# Python Advanced: Exam Preparation

[**Link to Judge**](https://alpha.judge.softuni.org/contests/python-advanced-retake-exam-11-december-2024/4989)

## Click Bait

A fish next to a sign

Description automatically generated

*It is difficult to remember a time when you could scroll through the social media outlet of your choice and not be bombarded with: You'll never believe what happened when … This is the cutest thing ever … This is the biggest mistake you can make … Take this quiz to see which character you are… They are all classic clickbait models.  
—Emily Shire*

### Main Idea

Your task is to simulate a "content optimization" process for a website using two sequences of scores and design an algorithm for a website to process user engagement on **two types of content**:

* **Suggested Links** - processed using the **FIFO principle**
* **Featured Articles** - processed using the **LIFO principle**

You must follow **specific rules to process the elements** and **calculate** an **Engagement Value**, which determines if the target **goal is achieved.**

### Input Data

* The first line contains **space-separated integers** representing the engagement scores of **Suggested Links**
* The second line contains **space-separated integers** representing the popularity scores of **Featured Articles**
* The third line contains a single integer - the **Target** **Engagement Value**

### Rules

You should **repeatedly** perform the following steps **until one of the sequences becomes empty**:

**Take one element** from each sequence:

* Take the **first** **element** from the **Suggested Links**
* Take the **last element** from the **Featured Articles**

**Compare the two elements:**

* Identify the **greater element** and the **smaller element**
* Remember the **sequence of origin** of the **greater element**, you will need it later

**Calculations**:

* **Find** the **remainder** between the two elements. The **greater element** will be used asa **dividend** and the **smaller** as a **divisor** using a [**Modulo Operation**](https://en.wikipedia.org/wiki/Modulo)
* If the **greater element** is from the **Featured Articles (LIFO) collection**:
  + **Add** the **remainder to the Final Feed Collection as a positive element**
  + **Double** the **remainder** and **return** the **result** to the **LIFO collection (at the end)**.If the **remainder equals zero**, **skip this operation** and **proceed** with the **next** calculation
* If the **greater element** is from the **Suggested Links (FIFO) collection**:
  + **Add** the **remainder to the Final Feed Collection as a negative element**
  + **Double** the **remainder** and **return** the **result** to the **FIFO collection (at the end)**.If the **remainder equals zero**, **skip this operation** and **proceed** with the **next** calculation
* If **elements** are **equal add zero (0)** to the **Final Feed Collection** and **do not return anything** to the **LIFO or FIFO collections**

**After processing**:

**Find** the **Total** **Engagement Value** as a **sum of all elements** in the **Final Feed Collection**.

* If the **Total Engagement Value** is **greater than or equal** **to** the **Target Engagement Value**, the **goal has been achieved**. See the[**Examples**](#_Examples) section
* If the **Total Engagement Value** is **less than** the **Target Engagement Value**, the **goal has not been achieved**
  + **Calculate how far short** the **Total Engagement Value** is **from the target** by **subtracting** the **Total Engagement Value** from the **Target Engagement Value:** *Shortfall = Target - Total*

Print the **Final Feed** collection on the Console, comma, and space separated **(", ")**.

Print the **appropriate output** for the **engagement value** on the Console. See the[**Examples**](#_Examples) section.

### Input

* On the **first line**, you will receive a **sequence of integers**, representing the engagement scores of   
  **Suggested Links, space separated (" ")**
* On the **second line**, you will receive a **sequence of integers**, representing the popularity scores of   
  **Featured Articles, space separated (" ")**
* On the **third line**, you will receive a **single integer** representing the **Target** **Engagement Value**

### Output

* Print the **Final Feed Collection**:

"Final Feed: {element1}, {element2} … {elementn}"

* Calculate the **Total** **Engagement Value**:
  + If the Total Engagement Value is **greater than or equal to the** **target**, print:

"Goal achieved! Engagement Value: {total\_engagement\_value}"

* + If the Total Engagement Value is **less than the target** (*Shortfall = Target - Total*), print:

"Goal not achieved! Short by: {shortfall}"

### Constraints

* Always process elements in the order described
* All of the given numbers will be **valid integers** in the range **[1-1000]**
* Use **modulo operation** (%) to calculate the remainder
* Pay attention to the **sign of the remainder in the Final Feed**
* **Both** sequences will **initially** have **at least one element**.

### Examples

|  |  |
| --- | --- |
| ****Input**** | ****Output**** |
| 25 10  40 30  35 | **Final Feed: 5, 0**  **Goal not achieved! Short by: 30** |
| ****Comment**** | |
| **First round:** Take 25 from the **Featured Articles** and 30 from the **Suggested Links.** The greater value is 30, the smaller value is 25, and the **remainder** is 30%25 = 5. Add **+5** to the **Final Feed** (**positive** because **the greater element is from the** **LIFO collection**). **Double the Remainder**: 5×2=**10**, and return it to the **original collection** - **LIFO collection**. Collections **updated** **state**:  **Featured Articles:** [10]  **Suggested Links:** [40,10]  **Final Feed:** [5]  **Second Round:** Take 10 from the **Featured Articles** and 10 from the **Suggested Links.** There is **no greater** or **smaller** value. Add **0** to the **Final Feed**. **Skip** the step for returning the **doubled remainder** to the original collection as it **equals zero** (0×2=**0**).Collections **updated state**:  **Featured Articles:** []  **Suggested Links:** [40]  **Final Feed:** [5,0]  **The Featured Articles collection is empty, so the program ends and the result is printed on the console.** | |
| ****Input**** | ****Output**** |
| 45 65 35 25 70  15 30 20 10 5 40  55 | **Final Feed: -5, 0, -5, -5, -10, 5, 0**  **Goal not achieved! Short by: 75** |
| ****Input**** | ****Output**** |
| 12 18 971 65 31  1 3 121 500 4 33  80 | **Final Feed: 9, 0, -3, 45, 28, 2**  **Goal achieved! Engagement Value: 81** |
| ****Input**** | ****Output**** |
| 77 233 12 66 6 34  1 12 900 999 33  26 | **Final Feed: -11, 67, 2, -2, 0, -10, 0**  **Goal achieved! Engagement Value: 46** |

## Space Mission

A space ship in space

Description automatically generated

*You are commanding a spaceship navigating through uncharted space in search of Planet B. The journey is perilous, with meteorites that drain your resources and the constant danger of running out of fuel. Your mission is to reach Planet B and save humanity.*

You will be given an integer **N** representing the size of the space field, which is a **square grid** (each position on the grid represents a sector of space). On the next **N** lines, you will receive the **rows** of the space **grid** with symbols, **separated by a single space** (**" "**). See the [**Examples**](#_Examples_1) section.

* A fireball in the air

  Description automatically generatedYour **spaceship** starts at a **random position**, marked by **'S'**.
* **Meteorites** are scattered across space, marked by **'M'**.
  + Free atmosphere earth world vectorEach time you pass through a meteorite sector, your resources are **decreased** by **5 units**.
* Free gasoline pump fuel dispenser filling pump vector**Planet B**, your **destination**, is marked by **'P'**.
* There are **space stations**, marked by **'R'** (for "Resources").
  + Where you can **refuel** and **gain 10 units** of resources (up to a maximum of 100 units).
* All other sectors are **empty** and marked by **'.'**.



**Mission Rules**:

* Your **mission** is to **reach** **Planet B** (**'P'**) with the resources you have and gain during the journey.
* Your **spaceship starts** with **100 units** of **resources initially** ata**position**, marked by **'S'**.
  + When the spaceship makes the **first move**, its **starting position is marked** by **'.'**.
* You will be given **commands** to **move** the **spaceship**.
  + The **valid commands** are: "**up**", "**down**", "**left**", and "**right**".
* Each **move decreases** your resources by **5 units**.
  + If the **spaceship moves** to a sector with a **space station** (**'R'**), you **gain 10 units of resources**, and the **station remains in place**.
    - You can **refuel** at the **same station** **each time you encounter it**. But your **resources cannot go above 100 units**. If they do, **set them to 100**.
  + If the **spaceship** **moves** into a sector with a **meteorite** (**'M'**), an **additional** **5 units** are **subtracted from your resources**, and the **meteorite is destroyed** by the collision (replaced by **'.'**).
  + If your **resources drop below 5 units before reaching Planet B** or a **space station to refuel**, the **mission fails**, and the **spaceship is stranded in space**.
    - Remember the **last known position** of the **spaceship**.
  + If the **spaceship moves** **out of the grid's boundaries**, it is considered **lost in space** and the **mission ends immediately** with a**failure**.
    - Remember the **last known position** of the **spaceship** **before moving out** of **boundaries**.

### Input

* On the **first line**, you will receive an integer **N** (the size of the square grid).
* The next **N** lines will represent the grid, with each sector marked as **'S'**, **'M'**, **'P'**, **'R'**, or **'.'**.
  + Letters **'S'**and **'P'**appear only once.
  + Symbols are **separated** by a **single space** (**" "**). See the [**Examples**](#_Examples_1) section.
* After the grid, a series of **valid** **movement commands** will follow, each on a new line.

### Output

* On the **first** line:
  + If the **spaceship reaches Planet B** with **zero or more** resources, print:
* **"Mission accomplished! The spaceship reached Planet B with {resources} resources left."**
* If the **spaceship runs out of resources** **before reaching Planet B** or a **space station**, print:
* **"Mission failed! The spaceship was stranded in space."**
* If the **spaceship exits the grid's boundaries** and **gets lost** in space, print:
* **"Mission failed! The spaceship was lost in space."**
* On the **next** lines:
* Print the **final state** of the grid, with the **spaceship's last known position** marked by **'S'**.
  + If the spaceship **successfully landed on Planet B**, it is **not displayed on the grid**.
  + Symbols should be **separated** by a **single space** (**" "**). See the [**Examples**](#_Examples_1) section.

### Constraints

* The size of the **square** matrix (grid) will be between **[2 -10] inclusive.**
* Each sector will be marked as **'S'**, **'M'**, **'P'**, **'R'**, or **'.'**.
* Letters **'S'**and **'P'**appear only once.
* There will always be **enough** commands to either **succeed or fail** the mission.
* The **spaceship** starts with **100 units of resources** **initially**.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 3  S . .  . M .  . R P  right  down  down  right | Mission accomplished! The spaceship reached Planet B with 85 resources left.  . . .  . . .  . R P |
| **Comment** | |
| The spaceship starts at **(0,0)** with **100** units of resources.  There is one meteorite **('M')** at **(1,1)**, and a space station **('R')** at **(2,1)**.  Planet B **('P')** is located at **(2,2)**.  Moves:  Move 1 (**right**): The spaceship moves to **(0,1)**, **-5 units for the move (95 remaining)**.  Move 2 (**down**): The spaceship moves to **(1,1)**, hitting the meteorite, **-5 units for the move and -5 units for the meteorite (85 remaining)**. The meteorite is replaced by '.'.  Move 3 (**down**): The spaceship moves to **(2,1)**, **-5 units for the move (80 remaining)**, and **refuels at the space station, gaining +10 units (90 remaining)**.  Move 4 (**right**): The spaceship moves to **Planet B (2,2)**, **-5 units for the move (85 remaining)**. Mission **accomplished**! | |
| **Input** | **Output** |
| 5  S M . . P  M M . R .  M . M . .  . . M . R  . . M . .  right  right  down  down  down  down  left  left  up  up  up  right  right | Mission failed! The spaceship was stranded in space.  . . . . P  . . S R .  . . . . .  . . . . R  . . . . . |
| **Comment** | |
| **Each move costs 5 units of resources**. **Each meteorite collision costs another 5 units**. Unfortunately, the spaceship **did not drop onto the space station** to refuel and **did not manage to reach Planet B**. The **mission failed** as the **spaceship ran out of resources** at position **(1, 2)**. This is the last known position and is marked by **'S'** | |
| **Input** | **Output** |
| 4  . M . .  . S R P  . . . .  . . . .  left  down  down  left | Mission failed! The spaceship was lost in space.  . M . .  . . R P  . . . .  S . . . |
| **Input** | **Output** |
| 5  M S . . .  M M . . .  M . . R .  . . M . P  . . R . M  left  right  down  down  down  down  left  up  up  up  right  up  right  right  down  down  right  down | Mission accomplished! The spaceship reached Planet B with 0 resources left.  . . . . .  . . . . .  . . . R .  . . M . P  . . R . M |

## Planting Garden System

A group of people working in a garden

Description automatically generated

*You are responsible for planting different types of plants in a garden. You need to design a system that tracks the planting process based on available space and the type of plants owners want to grow.*

Create a function named **plant\_garden** that **receives information** about the **available space** in the **garden**, the **plot size** for **each allowed plant type**, and the **requested** **plant types for planting** with their **quantities** (pieces) and **returns** a **sorted result** as **one formatted string**. It will receive **one positional argument**, followed by an **unknown number** of **arguments** (**tuples**) and **keyword arguments** (**key-value pairs**). See the [**Examples**](#_Examples_2) section.

The **arguments** will be passed as follows:

* The **positional argument** represents the **available garden space** (in square meters) - a **float number** in the range [**0.1, 100.0**].
* The **next group** of **arguments** represents the **allowed types of plants** and **their space requirements**, **containing** an **unknown number** of **tuples** with **two elements**:
  + The **first element** represents the **unique** **plant type** (a **valid** **string**).
  + The **second element** representsthe**space required per plant** (a positive **float number**).
* The **last group** of **arguments** represents the **planting requests** and **contains** an **unknown number** of **keyword arguments** (**key-value** **pairs**).
  + The **key** represents the **unique** **plant type** (a **valid string**).
  + The **value** representsthe **pieces** (quantity)from that **plant type** owners **want to grow** (a **positive integer**, greater than zero).

The function should **track** the **planting process**:

* **Sort** the **planting requests** by plant type **alphabetically**.
* For **each requested plant type** from the group of **sorted keyword arguments** (planting requests), **check** if it **exists** in the **provided** group of **allowed plant types** (**first group of arguments** with plant **type** and **space requirements**).
  + If the **plant type exists** and there is **enough space**, **plant as many pieces as possible** based on the provided **quantity** and **available garden space**. See the [**Examples**](#_Examples_2) section.
    - If space allows **only partial planting** (not all requested plant pieces can be planted), **plant as many as possible within the space limits**, but this will **affect** the **outcome**.
  + If the **requested plant type doesn't exist in the first group of allowed types**, **ignore it** and do **not count** it in the **outcome**.
* **Stop planting** when the **available garden space is used up** or there are **no more requests** for planting.

At the end, **return** the **output**, depending on the **outcome** asdescribed below:

* If you **managed to plant all the requested plants** at their **full quantity** (pieces), **return** the message:

**"All plants were planted! Available garden space: {available\_space} sq meters."**

* **Format** the **available space** to the **first decimal place**.
* **Partial planting** of the requested quantity will be considered an **unsuccessful outcome**.
* If you **ran out of space before planting all requested plants**, **return** the message:

**"Not enough space to plant all requested plants!"**

* On the following lines, **return** the **planted plant types and the pieces of each type planted**, **sorted alphabetically by plant type**, one per line (see the [**Examples**](#_Examples_2) section):

**"Planted plants:**

**{plant\_type1}: {pieces}**

**{plant\_type2}: {pieces}**

**...**

**{plant\_typen}: {pieces}"**

***Note: Submit only the function in the judge system***

### Input

* There will be **no input from the console**, just arguments passed to your function.

### Output

* **Return** the **appropriate message** based on **whether all plants were planted**, **along with details** about the **planted types** as **described above**.

### Constraints

* The first **positional argument** will always be a **float number** in the range [**0.1, 100.0**].
* The group of **arguments** will always be **before** thegroup of **keyword arguments**.
* Each **tuple** from the first group of arguments will always provide the **allowed** **unique** **plant type** (a **valid string**) and the **space required per plant** (a positive **float number**).
* Each **keyword argument** from the second group will always provide a **unique** **plant type** (a **valid** **string**) and the **pieces requested for planting** (a **positive integer**, greater than zero).
* You will always receive **at least** **one** **tuple** and **at least one** **keyword argument**.
* There will always be **at least one planted plant**.

### Examples

|  |  |
| --- | --- |
| **Test Code** | **Output** |
| print(plant\_garden(50.0, ("rose", 2.5), ("tulip", 1.2), ("sunflower", 3.0), rose=10, tulip=20)) | All plants were planted! Available garden space: 1.0 sq meters.  Planted plants:  rose: 10  tulip: 20 |
| print(plant\_garden(20.0, ("rose", 2.0), ("tulip", 1.2), ("sunflower", 3.0), rose=10, tulip=20, sunflower=5)) | Not enough space to plant all requested plants!  Planted plants:  rose: 10 |
| print(plant\_garden(2.0, ("rose", 2.5), ("tulip", 1.2), ("daisy", 0.2), rose=4, tulip=15, sunflower=3, daisy=4)) | Not enough space to plant all requested plants!  Planted plants:  daisy: 4  tulip: 1 |
| print(plant\_garden(50.0, ("tulip", 1.2), ("sunflower", 3.0), rose=10, tulip=20, daisy=1)) | All plants were planted! Available garden space: 26.0 sq meters.  Planted plants:  tulip: 20 |