AI Assisted Coding

Name : Vineeth-Chidurala **Date : 30-01-2026**

Ht.No. : 2303A52447

Task - 1 : Use AI to generate two solutions for checking prime numbers:

- Naive approach(basic)
- Optimized approach

Prompt : Generate Python code for two prime-checking methods and explain how the optimized version improves performance

Screenshots:

```
import math
```

Keep Undo

Generate code

Auto

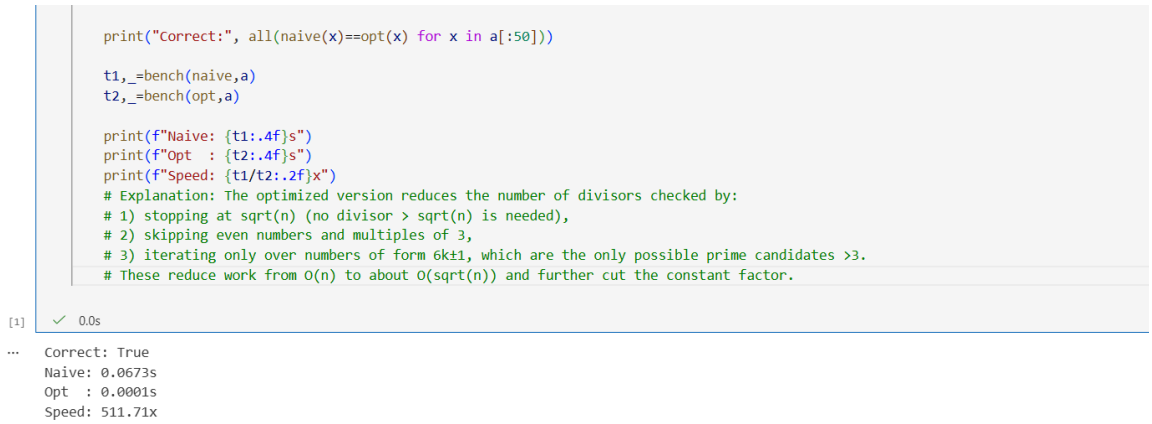
```
import random
import time

# Prime-checking methods and a small benchmark
# - Naive: tries every divisor from 2 to n-1 (O(n))
# - Optimized: handles small cases, checks 2 and 3, then tests divisors of form 6k±1 up to sqrt(n) (≈ O(sqrt(n)))

def is_prime_naive(n: int) -> bool:
    """Naive prime check: try every divisor from 2..n-1. O(n) time."""
    if n < 2:
        return False
    for d in range(2, n):
        if n % d == 0:
            return False
    return True

def is_prime_optimized(n: int) -> bool:
    """
    Optimized prime check:
    - handle n < 2, and small primes 2 and 3
    - eliminate multiples of 2 and 3 quickly
    - then test divisors 6k-1 and 6k+1 up to sqrt(n)
    This reduces number of candidate divisors and runs in roughly O(sqrt(n)) time.
    """
    if n < 2:
        return False
    if n in (2, 3):
        return True
```

Output:



```
    print("Correct:", all(naive(x)==opt(x) for x in a[:50]))

    t1,_=bench(naive,a)
    t2,_=bench(opt,a)

    print(f"Naive: {t1:.4f}s")
    print(f"Opt  : {t2:.4f}s")
    print(f"Speed: {t1/t2:.2f}x")
    # Explanation: The optimized version reduces the number of divisors checked by:
    # 1) stopping at sqrt(n) (no divisor > sqrt(n) is needed),
    # 2) skipping even numbers and multiples of 3,
    # 3) iterating only over numbers of form 6k±1, which are the only possible prime candidates >3.
    # These reduce work from O(n) to about O(sqrt(n)) and further cut the constant factor.
```

[1] ✓ 0.0s

```
Correct: True
Naive: 0.0673s
Opt  : 0.0001s
Speed: 511.71x
```

Justification : The naive prime method checks all numbers up to *n*, making it slow for large inputs.
The optimized version checks only up to √n, reducing time complexity and improving efficiency.

Task – 2 : (Transparency in Recursive Algorithms)

Objective: Use AI to generate a recursive function to calculate Fibonacci numbers.

Prompt : Generate a recursive function to calculate fibonacci numbers with clear comments explaining recursion and base cases and recursive calls

ScreenShots :

```python
#generate a recursive function to calculate fibonacci numbers with clear comments explaining recursion and base cases and recursiive calls
def fibonacci(n):
    # BASE CASE 1:
    # If n is 0, return 0 (first Fibonacci number)
    if n == 0:
        return 0

    # BASE CASE 2:
    # If n is 1, return 1 (second Fibonacci number)
    if n == 1:
        return 1

    # RECURSIVE CALL:
    # fibonacci(n) = fibonacci(n-1) + fibonacci(n-2)
    # The function calls itself to calculate previous two values
    return fibonacci(n-1) + fibonacci(n-2)


# Example usage
num = 6
print("Fibonacci of", num, "is", fibonacci(num))
```

Output :

```
        return 1

    # RECURSIVE CALL:
    # fibonacci(n) = fibonacci(n-1) + fibonacci(n-2)
    # The function calls itself to calculate previous two values
    return fibonacci(n-1) + fibonacci(n-2)


# Example usage
num = 6
print("Fibonacci of", num, "is", fibonacci(num))
```

[2] ✓ 0.0s

```
Fibonacci of 6 is 8
```

Justification : The recursive Fibonacci function works by calling itself until it reaches base cases (0 or 1).
This makes recursion transparent because each call directly matches the mathematical definition.

Task-3 : Use AI to generate a Python program that reads a file and processes data.

Prompt : Generate code with proper error handling and clear explanations for each exception.

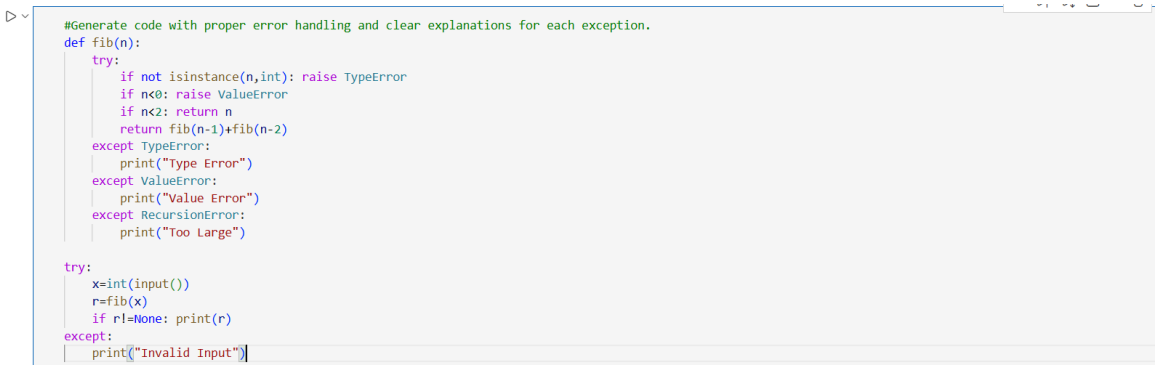
Expected Output:

• Code with meaningful exception handling.

• Clear comments explaining each error scenario.

• Validation that explanations align with runtime behavior

ScreenShots :

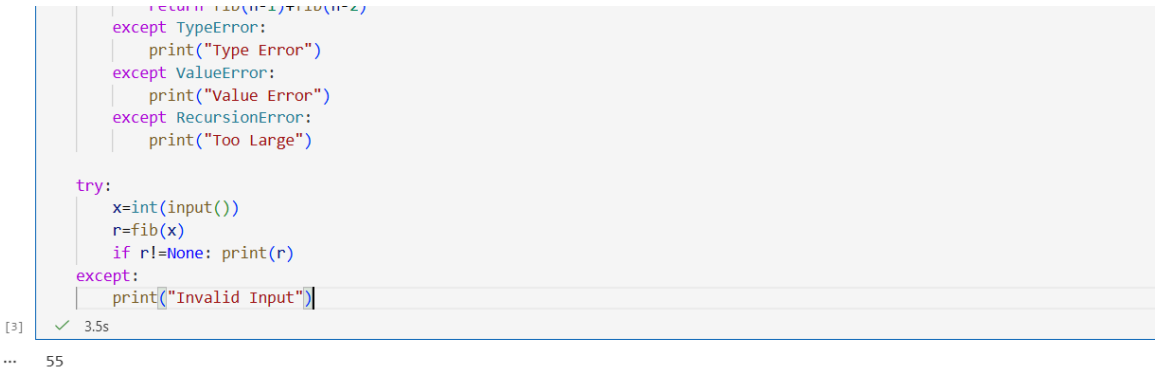


```
#Generate code with proper error handling and clear explanations for each exception.
def fib(n):
    try:
        if not isinstance(n,int): raise TypeError
        if n<0: raise ValueError
        if n<2: return n
        return fib(n-1)+fib(n-2)
    except TypeError:
        print("Type Error")
    except ValueError:
        print("Value Error")
    except RecursionError:
        print("Too Large")

try:
    x=int(input())
    r=fib(x)
    if r!=None: print(r)
except:
    print("Invalid Input")
```

Input : 10

Output :



```
    except TypeError:
        print("Type Error")
    except ValueError:
        print("Value Error")
    except RecursionError:
        print("Too Large")

try:
    x=int(input())
    r=fib(x)
    if r!=None: print(r)
except:
    print("Invalid Input")
```

[3] ✓ 3.5s

... 55

Justification : Proper exception handling prevents crashes when files are missing or contain invalid data.

Clear explanations ensure the program behaves safely and predictably during runtime errors.

Task-4 : Use an AI tool to generate a Python-based login system.

Analyze: Check whether the AI uses secure password handling

practices.

Prompt : Generate a python based login system with secured password handling

Screenshots :

```
#Generate a python based login system with secured password handling
import hashlib,getpass

def h(p): return hashlib.sha256(p.encode()).hexdigest()
db={"user":h("pass123")}
u=input("Username: ")
p=getpass.getpass("Password: ")

print("Login Success" if u in db and db[u]==h(p) else "Login Failed")
```

Input : Username: user
Password: pass123

Output :

```
def h(p): return hashlib.sha256(p.encode()).hexdigest()
db={"user":h("pass123")}
u=input("Username: ")
p=getpass.getpass("Password: ")

print("Login Success" if u in db and db[u]==h(p) else "Login Failed")
```

[5] ✓ 5.2s

```
Login Success
```

Justification : Basic login systems may store passwords in plain text, which is insecure.
Using hashing and input validation improves security and protects user credentials.

Task-5 : Use an AI tool to generate a Python script that logs user activity (username, IP address, timestamp).

Prompt : Generate a python script that logs users activities with username,IP address timestamps to a log file.

Screenshots :

```python
#generate a python script that logs users activities with username,IP address timestamps to a log file.
import socket,datetime

u=input("Username: ")
ip=socket.gethostbyname(socket.gethostname())
t=datetime.datetime.now().isoformat()

open("activity.log","a").write(f"{u},{ip},{t}\n")
print("Logged")
```

Input : 10

Output :

```python
ip=socket.gethostbyname(socket.gethostname())
t=datetime.datetime.now().isoformat()

open("activity.log","a").write(f"{u},{ip},{t}\n")
print("Logged")
```

[6] ✓ 3.3s

```
Logged
```

Justification : Logging sensitive details like full IP addresses can create privacy risks.
Masking or minimizing logged data ensures responsible and privacy-compliant activity tracking.