

JS FUNCTIONS COURSE

APRIL 2022

YONATAN BENEZRA

- DO NOT DISTRIBUTE –

- DO NOT COPY – ALL RIGHTS RESERVED TO THE
AUTHOR –

JavaScript Functions

In the following lesson we will learn:

-

It is one of the most es

Remember when we c

We needed to explain

output - a sandwich.

To do so, we needed s

If we were to enter the

output would be a san

Then, we can simply te

It is one of the most essential concepts in the language.

We needed to explain exactly what the process was, and what would be needed in order to get the desired output - a sandwich.

If we were to enter the above variables into the proper function, we will receive a simple function, who's output would be a sandwich.

we just need to pass differ

If we want different spreads or bread, we just need to pass different variables to the function!

Now, let's create a simple function to make us a sandwich.

```
let sandwich = ""  
  
function makeMeASandwich(bread, spread1, spread2) {  
    sandwich = "a" + " " + bread + " " + "with" + " " + spread1 + " " +  
    "and" + spread2  
}  
  
makeMeASandwich("white", "tomatoes", "lettuce");
```

So, what have we done here?

- We started with the function keyword. This is how we declare a function.
- Next, we defined a function name, which is makeMeASandwich, which is the given name for the function.
- Then, we added parenthesis. We use parenthesis to add parameters.
- Finally, we used curly braces to create the Function body.
All of the code that is within these curly braces is called the function body.
And it's this code that will be executed when we run the function.
- To use this function, we simply write the function name followed by parenthesis. This process is called "invoking", "running", or "calling" the function.

JS functions allow us to write a set of code which can be reused in the future, in order to eliminate the need to write the same code repeatedly. To better understand this concept, let's look at a simple example.

Let's say you want to add ten to a number.
Here's how this can be done without functions.

```
const number = 1 + 10;  
  
// Answer = 11
```

Now let's write this using a function.

```
function addTen(number) {  
  return number + 10;  
}  
  
const firstNumber = addTen(1); // Answer = 11
```

Although more lines of code have been written in the second example, this could be useful later on, and will result in much cleaner code, if we have additional cases of numbers that we would like to add 10 to, as shown below:

```
function addTen(number) {  
  return number + 10;  
}  
  
const firstNumber = addTen(1); // Answer = 11  
const secondNumber = addTen(2); // Answer = 12  
const thirdNumber = addTen(3); // Answer = 13  
const fourthNumber = addTen(4); // Answer = 14  
const fifthNumber = addTen(5); // Answer = 15
```

I hope that now you can understand the concept of writing something once and reusing it again later by using functions.

Functions help us reduce, reuse, and recycle our code, which is awesome! 🦾

Now, let's proceed to understanding several additional concepts within functions, including:

- Parameters and Arguments
- Return Statement
- Calling a Function

Parameters and Arguments

Parameter: placeholders to receive input values.

```
function addTen(number) {  
    return number + 10;  
}
```

```
const firstNumber = addTen(1); // Answer = 11
```

Argument: actual values of function parameters, to input data

We added the function parameter inside the parenthesis.

This is the input, or list of input values, that need to be received to perform the function.

You can think of it as an empty placeholder that needs to be replaced later on.

Arguments are actual values of function parameters to be placed in those input data placeholders.

Therefore, in the above examples, the placeholder is replaced by the actual data, in this case, the number "1".

Return Statement

Using the return keyword, we can return any value from the function. Some functions may not return a value, but most functions do.

We call this value the result of the function.

Then, this value that is returned can be used anywhere later in the code.

Example:

```
function addTen(number) {  
    console.log(number + 10);  
}
```

```
addTen(1); // Answer = 11
```

```
function addTwenty(number) {  
  return number + 20;  
}  
  
const firstNumber = addTwenty(1); // Answer = 21
```

The first function will simply console log the answer, while the second function will return it to the user as the answer. We use the returned answer as a new variable.

***Note:**

the return keyword immediately exits the function.

Exercise:

What will be the output of the following code?

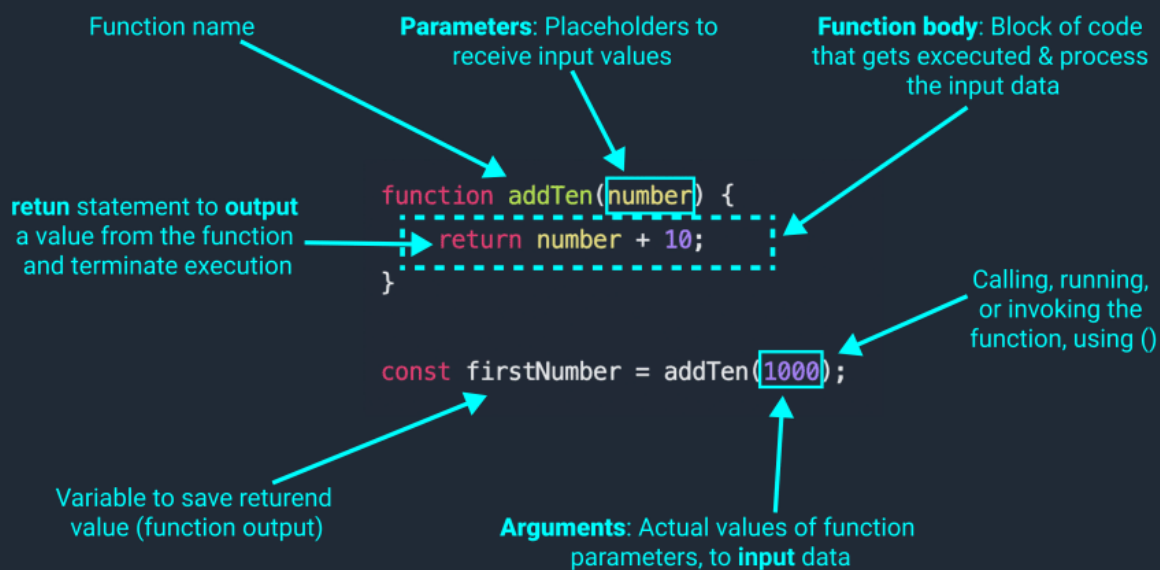
```
function addTen(number) {  
  return number + 10;  
  console.log("Let's add numbers!")  
}  
  
const firstNumber = addTen(1); // Answer = 11
```

See the below gif and image, which may help us to better understand this topic:

Anatomy of a Function

```
function addTen(number) {  
  return number + 10;  
}  
  
const firstNumber = addTen(1000);
```

Anatomy of a Function



There are different ways to write JavaScript functions.

- Function Declarations
- Function Expressions
- Arrow Function

Function Declarations

We use the function keyword to declare a function, same as we declare a variable. So, let's declare another function to calculate an age, based on a given birth year.

```
//Function declaration

function calcAge1(birthyear) {
  return 2022 - birthyear;
}

const age1 = calcAge1(1986);
```

Let's see what we have done here,

- We have created a function by giving the name calcAge1 to calculate the age.
- We have given the function a parameter called birthyear. Meaning, that is the input data that we will be requesting in order to calculate the age.
- We will be returning the results by subtracting the birthyear from the current year in order to calculate the age.
- Then, we stored this function in another variable (age1) and called the function, and we have given the actual data to calculate the age inside the calcAge1.

Function Expressions

Now, let's see how we can perform the same function with the function expression.

```
//Function expression

const calcAge2 = function(birthyear) {
  return 2022 - birthyear;
}

const age2 = calcAge2(1986);
```

Earlier, we started with the function keyword. With function expressions, however, we will write the function as an expression. Remember that an expression produces a value, so we must create a variable to store that value.

- First, we create a variable to store the function (calcAge2), which will be the actual function.
- Then, we write the function, same as before. In this case we will omit the function name, which makes this an “anonymous function”, however, this can be done using a function name as well.
- Next, we add the function parameters and function body, and then call the function.

Difference between declarations and expressions:

We can call a function declaration before it is defined in the code.

We call this hoisting.

Arrow Functions

There is another way of declaring functions in modern JavaScript, known as the arrow function. This function gets its name from the fact that it looks like an arrow: `() => {}`.

Arrow functions are shorter and faster to write, which is why...

We like arrow functions.

Let's look at the same example that we used before, only this time we will convert it to an arrow function.

```
//Arrow function
const calcAge3 = birthyear => 2022 - birthyear;

const age3 = calcAge3(1998);
```

Yes,

It is really as easy as it seems.

The above is an example of a one-line arrow function.

Let's see how this would look with more parameters.

Say we wanted to check how many years remain until my retirement.

To do so, we will need my birth year and my name as parameters.

Next, we will want to check my current age, and then check it against the eligible retirement age, which in this case will be 65.

Finally, we will produce a result stating how many years I have left until I am able to retire.

```
const yearsToRetirement = (birthyear, firstName) => {
  const age = 2022 - birthyear;
```

```
const retirement = 65 - age;

return `${firstName} will be able to retire in ${retirement} years!`;
}

const retirement = yearsToRetirement(1986, 'Sumudu');
console.log(retirement);
```

In the above example, we wrapped the parameters in parenthesis and wrapped our code in curly braces, similar to what we did with function declarations and expressions, which is the only difference that you will see between simple and complex arrow functions.

***Note:**

The arrow function automatically returns the value without explicitly defining the return keyword.

Functions Cheatsheet:

//Function declaration

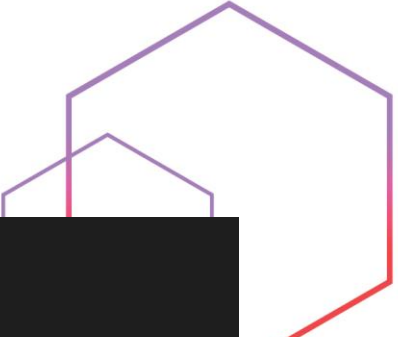
```
function calcAge1(birthyear) {
  return 2022 - birthyear;
}
```

//Function expression

```
const calcAge2 = function(birthyear) {
  return 2022 - birthyear;
}
```

//Arrow function

```
const calcAge3 = birthyear => 2022 - birthyear;
```



```
const yearsToRetirement = (birthyear, firstName) => {  
  const age = 2022 - birthyear;  
  const retirement = 65 - age;  
  return `${firstName} will be retired in ${retirement} years!`;  
}
```