# Python Advanced: Exam Preparation

[**Link to Judge**](https://alpha.judge.softuni.org/contests/python-advanced-regular-exam-22-june-2024/4796)

## Rapid Courier

A cartoon of a child driving a blue car

Description automatically generated

*You have created your own delivery service called "Rapid Courier". You want to analyze how well your couriers are doing in delivering all the packages.*

On the **first line**, you will be given a **sequence of integers** representing **packages** that need to be delivered, with the values of their **weight**.

On the **next line**, you will be given **another sequence of integers** representing **couriers** with their loading **capacities**.

Until **there are packages** to deliver **and couriers** available, the program continues running.

Track the **total weight of packages** **delivered** by your couriers.

You need to **compare** the [**last package**](#LastPackage) to the [**first courier**](#FirstCourier):

* If the courier can deliver the package (the **capacity of the courier is equal to or greater than the weight of the package**), he does the delivery:
  + If the **capacity of the courier is greater than the weight of the package**, **reduce** the **courier's capacity** by **twice the package's weight**:
    - If the **new courier's capacity** is **positive**, the **courier moves at the back** of the sequence with the **updated capacity**.
    - If the new capacity is **zero or negative**, **remove** the **courier.**
  + **Аdd the weight of the package** to the **total delivered weight.**
  + **Remove the package** from the sequence.
* If the courier cannot deliver the package (the **capacity of the courier is less than the weight of the package**), **subtract the courier's capacity** from the **package's weight**
  + Return the **remaining weight** to the **sequence** (on its initial position),and **remove the courier**.
  + **Add** the delivered **portion of the package's weight** to the **total delivered weight**.

### Input / Constraints

* On the **first line**, you will receive **integers** representing the **weight of the packages** to be delivered, separated by a **single space**.
* On the **second line**, you will receive **integers** representing the **capacities of the couriers**, separated by a **single space**.
* All given numbers will be valid **integers** in the range **[0; 100]**.

### Output

The output of your program should be printed on the Console, on separate lines, formatted according to the following rules:

* At the end of the program, print the weight of the packages delivered:
  + "**Total weight: {total\_weight} kg"**
* If all of the **packages are delivered** and there are **no couriers left**:
  + "**Congratulations, all packages were delivered successfully by the couriers today.**"
* If **there are packages** left but **no more couriers available**:
* **"Unfortunately, there are no more available couriers to deliver the following packages: {package1}, {package2}, (…),{packagen}"**
* Print the packages in their current order
* If **there are couriers** left but there are **no more packages to deliver**:
* **"Couriers are still on duty: {****couriers1}, {couriers2}, (…),{couriersn} but there are no more packages to deliver."**
* Print the couriers in their current order

### Examples

|  |  |  |
| --- | --- | --- |
| ****Input**** | ****Output**** | ****Comment**** |
| **2 4 6** **8**  **8 6 4 2** | **Total weight: 20 kg**  **Congratulations, all packages were delivered successfully by the couriers today.** | The first pair consists of the **last package** with a weight of 8 and the **first courier's** capacity with a value of 8. Since the two **values are equal**, the courier delivers the package successfully and **both values are removed** from the collections.  8 kilograms of packages are delivered.  Now, the sequences are as follows:  2 4 6  6 4 2  We repeat the same operations until all packages are delivered and no courier available is left.  2 4 6  6 4 2  8 + 6 = 14 kg of packages are delivered  2 4  4 2  8 + 6 + 4 = 18 kg of packages are delivered  2  2  8 + 6 + 4 + 2 = 20 kg of packages are delivered  Finally, since there are **no more elements in both sequences**, the program ends. The correct output is printed on the Console. |
| **13 11 5**  **5** 11 | **Total weight: 16 kg**  **Unfortunately, there are no more available couriers to deliver the following packages: 13** | The first pair consists of the **last package** with a weight of **5** and the **first courier's** capacity with a value of **5**. Since the two **values are equal**, the courier delivers the package successfully and **both values are removed** from the collections.  **5** kilograms of packages are delivered.  Now, the sequences are as follows:  13 11  11  We repeat the same operations until all packages are delivered and no courier available is left.  13 **11**  **5 + 11** = 16 kg of packages are delivered  Now, the sequences are as follows:  13  [the second sequence is empty]  Finally, since there are **no more elements in the courier sequence**, the program ends. The correct output is printed on the Console. |
| **3 7 14**  **2 2 2 1 7** | **Total weight: 14 kg**  **Unfortunately, there are no more available couriers to deliver the following packages: 3, 7** | The first pair consists of the **last package** with a weight of **14** and the **first courier's** capacity with a value of **2**. Since the **value of the package weight is greater than the value of the courier loading capacity**, we take the value of the package, decrease it by the value of the courier put it back on the top of the sequence. We remove the value of the courier from its sequence.  **2** kilograms of packages are delivered.  Now, the sequences are as follows:  3 7 **12**  2 2 1 7  **2** kilograms of packages are delivered.  We repeat the same operations until all packages are delivered and no courier available is left.  3 7 **10**  2 1 7  **4** kilograms of packages are delivered.  3 7 **8**  1 7  **6** kilograms of packages are delivered.  3 7 **7**  7  **7** kilograms of packages are delivered.  Now, the sequences are as follows:  3 7  [the second sequence is empty]  **14** kilograms of packages are delivered.  Finally, since there are no more elements in the courier sequence, the program ends. The correct output is printed on the Console. |

## Beesy



*Bees are indispensable for maintaining the health of ecosystems, promoting biodiversity, supporting agriculture, and ensuring food availability for human populations worldwide. Protecting bee populations and their habitats is crucial for the well-being of both ecosystems and humanity.*

You will be given an integer **n** for the **size** of the **field** with a **square** shape. On the next **n** lines, you will receive the **rows** of the **field**.

* Your bee will be placed in a **random position**, marked with the letter '**B**'.
* There will be flowers with nectar which need to be pollinated on **random positions**, marked with a **single digit**.
* There will be a **hive**, marked with the letter '**H**'.
* All of the empty **positions** will be marked with **'-'**.

The bee will have **15 units of energy initially**.

A command is received each turn **until the bee runs out of energy** or **reaches the hive**.

the bee must collect and take **at least** **30 units** **of nectar to the hive**. This would be the required quantity to make honey successfully.

Keep in mind that even if the **needed amount of nectar is collected, but the hive is not reached** the **bee continues to move** according to the commands.

You will be given **commands** for **the bee's movement**. The commands will be: "**up**", "**down**", "**left**", and "**right**". The bee moves towards the given direction.With **each** move, the **bee's energy** **decreases** by **1 unit.**

* If the **bee leaves the field** (goes out of the boundaries of the matrix) depending on the move command,  
   **it will be moved to the opposite side of the field.**   
  **Example:** In a 3x3 matrix the bee is at position **[1,2]** and receives the command "**right**" it will be moved to position **[1,0]**.
* If the bee **moves** to a **flower** (a position **marked** with a **digit**), it collects the nectar (**the value of the digit is** **added** to the **total amount of collected nectar**) the **flower disappears** and should be replaced by **'-'**.
* If the bee **runs out of energy**, **and** the **total amount of collected nectar** **is less than 30 units,** the **program ends**. [The correct output should be printed on the Console](#Output).
* If the bee **runs out of energy** **and** the **total amount of collected** **nectar is at least 30 units**, the energy will be restored with the **amount of the difference** between the nectar **collected up to this turn** and the **minimum required** amount for making honey *(30).* The **value of the collected nectar is dropped to 30 units**. The energy **can be restored only once**.

**Example:** Collected nectar is equal to **40 units.** 40 – 30 = **10**. The bee's energy is **increased** by 10, the nectar is decreased to 30 units.

**Hint: Check for zero energy after restoration.**

* + The **next time the bee runs out of energy**, the movement discontinues.The **program ends**.   
    [The correct output should be printed on the Console](#Output).
* If the bee **reaches a position**, **marked with**  '**H**', the hive is reached and the **program ends**.

**Hint: When reaching the hive with zero energy, the success will depend on the amount of the collected nectar.**  
[The correct output should be printed on the Console](#Output).

### Input

* On the first line, you are given the integer **n** – the size of the **square** **matrix**.
* The **next n lines** contain the values for **every matrix row**.
* After that, you will get **commands to move** (each one on a new line).

### Output

* On the first line:
* If the bee **reaches the hive** with at least **30 units of nectar collected**, print this message and stop the program:
* **"Great job, Beesy! The hive is full. Energy left: {energy}"**
* If the bee **reaches the hive** but **has not succeeded in collecting at least 30 units** of nectar:
* **"Beesy did not manage to collect enough nectar."**
* If the **bee runs out of energy**, **before returning** to the hive:
* **"This is the end! Beesy ran out of energy."**
* On the **next** lines.
* Print the **final state of the** **matrix**, **with** **the last known position of the bee**, **marked with 'B'**.

### Constraints

* The size of the **square** matrix *(field)* will be between **[2…10].**
* Only the letters '**B**' and '**H**' will be present in the matrix.
* The **flowers with nectar** are represented by **single positive digits** between **[0…9]**.
* There will always be **enough** commands to finish the program.
* The bee will always have **15 units of energy initially**.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 5  --B--  H-987  -4812  5----  2----  down  right  right  down  left  left  left  down  left  up  up  up | Great job, Beesy! The hive is full. Energy left: 4  -----  B----  -----  -----  2---- |
| **Comment** | |
| The bee starts from the **position [0,2]**. The first command is **"down"**. The bee moves to **position** **[1,2]** and gets **9** units of nectar. The initial **energy drops to 14 units**. The next command is **"right" -> position [1,3]** gets **8 units of nectar, and the energy drops to 13 units.** Then the command **"right"** again, **position ->** **[1,4]** and **nectar -> 17 + 7**, **energy -> 12.** The next commands are **"down", "left","left",** and **"left" ->** and the bee collects accordingly **2,1,8** and **4** units of nectar. Position -> [2,1], Nectar -> 24 + 2 + 1 + 8 + 4 = 39, Energy -> 8.  Then **"down"** and **"left"** commands follow and your bee adds another **5** units of nectar to the previous amount. Finally, we get a command **"up"** three times, but **the bee steps on the letter 'H'** and that means it has reached the hive successfully collecting **44 units** of nectar.  Position -> [1,0], Nectar -> 39 + 5 = 44, Energy -> 4. | |
| **Input** | **Output** |
| 4  B999  --5-  ---H  ----  right  right  right  down  left  left  left  left  down | Great job, Beesy! The hive is full. Energy left: 6  ----  ----  ---B  ---- |
| **Comment** | |
| The bee starts its movement following the given commands – three times **"right"** and collects the digits *(amounts of nectar)* -> **9+9+9 =27**. Next is a **"down"** command followed by four **"left"** commands *(collects another* ***5*** *units of nectar on the second* ***"left"*** *command)* on the **fourth** left command the bee **leaves** the field boundaries and then **appears** on the **opposite** side of the field **[1,3].** Finally, the bee gets the **"down"** command to reach the hive and the program ends with **32** units of nectar collected. | |
| **Input** | **Output** |
| 4  B---  1991  ----  ---H  down  right  right  right  down  down  left | Beesy did not manage to collect enough nectar.  ----  ----  ----  ---B |
| **Input** | **Output** |
| 6  B-----  111111  ------  111111  ------  H-----  down  right  right  right  right  right  down  left  left  left  left  left  down  right  right  right  right  right | This is the end! Beesy ran out of energy.  ------  ------  ------  --B111  ------  H----- |

## Boarding Passengers

A white ship with red and blue stripes

Description automatically generated

*Embarking on a luxury cruise ship marks the beginning of a grand adventure, and efficient boarding is the gateway to unforgettable experiences at sea. Your function will serve as the captain of boarding, ensuring smooth sailing for all passengers from the moment they step on board. Your mission is to develop a Python function that orchestrates the boarding process seamlessly, balancing capacity constraints and efficiently managing passengers as they board the vessel.*

Write a function named **"boarding\_passengers"** that **receives information** about the available capacity of the ship and a passenger list and **returns the result after boarding**. The function will receive a **different number of arguments**. The arguments will be passed as follows:

* The **first** positional **argument** is the **capacity of the ship**:
  + An **integer** in the **range [0, 150]** inclusive.
* The second **group of arguments** represents the **passenger groups** asan **unknown number** **of tuples**:
  + Each **tuple** contains two elements:
    - the **number of passengers** as a **positive** **integer** in the **range [1, 150]** inclusive.
    - the **benefits program name** as a **string** (e.g., "**Diamond**", "**Platinum**", "**Gold**", "**First Cruiser**", etc.).

After receiving the information and calling the function, the program should **start the boarding process**:

* Board passengers **only** if you have **enough capacity**.
* Remember that you need to **track** the **total** **number of guests** based on their **benefit program**.
* If the **available capacity is not enough** to accommodate the **current group**, then **move to the next one** that can fit in.
* If the available **capacity** is **0** **(zero)**, **STOP boarding**!

To gather useful information, you need to **sort the boarding details**:

* Sort the information based on the boarded **number of guests** per each **benefit program** in **descending order**.
* If there is **more than one benefit program** with the **same number of guests**, order them according to their **benefits program name**, **ascending**.

**In the end, return** the output as described below.

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

### Input

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

### Output

* Return the **sorted** **boarding details** per **each benefit plan**:  
  **"Boarding details by benefit plan:**  
  **## {benefit plan1}: {total number of passangers1} guests  
  ## {benefit plan2}: {total number of passangers2} guests  
  …  
  ## {benefit plann}: {total number of passangersn} guests"**
* Your **output string** should **also contain** one of the following messages:
  + If **all passengers are boarded**, return the message:   
    **"All passengers are successfully boarded!"**
  + If the **ship's capacity is occupied** but there are **still guests waiting to board**, then return the message:   
    **"Boarding unsuccessful. Cruise ship at full capacity."**
  + If there is **still available capacity** but **not all passengers have embarked the vessel**, return the message:   
    **"Partial boarding completed. Available capacity: {available\_capacity}."**

### Constraints

* The **first argument** will always be an **integer** in the **range [0, 150] inclusive**.
* The **second group of arguments** will be an **unknown number of tuples**.
* There will always be **at least one tuple**.

### Examples

|  |  |
| --- | --- |
| **Test Code** | **Output** |
| print(boarding\_passengers(**150, (35, 'Diamond'), (55, 'Platinum'), (35, 'Gold'), (25, 'First Cruiser')))** | **Boarding details by benefit plan:**  **## Platinum: 55 guests**  **## Diamond: 35 guests**  **## Gold: 35 guests**  **## First Cruiser: 25 guests**  **All passengers are successfully boarded!** |
| print(boarding\_passengers(100, (20, 'Diamond'), (15, 'Platinum'), (25, 'Gold'), (25, 'First Cruiser'), (15, 'Diamond'), (10, 'Gold'))) | Boarding details by benefit plan:  ## Diamond: 35 guests  ## First Cruiser: 25 guests  ## Gold: 25 guests  ## Platinum: 15 guests  Boarding unsuccessful. Cruise ship at full capacity. |
| print(boarding\_passengers(120, (30, 'Gold'), (20, 'Platinum'), (30, 'Diamond'), (10, 'First Cruiser'), (31, 'Platinum'), (20, 'Diamond'))) | Boarding details by benefit plan:  ## Diamond: 50 guests  ## Gold: 30 guests  ## Platinum: 20 guests  ## First Cruiser: 10 guests  Partial boarding completed. Available capacity: 10. |