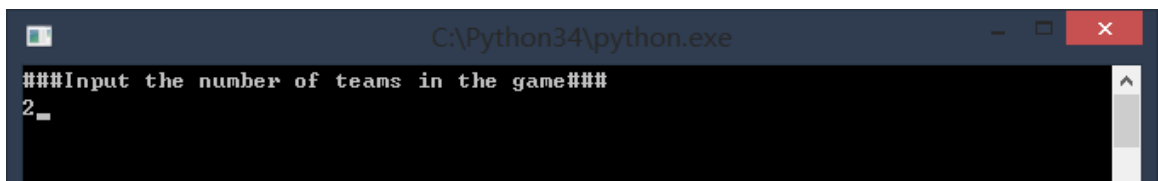


## Discussion of Bing Gan's solution to the test problem

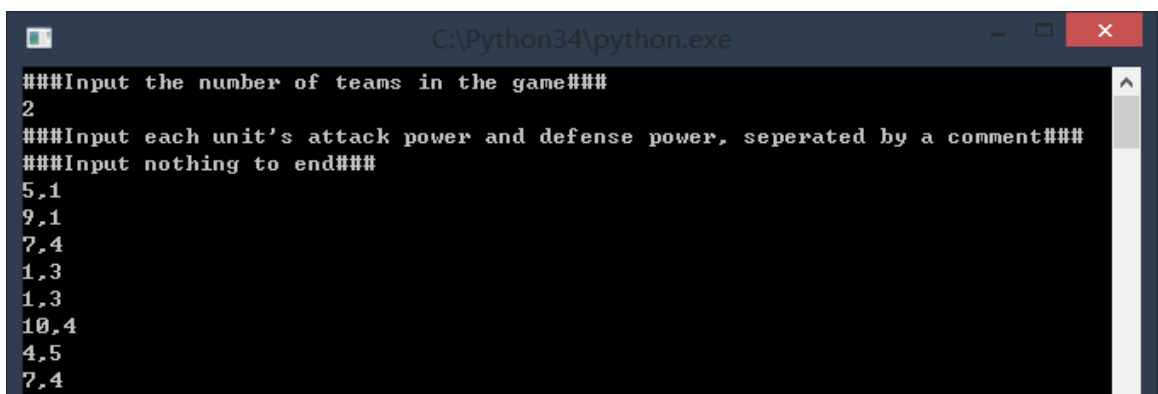
### Solution Description:

- **Language:** Python
- **Input:** keyboard (stdin)
- **Output:** Output Console
- **Assumptions:** the user who uses it knows the format of input. That is, there will not be unexpected inputs that cause the program crash.
- To use this solution, you first need to input the number of teams in the game.



```
C:\Python34\python.exe
###Input the number of teams in the game###
2_
```

Then you need to input the units, with each unit's attack power and defense power separated by a comment. End by hit enter without any input.



```
C:\Python34\python.exe
###Input the number of teams in the game###
2
###Input each unit's attack power and defense power, seperated by a comment###
###Input nothing to end###
5,1
9,1
7,4
1,3
1,3
10,4
4,5
7,4
```

It then shows the result. The result contains two parts: how it assigns the units into N groups and how strong each group is.

```

C:\Python34\python.exe
###Input the number of teams in the game###
2
###Input each unit's attack power and defense power, seperated by a comment###
###Input nothing to end###
5,1
9,1
7,4
1,3
1,3
10,4
4,5
7,4

final result:
[[5, 7, 0, 4], [1, 2, 6, 3]]
with attributes:
[[23, 12], [21, 13]]
Press any key to continue . . .

```

There are two internal override parameters in the program. Uncomment them to have easy testing without input the data manually.

```

23
24 #override parameters for easy testing
25 #list_units = [[5,1],[9,1],[7,4],[1,3],[1,3],[10,4],[4,5],[7,4]]
26 #team_count = 2
27
28 #list_units = [[6,10],[10,10],[4,8],[2,4],[8,4],[6,4],[8,2],[10,2],[8,8],[8,6],[2,10],[4,10],[10,6],[8,10],[4,6]]
29 #team_count = 3
30

```

### ● How the solution works:

1. read in the inputs from stdin.
2. If each group cannot have same amount of units (  $\text{Total\_unit\_amount} / \text{team\_amount} \neq \text{Integer}$  ), we fill up the units with 0 Attack power, 0 Defense power and index -1.
3. Sort the units by their Attack power from largest to smallest.
4. Then we start assigning. First, we put the largest N units out of the pool and put each of them into each group. We get the difference of each group's attack power and defense power compared to the first group (dA1, dA2, dA3, ...; dD1, dD2, dD3, ...).
5. Sum up the difference of attack power of each group and the defense power as well. We get sum\_A and sum\_D. We compare them and decide which attribute needs to be fixed more.
6. Knowing the type we are fixing, we sort the groups' attribute by the selected type and get the order of next assigning.
7. After this, we entering a loop of assigning. Each time, we take out N units ( $U_1, U_2, U_3 \dots U_N$ ) where N is the number of teams in the game. We put each of them into each group and get the new attack power and defense power of each group with the order we get from previous assign. We then get dA s and dD s again. By summing them up, we get our new sum\_A and sum\_D. We decide the type we are going to use next time and get the new order.
8. Finally, we get rid of the temporary units [0, 0, -1]s and output the result.

### Solution Analysis

- This solution works fine in the 10 trials with different group amount (2-6) and random attributes (0-10). It takes the largest numbers at first to create a potential large gap for the smaller units to

fit in. Although the groups' attributes are not absolute equal to each other, it's totally fine with a difference less than  $\pm 5\%$  of the total attribute sum since there is a random factor for each attributes when the groups are taking combat. For example, suppose we have a group with A/D (32,38) and another group with A/D (32,32). It looks like group 2 is absolutely weaker than group 1. However, there still is 43.4% chance that group 2 can beat group 1, which is quite an even possibility for either of the two group to win the combat.

- The limitation of my solution is that it cannot pass through extreme situations (i.e.: we have a super strong "Boss-leveled" unit with its attack power and defense power  $\gg$  the rest.) My solution works well when the unit group can be assigned equally not only their quality (attack power and defense power) but also quantity (Each group has similar amount of units where the difference is  $\pm 1$ ). The reason I still use this method is that I think for most situation in Games, we want the players/units amount to be close. Otherwise it will be a "Boss-stage" (One unit  $\gg$  rest) which we don't really want to assign the units by the system. Or if it's a PvP game, one of the players will get the big guy and the others will have number advantages. According to the combat rule I was offered, the extreme situation messed it up since either the rest will have "free hits" where the number advantage is huge, or the rest still need to hit the big guy where the big guy becomes both the Tank and the Nuke of the team and can swept out a large amount of enemies. It will need a more complicated combat rule (i.e. including taunt rules) for situations that have different unit amounts in different groups.
- I chose python as the programming language since I'm not so proficient in C/C++ and it may take more time to run an Objective-C code for you to test. Python is easy to run and use and it has a large amount of interface that it can almost be used on projects with any language.
- For further information of the solution (i.e.: trial data I used in the test) , please contact me at [co2.gerlaic@gmail.com](mailto:co2.gerlaic@gmail.com) / (206)-209-7523
- Hope you like my solution.