

Programming

by Biniyam Memberu

Submission date: 28-Apr-2025 10:58PM (UTC+0100)

Submission ID: 257030046

File name: 57479_Biniyam_Memberu_Programming_653609_1313896804.docx (386.5K)

Word count: 3045

Character count: 16233



Prifysgol Cymru
Y Drindod Dewi Sant
University of Wales
Trinity Saint David

Module: Computer Programming (ACCA4020)

Programme: Foundation (Computing)

Assignment title: Programming

Lecture: Dr. Seena Joseph

Memberu

Student Name: Biniyam

ID Number: 2405357

Contents

Introduction	3
Python programming	3
Statement of the problem	3
Flowchart	3
The purpose of the flowchart	3
The problem or task addressed by the flowchart.....	4
Identify the number of loops in the flowchart and the role of each loop.....	4
Loops	4
Termination Conditions of the Inner Loop:.....	5
Effect on Outer Loop's Processing	5
When the Inner Loop Exits Early.....	5
Boolean variable	5
The Boolean variable used.....	5
Role of the Boolean Variable.....	5
Conditional statements.....	6
Conditional statements in the program.....	6
The purpose of a conditional statement in programming.....	6
Explain how each of the above three conditional statements contributes to the results of the program	7
Trace table.....	9
Test the program with the following inputs and display the results.	10
Validate the program by using test cases.	11
Optimisation Techniques.....	12
Reflection	13
Conclusion.....	13
References.....	14

Introduction

Modern technology uses programming languages to enable developers to turn complex problems into operational solutions. The tool is fundamental for addressing computational problems while enabling industrial innovation. The assignment is the core material for Python studies to examine fundamental programming concepts. The flowchart requires hands-on experience that combines logical reasoning with the program's implementation. The assignment aims to build essential knowledge while teaching practical skills through coding exercises, testing scenarios, and optimisation strategies. The assignment aims to improve understanding of programming language constructs while developing problem-solving abilities. The assignments evaluate theoretical knowledge while demonstrating practical application of concepts to prepare students for future computer science development.

Python programming

Python is a versatile high-level programming language that developers use extensively because of its simplicity, readability, and effectiveness in solving computational problems. The programming language enables developers to concentrate on problem-solving logic rather than technical aspects. The implemented logic follows the flowchart structure that presents a visual breakdown of the number filtering process through its step-by-step approach. The flowchart uses essential programming elements, including loops for repeated operations, conditional statements for making decisions, and variables to store results dynamically. The conversion of the flowchart into Python code demonstrates how a visual representation transforms into functional programming to achieve efficient task execution [1].

Statement of the problem

Create a Python program that follows the same logic, using the flowchart as a guide. The program processes the range of numbers, checks them against the criteria, and outputs a list of qualifying numbers. To solve the problem efficiently, the assignment requires the application of programming constructs such as loops, conditional statements, and variables.

Flowchart

The purpose of the flowchart

The flowchart seems to identify numbers in each range that satisfy certain mathematical conditions. It probably filters numbers based on divisibility and Boolean checks. The flowchart aims to find and check the list of all prime numbers in the user's range. A prime number is a whole number greater than one that cannot be divided by any other whole number other than by itself or one. It checks each number in the range systematically and mathematically to determine if it is a prime number and collects all the prime numbers in a list. It then outputs the list of prime numbers.

The problem or task addressed by the flowchart

The flowchart addresses the problem of determining whether a given number is prime, which is divisible only by one and itself, by automating this process for an entire range of numbers. This task helps to ensure efficiency by automating the prime-checking process instead of manually evaluating each number, accurately applying a consistent algorithm to avoid errors in identifying prime numbers and providing a step-by-step logical structure for the task.

Identify the number of loops in the flowchart and the role of each loop.

Loops are essential for repetitive tasks, such as going through a set of items, working with data, or doing calculations [2]. A flowchart designed to identify prime numbers typically involves two loops.

✓ Outer Loop: Iterating Through the Range

The loop goes through each number in the specified range. It is to check every number in the range for primality. The number to be updated is the current number in each iteration, and the flow chart proceeds to check whether the number is prime or not using the inner loop. Checks each number in the range.

✓ Inner Loop: Checking Divisibility

This loop determines whether the current number (from the outer loop) is divisible by any number other than one and itself. It ensures that the logic for deciding primality is applied systematically. The loop checks potential divisors from 2 to the square root of the current number. If the number is divisible by any of these, it's marked as not prime, and the loop exits early. Tests whether the current number is divisible.

The two loops function together. The outer loop controls the movement through the complete range of numbers, and the inner loop executes the actual primality test for each number. The flowchart evaluates every number in the range through this combined approach.

Loops

The termination conditions of the inner loop in the flowchart play a critical role in the progression of the outer loop.

Termination Conditions of the Inner Loop:

The inner loop exits early if it finds that the current number being evaluated is divisible by any number other than 1 and itself. This means the current number is not prime, and further checks for divisors are unnecessary.

Effect on Outer Loop's Processing

The inner loop termination leads the outer loop to proceed with the following number from the range. The process becomes more efficient because the algorithm avoids unnecessary calculations when it determines a number is not prime, it exits early.

The program saves resources by terminating the inner loop early because it eliminates unnecessary condition checks. The inner loop stops its execution after verifying divisibility by 2. For example, for a number like 12, the inner loop exits after finding divisibility by 2 without checking divisors like 3 and 4. This means it saves time efficiently, thus avoiding further checks for divisors 3 and 4. The algorithm becomes efficient because it saves time.

The number receives the prime status when the inner loop finishes its entire run without detecting any divisors except for 1 and itself. After completing this number verification, the outer loop includes it in the list of prime numbers before proceeding to the following number.

When the Inner Loop Exits Early

The program uses an early exit from the inner loop to indicate that the current number is not prime. It avoids additional checks and moves directly to the following number. The inner loop finishes its execution without an early exit, which confirms that the number is prime and allows the outer loop to move on to the following number.

Boolean variable

The Boolean variable used

Boolean variables are data types that accept True or False as possible values. They serve as fundamental elements for decision-making and control flow operations in programming through conditional statements and loops[3] In a flowchart designed to identify prime numbers, the Boolean variable is often used as a flag to track whether a number is prime or not prime during the evaluation process.

The Boolean variable typically has two possible values

True: Indicates that the current number is a prime number

False: Indicates that the current number is not a prime number

Role of the Boolean Variable

- ✓ **Tracking**

The flowchart examines divisors of the current number while the Boolean variable monitors for any divisors except 1 and itself. The prime set starts as True because the number is initially assumed prime. The variable becomes false when a divisor is discovered.

✓ **Decision Making**

After conducting divisibility tests, the value of the prime gets checked. The number is confirmed as prime and added to the list of prime numbers when the prime remains true. The number gets skipped because it is not a prime when prime is false.

✓ **Optimizing the Process**

The Boolean variable allows early termination of the inner loop. If the prime set is false, then a divisor has been found and there is no need to check further divisors.

Conditional statements

Conditional statements in the program

These conditional statements each play distinct roles in determining a program's logic flow.

If Num1 > 1

The condition examines whether the variable Num1 exceeds 1. Identifying prime numbers would guarantee that only numbers above 1 are considered prime because numbers 1 and below are not prime; it should be whole numbers greater than one.

If Num1 % Num2 == 0

The condition verifies whether Num1 can divide Num2 without any remainder. The logic uses this condition to check for any divisors of Num1 besides 1 and itself during prime checking. The condition returns true when Num1 has divisors other than 1 and itself, thus showing Num1 is not prime.

If is_Correct == True

The condition checks for the Boolean variable Is_Correct, which is set to True. It could serve as a flag to verify whether a specific condition or state in the program is correct. For instance, this could be used after checking for divisibility, where Is_Correct tracks the correctness of an operation or result.

The purpose of a conditional statement in programming

Conditional statements are a programming feature that enables programs to choose actions based on conditions. The statement evaluates logical expressions about conditions to determine truth or falseness before running corresponding code blocks. A conditional statement in

programming enables decision-making by allowing a program to execute specific blocks of code based on whether certain conditions are met [1].

Decision-Making: Conditional statements check conditions to run code only when the condition evaluates to True during decision-making operations [4].

Control Flow: The program uses conditional statements to establish its execution sequence, producing different results based on the processed data or inputs [4].

Flexibility and Adaptability: Conditional statements allow programs to gain flexibility by enabling developers to handle multiple situations without writing explicit output code for each scenario [4].

Error Handling: Conditional statements function as error-handling mechanisms because they detect abnormal conditions to prevent system crashes and unexpected behaviour [4].

Logic Building: The statements function to build logical operations which solve problems and process data through algorithm implementation [4].

Explain how each of the above three conditional statements contributes to the results of the program

if Num1 > 1

The condition functions to verify only numbers above 1 for primality. The program uses this filter to exclude numbers less than or equal to 1 because these numbers are not considered prime. The program stops further evaluation when Num1 equals 0 or 1. By excluding numbers that cannot be prime, the program prevents wasteful computations.

```
if num1>1:
```

If Num1 % Num2 == 0

The condition verifies whether Num1 can divide Num2 without any remainder. The condition being True indicates Num1 has divisors besides 1 and itself, thus it cannot be considered prime. For example, if Num1 = 9, then $9 \% 3 == 0$, so 9 is not prime.

The condition is the fundamental principle for determining prime numbers through problem-solving and data processing based on logical decisions. When satisfied with this condition, the program can stop additional checks for that number, enhancing efficiency.

```
if num1 % num2==0:
```

If is_Correct == True

This condition evaluates whether a specific Boolean variable (is_Correct) is set to True. The variable likely acts as a flag to confirm the outcome of the primality test or some other

validation step. For example, if `is_Correct = True`, the number has been verified to meet the required condition. It adds a layer of verification to the program, which could ensure the output is accurate. It confirms the correctness of the results, particularly in cases where additional validation or error handling is required.

```
if is_correct:
```

Implement the Python program named `generate_special_numbers`

The program should have a criterion:

- Accept two inputs, start and end, defining the range of numbers to be processed.
- Implement the logic to filter numbers according to the criteria described in the flow chart.
- Print a list of numbers that meet the criteria.

The program requires users to provide start and end values, determining the range of numbers to process. The program eliminates all numbers that are 1 or below. The program checks for divisors starting from 2 to the square root of the number for numbers above 1 (optimized prime-checking logic). If no divisors are found, the number is considered "special" and added to the result list.

```
36 import math
37 def generate_special_number(start,end): 1 usage
38     value= []
39     for num1 in range(start,end+1):
40         if num1>1:
41             is_correct=True
42             for num2 in range(2,int(math.sqrt(num1))+1):
43                 if num1 % num2==0:
44                     is_correct=False
45                     break
46             if is_correct:
47                 value.append(num1)
48     print('The special number in the range:',value)
49     generate_special_number( start: 2, end: 10)
50
```

Prints the list of special numbers in the specified range.

```
The special number in the range: [2, 3, 5, 7]
```

Trace table

A trace table is a manual tool to monitor variable values and program execution flow within algorithms and programs. The tool enables step-by-step data processing observation, particularly in loops, conditional statements, and logical operations [5].

This is a trace table for the generate_special_numbers program, which processes numbers from 3 to 6. It demonstrates how the program identifies special numbers (prime numbers).

Num	Num>1	is_Correct	Divisor	Num% Divisor	Action	Output
3	TRUE	TRUE	2	3%2=1	Continue Loop	
			3	loops end	No Divisor found	[3]
4	TRUE	TRUE	2	4%2=0	Continue Loop	NO append
5	TRUE	TRUE	2	5%2=1	Continue Loop	
			3	5%3=2	Continue Loop	
			4	5%4=1	Continue Loop	
			5	loops end	No Divisor found	[3,5]
6	TRUE	TRUE	2	6%2=0	No Divisor found	NO append

In the above table, numbers 4 and 6 immediately exit early from the inner loop when a divisor is found. Numbers 3 and 5 run entirely in the inner loop until no divisors are found. The final output for the special number is [3,5].

Test the program with the following inputs and display the results.

start = 3, end = 16

```
import math
def generate_special_number(start,end): 1 usage
    value= []
    for num1 in range(start,end+1):
        if num1>1:
            is_correct=True
            for num2 in range(2,int(math.sqrt(num1))+1):
                if num1 % num2==0:
                    is_correct=False
                    break
            if is_correct:
                value.append(num1)
    print('The special number in the range:',value)
generate_special_number( start: 3, end: 16)
```

```
The special number in the range: [3, 5, 7, 11, 13]
```

start = 25, end = 42

```
import math
def generate_special_number(start,end): 1 usage
    value= []
    for num1 in range(start,end+1):
        if num1>1:
            is_correct=True
            for num2 in range(2,int(math.sqrt(num1))+1):
                if num1 % num2==0:
                    is_correct=False
                    break
            if is_correct:
                value.append(num1)
    print('The special number in the range:',value)
generate_special_number( start: 25, end: 42)
```

```
The special number in the range: [29, 31, 37, 41]
```

Validate the program by using test cases.

Test case 1: start = 12 and end = 22

```
The special number in the range: [13, 17, 19]
```

The program correctly checks each number in the range.

Numbers 12, 14, and 15 are excluded because they have divisors other than 1 and themselves.

The list [13, 17, 19] is printed as the result.

Test case 2: start = 30 and end = 8

```
The special number in the range: []
```

The range is invalid. The starting number must be less than or equal to the ending number.

The output is empty.

Test case 3: start = - 11 and end = 9

```
The special number in the range: [2, 3, 5, 7]
```

The program will process numbers from -11 to 9. Since prime numbers must be greater than 1, only [2, 3, 5, 7] are included in the result.

The program correctly excludes numbers less than or equal to 1.

It checks numbers from 2 to 9, identifies primes, and outputs [2, 3, 5, 7].

Test case 4: start = 24 and end = 27

```
The special number in the range: []
```

The program processes numbers from 24 to 27. None of these numbers are prime, so the output will be an empty list: "Special numbers in the range: []".

The program evaluates each number in the range:

24: Divisible by 2, not prime.

25: Divisible by 5, not prime.

26: Divisible by 2, not prime.

27: Divisible by 3, not prime.

The program correctly outputs an empty list.

Test case 5: start = 13 and end = 13

```
The special number in the range: [13]
```

The program processes only the prime number 13. The output will be "Special numbers in the range: [13]."

Since the range includes a single number, the program checks and confirms it's prime.

The result is [13], correctly identifying 13 as a special number.

Optimisation Techniques

Square Root Optimization

The program verifies divisors within the range of the square root value of the number ($\text{int}(\text{math.sqrt}(\text{num1})) + 1$) instead of checking every number from 2 to $\text{num1} - 1$. According to mathematical principles, every number with a divisor above its square root must also have a smaller divisor. The optimisation cuts down the number of iterations needed in the inner loop. Without this optimisation, the inner loop would perform (num1 divisors to check) iterations for each number. The optimisation cuts down the number approximately.

The inner loop needs to run 99 times when num1 equals 100 without optimisation, but it only needs to run 10 times with the optimisation because $\text{sqrt}(100)$ equals 10. The optimisation cuts down the time complexity.

Early Exit with a break

The program exits the inner loop when a divisor is found (if $\text{num1} \% \text{num2} == 0$), thus avoiding further unnecessary checks. The flag `is_correct = False` is set, and the loop exits early. This saves time for non-prime numbers, as divisors are usually found early in the iteration. For instance, for $\text{num1} = 10$, the inner loop exits after finding that 10 is divisible by two.

Filtering Numbers

The initial check excludes numbers 1 and below (when $\text{num1} > 1$) to prevent unnecessary divisibility tests for invalid candidates. The process becomes more efficient by avoiding unnecessary computations on numbers that cannot be prime.

Reflection

One of the challenging tasks in this assignment required me to tackle loops embedded within other loops. I handled nested loops by optimising them to prevent performance problems. I enhance my skills by dividing complex problems into smaller modules that are easier to handle. I developed skills using loop ranges, such as taking the square root for divisor tests and writing break statements to stop the loop prematurely. The prime number generation uses a loop to check divisors only up to the square root of the number for maximum efficiency. I have gained additional knowledge about Boolean variables (e.g., `is_correct`).

Conclusion

The assignment worked to implement essential programming concepts about conditional statements, loops, design, and optimisation techniques to create an efficient special number generator (prime numbers). Real-world problem-solving was reinforced by testing program functionality through test cases demonstrating the need for performance optimisation. The acquired knowledge demonstrates Python's effectiveness for logical problem-solving and shows how to create scalable solutions through transparent, modular, and efficient code. The path of exploration now leads to more complex programming techniques, which will be studied in upcoming projects.

References

- [1] "W3Schools," [Online]. Available: www.w3schools.com/python/python_intro.asp. [Accessed 02 04 2025].
- [2] D. David, "DPS Computing," 01 03 2021. [Online]. Available: www.dpscomputing.com/blog/2012/09/13/programming-the-purpose-of-loops/. [Accessed 08 04 2025].
- [3] "Real Python," [Online]. Available: www.realpython.com/python-boolean. [Accessed 08 04 2025].
- [4] "PYTHON," Python, [Online]. Available: www.python.org. [Accessed 05 04 2025].
- [5] "101 Computing net/using trace table," 05 10 2019. [Online]. Available: www.101computing.net/using-trace-tables/. [Accessed 09 04 2025].

Programming

GRADEMARK REPORT

FINAL GRADE

GENERAL COMMENTS

85 / 100

PAGE 1

PAGE 2

PAGE 3



Comment 1

Is this information significant?

PAGE 4



Comment 2

??

PAGE 5



Comment 3

Is it divisible by any number?



Comment 4

Boolean variable given in the algorithm and its role? Not the general explanation of the Boolean variable.

PAGE 6

PAGE 7



Comment 5

??

PAGE 8



Comment 6

need to provide more explanation

GRADING FORM: ACCA4020 RUBRIC 2025

BINIYAM MEMBERU

82

QUESTION 1

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



Correct answer provided

10

QUESTION 2

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



Information about the Boolean variable is incomplete.

6

QUESTION 3

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



In some sections, it lacks clarity

17

QUESTION 4

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



Correct code provided and sample results are provided. Input values are hardcoded.

22

QUESTION 5 [15]

Describes and explains five test case results [15]



All test cases are explained well

15

QUESTION 6 [10]

Identifies optimizations and explains performance effectiveness [10]



Good effort, lacks clarity

7

QUESTION 7 [10]

Reflection on learning with clear examples [10]

Basic mention of challenges

