

the Master Course

{C0DENATION}

JAVASCRIPT FUNDAMENTALS

Functions

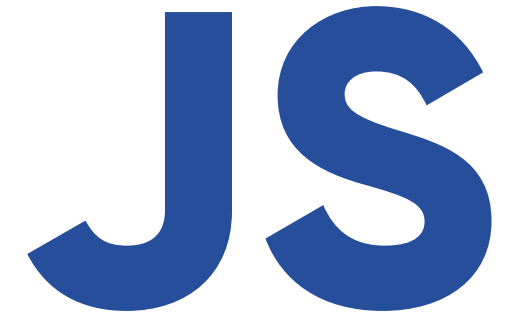
{ CODENATION }

Learning Objectives

To understand how functions work

To write programs with functions

To write programs with all three types of functions



First Things First!

Create a ticket machine for a cinema.

Write an if statement that checks the ages of cinema goers and displays the ticket prices.

- Child (below 18): £8**
- Adult (18+): £10.95**
- Senior (60+): £7.50**

JS

Introducing **Functions**

... functions **break our code up** into **smaller, reusable** chunks!



Let's take this in...

JS

```
const pressGrindBeans = () => {  
    console.log("Grinding for 20 seconds");  
}  
  
pressGrindBeans();
```

Let's take this in...

JS

```
const pressGrindBeans = () => {  
    console.log("Grinding for 20 seconds");  
}  
  
pressGrindBeans();
```

Declare a new function

Let's take this in...

JS

```
const pressGrindBeans = () => {  
    console.log("Grinding for 20 seconds");  
}  
  
pressGrindBeans();
```

Start grinding the coffee



Let's take this in...

JS

```
const pressGrindBeans = () => {  
    console.log("Grinding for 20 seconds");  
}
```

```
pressGrindBeans();
```



Runs the function
pressGrindBeans

JS

Activity

Create a function called **favColor**, inside the function log 'My favourite color is Blue' to the console.

JS

Lets level up....

... functions with **IF** statements included



Lets take this in...

JS

```
let coffeeIsGrinding = false;

const pressGrindBeans = () => {
  if (coffeeIsGrinding) {
    console.log("Stopping the grind");
    coffeeIsGrinding = false;
  } else {
    console.log("Grinding is about to begin");
    coffeeIsGrinding = true;
  }
}

pressGrindBeans();
```

JS

Parameters

... these really make functions **tick!**



JS

Parameters give functions flexibility

...they provide the ability for functions to act
based on **data inputs!**

Lets take this in...

These are the **Parameters**

```
const cashWithdrawal = (amount, accnum) => {  
    console.log(`Withdrawing ${amount} from account ${accnum}`);  
}
```

```
cashWithdrawal(300, 50449921);  
cashWithdrawal(30, 50449921);  
cashWithdrawal(200, 50447921);
```

These are the **Arguments**



We can use global variables in functions!

```
let accnumber = 50449921;

const cashWithdrawal = (amount, accnum) => {
    console.log(`Withdrawing ${amount} from account ${accnum}`);
}

cashWithdrawal(300, accnumber);
cashWithdrawal(30, 50449921);
cashWithdrawal(200, 50447921);
```




Activity

Change the **favColor** function to take your favourite color as a parameter, and inside the function log that parameter inside a template literal to the console.

JS

We can also call on functions

...to do a job and **return** the result!

Lets take this in...

JS

```
const addUp = (num1, num2) => {  
    return num1 + num2;  
}
```

Add up two numbers and return
the answer

```
addUp(7, 3);  
console.log(addUp(2, 5));
```

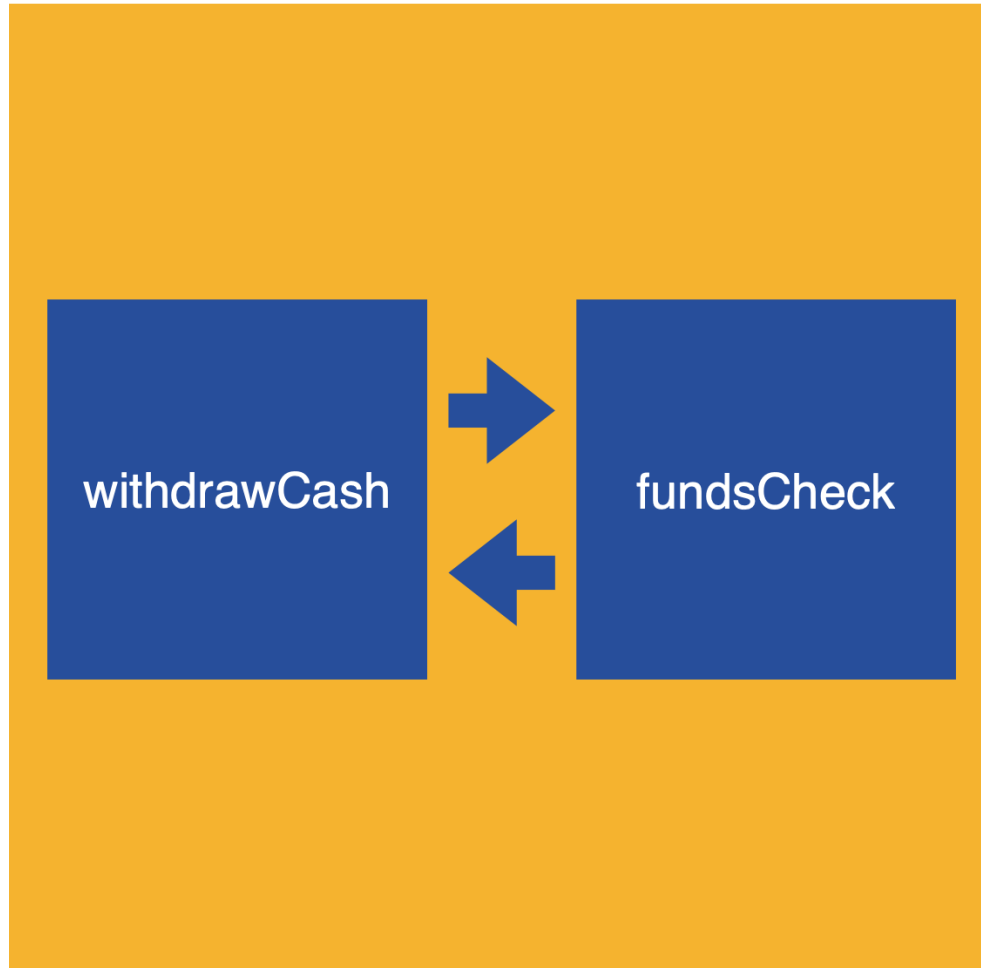
JS

Functions might call

... on other functions and **use that to achieve its goal**. For example

JS

**Does the customer have
enough funds requested?**
...check and **return** result to
withdrawCash



An example...

JS

```
const multiplyByNineFifths = (celsius) => {  
  return celsius * (9/5);  
};
```

```
const getFahrenheit = (celsius) => {  
  return multiplyByNineFifths(celsius) + 32;  
};
```

```
console.log("The temperature is " + getFahrenheit(15) + "°F");
```

```
// Output: The temperature is 59°F
```

JS

**Functions have so far been declared using =>
arrow function syntax** ... it's intended to **make it less
wordy!**

JS

There are other ways...

Function Declarations

Function Expressions



Declaration(1):

```
function square(number) {  
    return number * number;  
};
```

```
square(5);
```

```
// Output: 25
```

JS

Expression(2):

```
const square = function(number) {  
    return number * number;  
};
```

```
square(5);
```

```
// Output: 25
```

JS

Expression(2):

JS

```
const square = function(number) {  
    return number * number;  
};
```

```
square(5);
```

```
// Output: 25
```

Notice how we have the keyword `Function` but **no name?** That's why it's anonymous.

Arrow function syntax

```
const square = (number) => {  
  return number * number;  
};
```

```
square(5);
```

```
// Output: 25
```

Declaration

```
function square(number) {  
  return number * number;  
};
```

```
square(5);
```

```
// Output: 25
```

JS

Expression/anonymous function

```
const square = function(number) {  
  return number * number;  
};
```

```
square(5);
```

```
// Output: 25
```



So to recap....

Functions are **written to perform a task**.

Functions **take data**, perform a **set of tasks on the data**, then **return the result**.

We can **define parameters to be used** when calling the function.

When calling a function, we can **pass in arguments**, which will **set the functions parameters**.

We can use return to return the result of a function and use it elsewhere.

Learning Objectives

To understand how functions work

To write programs with functions

To write programs with all three types of functions

Activity 1:

Take this code and turn it into arrow function syntax:

```
function factorial (n) {  
    if ((n === 0) || (n === 1)) {  
        return 1;  
    } else {  
        return (n * factorial(n-1));  
    }  
}  
  
console.log(factorial(33));
```

JS

Activity 2:

JS

Edit the below snippet to include **two** parameters and a running order count updated when the function is called:

```
let orderCount = 0;

const takeOrder = (topping) => {
  console.log(`Pizza with ${topping}`);
  orderCount++;
}

takeOrder("pineapple");
```


Activity 3:

Cash machine time!

Let's create one that:

> Dispenses cash **if** your pin number is correct and your balance is equal to, or more than, the amount you're trying to withdraw!

Be Creative

JS