# Exercises: Dictionary

Problems for exercises and homework for the [“Python Fundamentals” course @ SoftUni](https://softuni.bg/opencourses/python-fundamentals-course). Submit your solutions in the SoftUni judge system at <https://judge.softuni.bg/Contests/1088/Dictionaries-Exercises>.

## Key-Key Value-Value

Write a program, which searches for a **key** and **value** inside of several **key-value** pairs.

### Input

* On the **first** line, you will receive a **key**.
* On the **second** line, you will receive a **value**.
* On the **third** line, you will receive **N**.
* On the next **N** lines, you will receive strings in the following format:

“key => {value 1};{value 2};…{value X}”

After you receive **N** **key -> values** pairs, your task is to go through them and print **only** the **keys**, which contain the **key** and the **values**, which contain the **value**. Print them in the following format:

|  |
| --- |
| **{key}**:  -**{value1}**  -**{value2}**  …  -**{valueN}** |

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| bug  X  3  invalidkey => testval;x;y  debug => XUL;ccx;XC  buggy => testX;testY;XtestZ | debug:  -XUL  -XC  buggy:  -testX  -XtestZ |
| key  valu  2  xkeyc => value;value;valio  keyhole => valuable;x;values | xkeyc:  -value  -value  keyhole:  -valuable  -values |

in\_key = input()  
in\_val = str(input())  
n = int(input())  
  
for item in range(1,n+1):  
 dict ={}  
  
 data = input()  
 splited\_data = data.split(" => ")  
 kkey = splited\_data[0]  
 vvalue = splited\_data[1].split(";")  
  
 dict[kkey] = []  
 dict[kkey].extend(vvalue)  
  
 for key in dict.keys():  
 if str(in\_key) in str(key):  
 print(f"{key}:")  
 for item in dict[key]:  
 if in\_val in str(item):  
 print(f"-{item}")

## Travel Company

Write a program, which categorizes information about a travel company.

Companies have various vehicles planned for different cities. Every city has prepared several **types of vehicles**. Each vehicle type has a different **capacity**.

Until you receive the command “ready”, you will receive all the **cities** the company offers holiday packages for, and their respective **vehicle types** + **capacities** in the format:

* “{city}:{type}-{capacity},{type}-{capacity}…”

An example city with its transportation options would look like this:

* “Athens:bus-40,airplane-300,train-150”

If a city is entered a **second time**, add all transport which **hasn’t already been added** and **replace** **existing** transports’ capacities with the **new** ones.

After the “ready” command, you will start receiving **groups** for various cities in the format: “{city} {peopleCount}” until you receive “travel time!”.

After that, calculate whether the **group** will have **enough seats** to accommodate the passengers and print the status per these conditions:

If the group’s size is **smaller than or equal to** the **combines seats** in all the vehicles, print:

* “{city} -> all {peopleCount} accommodated”

If the group’s size is **larger than** the **combines seats** in all the vehicles, print:

* “{city} -> all except {peopleCount - transportCapacities} accommodated”

### Constraints

* There will be **no redundant whitespaces** anywhere in the input.
* The input will **always** be in the **format specified**.
* The **group cities** will **always** be **existing** cities.
* The **group sizes** will **always** be **positive**.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| Athens:bus-40,airplane-300,train-150  Athens:minibus-8,airplane-350  Warsaw:bus-30,train-150,airplane-120  ready  Athens 400  Warsaw 500  travel time! | Athens -> all 400 accommodated  Warsaw -> all except 200 accommodated |

|  |  |
| --- | --- |
| **Input** | **Output** |
| Sofia:bus-30,airplane-150,train-25  Istanbul:minibus-6,train-80  Sofia:bus-45  Sofia:bus-50  Berlin:airplane-120  ready  Berlin 115  Istanbul 200  Sofia 200  travel time! | Berlin -> all 115 accommodated  Istanbul -> all except 114 accommodated  Sofia -> all 200 accommodated |

data = input()  
  
city\_capacity = {}  
while data != "ready":  
 list\_data = data.split(":")  
 city = list\_data[0]  
 list\_transport = list\_data[1].split(",")  
  
 if city not in city\_capacity:  
 city\_capacity[city] = {}  
 for item in list\_transport:  
 transport = item.split("-")[0]  
 capacity = item.split("-")[1]  
 city\_capacity[city][transport] = capacity  
  
 data = input()  
  
data = input()  
  
while data != "travel time!":  
 city = data.split()[0]  
 people = int(data.split()[1])  
 result = 0  
 for item in city\_capacity[city]:  
 result += int(city\_capacity[city][item])  
  
 if result >= int(people):  
 print(f"{city} -> all {people} accommodated")  
 else:  
 print(f"{city} -> all except {people - result} accommodated")  
  
 data = input()

## Dict-Ref-Advanced

Remember the Dict-Ref Problem from the previous exercise? Well this one is an Advanced Version.

You will begin receiving input lines containing information in one of the following formats:

* {key} -> {value 1, value 2, …, value n}
* {key} -> {otherKey}

The **keys** will always be **strings**, and the **values** will always be **integers**, **separated** by a **comma** and a **space**.

If you are **given a key** and **values**, you must **store** the **values** to the **given key**. If the **key** already **exists**, you must **add** the **given values** to the old ones.

If you are **given a key** and **another key**, you must **copy** the **values** of the **other key** to the **first one**. If the **other key** **does not exist**, this input line must be **IGNORED**.

When you receive the command “**end**”, you must stop reading input lines, and you must print all keys with their values, in the following format:

* {key} === {value1, value2, value3. . .}

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| Isacc -> 5, 4, 3  Peter -> 6, 3, 3  Derek -> 2, 2, 2  End | Isacc === 5, 4, 3  Peter === 6, 3, 3  Derek === 2, 2, 2 |
| Donald -> 2, 2, 2  Isacc -> 1  George -> John  John -> Isacc  End | Donald === 2, 2, 2  Isacc === 1  John === 1 |

data = input()  
people = {}  
while data != "End":  
  
 splited\_data = data.split(" -> ")  
 name = splited\_data[0]  
  
 if name not in people:  
 people[name] = []  
  
 if ',' in splited\_data[1]:  
 digits = splited\_data[1].split(", ")  
  
 for item in digits:  
 people[name].append(int(item))  
 else:  
 if splited\_data[1].isdigit():  
 people[name].append(int(splited\_data[1]))  
 elif splited\_data[1] in people:  
 people[name].extend(people[splited\_data[1]])  
 else:  
 continue  
  
 data = input()  
  
for name in people.items():  
 if len(people[name[0]])>0:  
 print(f"{name[0]} === ", end="")  
 print(\*name[1],sep=", ")

## Forum Topics

You have been tasked to store a few forum topics, and filter them by several given tags.  
You will be given several input lines, containing data about topics in the following format:

{topic} -> {tag1, tag2, tag3...}

**The topic and tags will be strings. They will NOT contain spaces or ‘**-**’, ‘**>**’ symbols.**

**If you receive an existing topic, you must add the new tags to it. There should be NO duplicate tags.**

**When you receive the command “**filter**”, you must end the input sequence. On the next line (after “**filter**”), you will receive a sequence of tags, separated by a comma and a space. You must print ONLY those topics, which contain all tags in the given sequence.**

**The topics must be printed in the following format:**{topic} | {#tag1, #tag2, …, #tagN}

**NOTE: The tags have a number sign (‘**#**’) as a prefix.**

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| HelloWorld -> hello, forum, topic  HelpWithHomework -> homework, forum, topic  filter  forum, topic | HelloWorld | #hello, #forum, #topic  HelpWithHomework | #homework, #forum, #topic |
| First -> this  First -> that  First -> who  Second -> this, what, why  First -> this  Third -> this, third  Third -> that  filter  that, this | First | #this, #that, #who  Third | #this, #third, #that |

forum = {}  
  
while True:  
 user\_input = input()  
 if user\_input == "filter":  
 break  
  
 user\_input\_args = user\_input.split(" -> ")  
  
 topic = user\_input\_args[0]  
  
 if topic not in forum:  
 forum[topic] = []  
  
 tags = user\_input\_args[1].split(", ")  
  
 for t in tags:  
 if t in forum[topic]:  
 continue  
 else:  
 forum[topic].append(t)  
  
user\_tags = input().split(", ")  
  
def if\_any(topic,tags):  
 for tag in tags:  
 if tag not in forum[topic]:  
 return False  
 else:  
 continue  
 return True  
  
  
for topic in forum:  
 if if\_any(topic,user\_tags):  
 print(f"{topic} | #"+", #".join(forum[topic]))

## ****Social Media Posts****

You have been tasked to create a Console Social Media Database.

You will receive several input lines in one of the following formats:

* post {postName}
* like {postName}
* dislike {postName}
* comment {postName} {commenterName} {content}

If you receive the **post** command, you must **create** a **post** with the **given name**.

If you receive the **like** command you must **add** a **like** to the **given post**.

If you receive the **dislike** command you must add a **dislike** to the **given post**.

If you receive the **comment** command, you must **add** a **comment** to the **given post**. The **comment’s** **writer** must be the **given commentator name**, and the **content** of the **comment** must be the **given content**.

By default, the posts have **0 likes**, **0 dislikes** and **0 comments** when created.

When you receive the command “drop the media”, you must end the input sequence, and you must print **every post** with its **likes**, **dislikes** and **comments** in the following format:

|  |
| --- |
| Post: {postName} | Likes: {likes} | Dislikes {dislikes} Comments: \* {commentator1}: {comment1} \* {commentator2}: {comment2} . . . |

If a certain **post** does **not** have **any** comments, you should just print “**None**”.

The comments have a **prefix** of a **single** **asterisk** (‘**\***’) and **2 spaces**.

There is **NO space** between the **commentator’s name** and the **colon**.

### Constraints

* The **post name** will be a **string** of **letters** and **digits**.
* The **commentator’s name** will be a **string** of **letters**.
* The comment’s **CONTENT**, may contain **ANY ASCII** character.
* There will be **NO invalid** input data.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| post FirstPost  like FirstPost  like FirstPost  dislike FirstPost  post SecondPost  comment FirstPost Isacc Cool  comment SecondPost Isacc Lol  like SecondPost  drop the media | Post: FirstPost | Likes: 2 | Dislikes: 1  Comments:  \* Isacc: Cool  Post: SecondPost | Likes: 1 | Dislikes: 0  Comments:  \* Isacc: Lol |
| post SomePost  like SomePost  like SomePost  dislike SomePost  post OtherPost  comment OtherPost Isacc Naaais  comment OtherPost Peter GoodPost  comment OtherPost John Meh...  drop the media | Post: SomePost | Likes: 2 | Dislikes: 1  Comments:  None  Post: OtherPost | Likes: 0 | Dislikes: 0  Comments:  \* Isacc: Naaais  \* Peter: GoodPost  \* John: Meh... |

data = input()  
  
posts = {}  
  
while data != "drop the media":  
 command = data.split(" ")[0]  
 post\_name = data.split(" ")[1]  
  
 if command == "post":  
 posts[post\_name] = {"Likes": 0, "Dislikes": 0, "Comments": {}}  
  
 elif command == "like":  
 posts[post\_name]["Likes"] += 1  
  
 elif command == "dislike":  
 posts[post\_name]["Dislikes"] += 1  
 elif command == "comment":  
 author = data.split(' ')[2]  
 comment = data.split(' ')[3:]  
 posts[post\_name]["Comments"].update({author: comment})  
  
 data = input()  
  
for post\_name, social\_media\_data in posts.items():  
 print(f"Post: {post\_name} ", end='')  
 for key, value in social\_media\_data.items():  
 if key == "Comments":  
 print()  
 print(f"Comments:")  
 if value:  
 for author, comment in value.items():  
 print(f"\* {author}: {' '.join(comment)}")  
 else:  
 print("None")  
 else:  
 print(f"| {key}: {value} ", end='')