Functions

CSCI 185: Taking a deeper dive into functions

Announcements

- Grading up-to-date
- Today is the **last day you can turn in HW4 for credit** (30 days after original due date).
- Coming up: final project proposal. Will post the specs by Wednesday

Finishing up from Wednesday...

- Before we move on to functions, let's finish up our lecture and activities from last Wednesday...

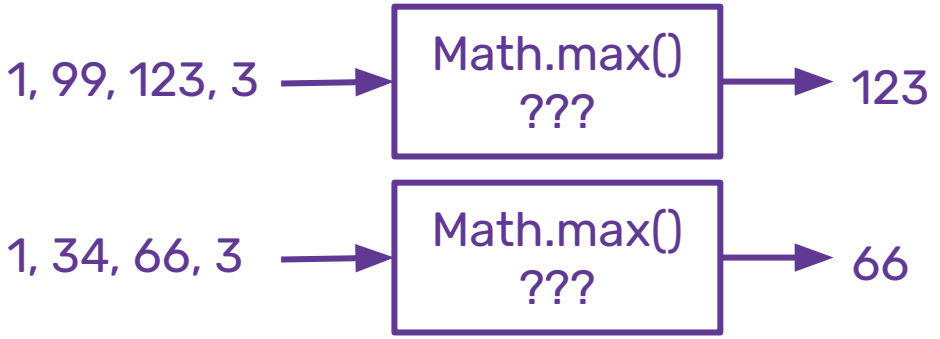
Functions

7 Facts to Know About Functions

1. They provide a way to encapsulate, organize, and reuse code
2. They are similar to operators (+ - * /), but more flexible
3. They are a type of expression, often called a “call expression.”
4. To “run” a function you have to call it or invoke it (using parentheses)
5. They can have required inputs (parameters/arguments), and an optional output value
6. Some functions are built-in. Others can be included from external files. You can also make your own!
7. Functions have scoping rules (variables and parameters created inside of a function cannot be accessed outside of a function)

1. Functions: provide a way to encapsulate and reuse code

- Functions allow you to encapsulate and group lines of code.
- With functions, you can perform the same operations over and over using **different data**
- JavaScript has many built-in functions
- You can also create your own functions, or use functions that other people have written.



2. Functions are similar to operators, but more flexible

Recall: when we examined arithmetic operators (+ - * /)...

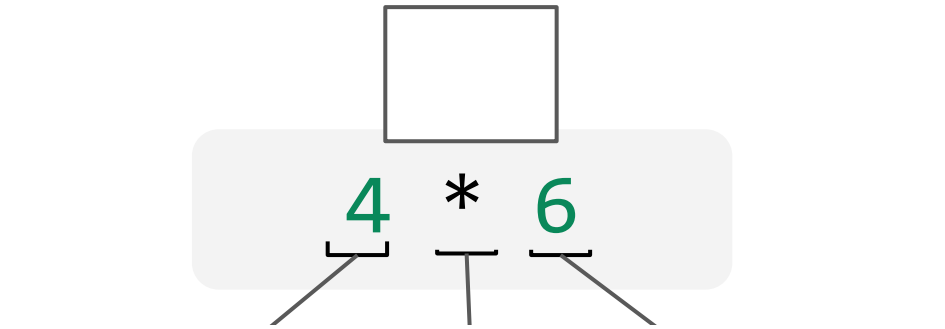
1. The ***operator*** referred to the operation we wanted to perform (addition, multiplication, subtraction, etc.)
2. The ***operands*** were the data we wanted to perform the operation on
3. The ***result*** of the operation on the data was stored in a variable (or printed directly to the screen)

Functions are similar....

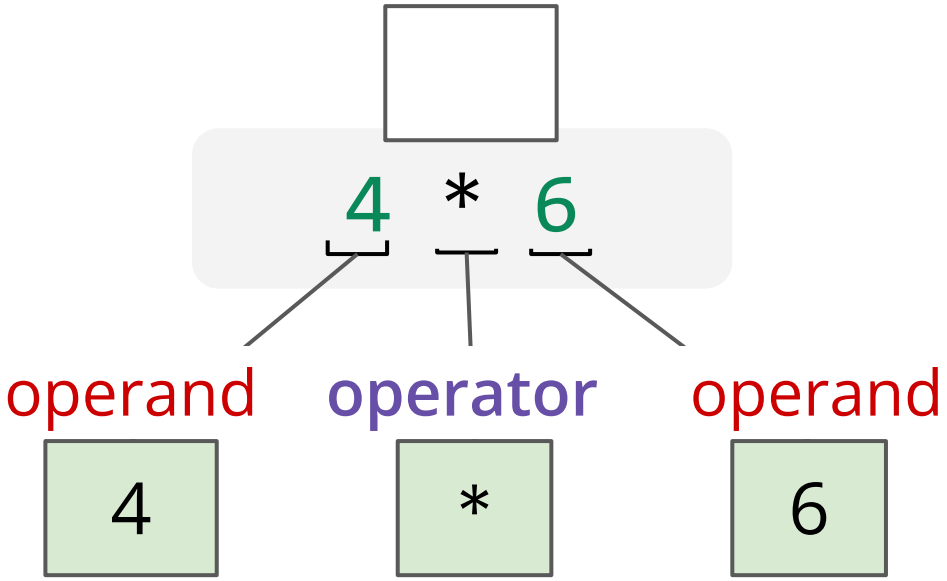
Operator Example

```
4 * 6
```

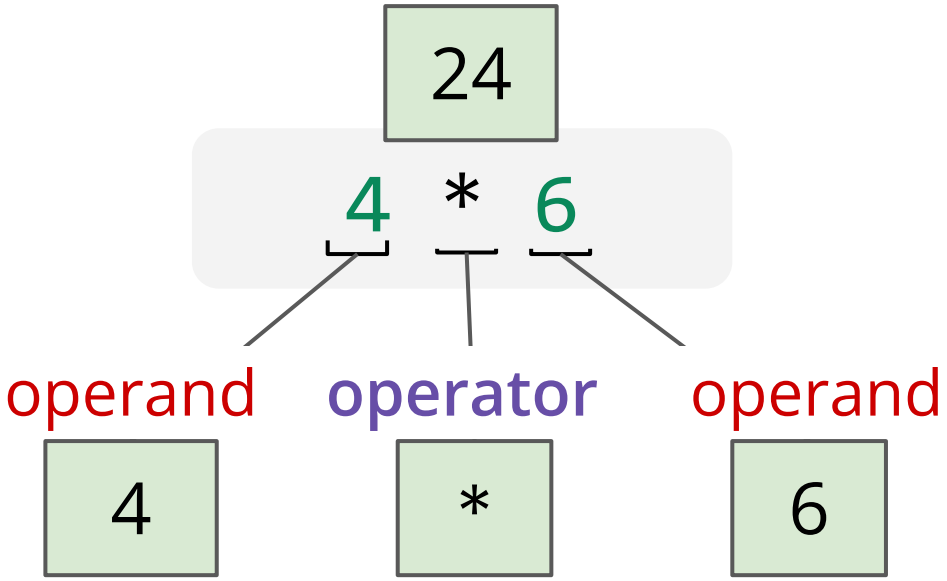
Operator Example



Operator Example



Operator Example



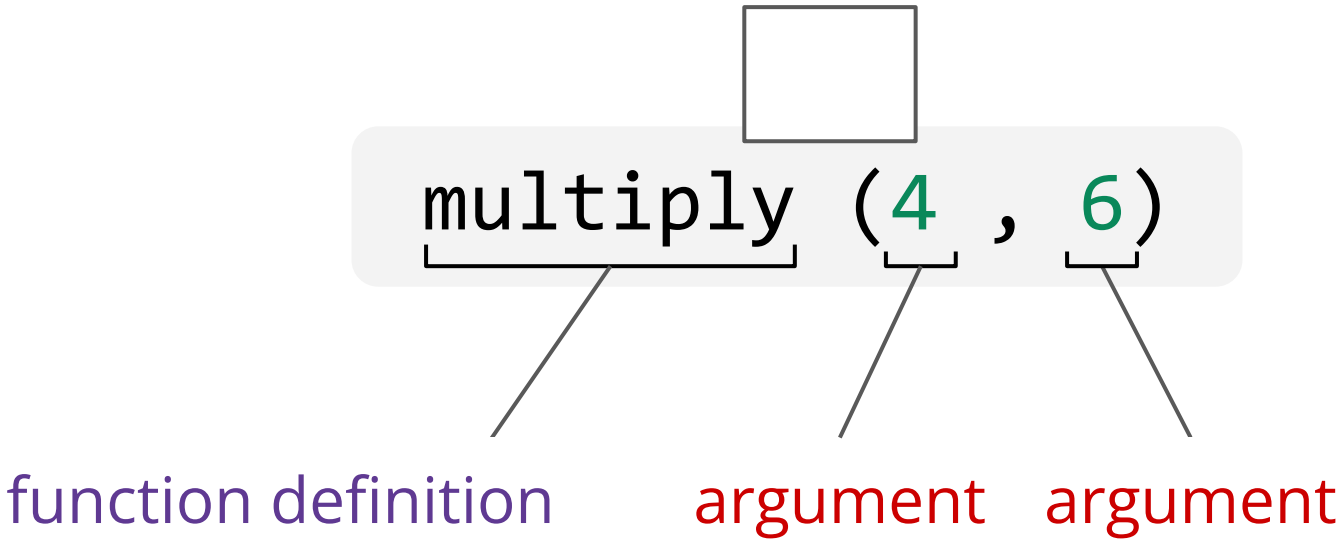
Function Example

```
function multiply(num1, num2) {
    return num1 * num2
}
```

```
multiply(4, 6)
```

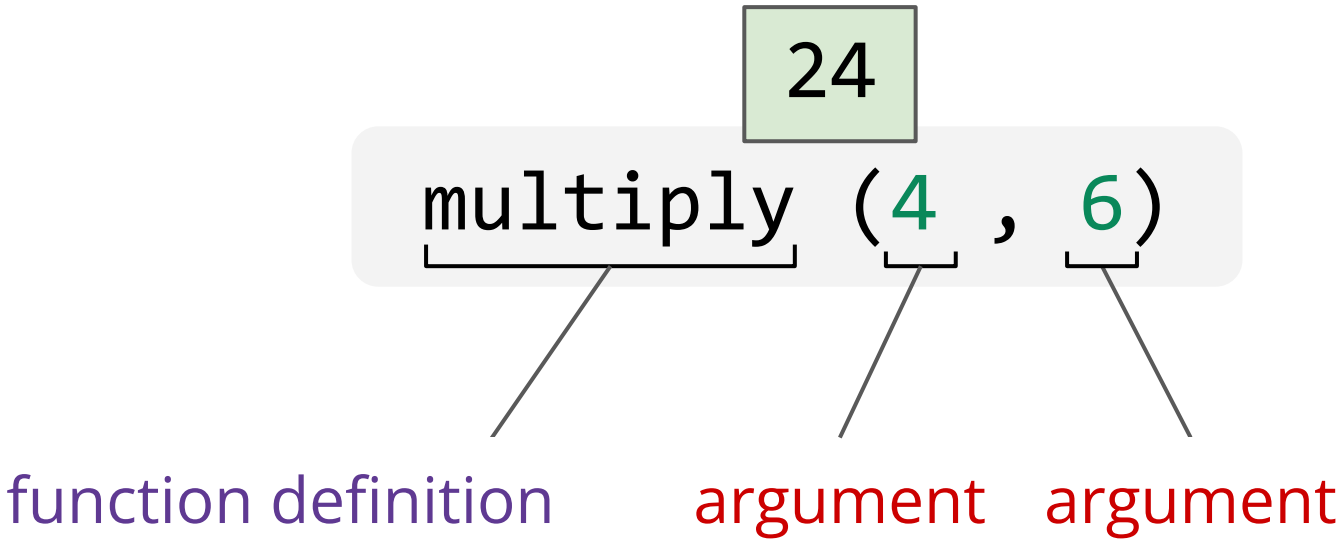
Function Example

```
function multiply(num1, num2) {
    return num1 * num2
}
```



Function Example

```
function multiply(num1, num2) {
    return num1 * num2
}
```



Let’s Practice…

Practice: What is stored in result1?

```
function multiply(num1, num2) {
        return num1 * num2
}


const result1 = multiply(2, 3);
```

For this function invocation:

num1=2
num2=3

Practice: What is stored in result2?

```
function multiply(num1, num2) {
        return num1 * num2
}


const result1 = multiply(2, 3);
const result2 = multiply(4, 3);
```

For this function invocation:

num1=4
num2=3

Practice: What is stored in result3?

```
function multiply(num1, num2) {
        return num1 * num2
}


const result1 = multiply(2, 3);
const result2 = multiply(4, 3);
const result3 = multiply(result1, result2);
```

For this function invocation:

num1=6
num2=12

Practice: What is stored in result4?

```
function multiply(num1, num2) {
        return num1 * num2
}


const result1 = multiply(2, 3);
const result2 = multiply(4, 3);
const result3 = multiply(result1, result2);
const result4 = multiply(multiply(2, 3), multiply(2, 3));
```

Solve the innermost parentheses before you solve the outermost ones.

Multiplication Demo

3. Functions are “Call Expressions”

A call expression invokes a function, providing zero or more inputs, and eventually “returns.” Some call expressions (hereafter “function calls”) return values, others return None (no value).

6	is an *expression* (a constant)
2 * 3	is also an *expression* that evaluates to 6
Math.imul(1, 6)	is a *call expression* that returns 6 as a result
Math.imul(3, 2)	is also a *call expression* that returns 6 as a result

All of these expressions produce the same value after they are evaluated: 6

4. To run a function, you have to call or “invoke” it

In order to run a function, you have to invoke it and give it the data it needs, using the correct type:

```
> Math.max;                                  // tells you that max is a function
> Math.max(2);
> Math.max(2, 55, 29, 88, 7);
88
```

5. Arguments & Return Values

Functions can be defined to have required inputs (parameters/arguments), and/or an optional output value.

The following functions **all return values**, but accept a different number of arguments.

- No arguments: **Math.random()**
- 1 or more required arguments:
  - **Math.max(1)**
  - **Math.max(1, 3, 5, 7, 9)**
- 2 arguments exactly (order matters):
  - **Math.pow(2, 3)**
  - **Math.pow(3, 2)**

Return values

The job of some functions is to give you back a value or a calculation (e.g., Math.random, Math.pow, etc.). You usually store that result of the function call in a variable, or else use that result in another calculation. The result is **“returned”** from the function.

```
let result1 = Math.pow(2, 3)
let result2 = Math.pow(Math.pow(2, 3), 2)
```

Other functions **do not** return values. For instance, console.log(“Hello world”) doesn’t give you back anything. Instead, it’s job is to output something to the console.

5. Arguments & Return Values (Continued)

- What can be an argument in the Math.imul function?
- What data types does Math.imul return?

```
let answer1 = Math.imul(2, 2.5)
let answer2 = Math.imul("2", "2")
let answer3 = Math.imul(2, "2")
let answer4 = Math.imul(2.5, 2)
let answer5 = Math.imul(??, ??)

console.log(answer1, answer2, answer3, answer4, answer5);
```

6. Where do function definitions come from?

- Some functions are built-in (like the Math functions or console.log). They come with the JavaScript language.
- Others can be accessed from other files. For instance, for we will be using functions from the p5.js library to make drawings and animations
- You can also make your own functions (like we did last week)!

7. Function scope

- Variables and parameters that are created within functions cannot be accessed outside of a function.
- The only way to access information that was created from within a function is to return it.

Example: Which variables are out of scope?

```
const z = 5;

function doStuff(a, b) {
   const c = 123;
   const d = 456;
    return a + b + c + d;
}

const result = doStuff(3, 4);
console.log(result);
console.log(z);
console.log(a);
console.log(b);
console.log(c);
console.log(d);
```

Note that a, b, c, and d are **local variables** that are **NOT** accessible outside of the function.

Only the return value can be accessed after the function finishes executing.

Example: Can the function body access the variable z?

```
let z = 5;

function doStuff(a, b) {
   const c = 123;
   const d = 456;
   const e = z + c;
   return a + b + c + d;
}

const result = doStuff(3, 4);
console.log(result);
```

Answer: **YES!** z is a **global variable** so it is accessible inside of any function.

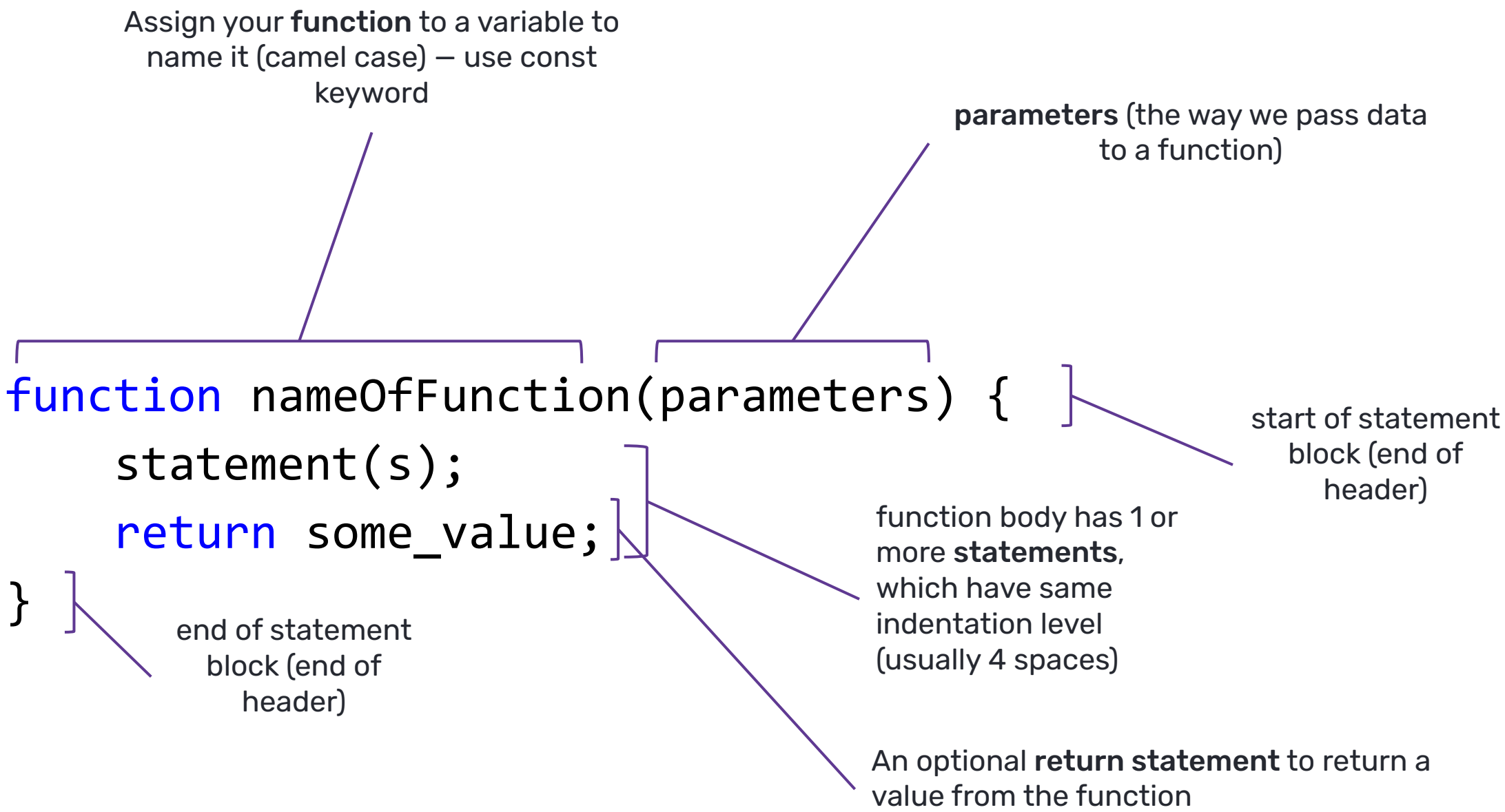
Creating your own functions

Functions: header and body

```
function nameOfFunction(parameters) {
    statement 1;
    statement 2;
    ...
    return some_value;
}
```

HEADER
Specifies the name of the function and the data that it needs. Also called the **SIGNATURE**

BODY
One or more indented statements that the function will execute when called


Assign your **function** to a variable to name it (camel case) – use const keyword
parameters (the way we pass data to a function)
function nameOfFunction(parameters) {
statement(s);
return some_value;
}
start of statement block (end of header)
function body has 1 or more **statements**, which have same indentation level (usually 4 spaces)
end of statement block (end of header)
An optional **return statement** to return a value from the function

There are alternative syntaxes for creating functions

```
// Function declaration
function addTwoNums(num1, num2) {
    return num1 + num2;
}

// Using arrow function (ES6 syntax):
const addTwoNums = (num1, num2) => {
    return num1 + num2;
}
```

Demo

01_first_function.js

Function Inputs: Parameters and Arguments

Reminder:

- Some functions don’t accept any data/arguments
- Some functions require certain kinds of data/arguments
- Some functions allow you to pass in multiple optional data/arguments

Terminology: Parameters & Arguments

arguments: the data that you pass into a function

parameters: local variables, inside a function, that are assigned when the function is invoked

function definition: tells you which parameters are required and which are optional (if any)

Exercise 2: Intro to Parameters

Open **02-functions-with-parameters**. Then:

1. Make 3 functions that set body element's background color to red, blue, and green respectively.
    - Then, attach each button’s event listeners to the appropriate function (event handler).
2. Now, see if you can create **ONE FUNCTION** that can do the same thing as the three functions you just made.
    - Hint: pass a "color" argument into the function.
    - What are the advantages of using the second technique over the first one?

Exercise 3: Drawing by using someone else's functions

Open **03-drawings**. Notice that the file includes some functions in the p5.js library (made by somebody else). It also includes a function that I made called drawGrid()

Your tasks

- Use the built-in p5.js shape functions to draw something (animal, face, tree, etc.).
- Create a function that allows someone to draw multiple copies of your face, tree, etc. at different locations.

Exercise 4: Functions and Return Values

Open **04_functions_that_return_things**. Then:

Part 1: Create a custom function called ***getArea*** that calculates the area of any right triangle. Your function will take 2 arguments: side1 and side2.

Formula: side1 * side2 / 2

Part 2. Create a custom function called ***getHypotenuse*** that calculates the hypotenuse of any right triangle. Your function will take 2 arguments: side1 and side2.

Formula: (side1 ** 2 + side2 ** 2) ** 0.5

Exercise 4: Functions and Return Values

Part 3. Create a custom function called ***getPerimeter*** that calculates the perimeter of any right triangle. Your function will take 2 arguments: side1 and side2

Formula: side1 + side2 + hypotenuse

- Hint: Can you figure out a way to use ***getHypotenuse*** function within the function body?

When you're done, test your functions using the console by invoking each of them with different values for side1 and side2.