Functions

JavaScript

What is a Function?

Think back to variables—they can be thought of as a labeled “container” for holding a value.

Functions are an (often labeled) container for code and instructions!

We can call upon a code block to run via a function’s name, much like we can grab a variable’s value by calling it by name.

In contrast to how we reference variables after they’re declared, functions are followed by a pair of parentheses when you want to run them.

Declaring a named function would look something so…

```
function myFirstFunction () {

}
```

Calling upon a declared function, to execute its code block, would look something like…

```
myFirstFunction();
```

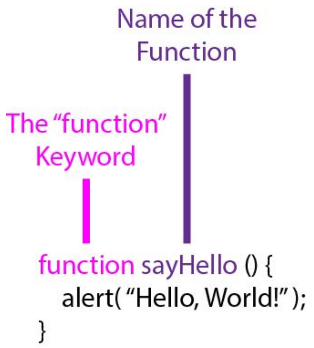
We’ll break this down a bit in the slides that follow.

Declaring a Basic Function

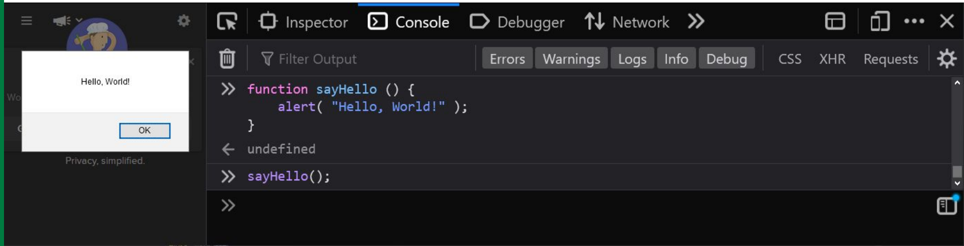
To declare a function, use the "function" keyword followed by a unique name for your function.

You'll notice that the keyword and name for the function declaration are followed by parentheses and curly braces.

The curly braces will contain all of the code and instructions that you'd like to run whenever you call upon this function.



```
function sayHello () {
   alert( "Hello, World!" );
}
```



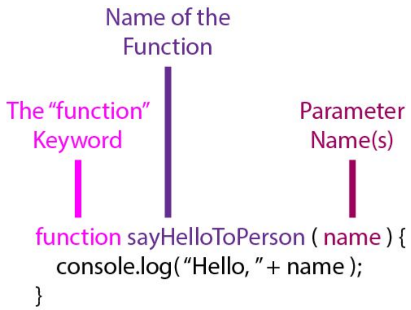
Hello, World!

```
>> function sayHello () {
       alert( "Hello, World!" );
   }
<- undefined
>> sayHello();
>>
```

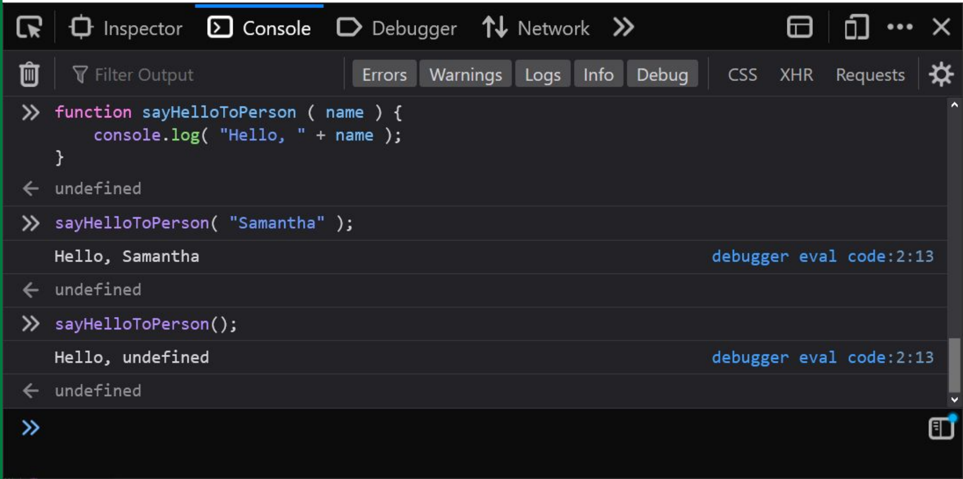
Parameters

If you'd like to accept a value into your function, you can do so by adding a parameter.

We name and add parameters into the space between our parentheses. If you'd like to accept more than one value, you can separate the names by comma.



```
function sayHelloToPerson ( name ) {
  console.log( "Hello, " + name );
}
```

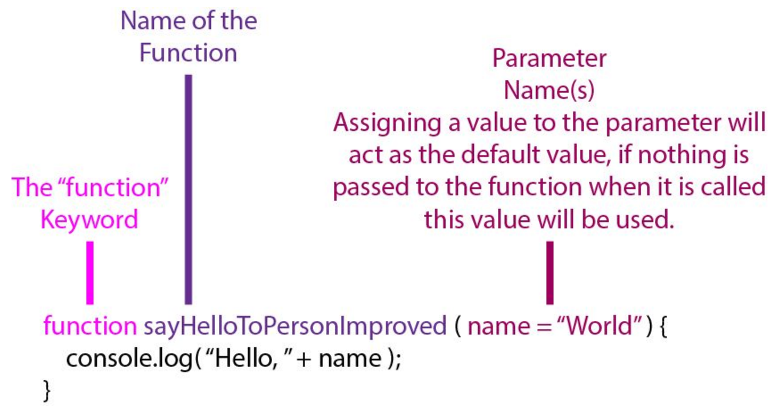


```
>> function sayHelloToPerson ( name ) {
       console.log( "Hello, " + name );
   }
<- undefined
>> sayHelloToPerson( "Samantha" );
   Hello, Samantha                                    debugger eval code:2:13
<- undefined
>> sayHelloToPerson();
   Hello, undefined                                   debugger eval code:2:13
<- undefined
>>
```

Default Parameters

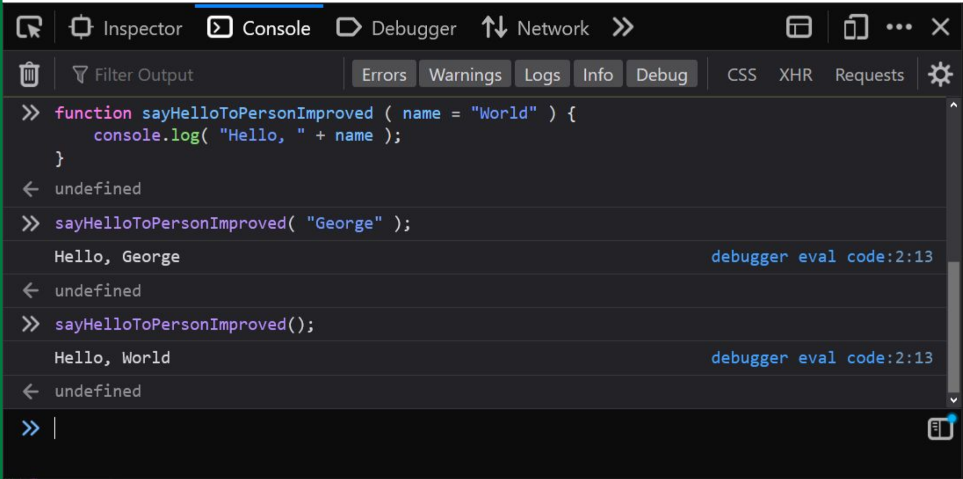
You can have a default value assigned to your parameter for cases where nothing is passed into the function when it is called.

Give default parameters a try…



Assigning a value to the parameter will act as the default value, if nothing is passed to the function when it is called this value will be used.

```
function sayHelloToPersonImproved ( name = "World" ) {
  console.log("Hello, " + name );
}
```

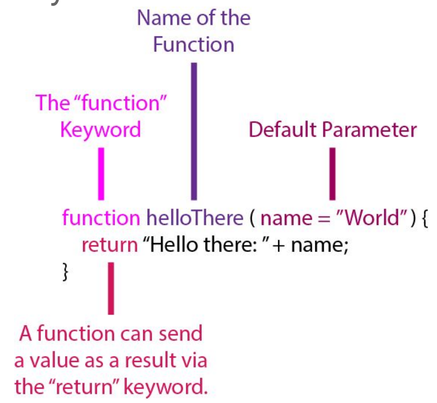


```
>> function sayHelloToPersonImproved ( name = "World" ) {
       console.log( "Hello, " + name );
   }
<- undefined
>> sayHelloToPersonImproved( "George" );
   Hello, George                                        debugger eval code:2:13
<- undefined
>> sayHelloToPersonImproved();
   Hello, World                                         debugger eval code:2:13
<- undefined
>>
```

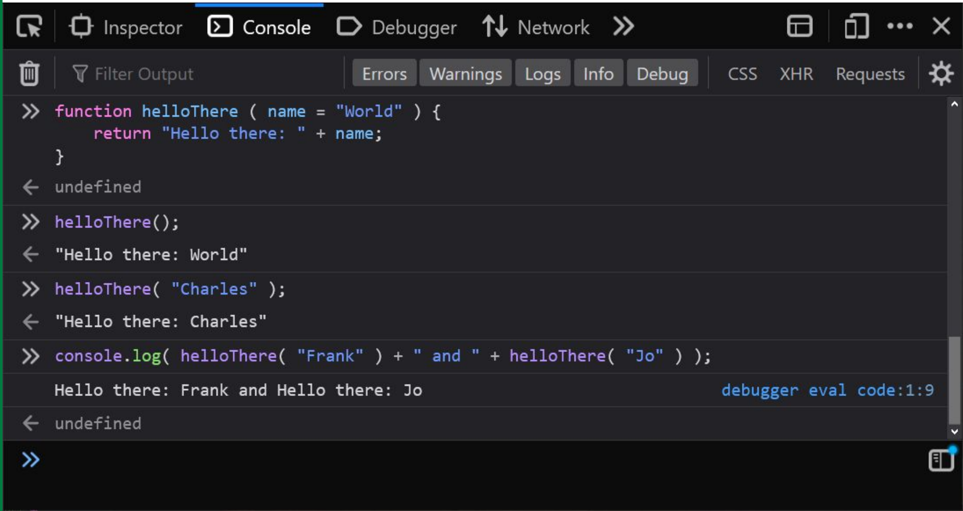
Return

Functions can be made to return a value when evaluated, you simply include the “return” keyword followed by the value.

Give return a try...


Name of the Function
The “function” Keyword
Default Parameter
A function can send a value as a result via the “return” keyword.

```
function helloThere ( name = ”World” ) {
  return “Hello there: ” + name;
}
```

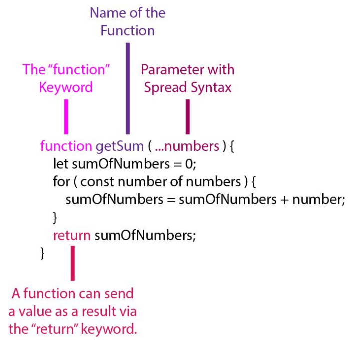


```
>> function helloThere ( name = "World" ) {
       return "Hello there: " + name;
   }
<- undefined
>> helloThere();
<- "Hello there: World"
>> helloThere( "Charles" );
<- "Hello there: Charles"
>> console.log( helloThere( "Frank" ) + " and " + helloThere( "Jo" ) );
   Hello there: Frank and Hello there: Jo                debugger eval code:1:9
<- undefined
>>
```

Spread

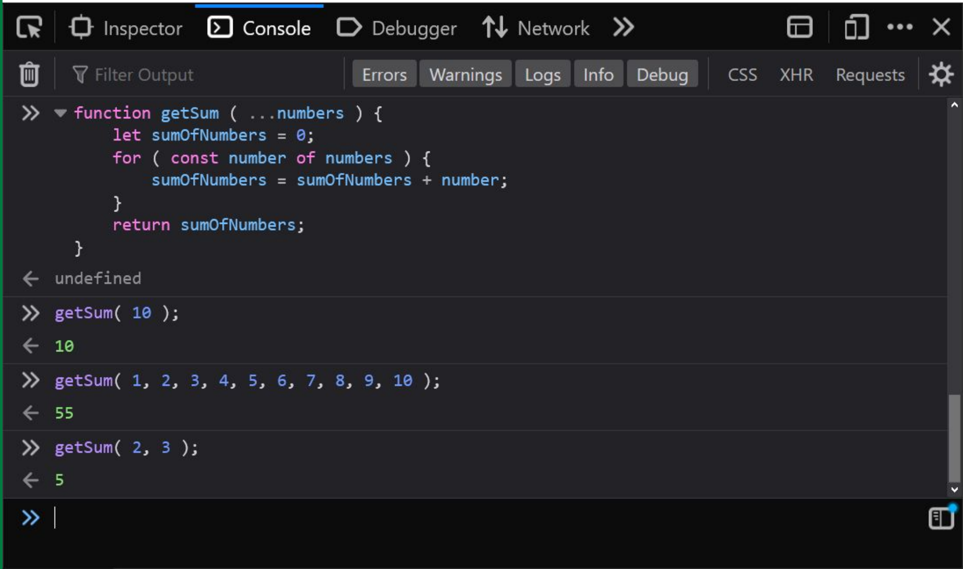
We can accept an “infinite” number of arguments if we use JavaScript’s spread syntax. Just prepend your last parameter with 3 periods.

Give the spread syntax a try...



Name of the Function
The “function” Keyword
Parameter with Spread Syntax
A function can send a value as a result via the “return” keyword.

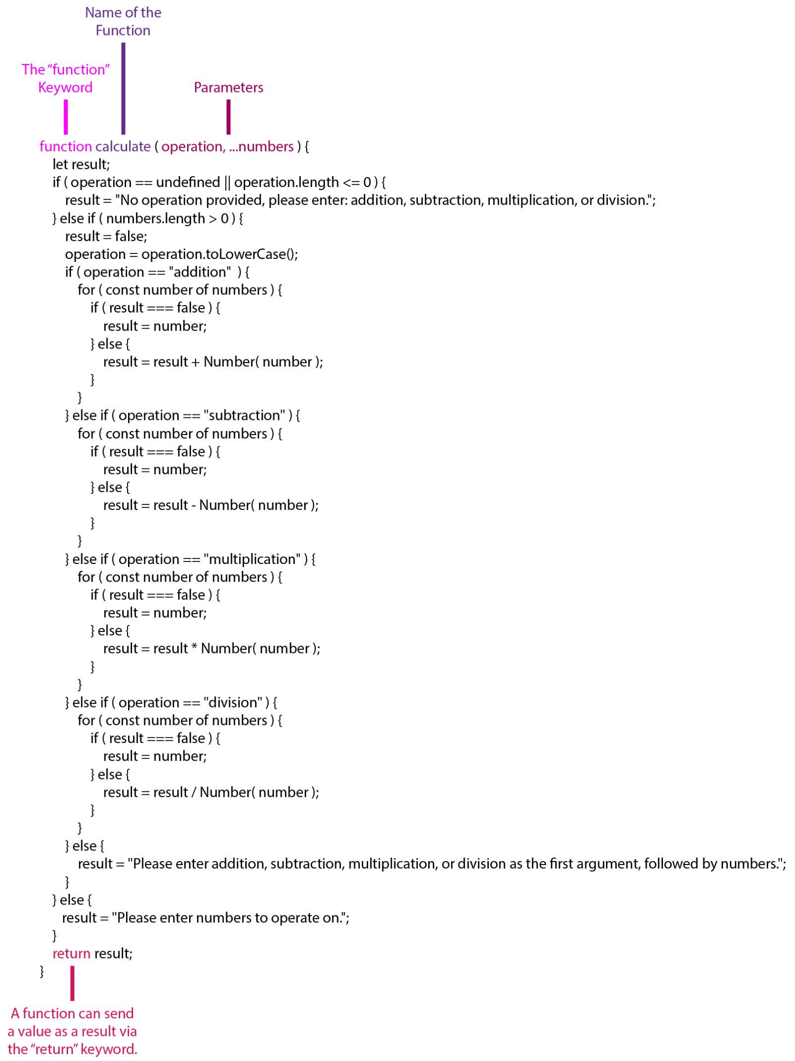
```
function getSum ( ...numbers ) {
  let sumOfNumbers = 0;
  for ( const number of numbers ) {
    sumOfNumbers = sumOfNumbers + number;
  }
  return sumOfNumbers;
}
```



```
>> function getSum ( ...numbers ) {
       let sumOfNumbers = 0;
       for ( const number of numbers ) {
           sumOfNumbers = sumOfNumbers + number;
       }
       return sumOfNumbers;
   }
<- undefined
>> getSum( 10 );
<- 10
>> getSum( 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 );
<- 55
>> getSum( 2, 3 );
<- 5
>>
```

Basic Calculator

Let's put everything we learned together to make an example that has a bit more going on!


Name of the Function
The "function" Keyword
Parameters
A function can send a value as a result via the "return" keyword.

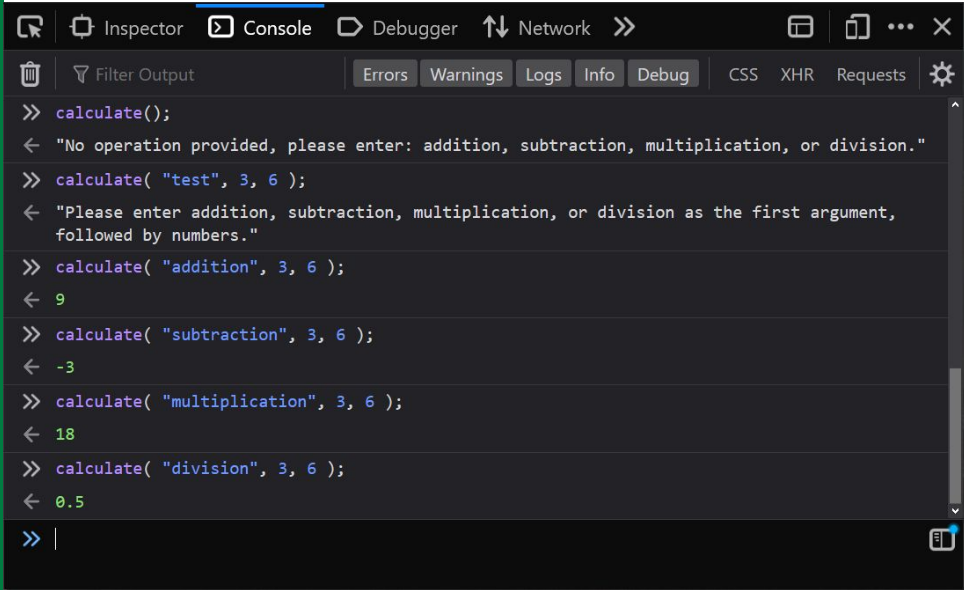
```
function calculate ( operation, ...numbers ) {
  let result;
  if ( operation == undefined || operation.length <= 0 ) {
    result = "No operation provided, please enter: addition, subtraction, multiplication, or division.";
  } else if ( numbers.length > 0 ) {
    result = false;
    operation = operation.toLowerCase();
    if ( operation == "addition"  ) {
      for ( const number of numbers ) {
        if ( result === false ) {
          result = number;
        } else {
          result = result + Number( number );
        }
      }
    } else if ( operation == "subtraction" ) {
      for ( const number of numbers ) {
        if ( result === false ) {
          result = number;
        } else {
          result = result - Number( number );
        }
      }
    } else if ( operation == "multiplication" ) {
      for ( const number of numbers ) {
        if ( result === false ) {
          result = number;
        } else {
          result = result * Number( number );
        }
      }
    } else if ( operation == "division" ) {
      for ( const number of numbers ) {
        if ( result === false ) {
          result = number;
        } else {
          result = result / Number( number );
        }
      }
    } else {
      result = "Please enter addition, subtraction, multiplication, or division as the first argument, followed by numbers.";
    }
  } else {
    result = "Please enter numbers to operate on.";
  }
  return result;
}
```

Trying out the Basic Calculator

Find ways to break it, and see if you can make it more and more bulletproof!

Always test code and functions that you write. See if you can get it to break, consider if there are ways to improve it and its use.

Rewriting and improving your code is called refactoring!



```
>> calculate();
<- "No operation provided, please enter: addition, subtraction, multiplication, or division."
>> calculate( "test", 3, 6 );
<- "Please enter addition, subtraction, multiplication, or division as the first argument,
   followed by numbers."
>> calculate( "addition", 3, 6 );
<- 9
>> calculate( "subtraction", 3, 6 );
<- -3
>> calculate( "multiplication", 3, 6 );
<- 18
>> calculate( "division", 3, 6 );
<- 0.5
>>
```