

Prime Number Finder (10 pts)

Write a Python program that finds all prime numbers up to a user-specified positive integer.

The program should:

- Define a function `is_prime(n)` that:
 - Takes an integer `n`.
 - Returns `True` if `n` is a prime number, or `False` otherwise.
 - Uses a **for loop** to check divisibility.
- Ask the user for a **positive integer** using `input()`.
- Use a **try/except** block to handle improper input (e.g., non-integer or negative values) and prompt the user again if the input is invalid.
- Print **all prime numbers from 2 up to the user-specified number**, inclusive.

Hints:

- A prime number is greater than 1 and divisible only by 1 and itself.
- To check if `n` is prime, you only need to test divisibility for integers from 2 up to `n-1` (or optionally up to \sqrt{n}).
- Use the modulus operator `%` to test divisibility.

Example Run:

```
Enter a positive integer: 20
Prime numbers up to 20: 2, 3, 5, 7, 11, 13, 17, 19
```

```
Enter a positive integer: abc
Invalid input. Please enter a positive integer.
Enter a positive integer: -5
Invalid input. Please enter a positive integer.
Enter a positive integer: 10
Prime numbers up to 10: 2, 3, 5, 7
```

CD Profit Calculator (20 pts)

Write a Python program that calculates the **profit** from a Certificate of Deposit (CD) when the money is withdrawn after a given number of months.

The bank rules are:

- **Annual interest rate:** 4%, using **simple monthly interest** (no compounding).
- **Automatic renewal:** The CD renews for another 12-month term at maturity unless withdrawn.
- **Early withdrawal penalty:** If withdrawn **before the end of a 12-month term**, the bank deducts **3 months of interest on the current balance**.

The program should:

- Define a **function** `compute_profit(initial_deposit, months)` that returns the total profit (final balance – initial deposit).
- Use `input()` to ask the user for the starting deposit and the number of months until withdrawal.
- Print the profit in dollars.

Remember:

- Monthly simple interest rate = $0.04 / 12$.
- Total interest = `initial_deposit * monthly_rate * months`.
- If withdrawal happens before completing a 12-month cycle, subtract **3 months of simple interest** from the current balance as the penalty.

Example Runs:

```
Enter initial deposit: 10000
Enter number of months until withdrawal: 24
Profit: $816.00
```

```
Enter initial deposit: 10000
Enter number of months until withdrawal: 9
Profit: $200.00    # after early withdrawal penalty
```

2-D Bag Management System (30 pts)

Write a Python program that manages items in a bag represented as a 2-dimensional grid.

The bag is a grid of cells (rows × columns). Each **item** occupies a rectangular area of cells, and **each cell can only hold one item at a time**.

The program should:

- Define a function `add_item(bag, name, row, col, height, width)`

- name must be **a single character** (e.g., 'A' , 'b' , '#').
- Place the item into the bag starting at (row, col) and filling the given height × width area.
- Return True if the item fits without overlap and is added successfully.
- If the item cannot fit (overlap or out of bounds), **print an error message** and return False .
- Define a function move_item(bag, name, new_row, new_col)
 - Move an existing item (identified by its single-character name) to a new top-left position if space is available.
 - If the move is illegal (overlap or out of bounds), **print an error message** and **leave the item in its original position**.
- Define a function print_bag(bag)
 - Print a “picture” of the bag grid, showing item characters in their occupied cells and . for empty cells.

The program should then:

1. Create an empty bag based on user input for number of rows and columns.
2. Use input() in a loop to let the user:
 - Add a new item (provide **single-character name**, position, size).
 - Move an existing item to a new position.
 - Print the current bag layout.
 - Quit the program.

Rules & Hints:

- An item cannot overlap another item or go outside the bag’s boundaries.
- Store the bag as a list of lists (2-D array):

```
bag = [['.' for _ in range(column)] for _ in range(row)]
```

- When moving an item, clear its old cells only after confirming the move is legal.
- name is exactly one character long and is not . .
- Print a clear error message if add_item or move_item fails.
- **You can also define a function to check whether a placement is legal.**

Example Run:

```
Enter bag rows: 5
Enter bag columns: 8
```

Command (add/move/print/quit): add
Item name (single character): A
Top-left row: 1
Top-left column: 2
Item height: 2
Item width: 3
Item A added.

Command (add/move/print/quit): print

```
.....  
..AAA..  
..AAA..  
.....  
.....
```

Command (add/move/print/quit): add
Item name (single character): B
Top-left row: 0
Top-left column: 1
Item height: 2
Item width: 3
Error: Cannot add item B. Space is occupied or out of bounds.

Command (add/move/print/quit): add
Item name (single character): B
Top-left row: 0
Top-left column: 0
Item height: 1
Item width: 2
Item B added.

Command (add/move/print/quit): print

```
BB.....  
..AAA..  
..AAA..  
.....  
.....
```

Command (add/move/print/quit): move
Item name (single character): A
New top-left row: 3
New top-left column: 4
Item A moved.

Command (add/move/print/quit): print
BB.....
.....
.....
....AAA.
....AAA.