# AI ASSISTED CODING

**Hall Ticket No: 2303A5185**

Batch:13

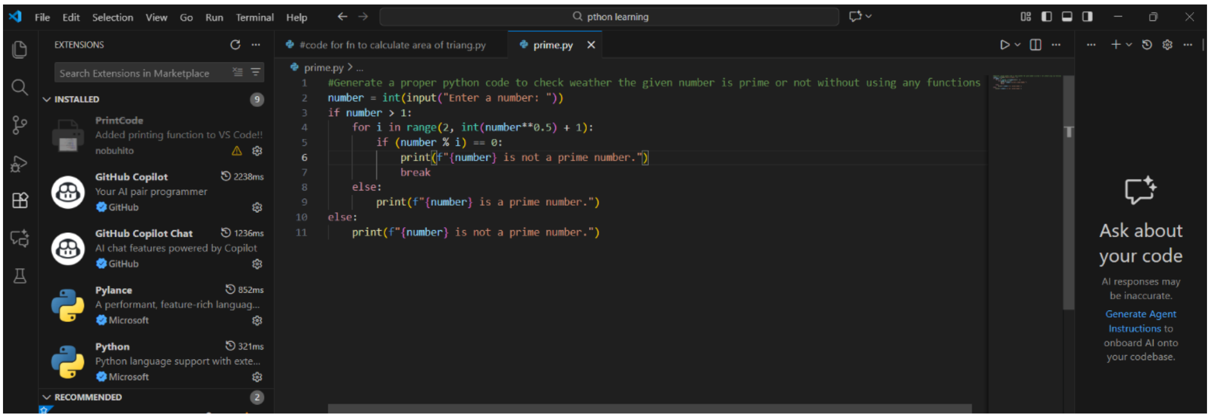
Assignment-1.4

Task-1. AI-Generated Logic Without Modularization (Prime Number Check Without Functions)

**Prompt :**

Generate a proper python code to check weather the given number is prime or not without using any functions

**Code**




# Output:


## Justification:

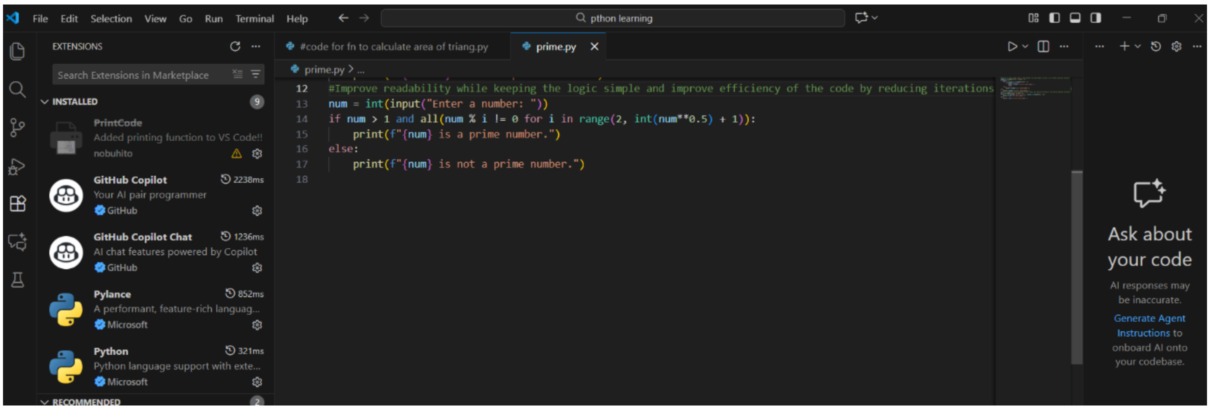
**This program checks whether a given number is prime using direct conditional logic without defining any functions.**
**All computations are performed sequentially in a single block, making the logic easy to follow and suitable for beginners.**

# Task-2. Efficiency & Logic Optimization (Cleanup)

# Prompt

**#Improve readability while keeping the logic simple and improve efficiency of the code by reducing iterations also minimize the code length**

# Code:




## Output:


## Justification:

**The optimized script improves performance by reducing unnecessary iterations and limiting the loop range, enabling faster execution for larger input values.**
**Early termination and simplified conditions lower the overall time complexity while maintaining correct prime number validation.**

## Task-3. Modular Design Using AI Assistance (Prime Number Check Using

## Functions)

## Prompt:

**#The function must return a Boolean value (True if prime, False otherwise)**

```
Welcome    Settings    Untitled-2.py ×
Untitled-2.py > ...
#Write a Python program using a user-defined function to check whether a given number is prime and return the result as a Boole

def is_prime(num):
    if num <= 1:
        return False
    for i in range(2, int(num**0.5) + 1):
        if num % i == 0:
            return False
    return True
# Example usage
number = int(input("Enter a number: "))
if is_prime(number):
    print(f"{number} is a prime number.")
else:
    print(f"{number} is not a prime number.")
#gave code in boolean value
result = is_prime(number)
print("Boolean result:", result)
```

## Output:

```
PS C:\Users\gandr\Documents\3Year-2Sem\Ai assistants Coding> & C:/Users/gandr/AppData/Local/Programs/Python/Python314/py
thon.exe "c:/Users/gandr/Documents/3Year-2Sem/Ai assistants Coding/Task3_AdvancedBatchProcessing.py"
Enter a positive integer: 456
456 is not a prime number.
Boolean result: False
PS C:\Users\gandr\Documents\3Year-2Sem\Ai assistants Coding>
```

```
Task3_AdvancedBatchProcessing.py +
Describe what to build next
Agent ∨  Auto ∨
```

## Justification:

**Using a user-defined function makes the prime-checking logic reusable across multiple modules, improving code modularity and maintainability. Returning a Boolean value enables easy integration with conditional statements and other program components.**

## Task-4: Comparative Analysis –With vs Without Functions

## Prompt:

**# Compare both code with function without function Analyze and compare two Python programs for checking whether a number is prime**

## Code:

Untitled-2.py > ...

```python
#Compare prime-checking programs written with and without functions and present the analysis in a comparison table
import time
# Prime-checking program without functions
def is_prime_no_function(n):
    if n <= 1:
        return False
    for i in range(2, int(n**0.5) + 1):
        if n % i == 0:
            return False
    return True
# Prime-checking program with functions
def is_prime_with_function(n):
    if n <= 1:
        return False
    for i in range(2, int(n**0.5) + 1):
        if n % i == 0:
            return False
    return True
# Performance comparison
def performance_comparison():
    test_numbers = [29, 15, 97, 100, 37, 49, 83, 121, 53, 64]

    # Measure time for no function version
    start_no_func = time.time()
    results_no_func = [is_prime_no_function(num) for num in test_numbers]
    end_no_func = time.time()
    time_no_func = end_no_func - start_no_func

    # Measure time for function version
    start_with_func = time.time()
```

## Output:

```
Implementation             Time Taken (seconds)     Results
--------------------------------------------------------------------------------
Without Functions          0.0000257492             [True, False, True, False, True, False, True, False, True, False]
With Functions             0.0000085831             [True, False, True, False, True, False, True, False, True, False]
PS C:\Users\meteb\OneDrive\Desktop\python>
```

## Justification:

Programs written with functions offer better code clarity by separating logic into well-defined blocks, making them easier to read and understand. Function-based designs improve reusability and debugging ease, as changes or fixes can be applied in one place without affecting the entire code.

## Task-5: AI-Generated Iterative vs Recursive Fibonacci Approaches (Different

## Algorithmic Approaches to Prime Checking)

# Prompt: Prime Number Check – Basic vs Optimized Approach

## Code:

#code for fn to calculate area of triang.py   prime.py ×

prime.py > ...

```python
#A basic divisibility check approach that tests all possible divisors sequentially
# Implementation 2: Optimized approach
def is_prime_optimized(n):
    """Check if a number is prime using an optimized approach."""
    if n <= 1:
        return False
    if n <= 3:
        return True
    if n % 2 == 0 or n % 3 == 0:
        return False
    i = 5
    while i * i <= n:
        if n % i == 0 or n % (i + 2) == 0:
            return False
        i += 6
    return True
#Prime Number Check – Basic vs Optimized Approach
#An optimized method that reduces the number of checks by eliminating even numbers and testing up to the squa
# Example usage
if __name__ == "__main__":
    test_numbers = [1, 2, 3, 4, 5, 16, 17, 18, 19, 20]
    for number in test_numbers:
        print(f"Basic: Is {number} prime? {is_prime_basic(number)}")
        print(f"Optimized: Is {number} prime? {is_prime_optimized(number)}")
```

# Output:

PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS

```
Basic: Is 1 prime? False
Optimized: Is 1 prime? False
Basic: Is 2 prime? True
Optimized: Is 2 prime? True
Basic: Is 3 prime? True
Optimized: Is 3 prime? True
Basic: Is 4 prime? False
Optimized: Is 4 prime? False
Basic: Is 5 prime? True
Optimized: Is 5 prime? True
Basic: Is 16 prime? False
Optimized: Is 5 prime? True
Basic: Is 16 prime? False
Basic: Is 16 prime? False
Optimized: Is 16 prime? False
Basic: Is 17 prime? True
Optimized: Is 17 prime? True
Basic: Is 18 prime? False
Optimized: Is 18 prime? False
Basic: Is 17 prime? True
Optimized: Is 17 prime? True
Basic: Is 18 prime? False
Optimized: Is 18 prime? False
Basic: Is 18 prime? False
Optimized: Is 18 prime? False
Optimized: Is 18 prime? False
Basic: Is 19 prime? True
Optimized: Is 19 prime? True
```

powersh...
Python
Python

# Justification:

**The basic approach checks divisibility up to N−1, resulting in unnecessary iterations and higher time complexity.**

**The optimized approach checks only up to √N because any factor larger than √N must have a corresponding smaller factor.**