First thing's first... (I'm the realest)

Lets go back to our naughts and crosses board

Write an if statement that checks if the items on the top row meet a winning condition. So the top row are all 'o's or all 'x's.

Second thing's second

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

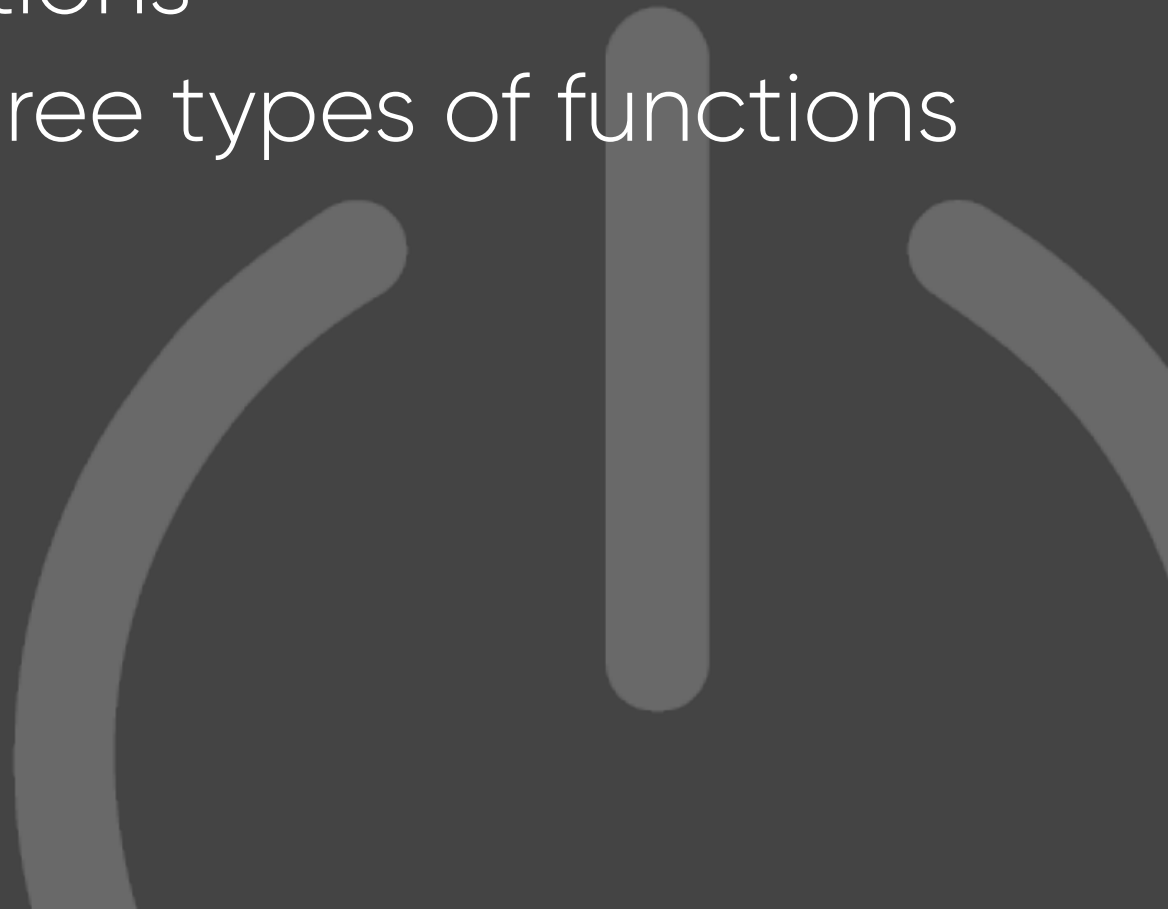
JavaScript Fundamentals

Functions

{codenation}®

Learning Objectives

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




Code is
~~dys~~functional


Functions let us
do things...

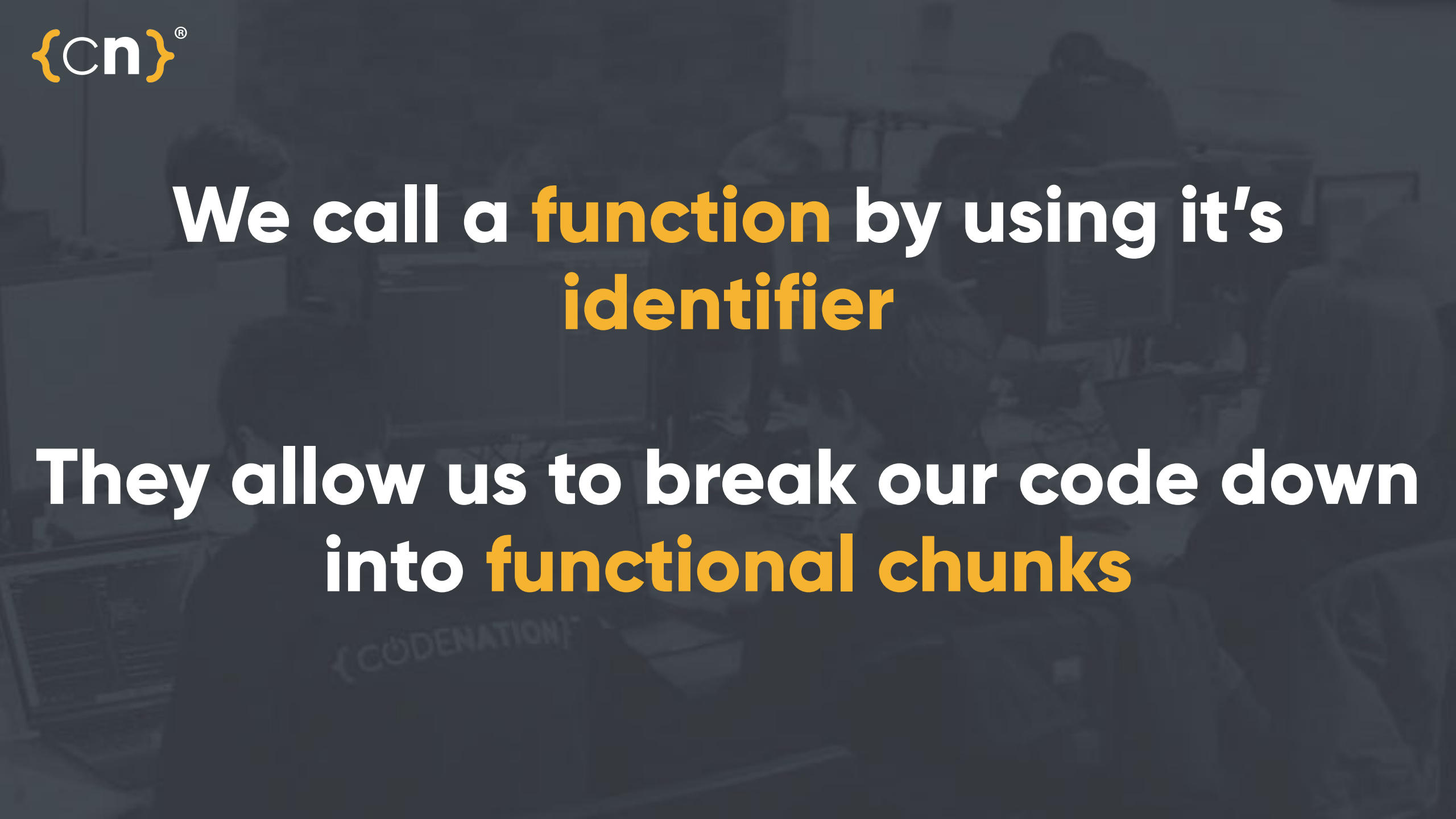

Again...


And again...

And again...


Reusability is key.


We call a function by using it's identifier


We call a function by using it's identifier
They allow us to break our code down into functional chunks

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


What if we have an on/off button?

```
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();
```


{cn}®
Parameters
... these really make functions tick

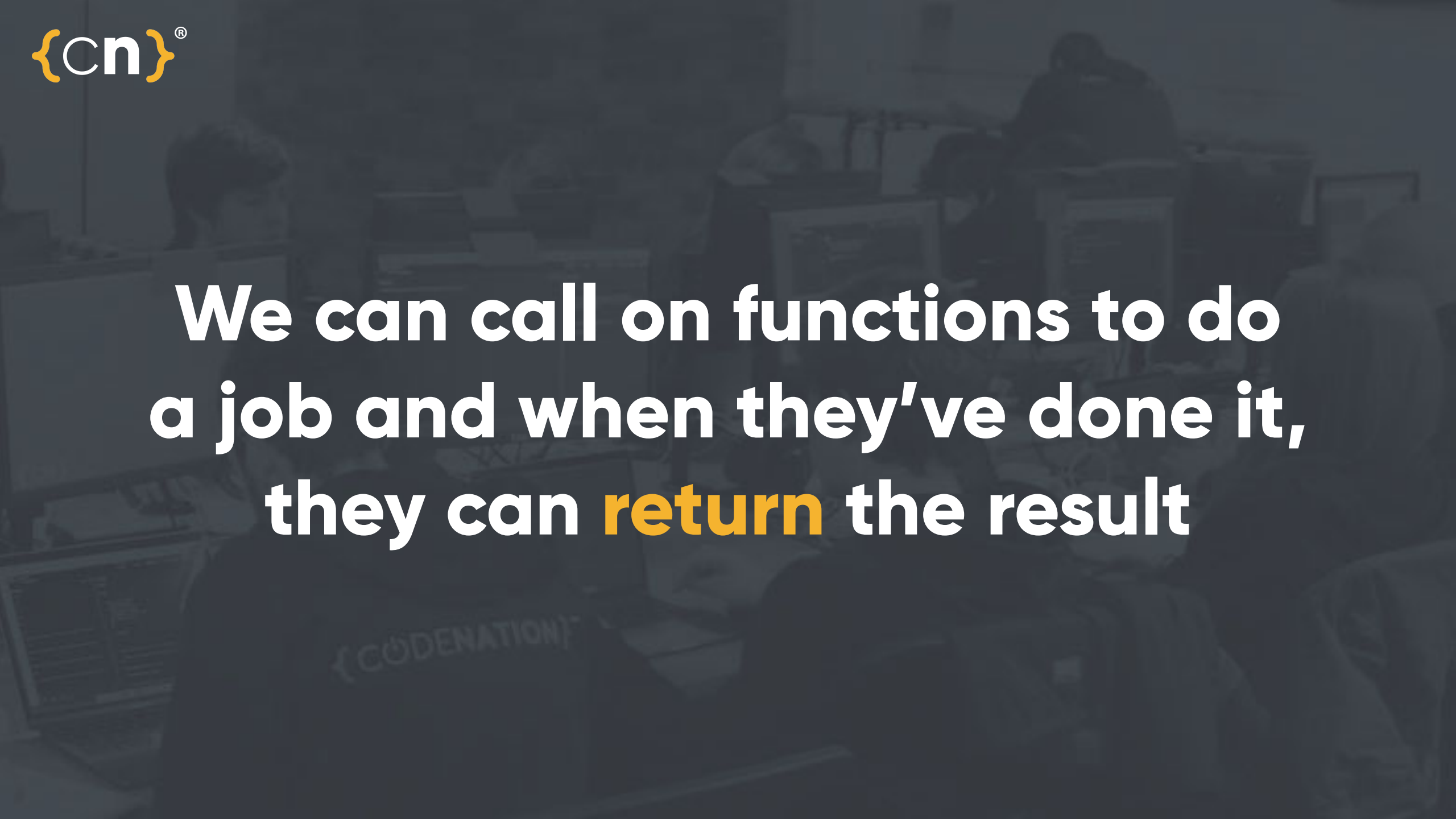

Parameters give our functions
their flexibility

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

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

}

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


What if we introduce global variables?

```
let accnumber = 50449921;

const cashWithdrawal = (amount, accnum) => {

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

}

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


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

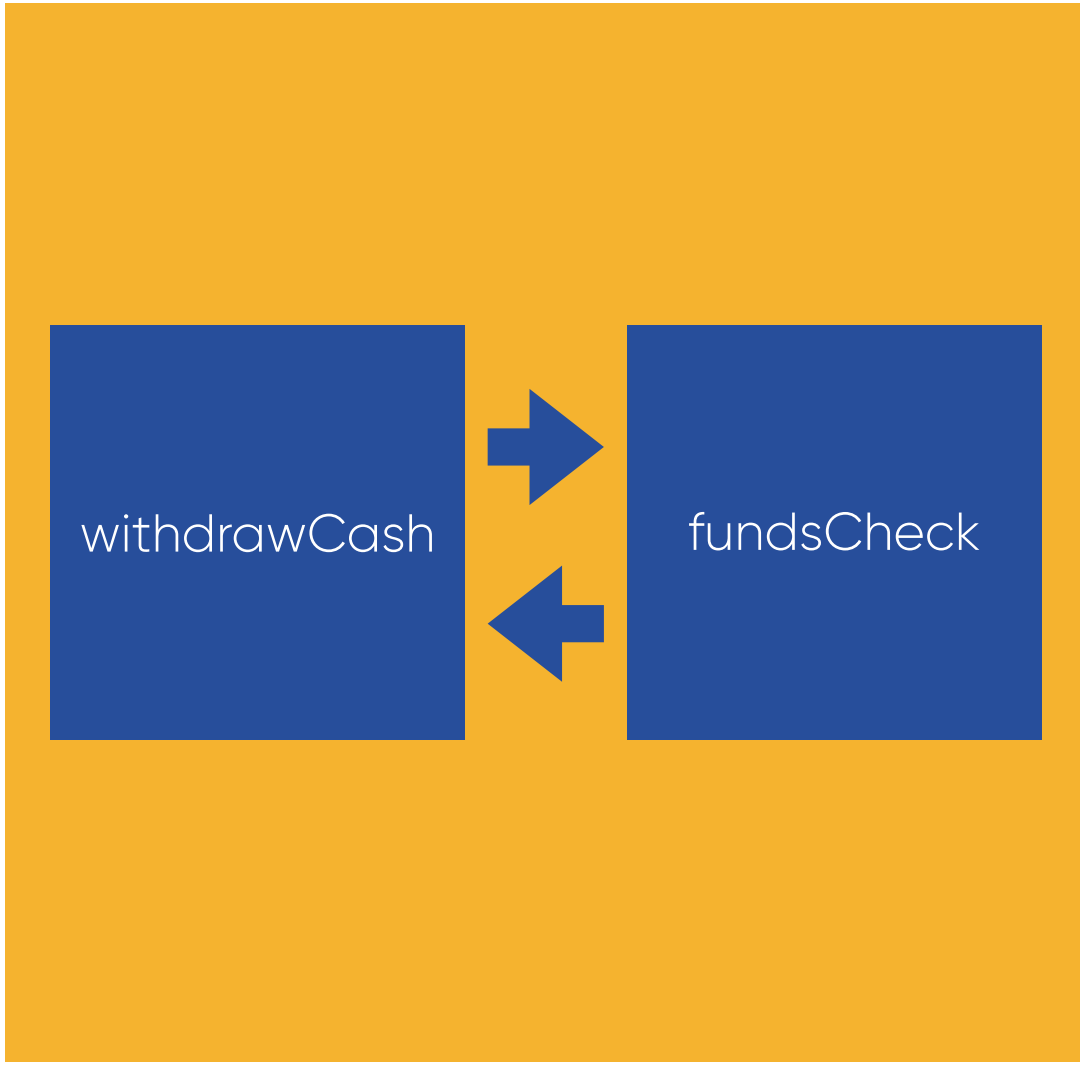
```
const addUp = (num1, num2) => {
    return num1 + num2;
}

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

In summary

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

```
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
```


{cn}®
More arrows than Robin Hood.

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

There are other ways to create functions, including...

Function declarations

&

Function expressions

Declaration(1)

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

square(5);

// Output: 25
```

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
};
```

So, the point* is...

* Arrow function.. point.. pointed arrow, get it? hehe

=>

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.

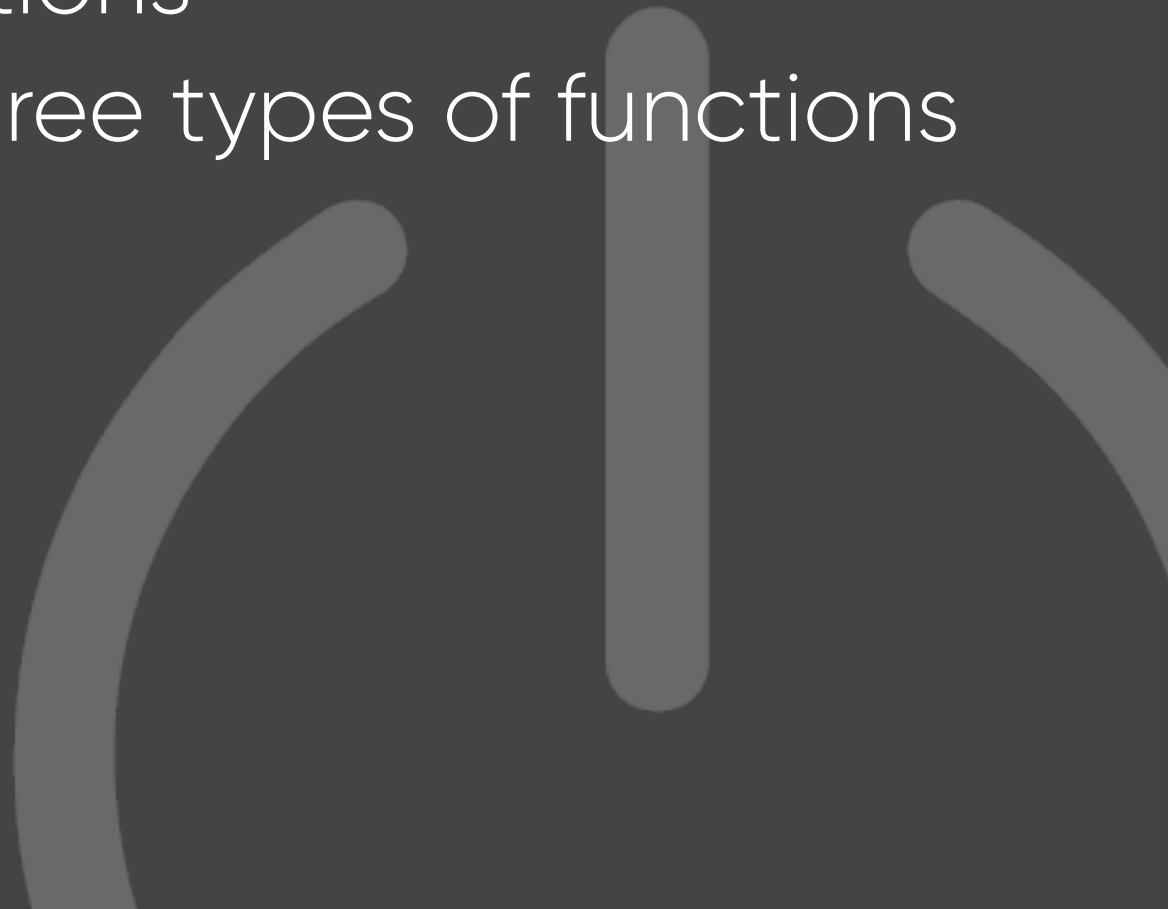
We can define parameters to be used when calling the function.

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:

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");
```

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));
```

You're done, bud.