Done

Description Submission view

Available from: Friday, 29 August 2025, 9:15 AM
Due date: Friday, 29 August 2025, 10:45 AM
Requested files: p1.py, p2.py, p3.py, p4.py, p5.py (Download)
Type of work: Individual work

Problem 1 - Pounds to Grams Converter

Description:
Write a program that converts a weight from pounds (lbs) to grams (g) using the standard conversion formula.

Concept:
This problem assesses fundamental skills: reading user input, converting the input to a numeric type (**typecasting**), applying a given mathematical formula, and formatting the output to a specific number of decimal places.

Task:
Your program must produce output that exactly matches the format specified. Pay close attention to all text, spaces, and punctuation in your input prompts and your final output.
1. Prompt the user for a weight with the message: `Enter weight in pounds: `.
2. Convert the input to a floating-point number.
3. Calculate the weight in grams using the conversion factor: `1 pound = 453.592 grams`.
4. Print the result in the format `Weight in grams: [grams_value]`, with the grams value rounded to two decimal places.

Note: To *round* your final answer to two decimal places, you can use the built-in `round()` function. It takes two arguments: the number to round and the number of decimal places.
For example, `round(123.4567, 2)` will result in `123.46`.

Example:
*(Text in **bold** is what the user types.)*
```
Enter weight in pounds: 5
Weight in grams: 2267.96
```

(Another example)
```
Enter weight in pounds: 2.5
Weight in grams: 1133.98
```

Restrictions:
The input will be a positive number.
No external libraries are necessary.

Problem 2 - Product of all numbers

Description:
Write a program to take two positive integers, `a` and `b`, as input and print the product of all integers from `a` to `b-1`. Range of input: `1 <= a < b <= 1000`

Concept:
This problem evaluates your understanding of looping between two given bounds and performing cumulative multiplication.

Task:
Your program must produce output that exactly matches the format specified including all text, spaces, and punctuation in your input prompts and your final output.
1. Prompt the user for entering two numbers: `Enter two numbers: `.
2. Multiply all integers starting from the first input value up to (but not including) the second input value.
3. Print the result in the specified format: `Result`:

?

Example:

```
Enter two numbers:10 15

Result: 240240
```

Explanation :
10 x 11 x 12 x 13x 14 = 240240

Restrictions:
The input will be two positive integers where range input: 1 <= a < b <= 1000.
No external libraries are necessary.

Problem 3 - Number Pattern using Nested Loops

Description:
Write a program that takes a positive integer `n` as input and prints a right–angled triangular number pattern. Each row should contain increasing numbers starting from 1 up to the row number. There is a space after every number.

Concept:
This problem tests the ability to use **nested loops** to generate patterns.

Task:
Your program must produce output that exactly matches the format specified. Pay close attention to all text, spaces, and punctuation in your input prompts and your final output.
1. Prompt the user with: `Enter your input:` .
2. Use nested loops to generate the pattern of numbers.
3. Print the pattern row by row. In each row, there is a space after every number.

Example:

```
Enter your input:3
```

```
1
1 2
1 2 3
```

Restrictions:
The input will be a positive integer.
No external libraries are necessary.

Hint:
Use the command: *print(x, end=" ")* to add a space insted of end of line after character 'x'

Problem 4: Ascending Order of Three Numbers

Description
Write a program that reads three integers and prints them in ascending order (smallest to largest).

Note: In this context, "ascending" means **non-decreasing** order.

Concept
This problem checks your ability to use **if–else conditions and swapping values** without built-in functions.

Task

1. Prompt the user with:
   `Enter three numbers:`
2. Read three integers in one line.
3. Use if–else conditions to rearrange them in ascending order.
4. Print the numbers on one line, separated by spaces.
   `Ascending order:`

Input format
You must use this exact prompt before input:

```
Enter three numbers:
```

Output format
Your output must begin with this exact text:

```
Ascending order:
```

Examples

Example 1

```
Enter three numbers: 5 2 9

Ascending order: 2 5 9
```

Example 2

```
Enter three numbers: 3 1 2

Ascending order: 1 2 3
```

Restrictions
Do not use any built-in Python functions or libraries.

Problem 5: Alternating Sum/Product Series

Description
Generate a series where:

- First term = 1
- Second term = 2
- Odd terms (3rd, 5th, ...) are the **sum** of the previous two terms
- Even terms (4th, 6th, ...) are the **product** of the previous two terms

Task
Read an integer `n` and print the series up to `n` terms. You can assume input will always be greater than zero.

Input format
Prompt before input:
`Enter n:`

Output format
Output must be the required series.

Examples

Example 1

```
Enter n: 4

1 2 3 6
```

Example 2

```
Enter n: 7

1 2 3 6 9 54 63
```

Restrictions

- Use only loops, addition, and multiplication.
- Do not use any external libararies.
- The input n will be always greater than zero(`n>0`).

Requested files

p1.py

```
1  # write your code here below
```

p2.py

```
1  # write your code here below
```

p3.py

```
1  # write your code here below
```

p4.py

```
1  # write your code here below
```

p5.py

```
1  # write your code here below
```

VPL