Problem Statement — A1

CPU Emulation

Computer programs are essentially a set of instructions that your CPU runs. Each instruction will manipulate the state of the machine; for example, one instruction might add two numbers together. The time to run an instruction can be measured in “cycles”. The more cycles a CPU can run, the more instructions it can execute.

Instruction ID	# of Cycles	Effect on Stored Integer
0	1	$x = x + 1$
1	2	$x = x * 2$
2	3	$x = x * 5$

Table 1. Instruction Set for the Wildcat 64

Listed above in Table 1 is the instruction set for the *Wildcat 64*’s CPU. Its CPU stores an integer in memory that is initialized to 0, and each instruction modifies the stored integer. The CPU is given a number of cycles to run, and it will execute instructions until it runs out of cycles. If the CPU tries to execute an instruction without sufficient cycles (i.e. executing instruction 2 with 1 cycle left), it will not execute the instruction and will terminate early.

Your task is to emulate the *Wildcat 64* machine by replicating the behavior described above. Your program should take the number of cycles to be emulated. Your program should then take the instruction id to execute until the number of cycles inputted has been exceeded. Finally, your program should output the stored integer after running the instruction(s). You may assume that the number of emulated instructions is between 1 and 25. You may also assume that the instruction id will always be 0, 1, or 2.

Example 1	Example 2
Input: Number of emulated cycles: 7 **Input:** Instruction: 0 **Input:** Instruction: 0 **Input:** Instruction: 1 **Input:** Instruction: 2 **Output:** The stored integer is 20	**Input:** Number of emulated cycles: 8 **Input:** Instruction: 0 **Input:** Instruction: 2 **Input:** Instruction: 0 **Input:** Instruction: 0 **Input:** Instruction: 2 **Output:** The stored integer is 7

2 - Advanced - Gremlin Lifts

Problem Statement

Gremlin Lifts, a division of CBG Inc (Cheap But Glitchy Incorporated), has installed an elevator in a multi-story building. Due to their upbringing, Gremlin Lifts has numbered the floors starting at 0 on the ground level and refuse to re-number them, claiming that the request is “out of scope”. Gremlin Lifts, having been built by actual gremlins, don’t necessarily work the way one might hope.

The Gremlin Lift elevator has the same four 4 buttons in the elevator and on each floor: Up, Down, eXpress and Off. Using a proprietary technology, the elevator only responds to a button if it is already at a floor. Buttons pressed while the elevator is in motion are ignored.

The transition from floor to floor is governed by the following rules:

- Pushing any button in the Off state has no effect—there is no on button. At installation the elevator was left on at the ground floor
- The up button always causes the elevator to go up 3 floors, if it cannot go up 3 floors it does nothing.
- Pushing the down button when on odd numbered floors causes the elevator to go down one third the number of floors it is on rounded up. If it is on an even numbered floor, the elevator goes down half the number of floors it is on. For example: pressing down on the 6th floor will cause the elevator to go down 3 floors ($6/2$) to **floor 3**. If the elevator was on the 5th floor, pressing down would cause the elevator to go down 2 floors ($5/3$ rounded up) to **floor 3**.
- The express button causes the elevator to go to the ground (0) floor, except a system glitch means the button does not work when the elevator is half way (rounded up) to the top floor.
- The off button only works if on the ground (0) floor. Any button pressed while the elevator is off will not work.

Input

- A whole number, $4 \leq n,$ that represents the total number of floors the elevator will be installed on
- A string that contains any number of the following characters representing each button press: "D" for Down, "U" for Up, "X" for eXpress, and "O" for Off

Output

The program should output the final floor that the elevator is at once all button presses have been processed as well as the number of invalid button presses (button presses that caused nothing to happen) that were made.

Example

Input	Output
7 UUDUXUDO	Stopped on floor 2 Invalid button presses: 1
10 UUUUUUXUDDO	Stopped on floor 1 Invalid button presses: 4
4 OUDU	Stopped on floor 0 Invalid button presses: 3

2023\gremlin_lifts\a2-gremlin-lifts-solution.py

```
"""
Author: Josh Weese

Gremlin Lifts Advanced Model Solution
"""
import math
floor = 0
top_floor = int(input("How many floors are in the building? ")) - 1
print(f"Top floor is {top_floor}")
mid_floor = math.ceil(top_floor/2)
print(f'Middle floor is {mid_floor}')
i = 0
on = True
invalid_count = 0
buttons = input("Enter the button presses: ")
while on and i < len(buttons):
    button = buttons[i].upper()
    i += 1
    if button == 'U' and floor <= top_floor - 3:
        floor += 3
    elif button == 'D' and floor != 0:
        if floor % 2 == 0:
            floor -= floor//2
        else:
            floor -= int(math.ceil(floor/3))
    elif button == 'X' and floor != mid_floor:
        floor = 0
    elif button == 'O' and floor == 0:
        floor = 0
        on = False
        print(f'***Lift is now off***')
    else:
        invalid_count += 1
        print(f'****Invalid button press: {button}***')
    print(f"At floor {floor} after {button}")
print(f'Stopped on floor {floor}')
print(f'Invalid button presses: {invalid_count + len(buttons) - i}')
```

3 Advanced — Divisibility
Problem Statement

A positive integer k is a *proper factor* of an integer n if:

- $k \neq n$ and
- k divides n; i.e., there is an integer i such that $ik = n$.

A positive integer n is *semiperfect* if some subset of its proper factors sum to n. For example, 12 is semiperfect because $1 + 2 + 3 + 6 = 12$. On the other hand, 10 is not semiperfect because its proper factors, 1, 2, and 5, sum to 8; hence, there can be no subset that sums to 10. Likewise, 70 is not semiperfect because its proper factors, 1, 2, 5, 7, 10, 14, and 35 sum to 74, and because no subset sums to 4, no subset can sum to 70. Note that the subset can be the entire set of proper factors; hence, 6 is semiperfect because its proper factors, 1, 2, and 3, sum to 6.

Write a program that takes as input a positive integer $n < 10{,}000$ and outputs whether n is semiperfect. You may find the following fact useful. Let $S = \{u_0, u_1, \ldots, u_{k-1}\}$ be a set of positive integers, and let n be an integer. Then:

- If $n < 0$, no subset of S sums to n.
- If $k = 0$ (i.e., if $S = \varnothing$), then a subset of S sums to n if and only if $n = 0$.
- If $k > 0$ and $n \geq 0$, then a subset of S sums to n if and only if a subset of $\{u_0, u_1, \ldots, u_{k-2}\}$ sums to either n or $n - u_{k-1}$.

Example 1:

```
Enter n: 12
12 is semiperfect.
```

Example 2:

```
Enter n: 70
70 is NOT semiperfect.
```

Example 3:

```
Enter n: 6
6 is semiperfect.
```

```
1  // 3 Advanced - Divisibility
2
3  namespace Advanced.Divisibility;
4
5  class Program
6  {
7      static void Main(string[] args)
8      {
9          Console.Write("Enter n: ");
10         int n = Convert.ToInt32(Console.ReadLine());
11         List<int> factors = new();
12         for (int i = 1; i < n; i++)
13         {
14             if (n % i == 0)
15             {
16                 factors.Add(i);
17             }
18         }
19         if (HasSubset(factors, factors.Count, n))
20         {
21             Console.WriteLine(n + " is semiperfect.");
22         }
23         else
24         {
25             Console.WriteLine(n + " is NOT semiperfect.");
26         }
27     }
28
29     private static bool HasSubset(List<int> values, int k, int n)
30     {
31         if (n < 0)
32         {
33             return false;
34         }
35         if (k == 0)
36         {
37             return n == 0;
38         }
39         return HasSubset(values, k - 1, n) || HasSubset(values, k - 1, n -
             values[k - 1]);
40     }
41 }
42
```

4 - Advanced - Efficient Gardening

Problem Statement

Alice is a tech-savvy gardener who is looking to maximize the yield from her rectangular-shaped garden plot. She has information on various crops, including the spacing needed between plants and the expected yield per plant.

Write a program that helps Alice determine which combinations of crops to plant in order to achieve the best yield while covering as much of the garden as possible. For each crop, the program should calculate and output the number of rows in the garden planted for that crop along with the number of plants planted and the expected yield.

Constraints

- Each plant is planted directly in the center of a square space that is SxS where S is the specified spacing of that plant type.
- The garden can accommodate only a whole number of plants. If the spacing causes a fractional number of plants to fit along the length or width, the number of plants should be floored to the nearest whole number.
- Rows are always planted along the longest length
- Two crops cannot be planted in the same row.
- You may assume the garden is large enough to plant at least one row of one of the crops given. However, some crops may not be planted.

Input

- Two integers `L` and `W` ($L \geq 0$, $W \geq 0$) representing the length and width of the garden in meters.
- An integer `N` ($1 \leq N \leq 5$) representing the number of available crops.
- For each crop `n` of `N`
    - a string representing the name of the crop,
    - a float `S` (0.1 ≤ `S` ≤ 10) representing the spacing between plants in meters, and
    - a float `Y` (0.1 ≤ `Y` ≤ 10) representing the expected yield per plant in kilograms.

Output Format

The program should output each crop along with the number of rows and number of plants that can be planted on the field with the total expected yield in kilograms rounded to two decimal places. Crops do not need to be outputted in any particular order.

See next page for example

Example

Input

```
10 23
3
Wheat 1 1.2
Rice 2 5
Potatoes 4 42
```

Output

```
Potatoes: 5 rows, 10 plants, 420.00 kg yield
Rice: 1 rows, 5 plants, 25.00 kg yield
Wheat: 1 rows, 10 plants, 12.00 kg yield
```

2023\crop_calculator\a4-efficient-gardening-solution.py

```python
"""
Author: Josh Weese
Advanced Efficient Gardening Solution
Calculate which crops to plant in a garden to maximize yield
"""

def calculate_yield(L, W, spacing, yield_per_plant, length_wise=True):
    """Calculate the yield of a crop given the dimensions of the garden
    and the spacing between plants
    param L: length of the garden
    param W: width of the garden
    param spacing: spacing between plants
    param yield_per_plant: yield of the crop per plant
    param length_wise: whether to plant the crops length wise or width wise
    return: a dictionary containing the number of rows, the number of plants, and the total yield
    """
    # Calculate the number of plants that can fit in length and width
    plants_along_length = int(L / spacing)
    plants_along_width = int(W / spacing)
    if length_wise:
        rows = plants_along_length
    else:
        rows = plants_along_width

    total_plants = plants_along_length * plants_along_width
    total_yield = total_plants * yield_per_plant

    return {'rows': rows, 'number_of_plants': total_plants, 'yield_for_crop': total_yield}


def read_input():
    """Read the input and return the data. This function was used only in testing and
    is not part of the solution. It can be replaced using user input or a different method.

    Returns:
        tuple: The length and width of the garden, number of crops, and a list of crops.
        The list of crops is a dictionary containing the name, spacing, and yield per plant.
    """

    L, W = map(int, input('Enter field size: ').split())
    N = int(input('Enter number of fields: '))

    crops = []
    for i in range(N):
        crop_name, spacing, yield_per_plant = input(f'({i+1}) Enter crop_name, spacing, yield_per_plant: ').split()
        spacing = float(spacing)
        yield_per_plant = float(yield_per_plant)
        crops.append({'name': crop_name, 'spacing': spacing,
                      'plant_yield': yield_per_plant})
```

```python

    return L, W, N, crops

field_length, field_width, number_of_crops, crops = read_input()

# Sort the crops by yield
print('_____sorting by yield_____')
results = {}
crops.sort(key=lambda x: x['plant_yield'] / x['spacing'] ** 2, reverse=True)
print('_____planting_____')

# keeping track of the area left to plant
unplanted_area = field_length * field_width
unplanted_length = field_length
unplanted_width = field_width
# Decide whether to plant length wise or width wise
if field_length > field_width:
    length_wise = True
else:
    length_wise = False
for i in range(number_of_crops):
    # Calculate the yield of the crop
    planted_result = calculate_yield(unplanted_length,
                                     unplanted_width,
                                     crops[i]['spacing'],
                                     crops[i]['plant_yield'],
                                     length_wise)

    # If the area of the crop is less than the area left to plant, plant the crop
    planted_area = planted_result['number_of_plants'] * crops[i]['spacing'] ** 2
    if planted_area <= unplanted_area:
        results[crops[i]['name']] = {
            'rows': planted_result['rows'],
            'planted': planted_result['number_of_plants'],
            'yield': planted_result['yield_for_crop']
        }
        unplanted_area -= planted_area
        # Adjusting the dimensions for the next crop
        row_space = crops[i]['spacing'] * planted_result['rows']
        if length_wise:
            unplanted_length -= row_space
        else:
            unplanted_width -= row_space
        print(f'planted {crops[i]["name"]}')
        print(f'field size left to plant:  length: {unplanted_length}, width: {unplanted_width}')
        print(f'area left: {unplanted_area}')

print('_____results_____')
# Print the results
for crop in results:
    print(f"{crop}: {results[crop]['rows']} rows, {results[crop]['planted']} plants, {results[crop]['yield']:.2f} kg yield")

```

A5 Minimizing Change in Pocket (CIP)

I don't like accumulating extra change in my pocket (CIP). Except for quarters, I want as few coins as possible. Design a calculator that keeps tracks of the CIP and when I have a bill to pay with cash, suggest what change to give the clerk to minimize the number of coins left in my pocket. Assume the clerk returns the minimal number of coins. Also report the new numbers of CIP after change is returned by the clerk.

To avoid some issues, we will not deal with dimes or dollar bills. Also, assume that I will not give the clerk more than 99 cents in change.

If the cents on the bill can be paid exactly by the CIP, do so using as many pennies and nickels as possible.

If the cents on the bill cannot be paid exactly by the CIP, but pennies and/or nickels can be used to decrease the final number of nickels and pennies in my pocket do so.

Example 1:

Input: Starting CIP: 8 pennies, 5 nickels, 4 quarters
The bill: $12.34

Output: Give 4 pennies, 1 nickel, 1 quarter
Final CIP: 4 pennies, 4 nickels, 3 quarters

Example 2:

Input: Starting CIP: 3 pennies, 2 nickels, 1 quarters
The bill: $20.18

Output: give 3 pennies
Final CIP: 0 pennies, 4 nickels, 4 quarters

Example 3:

Input:
Starting CIP: 3 pennies, 2 nickels, 1 quarters
The bill: $25.87

Output: give 2 pennies, 2 nickel
Final: 1 pennies, 0 nickels, 2 quarters

A5 Minimizing Change in Pocket (CIP)

I don't like accumulating extra change in my pocket (CIP). Except for quarters, I want as few coins as possible. Design a calculator that keeps tracks of the CIP and when I have a bill to pay with cash, suggest what change to give the clerk to minimize the number of coins left in my pocket. Assume the clerk returns the minimal number of coins. Also report the new numbers of CIP after change is returned by the clerk.

To avoid some issues, we will not deal with dimes or dollar bills. Also, assume that I will not give the clerk more than 99 cents in change.

If the cents on the bill can be paid exactly by the CIP, do so using as many pennies and nickels as possible.

If the cents on the bill cannot be paid exactly by the CIP, but pennies and/or nickels can be used to decrease the final number of nickels and pennies in my pocket do so.

Example 1:

Input: Starting CIP: 8 pennies, 5 nickels, 4 quarters
The bill: $12.34

Output: Give 4 pennies, 1 nickel, 1 quarter
Final CIP: 4 pennies, 4 nickels, 3 quarters

Example 2:

Input: Starting CIP: 3 pennies, 2 nickels, 1 quarters
The bill: $20.18

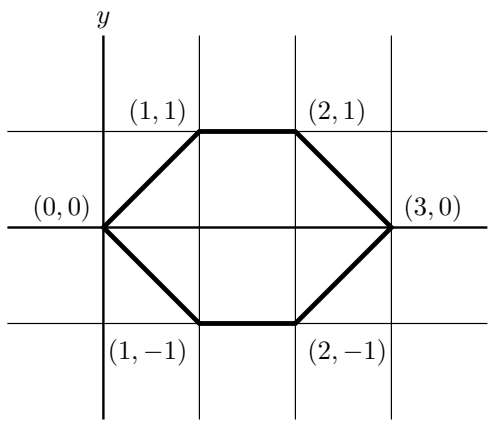
Output: give 3 pennies
Final CIP: 0 pennies, 4 nickels, 4 quarters

Example 3:

Input:
Starting CIP: 3 pennies, 2 nickels, 1 quarters
The bill: $25.87

Output: give 2 pennies, 2 nickel
Final: 1 pennies, 0 nickels, 2 quarters

6 Advanced — Hexagon
Problem Statement



In the hexagon shown to the right, the two horizontal edges have length 1, and each of the other four edges have length $\sqrt{2}$, which you should approximate as 1.414. We wish to divide the perimeter of this hexagon into n equal-length pieces, with the first piece beginning at $(0, 0)$ and going clockwise around the hexagon. Write a program that takes as input an integer $n > 1$ (the divisor) and a positive integer $k < n$ (the number of points to display). It should produce as output a list of the first k endpoints of these pieces. For example, if $n = 4$, we divide the perimeter of the hexagon into 4 equal-length pieces. Because the first segment must begin at $(0, 0)$, its endpoint will be 1/4 of the distance around the hexagon, or $(1.5, 1)$. The second endpoint will then be $(3, 0)$, etc.

Example 1:

```
Enter divisor: 4
Enter number to show: 2
(1.5, 1)
(3, -2.220446049250313E-16)
```

Example 2:

```
Enter divisor: 10
Enter number to show: 3
(0.5414427157001415, 0.5414427157001415)
(1.1172, 1)
(1.8827999999999998, 1)
```

Example 3:

```
Enter divisor: 3
Enter number to show: 2
(2.0975954738330977, 0.9024045261669021)
(2.0975954738330973, -0.9024045261669026)
```

Note: Your answers may not match the above exactly, but they should agree to three decimal places; for example, because the second y-coordinate for Example 1 rounds to 0.000, the value produced should satisfy $-0.0005 \leq y < 0.0005$.

```
// 6 Advanced - Hexagon

Console.Write("Enter divisor: ");
int n = Convert.ToInt32(Console.ReadLine());
Console.Write("Enter number to show: ");
int k = Convert.ToInt32(Console.ReadLine());
double diag = 1.414;
double len = (2 + 4 * diag) / n;
for (int i = 1; i <= k; i++)
{
    double d = i * len;
    double x, y;
    if (d <= diag)
    {
        x = d / diag;
        y = x;
    }
    else if (d <= 1 + diag)
    {
        d -= diag;
        x = 1 + d;
        y = 1;
    }
    else if (d <= 1 + 2 * diag)
    {
        d -= 1 + diag;
        x = 2 + d / diag;
        y = 1 - d / diag;
    }
    else if (d <= 1 + 3 * diag)
    {
        d -= 1 + 2 * diag;
        x = 3 - d / diag;
        y = -d / diag;
    }
    else if (d <= 2 + 3 * diag)
    {
        d -= 1 + 3 * diag;
        x = 2 - d;
        y = -1;
    }
    else
    {
        d -= 2 + 3 * diag;
        x = 1 - d / diag;
        y = d / diag - 1;
    }
    Console.WriteLine("(" + x + ", " + y + ")");
}
```