1. Intro to lessons 2-10

1.1. use **alert** to show a message

This 1st lesson contains 5 tasks that give examples **JavaScript** code you'll learn to read and write in lessons 2-10.

Lesson 2 teaches you how to display message boxes using the **alert** command. The code editor contains an example alert command.

Do the following to complete your 1st task:

1. On line 1 in the code editor, replace `???` with your name.
2. Click the button to run the code.

1.2. run code that uses **prompt** to ask a question

In lessons 3 and 4 you'll learn to do calculations with JavaScript and in lesson 8 you'll use **prompt** commands to ask questions and **variables** to store answers.

In the example code in the editor, line 1 contains the following: `//This code calculates your age in months`

In JavaScript, lines of code that begin with `//` are **comments** that explain the code to humans. They are ignored when the computer runs the code.

1. Click to run the code.
2. Type your answer to the questions then click **OK**.

Click and answer the questions

1.3. use commands to move a robot

This course includes several robot challenges that show you graphically how the computer runs your code.

The code in the editor contains a command that moves a robot forward 3 squares. The robot should move to the square marked **X**, however, the code has a **bug** (an error).

1. Click to see what happens.
2. Fix the error.
3. Click again.

1.4. run an example project

Code Avengers teaches you both how to write code ... and how to **think** like a coder.

This includes practicing finding and fixing **bugs** (mistakes) in broken code.

For example, the code in the editor has an error on line 1. It needs `//` at the start because it is a **comment** that should be ignored by the computer.

1. Put `//` at the start of line 1. Make sure you use `//` NOT `\\`.
2. Click to run code.
3. Type your answer to the question, then click **OK**.
4. **(Optional)** Try to add your own question.

1.5. review quiz

Every 10th lesson is a review quiz like the one in this task. After you complete each group of 10 lessons, there are projects in which you'll build your own apps and games.

In JavaScript code you will often see `;` (semi-colons) at the end of most lines. However, since they are mostly optional, we start using them in lesson 10.

Review Quiz Questions:

1. What JavaScript command displays a **message box**?
2. Which symbol means **multiply**?
3. What JavaScript command displays a **question box**?
4. Which is another word for **errors** in a computer program?

2. Display messages with [[alert]]

2.1. complete the alert message

The code `alert('Hello')` uses an **alert** command to display the message **Hello**.

`alert` is followed by `(`, then the message inside quotes `' '`, then `)`.

For this task:

1. Replace `M-------t` on line 1 of the code with the name of the top **software** company.
   If you're not sure, have a guess!
2. Click to run the code.

There is big **$$$** in creating **software**. Good **computer programmers** make **$100k+** working for companies like those mentioned in this lesson.

2.2. use `\n` to put text on a new line

`\n` is a special character called a **newline**. It splits a message into multiple lines.

1. Put `\n` before each of the 2 company names to show them on separate lines in the **alert** message.
2. Click to run the code and look carefully at the **alert** message.

When typing the newline character (`\n`), use backslash (`\`) **NOT** forward slash (`/`).

The `\` key, which also has the pipe symbol `|` is above `Enter` on US keyboards, and next to `Z` for UK keyboards.

2.3. add data on a new line in an alert message

For this task:

1. Add the 3rd highest company, which was **Oracle** with **US$20.96 billion**.
2. Display all 3 companies on separate lines in the alert message.
3. Click to run the code.

2.4. add another alert command

1. On l#1 add a 2nd `alert` command.
2. Make the 2nd **alert** message show the top 3 websites on separate lines:
   Google, **Facebook**, **YouTube**.
3. When you are done, click .

2.5. use `" "` instead of `' '`

Using `' '` or `" "` around the message does the same thing.

1. Put double quotes `"` around the alert messages instead of single quotes `'`.
2. Click

3. Order of operation math rules

3.1. do a simple calculation

JavaScript follows the regular math rules for **order of operations**. Programmers must understand these rules. If you can calculate `2 * (16 - 4) / 3 + 2 * 4` in your head (**check the answer here**) you may click here to skip to lesson 4.

For this task, use an `alert` to display the answer to a simple calculation.

1. Replace the `0` on line 2 with your age; **don't** put quotes around the numbers.
2. Click to see how many years until your 100th birthday.

When you click the computer runs the code line by line. Lines starting with `//` (e.g. line 1) are **comments** that explain the code to humans; they are ignored by the computer.

3.2. do addition and subtraction

As a **computer programmer** it is good if you can do basic math in your head. + and - are computed from left to right.

For example, to calculate `8 - 2 + 4 - 1` start with 8 and subtract 2 to get 6; add 4 to get 10; subtract 1 to get 9 as the final answer.

Compute the following and type the answers in the boxes. Click to check your answers.

1. `1 + 2 + 4` =
2. `8 - 4 - 2` =
3. `8 + 2 - 4 + 1` =

3.3. do multiplication and division

* means multiply, / means divide. * and / are also computed from left to right.

For example, to calculate `2 * 2 / 4 * 8` multiply 2 by 2 to get 4; divide by 4 to get 1; multiply by 8 to get 8.

Compute the following and type the answers in the boxes. Click to check your answers.

1. `8 * 4` =
2. `32 / 2` =
3. `2 * 2 * 2 * 2 / 16 * 2` =

3.4. do operations in the correct order

* and / are done before + and -.

For example, to calculate `64 - 4 * 8 + 16 / 2` first do `4 * 8` which is 32, and `16 / 2` which is 8; this leaves to `64 - 32 + 8` which equals 40.

Compute the following and type the answers in the boxes. All answers are whole numbers. Click to check your answers

1. `8 * 2 - 2 * 4` =
2. `4 + 4 / 2 + 4` =
3. `2 * 2 - 8 / 2 + 2` =

3.5. do operations in the correct order

Operations in `( )` take priority over other operations.

Remember * means multiply, / means divide.

Compute the following and type the answers in the boxes. All answers are whole numbers. Click to check your answers.

1. `16 - (1 + 2 + 4 + 8)` =
2. `(4 * 2 * 4) / (2 * 4 * 2)` =
3. `64 / 2 - (16 + 8)` =

4. Basic math calculations

4.1. do a power calculation by multiplying

Before we do more interesting stuff, let's do a little more math.

In computers, a **kilobyte** can have 2 meanings; 1**k**B equals 1000 bytes and 1**K**B equals 1024 (i.e. 2^{10}) bytes. The line of code in the editor is supposed to calculate **2^{10}**, but is missing a few `* 2`.

1. Complete line 1 to calculate **2^{10}**, by replacing `???` with the correct number of `* 2`.
2. Click the button to run the code.

4.2. do a decimal addition

Once you master JavaScript you can make awesome websites that the whole world can use! Who knows, maybe **you** will create the next Google or Facebook?

Google co-founders Sergey Brin and Larry Page have an estimated wealth of US$16.7 billion **each**.

1. In the code, replace each `???` with `16.7` to calculate the Google founders **combined** wealth in billion US$.
2. When you are done, click to run your code.

4.3. use + to join text

A sequence of characters surrounded by quotes (' or ") is called a **string**.

When + is used with two **numbers**, the numbers are added together. So `16.7 + 16.7` equals `33.4`.

When + is used between two **strings**, the strings are joined into a single string

So `'Code' + 'Avengers'` equals `'CodeAvengers'`.

1. What happens when + is used between a **string** and a **number**? Find out by putting quotes around the 1st `16.7` to turn it into a string.
2. Then click .

4.4. display the result of a calculation with an alert

If the co-founders gave 1/8th of their combined wealth to charity, how many US$ would the donation be?

Modify the code to calculate the answer.

1. Remove the quotes from the previous task.
2. Divide the combined wealth by 8.
3. Multiply by 1000000000 to convert from billion $ to $.
4. Click to check your answer.

4.5. add text to explain the calculation

Displaying a number on its own has little meaning. The number needs text to explain what it is. The example shows how to use + to join a **string** and calculation together.

Notice the space before the 2nd `'` in the example. This makes the message have a space between the text and number.

1. Put the text `'Google donation $'` immediately after `alert(`. Don't forget the `' '`.
2. Join the text to the calculation using +.

5. Store data with '''variables'''

5.1. display calculation results with an alert

WORM ALERT!

An unidentified **worm** has infected 160,000 computers and this is increasing by 25% every 20 minutes.

The code on lines 2-6 calculates and displays the total number of infected computers after 20, 40 and 60 minutes. The `)` on line 6 marks the end of the alert command.

1. the example in the bottom right to see what it does; **click here** to find out why the result in the example has lots of **0s**.
2. Replace the `???` on line 5 to calculate the infections after 60 minutes.
3. Click to run your code.

+ joins **strings** and calculations into a single string. \ is used to display text on separate lines.

The message in an `alert` must be surrounded by `()`.

5.2. update the calculation

The spread of the **worm** is worse than originally predicted! The new estimate is 240,000 infections, increasing by 50% every 20 minutes.

1. Replace `160000` with the new estimate of `240000`
2. Replace `1.25` with the new estimate of `1.5`
3. Click to run the code.

In JavaScript calculations, numbers can **not** include commas `,`. For example, **5,000** is written as `5000`.

5.3. create a variable

It is easy to make mistakes when changing the same number several times. To avoid this problem, we store repeated numbers in something called a **variable**.

The **keyword** `var` is used at the start of a line that creates a variable.

1. Put the code `var initial = 240000` on line 1; this creates a **variable** that stores the initial infection count.
2. On line 2, store the number `1.5` in a variable called `rate`.
3. In the alert message, replace each `240000` with `initial` and each `1.5` with `rate`.

5.4. update the variables

The estimates have been adjusted again to 280,000 infections increasing by 40% every 20 minutes.

1. Update the `initial` variable.
2. Update the `rate` variable.
3. Click to run the code.

Using variables make modifying code much easier!

5.5. set variable values using other variables

Variable names can use letters, digits (0-9), underscore (_) and $. The 1st character in a variable name can't be a digit. Also variable names can't contain spaces.

1. On line 3, create a variable called `_20min` and set it equal to `initial * rate`
2. On each of lines 6-9, replace `initial * rate` with `_20min`

6. Read error messages, fix mistakes

6.1. fix bugs in an alert command

It is easy to put mistakes in your code that make it fail. These mistakes are called **bugs**. In this lesson you will fix **bugs**.

1. the code and read the error message that appears in the output console.
2. Add 2 characters that are missing from the code (without changing the alert message).
3. Re-click to complete the task.

6.2. fix bugs in alert commands

Beware!

The error messages the **web browser** displays are confusing for beginners. But don't worry, you will understand them over time.

1. the code and study the error message.
2. Then fix the code and again to complete the task.

6.3. fix bugs in variable names

Variables have a name and a value In the code `var myAge = 27` the variable name is `myAge` and the value is **27**.

Variable names **can't** include **spaces**. Here is how to do variables with more than 1 word: `myAge`, `weeklyPay`, `noOfWorkers`. Notice the 1st letter of the 2nd and 3rd words is capitalized.

This code creates 2 variables and displays the variable values in an **alert** message. However, there are mistakes on each line.

1. the code and read the error message; in some browsers the error message says the line number of the 1st error in the code.
2. Fix the mistake on line 1.
3. Fix the mistake on line 2.
4. Fix the mistake on line 4.
5. Click .

6.4. fix bugs in variable names

JavaScript variable names are **case sensitive**. This means `myName`, `MyName` and `myname` are 3 different variables because different letters are uppercase. It is important to type variables carefully with the same case each time.

1. Click .
2. Fix the mistakes on line 7.
3. Click again.

6.5. fix variable names and ( )

WARNING: This task is difficult!

Agents and managers are paid $78,000 per year, and cleaners receive $500 per week. This code calculates and displays the average weekly pay of all the workers at S.C.R.I.P.T.

However, there are incorrectly spelled variables. Variable names can **only** use letters, numbers, _ (underscore) and $. The 1st character **can't** be a number.

The code is also missing ( ). Make sure there are the same number of (as) on each line.

1. Click .
2. Fix mistakes on line 5.
3. Fix mistakes on line 7.

7. Move a robot to its goal

7.1. move the robot forward

Move the robot forward 1 square using the command: `robot.forward()`

Move forward 3 squares with a single command using: `robot.forward(3)`

1. Write code to move the robot to the square marked **X**.
2. Click to run your code.

7.2. turn the robot right

Turn right 90 degrees using: `robot.right()`

Put each command on a separate line and don't forget the `()`.

1. Write code to move the robot to the **X**. It is facing **up** at the start.
2. Click to run your code.

When you click the commands are run 1-by-1 by your **web browser**.

7.3. turn the robot left

Turn left 90 degrees using: `robot.left()`

1. Write code to move the robot to the **X**.
2. Click to run your code.

7.4. turn the robot 180 degrees

Turn right 90 degrees using: `robot.right()`

Turn right 180 degrees by repeating `robot.right()` twice, or using: `robot.right(2)`

Notice that it faces **left** to begin with.

1. Write code to move the robot to the **X**.
2. Click to run your code.

7.5. move the robot through a path

1. Write code to move the robot to the **X**.
2. Click to run your code.

8. Ask questions with [[prompt]]

8.1. create a prompt

A `prompt` is used to ask a user questions. The code on line 2 asks the user's name.

1. On line 3 add a 2nd `prompt` that asks: `What is your age?`
2. Click to run the code.

8.2. add default answers to prompts

In addition to specifying the question displayed in the `prompt` box you can also specify a default answer.

A 3rd `prompt` on l#0 asks the user's gender. This prompt has a default answer of **Male**.

Notice that the default answer on l#0 is put after the question, and is separated by a comma.

1. Add a default value of **Anonymous** to the 1st prompt.
2. Add a default age of **21** to the 2nd prompt.
3. Then click to run the code.

8.3. store user responses with **variables**

The example code asks the user's name and stores the answer in a variable called name.

1. Store the result of the 1st prompt by putting `var name =` at the start of the line (as shown in the example code).
2. Store the result of the 2nd prompt in variable `age`.
3. Store the result of the 3rd prompt in variable `gender`.
4. Then the code and click **OK** for the name and gender prompts without changing the default values; type **24** in the prompt box asking your age.

8.4. use the variable values

The example code stores the user's name, then uses it to say hello.

For this task do the following:

1. Display an `alert` on l#1 that uses the `name` variable to **thank** the user for their time. E.g. if the user's name is **Mike**, the alert message should say: "Thanks for your time Mike"
2. Click to run the code; in the **age** message box type **36**; for the **name** type **Ada** (the 1st **female** computer scientist).

Many say **Ada Lovelace** was the 1st computer programmer. She died in 1852 aged 36.

8.5. give some feedback

1. On the line before the `alert`, add a 4th `prompt` that asks the user for feedback.
2. Save their response in a **variable** named `feedback`.
3. Click and fill in the boxes with **your** name, age and gender and at least **5 words** of feedback.

We really value your suggestions!

9. Joining strings and numbers with {{+}}

9.1. use + to join strings

Text that appears in quotes is called a **string**.

When + is used with strings they are joined together to form a new string.

For example, `'a' + 'b'` equals `'ab'`.

Study the code editor and work out the values of the variables after it runs. Enter the answers below then click .

The first answer is done for you.

1. a = Code Avengers
2. b =
3. c =
4. d =
5. e =

9.2. use + with numbers and strings

When + is used with 2 numbers they are added mathematically. For example `20 + 11` is `31`.

If + is used with a number and a **string** they are joined together like 2 strings. For example, `'20' + 11`, `20 + '11'` and `'20' + '11'` all equal `'2011'`.

When there is a mix of strings and numbers with several +, they are calculated from left to right using the rules above.

The example on line 1 is calculated as follows:

```
var a = 'Dec ' + 20 + 11;
```

First calculate `'Dec ' + 20` which equals `'Dec 20'`, then calculate `'Dec 20' + 11`, which equals `'Dec 2011'`.

Study the code editor and work out the values of the variables after it runs. Enter the answers below then click .

1. a = Dec 2011
2. b =
3. c =
4. d =
5. e =

9.3. use * and / with strings

If you put * between 2 **strings** or a string and a number, the strings are converted to numbers and multiplied together, e.g. `'4' * '8'` and `4 * '8'` are `32`.

The same rule applies to / (division) and - (subtraction).

Click **here to review** the math rules for **order of operations**.

Study the code editor and work out the values of the variables after it runs. Enter the answers below then click .

1. a = 32
2. b =
3. c =
4. d =
5. e =

9.4. practice using + to join strings

Avoid errors by surrounding calculations with ( ) when combining strings and calculations.

Study the code editor and work out the values of the variables after it runs. Enter the answers below then click .

1. a = 1KB = 1024B
2. b =
3. c =
4. d =
5. e =

9.5. practice using + to join strings

Remember that * is calculated before +.

Study the code editor and work out the values of the variables after it runs. Enter the answers below then click .

1. a = 5678
2. b =
3. c =
4. d =
5. e =

10. Review variable name rules

10.1. review variable creation

In JavaScript code on the web you will see a `;` at the end of most lines of code. In programming languages such as Java and C, a `;` **must** be put after each statement.

However, in JavaScript `;` is mostly optional. One situation where a `;` **must** be used in JavaScript is to separate two statements on a single line.

E.g. `alert('Hello'); alert('World')`

Click the **start quiz** button to start the review quiz.

Review Quiz Questions:

1. Which of the following JavaScript keywords is used to create a variable?
2. What character is used to separate two statements on the same line?
3. Which of the following correctly creates a variable `x` with an initial value of `5`?
4. Which of the following is an **invalid** way to create a string variable `e` with a value `'z'`?

10.2. review variable name rules

Variable names consist of letters, numbers and the symbols $ and _. They can't contain spaces.

Click the **start quiz** button to start the review quiz.

Review Quiz Questions:

1. Which of the following symbols **CAN** be used in a variable name?
2. Which of the following symbols **CAN** be used in a variable name?
3. {invalidVariableNameQuestion}
4. {invalidVariableNameQuestion}

10.3. review variable name rules

A variable name can't begin with a number; it **must** begin with a letter or 1 of the following 2 characters: _ $

Click the **start quiz** button to start the review quiz.

Review Quiz Questions:

1. {invalidVariableNameQuestion}
2. {invalidVariableNameQuestion}
3. {invalidVariableNameQuestion}
4. {invalidVariableNameQuestion}

10.4. review math calculations

Remember * means multiply and / means divide. * and / are computed before + and -.

Click the **start quiz** button to start the review quiz.

Review Quiz Questions:

1. {evaluateXQuestion}
   ```
   var x = 4 + 2 - 1 + 3
   ```
2. {evaluateXQuestion}
   ```
   var x = 4 + 2 * (1 + 3)
   ```
3. {evaluateXQuestion}
   ```
   var x = (4 + 2) / 1 * 3
   ```
4. {evaluateXQuestion}
   ```
   var x = 4 / 2 + 1 * 3
   ```

10.5. changing variable values

Variable values can change, as in the following example:

```
var x = 5
var y = 10
x = y - 2
y = 6
```

Lines 1 and 2 set the initial values of **x** and **y**.

Line 3 gives **x** a new value of **8**, and line 4 changes **y** to **6**.

Click the **start quiz** button to start the review quiz.

Review Quiz Questions:

1. {evaluateXQuestion}
   ```
   var y = 5; var x = y + 10
   ```
2. {evaluateXQuestion}
   ```
   var z = 5; var x = z * 2
   ```
3. {evaluateXQuestion}
   ```
   var y = 20; var z = 15; var x = y - z
   ```
4. {evaluateXQuestion}
   ```
   var x = 5; var y = x + 10; x = 5 + y
   ```

11. Intro to lessons 12-20

11.1. review \n

Great Work! You have completed 10/40 lessons in this Level 1 JavaScript course. This lesson will review lessons 1-10 as well as preview the next 10 lessons.

In lessons 1-10 you used the `alert` command to display a message. Remember to put `' '` or `" "` around the text to display.

Can you see the `\n` in the middle of the message on line 1 in the code editor? Do you remember what it does?

Click to run the code, then move onto the next review task.

11.2. review **prompt** commands

In lessons 1-10 you also used the `prompt` command to ask the user a question. The `var` keyword is used to create a variable to store the user's response.

You also learned about 2 types of values **strings** (sequence of 1 or more characters) and **numbers**. Recall that + is used to **add** numbers, or to **join** strings together.

Click to run the code, then move onto the next review task.

code explanation

The code on line 1 does 2 things. Firstly, it shows a `prompt` that asks the user's name, and waits for the user to type a response and click **OK**. Secondly, when the user clicks **OK**, the response is stored in a **variable** called `name`.

11.3. introduction to **if statements**

In lessons 12-20 you will learn about **if statements**. You will ask the user a question and use an **if statement** to show a different message depending on the user's answer.

1. Click and enter the correct answer to the quiz questions.
2. Click and enter an incorrect answer.

code explanation

The code on lines 3-6 is an **if statement**. Line 3 checks if the `guess` variable contains the correct answer to the question on line 1. **If** the answer is correct line 4 will run. The word `else` on line 5 is a special keyword, that preceeds the line of code (line 6) that runs if the condition on line 3 is false.

Click run, answer the question by typing **spam**, then click **OK**

11.4. detecting cancel

In lessons 12-20 you will also learn how to detect if the user clicks the **Cancel** button when you ask a question with a `prompt`.

1. Click then click **cancel** in the **prompt** box.
2. Click then click **OK**.

== versus =

Single = on line 1 and double == on line 3 have different meanings.

On line 1, a single = gets the value the user types in the `prompt` box, and stores it in the `name` variable.

Double == checks if two things are the same. On line 3, == checks if the `name` variable contains a special value called `null` (which you will learn about in lesson 17).

Click run then click **cancel**. Click run again then click **OK**

11.5. introduction to **confirm**

Finally, you will learn to use the `confirm` command to ask the user **yes/no** questions.

1. Click then click **cancel** in the **confirm** box.
2. Click then click **OK**.

Code Layout

The layout of your code is very important, especially as you begin to write large programs. You are less likely to make mistakes in your code if you lay it out nicely.

Notice that lines 4 & 6 are indented using the **Tab** key. When you write **if statements** try to use the same layout as the example code.

Click run then click **cancel**. Click run again then click **OK**.

12. Check user responses with [[if statements]]

12.1. run an **if statement**

The code on line 1 asks the user to enter a password and stores their response in a variable called `guess`.

Lines 3-4 are an **if statement**. When this code runs, the **web browser** checks if the **guess** variable equals **Firewall** and runs the alert command on line 4 if they are equal. The **alert** is not shown if the user types anything except **Firewall**. == and = are **not** the same. Double equals == is used in **if statements** (e.g. line 3) to check if 2 things are equal. Single equals = is used to store a value in a variable (e.g. line 1).

1. Click .
2. Type the S.C.R.I.P.T. secret password.

Click run, type **Firewall**, then click **OK**

12.2. add another if statement

Last month, the old S.C.R.I.P.T. password "SpamSniper" was compromised, so the password was changed to "Firewall".

1. Add a 2nd **if statement** (similar to the 1st) that checks if the user types the old password.
2. If they do, display the message: "Intruder alert!"
3. Click and type the **old password** to test the code.

code explanation

On line 1, = is used to store the text typed in the `prompt` in a variable called `guess`. A single = **stores a value** in a variable.

On line 3, `if (guess == 'Firewall')` uses == to check if the `guess` variable is equal to "Firewall". == is used to **check** if two things **are equal**.

12.3. change to if–else if–else

The example code asks the user to enter a test score and displays a grade of **A**, **B**, **C** or **F**. Read the info below for details on how the example code works.

Do the following, so that if the user guesses neither the old or new password, display the message: "Incorrect!"

1. Replace the 2nd `if` with `else if`.
2. Add the following on new lines to the end of your code:
```
else
  alert('Incorrect!')
```
3. your code 3 times; enter values that show a different alert each time.

example code explanation

The **if statement** in the example code has `if`, `else if` and `else` branches. When the code runs, the computer checks if `score >= 80`. If its **true** the message **A** is displayed and the `else if` and `else` branches are skipped.

If its **false** the computer checks the `else if` branch to see if `score >= 65`. If the `else if` is **true** the message **B** is displayed and the 2nd `else if` and `else` branches are skipped.

If both the `if` and `else if` conditions are **false**, the code immediately after the `else` is run.

In this case the message **F** is shown when `score < 50`.

12.4. **if statements** with multiple statements

To show 2 alerts when an `if` condition is true, you must surround the alert commands with { } as shown on lines 3-6 of the example code. The { and } keys are near the "Enter" key.

1. If the password is correct show the alert that says `Correct!`, then show another **alert** that says: "Welcome to S.C.R.I.P.T."
2. If the user types the old password, just show 1 message: "Intruder alert!"
3. If the password is incorrect show the **alert** that says "Incorrect!", then show another **alert** that says: "Please try again"
4. your code 3 times and enter values that show different messages each time.

12.5. test all **if statement** branches

To test code with an **if statement** you should more than once to ensure that each branch works correctly.

Visitors to S.C.R.I.P.T. enter "Guest" as the password.

1. Add an `else if` branch that checks for the guest password.
2. For guests, display 2 alert messages: "Welcome to S.C.R.I.P.T." and "Please wait for your host"
3. Then your code 4 times and enter values that show a different message each time.

13. true & false

13.1. greater > or less than <

A **boolean** (pronounced boo-lee-in) is a special type of value that is either **true** or **false**. The code `var x = false` creates a variable called x and stores the value **false** in it.

In JavaScript, < means **less than**; > means **greater than**.

<= means **less than or equal to**; >= means **greater than or equal** to.

== means **equal to**; != means **not equal to**.

The code `var x = 25 > 12 * 2` sets x to **true** since `25 > 12 * 2` is **true**.

Study the code editor and work out the values of the variables after it runs. Enter the answers below then click .

The first answer is done for you.

1. a = true
2. b =
3. c =
4. d =
5. e =
6. f =

13.2. using AND &&, OR ||

In JavaScript, && means **AND**. So x `&&` y is true if **both** x **and** y are true. || means **or**. So x `||` y is true if **either** x **or** y or **both** are true.

The code `var x = 8 < 9 && 2 >= 2` sets x to **true** since `8 < 9 && 2 >= 2` is **true**.

Note that > and < are evaluated before && and ||.

Study the code editor and work out the values of the variables after it runs. Enter the answers below then click .

1. a = true
2. b =
3. c =
4. d =
5. e =
6. f =

13.3. using **NOT** !

In JavaScript, `!` means not. So `!true` is false, and `!false` is true.

Remember that && is AND; || is OR.

`var z = x || y` sets z to true if either x **or** y is true.

`var z = x && y` sets z to true if both x **and** y are true.

The code `var z = true || false` sets z to true since `true || false` is true.

Study the code editor and work out the values of the variables after it runs. Enter the answers below then click .

1. a = false
2. b =
3. c =
4. d =
5. e =
6. f =

13.4. is equal to ==

== means **is equal to**. x `==` y is true if x and y have the same value.

E.g. the code `var x = true == false` sets x to **false** since **true** is not equal to **false**

Note that `!` is computed before || and &&. Click the following for definitions: &&, ||, !=, ==.

Study the code editor and work out the values of the variables after it runs. Enter the answers below then click .

1. a = false
2. b =
3. c =
4. d =
5. e =
6. f =

13.5. using < and > with strings

When < and > are used with text it compares the positions of the letters in the alphabet `'a' < 'b'` is **true** because **a** comes before **b** in the alphabet `'dog' > 'cat'` is also **true** because **d** is after **c** in the alphabet

Compute < and > before && and ||.

Study the code editor and work out the values of the variables after it runs. Enter the answers below then click .

1. a = false
2. b =
3. c =
4. d =
5. e =
6. f =

14. The [[confirm]] command

14.1. **confirm** box

Have you ever tried to leave a website and had a box appear asking if you **really** want to leave?

This is done using the `confirm` **function** (**function** is another word for what we called a **command** in earlier lessons).

1. Show a `confirm` box that says: "Are you sure you want to leave?"
2. Click and look for the difference between a `confirm` and `alert` box.

14.2. **confirm** box result

Now let's use a variable to check which button the user clicks.

1. Put the code `var clickedOk =` at the start of line 1; `clickedOk` will be set to **true** or **false** depending on which button the user clicks.
2. Put the code `alert(clickedOk)` on line 2; this displays the value of the `clickedOk` variable in an alert box.
3. the code twice and try clicking both **cancel** and **OK**.

Pay attention to the value shown in the alert when you click each button.

14.3. check the result with an **if statement**

1. Replace the **?** on line 3 to check if the user clicks **OK**.
2. Replace the **?** on line 4 to say **Goodbye** if the user clicks **OK**.
3. If they click **cancel** don't show the message.
4. Then the code twice, selecting both **cancel** and **OK**.

14.4. testing a **confirm** box

Modify the code to do the following:

1. **Don't** show an alert if the user clicks **OK**.
2. If they click **cancel** show an alert that says: "Thanks for staying"
3. the code twice, selecting both **cancel** and **OK**.

14.5. add another **confirm**

Modify the code by replacing the `?`s

1. If they click **OK** for the 1st confirm then show a 2nd confirm that asks: "Do you really want to leave?"
2. If they click **OK** for the 2nd confirm then display a message that says: "Goodbye"
3. If they click **Cancel,** for the 1st or 2nd confirm then show a message that says: "Thanks for staying"
4. your code 3 times and test the 3 combination of button clicks.

15. [[if statement]] practice

15.1. == in **if statement** conditions

In this lesson, you'll test **if statement** code from the **S.C.R.I.P.T.** agent training online application.

The application begins by asking the user's gender. Training courses start on different dates for males and females.

For this task, run the code 4 times in order to make sure you test every branch in the **if statement**.

1. Click and enter input to run the 1st branch of the **if statement**.
2. Click and enter input to run the 1st else if branch.
3. Click and enter input to run the 2nd else if branch.
4. Click and enter input to run the else branch.

Click run, type nothing in the prompt, then click **OK**. Click run again, type **Male**, then click **OK**. Click run again, type **Female**, then click **OK**. Click run once more, type **boy**, then click **OK**.

15.2. numbers in **if statement** conditions

The agent training course has a weight restriction, so the next question asks the user's weight.

Note that if you type **75kg** the alert says **Invalid input** because **75kg** is not a valid JavaScript number. Type **75** (without **kg**) and `weight < 130` will be true.

1. Click and run the 1st branch of the **if statement**.
2. Click and run the 1st else if branch.
3. Click and run the 2nd else if branch.
4. Click and run the else branch.

Click run, type nothing in the prompt, then click **OK**. Click run again, type **75**, then click **OK**. Click run again, type **135**, then click **OK**. Click run once more, type **75kg**, then click **OK**

15.3. testing **if statement** conditions

The agent training course has a height restriction.

1. Click and run the 1st branch of the **if statement**.
2. Click and run the 1st else if branch.
3. Click and run the 2nd else if branch.
4. Click and run the 3rd else if branch.
5. Click and run the else branch.

Note that if you type **1.5m** the alert says "Invalid input" because **1.5m** is not a valid JavaScript number. You must type a **1.5** without the **m** on the end.

Click run, type nothing in the prompt, then click **OK**. Click run again, type **1.5**, then click **OK**. Click run again, type **2.2**, then click **OK**. Click run again, type **1.85**, then click **OK**. Click run again, type **1.85m**, then click **OK**

15.4. complex **if statement** conditions

An agent's endurance is tested with 3km run. The time limits are different for males and females.

1. Click and run the 1st branch of the **if statement**.
2. Click and run the 1st else if branch.
3. Click and run the 2nd else if branch.
4. Click and run the 3rd else if branch.
5. Click and run the 4th else if branch.
6. Click and run the else branch.

If you type **10m45s** or **10:45** the alert will say "Invalid input", because they are not valid JavaScript numbers. Type **10.75** instead.

Run the code 6 times and do the following:

1. Put nothing in both prompts
2. Type **Male** for the gender prompt and **7** for the time prompt
3. Type **Female** then **16**
4. Type **Male** then **15**
5. Type **Female** then type **19**
6. Type **boy** then **20mins**

15.5. complex **if statement** conditions

Agents must pass a strength test. Male recruits must reach a minimum score in either push-ups and sit-ups, or a minimum combined score.

1. Click and run the 1st branch of the **if statement**.
2. Click and run the 1st else if branch.
3. Click and run the 2nd else if branch.
4. Click and run the else branch.

Click run, type **61** and **0**, then click **OK**. Click run again, type **0** and **121**, then click **OK**. Click run again, type **50** and **100**, then click **OK**. Click run once more, type **10** and **10**, then click **OK**

16. Move 2 robots to their goal

16.1. move the robots to the **X**

In lesson 6 you wrote code to move 1 robot to the **X**. Now you will write 1 set of code that moves 2 robots to 2 different goals.

1. Replace `?` with the smallest number that moves both robots to the **X** (if there is a wall in front of a robot moving forward the robot stays where it is).
2. For each task in this lesson click to run your code for each robot.

Remember: the command `robot.forward(2)` moves the robot forward 2 squares.

16.2. move the robots to the **X**

Write 1 set of code that moves each robot to its goal.

1. Replace the **?** on line 1.
2. Replace the **?** on line 2.
3. Replace the **?** on line 3.

`robot.right()` and `robot.left()` turn the robot 90 degrees. Always put `()` after every command.

16.3. move the robots to the **X**

The robot has a sensor that detects when it reaches the **X**. The function `robot.onX()` is true if the robot is on the **X**. Since `!` means **not**, the code on l#-4, `if (!robot.onX())`, checks if the robot is **not** on the **X**.

Edit the code to get the robots to the **X**, then click .

16.4. move the robots to the **X**

Moving the robot forward while facing a wall damages the robot. Edit the code, replacing `?` with commands and numbers that will move the robots to the **X** without bumping into a wall.

The robot has a sensor on the front that detects walls. The function `robot.atWall()` on line 4 checks if there is a wall directly in front of the robot.

Click to run your code.

Remember to use `()` after each command.

16.5. move the robots to the **X**

Now for a challenge! Move the robots to the **X** in as few steps as possible without bumping into walls. The code `if (robot.atWall())` checks if the robot is facing a wall.

Click to run your code.

Use code in the previous tasks as a guide and remember to put `()` after each command.

17. The special value '''null'''

17.1. **prompt** box cancel button

1. Create a `prompt` that asks the user what their native language is; use a default value of **English**.
2. Store the response in a **variable** called: `nativeLanguage`.
3. On the next line, display the value of `nativeLanguage` in an `alert` box.
4. Click then click **cancel** in the prompt and **read** the word in the alert message.

If you can't remember how to use a `prompt` review lesson 8.3.

17.2. `null` value

When you the code and click cancel in the prompt box, the `nativeLanguage` variable is set to `null`: a special value that means **nothing** or **no value**.

The code `if (nativeLanguage == null)` checks if the user clicked cancel. No quotes are needed around `null`.

1. Replace `???` with a prompt to re-ask the user's native language and store the response in the `nativeLanguage` variable.
2. your code twice to check that the 2nd prompt is only shown if the user cancels the 1st prompt.

`var` is only used to create a variable and store a value in it for the first time. `var` is not needed on line 4, because the `nativeLanguage` variable was already created on line 1.

17.3. check for cancel

1. At the end of the code, add an `alert` that thanks users for their time, as long as they don't click **cancel** in both **prompt** boxes.
2. the code twice to check that the alert does **not** appear when the user clicks cancel twice.

Remember: `!` means **not**, so `!=` means **not equal to**.

17.4. if statement practice

Replace the **?s** to display different messages depending on the user's native language.

1. If the language is **Spanish** thank them in Spanish by saying: "Gracias por tu tiempo"
2. If the language is **French** say: "Merci pour votre temps"
3. For any other language, show the English message: "Thanks for your time"
4. If the user clicks **cancel** no message is shown.
5. your code 4 times entering **English**, **Spanish**, **French** and clicking cancel.

17.5. using OR || in **if statement** conditions

The Spanish word for Spanish language is **español**. The French word for French language is **français** (in Spanish and French the names of languages are written in lowercase).

1. To thank the user in Spanish if they type either word for the Spanish language, change line 3 to:
   `if (nativeLanguage == 'Spanish' || nativeLanguage == 'español')`
2. Use the same pattern to modify line 5 to thank the user in French if they type **French** or **français**
3. Click to run the code and enter **your** real native language.

Your answer will be recorded and used to improve our site.

18. Read error messages, fix mistakes

18.1. review **if statement** mistakes

This lesson reviews common **if statements** mistakes. The code in this task is similar to lesson 12, except that it has a couple of mistakes.

1. Click and read the error message that appears in the output console.
2. Fix the mistakes.
3. Click twice to check both branches of the **if statement**.

18.2. review **if statement** mistakes

Beware, sometimes the actual error is a few lines **before** the line number in the error message. The reason for this is a little complicated.

1. Click and read the error message.
2. Fix the mistakes.
3. Click 3 times to check each branch of the **if statement**.

18.3. review **if statement** mistakes

1. Click and read the error message.
2. Fix the mistakes.
3. Click 4 times to check each branch of the **if statement**.

18.4. review **if statement** mistakes

In this task, there are a few missing characters.

1. Click and read the error message.
2. Add the missing characters on line 6.
3. Fix the other bug.
4. Click 4 times to check each branch of the **if statement**.

Remember: sometimes the mistake is a few lines **before** the line number in the error message.

18.5. review **if statement** mistakes

1. Click and read the error message.
2. Fix the mistakes.
3. Click 4 times to check each branch of the **if statement**.

19. Combining conditions with {{&&}} and {{||}}

19.1. re-ask a question if the user clicks cancel

1. Ask the user their age and store the response in a **variable** called age.
2. Check if the user clicks cancel.
3. If they do, re-ask their age and store the result in the age variable.
4. your code twice to check that the 2nd prompt is shown only if the user clicks cancel.
5. When your code works correctly, click to see if your code passes the task.

View the **prompt** example in the **reference** to see what happens when the user clicks **cancel**.

19.2. check the user types a valid age

1. On l#-4, check if the user types an age greater than 1 and no more than 115 (the age of the oldest living person).
2. On l#-3, show an `alert` that says: "thank you"
3. On l#-1, show an alert that says: "invalid age"
4. Click to run your code, then click to see if your code passes the task.

Click <, <=, >, >=, || and && for definitions.

19.3. check the code and fix the mistake

When you click Code Avengers automatically runs your code a few times to test different inputs. If your code fails to work correctly an error message appears.

The code in the editor is part of the **S.C.R.I.P.T.** job recruitment website. New employees must be **at least 18** years old. However, the code has a mistake. When the user types **18** the program incorrectly says they are too young.

1. Click and read the message in the results console.
2. Fix the mistake.
3. Click again.

19.4. complete if conditions

New employees at **S.C.R.I.P.T.** must be at least 18 and under 80 years of age. New Special Agents must be aged 22-28. Other ages can only apply for office and cleaning jobs.

1. Replace the `?` on line 7.
2. Replace the `?` on line 9.
3. Replace the `?` on line 12 to tell them to apply for an **office** or **cleaning** job.
4. Click to test your code to see if it passes.

Click <, <=, >, >=, || and && for definitions.

19.5. improve code layout

It is important to lay out code nicely, so it is easy to understand. The top code is a poorly laid out version of the bottom code.

1. Make the layout of the top code **exactly** the same as the bottom code. Use the **tab** key to move a line right 2 spaces.
2. Click and enter your **real** age (or cancel if you don't want to tell us).

We will record your answer to help us better understand your needs.

20. Review terminology

20.1. review **if statements**

The final task in each lesson is a review quiz.

Click the **start quiz** button to start the review quiz.

Review Quiz Questions:

1. In an if statement, which character comes immediately after the word `if`?
2. What keyword is used at the last branch of an if statement, to cover all other cases?
3. What pair of characters is used in an if statement to mean **less than or equal to**?
4. What pair of characters are put around several statements that are run when an if condition is true?

20.2. match the definitions with the terms

The final task in each lesson is a review quiz.

Click the **start quiz** button to start the review quiz.

Review Quiz Questions:

1. The data type with a value of true and false
2. The data type for a sequence of characters
3. A command that performs a specific task and may return a value.
4. Stores a value that can be used later and modified by the program.

20.3. review variable name rules

In past lessons you have used several **keywords** with special purposes in the JavaScript language. These can't be used for variable names.

Click the **start quiz** button to start the review quiz.

Review Quiz Questions:

1. {invalidNameQuestion}
2. {invalidNameQuestion}
3. {invalidNameQuestion}
4. {invalidNameQuestion}

20.4. review **alert**, **prompt**, and **confirm** boxes

An **alert** box displays a message to a user.

A **prompt** asks the user to type text in response to a question.

A **confirm** asks a question and displays 2 buttons, **OK** and **cancel**.

Click the **start quiz** button to start the review quiz.

Review Quiz Questions:

1. What special value means **no value**?
2. What value is returned when the user clicks cancel in a `prompt` box?
3. What value is returned when the user clicks cancel in a `confirm` box?
4. How many buttons are shown under the message on an `alert` box?

20.5. review some special symbols

The final task in each lesson is a review quiz.

Click the **start quiz** button to start the review quiz.

Review Quiz Questions:

1. What pair of characters is used in an if statement to mean **OR**
2. What is used in an if statement to check if two things are **equal**
3. What pair of characters is used in an if statement to mean **AND**
4. Which symbol means **NOT**

21. Intro to lessons 22-30

21.1. review if statements

Well done! You are half way through this level 1 course.

Lessons 11-20 focused on **if statements**. An if statement has 1 or more **branches**. First is the `if` branch, followed by 0 or more `else if` branches, and lastly the `else` branch.

The `if` and `else if` branches have a condition that must be true for the code in that branch to run. This condition **must** be surrounded by `( )`.

the code and enter your gender; then move onto the next review task.

Click run then type **Male** or **Female**. Make sure the 1st letter is a capital.

21.2. review logic

In lessons 11-20 you also learned how to ask questions with numerical answers, and to use < and > to check if the number is above or below a specified value.

You also learned how to use || and && to check if a number is within or outside a range of values.

the code and enter your height; then move onto the next review task.

Click run, type **1.8**, then click **OK**

21.3. review **confirm**

Finally, you learned how to use the `confirm` command to ask a question with a **yes** or **no** answer.

You also learned that `{ }` are used with **if statements**, when there are 2 or more lines of code to run in an `if`, `else if` or `else` branch. Remember to use the tab key to lay code out nicely; also remember that for every { there must be a matching }.

Click to run the code.

Click run, then click **OK** three times

21.4. introduction to **while** loop

In lessons 22-30 you will learn how to force a user to answer a question and click OK, using something called a `while` loop.

To complete this task:

1. Click to run your code.
2. Click cancel at least twice.
3. Type your name and click OK.

You may notice that lines 1, 4 & 7 end with a `;` (semi-colon). Semi-colons are optional except in a few special cases; in those cases forgetting them causes hard to find bugs; using `;` is recommended ...

The next few lessons explain which lines should end with `;`

Click run, then click **Cancel** twice, then type your name and click **OK**

21.5. introduction to **while loop**

You will also learn to use a `while` loop to ask the user to enter several numbers and to calculate the total. The loop in the example code, asks the user to enter 4 numbers. The alert on line 11 displays the total and average of the numbers.

The command `console.log` on line 8 is used to show messages in the results box, which programmers call the **console**.

the code and enter 4 numbers that add to 100.

Click run, type **25** and click **OK**, then repeat three more times.

22. Repeat code with [[while]] loops

22.1. ask questions and display responses

This lesson introduces loops in task 3.

But first, complete the following:

1. Ask the user why they want to learn JavaScript, and store the result in a variable called `response`.
2. Display a message thanking the user for their time.
3. For each task in this lesson click to test your code, then to see if your code passes.

22.2. re-ask questions if the user clicks cancel

1. Use an if statement to check if the user clicks cancel.
2. If they do, re-ask the question and store the reply in the **response** variable.
3. Display the thanks message after the **if statement**.

22.3. use a **while** loop to force the user to answer a question

The example shows that a **while** loop looks similar to an **if statement**. The code inside the `{ }` runs repeatedly as long as the while condition in the `( )` is true. In the example, the code keeps asking the user to enter the password until they get it correct.

Make the following changes to the code in the editor:

1. Replace `if` with `while` so the 2nd `prompt` repeats until the user types a response.
2. Display the thanks message after the loop.
3. your code and click cancel a few times to check it works.
4. Then click to confirm that your code passes.

22.4. re-ask a question if the user clicks OK without typing anything

1. Modify the code so that the question is re-asked if the user clicks cancel **OR** clicks OK without typing any characters.
2. Show the thanks message after the while loop ends.

Remember || means **OR**.

22.5. test on valid and invalid inputs

It is important to test your programs on valid and invalid inputs. In this case, clicking **cancel** and typing an empty response are invalid inputs.

1. Click to run your code.
2. Click **cancel**.
3. Click **OK** without typing anything.
4. Type your **real answer** to the question and click **OK**.

23. Loop a set number of times

23.1. loop a set number of times

In the previous lesson you used a **while loop** to re-ask a question until the user types a valid response. A loop is also used to repeatedly run lines of code a set number of times.

Line 5 increases the value of the **number** variable by 1. **Click here for more details**.

The code in the editor shows the numbers 1 to 5 in the console.

1. Modify the **loop** to show 10 console messages that count from 1 to 10.
2. For each task in this lesson click to test the code, then to see if your code passes.

The example code prints the numbers 1 to 10 **without** using a loop.

23.2. loop a set number of times

The code on line 5, `number = number + 1`, adds 1 to the number variable.

1. Modify line 5 to show **odd numbers**.
2. Modify line 3 to show the numbers 1 to 11.

23.3. loop a set number of times

Now modify the code to show 7 console messages that show the numbers: `5 10 15 20 25 30 35`

1. Modify line 1.
2. Modify line 3.
3. Modify line 5.

The statements inside the while loop `{ }` are **indented** with the **tab** key to make the code easy to read.

23.4. loop a set number of times

Now use 1 `console.log` message to display `5, 10, 15, 20, 25, 30, 35`

Replace `??` on lines 2, 5, 6 & 9 with the following lines of code (which are in scrambled order):

1. `number = number + 5;`
2. `var text = number;`
3. `console.log(text);`
4. `text = text + ', ' + number;`

Only change `??`. The rest of the code is correct

23.5. loop a set number of times

Writing a loop to list multiples of 5 up to 35 takes longer than typing the numbers. But what about a list of multiples of 5 up to 30,000?

The world's fastest typist would take half a day to type all the numbers. Whereas, using a loop, a computer program can print numbers in less than a second. Loops allow you to do repetitive tasks **quickly**!

Modify the code to show all multiples of 3 from 90 up to and including 300, in the same format as in the previous task, i.e. **90, 93, 96, ...**

1. Modify line 1.
2. Modify line 4.
3. Modify line 5.

24. [[While]] loops quiz

24.1. understand **while** loops

`while` loops repeat the code inside the `{ }` repeatedly, until the condition in the `( )` is false.

In this lesson you'll practice stepping through while loop code, to determine how it works.

Study the code editor and work out the values of the variables after it runs. Enter the answers below then click .

1. a =
2. b =

24.2. understand **while** loops

Remember that `x = x - 1` subtracts 1 from the x variable.

Study the code editor and work out the values of the variables after it runs. Enter the answers below then click .

1. a =
2. b =

24.3. understand **while** loops

The following loops look almost identical to those in the previous task. However, they give very different results.

Study the code editor and work out the values of the variables after it runs. Enter the answers below then click .

1. a =
2. b =

24.4. understand **while** loops

Study the code editor and work out the values of the variables after it runs. Enter the answers below then click .

1. a =
2. b =

24.5. understand **while** loops

Now let's see if you can handle negative numbers.

Study the code editor and work out the values of the variables after it runs. Enter the answers below then click .

1. a =
2. b =

25. Fix [[infinite loops]]

25.1. fix an **infinite loop**

Bugs in loop code can result in a loop that goes round and round and never ends. This is known as an **infinite loop**.

The code in the editor is supposed to show 5 console messages that count from 1 to 5. However, a line of code is missing, which causes the number 1 to be printed again and again; this is an infinite loop.

Normally the only way to stop a program with an infinite loop is to close the **web browser**. However, in this lesson we will stop the program for you once the console message is shown 15 times.

1. Click now to confirm the code is an infinite loop.
2. Add a single line of code so that it shows exactly 5 console messages that count from 1 to 5.
3. Click to re-run your code.

25.2. fix an **infinite loop**

Little mistakes can also cause loops to be skipped all together.

The code for this task is supposed to show **console** messages with the first 5 multiples of 3, followed by the first 5 multiples of 4.

A bug in the code causes the 1st loop to be skipped.

1. the code.
2. Fix the mistake on lines 1-6.
3. Add the missing code on lines 7-12.
4. again to complete the task.

When the 2nd loop starts, the value of `n` is what ever it was when the 1st loop ended.

25.3. fix an **infinite loop**

The code for this task is supposed to display 9 messages that show the numbers:
`4, 3, 2, 1, 0, 1, 2, 3, 4`

The 1st while loop should count from 4 down to 0. The 2nd loop should count back up to 4.

However, mistakes in the code have resulted in infinite loops!

1. the code.
2. Fix the mistake on lines 1-6.
3. Fix the mistakes on lines 8-11.
4. again to complete the task.

When the 2nd loop starts, the value of `n` should be 0.

25.4. fix **loop** bugs

This code asks the user their name. It should keep asking the question if the user clicks **cancel** or clicks **OK** without typing anything. However, there is a problem with the code.

1. the code.
2. Fix the mistakes.
3. Check the code is correct by clicking . When the 1st **prompt** appears click **cancel**. For the 2nd **prompt** click **OK** without typing anything. In the 3rd type **your name** and click **OK**.

25.5. fix an **infinite loop**

As in task 2, these loops are supposed to display the first 5 multiples of 3 and 4. Once again there are mistakes resulting in infinite loops!

In this task you must use `!=` instead of `<` or `>` in the `while` conditions.

1. the code.
2. Change the number on line 3.
3. Change the number on line 10.
4. again to complete the task.

26. Simplify code with [[while]] loops

26.1. use a loop to move the robot to the **X**

In this lesson you can only move the robot 1 square at a time using `robot.forward()`. The code in the editor moves the robot to the **X**.

1. Modify the code to use a **while** loop, so that the code contains only 1 **forward** command.
2. For each task in this lesson click to run your code.

26.2. extend the code to move the robot to the **X**

1. Use no more than 3 `while` loops, and no more than 3 `forward` commands.
2. Use `robot.right()` and `robot.left()` to turn the robot 90 degrees.
3. Make the robot get to the **X** in no more than 13 steps.

26.3. move each robot to the **X**

Each robot has a sensor that detects when it has reached the end. The function `robot.onX()` is true if the robot is on the **X**.

Use `robot.onX() == true` to check if the robot is on the **X**.

Use `robot.onX() == false` to check if the robot is **not** on the **X**.

Complete the code to move each robot to the square marked **X**.

1. Replace the `???` on line 1.
2. Replace the `???` on line 2.

26.4. move each robot to the **X**

1. Use a loop to move the robot forward until they reach the wall; use `robot.atWall()` to check if the robot is facing a wall.
2. Move to the **X**.

26.5. complete the code to move each robot to the **X**

Use the robot's **forward** function no more than **3** times. Use the `atWall` **5** times. Use the **right** and **left** functions as many times as you need.

For this task you can't use `robot.right(2)` to turn the robot 180 degrees.

1. Replace `?` on lines 2-3, lines 15-16 and lines 25-26.
2. Replace `?` on lines 7 & 19.
3. Replace `?` on lines 9-12 and lines 20-22.

27. [[While]] loop conditions

27.1. step through **while** loop code

In this lesson you'll practice stepping through while loop code. You may want to use a pen and paper.

Study the code editor and work out the values of the variables after it runs. Enter the answers below then click .

1. a =
2. b =

27.2. step through **while** loop code

The code inside a loop may run 0 or more times.

If the `while` condition is false from the start, the code in the loop runs 0 times.

Study the code editor and work out the values of the variables after it runs. Enter the answers below then click .

1. a =
2. b =

27.3. step through **while** loop code

Study the code editor and work out the values of the variables after it runs. Enter the answers below then click .

1. a =
2. b =

27.4. step through **while** loop code

Study the code editor and work out the values of the variables after it runs. Enter the answers below then click .

1. a =
2. b =

27.5. step through **while** loop code

In this task, after the 1st loop the value of x is **not** reset. It starts with the value it contained after the 1st loop ended.

Study the code editor and work out the values of the variables after it runs. Enter the answers below then click .

1. a =
2. b =

28. Weekly wage calculator

28.1. create a loop that repeats 5 times

In this lesson you will write a program to calculate and display the weekly wage of a **S.C.R.I.P.T.** employee.

As a 1st step, complete the `while` loop to display 5 alerts that show the numbers 1 to 5.

1. Replace the `???` on line 3.
2. Replace the `???` on lines 4-5.
3. For each task in this lesson click to try it out, then click to see if your code passes.

28.2. get the hours worked by the user each day of the week

1. Change the `alert` to a `prompt` that says **Enter hours worked on day 1** in the 1st prompt, then **Enter hours worked on day 2** for the 2nd prompt, etc.
2. Store the result of the `prompt` in a variable called: `dayHours`

28.3. calculate and display the total hours worked in a week

The prompt function returns a **string**; `Number(dayHours)` converts `dayHours` from a string to a number; this is explained in detail in the next task.

For this task modify the code to calculate and display the number of hours worked in a week by the employee.

1. Create a variable called `totalHours` **before** the start of the loop and set its initial value to **0**.
2. Put `totalHours = totalHours + Number(dayHours)` **inside** the loop to add the hours for each day to the total.
3. **After** the loop ends show the total in an **alert**.

28.4. see the problem when the **Number** function is not used

The value typed by the user in the **prompt** on line 5 is stored in the `dayHours` variable as a **string**. Remember that + joins a **string** and number into a single **string**. E.g. `'12' + 34` is `'1234'`

The `Number` function converts a **string** into a **number**. On line 6, `Number(dayHours)` converts `dayHours` to number so that the + **adds** instead of **joins** the numbers together.

1. Remove the **Number** function on line 6.
2. Click to see what happens, then click to complete the task.

If the user types an invalid number the program will not work properly.

28.5. calculate and display the weekly wage

The hourly pay rate is **$22.25**.

1. After the loop ends, calculate and store the total pay in a variable called: `totalPay`
2. Change the **alert** message to show the hours worked followed by $ paid for the week, on 2 separate lines.

29. Create a math quiz

29.1. ask the user a multiplication question

S.C.R.I.P.T. agents must exercise to stay in top physical and mental shape. This includes regular mental arithmetic training.

In this lesson you will create a program for practicing times tables.

1. Display a prompt that says "What is 3 x 8?" and store the response in a variable called: `guess`
2. Display a message that says "Correct" if the answer is right.
3. And says "Incorrect" if the answer is wrong.
4. For each task in this lesson click to try it out, then click to see if your code passes.

29.2. copy and paste code to ask 5 questions

Use **ctrl+c** (copy) and **ctrl+v** (paste) to use 5 **prompt** boxes to ask the following questions:

1. "What is 2 x 8?"
2. "What is 3 x 8?"
3. "What is 4 x 8?"
4. "What is 5 x 8?"
5. "What is 6 x 8?"

The **rule of three** says that if a section of code is repeated **3 or more times**, you should remove the repetition; in the next task you will use a **while** loop to do this.

29.3. use a **while** loop to remove repetition

Using copy and paste to repeat code 3 or more times is **bad** because it makes the code hard to modify.

For example, changing the code to test 9 times tables requires changing at least 5 numbers. The more changes required, the more likely you will make mistakes.

Loops remove repetition and make code easier to modify

Make the following changes to use a loop to practice the **8** times table:

1. Create a variable called `n` that starts counting at **2**.
2. Create a loop that counts from **2 to 6**.
3. Use a single **prompt** command to ask the question.
4. Use 2 alerts to say "correct" and "incorrect".
5. Increase the value of **n** before the loop repeats.

29.4. allow the user to choose which times table to practice

1. Use a `prompt` **before** line 1 to ask which times table the user wants to practice; save the response in a variable called: `toPractice`
2. Modify the code to use the `toPractice` variable (instead of the number 8) when asking the questions and checking the answer.

29.5. count how many the user gets correct

1. Create a variable called: `score`
2. Use `score` to count how many answers the user gets correct.
3. At the end of the program display the score using the following format: "You got 4/5"

30. Review key concepts

30.1. review JavaScript terms

The final task in each lesson is a review quiz.

Click the **start quiz** button to start the review quiz.

Review Quiz Questions:

1. What is the term for a sequence of characters (letters, numbers and symbols)?
2. Which of the following is **NOT** a command to display a message box?
3. Fill-the-blank. If statements can include an else branch, with 1 or more ____ branches in between
4. A loop that will never end is called an _____ loop

30.2. review programming concepts

The final task in each lesson is a review quiz.

Click the **start quiz** button to start the review quiz.

Review Quiz Questions:

1. Which of the following correctly creates a variable called `myVar` and sets an initial value of 5?
2. If the code `name = prompt('Whats your name?')` is used, which code checks the user did NOT click cancel?
3. Which statement correctly uses the newline character to show the numbers 1 and 2 on separate lines?
4. Which of the following is **always** true, no matter what the value of a is?

30.3. review && (and), || (or) and ! (not)

The final task in each lesson is a review quiz.

Click the **start quiz** button to start the review quiz.

Review Quiz Questions:

1. If `a = 3` and `b = -4`, which of the following is **true**?
2. If `a = 2` and `b = 4`, which of the following is **true**?
3. If `a = 5` and `b = 1`, which of the following is **true**?
4. If `a = 9` and `b = 3`, which of the following is **true**?

30.4. review math calculations

Remember to do calculations in the following order:

1. Brackets ( )
2. Multiplication * and division /
3. Addition + and subtraction -

Click the **start quiz** button to start the review quiz.

Review Quiz Questions:

1. `2 * 4 + 16 / 2 - 8`
2. `2 * 2 * 2 * 2 * 2 * 2 * 2 * 2`
3. `16 / (8 - 4) * (64 - 32)`
4. `4 * 4 + '1'`

30.5. review changing variable values

Remember `;` is used to separate two code statements on a single line.

Click the **start quiz** button to start the review quiz.

Review Quiz Questions:

1. What is the value of x when the following code runs?
   ```
   var y = 5; var x = y * y;
   ```
2. What is the value of x when the following code runs?
   ```
   var x = 5; x = x * 2;
   ```
3. What is the value of x when the following code runs?
   ```
   var x = 5; x = x + x; x = x + x;
   ```
4. What is the value of x when the following code runs?
   ```
   var x = 5; x = x + '5';
   ```

31. Intro to lessons 2-10

31.1. review loops

Well done! Ten lessons to go to complete this level 1 course.

Lessons 21-30 introduced loops. You learned how to use a loop to force the user to answer a question.

1. Replace `???` on line 3 so that the user is forced to enter an answer.
2. Click to test your code, and to confirm that your code passes.

31.2. review loops

You also used a while loop to print a series of numbers in the console.

1. Replace `???` on lines 1, 3 & 5 so that it prints out the multiples of **4**, from **40 to 100**.
2. Click to test your code, and to confirm that your code passes.

31.3. use special characters in strings: `\n \' \" \\ \t`

`\n` is a special character that means **new line**. Other special characters are `\t` tab, `\\` backslash, `\'` single quote, and `\"` double quote.

Use `\'` to put an apostrophe in strings surrounded by `' '`. E.g. `'Don\'t do it!'`

For your 3rd task:

1. Add the missing words on line 3.
2. Add the missing word on line 5.
3. Fix the code on lines 4 & 6 by adding a special character.

31.4. **Refactoring** code to improve readability and efficiency

In lessons 2-4 you will learn to write code that correctly handles invalid user input. You'll also learn to round decimal numbers using the **`toFixed`** function, as shown on line 6.

In lesson 5 you'll practice **refactoring**. **Refactoring** is making small changes to improve the readability and efficiency of code, without changing the results of running it.

For this task, make the following changes to improve the code:

1. Variable names should describe what they are used for; replace the variable `x` with: `dailyHours`
2. This code uses 2 if statements; in this case it is simpler to use 1 **if statement**; replace the 2 if statements with 1 if-else statement.

31.5. use good code comments

Code **comments** are also important for making code easy to read. Comments begin with `//` and explain (in human language) the **purpose** of code. Comments explain **why** code is written the way it is, and **not** just **what** it does.

1. Choose the best comment on lines 1 & 2 and remove the other.
2. Choose the best comment on lines 6 & 7 and remove the other.

32. Handle both valid and invalid user input

32.1. handle valid user input

Doctors recommend 30 minutes of exercise 5 times a week to maintain good physical and mental health. In this lesson you'll create an app that alerts users if they need to do more exercise.

You'll make sure it nicely handles any invalid input from the user.

Write code to do the following:

1. Ask the user how many minutes of exercises they do each week and store the response in a variable `minsPerWeek` (capitalization is important).
2. If the user does at least **150** minutes per week, say: "Very good"
3. Otherwise say: "You should exercise more"

32.2. handle user input outside boundary conditions

Boundaries are the minimum and maximum valid inputs. In this task, the weekly exercise must be between 0 and 10080 (the number of minutes in a week). Values outside this range are mistakes.

Modify your code to display error messages when the user enters inputs outside the boundaries.

1. If the user types a value **less than 0** then display the error message: "Error: number too low"
2. If the user types a value **greater than 10080** then display the error message: "Error: number too high"
3. Otherwise, display the messages from the previous task.

32.3. check for invalid numbers with `isNaN`

The `isNaN` function checks if a value **is N**ot **a N**umber. `isNaN(x)` is **true** only if `x` is **not** a number.

For example, the strings **ten** and **50 mins** are not valid JavaScript numbers So `isNaN('ten')` and `isNaN('50 mins')` are **true**, but `isNaN('10')` and `isNaN('50')` are **false**.

Use `if (isNaN(minsPerWeek))` to check if the user enters a value that is not a number. Notice that `isNaN` has 2 capital **N**s.

1. Click and enter "50 mins" to see what happens.
2. If the user enters a value that is not a number display the message: "Error: not a number"
3. Otherwise, display the messages from the previous tasks

32.4. check for empty input (`''`)

Users may click OK without typing anything in the box. If this happens the `minsPerWeek` variable is set to the **empty string** (`''`).

To check if `minsPerWeek` is set to the empty string, use: `if (minsPerWeek == '')`

1. Click , then click OK without typing anything, to see what happens.
2. At the start of the **if statement**, add an **if** branch to check for the empty string and display a message that says "Error: empty input"
3. Otherwise, display the messages from the previous tasks. Your code should display only 1 alert message each time it runs.

The empty string is converted to 0 by the `Number` and `isNaN` functions. So `Number('')` is 0 and `isNaN('')` is **false**.

32.5. check if user clicked **cancel**

If the user clicks cancel the **prompt** function returns the special value **null** (except in Safari 5, which returns the **empty string**).

1. Click , then click **cancel** and see what happens.
2. If the user clicks **cancel** or **OK** without typing anything display the message that says: "Error: empty input"
3. Otherwise, display the messages from the previous tasks.
4. **(Optional)** If the user does less than 150 minutes, state the number of extra minutes they should do to reach 150.

As with `''`, **null** is converted to 0 by the `Number` and `isNaN` functions. So `Number(null)` is 0 and `isNaN(null)` is **false**.

Remember that || means **OR** and && means **AND**.

33. Using isNaN and Number functions

33.1. use the `Number` function to add user input

To encourage exercise, **S.C.R.I.P.T.** has a 4 week **Step Up** challenge. Teams of 4 use step counters to count how many steps they take.

The code in the editor asks for each team member's step count, then tries to display the **total** and **average**. The code doesn't work correctly because **prompt** returns a string, and + **joins** strings together instead of **adding** them. E.g. `'4' + '5'` is `'45'` (see level 1, lesson 9).

The `Number` function is used to convert a string to a number. So `Number('4') + Number('5')` is 9.

1. Click to see what the code does.
2. Use the `Number` function on line 6 to make the code work correctly.

33.2. check for **cancel** and the **empty string** (`''`)

Both `Number('')` and `Number(null)` evaluate to 0. So if the user clicks OK without typing anything, or clicks cancel, 0 is added to the **total**.

Instead display the following error messages:

1. If the user clicks cancel for 1 of the numbers show a message that says: "Input cancelled"
2. If the user clicks OK without typing anything show a message that says: "Error: empty input"
3. Otherwise, show the message that displays the total and average of the numbers.

33.3. use `isNaN` to check for input that is not a number

This program also needs to check for input that is **not a number** using the `isNaN` function.

1. If the user enters input that is not a number display: "Error: not a number"
2. Otherwise, show the message that displays the messages from the previous tasks.

33.4. understand the special value **NaN**

When the `Number` function tries to convert a string that is not a number the result is a special value called `NaN`. For example, `Number('5')` and `Number(5)` are **5**, but `Number('five')` and `Number('5min')` are `NaN`.

Evaluate the following and replace `???` with the correct answers. Answer 1 is done; notice that there are no quotes around `NaN`. Remember to put quotes around **string** answers. E.g. `'$50.00'`

1. `Number('$5.25')`
2. `'$' + Number('5.25') * 4`
3. `Number('5,000') * 100`
4. `Number(null) + Number('')`
5. `Number('5000 * 100')`
6. `5000 * 'ten'`

Type the questions in the console below to check the answers.

33.5. understand the difference between `NaN` and `isNaN`

`NaN` is a special value that is returned when `Number` tries to convert a string that is not a number. `isNaN` is a function that returns **true** if a value is **not** a number and **false** if it **is** a number.

Evaluate the following and replace `???` with the correct answers. Answer 1 is done; notice that there are no quotes around **false**.

1. `isNaN('12.34')`
2. `isNaN('$125')`
3. `Number('XXX')`
4. `isNaN('  ')`
5. `isNaN(null)`
6. `isNaN(NaN)`

Type in the console to check the answers.

34. Create a BMI calculator

34.1. get input, do calculation, show output

S.C.R.I.P.T. employees have regular physical check-ups. This includes monitoring BMI (body mass index).

In this lesson you will create a BMI calculator.

1. Ask the user's weight in kg and store in a variable called: `mass`
2. Ask the user's height in meters and store in a variable called: `height`
3. Calculate the BMI and store in a variable called `bmi` (all lowercase).
4. Display the BMI in an **alert** message. Don't worry about rounding your answer. We'll do that in the next task.

34.2. use `toFixed` to show rounded numbers

The `toFixed` **function** is used to make numbers display with a set number of decimal places. For example, `alert( (12.3456).toFixed(2) )` displays **12.35** in an alert box.

1. the example code and look at the results in the console.
2. Use `toFixed` to display the BMI in the alert box to 1 decimal place.

34.3. use `isNaN` and `while` to force valid input

For this task, make sure the user enters a valid weight and height.

1. Use a while loop and the `isNaN` function to force the user to enter a number for the **weight**.
2. Use a while loop and the `isNaN` function to force the user to enter a number for the **height**.

34.4. add **boundary conditions** checks using ||

Now add extra checks to make sure the user enters numbers that are valid weights and heights.

1. The heaviest ever adult was 635kg and the lightest was 2.1kg; force the user to enter a weight in this range.
2. The tallest ever adult was 2.72m and the shortest was 0.55m; force the user to enter a height in this range.

34.5. use an **if statement** to display different messages

The World Health Organisation suggest that the healthy BMI range for adults is 18.5 to 25.

In this task check the BMI and show a 2nd alert (**after** the 1st one) with the following message:

1. If less than 18.5 say: "underweight"
2. If more than 25 say: "overweight"
3. If 18.5 to 25 say: "healthy weight"
4. **(Optional)** In the message for steps 1 and 2, state the number of kgs the user is underweight or overweight.

The healthy BMI is less than 23 for South East Asians and less than 26 for Polynesians (see Wikipedia).

35. Making bad code good

35.1. use tabs and blank lines to make code easier to read

This task uses a loop to ask how many minutes the user exercised in the past few days. When cancel is clicked, the loop ends and the code prints the average daily exercise.

The code has several problems that make it hard to read.

Tabs, spaces and blank lines are needed to improve readability.

1. Put blank lines before and after **while** loops and **if statements**.
2. Put a tab or 2 spaces before each line of code inside **while** loops and **if statements**.

If you are stuck, click then "Compare code".

35.2. use good variable names

Variable names should clearly describe what they are used for.

`a` , `b`, `c` and `d` are usually **bad** variable names.

Study the code and replace `a`, `b`, `c` and `d` with the correct **good** variable names from the following list:

1. `averageExerciseTime`
2. `exerciseTime`
3. `dayNo`
4. `totalExerciseTime`

35.3. use shorthands to increase and decrease variable values

The code `x = x + 1` increases the value of x variable by 1; `x += 1` is shorthand for doing the same thing. `y = y - 5` decreases the y variable by 5; `y -= 5` is the shorthand. This shorthand makes the code easier to read and faster to type.

1. Use shorthand once with the `dayNo` variable.
2. Use shorthand once with the `totalExerciseTime` variable.

35.4. use shorthands to increase and decrease variable values

The 2 shorthands for `x = x + 1` are `x += 1` and `x++`. You will use the `++` shorthand a lot!

The shorthands for `x = x - 1` are `x -= 1` and `x--`.

1. Use the `++` shorthand once in the code.
2. On line 1 put a comment about the purpose of the code.
3. Add a comment just before the **while** loop to explain that part of the code.

35.5. avoid repeated code

It's good to avoid repeated code. The prompt messages on lines 5 & 11 were very similar. On line 2 a variable is set to the part of the messages that was the same. The messages on lines 18 & 20 are very similar too. A variable on line 15 is set to the part of the message that's the same.

1. Use the `exerciseQuestion` variable in line 11 as done on line 5.
2. Use the `exerciseMessage` variable in lines 18 & 20 to remove the repeated code.
3. **(Optional)** Fix the code to work nicely when the user clicks cancel on the 1st prompt.

In the next lesson you'll learn about the benefits of replacing **"evil magic numbers"** (see line 17) with **named constants**.

36. Use {{break}} to exit loops

36.1. use a `while` loop with 3 conditions separated by ||

This lesson creates a shopping calculator.

For the task 1:

1. Ask the user's spending limit and store it in a variable called: `money`
2. If the user clicks cancel, types 0 or less, or a value that is not a number, keep asking the question until the user types valid input.

36.2. use `!=` (not equals) in a loop condition

For task 2:

1. After the loop, ask the user to enter the cost of an item; store the response in a variable called: `cost`
2. Re-ask the question **until** the user clicks cancel (use `!=` in the loop condition).

36.3. use `&&` in a loop condition

For task 3:

1. Subtract each item `cost` from the `money` remaining.
2. Loop while there is money remaining and the user doesn't click **cancel**.

36.4. use a **break** statement

The **break** statement is used to exit the middle of a loop before the loop condition is false.

1. Place the following code on the line after the last prompt (inside the 2nd loop)
```
if ( ??? ) {
  break;
}
```
2. Change `???` so the loop **breaks** when the user types a cost that is **not a number** or is **less than or equal to 0**.

36.5. stop invalid negative variable value

For the final task:

1. If the user enters a cost greater than the money remaining, show an `alert` message that says "Too expensive"; use the `Number` function to convert `money` and/or `cost` from **strings** to numbers so that the comparison works correctly. Then continue the loop and ask them to enter another value.
2. Subtract `cost` from `money` if cost is less than or equal to the money remaining.
3. The loops ends when there is no more money, or the user enters a value that is not a positive number. After the loop ends, display a message that states the amount of money left over, if and only if the user has money left over.

37. Learn to use {{for}} loops

37.1. create a `while` loop that repeats a set number of times

This lesson calculates non-work related time spent this week consuming electronic media, e.g. TV, movies, radio, social media and gaming.

This task uses a **while** loop; the next task replaces it with a **for** loop, which is a more concise loop in certain cases.

1. Replace `???` on line 5 to make the loop repeat 7 times (use the `totalDays` and `dayNo` variables).
2. Calculate and store the total hours in a variable called: `totalHours`. You may need to create an additional variable to do so.
3. Display the total hours in an **alert** message after the loop completes.

You can assume the user enters valid numbers.

37.2. change a `while` to a `for` loop

A **for** loop runs code a set number of times, using this format:
```
for (initialization; condition; final) {
  statements
}
```
The **initialization** statement runs 1st; the example initialization sets a variable `i` to **0**.

Next, if the **condition** is true the **statements** are run, followed by the **final** code. In the example in the bottom right, `i < 10` is true so `i` is printed in the console then increased by **1**.

The loop continues to run until the condition is false. In the example the loop runs 10 times until **i** becomes 10.

1. Programmers normally use `i` as a loop counter variable because it is quick to type; replace `dayNo` with `i`.
2. Change the **while** loop to a **for** loop.

37.3. calculate and display the average value

For this task do the following:

1. Store the average number of daily hours in a variable called: `averageHours`
2. Use `toFixed` to display the average rounded to 1 decimal place on the 2nd line in the **alert** message.

E.g. if the average is 10, the alert message will say: `The total hours is 70`
`The average hours is 10.0`

37.4. use `break` in a `for` loop

The `break` keyword can be used in both **for** and **while** loops.

1. If the user **cancels** the prompt or types a value that is **not a number** or **less than 0**, break out of the loop and show the average so far.
2. If the user doesn't enter any valid hours don't show the alert.

37.5. do calculation and show rounded answer

It is said that it takes 10,000 hours of practice to become an expert at something.

1. Calculate the number of years to be an expert media consumer; round to 1**dp** and put in a variable called `yearsToExpert`; use **365.242** for days per year.
2. If yearsToExpert is less than 10 show a 2nd alert message that says: **You'll be an expert timewaster in X.X years**, where X.X is the value of `yearsToExpert`.

38. Learn to understand {{for}} loops

38.1. count the number of times a loop runs

Work out the number of times the **alert** inside each of the following loops will be shown, then click and enter your answers.

1. ```
   for (var x = 0; x < 2; x++) {
     alert('Hello!');
   }
   ```
2. ```
   for (var y = 2; y >= 0; y--) {
     alert('Goodbye!');
   }
   ```

Remember `x--` is shorthand for `x = x - 1` and `x++` is shorthand for `x = x + 1`

The answers are **2** and **3**.

38.2. recognise infinite loops

An incorrect loop condition can cause an **infinite loop** - a loop that repeats over and over and never stops.

Work out the number of times the **alert** inside each of the following loops will show. Enter **i** if it is an infinite loop.

1. ```
   for (var x = 2;  x != 8; x += 2) {
     alert('Looping...');
   }
   ```
2. ```
   for (var x = 3;  x != 9; x += 4) {
     alert('More looping...');
   }
   ```

Remember `!=` means not equal to; `x += 2` is shorthand for `x = x + 2`

The answers are **3** and **i** (infinite loop)).

38.3. calculations in loops

Study the code then click . Use a pen and paper to work out the answer.

1. Answer the question in prompt 1.
2. Answer the question in prompt 2.

`x *= 2` is shorthand for `x = x * 2`

The answers are **8** and **64**.

38.4. complex calculations in loops

Study the code then click . Use a pen and paper to work out the answer.

1. Answer the question in prompt 1.
2. Answer the question in prompt 2.

The answers are **9** and **3**.

38.5. advanced calculations in loops

Study the code then click . Use a pen and paper to work out the answer.

1. Answer the question in prompt 1.
2. Answer the question in prompt 2.

The answers are **0** and **5**.

39. Practice {{for}} and {{while}} loops

39.1. use a `for` loop to move the robot forward

In this task you can only move the robot forward 1 square at a time using `robot.forward()`. You can't use `robot.forward(5)` to move forward 5 squares; you must use a loop to do so.

1. Use a **for** loop to move the robot to the **X**.
2. Click to run the code.

The example code shows an example **for** loop.

Programmers normally use the variable `i` as a loop counter.

39.2. use `distanceToWall` to decide how many times to loop

`robot.distanceToWall()` checks how far the robot can move forward to get to a wall.

1. Modify the code to get all 5 robots to the **X** without hitting a wall; the only number in the code should be **0**.
2. Click to run the code for each robot.

39.3. use a `for` loop inside a `while` loop

Loops can be placed inside of loops; this is called **nesting**.

Extend your code from the previous task to get the robot to the **X**.

1. Put the `for` loop inside a `while` loop.
2. In the while condition check if the robot is at the goal.
3. Use `forward`, `right`, `distanceToWall` and `distanceToGoal` once each.

In this code the only number allowed is **0**. Also, remember that each robot command begins with `robot.` and ends with `()`.

39.4. a difficult challenge for experts

Complete the code to get the robot to the **X**. Use the commands **forward** and `distanceToGoal` once, **right** 3 times and `distanceToWall` twice.

1. Replace the `???` on line 1.
2. Replace the `???` on lines 2 & 4.
3. Replace the `???` on lines 7-10.

39.5.

Complete the code to get the robot to the **X**. Use the `forward`, `right`, `distanceToWall` and `distanceToGoal` commands.

1. Replace the `???` on lines 1 with the correct command.
2. Replace the `???` on lines 4 & 16 with the same command.
3. Replace the `???` on lines 6 & 13 with the same command.
4. Replace the `???` on lines 7, 14 & 15 with the same command.

40. Review lessons 31-39

40.1. review the `Number` function and `NaN` special value

The **Number** function converts strings to numbers; if the string is not a number it returns **NaN**. The **Number** function must be used to add numbers entered in a prompt box.

The special value **null**, an empty string, and a string that contains only spaces and new lines all convert to 0, e.g. `Number(' \n ')` is 0.

Click the **start quiz** button to start the review quiz.

Review Quiz Questions:

1. {whatsTheResult} `'12' + '34'`
2. {whatsTheResult} `Number('12') + '34'`
3. {whatsTheResult} `Number('12') + Number('34')`
4. Which of the following is **NOT** equal to the number 0?

40.2. review the `isNaN` function

This task will review the **is not a number** function.

Remember that `isNaN(null)` and `isNaN('')` are all false, because `null` and `''` are automatically converted to the number 0.

Click the **start quiz** button to start the review quiz.

Review Quiz Questions:

1. What is the correct capitalisation for the **is not a number** function?
2. Which of the following is **true**?
3. Which of the following is **false**?
4. Which of the following is **true**?

40.3. review unexpected inputs

`Infinity` is a special value returned when you divide by 0.

Click the **start quiz** button to start the review quiz.

Review Quiz Questions:

1. {whatsTheResult} `1234 / 0`
2. {whatsTheResult} `'100 apples' / 5`
3. Which of the following checks if the user clicked cancel or typed an invalid number into a prompt box:
4. {whatsTheResult} `'5' * '4'`

40.4. review `for` loops

A **constant** is a variable with a value that should **not** be changed.

In JavaScript and **other programming languages**, we normally use all capitals with _ between words in constant variable names.

Click the **start quiz** button to start the review quiz.

Review Quiz Questions:

1. {howManyTimes}:
```
var MAX = 4;
for (var i = 0; i < MAX; i++) {
```
2. {howManyTimes}:
```
var POPULATION = 107;
for (var i = 0; i < POPULATION; i++) {
```
3. {howManyTimes}:
```
var MIN_DOGS = 5;
for (var i = MIN_DOGS; i > 1; i--) {
```
4. {howManyTimes}:
```
var MAX_CATS = 3;
for (var i = 0; i > MAX_CATS; i--) {
```

40.5. more `for` loop review

Remember that `!=` means **not equals**.

Click the **start quiz** button to start the review quiz.

Review Quiz Questions:

1. {howManyTimes}:
```
var GAP = 2;
for (var i = 0; i < 4; i+=GAP) {
```
2. {howManyTimes}:
```
for (var i = 9; i <= 15; i+=3) {
```
3. {howManyTimes}:
```
var LIVES = 10;
for (var i = LIVES; i != 0; i-=10) {
```
4. {howManyTimes}:
```
var START_POS = 8;
for (var i = START_POS; i != 1; i-=3) {
```