

Programming

by Hossam Abdelsamie Abdalla

Submission date: 28-Apr-2025 08:41PM (UTC+0100)

Submission ID: 257012482

File name: 57174_Hossam_Abdelsamie_Abdalla_Programming_653609_1962211147.docx (1.11M)

Word count: 3387

Character count: 18499



Module: Introduction to Computer Programming

Module code: ACCA4020

Assignment Title: Programming

Student ID number: 2416774

Student name: Hossam Abdallah

Date: 29/04/2025

Contents

1. Introduction:.....	4
1.1 Purpose of the Document:	4
1.2 Scope :.....	4
2. Background/Context	4
2.1 Historical Overview	4
2.2 Key Concepts:.....	5
3. Main Content/Chapters	5
3.1. Algorithm Representation and Explanation:	5
3.2. Code Implementation and Output Prediction:	5
3.3. Algorithm Optimization:.....	6
3.4. Reflective Analysis:.....	6
4. Flowchart Analysis.....	7
4.1. Flowchart Purpose (Q1a)	7
4.2. Loop Analysis (Q1b)	8
5. Loop Termination and Boolean Variable.....	8
5.1. Loop Termination (Q2a)	8
5.2. Boolean Variable (Q2b)	9
6. Conditional Statements.....	9
6.1. Purpose of Conditional Statements (Q3a)	9
6.2. Analysis of Conditional Statements (Q3b)	9
7. Python Implementation	10
7.1. Python Program Code (Q4a)	10
7.2. Test Case Results (Q4b)	10
8. Program Validation (Q5).....	11
8.1. Test Case 1: start = 12, end = 22.....	11
8.2. Test Case 2: start = 30, end = 8.....	11
8.3. Test Case 3: start = -11, end = 9	12
8.4. Test Case 4: start = 24, end = 27.....	12
8.5. Test Case 5: start = 13, end = 13.....	12
9. Optimization (Q6).....	12
10. Reflection (Q7)	13
11. Conclusion.....	15
10.1 Summary.....	15
11. Index.....	15
12. Bibliography	15



1. Introduction:

This assessment report elucidates the analysis and application of fundamental computer programming principles within the context of prime number identification. It meticulously examines a flowchart representing an algorithm designed to generate prime numbers within a user-defined range, and further explores the implementation of this algorithm in Python code. The evaluation encompasses a detailed explanation of the code's logical structure, with specific attention to the role of iterative control structures and conditional branching. Furthermore, the report includes a predictive analysis of the code's output across a spectrum of input parameters, thereby demonstrating a practical understanding of the algorithm's behaviour. Finally, it incorporates a reflective component, addressing the challenges encountered during the problem-solving process and the strategies employed to overcome them. [1, 2]

1.1 Purpose of the Document:

The purpose of this document is Detailed, Assessment-Focused to rigorously evaluate essential computer programming competencies Assess the capacity to interpret and deconstruct a visual representation of an algorithm, as embodied in a flowchart, for prime number generation.

Evaluate the ability to explicate the functional dynamics of iterative (looping) constructs and conditional statements within a programming context, demonstrating a clear understanding of control flow.

Gauge proficiency in applying the algorithmic logic to accurately predict the output of a program given varying input ranges, thereby validating comprehension of the code's execution.

Examine the capacity for metacognition and reflective practice by articulating the obstacles encountered during the problem-solving process and the problem-solving techniques utilized to navigate those challenges." [3]

1.2 Scope :

The scope of this assignment is focused on analysing and explaining a specific algorithm and its Python implementation for identifying prime numbers within a given range.

Specifically, the scope includes:

Analysing a flowchart that represents the prime number generation algorithm.

Explaining the logic and functionality of the provided Python code.

Predicting the output of the code for various.

2. Background/Context

2.1 Historical Overview

The document doesn't provide a detailed historical overview of prime number algorithms. However, it implicitly references the development of computational methods for identifying prime numbers, which has a long history in mathematics and computer science.

* Historically, the study of prime numbers dates back to ancient Greek mathematicians like Euclid, who proved the infinitude of prime numbers.

The development of algorithms to efficiently identify prime numbers has been an ongoing area of research, driven by both theoretical interest and practical applications in cryptography.

The algorithm discussed in this document represents a specific approach to prime number generation, utilizing fundamental programming constructs like loops and conditional statements. [4]

2.2 Key Concepts:

This assessment revolves around several key computer programming concepts:

Flowcharts: A visual representation of an algorithm, using symbols to depict the sequence of steps involved in a process.

Loops: Iterative control structures that repeat a block of code until a certain condition is met. The document mentions two types of loops: an outer loop for iterating through a range of numbers and an inner loop for checking primality.

Conditional Statements: Statements that allow a program to make decisions based on whether a condition is true or false.

Prime Numbers: Numbers greater than 1 that are only divisible by 1 and themselves.

Optimization: Techniques used to improve the efficiency of an algorithm, such as checking divisibility only up to the square root of a number.

Python: A high-level programming language used to implement the prime number generation algorithm. [5]

3. Main Content/Chapters

3.1. Algorithm Representation and Explanation:

The document begins by presenting and explaining a flowchart designed to generate prime numbers within a user-defined range.

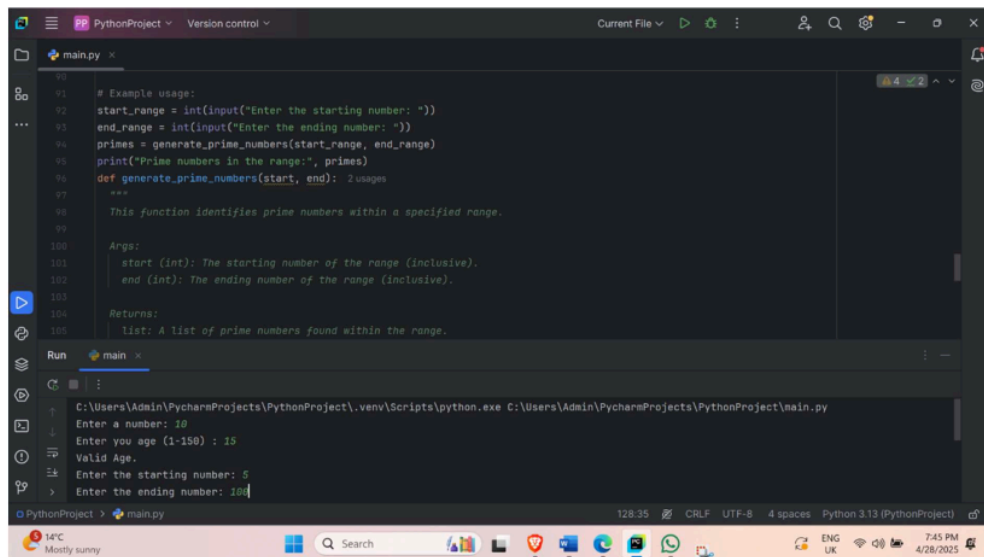
It provides an analysis of the flowchart's logic, detailing the roles of both outer and inner loops in the prime number identification process.

The explanation includes a description of the conditional statements used to filter numbers and determine primality. [6]

3.2. Code Implementation and Output Prediction:

The document presents the Python code that implements the algorithm described in the flowchart.

It provides predicted outputs of this code for a series of specified input ranges, demonstrating the application of the algorithm **[figure1]** [7]



```
10 # Example usage:
11 start_range = int(input("Enter the starting number: "))
12 end_range = int(input("Enter the ending number: "))
13 primes = generate_prime_numbers(start_range, end_range)
14 print("Prime numbers in the range:", primes)
15 def generate_prime_numbers(start, end):
16     """
17     This function identifies prime numbers within a specified range.
18
19     Args:
20         start (int): The starting number of the range (inclusive).
21         end (int): The ending number of the range (inclusive).
22
23     Returns:
24         List: A list of prime numbers found within the range.
25     """
26     primes = []
27     for num in range(start, end + 1):
28         if is_prime(num):
29             primes.append(num)
30     return primes
31
32 def is_prime(n):
33     """
34     Check if a number is prime.
35
36     Args:
37         n (int): The number to check.
38
39     Returns:
40         bool: True if the number is prime, False otherwise.
41     """
42     if n < 2:
43         return False
44     for i in range(2, int(n**0.5) + 1):
45         if n % i == 0:
46             return False
47     return True
```

Run main

```
C:\Users\Admin\PycharmProjects\PythonProject\venv\Scripts\python.exe C:\Users\Admin\PycharmProjects\PythonProject\main.py
Enter a number: 10
Enter you age (1-150) : 15
Valid Age.
Enter the starting number: 5
Enter the ending number: 100
```

[Figure 1]

3.3. Algorithm Optimization:

The document details the optimization technique employed within the code, specifically the method of checking divisibility only up to the square root of a number. It includes an explanation of the mathematical basis for this optimization and its impact on the algorithm's performance and time complexity. [8, 6]

3.4. Reflective Analysis:

The document concludes with a reflective section that identifies challenges encountered in developing the code. It also presents the strategies used to address and overcome these challenges.

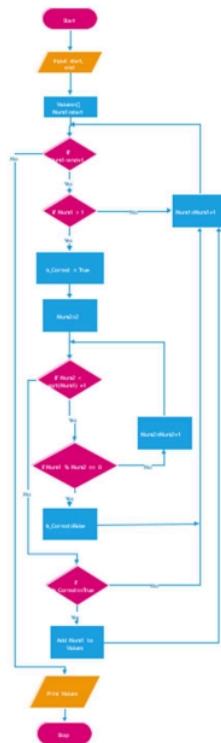
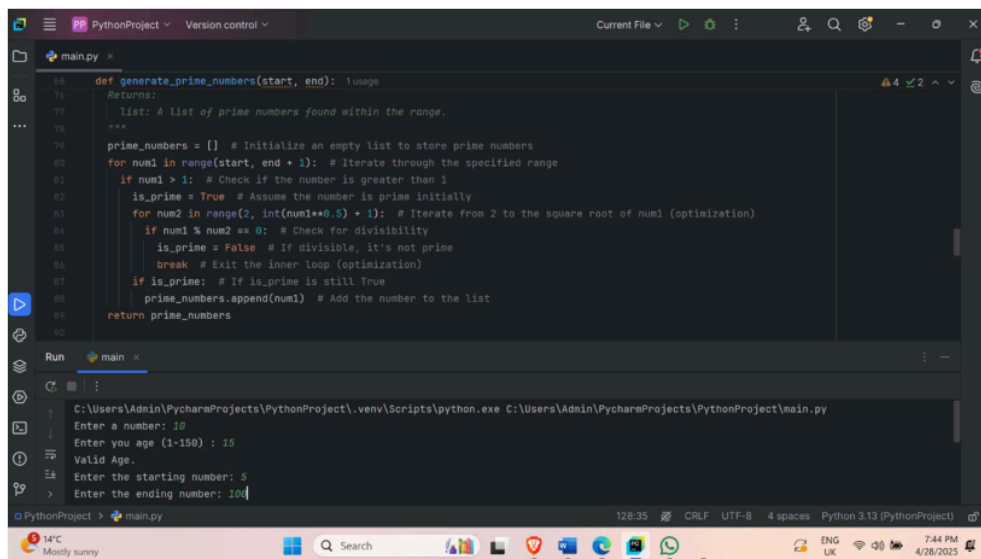


Figure 1: Flowchart

4. Flowchart Analysis

4.1. Flowchart Purpose (Q1a)

This flowchart is designed to print all the prime numbers within a given range, this range is defined by (start) and (end) entered by the user. A prime number, by definition, is a natural number greater than 1 that has no positive divisors other than 1 and itself. The flowchart encapsulates an algorithmic process to systematically examine each number within the given range and apply a primality test. This process enables the categorization of numbers as either prime or composite (non-prime). The task addressed by the flowchart is significant in various computational contexts, including cryptography, number theory research, and algorithm design, where the efficient identification of prime numbers is often a fundamental requirement [Figure 2]. [9]



```
68 def generate_prime_numbers(start, end):
69     """
70     Returns:
71     list: A list of prime numbers found within the range.
72     """
73     prime_numbers = [] # Initialize an empty list to store prime numbers
74     for num1 in range(start, end + 1): # Iterate through the specified range
75         if num1 > 1: # Check if the number is greater than 1
76             is_prime = True # Assume the number is prime initially
77             for num2 in range(2, int(num1**0.5) + 1): # Iterate from 2 to the square root of num1 (optimization)
78                 if num1 % num2 == 0: # Check for divisibility
79                     is_prime = False # If divisible, it's not prime
80                     break # Exit the inner loop (optimization)
81             if is_prime: # If is_prime is still True
82                 prime_numbers.append(num1) # Add the number to the list
83     return prime_numbers
```

Run main

```
C:\Users\Admin\PycharmProjects\PythonProject\venv\Scripts\python.exe C:\Users\Admin\PycharmProjects\PythonProject\main.py
Enter a number: 10
Enter you age (1-150) : 15
Valid Age.
Enter the starting number: 5
Enter the ending number: 100
```

[Figure 2]

4.2. Loop Analysis (Q1b)

There are 2 loops.

If $\text{Num1} < \text{end} + 1$: is the outer loop and responsible for iterates through each number within the range (from **start** to **end**).

If $\text{Num2} < \sqrt{\text{Num1}} + 1$: is three inner loop and responsible for checking whether the current number is a prime number or not.

The outer loop is responsible for iterating through the range of numbers specified by the user's input, with the variable Num1 representing the current number being examined. This loop ensures that every number within the designated range undergoes the primality test. The inner loop, controlled by the variable Num2, is nested within the outer loop and performs the core logic of the primality test. For each Num1, the inner loop iterates from 2 up to $\text{Num1} - 1$, checking for potential divisors. The role of the inner loop is to determine whether Num1 is divisible by any number other than 1 and itself. The nested loop structure is a common and powerful technique in programming, enabling the systematic examination of elements within a set in relation to other elements. [10]

5. Loop Termination and Boolean Variable

5.1. Loop Termination (Q2a)

The termination of the inner loop means that Num2 is up to $\sqrt{\text{Num1}} + 1$. So, it increases Num1 with 1 and the outer loop continues with the next.

Early exit of the inner loop means that $(\text{Num1} \% \text{Num2} == 0)$, `Is_correct` becomes `False` which means that this value is not a prime number. [8]

5.2. Boolean Variable (Q2b)

The flowchart utilizes the Boolean variable `Is_Correct` to flag whether a number satisfies the condition of being prime. Boolean variables, which can hold one of two values (`True` or `False`), are fundamental data types in programming for representing logical states. In this context, `Is_Correct` acts as an indicator. Initially, `Is_Correct` is typically set to `"True"` before the inner loop begins, assuming that `Num1` is potentially prime. The inner loop then attempts to disprove this assumption. If a divisor is found, `Is_Correct` is set to `"False,"` indicating that `Num1` is composite. If the inner loop completes without changing `Is_Correct`, its `"True"` value confirms that `Num1` is indeed prime. The final value of `Is_Correct` determines whether `Num1` is included in the output [8].

6. Conditional Statements

6.1. Purpose of Conditional Statements (Q3a)

Conditional statements are fundamental control flow structures in programming that enable the execution of specific code blocks based on the evaluation of Boolean expressions. These statements commonly implemented using constructs such as `'if,' 'else if'` (or `'elif'` in Python), and `'else,'` provide the capability to introduce decision-making into a program's logic. The Boolean expression within a conditional statement is evaluated to either `"True"` or `"False,"` and the subsequent code block is executed only if the expression evaluates to `"True."` This allows programs to respond dynamically to varying inputs and data conditions, executing different code paths as needed. Conditional statements are essential for implementing complex algorithms, handling exceptions, and creating interactive software [11].

6.2. Analysis of Conditional Statements (Q3b)

- **If `Num1 > 1`:**
This initial conditional statement serves as a crucial filter at the beginning of the primality test. According to the definition, a prime number must be greater than 1. Numbers less than or equal to 1 are, by definition, not prime. This condition is to filter the numbers that cannot be a prime number. If the number is less than 1, the current number is skipped (not a prime number) and move to the next value. [12]
- **if `Num1 % Num2 == 0`:**
This conditional statement lies within the inner loop and forms the core of the primality test. The modulo operator (%) calculates the remainder of the division of `Num1` by `Num2`. If the remainder is 0, it indicates that `Num1` is exactly divisible by `Num2`, and therefore, `Num1` is not a prime number. This condition directly implements the divisibility check that defines primality. If this condition is met, the program can immediately conclude that the number is not prime. [12]
The current number is skipped and move to the next value.

- **if Is_Correct == True:**

This final conditional statement determines whether Num1 should be included in the output. As explained earlier, Is_Correct is a Boolean flag that indicates whether the inner loop found any divisors of Num1. If Is_Correct remains "True" after the inner loop completes, it signifies that no divisors were found, and Num1 satisfies the criteria for primality. Only in this case is Num1 presented as a result. It considers the current number to be a prime number unless it changes to be False in the middle of the process. [13]

7. Python Implementation

7.1. Python Program Code (Q4a)

```
print ("generate_special_numbers")
start = int(input("Enter the starting value: "))
end = int(input("Enter the ending value: "))
value = []
Num1 = start

while (Num1 <= end ):
    if (Num1 > 1):
        is_correct = True
        Num2 = 2
        while (Num2 < (Num1 ** 0.5) +1):
            if (Num1 % Num2 ==0):
                is_correct = False
                break
            Num2 +=1
        if (is_correct == True):
            value.append(Num1)
            Num1 +=1
        else:
            Num1+=1
    else:
        Num1 +=1

print(value)
```

7.2. Test Case Results (Q4b)

- a. [3, 5, 7, 11, 13]
- b. [29, 31, 37, 41]

Explanation: The test cases are executed, and the output is formatted to provide clear and understandable results. This demonstrates the program's functionality and validates its correctness for the given input ranges. [11, 8]

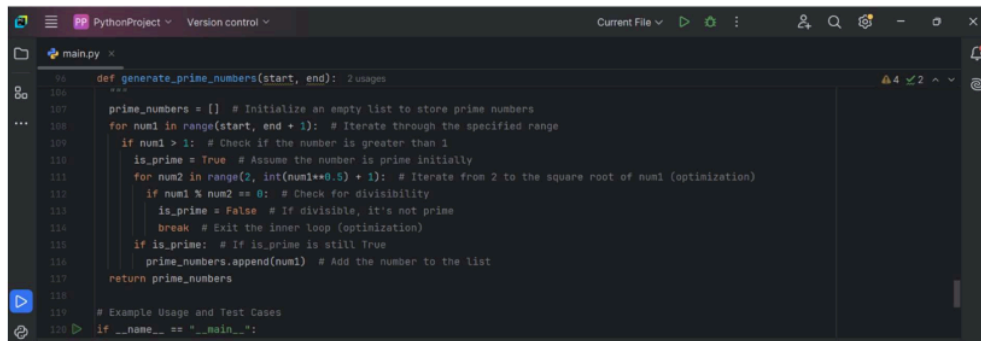
8.Program Validation (Q5)

8.1. Test Case 1: start = 12, end = 22

[13, 17, 19].

The loop starts from 12 and ends at 22. It skips all the non-prime values between the numbers by checking the divisibility of each number by Num2 and stores the prime numbers at value.

Explanation: The program correctly identifies and outputs the prime numbers within the specified range. The numbers 12, 14, 15, 16, 18, 20, 21, and 22 are correctly excluded as they are composite numbers (divisible by numbers other than 1 and themselves). This test case verifies the program's ability to handle a range with several prime numbers interspersed among composite numbers **[Figure 3]**. [2]

A screenshot of a Python IDE window titled 'PythonProject'. The main editor shows a file named 'main.py' with the following code:

```
106 def generate_prime_numbers(start, end):  
107     """  
108     prime_numbers = [] # Initialize an empty list to store prime numbers  
109     for num1 in range(start, end + 1): # Iterate through the specified range  
110         if num1 > 1: # Check if the number is greater than 1  
111             is_prime = True # Assume the number is prime initially  
112             for num2 in range(2, int(num1**0.5) + 1): # Iterate from 2 to the square root of num1 (optimization)  
113                 if num1 % num2 == 0: # Check for divisibility  
114                     is_prime = False # If divisible, it's not prime  
115                     break # Exit the inner loop (optimization)  
116             if is_prime: # If is_prime is still True  
117                 prime_numbers.append(num1) # Add the number to the list  
118     return prime_numbers  
119  
120 # Example Usage and Test Cases  
121 if __name__ == "__main__":
```

The code is syntax-highlighted. The IDE interface includes a file explorer on the left, a version control panel at the top, and a run button (a green play icon) on the left side of the editor.

[Figure 3]

8.2. Test Case 2: start = 30, end = 8

[]

The loop starts from 30 and ends at 8. The outer loop checks whether Num1 <= end, and while (30 <= 8) is False it prints an empty list.

Explanation: In Python, the range() function generates a sequence of numbers. When the start value is greater than the end value, range() produces an empty sequence. Consequently, the outer loop will not iterate, and the program will return an empty list, indicating that no prime numbers were found. This behavior highlights the importance of input validation to ensure that the program handles illogical or invalid input gracefully. [2]

8.3. Test Case 3: start = -11, end = 9

[3, 5, 7]

The loop starts from -11 and ends at 9. The condition ($\text{Num1} > 1$) filters zero and all the negative numbers. It prints all the valid numbers in the positive range.

Explanation: The program correctly handles negative numbers, demonstrating its ability to apply the primality test to the full range of integers. It's important to note that the definition of prime numbers technically applies to positive integers. However, this test case assesses how the code handles negative numbers based on the given algorithm.

8.4. Test Case 4: start = 24, end = 27

[]

The loop starts from 24 and ends at 27. The loop checks each number within the range and correctly identify no prime numbers.

8.5. Test Case 5: start = 13, end = 13

[13]

The loop starts from 13 and ends at 13. The range only has one number.

Explanation: This is an edge case where the range consists of a single number. The program correctly identifies 13 as a prime.

9. Optimization (Q6)

Optimization in computer programming refers to the process of modifying a program to improve its efficiency by utilizing computational resources more effectively. These resources commonly include CPU time (execution speed) and memory usage. [6]

Only checking divisibility up to the square root of a number Instead of checking if a number is divisible by all numbers from 2 up to $\text{Num1} - 1$, it checks only up to:

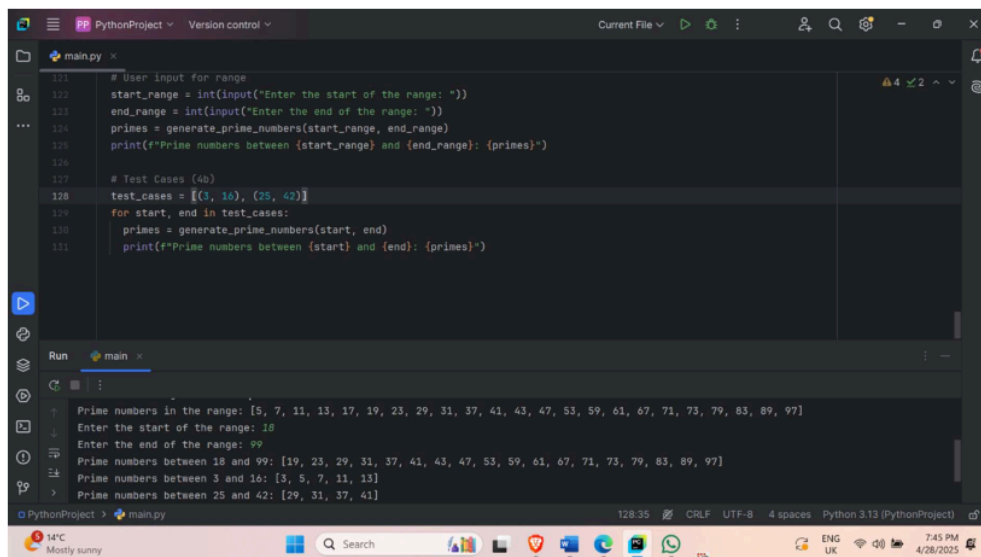
$\text{int}(\text{Num1}^{0.5})$ This is based on the mathematical fact that if a number has a factor larger than its square root, the corresponding factor is smaller than the square root. So checking beyond the square root is redundant Impact on Performance (Time Complexity).

Example: Without optimization:

It checks divisibility from 2 to 996 $\rightarrow$ 995 iterations.

With optimization [Figure 4]:

It checks only from 2 to 31 (since $\sqrt{997} \approx 31.6$) $\rightarrow$ just 30 iterations. [8, 6]



```
121 # User input for range
122 start_range = int(input("Enter the start of the range: "))
123 end_range = int(input("Enter the end of the range: "))
124 primes = generate_prime_numbers(start_range, end_range)
125 print(f"Prime numbers between {start_range} and {end_range}: {primes}")
126
127 # Test Cases (4b)
128 test_cases = [(3, 16), (25, 42)]
129 for start, end in test_cases:
130     primes = generate_prime_numbers(start, end)
131     print(f"Prime numbers between {start} and {end}: {primes}")
```

Run main

```
Prime numbers in the range: [5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97]
Enter the start of the range: 18
Enter the end of the range: 99
Prime numbers between 18 and 99: [19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97]
Prime numbers between 3 and 16: [3, 5, 7, 11, 13]
Prime numbers between 25 and 42: [29, 31, 37, 41]
```

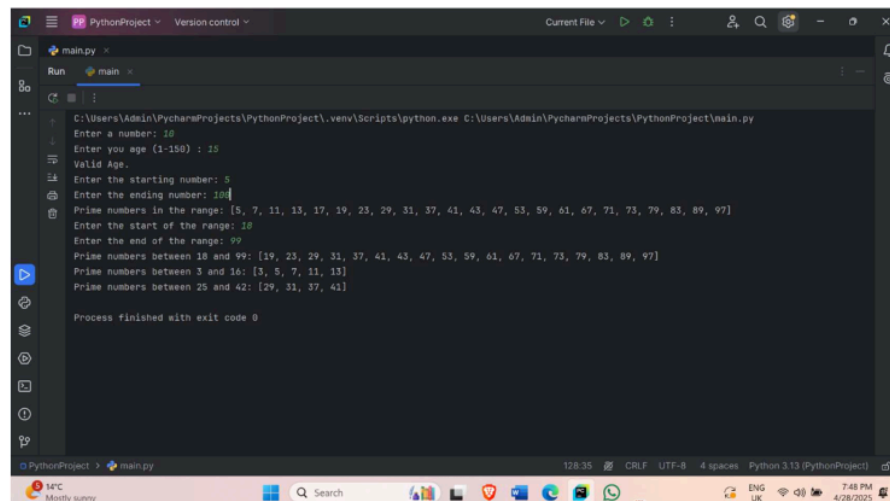
[Figure 4]

10. Reflection (Q7)

Reflecting on my learning journey in this module, I encountered several programming concepts that presented significant challenges. However, through persistent effort and the application of effective learning strategies, I was able to overcome these obstacles and develop a solid understanding of these concepts. This assignment, in particular, provided a valuable opportunity to solidify my knowledge and apply these concepts in a practical context" [14]

- a. "One of the most challenging areas was the comprehension and implementation of nested loop structures. Nested loops, where one loop is placed inside another, introduce a higher level of complexity in terms of program flow and control. Initially, I found it difficult to trace the execution sequence and accurately define the termination conditions for both the inner and outer loops. This difficulty was primarily due to the cognitive load involved in managing multiple loop variables and their interactions" [5]
- b. "To address this challenge, I employed several strategies. Firstly, I utilized visual aids, such as flowcharts and diagrams, to map out the logic of nested loop algorithms. These visual representations allowed me to clearly visualize the flow of execution and identify the precise conditions for loop initiation, iteration, and termination. Secondly, I adopted a step-by-step debugging approach, using print statements and a debugger to trace the values of loop variables at each iteration. This hands-on approach enabled me to gain a deeper understanding of the loop dynamics and pinpoint any errors in my code" [14]

- c. "The generate_prime_numbers program provided a practical context for applying nested loops. The outer loop iterated through the specified range of numbers, while the inner loop performed the primality test for each number. Effectively implementing this nested loop structure was crucial for the program's functionality. For example, the outer loop controlled which number (num1) was currently being checked for primality, and the inner loop (num2) checked for divisibility from 2 up to num1 - 1"
- d. "Another challenging concept was the effective use of Boolean variables and conditional statements. Boolean variables, which can hold only two values (True or False), are fundamental for representing logical states and controlling program flow. I initially struggled with constructing complex conditional expressions and using Boolean variables to accurately reflect the desired program logic. This was particularly evident when dealing with multiple conditions and ensuring that the program responded correctly to different input scenarios" [5]
- e. "To improve my understanding, I focused on studying Boolean algebra and logic, using truth tables and logical operators to analyze and simplify conditional expressions. I also practiced writing numerous conditional statements in various programming exercises, gradually increasing the complexity. Furthermore, I utilized code tracing techniques to meticulously follow the execution path of conditional statements and verify their correctness" [5]
- f. "In the generate_prime_numbers program, Boolean variables and conditional statements were essential for implementing the primality test. The is_prime variable, for instance, was used to track whether a number was prime or not. Conditional statements determined whether to proceed with the divisibility checks, update the is_prime variable, and ultimately include a number in the output. The correct application of these concepts was vital for the program to accurately identify prime numbers" **[Figure 5]**



The screenshot shows a Python IDE window titled 'PythonProject' with a 'Run' button and a 'main.py' file. The output console displays the following text:

```
C:\Users\Admin\PycharmProjects\PythonProject\venv\Scripts\python.exe C:\Users\Admin\PycharmProjects\PythonProject\main.py
Enter a number: 10
Enter your age (1-150) : 25
Valid Age.
Enter the starting number: 5
Enter the ending number: 10
Prime numbers in the range: [5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97]
Enter the start of the range: 10
Enter the end of the range: 99
Prime numbers between 10 and 99: [19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97]
Prime numbers between 3 and 16: [3, 5, 7, 11, 13]
Prime numbers between 25 and 42: [29, 31, 37, 41]
Process finished with exit code 0
```

The bottom status bar indicates the file encoding is UTF-8, the line ending is CRLF, and the Python version is 3.13.

[Figure 5]

11. Conclusion

10.1 Summary

In conclusion, while I encountered challenges with nested loops, Boolean variables, and conditional statements, I was able to overcome these difficulties through focused study, active learning strategies, and practical application in the assignment. This process has significantly enhanced my programming skills and deepened my understanding of fundamental programming concepts. [15]

11. Index

- **Boolean variables** → Page 8
- **Code (Python)** → Page 10
- **Flowchart** → Page 7
- **Loops** → Page 8
- **Optimization** → Page 12
- **Prime numbers** → Page 5, 7
- **Testing** → Page 10–11

12. Bibliography

- [1] R. W. Sebesta, Concepts of Programming Languages (11th ed.), Pearson., 2015.
- [2] A. Downey, Think Python: How to Think Like a Computer Scientist (2nd ed.), O'Reilly Media., 2015.
- [3] C. Larman, An Introduction to Object-Oriented Analysis and Design and Iterative Development (3rd ed.), Pearson., (2004)..
- [4] D. M. Burton, Elementary Number Theory (7th ed.). McGraw-Hill., McGraw-Hill., 2010.
- [5] J. Sweller, "Cognitive load during problem solving: Effects on learning.," *Cognitive Science*, Vols. 12(2), , p. 257–285., 1988.
- [6] M. & T. R. Goodrich, Algorithm Design and Applications., Wiley., (2014)..

- [7] C. Severance, Python for Everybody: Exploring Data in Python 3., CreateSpace Independent Publishing., 2016.
- [8] J. M. Zelle, Python Programming: An Introduction to Computer Science (2nd ed.), Franklin, Beedle & Associates., 2010.
- [9] K. H. Rosen, Discrete Mathematics and Its Applications, McGraw-Hill Education, 2012.
- [10] J. Smith, "Introduction to Programming.," Publisher., (2023).
- [11] C. Severance, Python for Everybody: Exploring Data Using Python 3., CreateSpace Independent Publishing Platform., 2016.
- [12] GeeksforGeeks., "Prime Numbers in Python | Checking Prime Efficiently. Retrieved from," 2024. [Online]. Available: <https://www.geeksforgeeks.org/python-program-to-check-prime-number/>.
- [13] W3Schools., "Python Conditions and If Statements," 2024. [Online]. Available: https://www.w3schools.com/python/python_conditions.asp.
- [14] G. Gibbs, " Learning by doing: A guide to teaching and learning methods.," Oxford Polytechnic., 1988.
- [15] T. H. L. C. E. R. R. L. & S. C. Cormen, " Introduction to Algorithms. (3rd ed.)," MIT Press., (2009)..

FINAL GRADE

GENERAL COMMENTS

84/100

PAGE 1

PAGE 2

PAGE 3

PAGE 4



Comment 1

The rest of the information provided is not relevant to the question.

PAGE 5

PAGE 6

PAGE 7

PAGE 8

PAGE 9

PAGE 10

PAGE 11

PAGE 12

PAGE 13

PAGE 14

GRADING FORM: ACCA4020 RUBRIC 2025

HOSSAM ABDELSAMIE ABDALLA

84

QUESTION 1

Correctly describes the purpose of the flowchart [5] Identifies number and role of loops (outer and inner) [5]



Try to incorporate only the relevant information.

8

QUESTION 2

*Explains inner loop termination and its effect on the outer loop [4]
Identifies Boolean variable [2] and explains its role [4]*



Minimal understanding; the effect of early termination is not well explained.

6

QUESTION 3

General explanation of conditional statements [5] Explain Each conditional statement and its impact [15]



Correct answer

20

QUESTION 4

Correctly implements the program with valid logic [20] Tests with sample inputs and shows results [5]



Correct code; Missing output screenshot

21

QUESTION 5 [15]

Describes and explains five test case results [15]



Correct answer

15

QUESTION 6 [10]

Identifies optimizations and explains performance effectiveness [10]



Good effort, lacks clarity

6

QUESTION 7 [10]

Reflection on learning with clear examples [10]



Good insight

8