C200 Programming Assignment № 7
©Dalkilic 2022

Dr. M.M. Dalkilic
Computer Science
School of Informatics, Computing, and Engineering

Indiana University, Bloomington, IN, USA

October 29, 2022

Due Date: 10:59 PM, Saturday, November, 05, 2022
Submit your work **using** the new autograder by the deadline and remember to add, commit and push to Github. Autograder Link: `http://c200.luddy.indiana.edu`. This homework is shorter, but requires some thought–**please start it early**. Because of the way pygame initializes the program, you'll need to work on it **after** you plot the population data. Otherwise, you'll need to remove it or restart the IDE.

You can choose your programming partner (up-to one programming partner), but write their name in the a7.py file. We have created a comment for that at the top of the file.

After completing problem-6, please remember to comment your code otherwise the Autograder will return an error. For problme-6, we will manually run the code to see if the code correctly draws the trainagles.

In problem-3, once you have solved the problem, you can uncomment the code that draws the plot in Figure-2, (line 226-236 in the starter code), once you are done drawing the plot, please comment that code to prevent any errors in the Autograder. The plot in Figure-2 is given as additional information for your own understanding, but it's not needed to solve the problem.

Note: Please ensure that you debug the code well (syntax, logical and implementation errors), before submitting to the Autograder.

Problem 1: Recursion & Sigma notation

The binomial coefficient is used in almost every area that involves computation. You've implementmed choice last homework, so we're including it here. A recurrence that uses both choice and sigma is shown below:

$$\binom{n}{0} = \binom{n}{n} = 1, \text{for all } n \geq 0 \tag{1}$$

$$\binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1} \tag{2}$$

$$B_0 = 1 \tag{3}$$

$$B_n = \frac{-\sum_{k=0}^{n-1} \binom{n+1}{k} B_k}{n+1} \tag{4}$$

Here are some outputs:

output

```
B(0) = 1
B(1) = -0.5
B(2) = 0.16666666666666666
B(3) = -0.0
B(4) = -0.033333333333333305
B(5) = -7.401486830834377e-17
```

Programming Problem 1: Recursion

- Complete B() using recursion. You should use the C() function which we have already completed for you to complete B() function.
- You can use sum(). This Python function sums a container of numbers:

```
>>> sum((1,2,3))
6
>>> sum({1,2,3})
6
>>> sum([1,2,3])
6
```

Problem 2:Recursion, Tail-recursion, and Generators

In this problem you'll be implementing recursive functions, tail-recursive functions, and generators. Please pay attention to what the problem asks–not every problem requires all implementations. Some tail-recursive and while-loops must build bottom-up. As a note, a functions can be made a generator function if it uses the **yield** keyword.

The first function is a():

$$
\begin{aligned}
a(n) &= 0 \quad n \leq 0 && (5)\\
a(1) &= 3 && (6)\\
a(2) &= 5 && (7)\\
a(n) &= a(n-1) + a(n-2) + a(n-3) && (8)
\end{aligned}
$$

You'll implement four functions:

- a(n) recursively
- aw(n) using a while-loop
- at(n) tail-recursive
- and a_gen() as generator

When run:

```
for i,j in zip(range(10),a_gen()):
        print(f"a({i}) {a(i)} {aw(i)} {at(i)} {j} \n",end="")

a(0) 2 2 2 2
a(1) 3 3 3 3
a(2) 5 5 5 5
a(3) 10 10 10 10
a(4) 18 18 18 18
a(5) 33 33 33 33
a(6) 61 61 61 61
a(7) 112 112 112 112
a(8) 206 206 206 206
a(9) 379 379 379 379
```

The next function is bb(n)

$$
\begin{aligned}
bb(0) &= 2 && (9)\\
bb(1) &= -3 && (10)\\
bb(x) &= bb(x-1)bb(x-2) && (11)
\end{aligned}
$$

You'll implement four functions:

- bb(n) recursively
- bbw(n) using a while-loop
- bbt(n) tail-recursive

- and bb_gen() as generator

When run:

```
for i,j in zip(range(10),bb_gen()):
        print(f"b({i}) {bb(i)} {bbw(i)} {bbt(i)} {j}")

b(0) 2 2 2 2
b(1) -3 -3 -3 -3
b(2) -6 -6 -6 -6
b(3) 18 18 18 18
b(4) -108 -108 -108 -108
b(5) -1944 -1944 -1944 -1944
b(6) 209952 209952 209952 209952
b(7) -408146688 -408146688 -408146688 -408146688
b(8) -85691213438976 -85691213438976 -85691213438976 -85691213438976
b(9) 34974584955819144511488 34974584955819144511488 34974584955819144511488 34974584955819144511488
```

The next function is F(n,m,p):

$$\begin{aligned} F(n,m,0) &= 100+n-m \qquad (12) \\ F(n,m,p) &= mn-p+F(n-3,m-2,p-1) \qquad (13) \end{aligned}$$

For this problem you'll implement

- F(n,m,p) recursively
- Fw(n,m,p) using a while-loop
- Ft(n,m,p) tail-recursive

When run (I'm giving you a lot of output so you have more ways to check), but it's so large, it will be found in Section Output. The next function is m(x,y)

$$m(x,y) = \begin{cases} 3 & x,y \leq 0 \\ 2 & x \leq 0 \\ 1 & y \leq 0 \\ m(x-1,y-1)+m(x-1,y-2) & otherwise \end{cases} \qquad (14)$$

This means that when both arguments are equal to or less than zero, then the function returns 3. If one of the arguments is equal to or less than zero, then the return value is either 2 (for $x \leq 0$) or 1 (for $y \leq 0$). The function can be implemented recursively as:

```
def m(x,y):
    if x <= 0 and y <= 0:
        return 3
    elif x <= 0:
```

```
        return 2
    elif y <= 0:
        return 1
    else:
        return m(x-1,y-1) + m(x-1,y-2)
```

Your task is to implement mw(x,y) as a while loop. You should create a dictionary and build the values up, and then ultimately return the value of the key you are looking for. I strongly encourage you to implement this as a dictionary. (**Note:** as a reminder, we did memo_factorial(n) in lab6, where we built a dictionary then returned a value from it, that may help to start thinking about this problem). See Fig. 1. I'll show the first part of my implementation:

```
def m(x,y):
    if x <= 0 and y <= 0:
        return 3
    elif x <= 0:
        return 2
    elif y <= 0:
        return 1
    else:
        return m(x-1,y-1) + m(x-1,y-2)

def mw(x,y):
    d = {(0,0):3,(0,1):2,(1,0):1}
...
```

To determine mw(x,y) you'll want to pull values directly from a dictionary you're building. The reason this is a different kind of while is that we have two arguments. So we have to build the dictionary using two loops.

The output is shown in (towards the end of the PDF).

Programming Problem 2: Recursion, Tail-Recursion, Generators

- Complete the functions above.
- The output for F and m are in the last section.

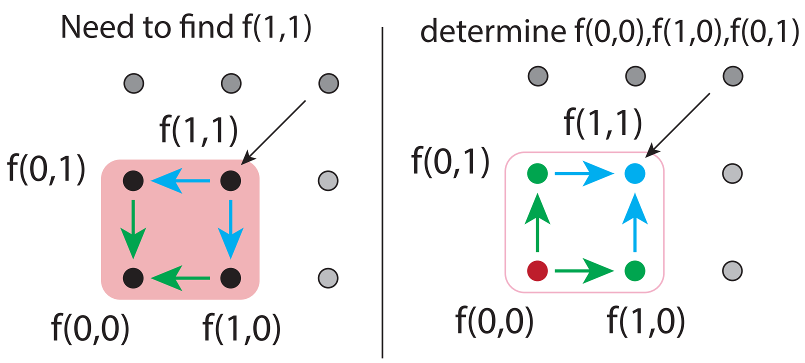

Figure 1: (left) In order to compute f(1,1), we must first compute f(1,0) and f(0,1). These both in turn require f(0,0). (right) We then add f(0,0), f(1,0),f(0,1) to the dictionary allowing us to calculate f(1,1).

Problem 3: Modeling Data and Judging Goodness of a Model

Years + 1900	Population $\times 10^6$
0	1650
10	1750
20	1860
30	2070
40	2300
50	2560
60	3040
70	3710
80	4450
90	5280
100	6080
110	6870

Table 1: Human Population from 1900-2010 in millions.

Exponential functions are a class of functions that find use in virtually every field. Here is the general form:

$$f(x) = a(b^x) \tag{15}$$

where a, b are constants. Exponential functions grow very fast–we can see it in real-life with the spread of COVID 19 infections. To get a better understanding of these kinds of functions, we'll look at the population growth of people. Table 1 has approximate values for a little over a century beginning with the earth's population of people in 1900s.

A proposed model of these data is:

$$pop(year) = 1436.53(1.01395)^{year} \tag{16}$$

Can we judge how good this model is? There are many ways, but the simplest is to check the difference between the value of the data and the value from the model for a particular year. This simply tells us how far away are the predictions of the model from the true population in that particular year. If the difference is not high then it means that the model is doing good otherwise if the difference is large then the model's predictions may not be that good as they tend to be far away from the actual population (as given in Table-1).

For example, in 1960, the data says the population was 3040×10^6. The first thing we can do to simplify this even more is to discard $\times 10^6$ and treat the population just as 3040. Similarly, we can subtract 1900 from the year (1960-1900), so the input to the function is 60.

We will write $p_{i\times 10}$ to indicate population for that year after subtracting 1900. For example, $p_{1900} = 1650 \times 10^6, p_{1910} = 1750 \times 10^6, p_{1920} = 1860 \times 10^6$ are $p_{0\times 10} = p_0 = 1650, p_{1\times 10} = p_{10} = 1750, p_{2\times 10} = p_{20} = 1860$. You can verify these from the table. Let's find the absolute value of the difference of the data p_{60} and our model $pop(60)$

$$|3040 - pop(60)| = |3040 - 3298.428492408121| \approx 258.4284924081212 \tag{17}$$

I'm using the value of the function I've written in Python–and kept the decimal places so you can replicate it–but it doesn't contribute significantly to the overall answer. The *closer* the answer is to zero, the better the model! We are interested in the average error–we'll sum the error and divide by the number of data points:

$$\begin{aligned} ave\_error &= \left(\frac{1}{n}\right)\sum_{i=0}^{11} |p_{i\times 10} - pop(i \times 10)| \quad (18) \\ &= (1/12)(|p_0 - pop(0)| + |p_{10} - pop(10)| + \ldots + |p_{110} - pop(110)|) \quad (19) \end{aligned}$$

where $p_{i\times 10}$ is the population (from the data) at year $1900 + (i \times 10)$ and n is the number of data points (we can use Python len() function).

When run (if you've uncommented the visualization also) you'll see:

```
    data = get_data(".", "pop.txt")
    print(f"data from file:",data)
    print(f"abs(3040 - {pop(60)}) = {abs(3040 - pop(60))}")
    ave_error = round(error(data),2)
    print(f"Average error: {ave_error}")

data from file: [(0, 1650), (10, 1750), (20, 1860), (30, 2070), (40, 2300), (50, 2560), (60, 3040), (70, 3710),
(80, 4450), (90, 5280), (100, 6080), (110, 6870)]
abs(3040 - 3298.428492408121) = 258.4284924081212
Average error: 191.68
```

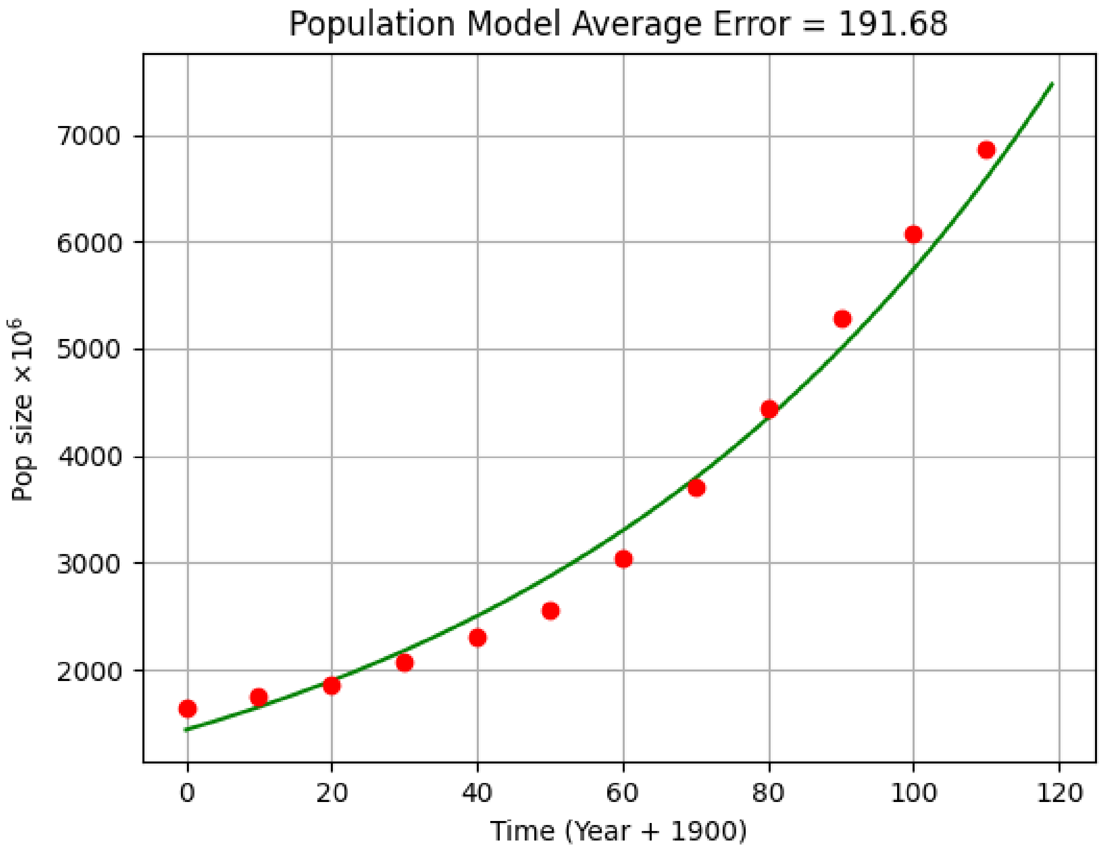

Figure 2: Plot of population growth data (blue dots) and model (green line).

Deliverables for Programming Problem 3

- Create a text file in Assignment7 directory, and name the file as **pop.txt**, the file must have the same data and format as Table 1. You can use any seperator, but I used a comma. The seperator is used to seperate the columns in the file.
- Read from the file you've created to get the data in the get_data(path, name) function.
- Complete the function pop(year) based on the equation provided on line 15, and complete the function error(data) based on the equation provided on line 17.
- Adapted and extended from, *Biocalculus*, by Steward and Day.
- Once you are done visualizing (the plot shown in Figure-2), please comment the code (line 226-236), othwerwise the Autograder will return error.

Problem 4: 11s

We saw a property of an integer when it's divisible by 9–the sum of the digits is also divisible by 9. Interestingly, we can determine if an integer is divisible by eleven with another trick. If you add and subtract the digits alternately and the result is either 0, 11, or -11, then it's divisible by 11. For example,

```
>>> 11*11
121
>>> 1 - 2 + 1
0
>>> 11*319
3509
>>> 3 - 5 + 0 - 9
-11
>>> 1429*11
15719
>>> 1 - 5 + 7 - 1 + 9
11
>>>
```

Also note that function div_11(n) should return True when n is divisible by 11, and False when it n is not divisble by 11.

Deliverables for Programming Problem 4

- Complete the function div_11().
- You *cannot* simply divide by 11 or use integer divide. We recommend using a loop that implements the algorithm as described above.

Problem 5: Sum Levels

This function sl(lst,p) takes a list of (possibly a list) of numbers and returns a list [[0,s_0],[1,s_1],...,[n,s_n]] where 0 is the depth of the list and s_i is the sum of numbers at that level. For example,

```
prob5 = [[1,4,[3,[100]],3,2,[1,[101,1000],5],1,[7,9]],
             [1,2,3,4,[10,20,30,40,[100,200,300,400]]],
             [[[[100,200,300,400]]],[[10,20,30,40]],1,2,3,4]]
    for lst in prob5:
        print(sl(lst))

[[0, 11], [1, 25], [2, 1201]]
[[0, 10], [1, 100], [2, 1000]]
[[0, 10], [1, 0], [2, 100], [3, 1000]]
```

because for the first list, [1,4,3,2,1] = 1+4+3+2+1 = 11 and is at the 0th level. For the 1st level we have [3,1,7,9] = 3+1+5+7+9 = 25. For the last list, there aren't any values at level 1, so the sum is zero.

Deliverables for Programming Problem 5

- Complete the function.
- Your choice whether to write recursively (which I did) or as a while loop.

Problem 6: Sierpinski Triangle

In this problem, you'll complete the pygames program that draws triangles recursively. When done you'll have this beautiful recursive drawn image:

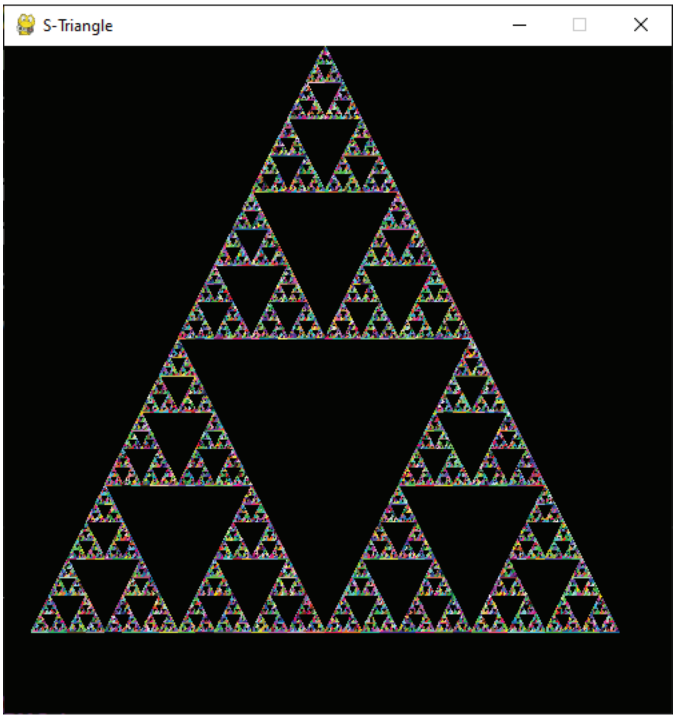

Figure 3: You experiment with the colors–perhaps making the larger triangles red and as they get smaller, darker red.

The **only** code you need to add is below:

```
def s(loc,width):
    if width > 1:
        x,y = loc
        z = math.sqrt(width**2 - (width/2))
        draw_triangle(loc,width)
        #draw three smaller triangles shown in figure
        #top where the original bigger triangle is but 1/2 smaller
        #left mid side 1/2 smaller
        #right mid side 1/2 smaller
```

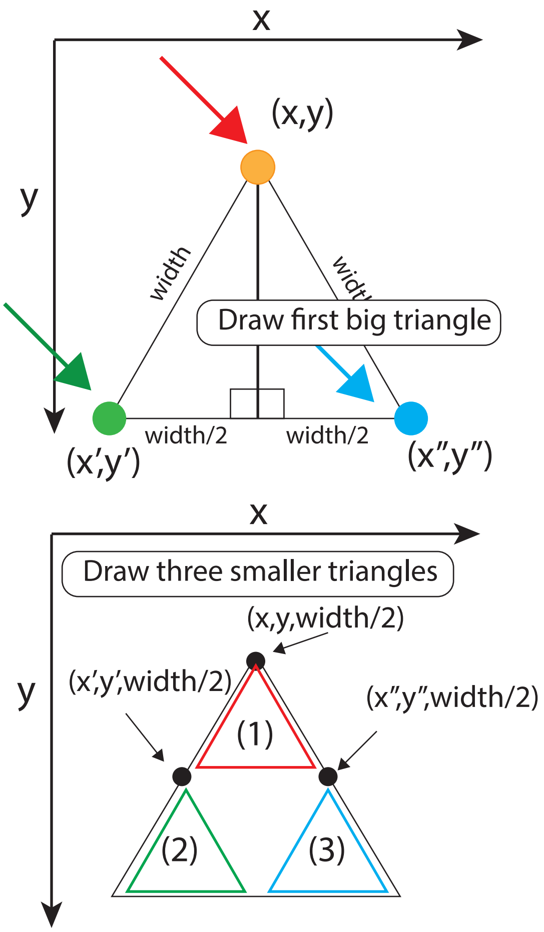

Figure 4: After drawing the big triangle (top), you draw three smaller triangles (1/2) the width at the three points shown. The top is the same location, but the other two you'll have to use your triangle to calculate the locations. I've made the triangle sizes a little smaller so you can see them better.

Deliverables for Programming Problem 6

- We have already given most code for this problem as part of starter code.
- Complete the s() function to draw the traingles as shown in the figure.
- Once you are done, make sure to comment the code for this problem to prevent errors in the Autograder. We will grade this problem separatly by running your code manually.

Output for Problem 2

```python
for i in range(6):
```

```
        for j in range(6):
            for k in range(6):
                print(f"{i,j,k} {F(i,j,k)} {Ft(i,j,k)} {Fw(i,j,k)}")

(0, 0, 0) 100 100 100
(0, 0, 1) 98 98 98
(0, 0, 2) 101 101 101
(0, 0, 3) 121 121 121
(0, 0, 4) 170 170 170
(0, 0, 5) 260 260 260
(0, 1, 0) 99 99 99
(0, 1, 1) 97 97 97
(0, 1, 2) 97 97 97
(0, 1, 3) 111 111 111
(0, 1, 4) 151 151 151
(0, 1, 5) 229 229 229
(0, 2, 0) 98 98 98
(0, 2, 1) 96 96 96
(0, 2, 2) 93 93 93
(0, 2, 3) 101 101 101
(0, 2, 4) 132 132 132
(0, 2, 5) 198 198 198
(0, 3, 0) 97 97 97
(0, 3, 1) 95 95 95
(0, 3, 2) 89 89 89
(0, 3, 3) 91 91 91
(0, 3, 4) 113 113 113
(0, 3, 5) 167 167 167
(0, 4, 0) 96 96 96
(0, 4, 1) 94 94 94
(0, 4, 2) 85 85 85
(0, 4, 3) 81 81 81
(0, 4, 4) 94 94 94
(0, 4, 5) 136 136 136
(0, 5, 0) 95 95 95
(0, 5, 1) 93 93 93
(0, 5, 2) 81 81 81
(0, 5, 3) 71 71 71
(0, 5, 4) 75 75 75
(0, 5, 5) 105 105 105
(1, 0, 0) 101 101 101
(1, 0, 1) 99 99 99
(1, 0, 2) 100 100 100
(1, 0, 3) 116 116 116
(1, 0, 4) 159 159 159
(1, 0, 5) 241 241 241
(1, 1, 0) 100 100 100
```

```
49  (1, 1, 1) 99 99 99
50  (1, 1, 2) 98 98 98
51  (1, 1, 3) 109 109 109
52  (1, 1, 4) 144 144 144
53  (1, 1, 5) 215 215 215
54  (1, 2, 0) 99 99 99
55  (1, 2, 1) 99 99 99
56  (1, 2, 2) 96 96 96
57  (1, 2, 3) 102 102 102
58  (1, 2, 4) 129 129 129
59  (1, 2, 5) 189 189 189
60  (1, 3, 0) 98 98 98
61  (1, 3, 1) 99 99 99
62  (1, 3, 2) 94 94 94
63  (1, 3, 3) 95 95 95
64  (1, 3, 4) 114 114 114
65  (1, 3, 5) 163 163 163
66  (1, 4, 0) 97 97 97
67  (1, 4, 1) 99 99 99
68  (1, 4, 2) 92 92 92
69  (1, 4, 3) 88 88 88
70  (1, 4, 4) 99 99 99
71  (1, 4, 5) 137 137 137
72  (1, 5, 0) 96 96 96
73  (1, 5, 1) 99 99 99
74  (1, 5, 2) 90 90 90
75  (1, 5, 3) 81 81 81
76  (1, 5, 4) 84 84 84
77  (1, 5, 5) 111 111 111
78  (2, 0, 0) 102 102 102
79  (2, 0, 1) 100 100 100
80  (2, 0, 2) 99 99 99
81  (2, 0, 3) 111 111 111
82  (2, 0, 4) 148 148 148
83  (2, 0, 5) 222 222 222
84  (2, 1, 0) 101 101 101
85  (2, 1, 1) 101 101 101
86  (2, 1, 2) 99 99 99
87  (2, 1, 3) 107 107 107
88  (2, 1, 4) 137 137 137
89  (2, 1, 5) 201 201 201
90  (2, 2, 0) 100 100 100
91  (2, 2, 1) 102 102 102
92  (2, 2, 2) 99 99 99
93  (2, 2, 3) 103 103 103
94  (2, 2, 4) 126 126 126
95  (2, 2, 5) 180 180 180
```

```
 96  (2, 3, 0) 99 99 99
 97  (2, 3, 1) 103 103 103
 98  (2, 3, 2) 99 99 99
 99  (2, 3, 3) 99 99 99
100  (2, 3, 4) 115 115 115
101  (2, 3, 5) 159 159 159
102  (2, 4, 0) 98 98 98
103  (2, 4, 1) 104 104 104
104  (2, 4, 2) 99 99 99
105  (2, 4, 3) 95 95 95
106  (2, 4, 4) 104 104 104
107  (2, 4, 5) 138 138 138
108  (2, 5, 0) 97 97 97
109  (2, 5, 1) 105 105 105
110  (2, 5, 2) 99 99 99
111  (2, 5, 3) 91 91 91
112  (2, 5, 4) 93 93 93
113  (2, 5, 5) 117 117 117
114  (3, 0, 0) 103 103 103
115  (3, 0, 1) 101 101 101
116  (3, 0, 2) 98 98 98
117  (3, 0, 3) 106 106 106
118  (3, 0, 4) 137 137 137
119  (3, 0, 5) 203 203 203
120  (3, 1, 0) 102 102 102
121  (3, 1, 1) 103 103 103
122  (3, 1, 2) 100 100 100
123  (3, 1, 3) 105 105 105
124  (3, 1, 4) 130 130 130
125  (3, 1, 5) 187 187 187
126  (3, 2, 0) 101 101 101
127  (3, 2, 1) 105 105 105
128  (3, 2, 2) 102 102 102
129  (3, 2, 3) 104 104 104
130  (3, 2, 4) 123 123 123
131  (3, 2, 5) 171 171 171
132  (3, 3, 0) 100 100 100
133  (3, 3, 1) 107 107 107
134  (3, 3, 2) 104 104 104
135  (3, 3, 3) 103 103 103
136  (3, 3, 4) 116 116 116
137  (3, 3, 5) 155 155 155
138  (3, 4, 0) 99 99 99
139  (3, 4, 1) 109 109 109
140  (3, 4, 2) 106 106 106
141  (3, 4, 3) 102 102 102
142  (3, 4, 4) 109 109 109
```

```
(3, 4, 5) 139 139 139
(3, 5, 0) 98 98 98
(3, 5, 1) 111 111 111
(3, 5, 2) 108 108 108
(3, 5, 3) 101 101 101
(3, 5, 4) 102 102 102
(3, 5, 5) 123 123 123
(4, 0, 0) 104 104 104
(4, 0, 1) 102 102 102
(4, 0, 2) 97 97 97
(4, 0, 3) 101 101 101
(4, 0, 4) 126 126 126
(4, 0, 5) 184 184 184
(4, 1, 0) 103 103 103
(4, 1, 1) 105 105 105
(4, 1, 2) 101 101 101
(4, 1, 3) 103 103 103
(4, 1, 4) 123 123 123
(4, 1, 5) 173 173 173
(4, 2, 0) 102 102 102
(4, 2, 1) 108 108 108
(4, 2, 2) 105 105 105
(4, 2, 3) 105 105 105
(4, 2, 4) 120 120 120
(4, 2, 5) 162 162 162
(4, 3, 0) 101 101 101
(4, 3, 1) 111 111 111
(4, 3, 2) 109 109 109
(4, 3, 3) 107 107 107
(4, 3, 4) 117 117 117
(4, 3, 5) 151 151 151
(4, 4, 0) 100 100 100
(4, 4, 1) 114 114 114
(4, 4, 2) 113 113 113
(4, 4, 3) 109 109 109
(4, 4, 4) 114 114 114
(4, 4, 5) 140 140 140
(4, 5, 0) 99 99 99
(4, 5, 1) 117 117 117
(4, 5, 2) 117 117 117
(4, 5, 3) 111 111 111
(4, 5, 4) 111 111 111
(4, 5, 5) 129 129 129
(5, 0, 0) 105 105 105
(5, 0, 1) 103 103 103
(5, 0, 2) 96 96 96
(5, 0, 3) 96 96 96
```

```
(5, 0, 4) 115 115 115
(5, 0, 5) 165 165 165
(5, 1, 0) 104 104 104
(5, 1, 1) 107 107 107
(5, 1, 2) 102 102 102
(5, 1, 3) 101 101 101
(5, 1, 4) 116 116 116
(5, 1, 5) 159 159 159
(5, 2, 0) 103 103 103
(5, 2, 1) 111 111 111
(5, 2, 2) 108 108 108
(5, 2, 3) 106 106 106
(5, 2, 4) 117 117 117
(5, 2, 5) 153 153 153
(5, 3, 0) 102 102 102
(5, 3, 1) 115 115 115
(5, 3, 2) 114 114 114
(5, 3, 3) 111 111 111
(5, 3, 4) 118 118 118
(5, 3, 5) 147 147 147
(5, 4, 0) 101 101 101
(5, 4, 1) 119 119 119
(5, 4, 2) 120 120 120
(5, 4, 3) 116 116 116
(5, 4, 4) 119 119 119
(5, 4, 5) 141 141 141
(5, 5, 0) 100 100 100
(5, 5, 1) 123 123 123
(5, 5, 2) 126 126 126
(5, 5, 3) 121 121 121
(5, 5, 4) 120 120 120
(5, 5, 5) 135 135 135
```

The output is:

```
or i in range(10):
        for j in range(10):
            print(f"{i,j} {m(i,j)} {mw(i,j)} ")

(0, 0) 3 3
(0, 1) 2 2
(0, 2) 2 2
(0, 3) 2 2
(0, 4) 2 2
(0, 5) 2 2
(0, 6) 2 2
```

```
(0, 7) 2 2
(0, 8) 2 2
(0, 9) 2 2
(1, 0) 1 1
(1, 1) 6 6
(1, 2) 5 5
(1, 3) 4 4
(1, 4) 4 4
(1, 5) 4 4
(1, 6) 4 4
(1, 7) 4 4
(1, 8) 4 4
(1, 9) 4 4
(2, 0) 1 1
(2, 1) 2 2
(2, 2) 7 7
(2, 3) 11 11
(2, 4) 9 9
(2, 5) 8 8
(2, 6) 8 8
(2, 7) 8 8
(2, 8) 8 8
(2, 9) 8 8
(3, 0) 1 1
(3, 1) 2 2
(3, 2) 3 3
(3, 3) 9 9
(3, 4) 18 18
(3, 5) 20 20
(3, 6) 17 17
(3, 7) 16 16
(3, 8) 16 16
(3, 9) 16 16
(4, 0) 1 1
(4, 1) 2 2
(4, 2) 3 3
(4, 3) 5 5
(4, 4) 12 12
(4, 5) 27 27
(4, 6) 38 38
(4, 7) 37 37
(4, 8) 33 33
(4, 9) 32 32
(5, 0) 1 1
(5, 1) 2 2
(5, 2) 3 3
(5, 3) 5 5
```

```
(5, 4) 8 8
(5, 5) 17 17
(5, 6) 39 39
(5, 7) 65 65
(5, 8) 75 75
(5, 9) 70 70
(6, 0) 1 1
(6, 1) 2 2
(6, 2) 3 3
(6, 3) 5 5
(6, 4) 8 8
(6, 5) 13 13
(6, 6) 25 25
(6, 7) 56 56
(6, 8) 104 104
(6, 9) 140 140
(7, 0) 1 1
(7, 1) 2 2
(7, 2) 3 3
(7, 3) 5 5
(7, 4) 8 8
(7, 5) 13 13
(7, 6) 21 21
(7, 7) 38 38
(7, 8) 81 81
(7, 9) 160 160
(8, 0) 1 1
(8, 1) 2 2
(8, 2) 3 3
(8, 3) 5 5
(8, 4) 8 8
(8, 5) 13 13
(8, 6) 21 21
(8, 7) 34 34
(8, 8) 59 59
(8, 9) 119 119
(9, 0) 1 1
(9, 1) 2 2
(9, 2) 3 3
(9, 3) 5 5
(9, 4) 8 8
(9, 5) 13 13
(9, 6) 21 21
(9, 7) 34 34
(9, 8) 55 55
(9, 9) 93 93
```