# Homework: Functions

Tasks with **\*** are considered hard or involve something we still haven't mentioned in the lectures. Try to solve them nevertheless. Do not worry about constraints how big the numbers can be, length of strings and so on. As we know JavaScript handles automatically the type and length of values. All tasks are submitted to GitHub. You can use HackerRank to check your solutions but the final tasks should be in GitHub.

## HackerRank in a String!

Follow the link for full details.

<https://www.hackerrank.com/challenges/hackerrank-in-a-string>

## Sock Merchant

Follow the link for full details.

<https://www.hackerrank.com/challenges/sock-merchant>

## Counting Valleys

Follow the link for full details.

<https://www.hackerrank.com/challenges/counting-valleys>

## Road Radar

Write a 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 in two rows. On the first row you will receive the current speed as a string and needs to be parsed as a number. On the second row you will be given the second element which is the area.

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

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 40  city |  |
| 30  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.

## Cooking by Numbers

Write a 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 in 2 lines. On the first line you will receive your starting point and must be parsed to a number. On the second line you will receive 5 commands separated by “, “ each one will be the name of the operation 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 |

## 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 a string of variable number of elements separated by “, “ that must be parsed to 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 a string with 6 elements separated by “, “ that must be parsed to 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.

## \*DNA Helix

Write a program that prints a DNA helix with length, specified by the user. The helix has a repeating structure, but the symbol in the chain follows the sequence ATCGTTAGGG. See the examples for more information.

The **input** comes as a single string element that must be parsed to a number. It represents the length of the required helix.

The **output** is the completed structure, printed on the console.

### Examples

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Input** | **Output** |  | **Input** | **Output** |
| 4 | \*\*AT\*\* \*C--G\* T----T \*A--G\* | 10 | \*\*AT\*\* \*C--G\* T----T \*A--G\* \*\*GG\*\* \*A--T\* C----G \*T--T\* \*\*AG\*\* \*G--G\* |