

LAB ASSIGNMENT – 2.5

PROMPT:

- ❖ **TASK 1 :** Refactoring Odd/Even Logic (List Version):
- ❖ **Scenario:** You are improving legacy code.
- ❖ **Task:** Write a program to calculate the sum of odd and even numbers in a list, then refactor it using AI.
- ❖ **Expected Output:** Original and improved code

PROMPT:

The screenshot shows a dark-themed user interface with a central text area. At the top, there is some very faint, illegible text. Below it, the main text area contains the following content:

MING ODD AND EVEN NUMBERS

Task 1: Refactoring Odd/Even Logic (List Version)

□ Scenario:
You are improving legacy code.

□ Task:
Write a program to calculate the sum of odd and even numbers in a list, then refactor it using AI.

□ Expected Output:
□ Original and improved code

In the bottom right corner of the text area, there is a small icon followed by the text "Task1.ipynb:1-".

REFACTORING CODE FOR SUMMING ODD AND EVEN NUMBERS

```
# Original version (legacy)
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
even_sum = 0
odd_sum = 0
for n in numbers:
    if n % 2 == 0:
        even_sum += n
    else:
        odd_sum += n

print(f"Even sum: {even_sum}")
print(f"Odd sum: {odd_sum}")
```

Refactored (improved) version — replace the notebook cell with this code:

```
# ...existing code...
def sum_even_odd(nums):
    """Return (even_sum, odd_sum) for an iterable of ints."""
    even = sum(x for x in nums if isinstance(x, int) and x % 2 == 0)
    odd = sum(x for x in nums if isinstance(x, int) and x % 2 != 0)
    return even, odd

if __name__ == "__main__":
    numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
    even_sum, odd_sum = sum_even_odd(numbers)
    print("Even sum:", even_sum)
    print("Odd sum:", odd_sum)
# ...existing code...
```



+ Task1_2.ipynb • Cell 1

Original version:

```
# Original version (legacy)
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
even_sum = 0
odd_sum = 0
for n in numbers:
    if n % 2 == 0:
        even_sum += n
    else:
        odd_sum += n

print(f"Even sum: {even_sum}")
print(f"Odd sum: {odd_sum}")
8] ✓ 0.0s
...
Even sum: 30
Odd sum: 25
```

Reference Code:

```
# Reference Code (improved version)
def sum_even_odd(nums):
    """Return (even_sum, odd_sum) for an iterable of ints."""
    even = sum(x for x in nums if isinstance(x, int) and x % 2 == 0)
    odd = sum(x for x in nums if isinstance(x, int) and x % 2 != 0)
    return even, odd

if __name__ == "__main__":
    numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
    even_sum, odd_sum = sum_even_odd(numbers)
    print("Even sum:", even_sum)
    print("Odd sum:", odd_sum)
[?] ✓ 0.0s
...
Even sum: 30
Odd sum: 25
```

Expected Output:

```
Even sum: 30
Odd sum: 25
```

Explanation:

give me the explanation of Refactored (improved) version

- ✓ Optimizing tool selection...
- ✓ Explained improvements of refactored code

Here's the explanation of the refactored version:

Key Improvements:

1. **Function Encapsulation** - `sum_even_odd(nums)` wraps the logic into a reusable function rather than loose statements. This makes the code modular and testable.
2. **List Comprehensions with `sum()`** - Replaces the manual loop with `sum(x for x in nums if ...)`. This is more Pythonic, concise, and often faster.
3. **Readability** - The logic is expressed declaratively: "sum all numbers where the condition is true" rather than manually accumulating in variables.
4. **Type Safety** - `isinstance(x, int)` ensures we only process integers, preventing errors with unexpected data types.
5. **Separation of Concerns** - The function handles the calculation logic, while the `if __name__ == "__main__":` block handles input/output. This follows best practices for scripts.
6. **Documentation** - The docstring explains what the function does and its return value.
7. **Return Tuple** - Returns both values as a tuple `(even_sum, odd_sum)` instead of modifying external variables, eliminating side effects.

PROMPT:

Task 2: Area Calculation Explanation

❖ **Scenario:**

You are onboarding a junior developer.

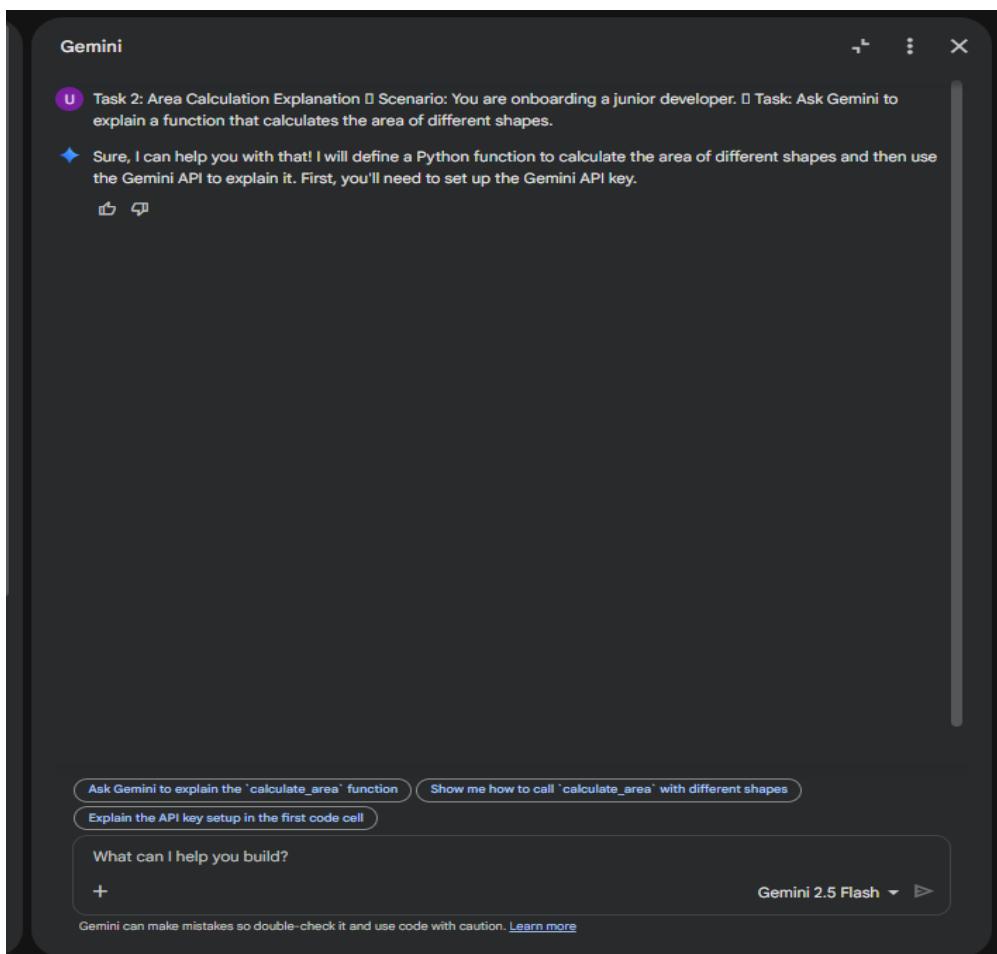
❖ **Task:**

Ask Gemini to explain a function that calculates the area of different shapes.

❖ **Expected Output:**

- Code
- Explanation

PROMPT:



Expected Output:

Code:

```
(s)  def calculate_area(shape, **kwargs):
    """
    Calculates the area of different shapes.

    Args:
        shape (str): The type of shape ('square', 'rectangle', 'circle', 'triangle').
        **kwargs: Keyword arguments for shape dimensions.
            - 'side' for square
            - 'length', 'width' for rectangle
            - 'radius' for circle
            - 'base', 'height' for triangle

    Returns:
        float: The calculated area of the shape.
        str: An error message if the shape is not recognized or dimensions are missing.
    """
    import math

    if shape == 'square':
        if 'side' in kwargs:
            return kwargs['side'] ** 2
        else:
            return "Error: Missing 'side' for square."
    elif shape == 'rectangle':
        if 'length' in kwargs and 'width' in kwargs:
            return kwargs['length'] * kwargs['width']
        else:
            return "Error: Missing 'length' or 'width' for rectangle."
    elif shape == 'circle':
        if 'radius' in kwargs:
            return math.pi * (kwargs['radius'] ** 2)
        else:
            return "Error: Missing 'radius' for circle."
    elif shape == 'triangle':
        if 'base' in kwargs and 'height' in kwargs:
            return 0.5 * kwargs['base'] * kwargs['height']
        else:
            return "Error: Missing 'base' or 'height' for triangle."
    else:
        return "Error: Unknown shape."

print(calculate_area('square', side=5))
print(calculate_area('rectangle', length=4, width=6))
print(calculate_area('circle', radius=3))
print(calculate_area('triangle', base=10, height=4))
print(calculate_area('octagon', side=5)) # Example of unknown shape
```

Expected Output:

```
*** 25
24
28.274333882308138
20.0
Error: Unknown shape.
```

Explanation:

Gemini

U Task 2: Area Calculation Explanation D Scenario: You are onboarding a junior developer. D Task: Ask Gemini to explain a function that calculates the area of different shapes.

◆ Sure, I can help you with that! I will define a Python function to calculate the area of different shapes and then use the Gemini API to explain it. First, you'll need to set up the Gemini API key.

U """ Calculates the area of different shapes. Args: shape (str): The type of shape ('square', 'rectangle', 'circle', 'triangle'). **kwargs: Keyword arguments for shape dimensions. - 'side' for square - 'length', 'width' for rectangle - 'radius' for circle - 'base', 'height' for triangle Returns: float: The calculated area of the shape. str: An error message if the shape is not recognized or dimensions are missing. """ manual explanation in ## points

◆ Certainly! Here's a manual explanation of the `calculate_area` function, broken down into key points:

U " manual explanation in points

◆ Certainly! Here's a manual explanation of the `calculate_area` function, broken down into key points:

- Purpose: The `calculate_area` function is designed to compute the area of various common geometric shapes, including squares, rectangles, circles, and triangles.
- Parameters:
 - `shape` (string): This required parameter specifies the type of shape whose area needs to be calculated. It accepts values like `'square'`, `'rectangle'`, `'circle'`, or `'triangle'`.
 - `**kwargs` (keyword arguments): This allows the function to accept a flexible number of named arguments, which are the dimensions specific to each shape. For example:
 - For a square, it expects `'side'`.
 - For a rectangle, it expects `'length'` and `'width'`.
 - For a circle, it expects `'radius'`.
 - For a triangle, it expects `'base'` and `'height'`.
- How it handles different shapes:
 - The function uses a series of `if/elif` statements to check the value of the `shape` parameter.
 - Based on the `shape`, it then checks if the necessary `kwargs` (dimensions) are provided.
 - If the shape and its corresponding dimensions are valid, it applies the standard mathematical formula to calculate the area.
 - It imports the `math` module specifically for `math.pi` when calculating the area of a circle.
- Error Handling:
 - If a required dimension for a specific shape is missing (e.g., `side` for a square), it returns an error string indicating which dimension is absent.
 - If an unrecognized `shape` is provided (e.g., `'octagon'`), it returns a generic "Error: Unknown shape." message.
- Return Value:
 - If the calculation is successful, it returns a `float` representing the area of the shape.
 - If there's an error (missing dimensions or unknown shape), it returns a `str` containing an error message.

Modify the function to raise exceptions for errors Add more test cases to the function Create a list of shapes and calculate their areas

What can I help you build?

+ Gemini 2.5 Flash ➤

Gemini can make mistakes so double-check it and use code with caution. [Learn more](#)

PROMPT:

Task 3: Prompt Sensitivity Experiment

❖ Scenario:

You are testing how AI responds to different prompts.

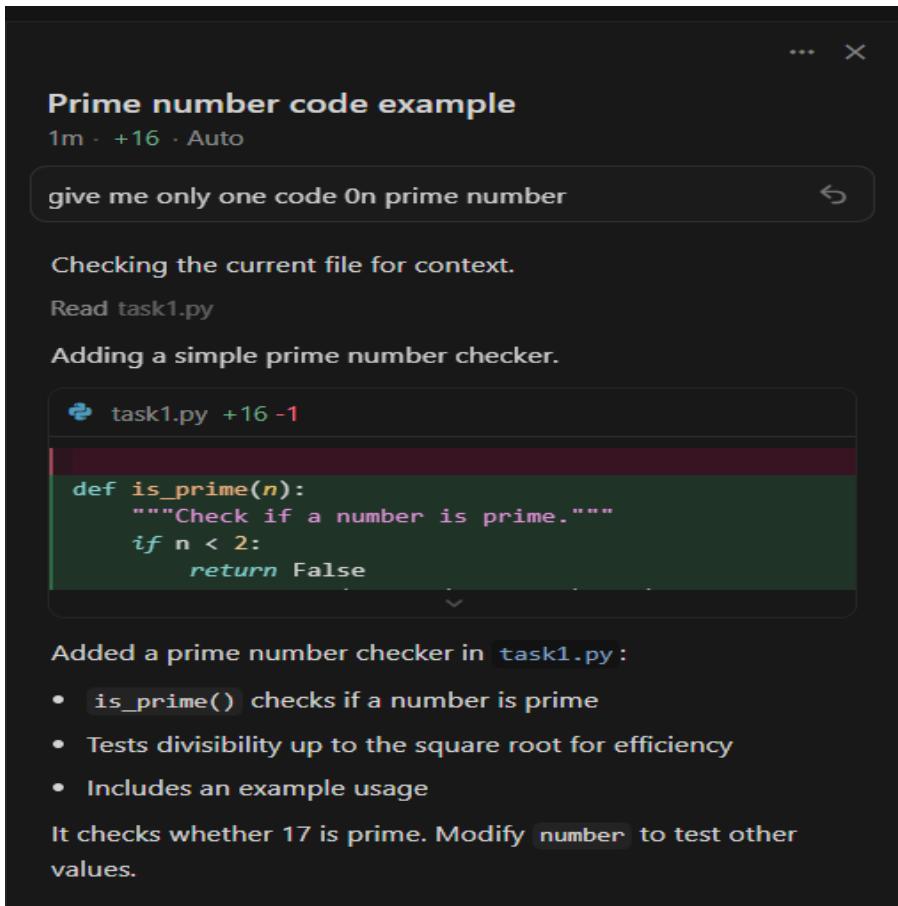
❖ Task:

Use Cursor AI with different prompts for the same problem and observe code changes.

❖ Expected Output:

- Prompt list
- Code variations

Cursor AI Prompt:



The screenshot shows the Cursor AI interface with a dark theme. At the top, it says "Prime number code example" with a timestamp "1m · +16 · Auto". Below this is a search bar containing the text "give me only one code On prime number". A message "Checking the current file for context." follows, followed by "Read task1.py". Then, "Adding a simple prime number checker." is shown. A code editor window displays "task1.py +16 -1" with the following Python code:

```
def is_prime(n):
    """Check if a number is prime."""
    if n < 2:
        return False
```

Below the code editor, a message says "Added a prime number checker in task1.py:" followed by a bulleted list:

- `is_prime()` checks if a number is prime
- Tests divisibility up to the square root for efficiency
- Includes an example usage

At the bottom, a note says "It checks whether 17 is prime. Modify `number` to test other values."

GitHub Copilot:

CHAT

REQUEST FOR PRIME NUMBER CODE SNIPPET

give me only one code On prime number

Task1_2.ipynb • Cell 4:37

- ✓ Optimizing tool selection...
- ✓ Retrieved Notebook summary

Now I'll add a new cell with a prime number program:

✓ Edit Task1_2.ipynb

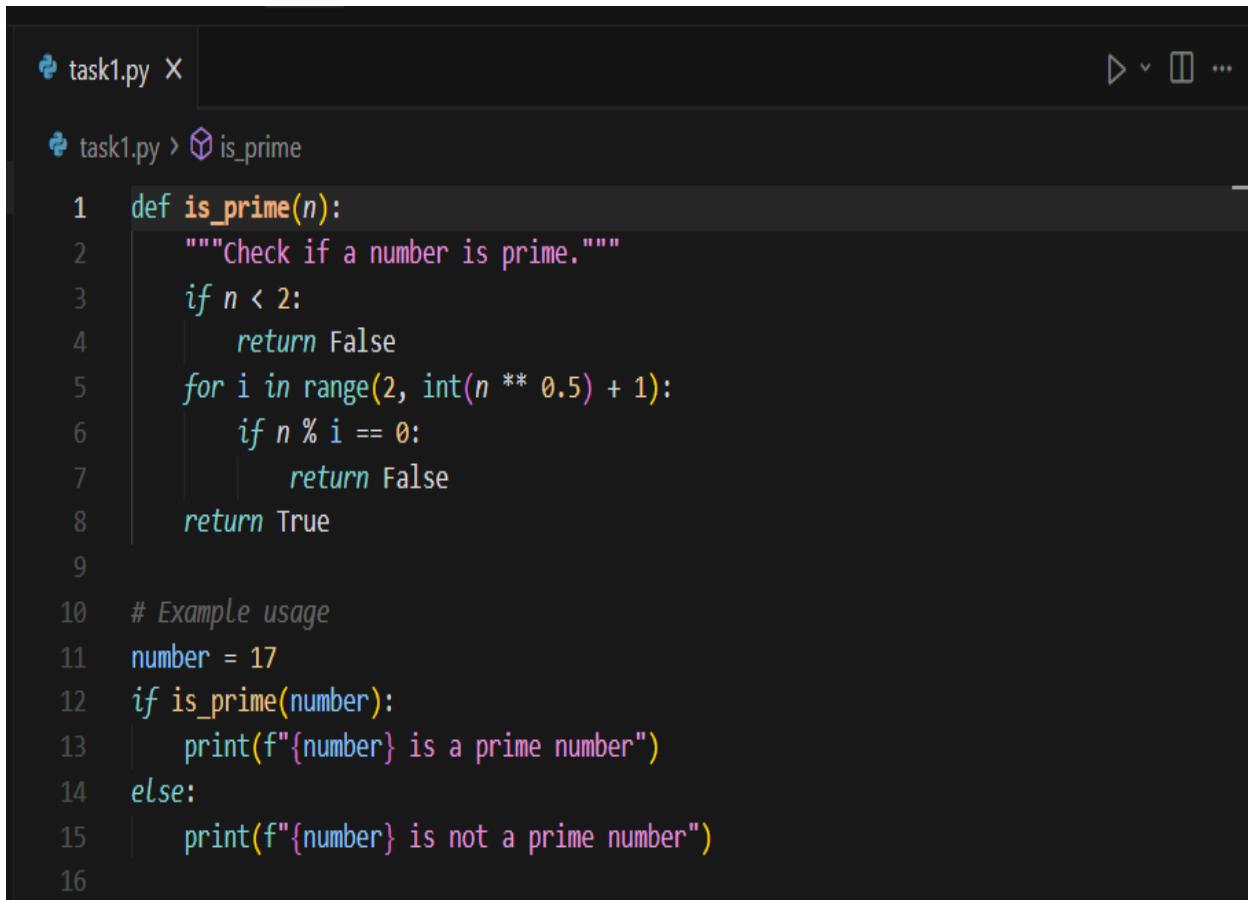
Done! I've added a prime number checker program to your notebook. It includes:

- A function `is_prime(n)` that efficiently checks if a number is prime
- Test cases showing which numbers are prime and which aren't

The code handles edge cases and uses an optimized approach by checking divisibility only up to the square root of the number.

C ⌂ ⌂ ⌂ ...

Cursor AI Code Variations:

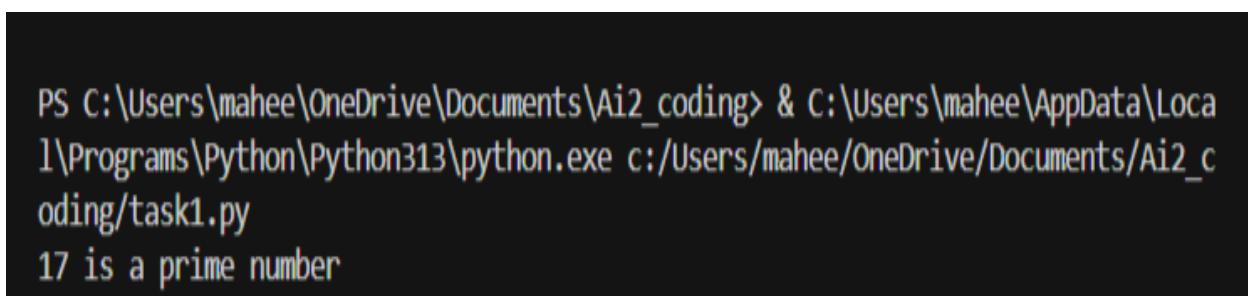


The screenshot shows a code editor window with a dark theme. The file is named 'task1.py'. The code defines a function 'is_prime' that checks if a number is prime. It includes a docstring, handles the case where n < 2, and uses a for loop to check divisibility up to the square root of n. An example usage is shown at the bottom.

```
task1.py X
task1.py > is_prime

1 def is_prime(n):
2     """Check if a number is prime."""
3     if n < 2:
4         return False
5     for i in range(2, int(n ** 0.5) + 1):
6         if n % i == 0:
7             return False
8     return True
9
10 # Example usage
11 number = 17
12 if is_prime(number):
13     print(f"{number} is a prime number")
14 else:
15     print(f"{number} is not a prime number")
16
```

Output:



The screenshot shows a terminal window with a black background and white text. The command 'python task1.py' is run, and the output shows that 17 is a prime number.

```
PS C:\Users\mahee\OneDrive\Documents\Ai2_coding> & c:\Users\mahee\AppData\Local\Programs\Python\Python313\python.exe c:/Users/mahee/OneDrive/Documents/Ai2_coding/task1.py
17 is a prime number
```

GitHub Copilot Code Variations :

```
# Prime Number Checker
def is_prime(n):
    """Check if a number is prime."""
    if n < 2:
        return False
    if n == 2:
        return True
    if n % 2 == 0:
        return False
    for i in range(3, int(n**0.5) + 1, 2):
        if n % i == 0:
            return False
    return True

# Test
numbers = [2, 3, 4, 5, 10, 17, 20, 23]
for num in numbers:
    print(f"{num} is prime: {is_prime(num)}")
✓ 0.0s
```

Output:

```
2 is prime: True
3 is prime: True
4 is prime: False
5 is prime: True
10 is prime: False
17 is prime: True
20 is prime: False
23 is prime: True
```

Explanation Of Cursor AI Code:

explaination of given above prime number code

Reviewing the code to explain how it works:

Read task1.py

Explanation of the Prime Number Code

What is a Prime Number?

A prime number is a number greater than 1 that is only divisible by 1 and itself.

Code Breakdown:

Line 1: Function Definition

```
def is_prime(n):
```

- Defines a function `is_prime` that takes a number `n` as input.
- Returns `True` if `n` is prime, `False` otherwise.

Line 3-4: Base Case Check

```
if n < 2:  
    return False
```

- Numbers less than 2 are not prime.
- Returns `False` immediately.

Line 5: Loop Range

› 1 File

Plan, @ for context, / for commands

∞ Agent ▾ Auto ▾

Line 5: Loop Range

```
for i in range(2, int(n ** 0.5) + 1):
```

- Checks divisors from 2 up to the square root of `n` (rounded up).
- Why square root? If `n` has a factor greater than \sqrt{n} , it must have a corresponding factor less than \sqrt{n} . So we only need to check up to \sqrt{n} .
- `n ** 0.5` computes \sqrt{n} ; `int()` rounds down; `+ 1` includes the boundary.

Line 6-7: Divisibility Check

```
if n % i == 0:  
    return False
```

- `n % i == 0` means `n` is divisible by `i`.
- If any divisor is found, `n` is not prime; return `False`.

Line 8: Prime Confirmed

```
return True
```

- If no divisors are found in the loop, `n` is prime; return `True`.

Explanation Of GitHub Copilot Code:

explaination of given above prime number code

Task1_2.ipynb • Cell 4:37 Task1_2.ipynb • Cell 4:37

✓ Explained prime number code logic and functionality.

Here's an explanation of the prime number checker code:

Function: `is_prime(n)`

Purpose: Determines if a number is prime (only divisible by 1 and itself).

Step-by-step logic:

1. `if n < 2: return False`
 - Numbers less than 2 (including negatives and 0, 1) are not prime
2. `if n == 2: return True`
 - 2 is the only even prime number, so it returns True immediately
3. `if n % 2 == 0: return False`
 - If the number is divisible by 2 (even), it's not prime
4. `for i in range(3, int(n**0.5) + 1, 2):`
 - Checks divisibility by odd numbers from 3 up to the square root of `n`
 - Only checks odd numbers (step of 2) since even divisors are already ruled out
 - Optimization: Only need to check up to \sqrt{n} because if `n` has a divisor greater than \sqrt{n} , it must also have a corresponding divisor less than \sqrt{n}
5. `if n % i == 0: return False`
 - If `n` is divisible by any of these numbers, it's not prime
6. `return True`
 - If no divisors found, `n` is prime

Task1_2.ipynb • Cell 5

Describe what to build next

Task 4: Tool Comparison Reflection

❖ **Scenario:**

You must recommend an AI coding tool.

❖ **Task:**

Based on your work in this topic, compare Gemini, Copilot, and Cursor AI for usability and code quality.

❖ **Expected Output:**

Short written reflection

Task 4 : Tool Comparison Reflection Recommendations :

- **If you are a student or researcher:** Use Gemini. The free tier is generous, and its ability to explain "why" is unmatched for learning.
- **If you work in a large enterprise:** Stick with Copilot. The security, GitHub integration, and "set it and forget it" nature make it the safest bet.
- **If you are building a product from scratch:** Use Cursor. The ability to refactor across multiple files at once will save you hundreds of hours of manual "plumbing."