JavaScript Functions

Today's Objectives

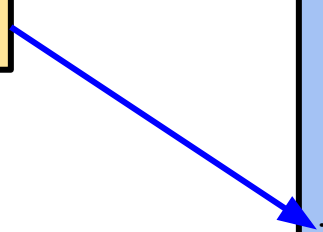
- **Named Functions**
- **Function Parameters**
  - Optional Parameters
  - Parameter Default Values
  - arguments variable
- **Anonymous Functions**
- **Array Functions**
- **Function Documentation**

JavaScript Functions

- JavaScript allows us to write small pieces of reusable code called functions.
- Functions are similar to methods in Java.
- JavaScript has two types of functions:
  - Named functions
  - Anonymous functions

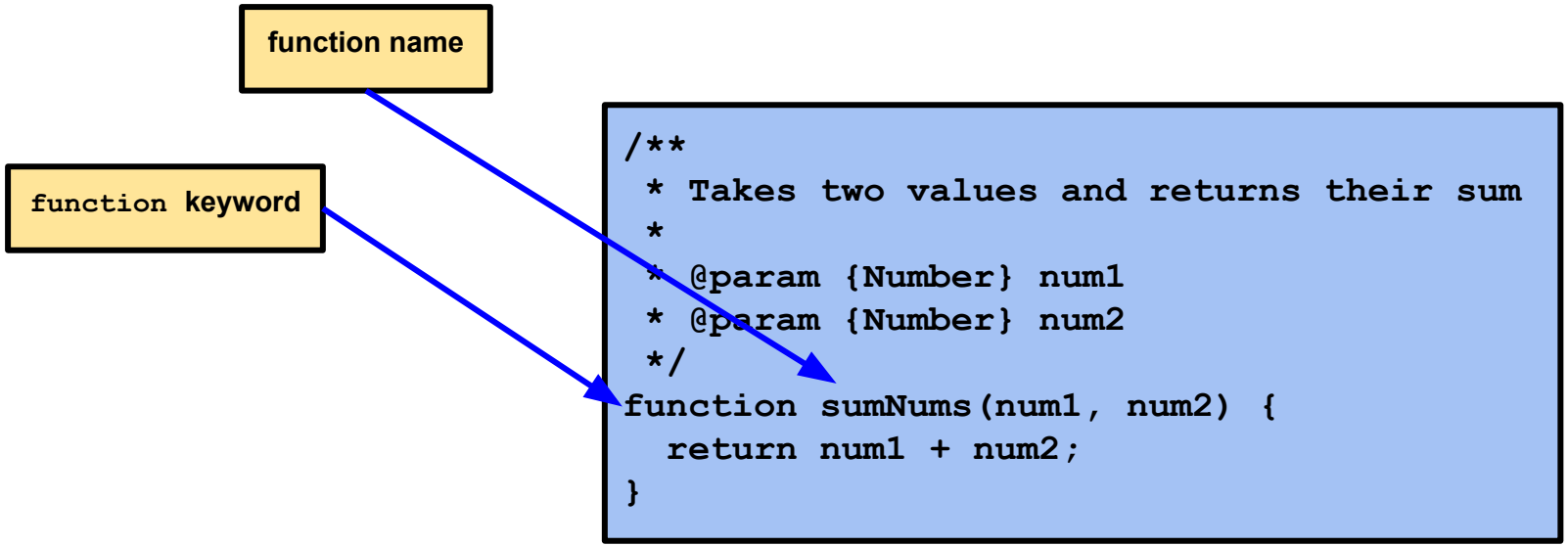
Named Functions

`function` keyword

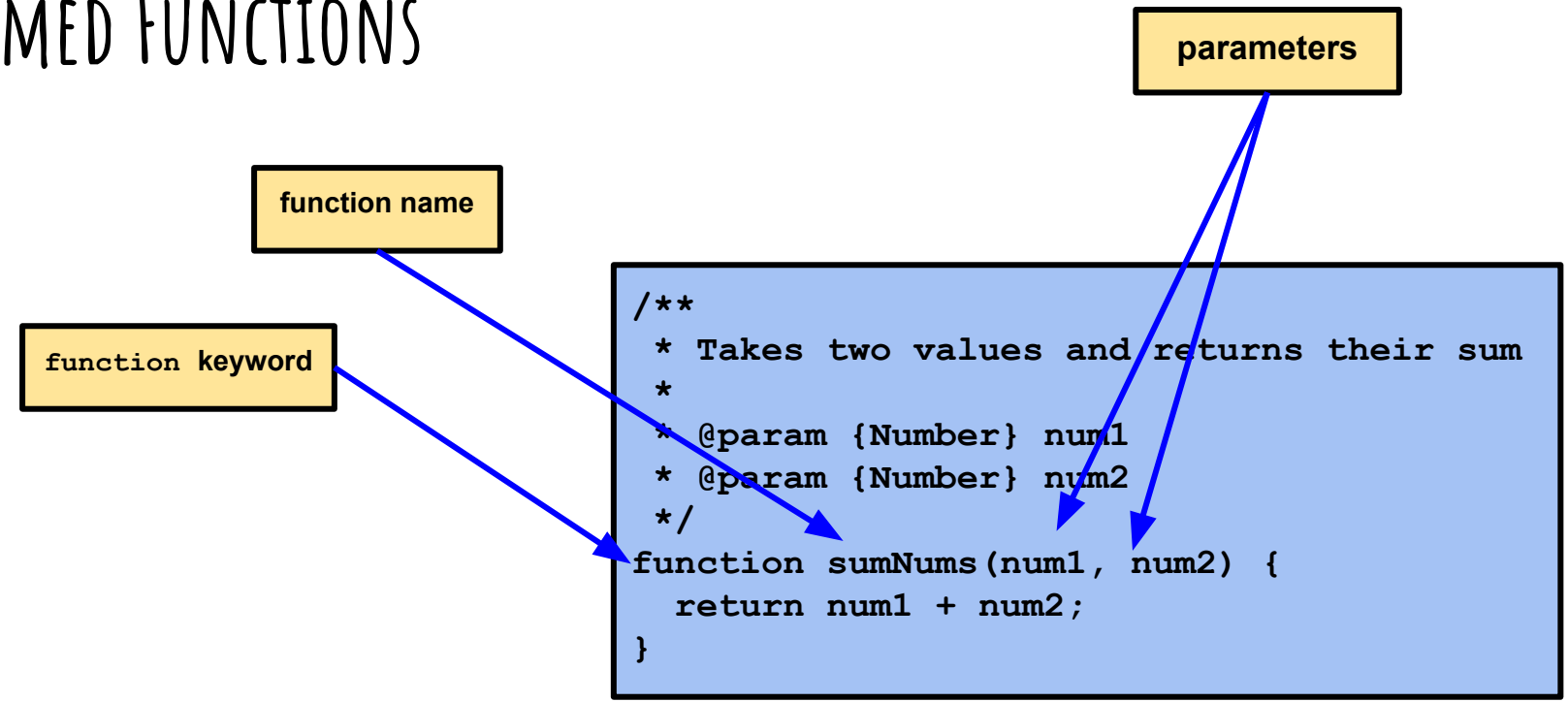


```
/**
 * Takes two values and returns their sum
 *
 * @param {Number} num1
 * @param {Number} num2
 */
function sumNums(num1, num2) {
  return num1 + num2;
}
```

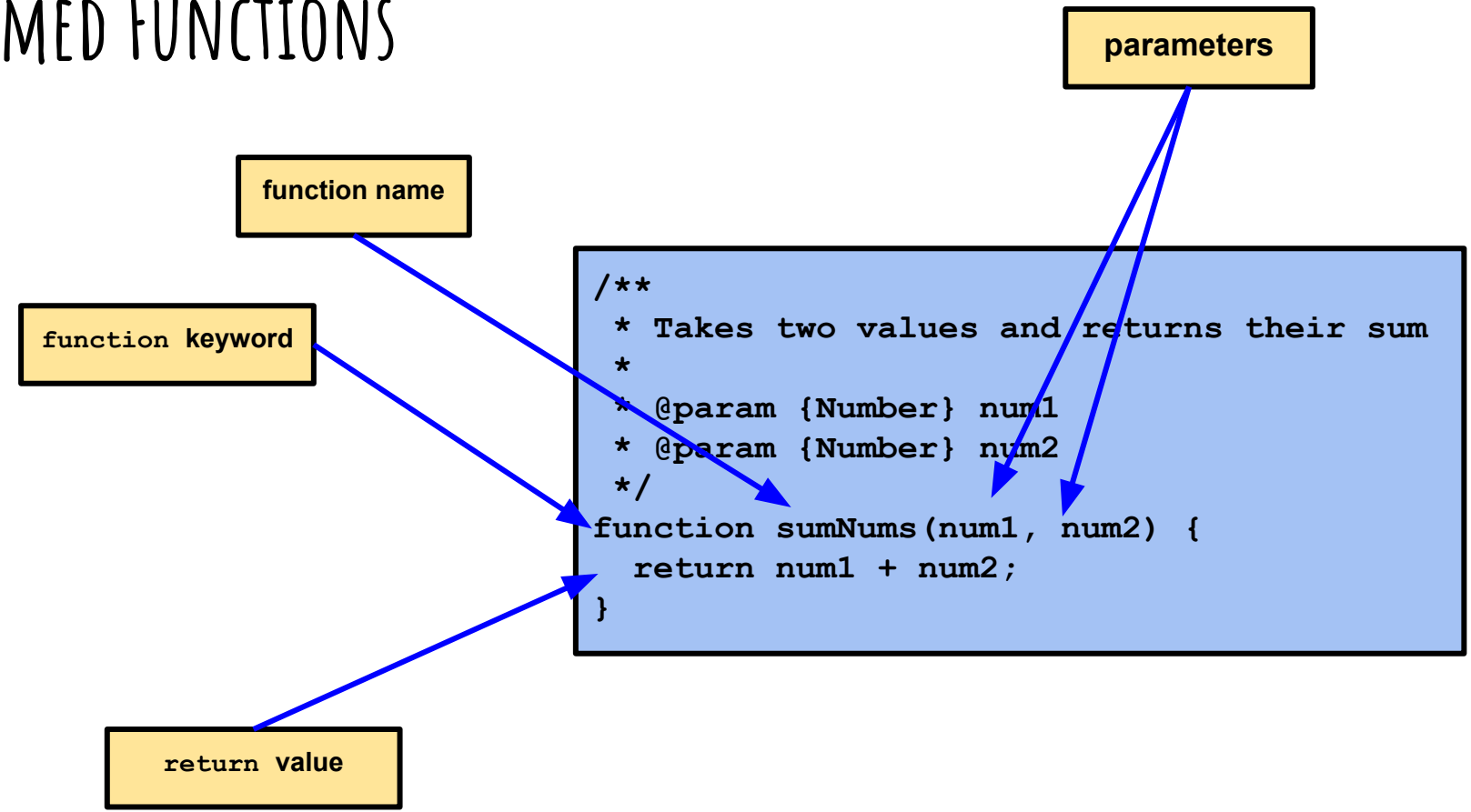
Named Functions



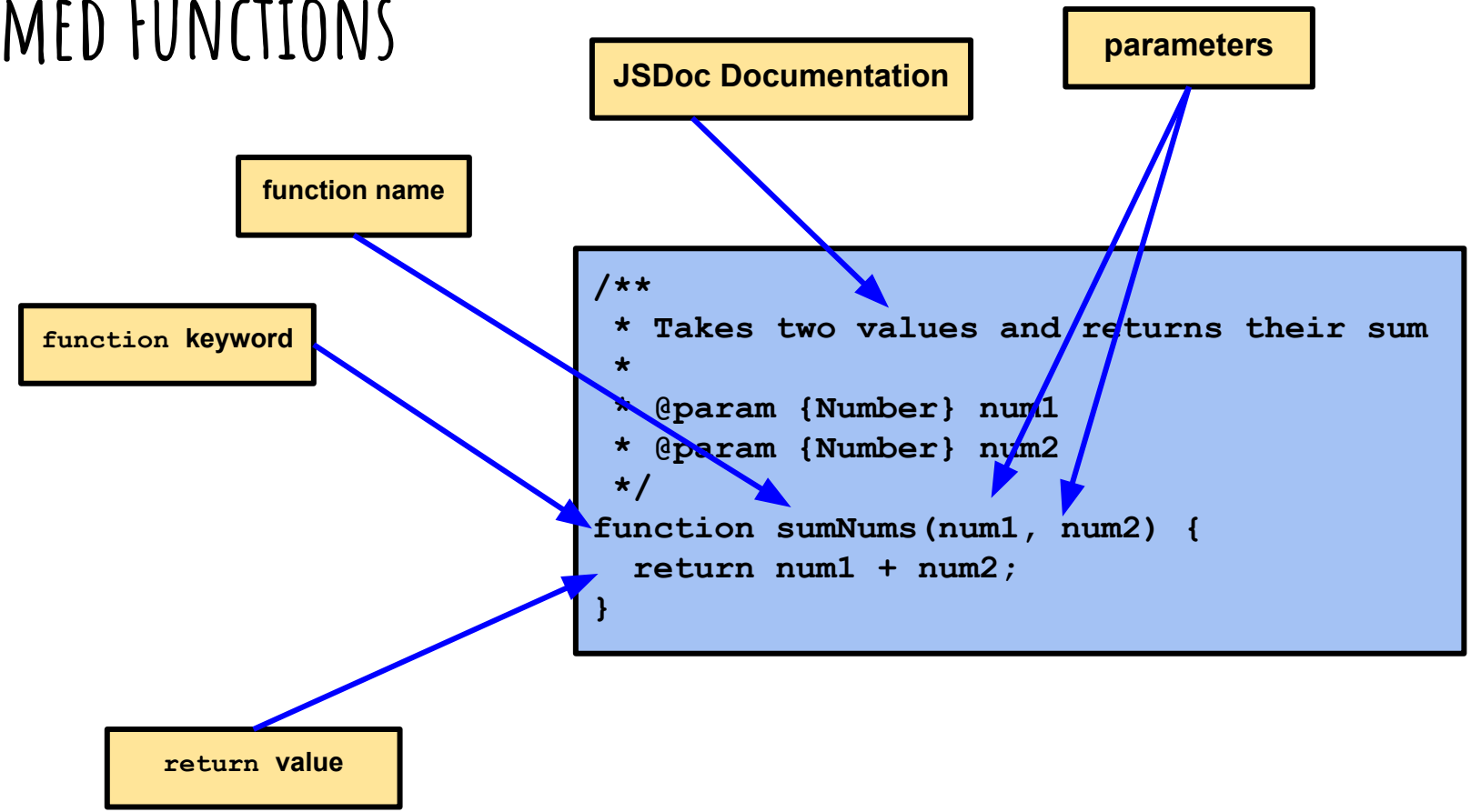
Named Functions



Named Functions



Named Functions



Default Parameters

Parameters are **always optional** and default to **`undefined`** if omitted.

Default values can be assigned

- Can be expressions
- Can call functions
- Can create new objects
- Can create arrays.
- Can refer to parameters to the left in the list.

Default value used if param is **`undefined`**.

```
function multiply(a, b = 1) {
    return a * b
}

function callSomething(thing = something()) {
    return thing
}

function append(value, array = []) {
    array.push(value)
    return array
}

function greet(name, greeting, message = greeting +
     ' ' + name) {
    return [name, greeting, message]
}
```

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

The arguments Variable

- The **`arguments`** variable is an array which contains all the parameters passed to a function.

```
 // No parameters defined,but we still might get some

function concatAll() {
    let result = '';
    for(let i = 0; i < arguments.length; i++) {
        result += arguments[i];
    }
    return result;
}
```

Comments

- Single line comment: **`// comment`**
- Comment block (can be multiline): **`/* comment */`**
- In general, comments should describe the **purpose** of code, NOT what it does.

JSDoc

- Use JSDoc to create documentation for VS Code or other IDEs to use in IntelliSense.
  - Generate by typing `/**` and hitting enter above a function.

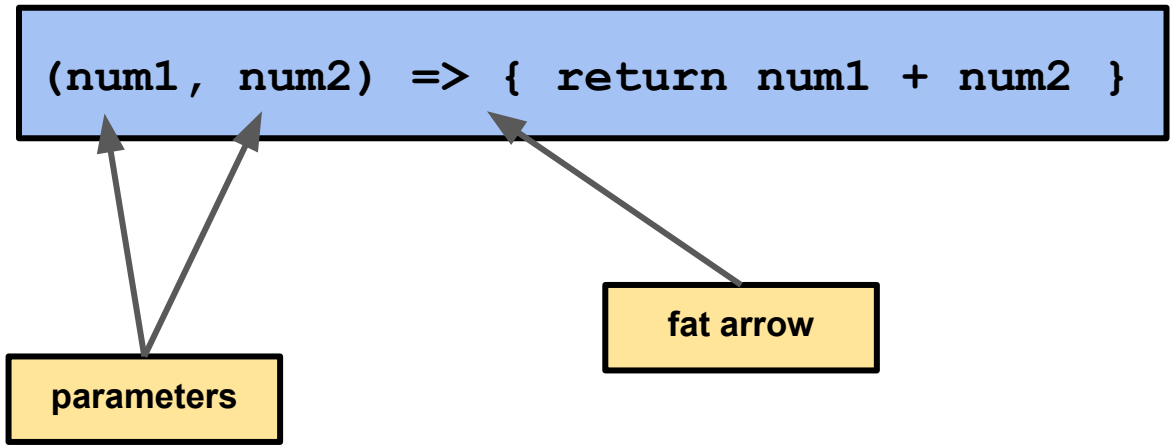
```
/**
 * Takes two numbers and returns the product of
 * those two numbers.
 *
 * Will return NaN if exactly two numbers are not
 * given.
 *
 * @param {number} multiplicand a number to multiply
 * @param {number} multiplier a number to multiply by
 * @returns {number} product of the two parameters
 */
function multiplyBy(multiplicand, multiplier) {
    let result = multiplicand * multiplier;

    return result;
}
```

JSDoc

- @param {type} param-name Parameter-description
- @param {type} [param-name=default-value] Description…
- @returns {type} Return-value-description
- https://jsdoc.app/, https://jsdoc.app/index.html#block-tags

Anonymous Functions

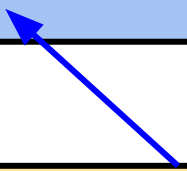
- Functions can be defined without a name.



Anonymous Functions

- Can be assigned to a variable, which can be passed as a method parameter.

```
let sumNum = (num1, num2) => { return num1 + num2 };

console.log(sumNum(5, 6));
```



After being defined, **`sumNum`** can be used like any other function.

Anonymous Functions

- The array function **`reduce`** takes a function that accepts two values as a parameter.

We can use the variable that holds our anonymous function as the parameter:

```
let numsArray = [5, 9, 8, 3];

let sumNum = (num1, num2) => { return num1 + num2 };

console.log(numsArray.reduce(sumNum));
```

Anonymous Functions

Alternately, we can simply pass an anonymous function itself as the parameter:

```
let numsArray = [5, 9, 8, 3];

console.log(numsArray.reduce( (num1, num2) =>
                              { return num1 + num2 } ));
```

Anonymous Functions

Alternately, we can simply pass an anonymous function itself as the parameter:

```
let numsArray = [5, 9, 8, 3];

console.log(numsArray.reduce( (num1, num2) =>
                              { return num1 + num2 } ));

console.log(numsArray.reduce( (num1, num2) =>
                              { return num1 + num2 } , 0));
```

`reduce` has an optional parameter which can be used to provide an initial value. This example passes 0 in as the initial **`num1`** value.

Anonymous Functions

- The array function **`filter`** expects a function as a parameter. It uses the function to return a filtered set of data.

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

let evenNumbers = numbers.filter( (number) => {
    return number % 2 === 0;
});

console.table(evenNumbers);
```

`filter` will return an array of all the values that the anonymous function returns true for.

`console.table` will output a table with array indexes and values.

Anonymous Functions

- The array function **`map`** expects a function as a parameter. It uses the function to return a modified set of data.

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

let numbersPlusThree = numbers.map( (num) => {
    return num + 3;
});

console.table(numbersPlusThree);
```

`map` will use the anonymous function passed in to return a modified set of data.

Anonymous Functions

- The array function **`forEach`** iterates through and can perform an action on each array element.

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

numbers.forEach( (element) => {
    console.log(element);
});
```

`forEach` will use the anonymous function passed in to perform an action on each element of te array.

Array Functions That Take a Function Parameter

JS Function	Parameters	Returns	C# Linq
forEach	Item	Executes the code iteratively, for each element in the array. No return value.	
filter	Item	Array of the same type (<= original size), filtered by the function	Where
map	Item	Array of same size, original elements "mapped" to something new	Select
sort	Item1, Item2	Array of same size and type, with elements sorted. Return 1 if Item1 > Item2, -1 if item1 < item2, 0 otherwise	OrderBy
reduce	Accum, Item	A single value, allows calculating a running value	Sum, Aggregate
every	Item	Boolean, true if every item meets the condition	All
some	Item	Boolean, true if at least one item meets the condition	Any

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