## 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'. |