# Programming Fundamentals with Python: Exam Preparation

## 01. Activation Keys

**Submit your solutions in the SoftUni judge system at** [**https://judge.softuni.org/Contests/Practice/Index/2302#0**](https://judge.softuni.org/Contests/Practice/Index/2302#0)**.**

*You are about to make some good money, but first, you need to think of a way to verify who paid for your product and who didn't. You have decided to let people use the software for a free trial period and then require an activation key to continue using the product. Before you can cash out, the last step is to design a program that creates unique activation keys for each user. So, waste no more time and start typing!*

The first line of the input will be your raw activation key. It will consist of **letters and numbers only**.

After that, until the "Generate" command is given, you will be receiving strings with instructions for different operations that need to be performed upon the raw activation key.

There are several types of instructions, split by ">>>":

* "Contains>>>{substring}":
  + If the raw activation key contains the given substring, prints: "{raw activation key} contains {substring}".
  + Otherwise, prints: "Substring not found!"
* "Flip>>>Upper/Lower>>>{startIndex}>>>{endIndex}":
  + Changes the substring **between the given indices (the end index is exclusive)** to upper or lower case and then prints the activation key.
  + All given indexes will be valid.
* **"Slice>>>{startIndex}>>>{endIndex}**":
  + **Deletes** the characters between the start and end indices (**the end index is exclusive) and** prints the activation key.
  + Both indices will be **valid**.

### Input

* The first line of the input will be a string consisting of **letters and numbers only**.
* After the first line, until the "Generate" command is given, you will be receiving **strings**.

### Output

* After the "Generate" command is received, print:
  + "Your activation key is: {activation key}"

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| abcdefghijklmnopqrstuvwxyz  Slice>>>2>>>6  Flip>>>Upper>>>3>>>14  Flip>>>Lower>>>5>>>7  Contains>>>def  Contains>>>deF  Generate | abghijklmnopqrstuvwxyz  abgHIJKLMNOPQRstuvwxyz  abgHIjkLMNOPQRstuvwxyz  Substring not found!  Substring not found!  Your activation key is: abgHIjkLMNOPQRstuvwxyz |
| **Comments** | |
| 1. **Slice>>2>>6**   abcdefghijklmnopqrstuvwxyz becomes abghijklmnopqrstuvwxyz   1. **Flip>>>Upper>>>3>>>14**   abghijklmnopqrstuvwxyz becomes abgHIJKLMNOPQRstuvwxyz   1. **Flip>>>Lower>>>5>>>7**   abgHIJKLMNOPQRstuvwxyz becomes abgHIjkLMNOPQRstuvwxyz   1. **Contains>>>def**   abgHIjkLMNOPQRstuvwxyz does not contain def   1. **Contains>>>deF**   abgHIjkLMNOPQRstuvwxyz does not contain deF  The final activation key is abgHIjkLMNOPQRstuvwxyz | |
| **Input** | **Output** |
| 134softsf5ftuni2020rockz42  Slice>>>3>>>7  Contains>>>-rock  Contains>>>-uni-  Contains>>>-rocks  Flip>>>Upper>>>2>>>8  Flip>>>Lower>>>5>>>11  Generate | 134sf5ftuni2020rockz42  Substring not found!  Substring not found!  Substring not found!  134SF5FTuni2020rockz42  134SF5ftuni2020rockz42  Your activation key is: 134SF5ftuni2020rockz42 |

## 02. Fancy Barcodes

**Submit your solutions in the SoftUni judge system at** [**https://judge.softuni.org/Contests/Practice/Index/2303#1**](https://judge.softuni.org/Contests/Practice/Index/2303#1)**.**

Your first task is to determine if the given sequence of characters is a **valid** barcode or **not**.

**Each line must not contain anything else but a valid barcode**. A barcode is **valid** when:

* It is surrounded by a "@" followed by one or more "#"
* It is **at least 6 characters long** (without the surrounding "@" or "#")
* **It starts** with a **capital letter**
* It contains **only letters** (lower and upper case) **and digits**
* **It ends** with a **capital letter**

Examples of valid barcodes: @###Che46sE@##, @#FreshFisH@#, @###Brea0D@###, @##Che46sE@##

Examples of invalid barcodes: **##InvaliDiteM##**, **@InvalidIteM@**, **@#Invalid\_IteM@#**

Next, you have to determine the **product group** of the item from the **barcode**. The product group is obtained by **concatenating** **all the digits** found in the barcode. If there are **no digits** present in the barcode, the **default** product group is "00".

Examples:

@#FreshFisH@# -> product group: 00

@###Brea0D@### -> product group: 0

@##Che4s6E@## -> product group: 46

### Input

On the first line, you will be given an integer **n** – the count of barcodes that you will be receiving next.

On the following **n** lines, you will receive different strings.

### Output

For each barcode that you process, you need to print a message.

* If the barcode is invalid: "**Invalid barcode**"
* If the barcode is valid: "**Product group: {product group}**"

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 3  @#FreshFisH@#  @###Brea0D@###  @##Che4s6E@## | Product group: 00  Product group: 0  Product group: 46 |
| **Input** | **Output** |
| 6  @###Val1d1teM@###  @#ValidIteM@#  ##InvaliDiteM##  @InvalidIteM@  @#Invalid\_IteM@#  @#ValiditeM@# | Product group: 11  Product group: 00  Invalid barcode  Invalid barcode  Invalid barcode  Product group: 00 |

## 03. Need for Speed III

**Submit your solutions in the SoftUni judge system at** [**https://judge.softuni.org/Contests/Practice/Index/2307#2**](https://judge.softuni.org/Contests/Practice/Index/2307#2)**.**

*You have just bought the latest and greatest computer game – Need for Seed III. Pick your favorite cars and drive them all you want! We know that you can't wait to start playing.*

On the first line of the standard input, you will receive an integer **n** – the **number of cars** that you can obtain. On the next **n** lines, the **cars themselves** will follow with their **mileage** and **fuel** **available**, separated by "|" in the following format:

"{car}|{mileage}|{fuel}"

Then, you will be receiving different **commands**, each on a new line, separated by " : ", until the "Stop" command is given:

* "Drive : {car} : {distance} : {fuel}**"**:
  + You need to **drive the given distance**, and you will **need the given** fuel to do that. If the car **doesn't have enough fuel**, print: "**Not enough fuel to make that ride**"
  + If the car has the required fuel available in the tank, **increase its mileage** with **the given distance**, **decrease its fuel with the given fuel,** and **print**:   
    "{car} driven for {distance} kilometers. {fuel} liters of fuel consumed."
  + You like driving new cars only, so if a car's mileage reaches **100 000** km, remove it from the collection(s) and print: "**Time to sell the {car}!**"
* "Refuel : {car} : {fuel}**"**:
  + **Refill** the tank of your car.
  + Each tank can hold a **maximum of 75 liters of fuel**, so if the given amount of fuel is more than you can fit in the tank, take only what is required to fill it up.
  + Print a message in the following format: "{car} refueled with {fuel} liters"
* "Revert : {car} : {kilometers}**"**:
  + Decrease the **mileage** of the given **car with the given kilometers** and print the kilometers you have decreased it with in the following format:  
    "{car} mileage decreased by {amount reverted} kilometers"
  + If the mileage becomes **less** **than** **10 000km** **after** it is decreased, **just set it to 10 000km** and   
    **DO NOT print anything.**

Upon receiving the "Stop" command, you need to print all cars in your possession in the following format:  
"**{car} -> Mileage: {mileage} kms, Fuel in the tank: {fuel} lt.**"

### Input/Constraints

* The **mileage** and **fuel** of the cars will be valid, 32-bit integers, and will never be negative.
* The **fuel** and **distance** amounts **in the commands will never be negative**.
* The **car** **names** in the **commands** will always be **valid cars in your possession**.

### Output

* All the output messages with the appropriate formats are described in the problem description.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 3  Audi A6|38000|62  Mercedes CLS|11000|35  Volkswagen Passat CC|45678|5  Drive : Audi A6 : 543 : 47  Drive : Mercedes CLS : 94 : 11  Drive : Volkswagen Passat CC : 69 : 8  Refuel : Audi A6 : 50  Revert : Mercedes CLS : 500  Revert : Audi A6 : 30000  Stop | Audi A6 driven for 543 kilometers. 47 liters of fuel consumed.  Mercedes CLS driven for 94 kilometers. 11 liters of fuel consumed.  Not enough fuel to make that ride  Audi A6 refueled with 50 liters  Mercedes CLS mileage decreased by 500 kilometers  Audi A6 -> Mileage: 10000 kms, Fuel in the tank: 65 lt.  Mercedes CLS -> Mileage: 10594 kms, Fuel in the tank: 24 lt.  Volkswagen Passat CC -> Mileage: 45678 kms, Fuel in the tank: 5 lt. |
| **Comments** | |
| After we receive the cars with their mileage and fuel, we start driving them. When we get to "**Drive : Volkswagen Passat CC : 69 : 8**" command, our program calculates that there is not enough fuel, and we print the appropriate message. Then we refuel the Audi A6 with 50 l of fuel and Revert the Mercedes with 500 kilometers.  When we receive the "Revert : Audi A6 : 30000", we set its mileage to **10000** km, because if the current mileage of the Audi is **38543** kms and if we subtract **30000** from it, we receive **8543** kms, which is less than 10000 kms.  After all the commands, we print our current collection of cars with their current mileage and current fuel. | |
| **Input** | **Output** |
| 4  Lamborghini Veneno|11111|74  Bugatti Veyron|12345|67  Koenigsegg CCXR|67890|12  Aston Martin Valkryie|99900|50  Drive : Koenigsegg CCXR : 382 : 82  Drive : Aston Martin Valkryie : 99 : 23  Drive : Aston Martin Valkryie : 2 : 1  Refuel : Lamborghini Veneno : 40  Revert : Bugatti Veyron : 2000  Stop | Not enough fuel to make that ride  Aston Martin Valkryie driven for 99 kilometers. 23 liters of fuel consumed.  Aston Martin Valkryie driven for 2 kilometers. 1 liters of fuel consumed.  Time to sell the Aston Martin Valkryie!  Lamborghini Veneno refueled with 1 liters  Bugatti Veyron mileage decreased by 2000 kilometers  Lamborghini Veneno -> Mileage: 11111 kms, Fuel in the tank: 75 lt.  Bugatti Veyron -> Mileage: 10345 kms, Fuel in the tank: 67 lt.  Koenigsegg CCXR -> Mileage: 67890 kms, Fuel in the tank: 12 lt. |