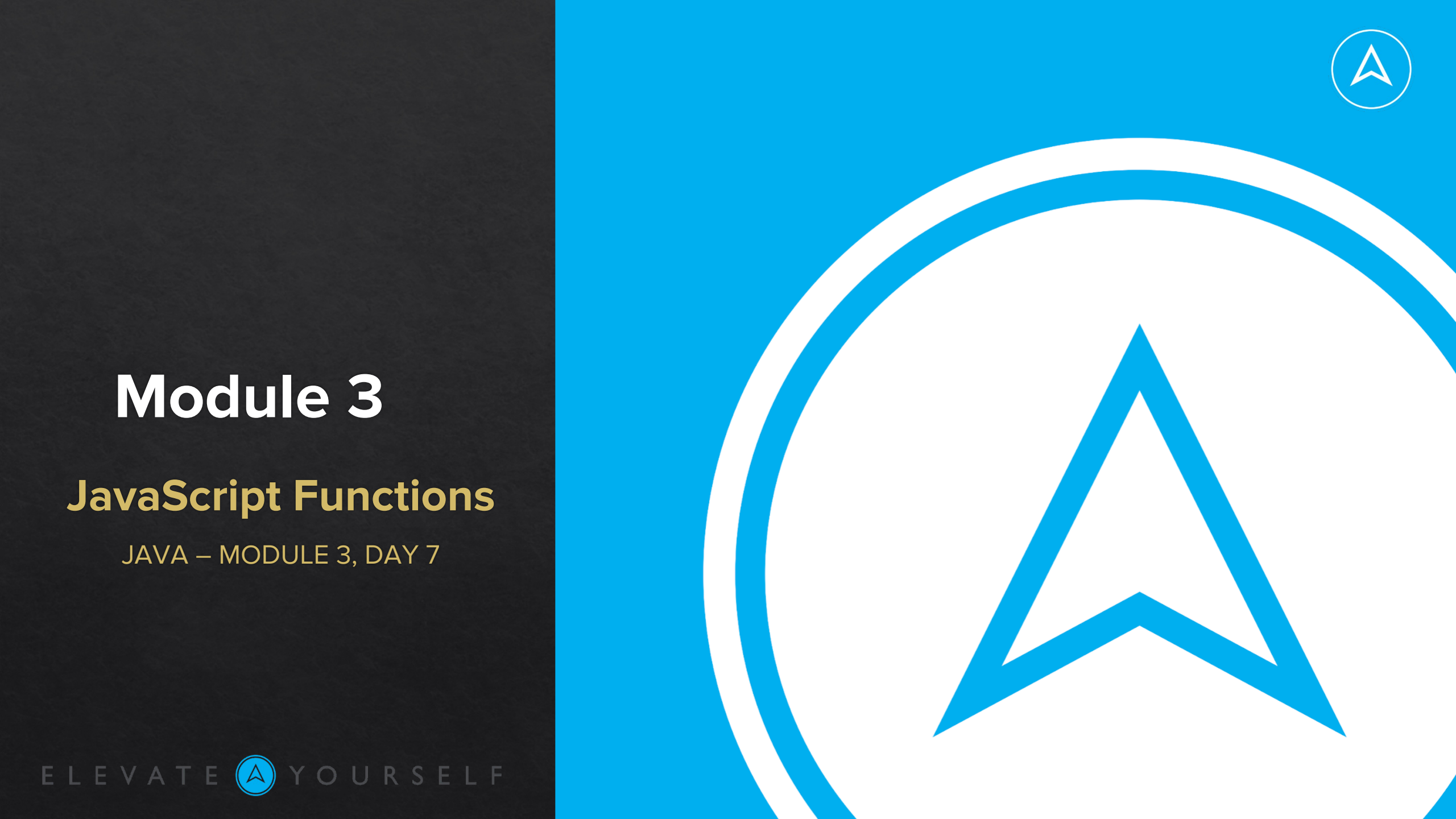
Module 3

JavaScript Functions

JAVA – MODULE 3, DAY 7



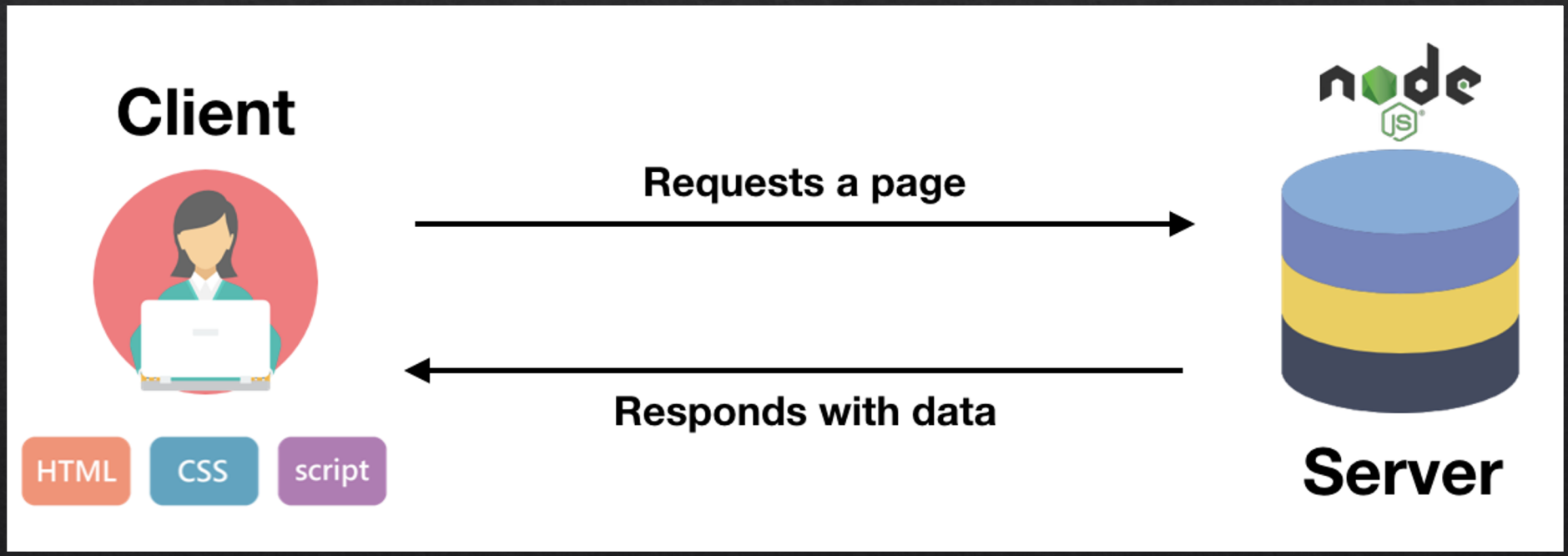
ELEVATE YOURSELF

Review

RECAP

Review

WHY JAVASCRIPT?



Review

JAVA vs JAVASCRIPT

- Runs on the Server-Side
- Compiled
- Must be syntactically correct or will not run

- Runs in the Browser
- Interpreted
- Does not have to be syntactically correct to run

DATA TYPES

Some of the more important JavaScript data types to be familiar with are:

- const numberOfDaysInWeek = 7; //Number
- const favoriteDessert = "Bread Pudding"; //String
- const person = {name: "Chewbacca", age: 876}; //Object
- const isFriendsWithHanSolo = true; //Boolean
- const logFunction = console.log; //function
- null
- Undefined

In JavaScript, variables aren't associated with any particular data type when you declare them. This makes JavaScript a loosely typed language.

JS OBJECTS

```
1 const myCar = {
2     make: "Tesla",
3     model: "Model 3",
4     year: 2020,
5     color: "Black",
6 };
```

Review

JS IS LOOSELY TYPED

```
let myVariable = 42;

myVariable = 'JavaScript is kind of like the multiverse.';

myVariable = {
  js: "Even lets you",
  assign: "objects into things that",
  stored: "other data types just before"
};

myVariable = ["And", "then", "use", "arrays"];

myVariable = null;

myVariable = function someFunction() {
  console.WriteLine("You can even assign functions to variables");
};

myVariable(); // And call them!
```

== VS ===

Equals (==)

- 1 == 1
- 1 == "1"

Strictly Equals (===)

- 1 === 1
- 1 !== "1"

TRUTHY

Truthy values
‘false’ (quoted false)
‘0’ (quoted zero)
( ) (empty functions)
[ ] (empty arrays)
{ } (empty objects)
All other values

Falsy values
false
0 (zero)
‘ ‘ (empty string)
null
undefined
NaN

In JavaScript, a **truthy** value is a value that is considered true when encountered in a Boolean context. All values are truthy unless they are defined as falsy (i.e., except for false, 0, -0, 0n, "", null, undefined, and NaN).

ex... if (25)

TRUTHY

```
1 const result = calculate(); // Might be null or undefined
2
3 if (result) {
4   // Wasn't null / undefined
5 } else {
6   // Was null, undefined, or otherwise falsey
7 }
8
9 // Alternative way of writing this could be:
10 if (result !== null && result !== undefined) {
11   // ...
12 }
```

Review

IF STATEMENTS

```
1 if (user.currentStatus == 'OnFire') {
2   user.Stop();
3   user.Drop();
4   user.Roll();
5 } else {
6   user.conductBusinessAsUsual();
7 }
```

FOR LOOPS

```
let sum = 0; // the sum of all our scores

for(let i = 0; i < testScores.length; i++)
{
    sum = sum + testScores[i];  // add each score to the sum
}

const average = sum / testScores.length;
```

Review

FUNCTIONS

```
1 function addNumbers(x, y) {
2   const sum = x + y;
3   return sum;
4 }
5
6 const result = addNumbers(60, 6);
```

ARRAYS

Declaring and initializing an array

```
const testScores = [];
```

```
const testScores = [ 85, 96, 80, 98, 89, 70, 93, 84, 66, 96 ];
```

Determining the length of an array

```
const size = testScores.length;
```

Accessing elements within an array

```
const testScores = [ 85, 96, 80, 98, 89, 70, 93, 84, 66, 96 ];

testScores[0] = 82; // update the value at index 0 to 82
testScores[1] = 72; // update the value at index 1 to 72
testScores[4] = 80; // update the value at index 4 to 80

const highScore = testScores[3]; //set highScore to 98
```

ARRAY FUNCTIONS

- myArray[**42**] - Java style array index
- **Push / Pop** (from the end of the array)
- **Unshift / Shift** (from the beginning of the array)
- **Splice** - Remove and/or add elements from an array
- **Slice** - Clone all or part of an array
- **Concat** - Create a new array from other arrays

AGENDA

- Functions
- Anonymous Functions
- Array Functions
- Function Documentation

FUNCTIONS

To write maintainable code, one of the primary things to avoid is code duplication. Programming languages provide many features and constructs to achieve this, and one of the most common is functions.

In the context of programming, a "function" is a way to package up a block of code, allowing you to reuse that block over and over again. This is especially helpful when you have a piece of logic that's needed in more than one place in a system.

There are two types of functions that you'll learn about in JavaScript: named functions and anonymous functions.

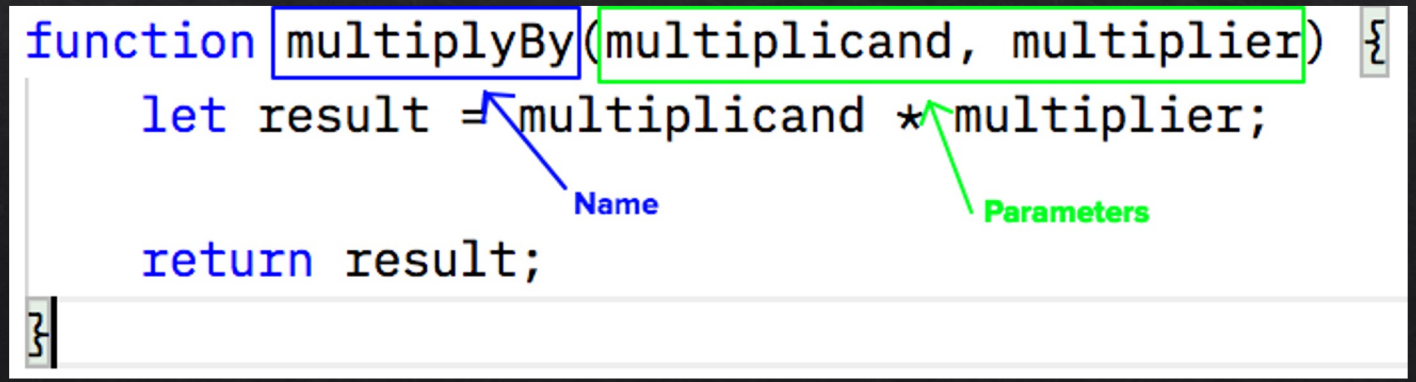
NAMED FUNCTIONS

Named functions

Two parts of a named function: the **function signature** and the **function body**.

The components of a function signature are:

- The function name
- The function parameters



Lecture

NAMED FUNCTIONS

Function name
Like variables, functions have names that can be used to reference them. Also, like variable names, careful consideration should be given to choosing names for functions. Function names should be:

- **descriptive** - it should be clear what type of action or calculation the function performs when invoked
- **camelCase** - the first letter of the name is lowercase and the first letter of each subsequent word is uppercase
- **unique** - function names need to be unique across all JavaScript code that's loaded into the page. If a name conflicts with another function, the one that's loaded last overwrites the other one

```
function addTwoNumbers(x, y){
  //Logic
  let result = x + y;
  return result;
}
```

FUNCTION PARAMETERS

Function parameters
Parameters are variables that can provide input values to a function. When functions are created, parameter lists indicate what inputs the function can accept.

Optional parameters
Parameters in JavaScript are always optional. So what if a value isn't provided for a parameter when calling a function? if you don't assign a value to a variable, the variable is set to undefined.

Default parameter values
In some cases, there's a reasonable default value that you can use if a parameter value isn't supplied. You can use default parameters to make your functions more robust and useful in varying scenarios.

FUNCTION PARAMETERS

Lecture

Handling an unknown number of parameters

There may be times when you want to handle an unknown number of parameters. A good example of this is writing a function that concatenates an unknown number of strings together. If you call a function like this:

```
let name = concatAll(firstName, lastName);
```

What if you want/need to call concatAll like the following:

```
let name = concatAll(honorific, firstName, mothersMaidenName,
 '-', fathersLastName);
```

Lecture

UNKNOWN PARAMETERS

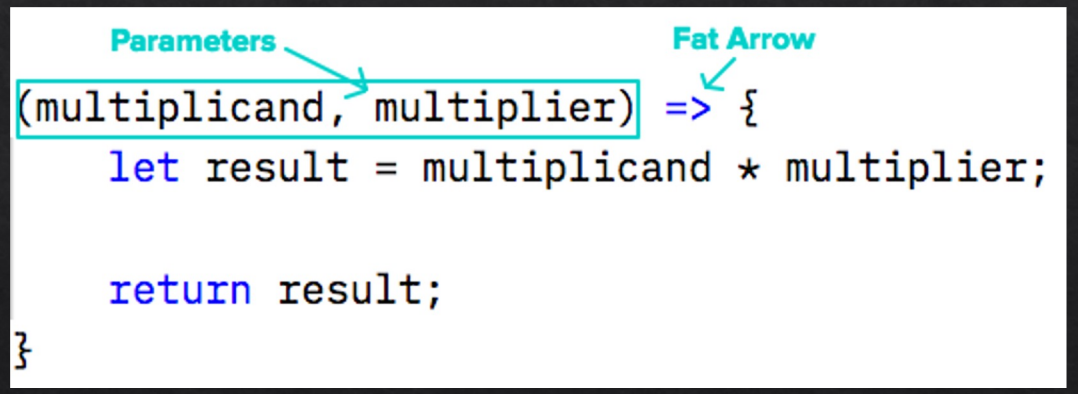
Handling an unknown number of parameters
The issue here is that there's no way to know how many parameters you have for that function. But in JavaScript, you can use a special variable to get at all of the given parameters, whether you expect them or not. This variable is called arguments.

With arguments, you can treat all parameters that have been passed to the function as an array, even if you don't have any parameters defined in the actual function definition.

```
function concatAll() { // No parameters defined, but we still might get some
  let result = '';
  for(let i = 0; i < arguments.length; i++) {
      result += arguments[i];
  }
  return result;
}
```

ANONYMOUS FUNCTIONS

Anonymous functions are functions that don't have a name. Functions in JavaScript can be used like any other value, so creating a function without a name is possible. You create an anonymous function with the following syntax:



Lecture

ALTERNATIVE SYNTAX TO FUNCTIONS

Lecture

NAMED FUNCTIONS

```
1 const multiplyByTwo = function multTwo(num) {
2   return num * 2;
3 }
4
5 const answer = multiplyByTwo(21);
```

Lecture

ANONYMOUS FUNCTIONS

```
1 const multiplyByTwo = function(num) {
2   return num * 2;
3 }
4
5 const answer = multiplyByTwo(21);
```

Older style

ANONYMOUS FUNCTIONS

Arrow Syntax

```
1 const multiplyByTwo = (num) => {
2   return num * 2;
3 }
4
5 const answer = multiplyByTwo(21);
```

Lecture

ANONYMOUS FUNCTIONS

Arrow Syntax

```
1 const multiplyByTwo = (num) => num * 2;
2
3 const answer = multiplyByTwo(21);
```

ANONYMOUS FUNCTIONS

★ Arrow Syntax

```
1 const multiplyByTwo = (num) => {
2   return num * 2;
3 }
4
5 const answer = multiplyByTwo(21);
```

ARRAY FUNCTIONS

Array functions using anonymous functions

Arrays in JavaScript have many useful functions themselves that use anonymous functions.

forEach()

Performs like a for loop, running a passed in anonymous function for every element of an array.

```
let numbers = [1, 2, 3, 4];

numbers.forEach( (number) => {
    console.log(`This number is ${number}`);
});
```

ARRAY FUNCTIONS

find()

The find() method returns the value of the first element in the provided array that satisfies the provided testing function. If no values satisfy the testing function, undefined is returned.

```
1 const array1 = [2, 12, 16, 181, 59];
2
3 const found = array1.find(element => element > 14);
4
5 console.log(found);
6 // expected output: 16
7
```

ARRAY FUNCTIONS

findIndex()

The findIndex() method returns the index of the first element in the array that satisfies the provided testing function. Otherwise, it returns -1, indicating that no element passed the test.

```
1 const array1 = [2, 9, 15, 49, 44];
2
3 const isLargeNumber = (element) => element > 13;
4
5 console.log(array1.findIndex(isLargeNumber));
6 // expected output: 2
7
```

ARRAY FUNCTIONS

filter()

The filter() method creates a new array with all elements that pass the test implemented by the provided function.


```
1 const words = ['Neo', 'Morpheus', 'Matrix', 'Trinity', 'Apoc', 'Switch'];
2
3 const result = words.filter(word => word.length > 6);
4
5 console.log(result);
6 // expected output: Array ["Morpheus", "Trinity"]
7
```

ARRAY FUNCTIONS

map()

The map() method creates a new array populated with the results of calling a provided function on every element in the calling array.

```
1 const array1 = [1, 4, 9, 16];
2
3 // pass a function to map
4 const map1 = array1.map(x => x * 2);
5
6 console.log(map1);
7 // expected output: Array [2, 8, 18, 32]
8
```

ARRAY FUNCTIONS

reduce()

The reduce() method executes a user-supplied “reducer” callback function on each element of the array, passing in the return value from the calculation on the preceding element. The final result of running the reducer across all elements of the array is a single value.

```
1  const array1 = [1, 2, 3, 4];
2  const reducer = (previousValue, currentValue) => previousValue + currentValue;
3
4  // 1 + 2 + 3 + 4
5  console.log(array1.reduce(reducer));
6  // expected output: 10
7
8  // 5 + 1 + 2 + 3 + 4
9  console.log(array1.reduce(reducer, 5));
10 // expected output: 15
11
```

JS DOC

Documentation

One of the core responsibilities of a programmer is writing documentation for the code they create. Documentation is more than comments on the code, and there are a lot of comments that are considered bad practice.

Line Comments

```
// Set number of phones to one
let number = 1;
```

```
// Set number of phones to one
let numberOfOwnedPhones = 3;
```

✓

```
let numberOfOwnedPhones = 3;
```

← Self-Documenting Code

✓

```
// Needed later to build the display table
let numberOfOwnedPhones = 3;
```

Lecture

Lecture

JS DOC

Documentation

Function Comments – JSDoc

Comments on functions are typically integrated into the IDE and are used to create documentation of your code for other programmers to use. They follow a standard format called JSDoc.

```
/**
 * Takes two numbers and returns the product of
 * those two numbers.
 *
 * Will return NaN if exactly two numbers are not
 * given.
 *
 * @param {number} multiplicand a number to multiply
 * @param {number} multiplier a number to multiply by
 * @returns {number} product of the two parameters
 */
```

Lecture

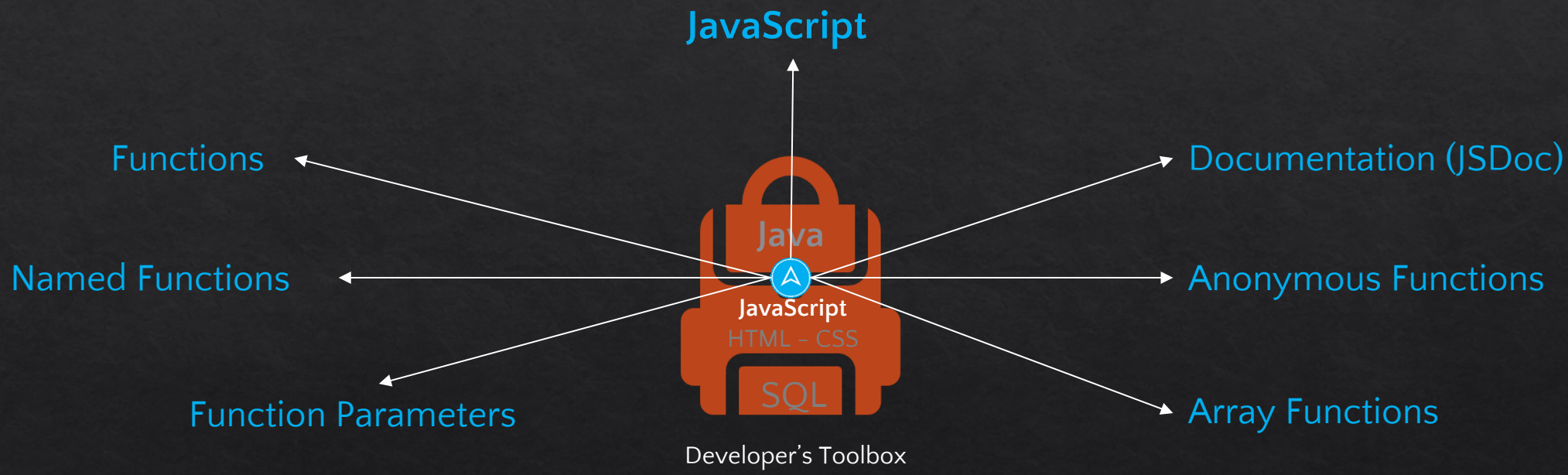
REFERENCES

- Bootcamp OS
- MDN Web Docs
  https://developer.mozilla.org/

Module 3

JavaScript Functions

New Tools! Let's Code!



Exercise HW