Function Arguments

We'll add on to functions by describing function arguments. These make functions considerably more powerful, as they allow us to pass information into a function. The code inside functions becomes dynamic.

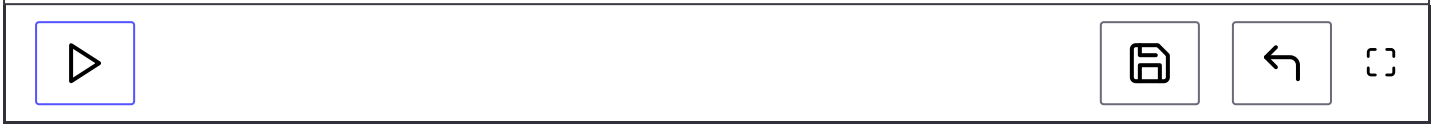
Arguments#

Functions have more stuff built into them that makes them extremely powerful. They don't just do the same exact thing every time. They're dynamic.

We can write a function in a way that makes it dynamic. We can *give* the function a value to use when we call it. The function will then use this value when it runs its code.

Have a look at this code and we'll discuss it afterward.

```
1  function printValue(someValue) {
2      console.log('The item I was given is: ' + someValue);
3  }
4
5  printValue('abc'); // -> The item I was given is: abc
```



We have a function named `printValue`. Inside the parentheses on line 1, we put a word: `someValue`. This word symbolizes a future value. It's a variable that we can use inside the function.

`someValue` is a placeholder variable. It will be different every time the function is called, depending on how we call it.

This variable gets a value when we call the function. When we call it, we *pass in* the value we wish `someValue` to be equal to. In this case we give it `'abc'`.

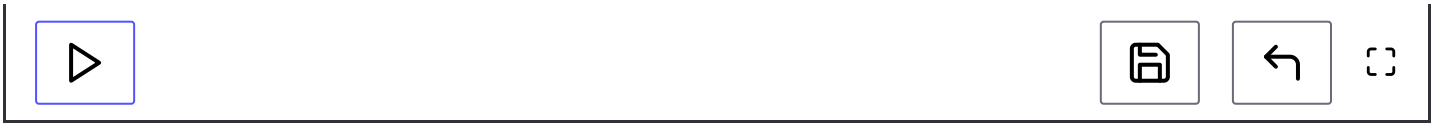
So, when the function is called, the variable `someValue` inside the function is equal to `'abc'`. That's why the console log statement works.

`someValue` is called an **argument** to the function. Another word for argument is **parameter**.

We can pass in any argument we like to `printValue` when we call it. The function will accept it and use it in its function body.

```
function printValue(someValue) {
    console.log('The item I was given is: ' + someValue);
}

printValue('abc');     // -> The item I was given is: abc
printValue('Hello!');  // -> The item I was given is: Hello!
printValue(6);         // -> The item I was given is: 6
printValue(false);     // -> The item I was given is: false
```



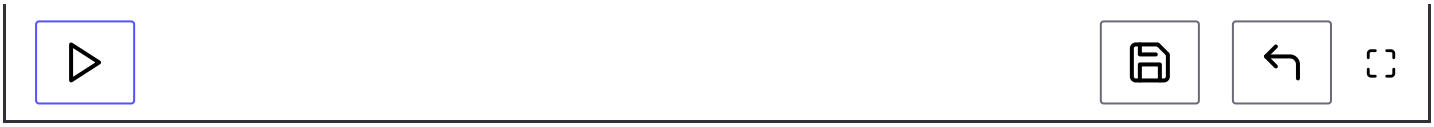
Look at this pattern and think about it and try to make sure that it makes sense.

When we call a function by passing in an argument, the argument variable inside the function gets assigned whatever was passed in. It's as if we used an equals sign. The item we passed in gets copied to the variable in the function.

We can pass in variables as arguments as well.

```
function printValue(someValue) {
    console.log('The item I was given is: ' + someValue);
}

let variable = 17;
printValue(variable); // -> The item I was given is: 17
```



We can name the argument variable in the function anything we want. This means the following functions are equivalent.

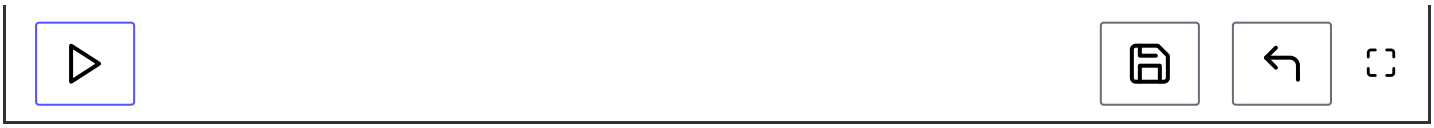
```
function print1(someValue) {
    console.log('The item I was given is: ' + someValue);
}

function print2(arg) {
    console.log('The item I was given is: ' + arg);
}

function print3(xxx) {
    console.log('The item I was given is: ' + xxx);
}

let variable = 20;

print1(variable); // -> The item I was given is: 20
print2(variable); // -> The item I was given is: 20
print3(variable); // -> The item I was given is: 20
```

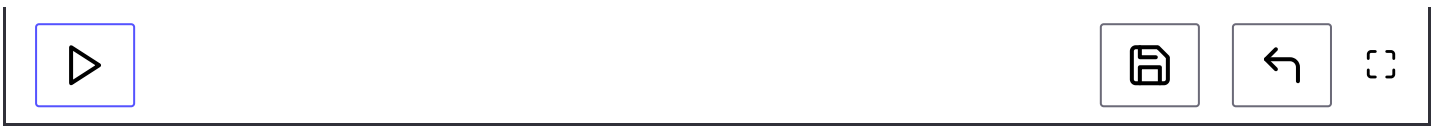


Multiple Arguments#

We can write a function that takes in multiple arguments.

```
function printValues(value1, value2) {
    console.log(value1 + ', ' + value2);
}

printValues('abc', 123); // -> abc, 123
```



Functions can take in as many arguments as we wish to provide. We can pass in as many as we like when we invoke them.

`undefined` arguments#

What happens if we don't pass in any arguments to a function that expects some? Remember our discussion of `undefined`.

```
function printValue(value) {
    console.log(value);
}

printValue(10); // -> 10
printValue();   // -> undefined
```

If we don’t pass in an argument to a function, it receives the value of `undefined` when the function runs. This follows the guideline spelled out in an earlier lesson. `undefined` is meant to symbolize something missing. In this case, it shows a missing argument.

If there is a function that takes in multiple arguments and we provide only one, all argument variables in the function except the first one will become `undefined`. The first argument variable will receive the value we pass in.

```
function printValues(value1, value2, value3) {
    console.log(value1 + ', ' + value2 + ', ' + value3);
}

printValues('abc', 123, true); // -> abc, 123, true
printValues('abc', 123); // -> abc, 123, undefined
printValues('abc'); // -> abc, undefined, undefined
printValues(); // -> undefined, undefined, undefined
```

Local Variables#

Any variable created inside a function is local to that function. This includes all variables created with `let` and argument variables.

If we declare a variable inside a function and try to use it outside, we’ll get an error.

```
function printValue(value) {
    let stringToPrint = 'The value I was given is: ' + value;
    console.log(stringToPrint);
}

printValue(10); // -> The value I was given is: 10
console.log(stringToPrint); // -> error
```

```
function printValue(value) {
    console.log(value);
}

printValue(10); // -> 10
console.log(value); // -> error
```

A function can, however, use variables that were declared outside the function.

```
function printValue(value) {
    console.log(startString + value);
}

let startString = 'The value I was given is: ';
printValue(10); // -> The value I was given is: 10
```

Note that we write the function above `startString`, but we can still use it in the function. Because we call `printValue` after we declare the variable `startString`, it can use that variable.

It doesn't matter that we wrote the function above where we wrote the variable. It only matters that we invoke the function after writing the variables that it needs.

`console.log()`

We've alredy been using a function. We introduced it in lesson 2. Every time we run `console.log()`, we invoke a function and the item we provide is its argument.

`console.log` can actually accept multiple arguments. It can take in as many as we provide.

```
console.log(1, 2, true, 'abc', 17);
// -> 1 2 true 'abc' 17
```

It'll print everything we give it, separated by spaces. We'll be using this more often from now on.

Back
Introduction to Functions

Next
Return Statements

Completed

Report an Issue