First thing's first

Let's create a ticket machine for a cinema

Write an if statement that checks the ages of cinema goers, and display the ticket prices:

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

Code Nation

JavaScript Fundamentals

Functions

{codenation}®

Learning Objectives

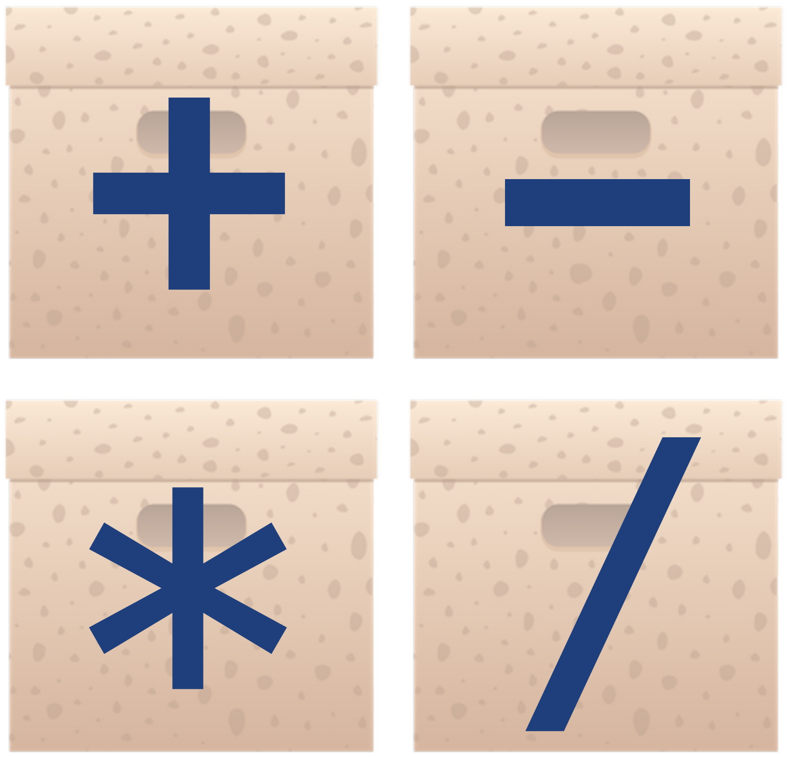
- To understand how functions work
- To write programs with functions
- To write programs with all three types of functions

Introducing Functions

Functions let us do the things we need our code to do

We call functions by using their identifiers

They break our code up into small chunks



Separate functions for each operator

Let's take this in

```
const pressGrindBeans = () => {

    console.log("Grinding for 20 seconds");

}

pressGrindBeans();
```

Let's take this in

```
const pressGrindBeans = () => {

    console.log("Grinding for 20 seconds");

}

pressGrindBeans();
```

Declare new function

Let's take this in

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

pressGrindBeans();
```

Declare new function

Start grinding the coffee

Let's take this in

```
const pressGrindBeans = () => {

    console.log("Grinding for 20 seconds");

}

pressGrindBeans();
```

Declare new function

Start grinding the coffee

Run the function pressGrindBeans

What if I have an on/off button?

Let's take this in

```javascript
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();
```

Let's take this in

Declare new variable with boolean value

```
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();
```

Let's take this in

```
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();
```

Declare new function

Let's take this in

```
let coffeeIsGrinding = false;

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

pressGrindBeans();
```

If coffeeIsGrinding is true…

Let's take this in

```
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();
```

Stop it grinding

Let's take this in

```
let coffeeIsGrinding = false;

const pressGrindBeans = () => {
    if (coffeeIsGrinding) {
        console.log("Stopping the grind");
        coffee
    } else {
        conso                                        ');
        coffeeIsGrinding = true;
    }
}

pressGrindBeans();
```

Else if coffeeIsGrinding is false…

Let's take this in

```
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();
```

Start grinding the coffee

Let's take this in

```
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();
```

Run the function pressGrindBeans

Let's take this in

```
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();
```

Parameters

… these really make functions tick

Parameters give functions their flexibility

They provide the ability to call functions to act on different data inputs

Let's take this in

```
const cashWithdrawal = (amount, accnum) => {

    console.log(`Withdrawing ${amount} from account ${accnum}`);

}

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

Activity:

Create a function that takes two parameters for a coffee order (size, type of drink)

Let’s take this in

```
const takeOrder = (size, drinkType) => {
    console.log(`Order received: ${size} ${drinkType}`);
}

takeOrder("Tall","Latte");
```

Global variables and parameters?

Let's take this in...

```
let accnumber = 50449921;

const cashWithdrawal = (amount, accnum) => {

    console.log(`Withdrawing ${amount} from account ${accnum}`);

}

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

No longer the point of no return

We can call on functions to do a job and when they've done it, they can **return** the result

Let's take this in

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

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

Let's take this in

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

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

Add up two numbers and return the answer

Let's take this in

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

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

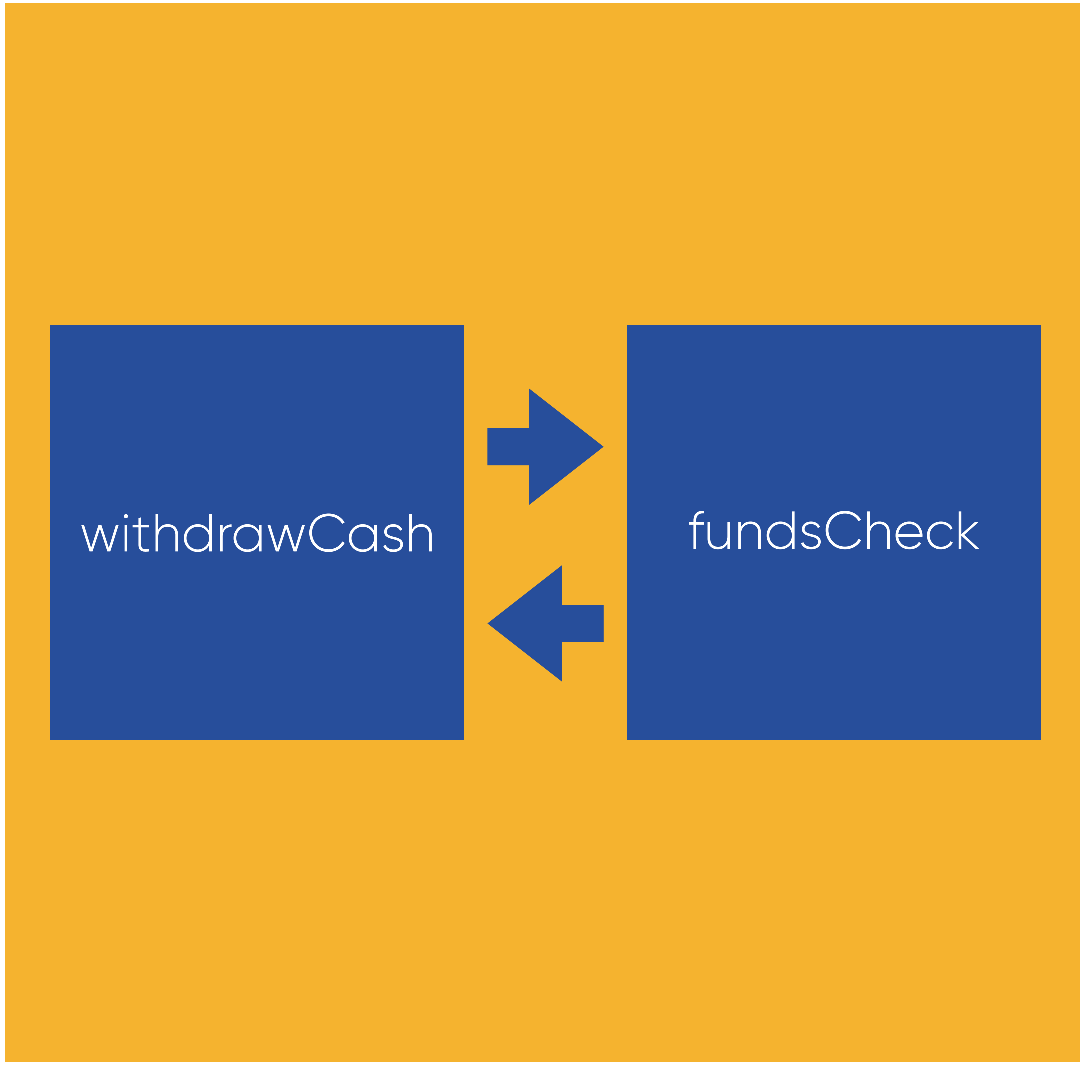
Add up two numbers, return the answer, and then print the result in the console

So, you see...

one function might call another function

and use the result of that function to achieve its goal

For example, in our wonderful cash machine, we might have something like ...



Does customer have enough funds requested?

Check and return result to withdrawCash

Let's take this in

```
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
```

Anything to declare?

We've seen => to create functions. It's called arrow function syntax and it's intended to make it less wordy when creating functions

Other ways to create functions include...

Function declarations

Function expressions

Declaration(1)

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

square(5);

// Output: 25
```

Declaration

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

square(5);

// Output: 25
```

function [name](parameters)

Declaration(2)

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

Function linked to an identifier;

call factorial to get it to do something

Expression

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

square(5);

// Output: 25
```

Create variable that stores an anonymous function

Expression

```
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.

Create variable that stores an anonymous function

Arrow function syntax

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

square(5);

// Output: 25
```

Arrow function syntax

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

square(5);

// Output: 25
```

Declaration

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

square(5);

// Output: 25
```

Expression/anonymous function

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

square(5);

// Output: 25
```

Arrow function syntax

```
const functionName=(parameters)=>{
    // do code
};
```

Declaration

```
function functionName(parameters){
    // do code
};
```

Expression/anonymous function

```
const functionName=function(parameters){
// do code
};
```

Activity:

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));
```

Activity:

Take this code and turn it into arrow function syntax

```
const factorial = (n) => {
      if ((n === 0) || (n === 1)) {
            return 1;
      } else {
            return (n * factorial(n-1));
      }
}

console.log(factorial(33));
```

Functions

Functions are written to perform a task.

Functions are written to perform a task.

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

Functions are written to perform a task.

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

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

Functions are written to perform a task.

Functions take data, perform a set of tasks on the data, and 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 function's parameters.

Functions are written to perform a task.

Functions take data, perform a set of tasks on the data, and 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 function's parameters.

We can use return to return the result of a function which allows us to call functions anywhere, even inside other functions.

Learning Objectives

- To understand how functions work
- To write programs with functions
- To write programs with all three types of functions

Activity(1):

Here's an example of a function that includes a parameter. Parameters are responsible for functions being able to work on different data inputs. Edit the snippet below to include two parameters and a running order count updated when the function is called :

```javascript
let orderCount = 0;

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

takeOrder("pineapple");
```

Activity(2):

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!