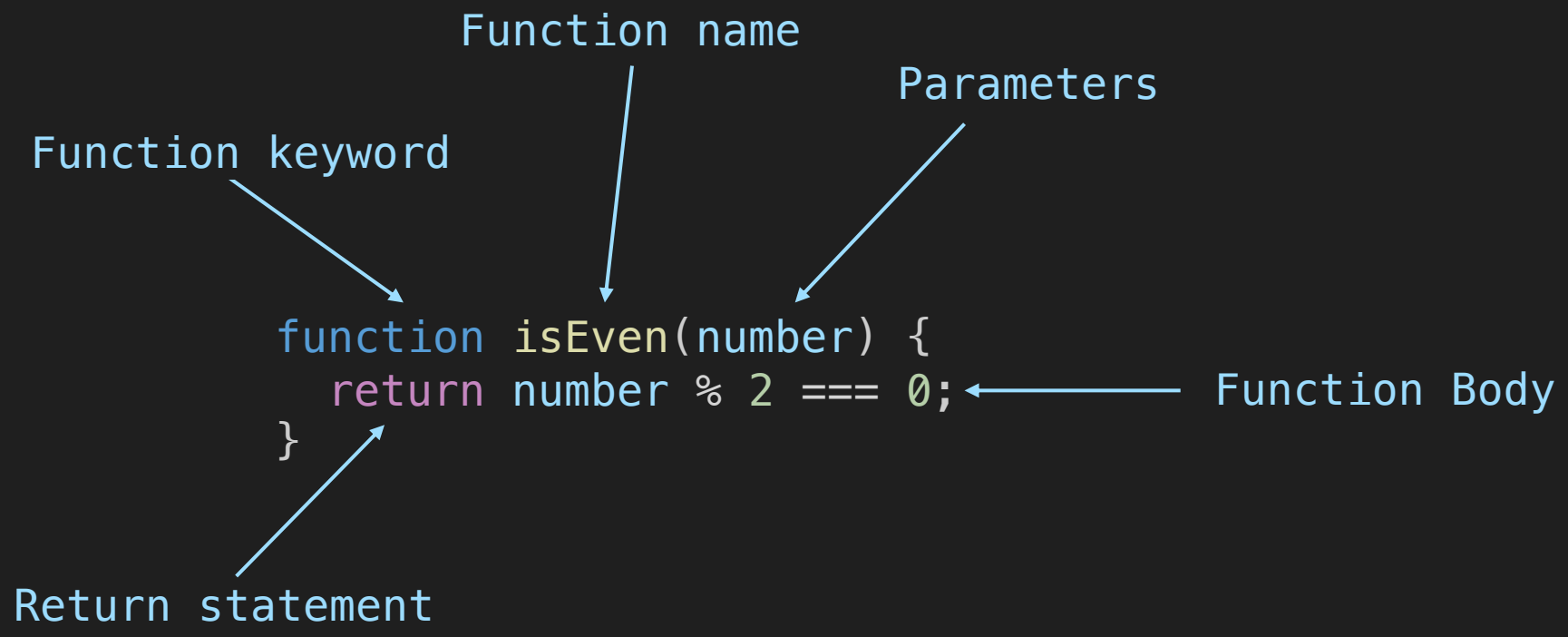
```javascript
function isEven(number) {
  return number % 2 === 0;
}

// Example usage:
console.log(isEven(4)); // true
console.log(isEven(7)); // false
```

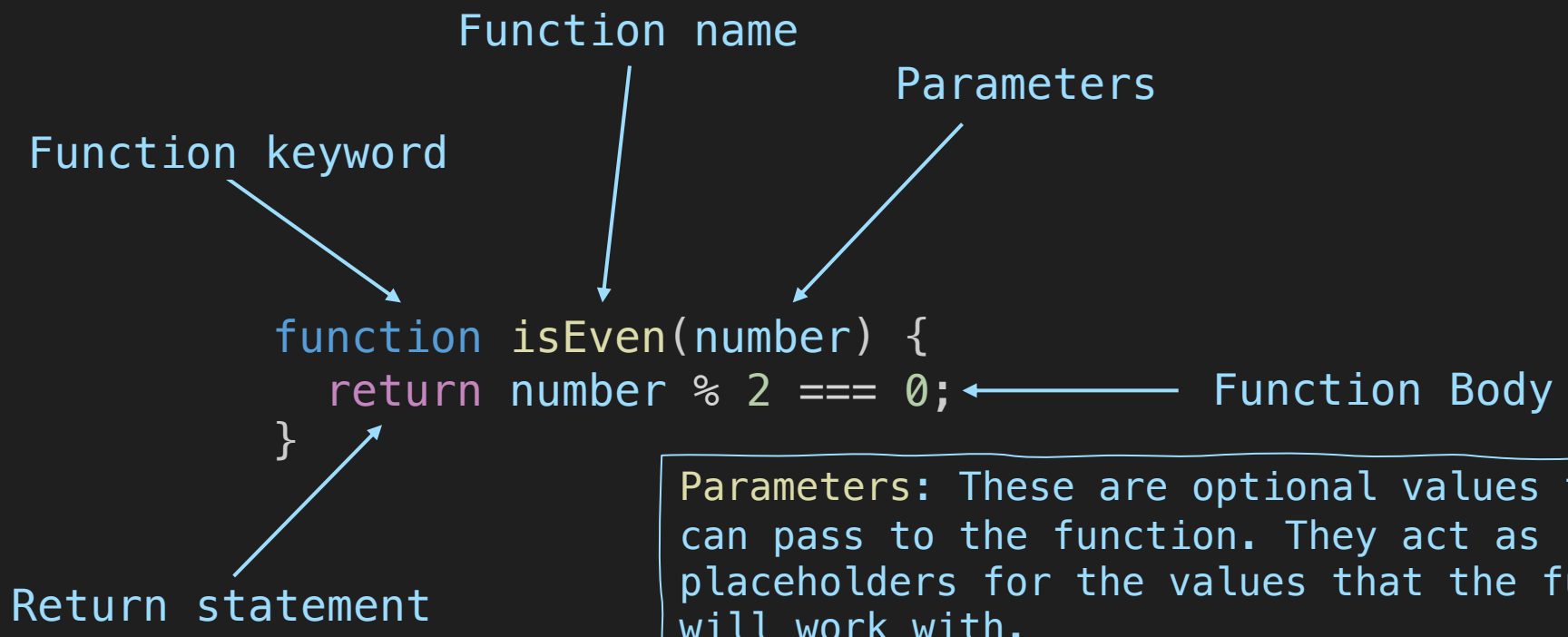
```javascript
function fahrenheitToCelsius(fahrenheit) {
  return (fahrenheit - 32) * 5 / 9;
}

// Example usage:
console.log(fahrenheitToCelsius(32)); // 0
console.log(fahrenheitToCelsius(68)); // 20
```

Anatomy of a Function



Anatomy of a Function



Function keyword: This is the keyword function that is used to declare a function in JavaScript.

Function name: This is an optional identifier for the function. It allows you to give a name to the function so that you can refer to it later in your code.

Parameters: These are optional values that you can pass to the function. They act as placeholders for the values that the function will work with.

Function body: This is a block of code enclosed within curly braces {}. It contains the set of instructions that define what the function does when it is called.

Return statement: This is optional and is used to specify the value that the function should return when it is called. The return statement terminates the function and passes a value back to the caller.

Function Name is Optional?

Named function

```
function isEven(number) {
  return number % 2 === 0;
}

const result = isEven(3);

console.log(result); // false
```

Anonymous function assigned to variable

```
const isEven = function(number) {
  return number % 2 === 0;
}

const result = isEven(3);

console.log(result); // false
```

Invoking or Executing a Function

The Function body is not executed until a Function in invoked

Function name

Parameters

```
function isEven(number) {
  return number % 2 === 0;
}
```

Return value assigned to variable

3 is the argument we are passing to the number parameter

```
const isThreeEven = isEven(3);
```

A function is invoked or called or executed using the function name immediately following by () with a list of arguments

Arguments or Parameters

In JavaScript, the terms "parameters" and "arguments" are often used interchangeably, but they do have slightly different meanings:

Parameters are the placeholders defined in the function declaration.

They act as **variables** that represent the values that will be passed to the function when it's called.
In the function definition function *add(a, b)*, a and b are the parameters.

Arguments are the actual values passed to the function when it's called.
In the function **call** *add(3, 5)* , 3 and 5 are the arguments.

Parameters

```
function isEven(number) {
  return number % 2 === 0;
}
```

3 is the argument we are passing to the number parameter

```
const isThreeEven = isEven(3);
```

Optional Arguments

- In JavaScript, arguments are optional
- This means you can't really overload functions like in Java
- But it does allow you to put optional parameter at the end
- This will require more checks and testing in your code

```
function greet(name) {
  if (name) {
    console.log(`Hello, ${name}!`);
  } else {
    console.log('Hello, there!');
  }
}

greet(); // Output: Hello, there!
greet('John'); // Output: Hello, John!
```

Parameters

```
function isEven(number) {
  return number % 2 === 0;
}
```

No arguments

```
const isThreeEven = isEven();
```

The result here is false, can you figure out why?

When do we typically use functions?

In Objects as methods
All methods are functions, not all functions are methods

```
const car = {
  brand: "ford",
  model: "focus",
  beep: function(){
      console.log("beep")
    }
}

car.beep() // prints: beep
```

Method Shorthand Syntax

```
const car = {
  brand: "ford",
  model: "focus",
  beep(){
      console.log("beep")
    }
}

car.beep() // prints: beep
```

When do we typically use functions?

Iterating Arrays

```
const colours = ["red", "green", "blue"];

colours.forEach(function(colour){
  console.log(`My favourite colour is ${colour}`)
});
```

Note that this is the “old” way of iterating, read on for the new way

```
function calculateTotalCost(items) {
  let total = 0;
  // Iterate through each item in the shopping cart
  items.forEach(function(item) {
    total += item.price; // Add the price of each item to the total
  });
  return total;
}

// Define a function to apply a discount to the total cost
function applyDiscount(total, discountPercentage) {
  // Calculate the discount amount
  const discountAmount = total * (discountPercentage / 100);
  // Apply the discount to the total cost
  const discountedTotal = total - discountAmount;
  return discountedTotal;
}

// Sample shopping cart items
const shoppingCart = [
  { name: 'Shirt', price: 20 },
  { name: 'Pants', price: 30 },
  { name: 'Shoes', price: 50 }
];

// Calculate the total cost of items in the shopping cart
const totalCost = calculateTotalCost(shoppingCart);

// Apply a 10% discount to the total cost
const discountedTotal = applyDiscount(totalCost, 10);

console.log('Total cost before discount: $' + totalCost);
console.log('Total cost after 10% discount: $' + discountedTotal);
```

```
Total cost before discount: $100
Total cost after 10% discount: $90
```

Recap

- Function are executable blocks of code
- You can declare a function and use it later, you can even pass a function around your program as a parameter
- Functions are useful for promoting reuse, encapsulation and sometimes just simply improving readability
- You can make your code tell a story that is easily understandable

```
function tellStory() {
  setScene();
  introduceCharacter("Sir Lancelot");
  describeProblem("a fierce dragon");
  describeQuest("to slay the dragon and save the kingdom");
  narrateJourney();
  concludeStory();
}
tellStory();
```


Great……
But there’s more….


Arrow Functions!

Arrow Functions

- An arrow function expression is a compact alternative to a traditional function expression, with some semantic differences and deliberate limitations in usage:
  - Arrow functions don't have their own bindings to this, arguments, or super, and should not be used as methods.
  - Arrow functions cannot be used as constructors. Calling them with new throws a TypeError. They also don't have access to the new.target keyword.
  - Arrow functions cannot use yield within their body and cannot be created as generator functions.

Arrow Functions

If you are new to computer science and JavaScript, a lot of that won't make sense.

But for now...

- Arrow Functions are a new syntax whose semantic properties make them ideal for usage as functions passed to iterators

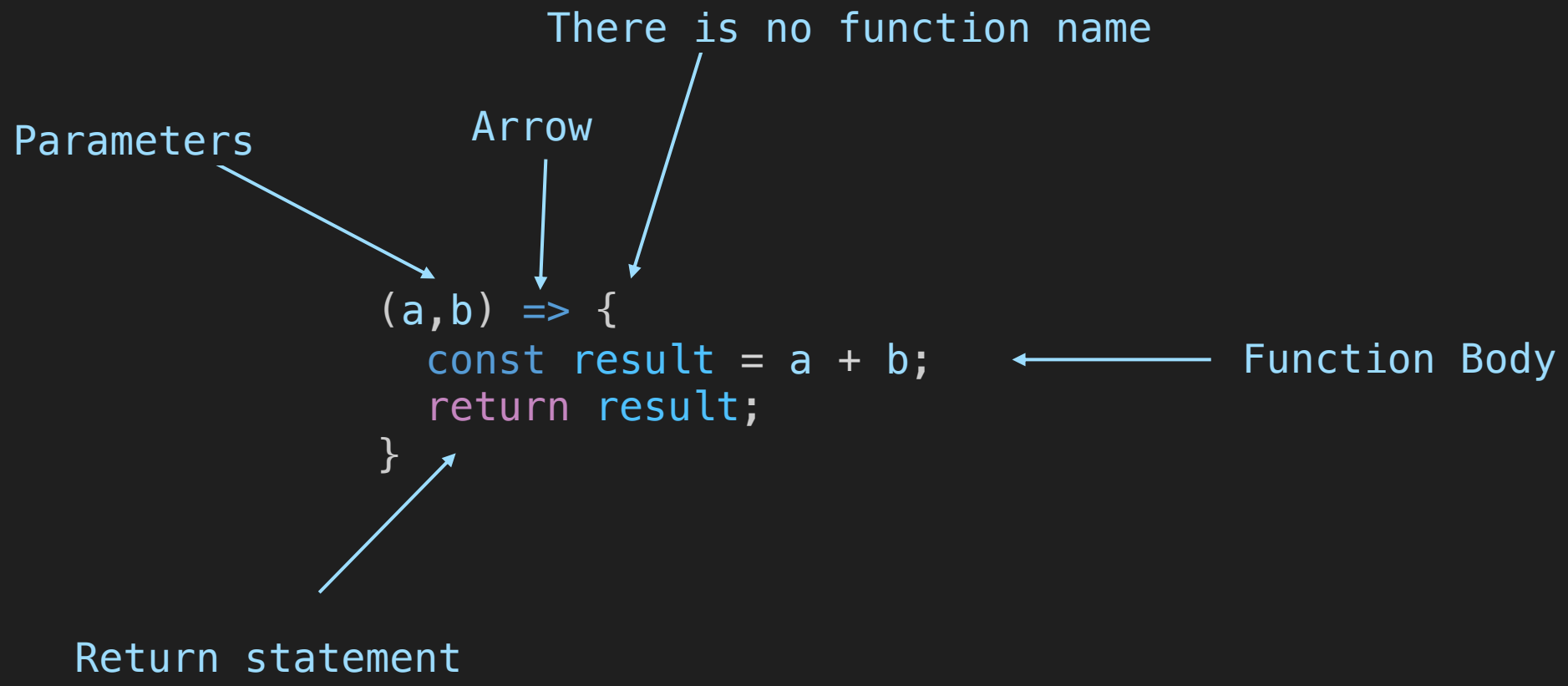
```
const colours = ["red", "green", "blue"];

colours.forEach(function(colour){
  console.log(`My favourite colour is ${colour}`)
});
```

```
const colours = ["red", "green", "blue"];

colours.forEach(colour => console.log(`My favourite colour is ${colour}`));
```

Anatomy of an Arrow Function



Invoking an Arrow Function

```
const add = (a,b) => {
    const result = a + b;
    return result;
}

add(2,3);
```

Single expression Arrow Functions

```
const add = (a,b) => {
    const result = a + b;
    return result;
}

add(2,3);
```

No return or {} needed

```
const add = (a,b) => a + b;

const result = add(2,3)
```

```
const square = a => a * a;

square(2);
```

Single parameter, no ()

Both examples of shorthand syntax are optional, when starting out it is usually easier to stick to the long versions. Sometimes your IDE will auto-convert to the shorter syntax and confuse you quite a bit ☺

The Array map function

- Array.map applies a function to each item in an array
- It then maps the **return** value of the function to the same index of in completely new array

```
const words = ['metal', 'rock', 'folk']

const wordToDescription = (word) => {
  return `The word "${word}" has ${word.length} letters`
}

const descriptions = words.map(wordToDescription)
```

```
[
  'The word "metal" has 5 letters',
  'The word "rock" has 4 letters',
  'The word "folk" has 4 letters'
]
```

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Map

The Array map function

```
const words = ['metal', 'rock', 'folk']

const wordToDescription = (word) => {
  return `The word "${word}" has ${word.length} letters`
}

const descriptions = words.map(wordToDescription)
```

- The function is typically declared within the parentheses

```
[
  'The word "metal" has 5 letters',
  'The word "rock" has 4 letters',
  'The word "folk" has 4 letters'
]
```

```
const descriptions = words.map((word) => {
  return `The word "${word}" has ${word.length} letters`
})
```

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Map

The Array map function

```
const words = ['metal', 'rock', 'folk']

const wordToDescription = (word) => {
  return `The word "${word}" has ${word.length} letters`
}

const descriptions = words.map(wordToDescription)
```

- This can be shortened further from:

```
const descriptions = words.map((word) => {
  return `The word "${word}" has ${word.length} letters`
})
```

```
[
  'The word "metal" has 5 letters',
  'The word "rock" has 4 letters',
  'The word "folk" has 4 letters'
]
```

- To a one-line assignment:

```
const descriptions = words.map(word => `The word "${word}" has ${word.length} letters`)
```

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Map

The Array forEach function

- Array.forEach runs a function for each item in an array
- Unlike “map” it has no return value

```
const words = ['metal', 'rock', 'folk']

const descriptions = [] // we use forEach to populate one by one

words.forEach(word => descriptions.push(`The word "${word}" has ${word.length} letters`))

console.log(descriptions)
```

```
[
  'The word "metal" has 5 letters',
  'The word "rock" has 4 letters',
  'The word "folk" has 4 letters'
]
```

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/forEach

To map or forEach

- Use map when you want to **transform** each item in an array into a new array, e.g. square every number in an array.
- Use forEach when you want to use each item in an array to act upon another piece of data e.g. add every item of fruit into a shopping basket

```
const words = ['metal', 'rock', 'folk']

const uppercaseWords = words.map(word => word.toUpperCase());
// new array with ["METAL", "ROCK", "FOLK"]
```

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/forEach

find and filter

- Use Array.find to find the **FIRST ELEMENT** that matches a condition
- Use Array.filter to find **ALL THE ELEMENTS** as an array that match a condition

```
const numbers = [5, 12, 8, 130, 44];

const foundNumber = numbers.find((element) => element > 10); // 12

const foundNumbers = numbers.filter((element) => element > 10); // [12, 130, 44]
```

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/find
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/filter

Recap

- Functions are blocks of code that should have an input and produce an output based on that input.
- Functions are useful for reusing blocks of code resulting in code that is easier to maintain
- When a function is used as a value of an object in a property, we can call it a ”method”. (Remember there’s a shorthand syntax)
- We have classical functions and arrow functions
- Arrow functions are well suited to be used as iterator functions in the likes of Array.map and Array.forEach

Further Reading

- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions