[1. Python advanced 1](#_Toc147401194)

[1.1 6. Paint Colors - Exercise: Stacks, Queues, Tuples, and Sets 1](#_Toc147401195)

[1.2 6. Summation Pairs - 02\_Tuples\_and\_Sets 3](#_Toc147401196)

# **Python advanced**

## 6. Paint Colors - Exercise: Stacks, Queues, Tuples, and Sets

G:\Other computers\My Computer\1-Training\1\_Projects\1st\_Project\03\_Advanced\03\_Stacks\_Queues\_Tuples\_and\_Sets\_Exercise\Recapitulate\Exercises\06\_paint\_colors\_c.py

**6. Paint Colors**

*You will have to find all possible color combinations that can be used.*

Write a program that finds colors in a string. You will be given a string on a **single line** containing **substrings** (separated by a **single space**) from which you will be able to form the following colors:

Main colors: **"red"**, **"yellow"**, **"blue"**

Secondary colors: **"orange"**, **"purple"**, **"green"**

To form a color, you should concatenate the **first** and the **last** **substrings** and check if you can get **any** of the **above colors'** **names**. If there is **only one substring left**, you should **use it** to do the same check.

You can only **keep** a **secondary color** if the **two main colors needed** for its creation could be **formed from the given substrings**:

* **orange** **= red + yellow**
* **purple = red + blue**
* **green = yellow + blue**

**Note:** You could find some of the main colors needed to keep a secondary color **after** it is found.

When you form a color, **remove both** substrings. Otherwise, you should **remove the last character** of **each** substring and **return** them in the **middle** of the **original string**. If the string contains an **odd number of substrings**, you should put the substrings **one position ahead**.

For example, if you are given the string **"re yellow bye"** you could not form a color with the substring **"re"** and **"bye"**, so you should remove the last character and return them in the middle of the string: **"r by yellow"**.

In the end, **print out the list with colors** in the order in which they are found.

### Input

* Single line **string**

### Output

* The **list** with the collected colors

### Constrains

* You will not receive an empty string
* Please consider only the colors mentioned above
* There won't be any cases with repeating colors

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| d yel blu e low redd | ['yellow', 'blue', 'red'] |
| **Comment** | |
| First, we take "**d"** and "**redd"**. After combining those substrings, we don't get any of the needed colors, so we remove the last characters from both substrings and return them in the middle of the original string, and it becomes "**yel blu red e low"**.  After that, we take "**yel"** and "**low"** so the first color we add to our list is yellow, and the string we are searching in looks as follows: "**blu red e"**  Then we take "**blu"** and "**e"**, and since this color is one of the searched ones (blue), we add it to our collection, and the state of the string is now "**red"**.  We should take the last substring and check if it matches some of the colors, and since it does, we add it (red) to our colors collection.  Finally, we print all the colors found: yellow, blue, and red in the format shown above. | |
| **Input** | **Output** |
| r ue nge ora bl ed | ['red', 'blue'] |
| **Comment** | |
| We don't keep orange because we don't have yellow in the final list with colors (combining red and yellow gives us orange). | |
| **Input** | **Output** |
| re ple blu pop e pur d | ['red', 'purple', 'blue'] |

## 6. Summation Pairs - 02\_Tuples\_and\_Sets

W:\1\_Python\1-Training\1\_Projects\1st\_Project\03\_Advanced

\02\_Tuples\_and\_Sets\Lab\6\_summation\_pairs.py

**6. Summation Pairs**

***The task is not included in the Judge system.***

On the first line, you will receive a sequence of **numbers** separated by space. On the second line, you'll receive a **target** number. Your task is to **find** the **pairs of numbers** whose **sum** **equals** the **target number**. For each found pair print **"{number} + {number} = {target\_number}"**. You may **NOT** use the **same element twice to fulfill the condition above**.

Can you come up with an algorithm that has less time complexity?

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 1 5 4 2 2 3 1 3 2  4 | 1 + 3 = 4  1 + 3 = 4  2 + 2 = 4 |
| 11 8 5 6 9 2 9 7 3 4  11 | 8 + 3 = 11  5 + 6 = 11  9 + 2 = 11  7 + 4 = 11 |