

```

def equationformat(equation):
    equation += ' '
    tempside = []
    tempcompound = []
    result = []
    lastchar = ''
    num = ''
    for i in equation:
        if not (i == ' ' or i == '+'):
            tempcompound += [i]
        if i == ' ' and tempcompound != []:
            tempcompound += ' '
            tempside += [tempcompound]
            tempcompound = []
    for i in tempside:
        for q in i:

            if (q.isalpha() or q == ' ') and num != '':
                tempcompound += ([lastchar] * (int(num)-1))
                num = ''

            if q.isalpha():
                tempcompound += [q]
                lastchar = q

            elif q.isnumeric():
                num += q

        result += [tempcompound]
        tempcompound = []
    tempside = []
    return result

```

This function puts the equation that was input into the format that the program can read. By iterating through the characters of the equation and stopping at spaces and numbers, it turns things like this “H2O + CO2” into [[H,H,O], [C,O,O]]. This is a list, which is very easy for the computer to read.

```

def balancetest(list1,list2):
    rawlist1 = []
    rawlist2 = []
    for i in list1:
        for q in i:
            rawlist1 += [q]
    for i in list2:
        for q in i:
            rawlist2 += [q]
    rawlist1.sort()
    rawlist2.sort()
    return (rawlist1 == rawlist2)

```

This function checks to see if the equation is balanced. It does this in two steps. First, it ignores how the elements are split up, changing “[H,H,O],[C,O,O]” to “[H,H,O,C,O,O].” Then it sorts them alphabetically, making it [C,H,H,O,O,O]. It does this to both sides of the equation. It then gives you a true or false answer for whether or not the sides are the same thing.

```

def baseconverter(value,baseX,digits):
    dividend = value
    result = []
    for i in range(digits):
        result += [dividend%baseX]
        dividend //= baseX
    return result

```

This function is used when iterating through the possibilities to balance the equation. It converts an integer into a list of numbers. This list of numbers is in whatever base you want. For example, I could ask, “23 in Base 12” and it would give me [1,11], since $12*1 + 1*11 = 23$. I need this because I have to be able to select how many possibilities I want to iterate through, and I assign each ‘digit’ to a different compound when balancing the equation.

```

while not done:
    side2 = equationformat(inputequation[inputequation.index('=')+1:])
    side1 = equationformat(inputequation[:inputequation.index('=')])

    componentcount = len(side1) + len(side2)
    balancefound = False

```

The actual code starts off by setting some variables. Next, it has a couple lines that warn the user if the equation inputted will take unreasonably long.

```

if len(str(inputdepth ** componentcount)) > 9 and not unsolvable:
    if len(str(4 ** componentcount)) > 9:
        print('Unfortunately, the program will take an unfeasible amount of time to solve this.')
        done = True
        unsolvable = True
    else:
        inputdepth = int(input('Unfortunately, the program will take a long time to solve this. Tr

```

As long as the code doesn't take a unreasonable amount of time to solve, the program moves on to the next part of the loop.

```

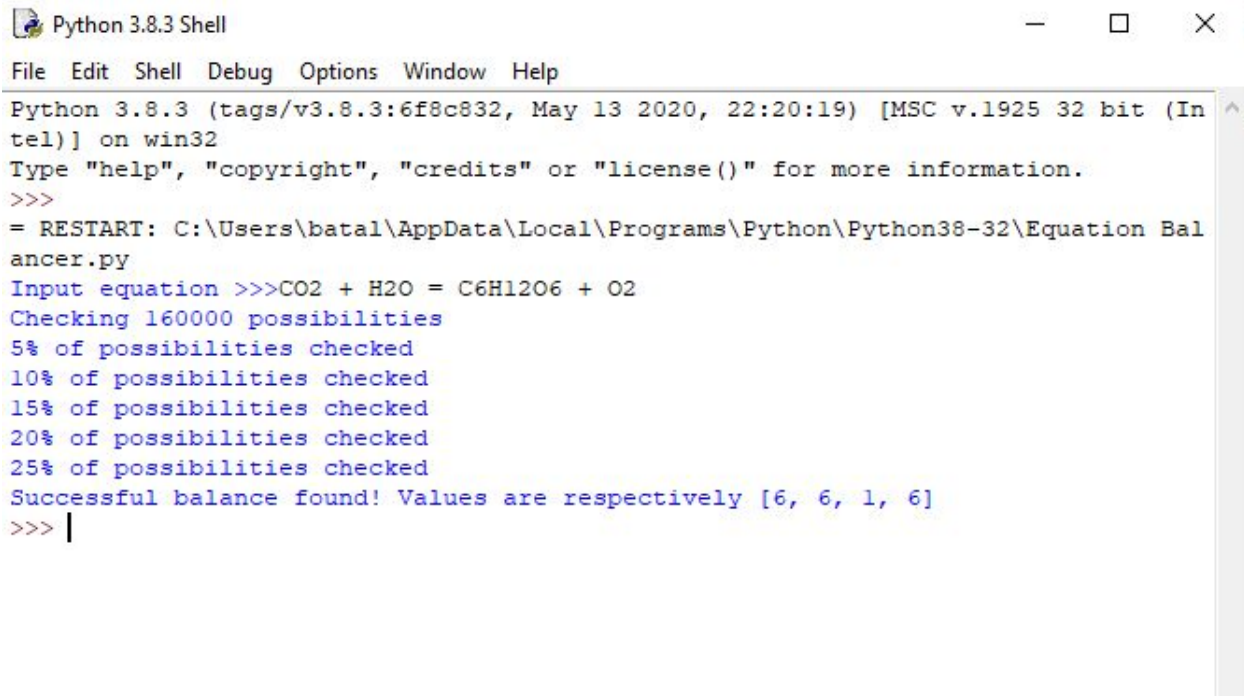
else:
    i=0
    print('Checking ' + str(inputdepth ** componentcount) + ' possibilities')
    while (not balancefound) and (i < inputdepth ** componentcount):
        i += 1
        if (i/(inputdepth ** componentcount))*100 > percent + 5:
            percent += 5
            print(str(percent) + '% of possibilities checked')
        sidelclone = []
        side2clone = []
        baseXiteration = baseconverter(i-1,inputdepth,componentcount)
        for z in range(len(baseXiteration)):
            baseXiteration[z] += 1
        for q in range(componentcount):
            if q+1 <= len(sidel):
                sidelclone += [sidel[q]] * baseXiteration[q]
            else:
                side2clone += [side2[q-(len(sidel))]] * baseXiteration[q]
        if balancetest(sidelclone,side2clone):
            balancefound = True

if balancefound == True:
    print("Successful balance found! Values are respectively " + str(baseXiteration))
    done = True
else:
    inputdepth = int(input("No balance found. Try increasing the maximum ratio. \nThe default i

```

This code does three things. First, it generates a combination of numbers to try balancing with, and it assigns those numbers to the equation. Second, it checks if the equation is balanced with the new numbers it just assigned. Thirdly, it prints the % of possibilities that have been checked.

Here are a couple of screenshots of the code in action.



```
Python 3.8.3 Shell
File Edit Shell Debug Options Window Help
Python 3.8.3 (tags/v3.8.3:6f8c832, May 13 2020, 22:20:19) [MSC v.1925 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:\Users\batal\AppData\Local\Programs\Python\Python38-32\Equation Balancer.py
Input equation >>>CO2 + H2O = C6H12O6 + O2
Checking 160000 possibilities
5% of possibilities checked
10% of possibilities checked
15% of possibilities checked
20% of possibilities checked
25% of possibilities checked
Successful balance found! Values are respectively [6, 6, 1, 6]
>>> |
```