More Exercises: Advanced Functions

1. Breakfast Robot

It's finally the future! Robots take care of everything and man has been freed from the mundane tasks of living. There is still worked to be done though, since those robots need to be programmed first - we may have robot chefs, but we do not yet have robot software developers.

Your task is to write the management software for a breakfast chef robot - it needs to **take orders**, keep track of available **ingredients** and output an **error** if something's wrong. The cooking instructions have already been installed, so your module needs to **plug into** the system and only take care of **orders** and **ingredients**. And since this is the future and food is printed with nano-particle beams, all ingredients are microelements - **protein**, **carbohydrates**, **fat** and **flavours**. The library of recipes includes the following meals:

- **`Apple`** - made with **1 carb** and **2 flavour**
- **`Lemonade`** - made with **10 carb** and **20 flavour**
- **`Burger`** - made with **5 carb**, **7 fat** and **3 flavour**
- **`Eggs`** - made with **5 protein**, **1 fat** and **1 flavour**
- **`Turkey`** - made with **10 protein**, **10 carb**, **10 fat** and **10 flavour**

The robot receives instructions either to **restock** the supply, **cook** a meal or **report** statistics. The input consists of one of the following commands:

- **`restock <microelement> <quantity>`** - increases the stored quantity of the given microelement
- **`prepare <recipe> <quantity>`** - uses the available ingredients to prepare the given meal
- **`report`** - returns information about the stored microelements, in the order described below, including zero elements

The robot is equipped with a quantum field storage, so it can hold an **unlimited quantity** of ingredients, but there is no guarantee there will be enough available to prepare a recipe, in which case an **error message** should be returned. Their availability is checked in the **order** in which they **appear** in the recipe, so the error should reflect the first requirement that was not met.

Submit a **closure** that returns the management function. The management function takes one parameter.

Input

Instructions are passed as a **string argument** to your management function. It will be called **several times** per session, so internal state must be **preserved** throughout the entire session.

Output

The **return** value of each operation is one of the following strings:

- **`Success`** - when restocking or completing cooking without errors
- **`Error: not enough <ingredient> in stock`** - when the robot couldn't muster enough microelements
- **`protein={qty} carbohydrate={qty} fat={qty} flavour={qty}`** - when a report is requested, in a single string

Constraints

- Recipes and ingredients in commands will always have valid names.

Examples

Execution

`let manager = solution();`
`manager("restock flavour 50");  // Success`
`manager("prepare lemonade 4");  // Error: not enough carbohydrate in stock`

Input	Output
`restock carbohydrate 10`	`Success`
`restock flavour 10`	`Success`
`prepare apple 1`	`Success`
`restock fat 10`	`Success`
`prepare burger 1`	`Success`
`report`	`protein=0 carbohydrate=4 fat=3 flavour=5`

Input	Output
`prepare turkey 1`	`Error: not enough protein in stock`
`restock protein 10`	`Success`
`prepare turkey 1`	`Error: not enough carbohydrate in stock`
`restock carbohydrate 10`	`Success`
`prepare turkey 1`	`Error: not enough fat in stock`
`restock fat 10`	`Success`
`prepare turkey 1`	`Error: not enough flavour in stock`
`restock flavour 10`	`Success`
`prepare turkey 1`	`Success`
`report`	`protein=0 carbohydrate=0 fat=0 flavour=0`

What to submit?

Function Signature: function main()

2. Monkey Patcher

Your employer placed you in charge of an old forum management project. The client requests new functionality, but the legacy code has high coupling, so you don't want to change anything, for fear of breaking everything else. You know which values need to be accessed and modified, so it's time to monkey patch!

Write a program to extend a forum post record with voting functionality. It needs to have the options to **upvote**, **downvote** and tally the **total score** (positive minus negative votes). Furthermore, to prevent abuse, if a post has more than 50 **total votes**, the numbers must be obfuscated – the stored values remains the same, but the **reported** amounts of upvotes and downvotes have a number **added** to them. This number is 25% of the **greater number** of votes (positive or negative), rounded up. The actual numbers should **not be modified**, just the reported amounts.

Every post also has a **rating**, depending on its score. If **positive** votes are the overwhelming majority (>66%), the rating is **`hot`**. If there is no majority, but the balance is non-negative and **either** votes are more than 100, its rating is **`controversial`**. If the balance is negative, the rating becomes **`unpopular`**. If the post has less than 10 **total** votes, or no other rating is met, it's rating is **`new`** regardless of balance. These calculations are performed on the actual numbers.

Your function will be invoked with **`call(object, arguments)`**, so treat it as though it is internal for the object. A forum post, to which the function will be attached, has the following structure:

JavaScript

```
{
  id: <id>,
  author: <author name>,
  content: <text>,
  upvotes: <number>,
  downvotes: <number>
}
```

The arguments will be one of the following strings:

- **`upvote`** – increase the positive votes by one
- **`downvote`** – increase the negative votes by one
- **`score`** – report positive and negative votes, balance and rating, in an array; obfuscation rules apply

Input

Input will be passed as arguments to your function through a **`call()`** invocation.

Output

Output from the report command should be **returned** as a result of the function in the form of an **array** of three **numbers** and a **string**, as described above.

Examples

Sample execution

`let post = {`
`  id: '3',`
`  author: 'emil',`
`  content: 'wazaaaaa',`
`  upvotes: 100,`
`  downvotes: 100`
`};`
`solution.call(post, 'upvote');`
`solution.call(post, 'downvote');`
`let score = solution.call(post, 'score'); // [127, 127, 0, 'controversial']`
`solution.call(post, 'downvote'); …  // (executed 50 times)`
`score = solution.call(post, 'score');  // [139, 189, -50, 'unpopular']`

Explanation
The post begins at 100/100, we add one upvote and one downvote, bringing it to 101/101. The reported score is inflated by 25% of the greater value, rounded up (26). The balance is 0, and at least one of the numbers is greater than 100, so we return an array with rating 'controversial'. We downvote 50 times, bringing the score to 101/151, the reported values are inflated by 151*0.25=38 (rounded up) and since the balance is negative with return an array with rating 'unpopular'.

What to submit?

Function Signature: function main()