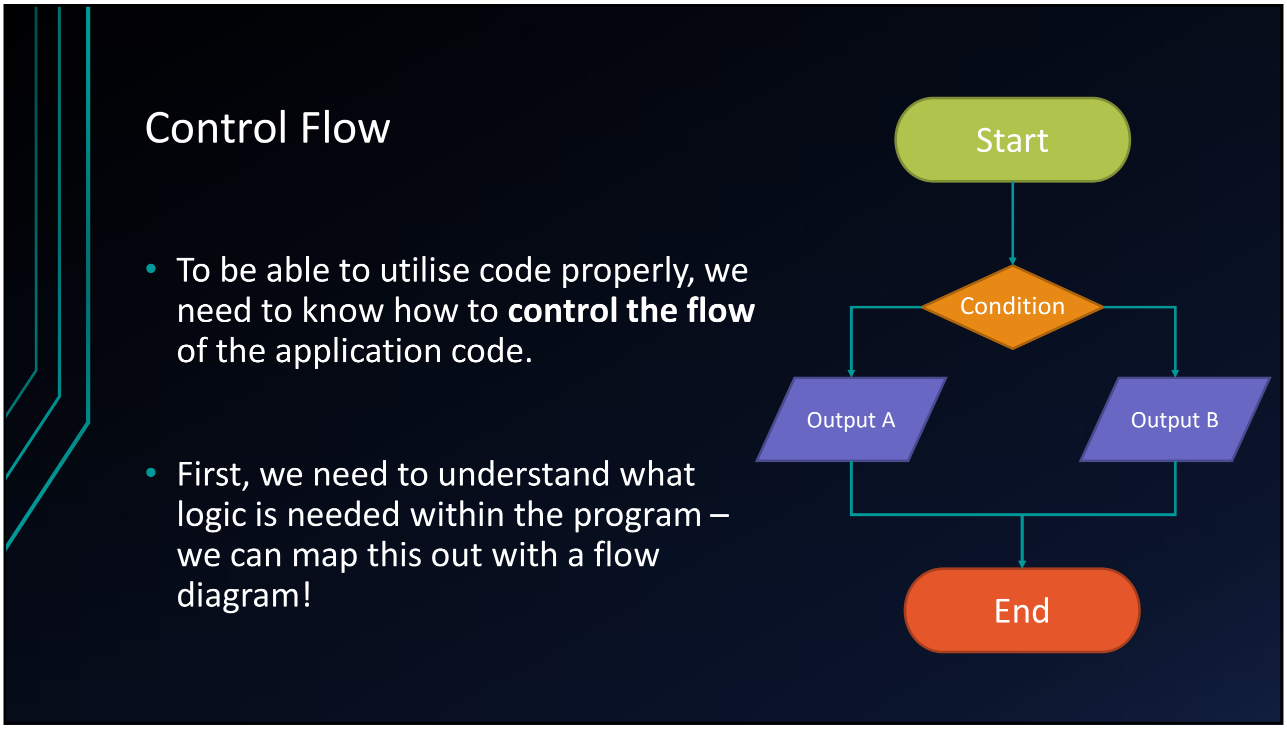
JavaScript – Part 2

A LOOK INTO CONTROL FLOW

1



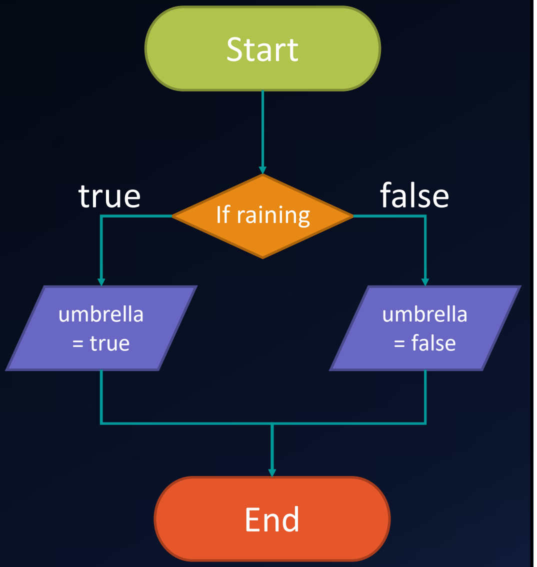
Control Flow

- To be able to utilise code properly, we need to know how to **control the flow** of the application code.
- First, we need to understand what logic is needed within the program – we can map this out with a flow diagram!

2

A Simple Flow

- Let us create our first program using control flow. Consider the following statement:
  - If it is raining, I will take an umbrella, otherwise, I will not take an umbrella
- This is our first conditional statement!



Start

true

If raining

false

umbrella = true

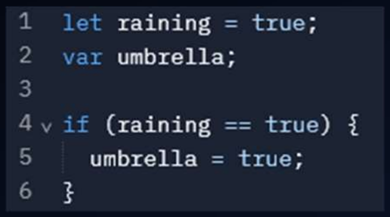
umbrella = false

End

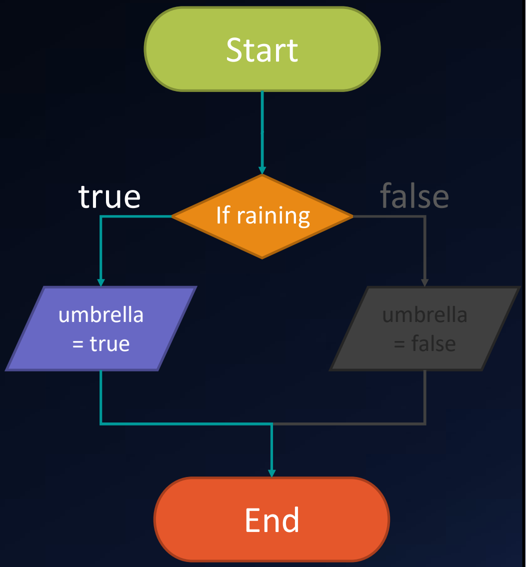
3

A Simple Flow – One Path

- Lets start our code by looking at one path through the flow diagram – if it is raining.



```
let raining = true;
var umbrella;

if (raining == true) {
  umbrella = true;
}
```



Start

true

If raining

false

umbrella = true

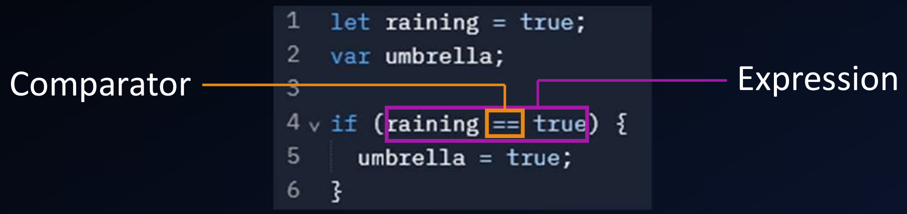
umbrella = false

End

4

Comparators

- What the code has shown is an ‘if statement’. If statements evaluate an **expression** to either true or false using a **comparator.**



```
let raining = true;
var umbrella;

if (raining == true) {
  umbrella = true;
}
```

Comparator

Expression

- In this instance, raining is set to true, so we enter the code block.

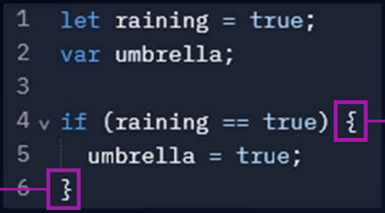
Comparison Operator

- The double equal (==) notation is used to compare the values on the left of the sign, with the values on the right of the sign. This will give us an output of either true or false.

- What is the result of the below?
  1. 10 == 10
  2. ‘tree’ == ‘branch’
  3. true == true
  4. ‘car’ == ‘car’

Code Blocks

- In the umbrella/raining example, we created an ‘if statement’ that has a code block after it denoted with curly braces { }.
- This means when the ‘if statement’ evaluates to true, the code within the code block can be executed, if it is not true, it will **skip the code within the block**.



```
let raining = true;
var umbrella;

if (raining == true) {
  umbrella = true;
}
```

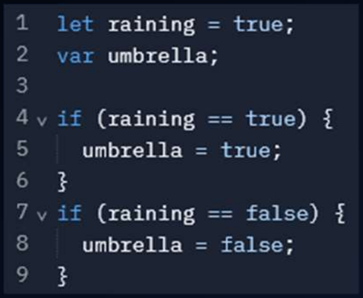
Opening code block

Closing code block

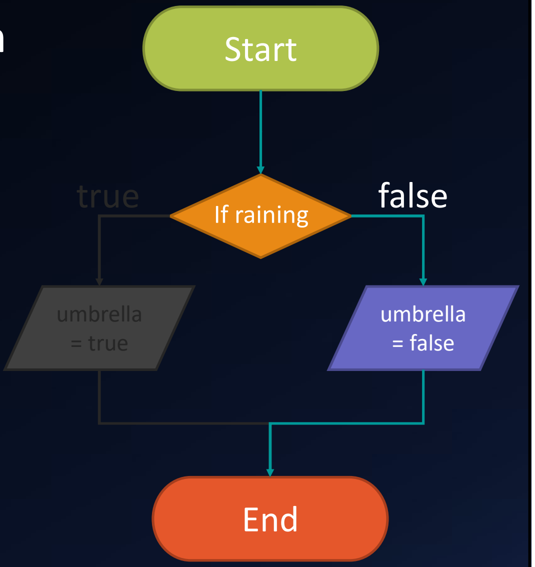
7

A Simple Flow – The Other Path

- The other path can be written out in a similar way, using another ‘if statement’.



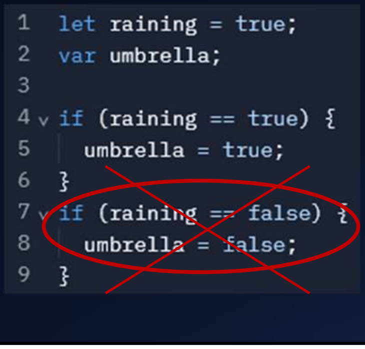
```
let raining = true;
var umbrella;

if (raining == true) {
  umbrella = true;
}
if (raining == false) {
  umbrella = false;
}
```



8

Best Practice

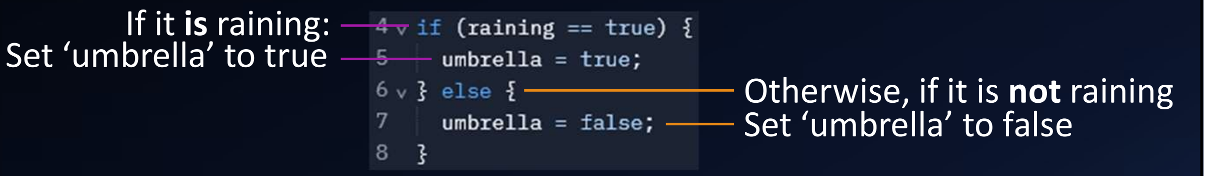
- Although this would give us the result we are looking for, whether ‘raining’ has been set to true or to false, we consider this to be bad practice!
- This is because we are asking the computer to evaluate the variable ‘raining’ twice, but we already have the answer from the first ‘if statement’. Therefore, we can use an ‘else’ clause.



```
let raining = true;
var umbrella;

if (raining == true) {
  umbrella = true;
}
if (raining == false) {
  umbrella = false;
}
```

The Else Clause

- The ‘else’ clause allows us to work with the opposite result of the ‘if statement’. You can think of ‘else’ as ‘otherwise’ or ‘if not’.
- Note that there is a separate code block for the ‘else’ clause.



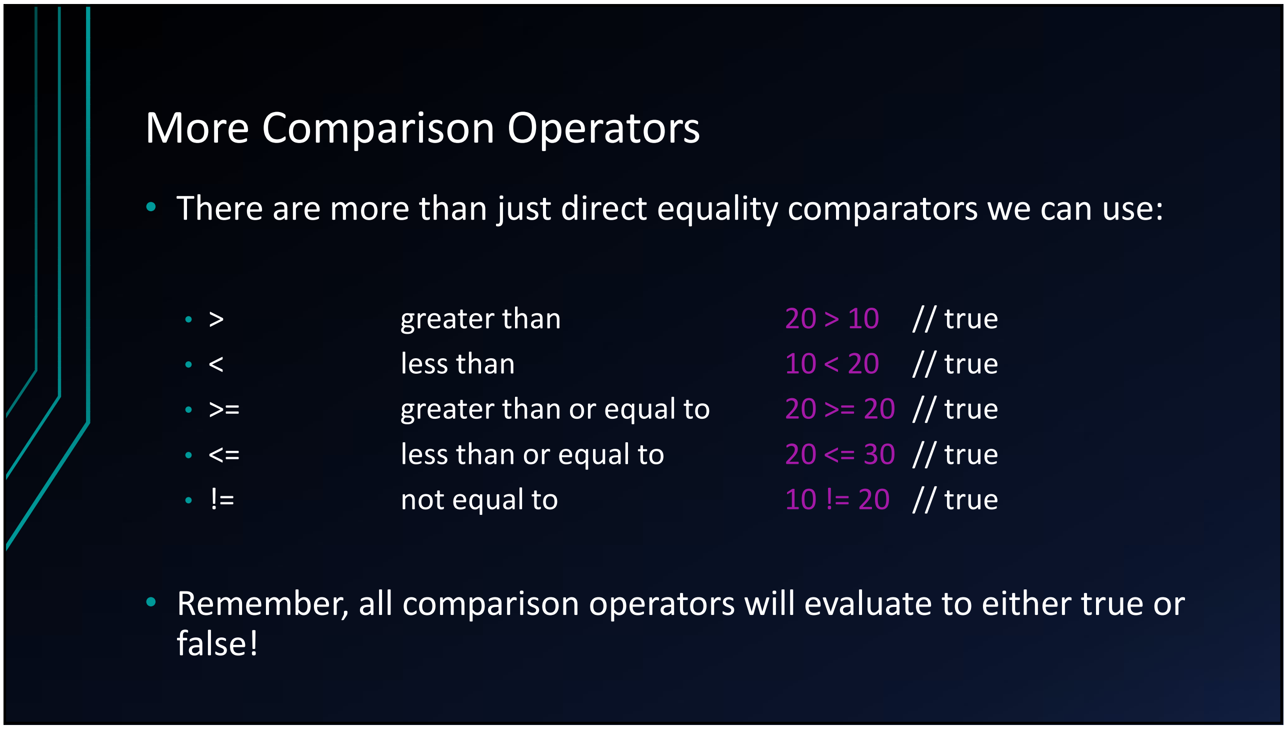
```
if (raining == true) {
  umbrella = true;
} else {
  umbrella = false;
}
```

If it **is** raining:
Set ‘umbrella’ to true

Otherwise, if it is **not** raining
Set ‘umbrella’ to false

Any Questions So Far?

11



More Comparison Operators

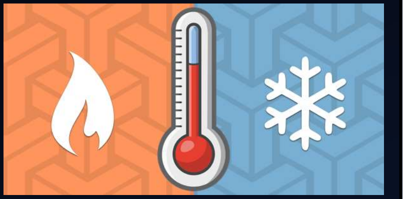
- There are more than just direct equality comparators we can use:

• >	greater than	20 > 10	// true
• <	less than	10 < 20	// true
• >=	greater than or equal to	20 >= 20	// true
• <=	less than or equal to	20 <= 30	// true
• !=	not equal to	10 != 20	// true

- Remember, all comparison operators will evaluate to either true or false!

12

Task 1 – How Warm?

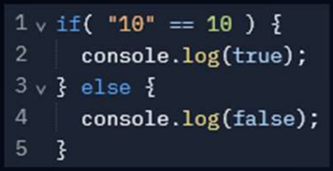


- Write a program that uses an ‘if statement’:
  - Declare a variable to hold a value for temperature
  - Use a conditional if statement to evaluate the temperature
  - Should the temperature be equal to or higher than 20, print to the console:
    - “It is hot today!”
  - Should the temperature be less than 20, print to the console:
    - “It is not very hot today!”
- If you have managed to complete the above, try comparing:
  - 10 as a string (“10”), and 10 as a number (10) – what is the result?

13

Type Comparison Operator

- Did anyone try the additional challenge on the last task?
- What is the result of the following code?

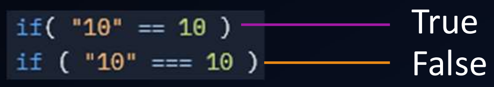


```
if( "10" == 10 ) {
    console.log(true);
} else {
    console.log(false);
}
```

14

Type Comparison Operator

- To evaluate the “10” and 10 values for their **type** (that is string and number), we need to use the triple equals operator (===).



```
if( "10" == 10 )
if ( "10" === 10 )
```

True

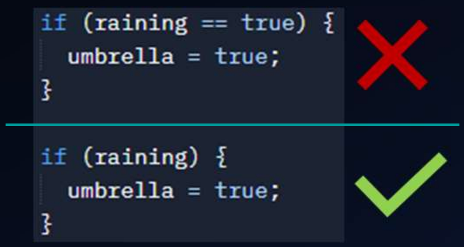
False

- The above expresses that yes, the values are the same if taken literally, but the **types are different**.

- This is called a **strict comparison**!

Shorthand – Best Practice

- When writing ‘if statements’, if the value you are evaluating is a Boolean, you do not need to add the comparison operator.



```
if (raining == true) {
  umbrella = true;
}
```

✗

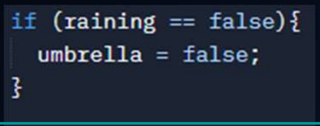
```
if (raining) {
  umbrella = true;
}
```

✓

- This is considered best practice as you are using less processing power, less RAM, and less storage when saving the file!

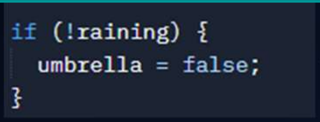
Another Shorthand Technique

- We can extend the previous shorthand technique by testing for the opposite too, using the not operator (!).



```
if (raining == false){
  umbrella = false;
}
```

✗



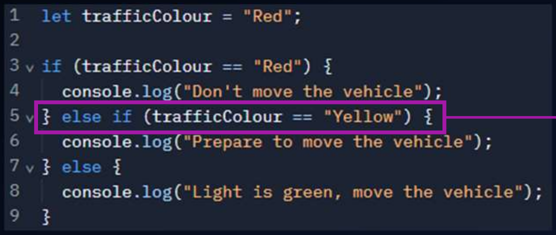
```
if (!raining) {
  umbrella = false;
}
```

✓

- The bottom statement now reads “If it is **not** raining…”.

If, else-if, else

- Now that we have a grasp of the if-else paradigm, we can build on this to evaluate more than just one statement:



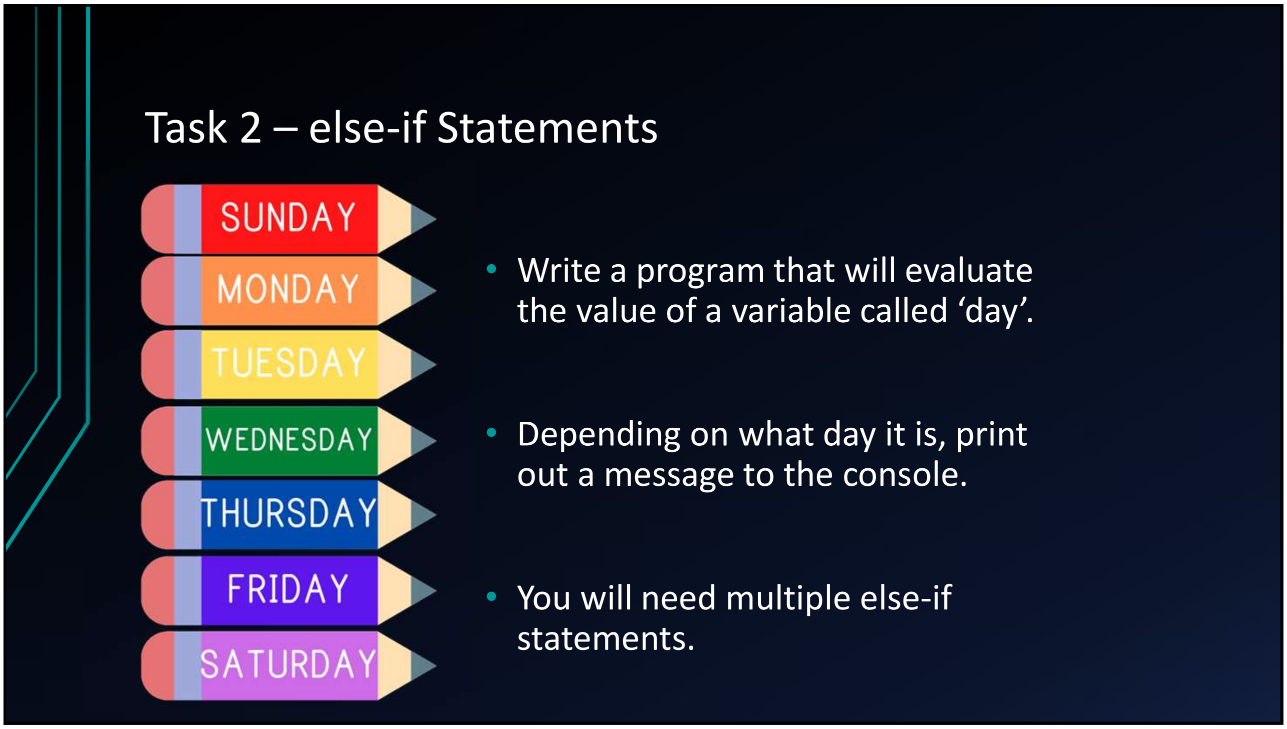
```
let trafficColour = "Red";

if (trafficColour == "Red") {
  console.log("Don't move the vehicle");
} else if (trafficColour == "Yellow") {
  console.log("Prepare to move the vehicle");
} else {
  console.log("Light is green, move the vehicle");
}
```

else-if statement

- This allows us to evaluate variables that can have more than just two values like a Boolean has.

Any Questions?

19



Task 2 – else-if Statements

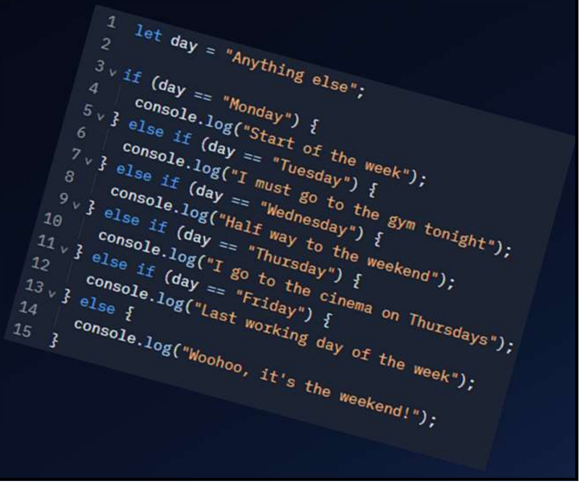
SUNDAY
MONDAY
TUESDAY
WEDNESDAY
THURSDAY
FRIDAY
SATURDAY

- Write a program that will evaluate the value of a variable called ‘day’.
- Depending on what day it is, print out a message to the console.
- You will need multiple else-if statements.

20

Large if, else-if, else-if, else-if… Statements

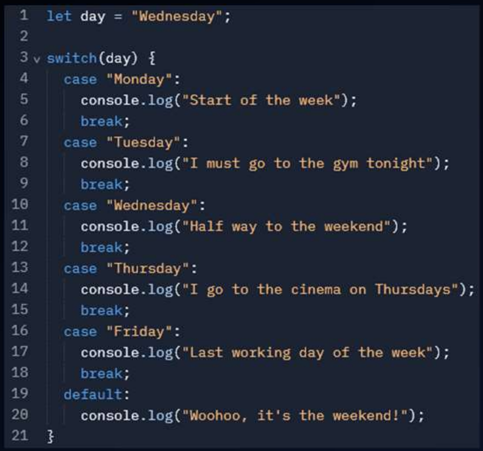
- The code below has a lot of else-if statements in them, and looks really difficult to read.



```
let day = "Anything else";

if (day == "Monday") {
  console.log("Start of the week");
} else if (day == "Tuesday") {
  console.log("I must go to the gym tonight");
} else if (day == "Wednesday") {
  console.log("Half way to the weekend");
} else if (day == "Thursday") {
  console.log("I go to the cinema on Thursdays");
} else if (day == "Friday") {
  console.log("Last working day of the week");
} else {
  console.log("Woohoo, it's the weekend!");
}
```

- Not only is it hard to understand, it also processes the 'day' variable 5 times! This is considered bad practice.
- There is a better way of doing this!

Case Switch / Switch Statement

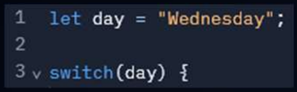


```
let day = "Wednesday";

switch(day) {
  case "Monday":
    console.log("Start of the week");
    break;
  case "Tuesday":
    console.log("I must go to the gym tonight");
    break;
  case "Wednesday":
    console.log("Half way to the weekend");
    break;
  case "Thursday":
    console.log("I go to the cinema on Thursdays");
    break;
  case "Friday":
    console.log("Last working day of the week");
    break;
  default:
    console.log("Woohoo, it's the weekend!");
}
```

- This may look scary at first, being a lot of code with a few new concepts to cover, but let's break it down 1 step at a time.

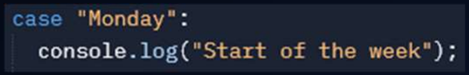
Case Switch / Switch Statement

- The word ‘switch’ is saying: “let me look at a variable, and depending on what that variable is, I will do something”.



```
let day = "Wednesday";

switch(day) {
```

- The word ‘case’ is saying: “if the value is XYZ, do this thing in particular”.

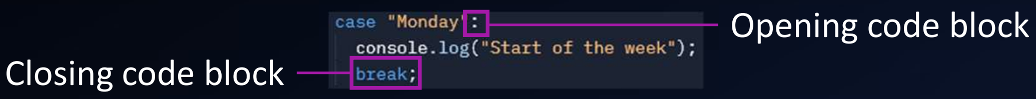


```
case "Monday":
  console.log("Start of the week");
```

- Note the **colon** after the “Monday”.

Case Switch / Switch Statement

- The ‘break’ keyword stops the code from executing any more within the switch statement. You can think of anything between the colon (:) and the ‘break;’ as a code block.

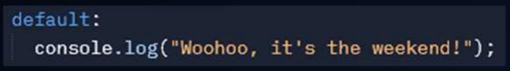


```
case "Monday":
  console.log("Start of the week");
  break;
```

- Anything between those two will be executed, should the case be “Monday”, in this instance.

Case Switch / Switch Statement

- If the value of the variable being switched does not match any of the cases presented in the switch statement, then the ‘default’ case is used.



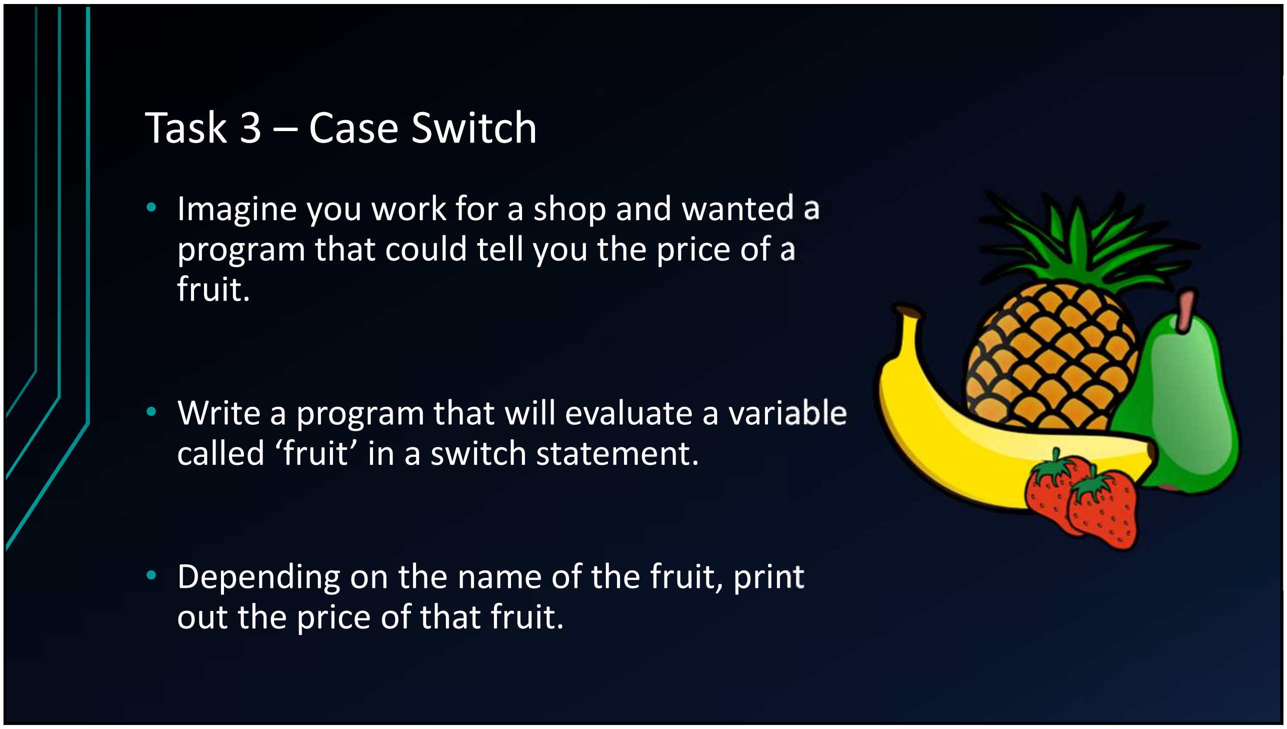
```
default:
  console.log("Woohoo, it's the weekend!");
```

- A final note on case switches:
  - Case switches evaluate true/false like an ‘if statement’, but cannot do other operators such as greater than (>) etc. It is in fact **strict comparison** so it will also test the type of the variable!

25

Any Questions?

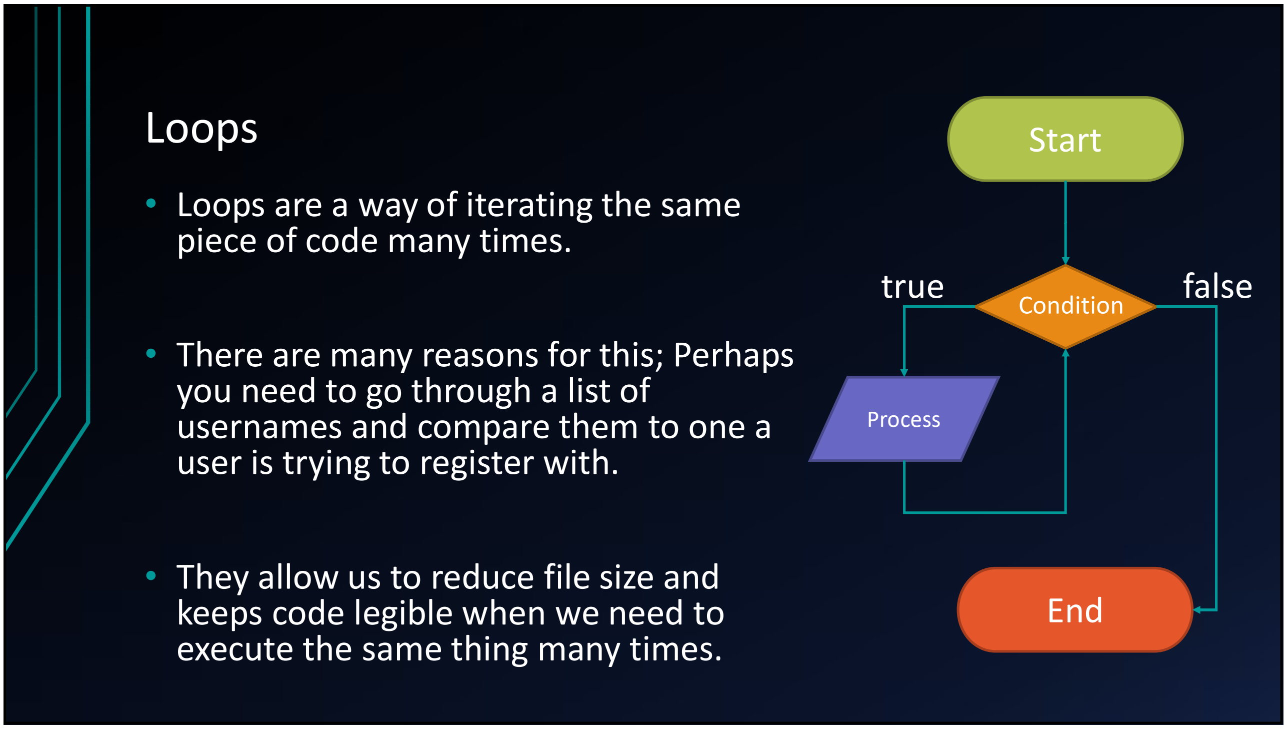
26



Task 3 – Case Switch

- Imagine you work for a shop and wanted a program that could tell you the price of a fruit.
- Write a program that will evaluate a variable called ‘fruit’ in a switch statement.
- Depending on the name of the fruit, print out the price of that fruit.

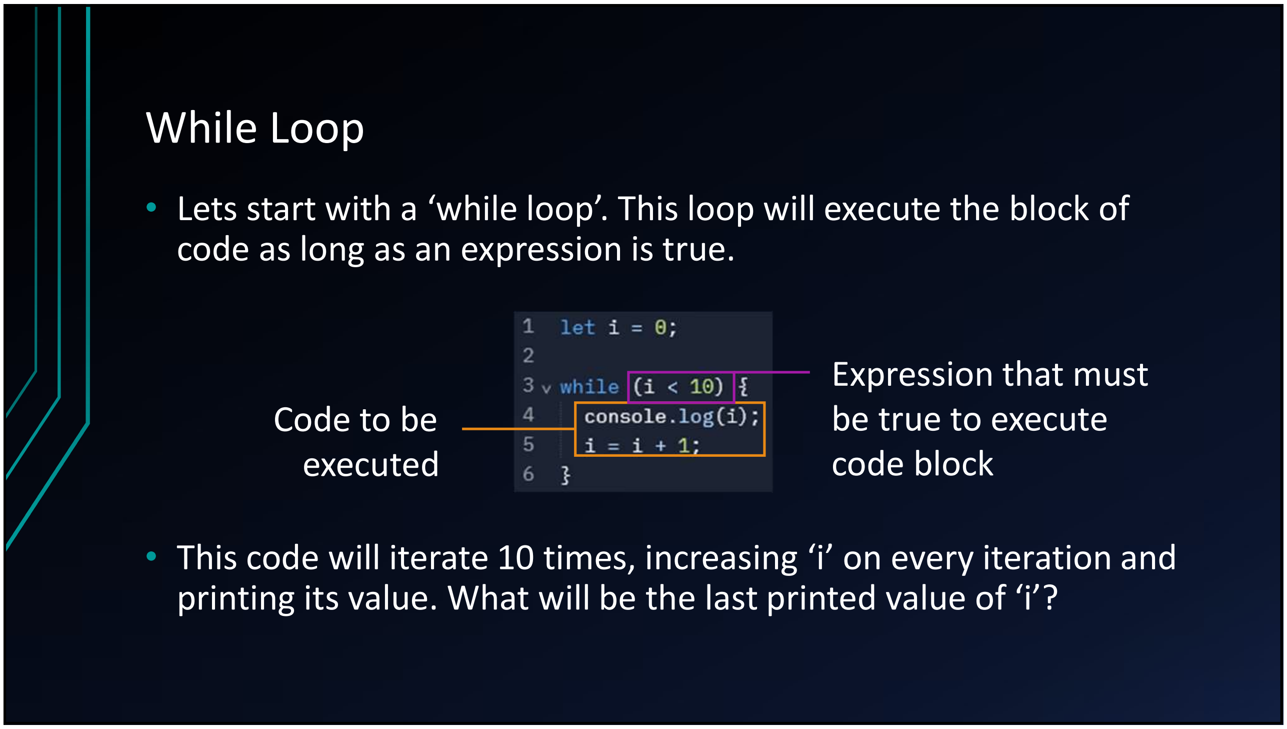
27



Loops

- Loops are a way of iterating the same piece of code many times.
- There are many reasons for this; Perhaps you need to go through a list of usernames and compare them to one a user is trying to register with.
- They allow us to reduce file size and keeps code legible when we need to execute the same thing many times.

28



While Loop

- Lets start with a ‘while loop’. This loop will execute the block of code as long as an expression is true.

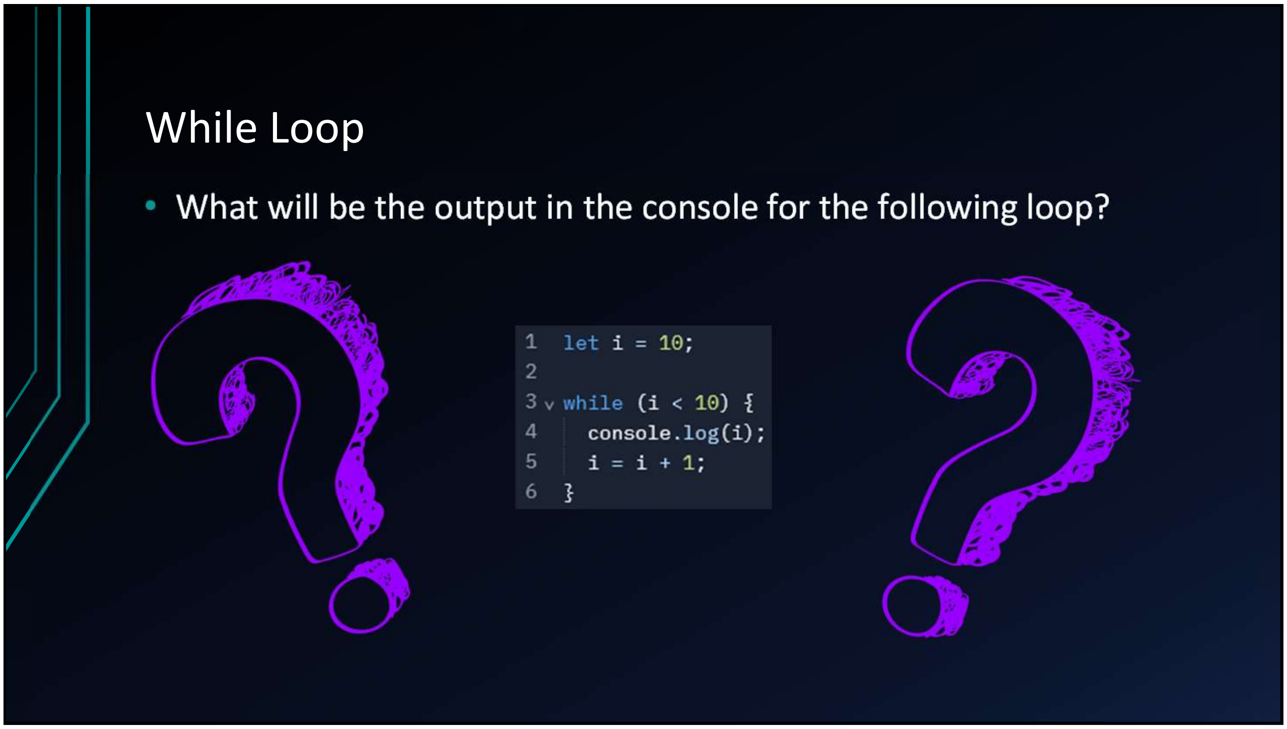
```
let i = 0;

while (i < 10) {
  console.log(i);
  i = i + 1;
}
```

Code to be executed

Expression that must be true to execute code block

- This code will iterate 10 times, increasing ‘i’ on every iteration and printing its value. What will be the last printed value of ‘i’?

29

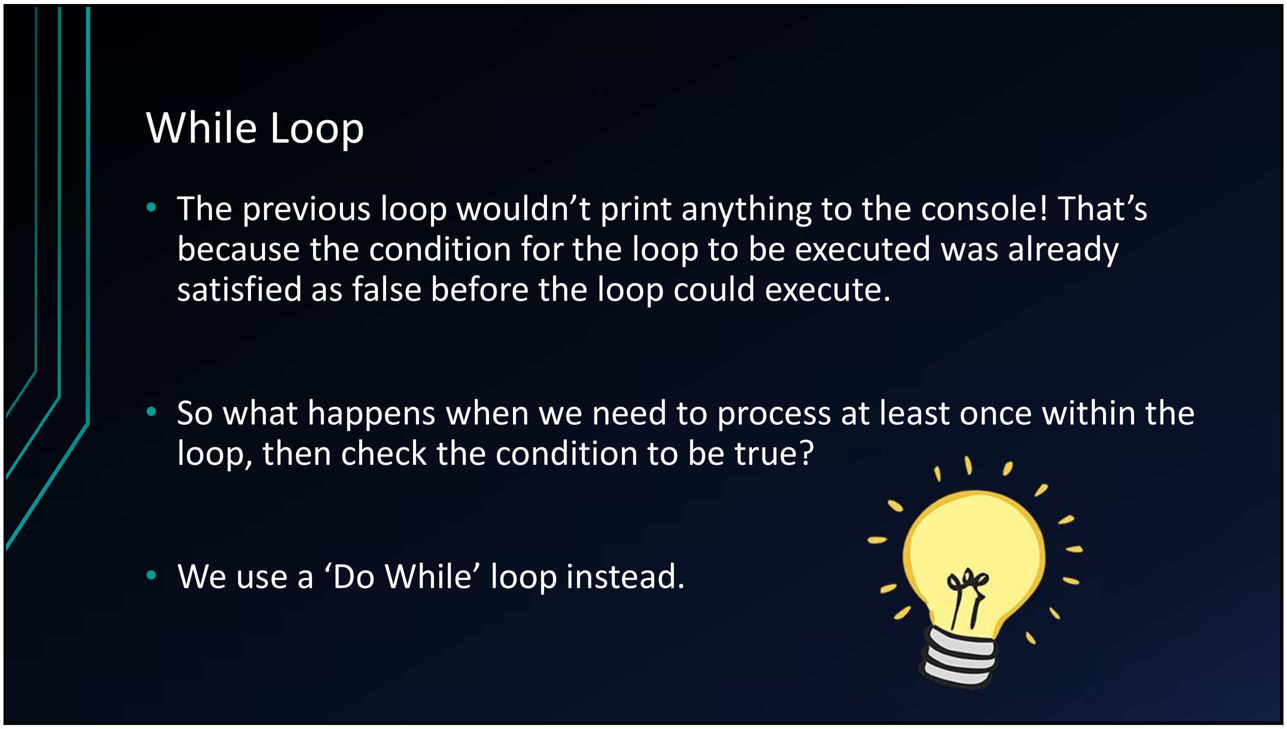


While Loop

- What will be the output in the console for the following loop?

```
let i = 10;

while (i < 10) {
  console.log(i);
  i = i + 1;
}
```

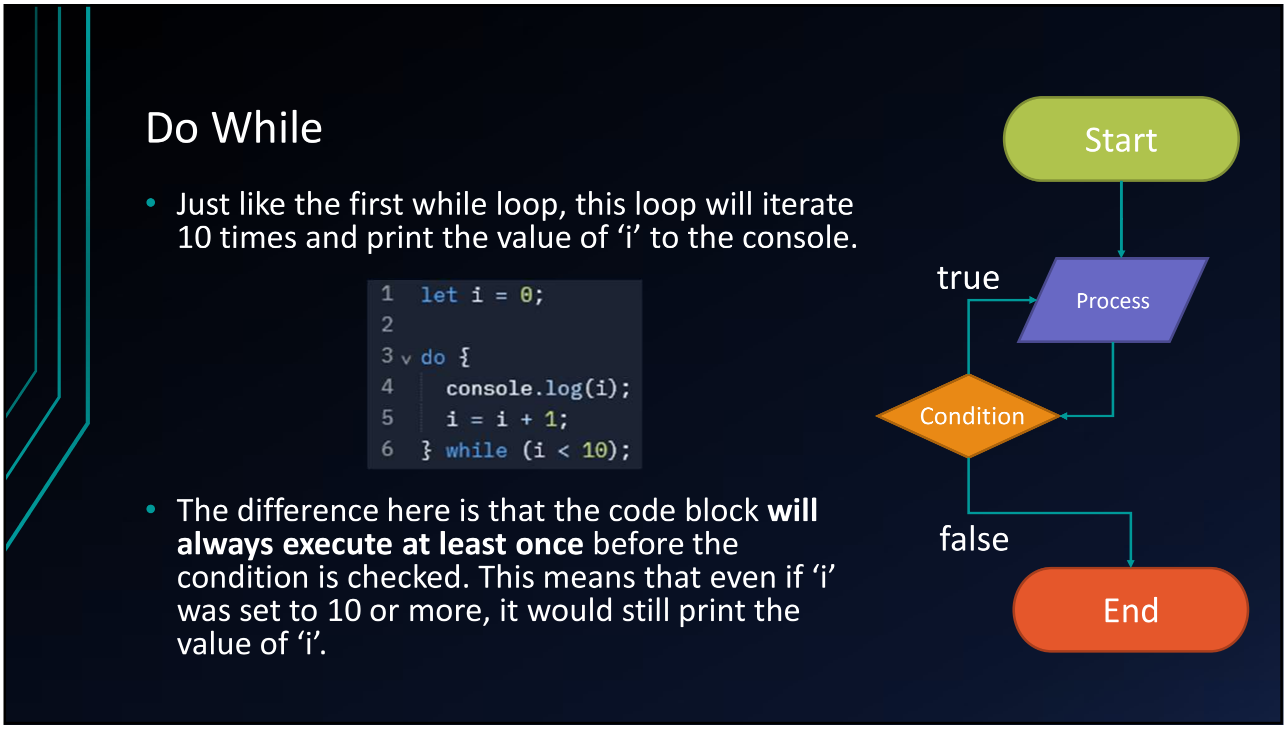
30



While Loop

- The previous loop wouldn’t print anything to the console! That’s because the condition for the loop to be executed was already satisfied as false before the loop could execute.
- So what happens when we need to process at least once within the loop, then check the condition to be true?
- We use a ‘Do While’ loop instead.

31



Do While

- Just like the first while loop, this loop will iterate 10 times and print the value of ‘i’ to the console.

```
let i = 0;

do {
    console.log(i);
    i = i + 1;
} while (i < 10);
```

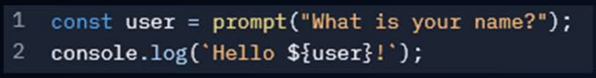
- The difference here is that the code block **will always execute at least once** before the condition is checked. This means that even if ‘i’ was set to 10 or more, it would still print the value of ‘i’.

32

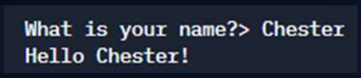
Any Questions?

Prompt – User Input

- Although user input isn't normally captured from the console with JavaScript as we would normally be capturing their inputs from a HTML form, we are able to get information from the user with the 'prompt()' method (more on methods later).

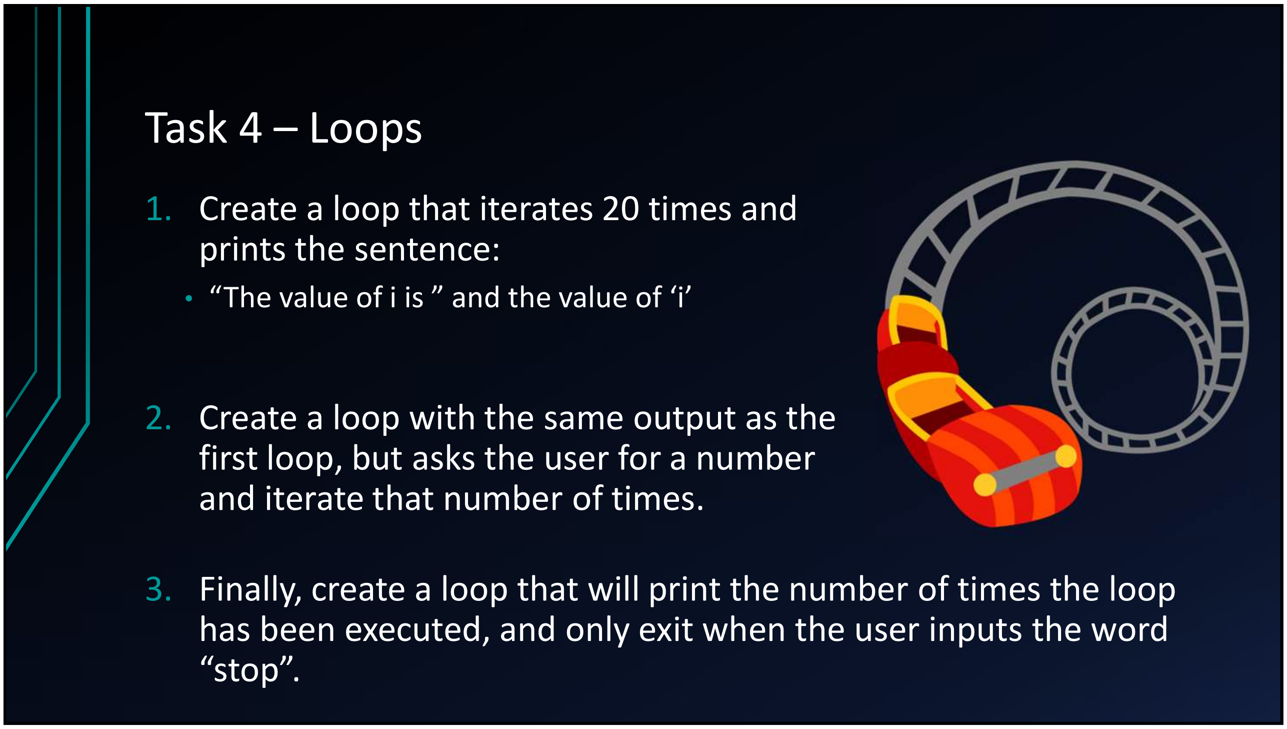


```javascript
const user = prompt("What is your name?");
console.log(`Hello ${user}!`);
```



```
What is your name?> Chester
Hello Chester!
```

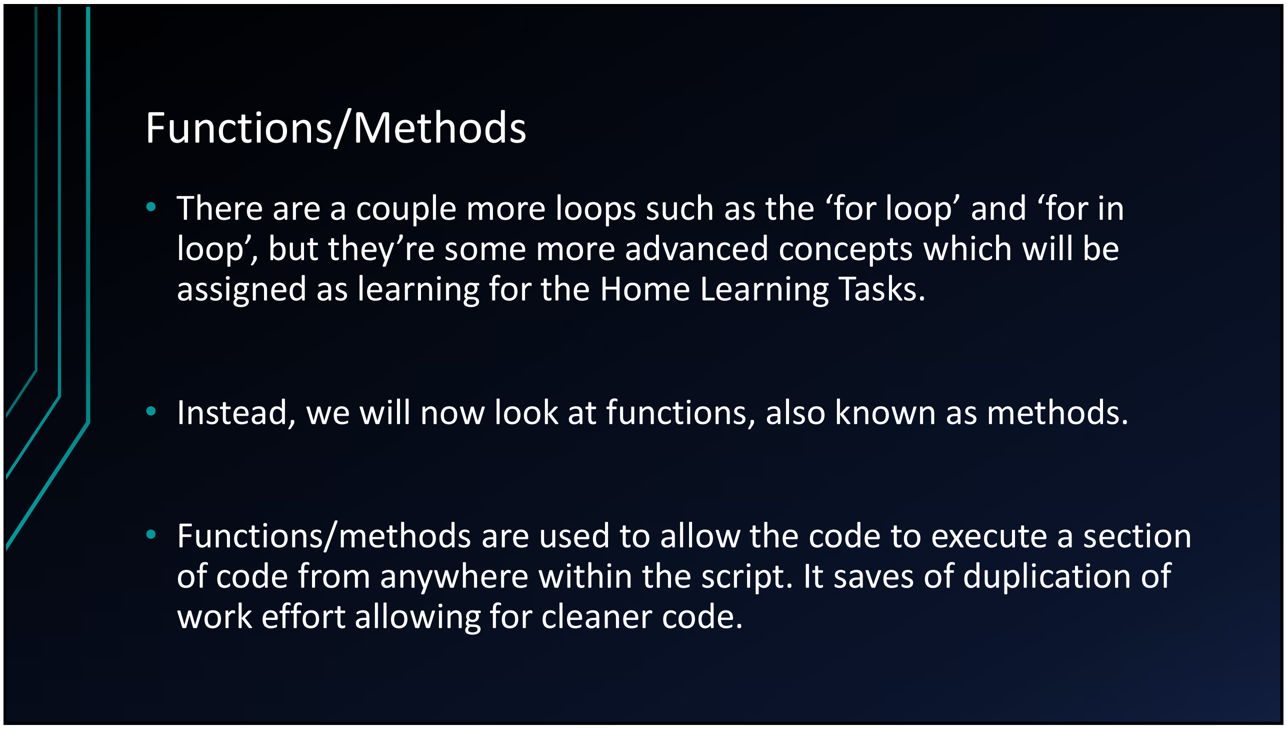
- With the inputted information, we are able to store it into a variable, in this case, a constant called "user".



Task 4 – Loops

1. Create a loop that iterates 20 times and prints the sentence:
   - “The value of i is ” and the value of ‘i’
2. Create a loop with the same output as the first loop, but asks the user for a number and iterate that number of times.
3. Finally, create a loop that will print the number of times the loop has been executed, and only exit when the user inputs the word “stop”.

35



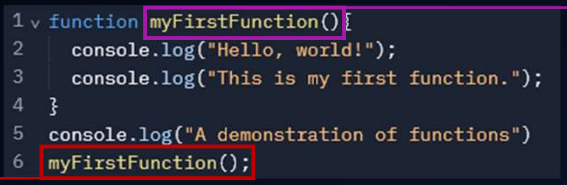
Functions/Methods

- There are a couple more loops such as the ‘for loop’ and ‘for in loop’, but they’re some more advanced concepts which will be assigned as learning for the Home Learning Tasks.
- Instead, we will now look at functions, also known as methods.
- Functions/methods are used to allow the code to execute a section of code from anywhere within the script. It saves of duplication of work effort allowing for cleaner code.

36

Functions/Methods

- When writing a function, we start with the ‘function’ key word followed by a name for that function, as well as parenthesis “ ( ) ”.



```
1 function myFirstFunction(){
2   console.log("Hello, world!");
3   console.log("This is my first function.");
4 }
5 console.log("A demonstration of functions")
6 myFirstFunction();
```

Name of the function

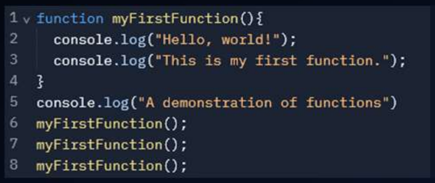
Executing the function

- The code in this example will **not** execute lines 1 to 4 first, but instead will execute line 5, then 6, then 1 to 4

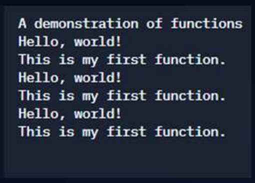
37

Functions/Methods

- We can run the same function several times without having to write the same code over and over again.



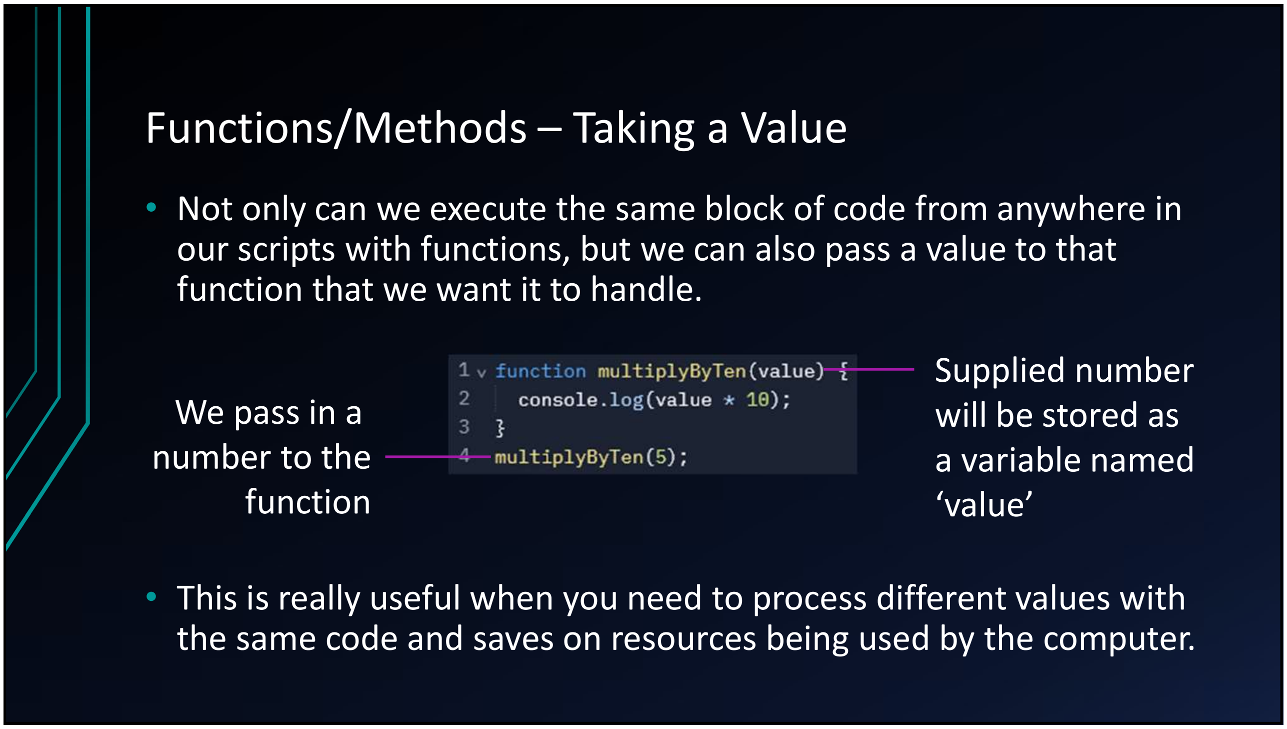
```
function myFirstFunction(){
  console.log("Hello, world!");
  console.log("This is my first function.");
}
console.log("A demonstration of functions")
myFirstFunction();
myFirstFunction();
myFirstFunction();
```



```
A demonstration of functions
Hello, world!
This is my first function.
Hello, world!
This is my first function.
Hello, world!
This is my first function.
```

- What advantages do you think this has?

38



Functions/Methods – Taking a Value

- Not only can we execute the same block of code from anywhere in our scripts with functions, but we can also pass a value to that function that we want it to handle.

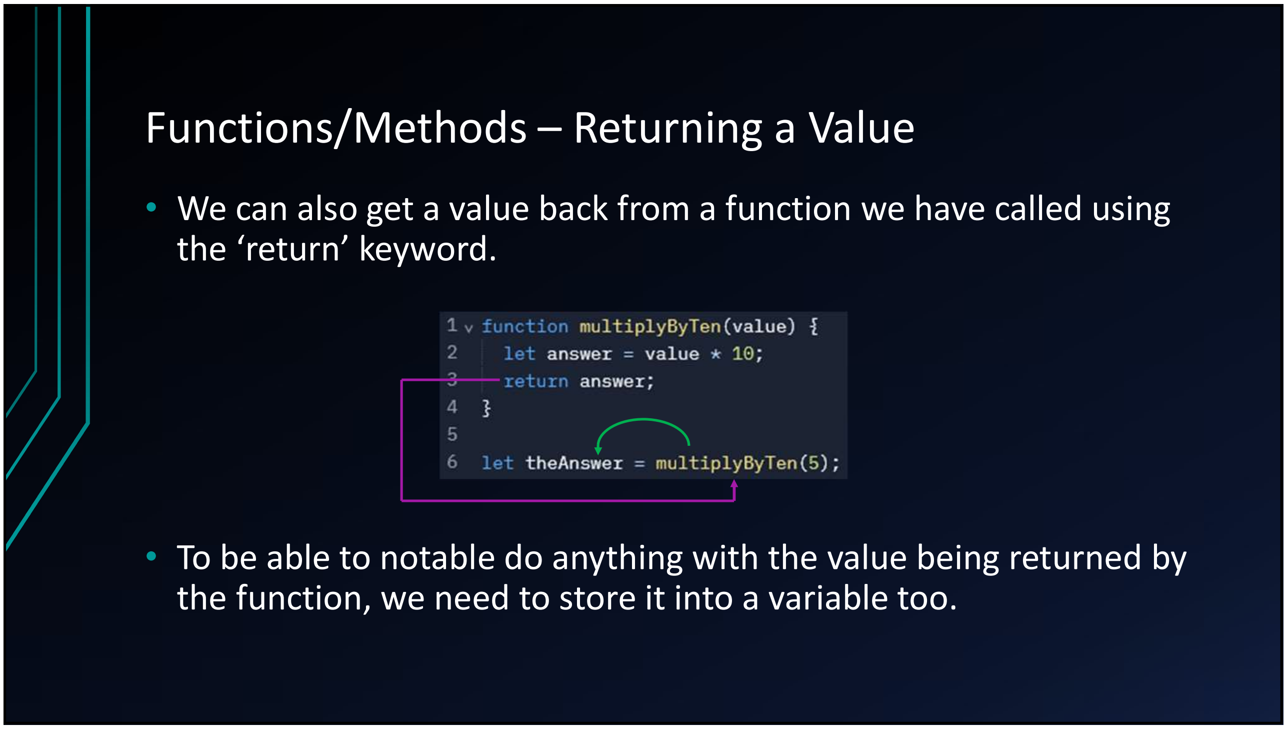
```
function multiplyByTen(value) {
  console.log(value * 10);
}
multiplyByTen(5);
```

We pass in a number to the function

Supplied number will be stored as a variable named 'value'

- This is really useful when you need to process different values with the same code and saves on resources being used by the computer.

39



Functions/Methods – Returning a Value

- We can also get a value back from a function we have called using the 'return' keyword.

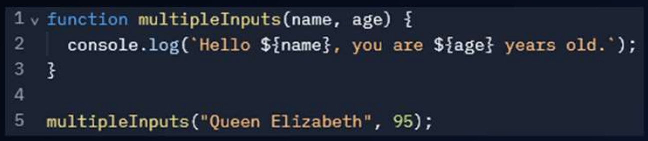
```
function multiplyByTen(value) {
  let answer = value * 10;
  return answer;
}

let theAnswer = multiplyByTen(5);
```

- To be able to notable do anything with the value being returned by the function, we need to store it into a variable too.

40

Functions/Methods – Multiple Inputs

- Finally, we are also able to take in multiple inputs into a function by separating them with a comma.



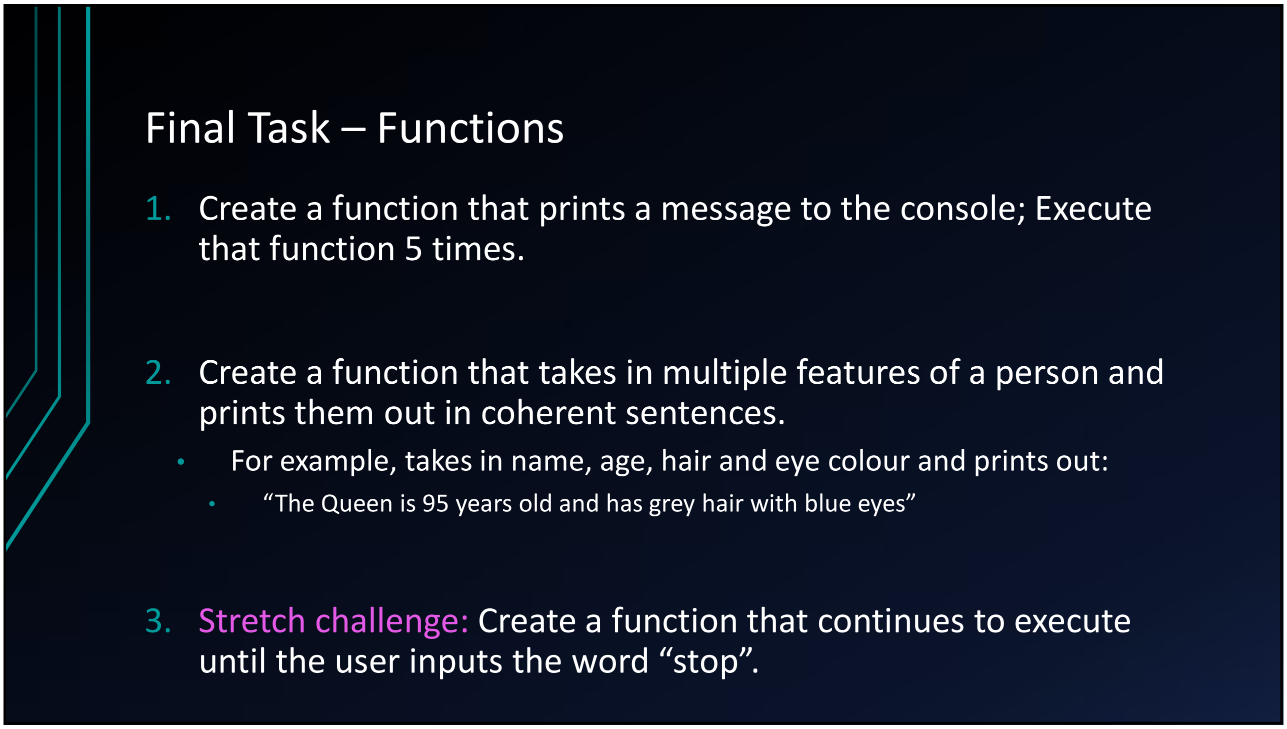
```
function multipleInputs(name, age) {
  console.log(`Hello ${name}, you are ${age} years old.`);
}

multipleInputs("Queen Elizabeth", 95);
```

- However, please note that you can only ever return a **single value**!

41

Any Questions?

42



Final Task – Functions

1. Create a function that prints a message to the console; Execute that function 5 times.

2. Create a function that takes in multiple features of a person and prints them out in coherent sentences.
   - For example, takes in name, age, hair and eye colour and prints out:
     - “The Queen is 95 years old and has grey hair with blue eyes”

3. Stretch challenge: Create a function that continues to execute until the user inputs the word “stop”.