# Exercises: Functions and Arrow Functions

Problems for exercises and homework for the [“JavaScript Fundamentals” course @ SoftUni](https://softuni.bg/courses/javascript-fundamentals). Submit your solutions in the SoftUni judge system at <https://judge.softuni.bg/Contests/310/>.

## Inside Volume

Write a JS function that determines whether a point is inside the volume, defined by the box, shown on the right.

The **input** comes as an array of numbers. Each set of 3 elements are the x, y and z coordinates of a point.

The **output** should be printed to the console on a new line for each point. Print inside if the point lies inside the volume and outisde otherwise.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| [8, 20, 22] | outside |
| [13.1, 50, 31.5,  50, 80, 50,  -5, 18, 43] | inside  inside  outside |

### Hints

This task is very similar to previous assignments where a point might lie inside an area in 2D space, with just an extra dimension added. If we look at a classic conditional statement, which checks whether a point is inside a rectangle:



It checks whether a coordinate is greater than the minimum and at the same time less than the maximum bounding value for both axes (x and y). All we have to do is to include an additional check for a coordinate to be within the vertical limits of the volume (z-axis):



We can then wrap this whole statement in a function and as we process each set of coordinates, pass them to see if they are inside the volume and print the correct message to the console. Since the volume is the same every time, we can hardcode the values, but it’s generally good practice to pass them as function arguments, so that the function may work with any arbitrary volume. Later in the course we’ll learn how to shorten this with the use of objects.



We can extract the sets of coordinates with a loop that skips 3 elements at a time and assigns them to temporary variables:



We know from the problem description that the input array will contain sets to three coordinates. Starting at 0, the current element (denoted by index ***i*** inside the loop) is the x-coordinate, the element after the current (***i + 1***) is the y-coordinate, and the element two indices after the current (***i + 2***) is the z-coordinate. At the end of the cycle, the index is increased by 3 and we can obtain the coordinates of the next point, using the same arithmetic (instead of 0, 1 and 2 we will get 3, 4 and 5) and so on, until there are no more elements in the array. The three coordinates are passed into our function and we get a Boolean value as a result. If it’s true, we print inside for the current point and otherwise we print outside.

The solution may now be submitted to the judge system at <https://judge.softuni.bg/Contests/310>

## Road Radar

Write a JS function that determines whether a driver is within the speed limit. You will receive his speed and the area where he’s driving. Each area has a different limit: on the **motorway** the limit is **130** km/h, on the **interstate** the limit is **90**, inside a **city** the limit is **50** and within a **residential** area the limit is **20** km/h. If the driver is within the limits, your function prints nothing. If he’s over the limit however, your function prints the severity of the infraction. For speeds up to **20** km/hover the limit, he’s speeding; for speeds up to **40** over the limit, the infraction is **excessive speeding** and for anything else, **reckless driving**.

The **input** comes as an array of elements. The first element is the current speed (as number), the second element is the area where the vehicle is driving.

The **output** should be printed to the console. Note in certain cases there will be no output.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| [40, city] |  |
| [21, residential] | speeding |
| [120, interstate] | excessive speeding |
| [200, motorway] | reckless driving |

### Hints

We can divide the task in two functions – one that determines what the current speed limit is, depending on zone, and another which tells us if an infraction is being made, depending on current speed and current limit. Determining the limit is achieved with a **switch** statement on the input:



This function takes a string as an argument and returns a number, depending on what that string is. We can take this directly from the input, pass it to this function and save the return value in a variable. In our second function, we pass the current speed and the limit, which we just saved.



We calculate the difference between the current speed and the limit – if it’s negative or zero, this means the driver is within the rules and we return false, and in any other case, return the infraction as a string and store the result of the operation in a variable.



We can use the fact that JavaScript functions can return different data types and directly use the result we stored in a conditional statement – if it’s **false** (no infraction), do nothing, if it’s **truthy** (non-empty string in this case), print the value store in the variable.

## Template format

Write a JS program that receives data about a quiz and prints it formatted as an XML document. The data comes as pairs of question-answer entries. The format of the document should be as follows:

|  |
| --- |
| **XML** |
| <?xml version="1.0" encoding="UTF-8"?>  <quiz>  <question>  {question text}  </question>  <answer>  {answer text}  </answer>  </quiz> |

The **input** comes as an array of string elements.

The **output** should be printed on the console.

### Examples

|  |
| --- |
| **Input** |
| ["Who was the forty-second president of the U.S.A.?",  "William Jefferson Clinton"] |
| **Output** |
| <?xml version="1.0" encoding="UTF-8"?>  <quiz>  <question>  Who was the forty-second president of the U.S.A.?  </question>  <answer>  William Jefferson Clinton  </answer>  </quiz> |

|  |
| --- |
| **Input** |
| ["Dry ice is a frozen form of which gas?",  "Carbon Dioxide",  "What is the brightest star in the night sky?",  "Sirius"] |
| **Output** |
| <?xml version="1.0" encoding="UTF-8"?>  <quiz>  <question>  Dry ice is a frozen form of which gas?  </question>  <answer>  Carbon Dioxide  </answer>  <question>  What is the brightest star in the night sky?  </question>  <answer>  Sirius  </answer>  </quiz> |

## Cooking by Numbers

Write a JS program that receives a number and a list of five operations. Perform the operations in sequence by starting with the input number and using the result of every operation as starting point for the next. Print the result of every operation in order. The operations can be one of the following:

* chop – divide the number by two
* dice – square root of number
* spice – add 1 to number
* bake – multiply number by 3
* fillet – subtract 20% from number

The **input** comes as an array of 6 string elements. The first element is your starting point and must be parsed to a number. The remaining 5 elements are the names of operations to be performed.

The **output** should be printed on the console.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| [32, chop, chop, chop, chop, chop] | 16 8 4 2 1 |

|  |  |
| --- | --- |
| **Input** | **Output** |
| [9, dice, spice, chop, bake, fillet] | 3  4  2  6  4.8 |

## Modify Average

Write a JS program that modifies a number until the average value of all of its digits is **higher than 5**. In order to modify the number, your program should append a **9** to the end of the number, when the average value of all of its digits is **higher than 5** the program should stop appending. If the number’s average value of all of its digits is already **higher than 5**, no appending should be done.

The **input** is a single number.

The **output** should consist of a single number - the final modified number which has an average value of all of its digits **higher than 5**. The **output** should be printed on the console.

### Constraints

* **The input number will consist of no more than 6 digits.**
* **The input will be a valid number (there will be no leading zeroes).**

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 101 | 1019999 |
| 5835 | 5835 |

## Validity Checker

Write a JS program that receives two points in the format **[x1, y1, x2, y2]** and checks if the distances between each point and the start of the cartesian coordinate system (0, 0) and between the points themselves is **valid**. A distance between two points is considered **valid**, if it is an **integer value**. In case a distance is valid write "**{x1, y1} to {x2, y2} is valid"**, in case the distance is invalid write **"{x1, y1} to {x2, y2} is invalid"**.

The order of comparisons should always be first **{x1, y1}** to **{0, 0}**, then **{x2, y2}** to **{0, 0}** and finally **{x1, y1}** to **{x2, y2}**.

The **input** consists of two points given as an array of numbers.

For each comparison print on the **output** either "**{x1, y1} to {x2, y2} is valid"** if the distance between them is valid, or **"{x1, y1} to {x2, y2} is invalid"**- if it’s invalid.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| [3, 0, 0, 4] | {3, 0} to {0, 0} is valid  {0, 4} to {0, 0} is valid  {3, 0} to {0, 4} is valid |
| [2, 1, 1, 1] | {2, 1} to {0, 0} is invalid  {1, 1} to {0, 0} is invalid  {2, 1} to {1, 1} is valid |

## Treasure Locator

You will be given a series of coordinates, leading to a buried treasure. Use the map to the right to write a program that locates on which island it is. After you find where all the treasure chests are, compose a list and print it on the console. If a chest is not on any of the islands, print “On the bottom of the ocean” to inform your treasure-hunting team to bring diving gear. If the location is on the shore (border) of the island, it’s still considered to lie inside.

The **input** comes as an array with a variable number of elements that are numbers. Each pair is the coordinates to a buried treasure chest.

The **output** is a list of the locations of every treasure chest, either the name of an island or “On the bottom of the ocean”, printed on the console.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| [4, 2, 1.5, 6.5, 1, 3] | On the bottom of the ocean  Tonga  Tuvalu |
| [6, 4] | Samoa |

## Trip Length

You will be given the coordinates of 3 points on a 2D plane. Write a program that finds the two shortest segments that connect them (without going back to the starting point). When determining the listing order, use the order with the lowest numerical value (see the figure in the hints for more information).

The **input** comes as an array of 6 numbers. The order is **[x1, y1, x2, y2, x3, y3]**.

The **output** is the return value of your program as a string, representing the order in which the three points must be visited and the final distance between them. See the examples for more info.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| [0, 0, 2, 0, 4, 0] | 1->2->3: 4 |

|  |  |
| --- | --- |
| **Input** | **Output** |
| [5, 1, 1, 1, 5, 4] | 2->1->3: 7 |

|  |  |
| --- | --- |
| **Input** | **Output** |
| [-1, -2, 3.5, 0, 0, 2] | 1->3->2: 8.154234499766936 |

### Hints

You can find the horizontal and vertical offset between two points by calculating the difference between their coordinates. Use Pythagoras’ theorem to find the distance.

If more than one shortest paths exist, choose the one with lowest numerical value. For instance, in the figure on the right, 1🡪2🡪3 is the same distance as 3🡪2🡪1, but we chose to start at 1, since it’s lower than 3. When choosing the second point, we encounter the same issue – 1🡪3🡪2 would be the same as 1🡪2🡪3, but we chose to visit 2 first, because it’s lower than 3.