CODE AVENGERS

JavaScript 2 Instructions

1. Intro to lessons 2-10

1.1. revise input and output

Level 2 **JavaScript** has 40 lessons that cover **strings**, **functions**, **arrays** and the **Math object**. You'll do basic graphics, debugging, robot challenges, quizzes and more.

This lesson has 5 tasks that introduce lessons 2-10.

For task 1 make the code do the following:

1. Display a message box that asks the user their name, and store the result in a variable called: `name`
2. Display a message box that says hello to the user, and on a **new line** says: "Welcome to level 2 JavaScript!"

Click to run your code and to check if it's correct.

If you are stuck click or use the reference.

1.2. loop revision

In level 1 JavaScript you used **while** loops to ask the user the same question several times.

The code in the editor calculates and displays the total of numbers entered by the user. Remember that `Number(number)` on line 6 is used to convert the **prompt** input from a string to a number so that + works correctly.

1. Click .
2. Enter 4 numbers that add to 90.
3. Enter a value that stops the loop.

Click run, type **20** and click **OK**, then repeat three more times typing the numbers **20**, **20**, and **30**. Type **0** then click **OK**

1.3. **while** loop revision

In level 1 JavaScript you used loops to run the same code multiple times.

The 2 loops in the editor print the numbers 1 to 4.

For this task:

1. Change the while loop to print the numbers: "0, 2, 4, 6, 8"
2. Try to change the for loop to print the numbers: "4, 8, 12, 16, 20"

1.4. introduction to string functions

In lessons 2-10 you will learn how to use **string functions** and **properties**.

You will learn how to use the `length` property to get the number of characters in a string.

You will also use the `charAt` function to get individual characters in a string.

the code, enter your names and click OK.

Click run, type your first name and click **OK**, then type your last name and click **OK**.

1.5. introduction to string functions

You will also learn 3 other **string functions**: `toUpperCase`, `toLowerCase` and `substring`.

the code and enter your name to see `toUpperCase` in action.

Click run, type your name, then click **OK**

2. Encoding letters with [[ASCII]] codes

2.1. ASCII codes

A computer stores each character in a string as a number called an **ASCII** code.

The table below contains the **ASCII** codes for 128 common characters.

Use the table to set the variables `answer2`, `answer3` and `answer4` to the correct **ASCII** codes.

Question 1 is an example, and has been done for you.

1. The **space** character.
2. The lowercase letter **a**.
3. The uppercase letter **Z**.
4. The number **0**.
5. Click to check your answers.

The codes 0-31 are for unprintable characters like tab, newline and backspace.

The code 32 is **space**.

The code 127 is **delete**.

2.2. ASCII symbols

Question 1 is an example, and has been done for you.

What are the **ASCII** codes for:

1. The symbol used in the boolean **OR**.
2. The multiplication symbol.
3. The division symbol.
4. The less than symbol.

2.3. decode ASCII

Question 1 is an example, and has been done for you.

Decipher the following words and put the answers in the variables:

1. 65 83 67 73 73
2. 83 112 97 109
3. 87 48 82 77
4. 77 97 76 119 65 114 69

2.4. comparing strings

< and > can be used to compare characters based on the value of their **ASCII** codes.

Question 1 is an example, and has been done for you.

Set the value of the variables to `true` or `false` to answer the following:

1. `'9' < 'A'`
2. `'a' < 'z'`
3. `'@' > 'A'`
4. `'a' < 'Z'`

2.5. ASCII codes

Question 1 is an example, and has been done for you.

Enter the **ASCII numbers** for the following:

1. Symbol that is > `'9'` and < `'A'` and is used in all email addresses.
2. Punctuation symbol that is > `'='` and < `'Z'`.
3. A valid variable name character that is > `'Z'` and < `'a'`.
4. The symbol that is put at the end of a group of statements in an **if statement** branch.

3. String [[length]] & [[charAt]]

3.1. string `length` property

In this lesson you'll practice using several string functions and properties.

1. Ask the user which country they live in and store it in a **variable** called: `country`
2. On the next line add the code: `alert(country.length)`
3. the code at least twice and see what `.length` does.

`.length` gets the number of characters in a **string** including spaces.

3.2. use **length** to check for valid input

The shortest valid country is 2 letters, e.g. NZ, US, UK etc.

The longest official country name is **al-Jumhuriyah al-Jaza'iriyah ad-Dimuqrāṭīyah ash-Sha'bīyah** (a.k.a. Algeria) at 62 characters including spaces.

Thank the user if they type a response with a length between **2** and **62** inclusive.

1. Replace the 1st `?` to check if the name is at least 2 letters long.
2. Replace the 2nd `?` with the correct symbol.
3. Replace the 3rd `?` to check if the name is no more than 62 letters.
4. Replace the last `?` to display an alert that thanks the user. If the country is invalid no alert is shown.
5. For each task in this lesson click to test your code, then click to see if it passes.

Click for definitions: <, <=, >, >=, ||, &&

3.3. use **length** to check for valid input

S.C.R.I.P.T. Agents use a 3 letter codename instead of their real names. Newly hired Agents choose a codename.

1. Replace the `?` on line 3.
2. Replace the `?` on line 5.

3.4. use `charAt` to show each letter in a string

If the user types a 3 letter codename, show each letter in a separate alert. Use the `charAt` string function to complete the task.

The `charAt` function gets a particular character in a string.
```
var s = 'Firewall'
var c = s.charAt(3)
```

In the code above, the value of the variable `c` is `'e'`, the 4th character in `'Firewall'`.

You may ask, why not the 3rd character? In computer programming we begin counting from 0, so `s.charAt(0)` would be `'F'`, `s.charAt(1)` would be `'i'` etc.

1. Study the code in the editor.
2. Replace the `?` on line 8 to show the 1st letter of the codename.
3. Replace the `?` on line 9 to show the 2nd letter of the codename.
4. Replace the `?` on line 10 to show the 3rd letter of the codename.

3.5. use **charAt** to check for valid input

There are more rules when choosing your 3 letter codename. The 1st and 2nd characters must be capital letters, and the 3rd must be a number from 1 to 9.

For this task, alert the user if any letters in the codename are **invalid**.

1. Replace the `?` on line 7.
2. Replace the `?` on line 9.
3. Replace the `?` on line 11.
4. Replace the `?` on line 12.

Type `||` using the key near **Enter** that also has ****. `codeName.charAt(0)` gets the 1st letter in the codename. You can find the **ASCII table** in the toolbox.

4. [[alert]], [[prompt]] & [[string]] review

4.1. review **alert**

This lesson combines if statements, loops and string functions.

Complete the following steps:

1. Use `alert` to say `hello` to the user.
2. After each task in this lesson click to run your code and see if the code passes.

4.2. review **prompt** and **length**

Complete the following steps:

1. Change **alert** to **prompt** and change the message to **say hello** and ask the **user's name**.
2. Store the user's response in a **variable** called: `name`
3. On the next line of code, use an **alert** to show the number of letters in the name.

4.3.

Complete the following steps:

1. Use a **while** loop to re-ask the user's name if they click cancel; put the response in the **name** variable.
2. At the end of your code show the `alert` with the number of letters.

Do you remember what to do if the user clicks cancel **twice**?

4.4. show a different message depending on **string** length

Show the following **alert** messages:

1. If the name length is **less than 3** say: "too short"
2. If the length is **15 or more** characters say: "too long"
3. If the length is in between show a message that says the length of the name

Click for definitions: <=, >=, ==, !=

4.5. check for a valid human name

Remember that in computer programming we often begin counting at 0.

1. After the end of the **loop** store the 1st letter of **name** in a variable called: `first`
2. On the next line store the last letter of **name** in a variable called: `last`
3. Add an `else if` to display the message **invalid name** if the 1st letter is not an uppercase letter (A-Z) or the last letter is not a lowercase letter (a-z).
4. As in the previous task, if the name is valid, show the message stating the length of the name.

5. Fix broken code

5.1. fix 2 bugs so that the code runs correctly

If a **string** value has an apostrophe (') in it, put " around the text instead of ' , as on line 2.

1. Fix the bug on line 3.
2. Fix the bug on line 4.
3. For each task in this lesson click to run your code, then click to see if it passes.

5.2. find and fix 3 bugs

The string function `toLowerCase` converts a **string** to its lowercase form. This enables you to ignore capitalization when comparing strings.

1. Fix the bug on line 1.
2. Fix the bug on line 2.
3. Fix the bug on line 3.

5.3. find and fix 2 bugs

1. Fix the bug on line 2.
2. Fix the bug on lines 3-4

5.4. find and fix 2 bugs

A program works correctly without spaces around =, as on line 1. Spaces are usually put around = to make code easier to read.

1. Fix the bug on line 1.
2. Fix the bug on line 2.

5.5. find and fix 2 bugs

1. Fix the 1st bug.
2. Fix the 2nd bug.

6. Move robots with [[if statements]]

6.1. move the robots to the **X**

For this lesson move the robots **without** using loops, e.g. `while`.

`if (!robot.onX())` checks if the robot has **not** reached the **X**.

1. Replace the `?` on line 1 with a number.
2. Replace the `?` on line 3 with a number.
3. Copy (ctrl+c) and paste (ctrl+v) the **if statement** twice to move all 4 robots to the **X**.
4. For each task in this lesson click to see if your code passes.

6.2. move the robots to the **X**

Extend your code to move all 6 robots to the **X** and stop.

`robot.onX()` is true if the robot is on the **X**.

`robot.left()` and `robot.right()` turn the robot 90 degrees.

6.3. move the robots to the **X**

The robot has another **sensor** that detects the distance to the **X**. The command `robot.distanceToX()` gets the number of blocks the robot must move forward to reach the **X**.

For example, with the top robot `robot.distanceToX()` is equal to **6**; for the bottom robot it equals **2**.

Use 1 `forward` and 1 `distanceToX` command to get all 4 robots to the **X**.

6.4. move the robots to the **X**

The command `robot.distanceToX()` gets the **distance** to the **X**, but not the direction.

1. Write code to get all 6 robots to the **X** using no more than 8 commands.
2. to see how your code for the previous task works with this task.

This task can be done with 1 `right`, 2 `forward` and 2 `distanceToX` commands.

6.5. move the robots to the **X**

Complete the code to get each robot to the **X** in no more than 8 steps.

Each 90 degree turn and move forward by a block counts as 1 step.

Compare `d` and `d2` to work out if the **X** is on the left or right of the robot.

7. [[toLowerCase]], [[toUpperCase]], [[substring]]

7.1. `toLowerCase` and `toUpperCase` introduction

This lesson introduces several functions that are used to modify strings. `toUpperCase()` converts all the letters in a string to uppercase. `toLowerCase()` converts all letters to lowercase.

7.2. `substring` instruction

The `substring(n)` function gets part of a string, from the **nth** character to the end of the string.

Since we begin counting at 0, `substring(2)` starts at the 3rd character.

7.3. `substring` introduction

To get the middle part of a string use `substring(start, end)` where start and end are the number of the characters that start and end the substring. The substring goes up to but **not** including the end character.

For example, `var a = 'example'.substring(2,4)` sets the variable a to **am**. Remember to start counting at 0, so `substring(2,4)` starts at the 3rd character and goes up to but **not** including the 5th character.

7.4. `substring` introduction

7.5. `charAt` introduction

Functions can be used one after another on the same line of code.

For example, line 3 in the example editor uses `charAt` to get the 1st character in a string and then uses `toLowerCase` to convert the character to lowercase.

8. String functions practice

8.1. using string functions

In the last lesson you learned about the string functions `toLowerCase`, `toUpperCase` and `substring`. In this lesson you will practice using them.

1. the code twice to see what the functions do.
2. Change the numbers on line 4 to experiment with `substring`.

Click run, type **Student** and click **OK**. Run the code again but type **Teacher** this time

8.2.

S.C.R.I.P.T. is recruiting 2 new agents and 3 office staff, but 0 engineers and cleaners.

Modify the code to do the following:

1. If the user types **agent** or **office**, display a message stating the number of jobs available.
2. If the user types anything else, display a message that says: "0 jobs available"
3. For the rest of the tasks in this lesson click to test your code, then click to see if your code passes the task.

Remember to use == to check if a variable is equal to some value.

8.3.

With the code from task 2, if the user types **Agent** (with a capital A) or **OFFICE** (all capitals), the message will say **0 jobs available**.

It is better to show the same message whether the user types **Agent**, **AGENT** or **agent**.

1. Use the `toLowerCase` string function to improve the code.
2. Click to test and to see if your code passes.

8.4.

What if the user misspells **ofice** or adds an **s** and types **offices**? With the code from task 3 the message will say **0 jobs available**.

1. Use `substring` and `toLowerCase` to put the lowercase form of the 1st **2** characters of the **job** variable in a variable called: `jobStart`
2. Show the message for **agent** if the user input begins with **ag**.
3. Show the message for **office** if the input begins with **of**.
4. Otherwise, show the message: "0 jobs available"

the example code to experiment with `substring` and `toUpperCase`.

8.5.

New jobs are available at **S.C.R.I.P.T.**

We are now hiring the following:

1. 2 agents
2. 2 office workers
3. 1 engineer
4. 1 cleaner

Update the code for the new jobs; as with the others check the 1st 2 letters of the input. For all other jobs, display the message: "0 jobs available"

9. Create a multi-choice quiz

9.1. add multi-choice quiz question

The **S.C.R.I.P.T.** job application site includes a test of computer knowledge. The code in the editor contains the 1st question in the test.

the code to see how it works.

1. Add the 2nd question: **What is a firewall?**
   The 4 options are **a. Antivirus program**, **b. Internet blocker**, **c. Internet filter** and **d. Internet logger**.
   The correct answer is **C**
2. For each task in this lesson click to test your code, then click to see if your code passes the task.

Remember that `\n` is a special value meaning newline.

Also remember that the keyword `var` is only needed the 1st time you create a variable; so the code for the 2nd question should begin with `guess = prompt('`

9.2. add string quiz question

The code in the editor contains the 3rd test question. Your task is to add the 4th question after the 3rd.

1. Question 4 is **What 8 letter D word means to retrieve a file from the internet?**
2. Click if you don't know the answer.

On line 3 `guess != null` checks that the user did **not** click cancel, and `guess.toLowerCase()` is used so that **SPAM**, **Spam** or **spam** will all be correct.

9.3. increase a score variable

In the next task you will use a **variable** to count the number of correct answers. To do this you create a variable and add 1 to its value each time the user answers a question correctly.

For this task, the code in the editor increases the **score** variable by 1.

1. Change lines 4 & 7 to increase the **score** variable by **5**.
2. Add code to line 10 to increase the **score** by 5 for a 3rd time.
3. On line 11 show the final value of **score** with an alert.

The code `score++` and `score+=1` are shortcuts for `score = score + 1`

9.4. count the quiz score

Now let's add the scoring code to our test.

1. Add 1 to the `score` variable (created on line 1) for every correct answer.
2. At the end of the test display the user's score.

9.5. add the final question

1. Add the 5th and final test question: "What 6 letter C word stores your settings when browsing web pages?"
2. This question is difficult, so make it worth **2 points**
3. Click if you don't know the answer to the test question

10. Review strings

10.1. review **string** creation

Review Quiz Questions:

1. Which of the following pair of characters is used to surround a string value?
2. Which of the following symbols joins two strings together?
3. What other pair of characters can be used to surround a string value?
4. What special character is used to display part of a string on a **newline**?

10.2. review **string functions**

Remember to start counting at 0 for `charAt`.

Review Quiz Questions:

1. Which of the following is `true`
2. `'Dr. Hackit'.length`
3. `'FiReWaLL'.charAt(5)`
4. `'virus alert'.charAt('virus'.length)`

10.3. review **string** more **functions**

Review Quiz Questions:

1. `'FiReWaLL'.toUpperCase()`
2. `'firewall'.substring(4)`
3. `'firewall'.substring(2,6)`
4. `'firewall'.substring(0, 12)`

10.4. review using + **strings** and **numbers**

+ is used to **join** 2 **strings** together E.g. `'Code' + 'Avengers'` equals `'CodeAvengers'`

+ is used to **add** numbers together E.g. `5 + 8` equals `13`

+ is used to **join** a string and a number into a single string E.g. `5 + '8'` equals `'58'`

Review Quiz Questions:

1. `'a' + 'b'`
2. `'5' + '6'`
3. `2 + '4'`
4. `1 + 2 + '3'`

10.5. final review

Review Quiz Questions:

1. If `a = '2'` and `b = '4'`, then `a + b` is ...
2. If `a = '2'` and `b = '4'`...
   then `Number(a) + Number(b)` is ...
3. If `a = 'JAVA'` and `b = 'SCRIPT'`...
   then `b.charAt(1) + a.charAt(a.length-1)` is ...
4. If `a = 'JAVA'` and `b = 'SCRIPT'`...
   then `a.substring(1,3) + b.substring(3,4)` is ...

11. Intro to lessons 12-20

11.1. introduction to **functions**

A **function** is a named reusable section of code that does a specific task and may return a value.

In level 1 you used **alert**, **prompt** and **confirm** functions (commands provided by your web browser) to display message boxes. You used the **Number** function to convert strings to numbers, and robot functions to move a robot.

For your 1st task:

1. Replace `???` on line 1 with the correct function name.
2. Replace `???` on line 4 with the function to show a message box.

11.2. **call** a function and **comment out**

S.C.R.I.P.T. is renovating one of its buildings. In the next few lessons you'll create **functions** to calculate the renovation costs.

Lines 2-4 create a simple function; line 6 **calls** (or runs) the function.

1. Click to see what happens.
2. **Comment out** line 6.
3. Click and see what happens.

11.3. introduction to **vector graphics** with Paper.js

Lessons 17-19 use the Paper.js library to draw simple graphics. Lessons 27-29 create games with Paper.js.

The code in the editor uses Paper.js to create bouncing balls. It may look confusing now, but when you are finished Code Avengers it will all make sense!

1. Click to show the Paper.js Canvas, which automatically updates as the code changes.
2. Click the canvas to make balls appear.
3. Change the **gravity** on line 2 to **-2**.
4. Change the **radius** on line 4 to **40**.
5. Change the **color** on line 5 to **yellow**.
6. Click when you are done.

11.4. modify another Paper.js example

This task has some more example Paper.js code. Even simple 3d graphics like this one use complicated math!

1. Click to show the canvas.
2. Change the number of rounded rectangles to **10**.
3. Change the list of colors to **black**, **red**, **black**, **gold**.

Click here for lots of cool Paper.js examples.

11.5. move Paper.js objects

S.C.R.I.P.T. is constructing a garden containing scale models of man-made wonders from around the world like the Pyramids of Giza, the Parthenon and the Leaning Tower of Piza.

In lessons 17-19 you will draw these buildings. But before you create complex buildings, let's start by laying a few bricks.

The function on lines 1-6 draws a brick. The code on lines 10-12 calls the `drawBrick` function 3 times.

1. Click to automatically update the canvas as the code changes.
2. Change line 11, to draw the brick next to the brick on line 10; the answer appears lightly in the background.
3. Change line 12, to draw the brick on top of the other 2 bricks.

12. Functions introduction

12.1. review functions from level 1

This code should ask the user to enter 3 numbers, then display the total in a message box.

1. Replace `???` on lines 1-3 with the correct function name.
2. Replace `???` on line 5 with the correct function name.
3. Replace `???` on line 7 with the function to show a message box.

12.2. use a simple function

A simple function has the following format: `function nameOfFunction() { //a few lines of code }`

The code from task 1 is now inside a function called `getItemTotal`.

1. Click and see that the code in the function does NOT run.
2. Put the code `getItemTotal()` twice on lines 14 & 15.
3. Click and see that the function code runs twice.

12.3. use a function with a parameter

The code `alert('Hello')` **calls** or runs the alert function and displays the message: "Hello"

The value between the `()` is called a **parameter** Parameters make functions more useful.

The `getItemTotal` function on lines 14-22 always calculates the total of 3 item counts. On lines 2-11 `getItemTotal2` uses a parameter to choose how many items it asks for. E.g. the code `getItemTotal2(5)` gets the function to calculate the total of 5 item counts.

1. On line 24 use `getItemTotal2` to get the total of 2 item counts.
2. On line 25 use `getItemTotal2` to get the total of 4 item counts.

12.4. use a function with 2 parameters

Functions like **prompt** can take more than 1 parameter. In the code: `var name = prompt('Enter your name:', 'Anon')` parameter 1 is the message and parameter 2 is the default text.

S.C.R.I.P.T. is replacing computers in its training labs. The `getTotalCost` function on lines 2-16 asks the user to enter the number of computers that need replacing in each lab, then calculates and displays the total cost The function takes 2 parameters: `labCount` is the number of labs that need new computers; `itemCost` is the cost of each computer.

1. Replace the `???` on line 13 to calculate the total cost of all the computers.
2. On line 18 use the function to calculate and display the total cost of purchasing computers for **$875** each for **5** labs.

12.5. use a function with a return value

The JavaScript **prompt** function on line 6 **returns** the value entered by the user and stores it in the `count` variable.

The new `getTotalCost` function **returns** the total cost on line 13 (instead of showing it in the console). Lines 16-17 store the result of `getTotalCost` in a variable called `computerCost` and show it in the console.

In addition to the new computers, 3 labs need new desks costing $89 each.

1. On line 19 call the `getTotalCost` function to get the cost of the new desks; store the result in a variable called: `deskCost`
2. On line 20, use the same format as line 17 to display the total cost of the desks in the console.
3. On line 22, use the same format to display the total cost of desks and computers in the console.

13. Learn to understand functions

13.1. determine return values of functions with 1 parameter

This lesson uses 1 letter function and variable names so you can focus on working out what the functions do. Normally, function and variables names should clearly describe their purpose.

13.2. determine return values of functions with 2 parameters

It's ok for 2 functions to have the same parameter names.

The `m` and `n` variables in function **x** can only be accessed inside that function; `m` and `n` are deleted when the function ends.

13.3. determine return values of functions with if statements

13.4. determine return values of boolean functions with 1 parameter

As well as returning numbers and letters, functions can also return the **boolean** values: **true** or **false** as with function **y** on lines 5-10.

A function can also return the result of an **expression** that is either **true** or **false**, as with `function x` on lines 1-3.

When function **x** is called on line 12, the **a** variable is set to **false** since `l >= 'A' && l <= 'Z'` is **false** when the `l` parameter has a value of `'m'`.

13.5. determine return values of boolean functions with 1 parameter

These functions check if a letter is a vowel; however, they are slightly different.

14. Intro to drawing

14.1. create a **Path** object to draw a straight line

Up till now, you have set variables to numeric, string and boolean values. You can also set variables to more complex objects. To draw on the canvas you create a Path object as shown in the editor.

1. Line 1 creates an empty path and assigns it to `p`.
2. Line 2 sets the color of the path.
3. Lines 3 and 4 create and add start and end points to the path.

Notice that The **new** keyword is used to create **Point** and **Path** objects.

For this task:

1. Click to show the canvas in auto update mode.
2. Move the **blue** line to cover the red line at the top of the canvas.
3. Change the color to **red**.

14.2. use a **Path** object to draw a square

You can create a triangle by adding another point to the path.

1. Click to show the canvas in auto update mode.
2. Add `p.add(new Point(250, 300))` to the end of the code.
3. On the next line, add `p.closed = true`; this closes the path to make a triangle.
4. Change the last point and add a new point to create a square that goes around the outside of the canvas.
5. Change the color to **green**.

Use the grid on the canvas as a guide; make sure your square covers the answer that appears lightly in the background.

14.3. use the `strokeWidth` property to increase the width of a line

Your drawing canvas has a width and height of 500 pixels. A **pixel** is one of the thousands of tiny dots of light which display colors, images and text on a screen.

By default the width of a path object is 1 pixel. This width is increased by setting the `strokeWidth` property.

1. Add the code `p.strokeWidth = 5` to increase the stroke width of the **green** box to 5px.
2. After the box code, add the code `p = new Path()` to assign p to a new path object.
3. Set the stroke width of p to **10** and the color to **black**.
4. Add 2 points to p to make it go from top to bottom through the middle of the square.
5. Create a 3rd path object that makes a similar line from left to right.

14.4. create circle objects

The picture you are drawing is a map of the **S.C.R.I.P.T.** headquarters, where each pixel is 1 meter. The black lines are 10 meter wide roads.

The example code draws a circle with a center point of 50, 50 and a radius of 40.

1. the example code; change the code to move the circle and make it bigger; click to redraw it.
2. Click to show the map.
3. Create a **black** circle object called **`roundabout`** with a `strokeWidth` of 10; position it over the small **black** circle in the middle.
4. Create a **red** circle object called `mainOffice` with a `strokeWidth` of 3; position it over the big **red** circle.

14.5. create regular polygon objects

The red circle in the top left is the main office building. It's actual shape is a **pentagon** (a 5-sided regular polygon).

The example code draws an 8-sided **regular polygon** with a radius of 40. All the sides on a regular polygon are the same length.

1. Change `mainOffice` to a pentagon.
2. Create the small green pentagon using a variable called `plaza`; use the same center as `mainOffice`.

15. Fix broken functions

15.1. fix bugs in function code

Each task has a couple of mistakes to fix.

1. the code and read the error message in the console.
2. Fix the mistake on line 1.
3. Fix the mistake on line 10.

15.2. fix **syntax errors**

The **syntax** of a programming language is the rules that define how to combine the symbols, keywords and variable names etc. into valid code. Bugs that break these rules are called **syntax errors**.

This task has 2 **syntax errors** to fix.

1. the code and read the error message in the console.
2. Fix the mistake with the function on lines 13-17.
3. Fix the mistake on line 20.

15.3. fix **logic errors**

Logic errors cause a program to run incorrectly. For example, the **min** function on lines 2-8 has a logic error that makes it return the maximum value instead of the minimum value.

1. the code and read the error message in the console.
2. Fix the **logic error** in the function on lines 2-8.
3. Fix the **logic error** in the function on lines 11-13.

15.4. fix **syntax errors** in function code

This code is the same as the previous task, except this time there are syntax and logic errors.

1. the code and read the error message in the console.
2. Fix the **logic error** in the function on lines 2-7.
3. Fix the **syntax error** in the function on lines 11-13.

15.5. fix **syntax errors** in function calls

In this task, the function code is correct. However, the code that calls the function has **syntax errors**.

1. the code and read the error message in the console.
2. Fix the **syntax errors** on line 19.
3. Fix the **syntax errors** on lines 21 & 22.

16. Variable scope and parameters

16.1. determine the value of variables changed by a function

Functions can change the value of variables **declared** outside the function.

In the example below, `function y` changes the value of the a variable from 10 to 20.

```
1. var a = 10;
2.
3. function y() {
4.   a = 20;
5. }
6.
7. alert(a); //This message says "10"
8. y();
9. alert(a); //This message says "20"
```

16.2. determine the value of variables when a function runs twice

16.3. variables and parameters with the same name

In this example, the function **parameters** a and b have the same names as the variables declared on lines 1-2.

In this case, when a is changed inside the function, it changes the value of the **function parameter**.

So, on line 8 inside the function, the code `a = 10;` changes the parameter, but the value of variable a created on line 1 is **unchanged**.

Inside the function on line 7, `c = a + b;` calculates the sum of the function parameters.

Outside the function on line 13, `d = a + b;` calculates the sum of the variables declared on lines 1-2.

16.4. calling functions with variables

When you call a function you can pass in **literal values** and/or variables.

```
//Call alert with a literal string
alert('Hello');

//Call an alert with a variable
var msg = 'Hello';
alert(msg);
```

On line 12, the c variable and literal value `100` are passed to the function. When `lines 6-10|function x` runs, the a and b parameters are set to **3** (the value of c) and **100**.

When the value of a is changed inside the function, this does not change the value of the c variable that was passed into the function.

16.5. function variables

When the `var` keyword is used inside a function it creates a **new** variable.

For example, the code below creates 2 variables called a:

```
1. var a = 10;
2.
3. function x() {
4.   var a = 5;
5.   alert(a); //Shows the message "5"
6. }
7. alert(a); //Shows the message "10"
```

The `alert` inside the function displays the number **5**. The `alert` outside the function displays the number **10**.

17. Drawing rectangles

17.1. create rectangle objects

In an earlier lesson you drew rectangles by creating a path with 4 points. An easier way is to create a `Path.Rectangle` object.

The example code draws a rectangle with a top left corner at 50, 50 and bottom right corner at 300, 400.

For each task in this lesson you can see the answer lightly in the background when you click .

1. Create a `Path.Rectangle` object called `carPark`, which covers the **black** rectangle in the bottom left quarter.
2. Create a `Path.Rectangle` object called `lab` to draw the **brown** research lab below the **car park**.

17.2. draw several rectangles with a loop

Under the car park there are 3 research labs of equal size. A loop can help to draw the labs with less code.

The example has code that uses a loop to draw 8 rectangles.

1. Replace the code to draw the lab with the example loop code.
2. Modify the loop code to draw 3 rectangles.
3. Modify the location of the rectangles to cover the 3 labs.

In the example, the variables `tl` and `br` are short for **top left** and **bottom right** corner.

17.3. draw filled shapes

The bottom right section is the **S.C.R.I.P.T.** gardens. In the middle of the garden is a round lake.

The example code uses the `fillColor` property to draw a filled **purple** triangle.

1. In the bottom right, create a circle object called: `lake`
2. Use a stroke width of **2**.
3. Use a stroke color of **blue**.
4. Use a fill color of **lightBlue**.

17.4. practice filling shapes

1. Fill the **main office** with **pink**.
2. Fill the **plaza** with **lightGreen**.
3. Fill the **car park** with **darkGray**.
4. Fill the **labs** with **orange**.

17.5. practice drawing shapes

The **S.C.R.I.P.T.** physical training area is in the top right.

Draw the following shapes with a stroke width of 2:

1. Create a `Path.Rectangle` object called **fields** and draw the sports fields with a stroke color of **green** and fill color of **lightGreen**.
2. Create a 4 sided `Path.RegularPolygon` object called **pools** with a stroke color of **blue** and fill color of **lightBlue**.
3. Create a 4 sided `Path.RegularPolygon`object called **gym** and draw the gymnasium with a stroke color of **gold** and fill color of **yellow**.

18. Functions practice

18.1. complete the function

In this lesson you'll write your own functions.

S.C.R.I.P.T. is painting 3 of the computer labs.

The size of lab 1 is **8m x 5m**, lab 2 is **12m x 6m** and lab 3 is **8m x 7m**

The cost to hire a painter is $12.50 per meter^2.

1. Complete the function on lines 2-4.
2. Replace the `???` on line 8 to use the function to calculate the ceiling area of lab 3.
3. Replace the `???` on line 10.
4. Replace the `???` on line 11.

18.2. complete the function

The cost to paint the lab walls is $1 per meter^2 more than the ceiling.

1. Complete the `calculateWallArea` function on lines 7-9; include 2 calls to the `calculateArea` function.
2. Complete the `calculateCostToPaint` function on lines 12-14; include a call to the `calculateWallArea` and `calculateArea` functions.

18.3. update the function

Using functions makes it easier to change code in the future.

S.C.R.I.P.T. decided to paint a 4th lab with dimensions of **6m x 5m** and a height of **3.5m**. They also got discounted paint at $11 per m^2 for both ceiling and walls.

1. Update the code to include the cost of the 4th lab.
2. Update `calculateCostToPaint` to use the discounted paint cost.

18.4. complete the function

Several companies gave quotes to lay carpet in the 4 labs.

Company **A** charges **$40** per m^2.

Company **B** charges **$45** fixed costs plus **$38** per square meter.

Company **C** charges **$500** fixed cost plus **$35** per m^2.

1. Complete the `calculateCarpetCost` function on lines 7-9.
2. Complete line 20 by replacing `???`.

18.5. create a function

For your final task:

1. Write a function called `min` that returns the lowest value of 3 parameters `a`, `b` and `c`.
2. Use the `min` function to store the lowest carpet cost in a variable called: `minCost`
3. Display the lowest cost in the console.

19. Functions practice

19.1. complete `calculateArea`

S.C.R.I.P.T. is constructing a new 2 story building that contains 4 bathrooms. In this lesson you will write a function to calculate the number of tiles needed for each bathroom.

Bathroom no.1 has a length of 5m and width of 3m; bathroom 2 is 6.5m by 3.5m; **bathroom 3** is **9m** by **4m**; **bathroom 4** is **11m** by **4m**.

For task 1:

1. Replace `???` on lines 2-4 to complete a function to calculate the area of a rectangle.
2. On lines 8 & 9, calculate the area for bathrooms 3 and 4.

19.2. create `calculateTileCount`

In this task you'll calculate the number of square tiles required for the floor of each bathroom. The 2 smaller bathrooms will use 0.3m wide tiles; the 2 large bathrooms will use 0.45m wide tiles.

1. Create a function called `calculateTileCount` with 3 parameters in this order: `tileWidth`, `length`, `width`. The last 2 parameters are the length and width of the bathroom.
2. In that function, calculate and return the required tile count (without rounding); include a call to `calculateArea`.
3. Set the 4 **bathroom** variables to the tile counts instead of area.
4. Modify the `console.log` statement to print the tile counts to the nearest whole number using `toFixed`.

19.3. create `calculateTileCost`

In this task, you'll calculate the cost of tiles for each bathroom floor. The small tiles are $1.50 each (use for bathroom 1 and 2); the large tiles are $3.50 each (use for bathroom 3 and 4).

1. Create a 3rd function called `calculateTileCost` with 4 parameters in this order: `costPerTile`, `tileWidth`, `length`, `width`.
2. Calculate and return the tile cost; include a call to `calculateTileCount`.
3. Set the 4 **bathroom** variables to the total tile cost instead of count.
4. In the `console` message, put `$` in front of the costs and round the costs to the nearest cent.

19.4. create `calculateTilingCost`

In this task, you'll calculate the total cost to tile each bathroom.

Labour and other tiling supplies (fasteners, adhesives etc.) cost $18/$m^2$. Equipment allowance (diamond cutter, stone saw etc.) is $60 total per bathroom.

1. Create a 4th function called `calculateTilingCost` with 4 parameters in this order: `costPerTile` (cost of each tile), `tileWidth`, `length`, `width`.
2. Calculate and return the total cost to tile the bathrooms (including labour, supplies and equipment); include calls to `calculateTileCost` and `calculateArea`.
3. Use `calculateTilingCost` to set the 4 **bathroom** variables to the total tiling cost.

19.5. reuse the functions

The manager is considering using more expensive tiles ($2.00 each) for the 2 smaller bathrooms.

1. At the end of the code calculate the total combined cost of all 4 bathrooms with the cheap tile and display the cost in the console rounded to the nearest cent.
2. Calculate the cost of using $2.00 tiles for bathroom 1 and 2 and store in variables called: `bathroom1b` and `bathroom2b`
3. Calculate the combined cost of all 4 bathrooms with the expensive tile; display the cost in the console rounded to the nearest cent.

20. Review lessons 11-19

20.1. review functions that do calculations

The following example function returns the **absolute** value of a number.

```
1. function abs(a) {
2.   if (a < 0) {
3.     return a * -1;
4.   }
5.   return a;
6. }
7.
8. console.log( abs(-5) )
9. console.log( abs(15) )
```

Lines 2-3 return a positive value if a is negative. Line 5 returns a as is, if it is already positive.

Review Quiz Questions:

1.
2.
3.
4.

20.2. review functions that return boolean values

The following example has 2 functions that do exactly the same things in slightly different ways.

```
1. function isNegative(x) {
2.   if(x < 0) {
3.     return true;
4.   }
5.   return false;
6. }
7.
8. function isNegative(x) {
9.   return x < 0;
10. }
11.
12. console.log( isNegative(-5) )
13. console.log( isNegative(15) )
```

The function on lines 1-6 returns true if x is a negative number and false otherwise.

The function on lines 8-10 returns the result of `x < 0`, which is true if x is negative and false otherwise.

Review Quiz Questions:

1.
2.
3.
4.

20.3. review functions that check a character variable

The function on lines 1-3 returns true if a character is a digit, i.e. 0-9.

In this quiz you will study functions that check characters.

Review Quiz Questions:

1.
2.
3.
4.

20.4. review function return values

Review Quiz Questions:

1.
2.
3.
4.

20.5. review variable and parameters

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

Review Quiz Questions:

1.
2.
3.
4.

21. Intro to lessons 22-30

21.1. introduction to `Math.pow`

The code in the editor shows 2 ways to calculate 2^4. The 2nd way uses the JavaScript Math object's power function.

1. Change line 1 to calculate 2^{10} (**without** using `Math.pow`).
2. Change line 2 to calculate 2^{10} (**using** `Math.pow`).
3. For each task in this lesson, click when you are done.

21.2. introduction to `Math.abs` and `Math.sqrt`

On line 1 the Math object is used to calculate the square root of 64. On lines 2 & 3 the Math object is used to calculate the absolute values of -64 and 64.

1. Change line 1 to calculate the square root of **256**.
2. Change line 2 to calculate the absolute value of **-256**.
3. Change line 3 to calculate the absolute value of **256**.

21.3. introduction to **Math.PI**

The Math object also has some special properties such as `Math.PI` (i.e. 3.14...). `Math.PI` is used to do accurate calculations with circles.

For example, the code on line 2 calculates the volume of a cylinder (or drum) with a height of 1.5 meters and a radius of 0.5 meters.

1. Change line 1 to use the `pow` function to calculate `PI` squared.
2. Change line 2 to calculate the volume of a cylinder with a height of **2** meters and radius of **0.8** meters.

The formula for the volume of a cylinder is: **$V = \pi r^2 h$**
Where **π** is PI, **r** is radius and **h** is height.

21.4. introduction to `Math.round`

The `Math.round` function is used to round numbers to the nearest whole number. If the decimal fraction is .5 then it is rounded **up** (as shown in the example code).

1. Use `Math.round` to round the result of the calculation on line 1.
2. Use `Math.round` to round the result of the calculation on line 2.

21.5. introduction to `Math.random`

In this lesson you'll use random numbers to create several basic games.

1. Repeat the code on line 1 at least **5** times.
2. click at least twice and notice that the numbers are different each time.

22. Use Math object functions

22.1. use `Math` functions

#The JavaScript Math object performs various mathematical tasks.

The example code has 5 Math object examples. The 1st 3 are **functions** that do calculations, and the last 2 are **properties** with fixed values.

In the code editor replace the `???` to calculate the following:

1. The volume of **S.C.R.I.P.T.**'s 25m cubed shaped dive tank.
   Hint: `volume = width`^{3}
2. The area of the circular Code Avengers logo on the side of the main office building; it is 22 meters in diameter.
   Hint: `area = πr`^{2}
3. The width of the square shaped **S.C.R.I.P.T.** headquarters, which has a land area of 375000m^2.
   Hint: `area = width`^{2}

22.2. use `Math` functions

Modify the calculations to do the following:

1. The water in the dive tank is heated to 22 degrees, this cost $1 a year per 15m^3. What is the total heating cost each year?
2. The cost of paint for the Code Avengers logo is $2 per m^2. What is the total paint cost?
3. The total distance to run around the **S.C.R.I.P.T.** headquarters.
4. Change the variable names to describe the new calculations.

22.3. use `Math` functions

The Math object has 3 functions for rounding numbers: `round`, `floor` and `ceil`.

the example to see how the functions work.

Use each function once to do the following:

1. Round the heating cost **down** to the nearest **$**.
2. Round the paint cost **up** to nearest **$**.
3. Round the perimeter length to the **nearest m**.

22.4. round numbers with `toFixed`

`Math.round` rounds a number to exactly 0 decimal places.

The `toFixed` function rounds to 0 or more decimal places.

As shown in the example, `toFixed` is not part of the Math object, and is used differently to `round`.

Use **`toFixed`** to do the following:

1. Round the heating cost to 2 decimal places.
2. Round the paint cost to 2 decimal places.
3. Round the land perimeter to 0 decimal places (using `toFixed`).

22.5. use `toPrecision`

The `toPrecision` function rounds to 1 or more **significant figures** (sf).

E.g. `(1.2345).toPrecision(2)` equals `1.2`.

If the precision is less than the number of digits to the left of the decimal, the number is converted to scientific notation.

E.g. `(12345).toPrecision(2)` equals `1.2e+4` (i.e. 1.2x10^4). `Math.round` can be used to convert from scientific form to standard form, as shown in the example.

the example, then use **`toPrecision` and `Math.round`** to display the following in standard **decimal form** (e.g. 5.4):

1. Heating cost with 2 significant figures.
2. Paint cost with 2 sf.
3. Land perimeter with 3 sf.

23. Generate random numbers

23.1. random number from 0–1

The `Math` object is also used to get random numbers.

The `Math.random` function gets a random decimal number from **0** up to but not including **1**.

The highest possible random number is `0.9999...`

1. Copy and paste the code on line 1 so that 3 random numbers are displayed in the console.
2. the code at least twice to confirm that different numbers are shown each time.

23.2. random integers

Random numbers are used a lot in computer games, from dealing out cards to moving bots in a shoot-em-up.

The example code displays a random number from 0 to 99.

the example, then modify your code to show the following numbers in the console messages:

1. A spin on a **roulette** wheel: a random number from **0–37**.
2. Damage to health % from a bullet in a shoot-em-up: a random number from **0–70**.
3. Degrees an enemy turns when they hear you coming: a random number from **0–359**.

Round all numbers down using `Math.floor`. Show 0 if Math.random returns 0, and the highest number if Math.random returns `0.999...`

23.3. random integers in a range

The example code gets a random number from 5–15.

Modify your code to show the following:

1. A dice roll: a random number from **1–6**.
2. Damage to health % from a bullet: a random number from **30–70**.
3. Degrees an enemy turns: a random number from **-180 to 179**.

Round all numbers down using `Math.floor`. Show 0 if Math.random returns 0, and the highest number if Math.random returns `0.999...`

23.4. random multiples

In the example code, `Math.floor(51 * Math.random())` gets a random **integer** from 0–50.

This number is multiplied by 4 to get a random number from **0-200** that is a **multiple of 4**, i.e. 0, 4, 8, 12, 16 etc.

Modify your code to show the following:

1. Sum of **2 dice rolls**: add 2 random numbers from **1–6**.
2. Damage to health % from a bullet: a random **multiple of 5** from **0–100**.
3. Degrees an enemy turns: a random **multiple of 10** from **0–360**.

Round all numbers down using `Math.floor`. Show 0 if Math.random returns 0, and the highest number if Math.random returns `0.999...`

23.5. random workout

At **S.C.R.I.P.T.** we finish work with a team workout and optional swim. We use a computer program to randomly create a different workout each day.

For this task, write code to create a random workout by doing the following:

1. Store a random number from 10–30 in a variable called: `runTime`
2. Store a random multiple of 10 from 50–100 in a variable called: `situpCount`
3. Store 200 minus the number of situps in a variable called: `pushupCount`
4. Show all 3 values on separate lines in an **alert** message.

Round all numbers down using `Math.floor`. Show 0 if Math.random returns 0, and the highest number if Math.random returns `0.999...`

24. Draw the Pyramids of Giza

24.1. move 3 blocks

In this lesson you will build a model of the Great Pyramid of Giza using lots of small blocks.

For each task in this lesson you will see the answer lightly in the background when you click .

The code in the editor draws 3 **red** rectangular blocks in the top left.

Your task is to make the bricks **gold squares** (30px by 30px) and to move them to the bottom left.

1. Give the blocks a **gold** border and a **yellow** fill.
2. Move `brick1` to the bottom left.
3. Move `brick2` to right of `brick1`.
4. Move `brick3` on top of `brick1` and `brick2`.

24.2. use a function to draw blocks

Task 1 used 6 lines of code for each brick. It would take a lot of code to draw all 2.3 million blocks in the Great Pyramid of Giza!

However, most of the 6 lines of code is the same for all 3 bricks. Since the bricks are the same color and size, repeated code can be avoided by using a **function**.

The code in the editor uses a function to draw 3 red bricks.

Your task is the same as task 1:

1. Give the blocks a **gold** border and a **yellow** fill.
2. Make the blocks 30 by 30 pixels.
3. On line 12, draw brick 1 in the bottom left.
4. On line 13, draw brick 2 to right of brick 1.
5. On line 14, draw brick 3 on top of bricks 1 and 2.

24.3. use a loop to draw a row of blocks

In task 2 you used 1 line of code for each brick. This is better, but it would still take a long time to write code to draw an entire pyramid. By using loops, we can draw millions of bricks with just a few lines of code!

This code has a loop that draws a row of 12 bricks across the top of the screen.

Modify the code to do the following:

1. Draw a row of **16** bricks.
2. Give the blocks a **gold** border and a **yellow** fill.
3. Make the blocks 30 by **30 pixels**.
4. Change line 13 to move the row to the bottom of the screen.

24.4. draw several rows of blocks

The code in the editor has 2 loops to draw 2 rows of blocks.

1. Copy lines 16-18 and draw the 3rd row of blocks.
2. Draw the 4th row of blocks.

24.5. practice drawing shapes

The code in the editor draws a complete pyramid!

The function on lines 14-18 draws a row of **n** blocks starting at a specified **x** and **y** location.

The code on lines 21-27 uses a loop to draw all 16 rows of the pyramid from top to bottom.

1 bad thing about this code is that the number 30 appears several times. It is better to create a variable (as on line 1) and use that variable instead of the number 30. This will make it easy to change the size of the pyramid blocks.

1. Replace the number 30 on lines 4-27 with the variable `BLOCK_SIZE`
2. Replace the number 15 on line 24 with `BLOCK_SIZE / 2`
3. Change the value of the `BLOCK_SIZE` variable to 10.
4. Change the number of rows to **50**.

25. Fix Math object bugs

25.1. fix bugs in code

Each task in this lesson has a couple of mistakes to fix.

1. the code and read the error message in the console.
2. Fix the mistake on lines 1-2.
3. Fix the mistake on line 3-6.

25.2. fix bugs in code

This task has 2 more bugs to fix. You may need to use the reference for this one.

1. the code and read the error message in the console.
2. Fix the mistake on lines 9-10.
3. Fix the mistake on line 12-14.

25.3. fix bugs in code

This code asks the user to guess **heads** or **tails**, then tosses 2 coins. They get 2 points for guessing both correct, and 1 point if they get 1 correct.

However, there are 3 bugs!

1. Click to test the code.
2. Line 17 is supposed to check if the user guessed both correct; fix the mistake.
3. Fix the mistake on lines 19-20.
4. Fix the mistake on lines 21-22.

25.4. fix bugs in code

This code keeps tossing coins and adding points until the user guesses incorrectly. The code `while (true)` makes the loop repeat until the `break` statement on line 23 runs.

However, there are 3 bugs!

1. Click to test the code.
2. Fix the error on lines 12-13.
3. Fix the error on line 18.
4. Fix the error on line 20.

25.5. fix bugs in code

One problem with the code from task 4 is that the program doesn't work if the user clicks **cancel** when asked **Heads or tails**.

The program needs to exit the loop if the user clicks cancel. The code on lines 14-15 should do this, but it has 2 mistakes.

1. Click to test the code.
2. Fix the error on line 14.
3. Fix the error on lines 15.

26. Create a fortune teller

26.1. display a random message

In this lesson you will code a fortune teller. First you will use a random number to choose a message to show.

Replace the `???` to do the following:

1. On line 1 ask the user's name and store it in a variable called: `name`
2. On line 4 put a random number between 0 and 1 in a variable called: `r`
3. Replace the `???` on line 8 so that the message "I see your future clearly" is shown 20% of the time.

Note that on lines 7, 9 & 11, `msg += '...';` adds text onto a string variable and is a shortcut for `msg = msg + '...';`

26.2. predict years to retire

Let's create the next part of the fortune teller. At the end you will join them into 1 app.

In this task you'll predict the age that the user will **retire** at.

Remember that `Math.random()` returns a decimal number from 0 up to but not including 1. View the **reference** or review lesson 23 to see how to get random **integers**.

Replace the `???` to do the following:

1. On line 1, ask the user's age and store in a variable called: age
2. On line 2, store a random **integer** between 35 and 75 in the `retire` variable. If `Math.random()` returns 0 set `retire` to 35; if it's 0.999 set `retire` to 75. The `retire` variable stores the age the user will retire.
3. Calculate the number of years untill the user will retire and store it in a variable called: `yearsToRetire`
4. If the years until retirement is more than 1, show an alert that says the years till retirement.
5. Otherwise show an alert that says: "You'll retire soon."

26.3. predict a job

In this task you will predict the user's future as a computer programmer.

Replace the `???` on lines 5, 7 & 9 with the correct numbers.

The probability that each message is shown is as follows:

1. **15%**: start your own software company.
2. **25%**: work at Google, Facebook or Microsoft.
3. **55%**: quit learning to code.
4. **5%**: become a Code Avenger.

26.4. answer a question

In this code the user asks the fortune teller a question, and the fortune teller gives a random answer.

Replace the `???` with the correct messages, and keep the rest of the code the same.

The probability of each answer are as follows:

1. **30%**: Yes.
2. **25%**: No.
3. **20%**: Probably.
4. **15%**: Maybe.
5. **10%**: Possibly.

26.5. complete the app

For your final task do the following:

1. Copy and paste the code from tasks 1 to 4 in order into this task.
2. Remove `var` so that it's only used when each variable is **declared** (i.e. created or used for the 1st time).
3. **(Optional)** Use a loop so that the user can keep asking the fortune teller questions until they click cancel.
4. **(Optional)** Add additional predictions from the fortune teller, such as the amount of money the user will earn in their job.

27. Refactor graphics code

27.1. replace repeated code with a function

In this lesson you refactor code that draws a model of the 2500 year old Greek building: the **Parthenon**.

In this lesson you'll see the answer lightly in the background when you click .

The code on lines 10-30 draws the 3 steps at the base of the Pathenon. The repetition in this code is **bad** and can be avoided by using a function.

1. Complete the function on lines 2-4 so that it draws 1 step; the parameters `startX` and `startY` are the position of the top left corner of the rectangle; `endX` and `endY` are the bottom right.
2. Complete the last line of code to draw the top step.
3. Delete the bad code.

27.2. use a function to draw blocks

The code on lines 30-46 is a bad way to draw the pillars, because there are lots of repeated numbers. This makes the code hard to modify. For example, to change the width or height of each pillar would be painful!

A better way is to create a function that draws 1 pillar (see lines 17-22); the use that function several times inside a loop (see lines 25-27)

Complete the function on lines 17-22 by doing the following:

1. Replace the **0**s on line 19 with the correct numbers.
2. Replace the **0**s on line 21 with the correct numbers.
3. Delete the bad code.

27.3. use a loop to draw a row of blocks

There are actually 8 pillars on the front of the Parthenon, not 6.

In order to make code easy to understand and easy to modify, all numbers should be replaced with variables.

To start with we have created 2 variables on lines 16 & 17.

On line 22 the number 55 was replaced with `PILLAR_WIDTH + 5`

On line 29 the number 80 was replaced with `PILLAR_WIDTH + 30`

1. Use the `PILLAR_WIDTH` variable to replace **60** on line 24.
2. Use the `PILLAR_COUNT` variable on line 28.
3. Change the `PILLAR_COUNT` variable value to **8**.
4. Change the `PILLAR_WIDTH` variable value to **30**.

27.4. draw several rows of blocks

Several new variables have been created (on lines 11-19) and used to replace numbers scattered throughout the code.

The special variables on lines 11-19 are known as **constants**, because their value doesn't change while the program runs. In JavaScript and **other programming languages**, we normally use all capitals with _ between words in constant variable names.

Now that our code uses **constants** see how easy it is to change the size of our pillars:

1. Change the height of the slab on top of each pillar to **10**.
2. Change the height of each pillar to **190**.

27.5. practice drawing shapes

The Parthenon is almost complete!

Use the `drawTriangle` function on lines 56-65 to draw the 2 sections of the roof.

1. Click to see what the roof should look like.
2. On line 67, replace the number 50 with the correct values to draw the outer triangle of the roof.
3. On line 68, replace the number 50 with the correct values to draw the inner triangle of the roof.

28. Refactor a dice game called Chuck-A-Luck

28.1. create a function to avoid repeated code

In this lesson you'll practice writing **good code** as you build a game of chance called **chuck-a-luck**.

In this game, the player rolls 3 dice. On lines 3-5, the same code is repeated to roll each die.

Functions should be used to avoid repeated code, and make code easier to change in the future.

1. Starting on line 1, create a function called `getDiceRoll` that returns a random number between 1 and 6 using the calculation on lines 5-7.
2. Use the `getDiceRoll` function to set the values of the `roll1`, `roll2` and `roll3` variables on lines 5-7.

28.2. refactor an **if statement** to remove repeated code

In chuck-a-luck, the player guesses a number then rolls 3 dice. They win if their number is rolled on at least 1 of the 3 dice.

The code on lines 10-22 works correctly, however, there are several things we can do to make it simpler and easier to understand. Your task it to make it look lines 12-18 in the example code.

Complete the following:

1. On the line before the **if statement**, add the code: `var result = 'lose';`
2. Combine the 4 **if statement** branch conditions into a single condition that checks if 1 of the rolls matches the player's guess.
3. If 1 of the rolls is correct, set the `result` variable to: **win**
4. After the if statement display the result, as on lines 17-18 in the example. Your code must only inlcude 1 `alert` command.

28.3. refactor by adding a loop

In this version of chuck-a-luck, the player scores 10 points for every dice that matches their guess.

In this code, lines 12-22 count the number of dice that match the players's guess.

Now let's use a **for loop** to remove repetition and simplify lines 8-22. Using a loop is more flexible, since it makes it easy to change the number of dice rolls.

Look at lines 11-16 in the example for help. The example asks the user to guess heads or tails, and counts the number correct out of 4 coin tosses.

1. Use a **for loop** so that you only need 1 if statement and 1 call of the `getDiceRoll` function.
2. Replace `correct = correct + 1` with its shorthand.
3. Make sure your code only calls `getDiceRoll` **once**.

28.4. refactor by combining if statements

In some versions of chuck-a-luck, the player scores **10** points for getting **1** correct, **20** points for getting **2** correct and **100** points for getting **3** correct.

The code on lines 17-31 uses 4 separate if statement to set the scores. However, in this case it is better to use an `if, else if, else` statement.

1. For the `if` branch of the **if statement** give **10** points if the player got **1** correct.
2. For the 1st `else if` branch give **20** points if they got **2** correct.
3. For the 2nd `else if` branch give **100** points if they got **3** correct.
4. Use the `else` branch to give **-10** points if they got **0** correct.

28.5. code shortcuts

The editor has the complete code for the **chuck-a-luck** game.

The player keeps guessing a number and scoring points until they lose all their points.

1. Use the shortcut code on line 26.
2. Use the shortcut code on line 42.

29. Create a number guessing game

29.1. get a random number from **1-100**

In this lesson you'll create a **guess the random number game**.

To get started:

1. Store a random whole number from 0 to 99 inclusive in a variable called: `randomNumber`
2. Display the random number in the console.
3. For each task in this lesson click to try it out, then to see if your code passes.

The `Math.random` function returns a random decimal number from 0 up to but not including 1 (i.e. the largest number is 0.999...).

The `Math.floor` function called rounds a number **down** to the next integer.

29.2. ask the player to guess

While we are building the game, we display the answer in the console in order to make it easy to test the game.

For your next task:

1. Asks the player to guess the number and store the response in a variable called: `guess`
2. Keep looping until the user guesses correctly.
3. When they guess correctly, display an alert message that says: "Congratulations"

29.3. ending the game early

Do the following to allow the player to end the game early:

1. If the player clicks **cancel**, clicks **OK** without typing anything, or types something that is **not a number**, `break` out of the loop.
2. Only show the congratulations message if the player guesses the number correctly.

29.4. give hints

Now let's give the player hints when they guess incorrectly.

Do the following inside the loop:

1. If the player guesses **too low** display an `alert` that says: "higher"
2. If the guess is **too high** show a message that says: "lower"

29.5. 2 player game

Now make it a 2 player game, with players taking turns making guesses.

1. On the line before the `while` loop create a variable called `whosTurn` and set it equal to **2**.
2. Put `whosTurn = 3 - whosTurn;` as the 1st line inside the while loop (the line before the prompt); this code switches whos turn it is and makes player 1 go 1st.
3. Your code must include a `prompt` that says: "Player 1, guess a number" when it is player 1's turn and "Player 2, guess a number" when it is player 2's turn. The `prompt` command should only appear once in your code.
4. At the end show the message "Player 1 wins" or "Player 2 wins"

30. Review lessons 21-29

30.1. review rounding functions

Review Quiz Questions:

1.
2.
3.
4.

30.2. review Math object functions

Review Quiz Questions:

1.
2.
3.
4.

30.3. review the random function

Review Quiz Questions:

1. What is the highest possible value of a after this code runs?
2. What is the lowest possible value of a after this code runs?
3. What is the value of a if `Math.random` returns **0.862**?
4. What is the value of a if `Math.random` returns **0.118**?

30.4. review the random function

The `Math.random` function returns a random number from 0 up to but not including 1, i.e. the highest random number is `0.99999999...`

Review Quiz Questions:

1. What is the highest possible value of a after this code runs?
2. What is the lowest possible value of a after this code runs?
3. What is the value of a if `Math.random` returns **0.862**?
4. What is the value of a if `Math.random` returns **0.118**?

30.5. review Math object function names

Review Quiz Questions:

1. What's the correct name of the Math object function that **rounds up** to the nearest whole number?
2. What's the correct name of the Math object function that calculates the **square root** of a number?
3. What's the correct name of the Math object function that is used to calculate $\mathbf{2^{20}}$?
4. What's the correct name of the Math object function that calculates the **absolute value** of a number?

31. Intro to lessons 32-40

31.1. create a list with separate variables

In lessons 32-39 you will write code to track progress of students in a **S.C.R.I.P.T.** training course.

One way to store a list of names of students in a class is to use a string variable for each student as shown in the editor.

1. Store the name **Mike W** in a variable called: `name8`
2. Add the `name8` variable to the console statement that prints the class list.

31.2. create a list with an array

A much better way to store lists of data is using a special type of variable called an **array**.

In the next 10 lessons you'll learn all about **arrays**.

The code on line 1 creates an array (or list) of student names.

The code on line 6 prints all the names in the list in the console using the same format as the previous task. Notice that the code is much simpler than in the previous task!

1. Add **Adam W** to the end of the array.
2. Click to run the code.
3. Click when you are done.

31.3. print single items from an array

The code `names[0]` gets the 1st item in the array.

The code `names[1]` gets the 2nd item in the array.

Remember, that in computer science we often start counting at 0. So to get the 9th item in the array the code is: `name[8]`

1. Add "Holly O" to the end of the array.
2. Print her name at the end of the female list.

31.4. manually sort a list

S.C.R.I.P.T. teachers need alphabetically sorted lists of their students.

1. Add **Chris B** to the end of the array.
2. Insert his name in the correct place in the sorted list.

31.5. print sorted list

Manually sorting a large list of names takes a long time!

Thankfully arrays have a `sort` function that sorts arrays for you.

Line 8 prints an **unsorted** list of names.

Line 11 in prints a **sorted** list.

1. Add your own name to the end of the array.
2. Click to display the lists.
3. Click when you're done.

32. Sorting and printing lists of strings

32.1. sort an array and print each item on a separate line

Each **S.C.R.I.P.T.** class is sorted alphabetically and students are put in teams of 4.

The 1st 4 students are **Team 1**, the next 4 are **Team 2** and the last 4 are in **Team 3**.

The code in the editor diplays team 1 and 2, but the list is not sorted.

1. On line 7 use the array `sort` function to sort the list of students.
2. On lines 17-20 add code to print the 4 students in **Team 3**.

32.2. use a loop to print items

The power of arrays comes when you combine them with loops!

It is better to use loop to print all the array items on separate lines.

The code in the editor uses a loop to print the 1st 10 names in the sorted list.

1. Change line 14 to start at student number **11**.
2. Change line 14 to print all students from number **11-25**.

32.3. add team numbers to the list

When the list is printed, each name should be preceded by its team number.

1. Change line 16 to print all the students.
2. Replace the `???` on line 17; the `teamNo` should be equal to `i` divided by the team size plus 1, **rounded down**.
3. Change line 18 to print student names using the following format:
   `Team 1 Aaron M`

32.4. get the length of an array

The code on lines 12-17 allows users to add more students to the class list, and click cancel when they are done.

Line 15 uses the array `push` function to add students to the end of the list.

Line 21 uses the array `length` property to print the number of items in the list. When an item is added to the list, the array `length` increases by 1.

1. On line 25, replace `???` to use the `length` property to print the **last item** in the list.
2. On line 26, replace `???` to use the `length` property to print the **2nd last item** in the list.
3. Click and add a few students to the list to check that your code works correctly.

32.5. use length to print an array of unknown length

For some class activites students work in teams of 8.

1. Change the code to make teams of 8.
2. Use the `length` property to make the loop print all the students in the list (including those added via the prompt box).
3. Click and add a few students to the list to check that your code works correctly.

Note that making teams this way does not work well in all cases. If there were 34 people it would create 4 groups of 8 and 1 group of 2!

As an optional challenge, see if you can work how to to make the teams as even as possible, i.e. if there are 34 people have 4 groups of 7 and 1 group of 6.

33. Practice creating arrays

33.1. use push to add to an array

The code on line 1 creates a list of 4 numbers.

This can also be done by creating an empty array (see line 1 of the example code), then adding items using the array **push** function (see lines 3-5 of the example).

1. On line 3 in the code create an empty array called `array2` (use the same way as the example).
2. Use the push function to add the numbers `72, 15, 22, 99` to `array2`.
3. Use the code `console.log(array2)` to display the array contents in the console.

33.2. modify items in an array

Line 4 in the example changes the 3rd item in the `exampleArray` from 3 to 30 using the following code:

```
exampleArray[2] = 30;
```

1. Use the same approach to change the value of the **2nd** item in `array` from 15 to **150**.
2. Change the value of the **3rd** item from 22 to **220**.
3. Then print the array contents on a single line in the console.

33.3. modify undefined items in an array

What happens when you try to print or modify array items that **don't exist**?

1. Use a console message to print the value of the 5th item in the array.
2. On the next line set the value of the 10th item in the array to 44.
3. Use a console message to print the entire contents of the array.

33.4. use a loop to print values in an array on separate lines

`undefined` is a special JavaScript value for variables that don't exist or don't have a value yet.

For example, line 1 declares a variable called a; since this variable was not assigned a value, it's value is `undefined`.

1. On lines 2 & 8, replace the `???` to print values.
2. On line 7, replace the `???` to print the value of all 10 items in the array on separate lines in the console. Use the `length` property to set when the loop should stop.

33.5. modifying items in an array by looping

Loops can also be used to setup arrays.

Line 2 creates an empty array. Lines 4-6 are supposed to add the even numbers from 2 to 100 to the array.

The loop on lines 11-13 is supposed to double each value in the array.

1. On line 4, replace `???` with the correct number.
2. On line 5, replace `???` with the correct calculation to store the even numbers from 2 to 100 in the array.
3. On line 12, replace the `???` to use the shortcut form `*=` to double the value of each item in the array.

34. Refactor arrays with graphics

34.1. move 3 blocks

In this lesson you'll create a game of peg solitaire.

The code in the editor completes step 1, which is to draw 7 pegs in a row.

Refactor the code to use a loop to draw the 7 pegs.

1. Create a `for` loop that uses `i` to count from 0 to 6.
2. Put the code to create 1 circle inside the for loop.
3. Use the formula `var x = 45 + i * ???;` to set the value of x for each circle; replace `???` with the correct value.
4. Delete all the extra code.

34.2. replace numbers with constants

Now refactor by replacing the numbers in the code with constant variables, to make the code easier to modify in the future.

1. Before the loop, declare a constant variable called `RADIUS` with a value of **25**.
2. Replace the number 25 in the code with the constant variable.
3. Replace the number 50 with: `RADIUS * 2`
4. Before the loop, declare a constant variable called `ORIGIN` with a value of **45**.
5. Replace the number 45 in the code with the constant variable.

34.3. replace strings with constants

The code on lines 13-19 changes the color of the 1st 3 pegs to **blue** and the last 3 to **red**.

To make it easy to change the colors of the pegs in the future the color values should be replaced by constant variables.

1. Create a constant variable called `COLOR1` and set it to **blue**, a constant called `COLOR2` set to **red** and a constant called `EMPTY` set to **white**.
2. Replace the fill color values with the constant variables.

34.4. use a loop to set initial colors

The code on lines 20-26 can be refactored by setting the fill color of each peg inside the `for` loop.

1. Modify lines 8-9 so that it contains a list of the start colors for each peg.
2. On line 16, add a line of code that sets the fill color using the values in the `START_COLORS` array.
3. Delete lines 20-26.

34.5. draw several rows of blocks

The editor contains the complete code for the game. The player clicks on a peg to move it.

Using constant variables makes it easy to modify the game!

1. Read through the code and see how much you understand.
2. Change the radius to 35.
3. Change the origin to 40.
4. Change the `COLOR1` to **orange** and `COLOR2` to **purple**.

The level 3 course teaches how to add buttons to restart a game

35. Arrays practice

35.1. determine the value elements in an array

Computer programmers call the items in an array **elements**.

Remember that `push` is used to add **elements** on the end of an array.

35.2. undefined array elements

If a variable is not assigned a value, it has the special value `undefined`.

35.3. array creation practice

35.4. changing arrays with a function

Notice that inside the function the array is called `y`.

35.5. adding to an array with a function

You might want to use a pen and paper for this one.

36. Functions with an array as a parameter

36.1. calculate sum and mean of a list of variables

S.C.R.I.P.T. students have weekly tests. In this lesson you will write code to get statistics about the class test scores.

Lines 1-5 are arrays that contain the test scores of the students in 2 classes. The function on line 11-13 calculates the **mean** (or average) of all the students in the class using the **sum** function on lines 7-9.

1. On line 8, complete the `sum` function to return the sum of the 10 parameters named `a` to `j`.
2. On line 12, complete the `mean` function by replacing `???`.
3. On line 17, Use the `mean` function to calculate and store the average test score in class 2 in a variable called `mean2`.

36.2. create a function with an array as a parameter

The function in task 1 calculates the mean of a class with exactly 10 students.

Using a loop and an array as a parameter, we can create a function that calculates the mean of a class with **any** number of students.

Complete the code by doing the following:

1. On line 9, replace the `???` to loop through elements in the array; use `i` as a loop counter.
2. On line 10, replace the `???` to **sum** the elements in the array.
3. On line 16, complete the `mean` function by replacing `???`.
4. On line 20, use the `mean` function on to calculate the class 2 average.

36.3. sort an array

In order to calculate the class **median** (the middle student) or **range** (highest minus lowest) you need to sort the test scores from highest to lowest.

1. On line 8, sort the class 2 list.
2. On line 11, print the class 2 list in the console.
3. Click and study the output carefully. It is incorrect!

The next task explain how to correctly **sort** numbers.

36.4. sort an array of numbers

The `sort` function converts array items to strings and basically sorts them **alphabetically**, by comparing character positions in the **ASCII** table.

If you sort an array containing the numbers 1 to 10 you get: `1, 10, 2, 3, 4, 5, 6, 7, 8, 9`

10 comes before **2** because **1** is before **2** in the **ASCII** table.

To sort a list **numerically** you need to create a **compare function** (see lines 7-9) that tells the `sort` function how to sort the data (see line 19).

For example, the following compare function (lines 7-9) sorts an array from **lowest** to **highest**:

```
function compareNumbers(a, b) {
  return a - b;
}
```

Compare functions take 2 parameters. They return a negative value if `a` comes before `b` in the sorted array; return 0 if `a` and `b` are equal and a positive number if `a` goes after `b`.

The following code uses the `compareNumbers` function is used to sort the scores for `class1` from lowest to highest:

```
class1.sort(compareNumbers);
```

1. Complete line 12 to create a compare function to sort from **highest** to **lowest**.
2. Complete line 20 to sort `class2` from **highest** to **lowest**.

36.5. calculate the range of an array

The **range** of a list of numbers is the highest minus the lowest number.

The function on lines 11-14 returns the lowest number in an array of numbers by sorting the array from lowest to highest.

For this task complete the `max` function on lines 16-19.

1. On line 17, use the `compareNumbers` function to sort the array.
2. On line 18, return the highest value in the sorted array.
3. Complete the `range` function on lines 21-23 by calling the `min` and `max` functions.

In the next lesson you'll calculate the **median** of a list of numbers.

37. Using the modulus operator

37.1. check if a number is even or odd

To calculate the **median** the 1st step is to determine if there is an even or odd number of elements in the array.

One way to check if a number is odd is to divide it by 2 and see if the result is a decimal number by rounding it.

37.2. **modulus** operator

Another way to check if a number is even is using the **modulus** operator: `%`.

This operator finds the remainder of the division of one number by another.

For example, `x = 10 % 3` sets `x` to **1**, since `10 / 3` is 3 remainder 1. `y = 6 % 2` sets `y` to **0** since `6 / 2` is 3 remainder 0.

37.3. functions that check even and odd

The example functions `x` and `y` check if a number is odd or even.

You need to work out which function checks for even and which checks for odd numbers.

37.4. using **modulus** to calculate time

Modulus is needed to calculate the new time after a period of time passes.

For example, if the time is **1400** (in 24hr time) and 30 hours pass, the new time is `(14 + 30) % 24` which equals **20**.

37.5. using **modulus** to covert times

Modulus is also used to convert a time in seconds to hours, minutes, seconds.

38. Fix array code

38.1. fix bugs in the code

In this lesson each task has a couple of bugs to fix.

The code contains a broken function called `median` that is supposed to get the middle number in a sorted list.

If there are an even number of items in a list, the median is the average of the 2 middle items. For example, the median of `1, 2, 3, 4` is **2.5**.

Read the comments and fix the bugs in the `median` function.

1. Click to test the code.
2. Fix the bug on lines 3-9.
3. Fix the bug on lines 12-13.

38.2. fix bugs in the code

This code is supposed to allow a teacher to enter a list of test scores for their class.

However, there are a couple of bugs.

1. Click to test the code.
2. Fix the bug on line 2.
3. Fix the bug on line 8.

38.3. fix bugs in code

In the next 3 tasks, you'll fix code that draws a graph of the scores in a class.

The `drawAxis` function is supposed to draw the axis for the graph.

However, there are 2 mistakes.

1. Click to test the code.
2. Fix the mistake on lines 2-3.
3. Fix the mistake on lines 4-6.

38.4. fix bugs in code

The `drawBars` function draws the bars on the chart.

Both the `drawBars` and `drawAxis` functions use the constants on lines 1-5 to set the size and location of the chart.

Change the constants so that the chart is drawn in the same position as the solution in the background.

1. Click to test the code.
2. Change one of the constants on lines 1-2 to move the chart to the correct position.
3. Change one of the constants on lines 3-4 to make the chart the correct size.
4. Change the constant on line 5.

38.5. fix bugs in code

The final code draws black outlines around the bars and the bar colors alternate **blue** and **red**.

However, there are 3 mistakes!

1. Click to test the code.
2. Fix the mistake on line 33.
3. Fix 2 mistakes on lines 34-36.

39. Create a pig latin translator

39.1. check if a letter is a vowel

In this lesson you'll write **functions** to translate sentences to and from **Pig Latin**.

A word translates to Pig Latin by moving the characters before the 1st vowel to the end of the word, and adding -ay on the end.

E.g. "code avengers" is "ode-cay avengers-ay".

1. On line 1 define a function called `isVowel` with a single parameter called `c`, which will be a single character.
2. On line 2 convert `c` to lowercase with the `toLowerCase` string function.
3. Make the function return **true** if `c` is a vowel (a, e, i, o, u) or **false** otherwise. Return **false** for the letter **y**, which is a special case that is covered in task 5.
4. For each task in this lesson click to test your code, then to see if it passes.

39.2. write a function to check if a word is all letters

To keep things simple, you will not translate words with numbers, hyphens and apostrophes etc.

The function `isTranslatable` (lines 9-27) should check each letter in a word, and return **true** only if all characters in the word are letters and there is **at least 1 vowel**.

1. On lines 16-17, if `c` is a vowel increase the value of `vowelCount` by 1.
2. On line 21, check if the character is **NOT** a letter.
3. On line 26, replace the `???`.

39.3. write a function that translates a word

To translate to pig latin, add **-ay** to the end of words beginning with vowels.

E.g. **avengers** translates to **avengers-ay**.

For other words move the initial consonants to the end of the word and add **ay**.

E.g. **code** becomes **ode-cay** and **programming** becomes **ogramming-pray**

The `translateWord` function should return the translated form of translatable words and return untranslatable words unchanged.

1. You can click the arrow in the editor by the word **function** to hide the code for `isVowel` and `isTranslatable`
2. On line 30, check if `w` is a translatable word.
3. On line 34, check if the **i**th character in `w` is a vowel.
4. On lines 39-40, use the `substring` method to return the pig latin form of the word.

39.4. write a function that translates a sentence to pig latin

On lines 45-54, complete the function to translate a sentence into Pig Latin.

Line 47 uses the `split` function to break a sentence into an array of words.

Line 53 uses the `join` function to join the array of translated words back together.

1. Complete line 49 so that it loops through all the words in the **words** array.
2. Complete line 50 so that it translates each word and uses `push` to add the translated word to the end of the `results` array.

To keep things simple we are assuming sentences don't contain punctuation or capital letters.

39.5. write a function that translates a sentence from pig latin

The `untranslateSentence` function (lines 59-68) is almost the same as the `translateSentence` function, except it calls `untranslateWord`.

The `untranslateWord` function (lines 49-57) should translate words from Pig Latin to English.

E.g. **ode-cay** becomes **code**.

Words that don't have a hyphen `-` in them are returned as is.

Line 50 splits a word containing a hyphen into an array with 2 elements, the part before the hypen and the part after the hyphen.

1. On line 52, replace the `???`.
2. On line 56, replace the `???`.
3. **(Optional)** The letter **Y** should be treated like a consonant if it is the 1st letter of the word, and vowel in any other position.

40. Review lessons 31-39

40.1. review creating, accessing and modifying arrays

Review Quiz Questions:

1.
2.
3.
4.

40.2. review using loops to add to arrays

Review Quiz Questions:

1.
2.
3.
4.

40.3. review the modulus operator

Review Quiz Questions:

1.
2.
3.
4.

40.4. review modulus functions

Review Quiz Questions:

1.
2.
3.
4.

40.5. review array modification

Review Quiz Questions:

1.
2.
3.
4.