Functions and Statements - More Exercise

Problems with exercise and homework for the "JS Front-End" Course @ SoftUni.

1. Car Wash

Write a JS function that receives some commands. Depending on the command, **add** or **subtract** a percentage of how much the car is **cleaned** or **dirty**. Start from 0. The first command will always be '**soap**':

- **`soap`** – add 10 to the value
- **`water`** – increase the value by 20%
- **`vacuum cleaner`** – increase the value by 25%
- **mud** – decrease the value by 10%

The **input** comes as an **array of strings**. When finished cleaning the car, **print** the resulting value in the format:

```
`The car is {value}% clean.`
```

Note: The **value** should be rounded to the second decimal point.

Examples

Input	Output
`['soap', 'soap', 'vacuum cleaner', 'mud', 'soap', 'water']`	`The car is 39.00% clean.`
`["soap", "water", "mud", "mud", "water", "mud", "vacuum cleaner"]`	`The car is 13.12% clean.`

2. Number Modification

Write a JS program that changes a number until the average of all its digits is **not higher than 5**. To modify the number, your program should append a **9** to the end of the number, when the average value of all its digits is **higher than 5** the program should stop appending.

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 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

3. Points Validation

Write a JS program that receives two points in the format **[x1, y1, x2, y2]** and checks if the distances between each point (**x, y**) 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** print: **`{x1, y1} to {x2, y2} is valid`**
- In case the distance is **invalid** print: **`{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.

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`

Hints

You can use the following **formula** to help you calculate the distance between the **points** (x1, y1) and (x2, y2).

$$\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

4. Radio Crystals

You need to write a JS program that monitors the **current thickness** of the crystal and recommends the next procedure that will bring it closer to the desired frequency. To **reduce** waste and the time it takes to make each crystal your program needs to **complete** the process with the **least number of operations**. **Each operatio**n takes the **same amount of time**, but since they are done at different parts of the factory, the crystals have to be transported and thoroughly washed **every time** an operation **different** from the previous must be performed, so this must also be taken into account. When **determining** the order, always attempt to start from the operation that **removes** the largest amount of material.

The different operations you can perform are the following:

- **`Cut`** – cuts the crystal in 4
- **`Lap`** – removes 20% of the crystal's thickness
- **`Grind`** – removes 20 microns of thickness
- **`Etch`** – removes 2 microns of thickness
- **`X-ray`** – increases the thickness of the crystal by 1 micron; this operation can only be done once!
- **`Transporting and washing`** – removes any imperfections smaller than 1 micron (round down the number); do this after every batch of operations that remove material

At the beginning of your program, you will receive a number representing the desired **final thickness** and a series of **numbers**, representing the thickness of crystal ore in microns. Process each chunk and **print** to the console the order of **operations** and the **number** of times they need to be **repeated** to bring them to the desired thickness.

The **input** comes as a numeric array with a variable number of elements. The **first number** is the **target** thickness and **all following numbers** are the thickness of **different chunks** of quartz ore.

The **output** is the order of operation and how many times they are repeated, every operation on a new line. See the examples for more information.

Examples

Input	Output

`[1375, 50000]`	`Processing chunk 50000 microns`
	`Cut x2`
	`Transporting and washing`
	`Lap x3`
	`Transporting and washing`
	`Grind x11`
	`Transporting and washing`
	`Etch x3`
	`Transporting and washing`
	`X-ray x1`
	`Finished crystal 1375 microns`

`[1000, 4000, 8100]`	`Processing chunk 4000 microns`
	`Cut x1`
	`Transporting and washing`
	`Finished crystal 1000 microns`
	`Processing chunk 8100 microns`
	`Cut x1`
	`Transporting and washing`
	`Lap x3`
	`Transporting and washing`
	`Grind x1`
	`Transporting and washing`
	`Etch x8`
	`Transporting and washing`
	`Finished crystal 1000 microns`

5. Print DNA

Write a JS program that **prints** a DNA helix with a **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 number. It represents the length of the required helix.

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

Examples

Input	Output
`4`	`**AT**` `*C--G*`

Input	Output

`10`	`**AT**`
	`*C--G*`
	`T----T`

	T----T *A--G*			*A--G* **GG** *A--T* C----G *T--T* **AG** *G--G*