Diference between Normal function & Arrow function

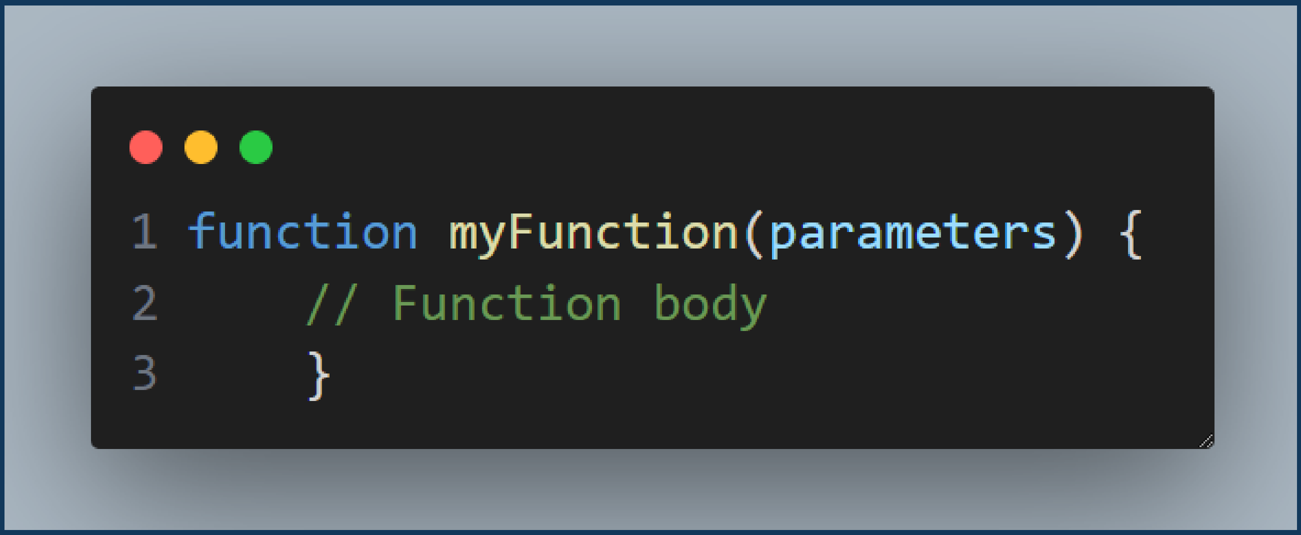
SWIPE

Normal and Arrow

Normal functions and arrow functions are two different ways of defining functions in JavaScript, each with its own syntax and behavior. Here are the main differences between them:

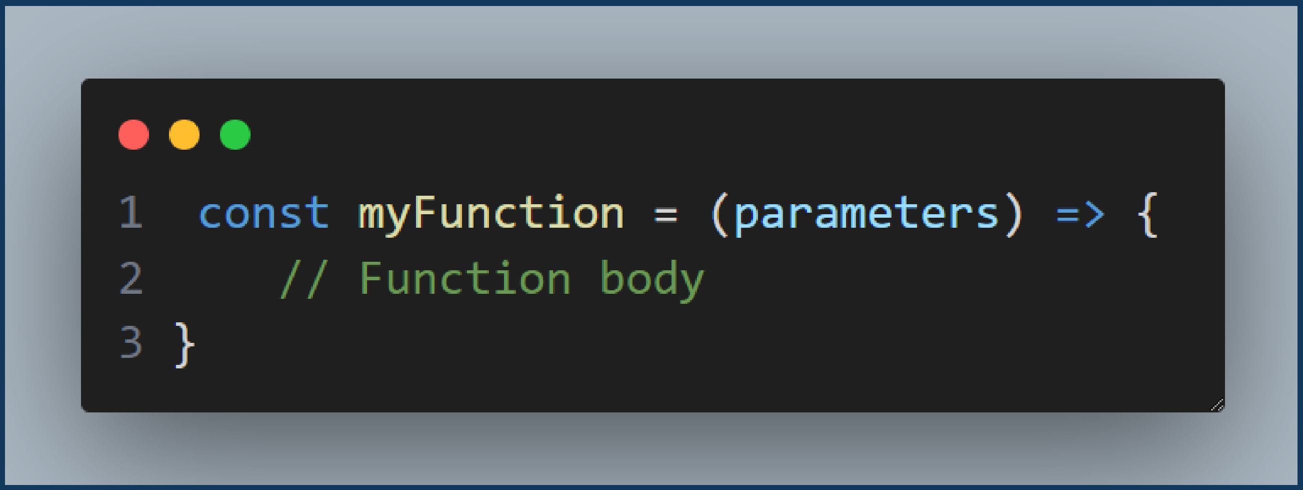
SYNTAX:

NORMAL FUNCTION:



```
function myFunction(parameters) {
    // Function body
    }
```

ARROW FUNCTION:



```
const myFunction = (parameters) => {
    // Function body
}
```

KEEP SWIPING

‘this’ Binding:

- Normal functions have their own **‘this’** context, which is determined by how they are called. The **‘this’** keyword inside a normal function refers to the object that called the function.
- Arrow functions do not have their own **‘this’** context. Instead, they inherit the **‘this’** value from the surrounding lexical context when the function is defined. This behaviour can be particularly useful when dealing with event handlers or callbacks, as it eliminates the need to use **‘bind()’** or store **‘this’** in a variable.

Implicit Return:

- Arrow functions have implicit return. If the function body consists of a single expression, it will be implicitly returned without needing the **‘return’** keyword.
- Normal functions require the return keyword to explicitly **‘return’** a value.

‘arguments’ Object:

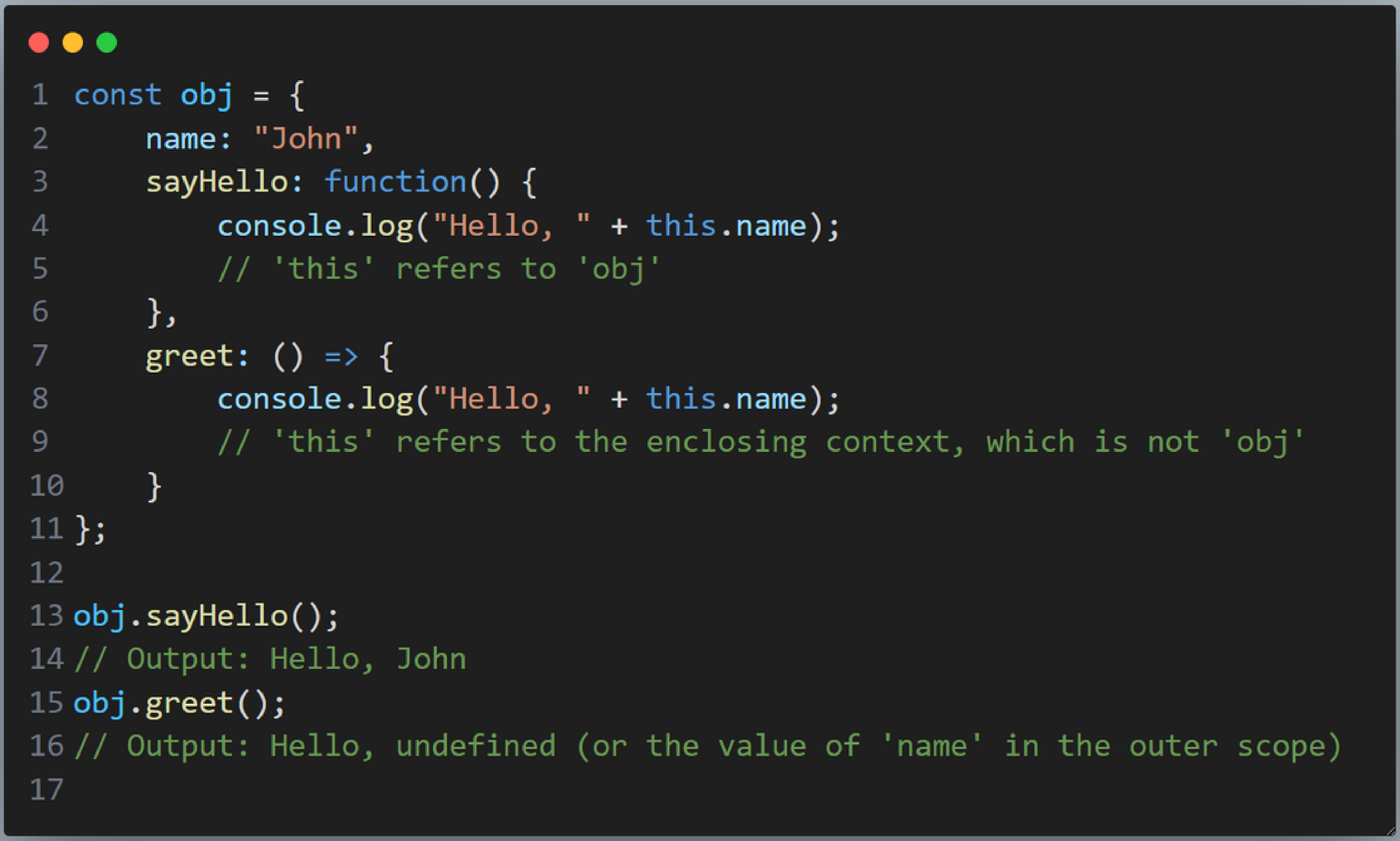
- Normal functions have access to the **‘arguments’** object, which contains all the arguments passed to the function.
- Arrow functions do not have their own **‘arguments’** object. Instead, they inherit the arguments object from the enclosing scope.

ONE MORE

‘new’ Keyword:

- Normal functions can be used as constructor functions with the new keyword to create **‘new’** objects.
- Arrow functions cannot be used as constructor functions and will throw an error if attempted to be used with **‘new’**.

Here's a quick example demonstrating some of these differences:



```
const obj = {
    name: "John",
    sayHello: function() {
        console.log("Hello, " + this.name);
        // 'this' refers to 'obj'
    },
    greet: () => {
        console.log("Hello, " + this.name);
        // 'this' refers to the enclosing context, which is not 'obj'
    }
};

obj.sayHello();
// Output: Hello, John
obj.greet();
// Output: Hello, undefined (or the value of 'name' in the outer scope)

```

In this example, **‘sayHello()’** is a normal function and **‘greet()’** is an arrow function. You can see how this behaves differently in each case.

COMMENT BELOW