Exercises: Advanced Functions

1. Sort Array

Write a function that **sorts an array** with **numeric** values in **ascending** or **descending** order, depending on an **argument** that is passed to it.

You will receive a **numeric array** and a **string** as arguments to the first function in your code.

- If the second argument is **`asc`**, the array should be sorted in **ascending order** (smallest values first).
- If it is **`desc`**, the array should be sorted in **descending order** (largest first).

Input

You will receive a **numeric array** and a **string** as input parameters.

Output

The output should be the **sorted array**.

Examples

Input	Output
`[14, 7, 17, 6, 8], 'asc'`	`[ 6, 7, 8, 14, 17 ]`
`[14, 7, 17, 6, 8], 'desc'`	`[ 17, 14, 8, 7, 6 ]`

What to submit?

Function Signature: function main(array, order)

2. Argument Info

Write a function that displays **information** about the **arguments** which are passed to it (**type** and **value**) and a **summary** about the number of each type in the following format:

`"{argument type}: {argument value}"`

Print **each** argument description on a **new line**. At the end print a **tally** with counts for each type in **descending order**, each on a **new line** in the following format:

`"{type} = {count}`"

If two types have the **same count**, use **order of appearance**.

Do **NOT** print anything for types that do not appear in the list of arguments.

Input

You will receive a series of arguments **passed** to your function.

Output

Print on the **console** the **type** and **value** of each argument passed into your function.

Example

Input
`'cat', 42, function () { console.log('Hello world!'); }`
Output

`string: cat`
`number: 42`
`function: function () { console.log('Hello world!'); }`
`string = 1`
`number = 1`
`function = 1`

What to submit?

Function Signature: function main(arguments)

3. Functional Sum

Write a function that **adds** a number passed to it to an **internal sum of the outer function** and **logs the current internal sum** then returns **itself**, so it can be **chained** in a functional manner.

Input

Your function needs to take one **numeric argument**.

Output

Your function needs to have a state for internal sum and another inner function that **returns** itself after modifying the state of the internal sum.

Example

Input	Output
`add(1)`	1
`add(1)(6)(-3)`	1 7 4

What to submit?

Function Signature: function main(value)

4. Personal BMI

A wellness clinic has contacted you with an offer - they want you to write a program that composes **patient charts** and performs some preliminary evaluation of their condition. The data comes in the form of **several arguments**, describing a person - their **`name`**, **`age`**, **`weight`** in kilograms and **`height`** in centimeters. Your program must compose this information into an **`object`** and **`return`** it for further processing.

The patient chart object must contain the following properties:

- **`name`**
- **`personalInfo`**, which is an object holding their **`age`**, **`weight`** and **`height`** as properties
- **`BMI`** - body mass index. You can find information about how to calculate it here: https://en.wikipedia.org/wiki/Body_mass_index
- **`Status`**

The status is one of the following:

- **`underweight`**, for BMI less than 18.5;
- **`normal`**, for BMI less than 25;
- **`overweight`**, for BMI less than 30;
- **`obese`**, for BMI 30 or more;

Once the BMI and status are calculated, you can make a recommendation. If the patient is obese, add an additional property called recommendation and set it to "**`admission required`**".

Input

Your function needs to take four arguments - **`name`**, **`age`**, **`weight`** and **`height`**

Output

Your function needs to **return** an **object with properties** as described earlier. All numeric values should be **rounded** to the nearest whole number. All fields should be named **exactly as described** (their order is not important).
Look at the sample output for more information.

Input	Output

`"Peter", 29, 75, 182`	`{`
	`  name: 'Peter',`
	`  personalInfo: { age: 29, weight: 75, height: 182 },`
	`  BMI: 23,`
	`  status: 'normal'`
	`}`

`"Honey Boo Boo", 9, 57, 137`	`{`
	`  name: 'Honey Boo Boo',`
	`  personalInfo: { age: 9, weight: 57, height: 137 },`

	BMI: 30, status: 'obese', recommendation: 'admission required' }

What to submit?

Function Signature: function main(name, age, weight, height)

5. Vector Math

Write several functions for performing **calculations** with **vectors** in 2D space $\vec{a} = (x, y)$ and collect them all in a **single object** (namespace), so they don't pollute the global scope. Implement the following functions:

- **`add(vec1, vec2)`** - Addition of two vectors - $f(\vec{a}, \vec{b}) = \overrightarrow{\begin{pmatrix} x_a + x_b \\ y_a + y_b \end{pmatrix}}$
- **`multiply(vec1, scalar)`** - Scalar multiplication - $f(\vec{a}, s) = \overrightarrow{\begin{pmatrix} x_a \times s \\ y_a \times s \end{pmatrix}}$
- **`length(vec1)`** - Vector length - $f(\vec{a}) = \sqrt{x_a^2 + y_a^2}$
- **`dot(vec1, vec2)`** - Dot product of two vectors - $f(\vec{a}, \vec{b}) = x_a\, y_a + x_b\, y_b$
- **`cross(vec1, vec2)`** - Cross product of two vectors - $f(\vec{a}, \vec{b}) = x_a y_b - y_a x_b$

The math-savvy may notice that the given cross product formula results in a scalar, instead of a vector - we're only measuring the length of the resulting vector, since cross product is not possible in 2D, it will exist purely in the z-dimension. If you don't know what this all means, ignore this paragraph, it's irrelevant to the solution.

Input

Each separate function in your namespace will be tested with individual values. It must expect **one or two arguments**, as described above, and **return** a value. Vectors will be 2D **arrays** with format **`[x, y]`**.

Output

Your program needs to **return** an object, containing **all functions** described above. Each individual function must **return** a value, as required. Don't round any values.

Input	Output	Explanation
`solution.add([1, 1], [1, 0]);`	`[2, 1]`	`[1 + 1, 1 + 0] = [2, 1]`
`solution.multiply([3.5, -2], 2);`	`[7, -4]`	`[3.5 * 2, (-2) * 2] = [7, -4]`

`solution.length([3, -4]);`	`5`	`sqrt(3 * 3 + (-4) * (-4)) = 5`
`solution.dot([1, 0], [0, -1]);`	`0`	`1 * 0 + 0 * (-1) = 0`
`solution.cross([3, 7], [1, 0]);`	`-7`	`3 * 0 - 7 * 1 = -7`

What to submit?

Write a function named **main** that returns the object.

For example:

```
function main() {
    return { add: function(){}, multiply: ... }
}
```

Function Signature: function main()