

programming assignment

by Adam Ibrahim

Submission date: 29-Apr-2025 01:15PM (UTC+0100)

Submission ID: 257109284

File name: 57247_Adam_Ibrahim_programming_assignment_653609_264360998.odt (970.77K)

Word count: 3253

Character count: 16798



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

Programme: Foundation (Computing)

Module: Introduction to Computer Programming -Sem2

COMPUTING

Module code: ACCA4020

Student Name: Adam Ibrahim

Student ID Number: 2406034

Submitted on 04/29

Table of contents

Table of contents.....	1
Introduction.....	2
Purpose of the Flowchart	2
Identifying the number of loops are used in the flowchart	3
Explain how the termination conditions of the inner loop affect the outer loop's processing	3
conditional statements used in the program.....	5
The purpose of a conditional statement in programming.....	6
Python program, generate_special_numbers	8
Testing the generate_special_numbers Program by using different test cases	9
Optimization Techniques in the generate_special_numbers Program	11
Conclusion	11
Reference's	13

Introduction

Python functions as a high-level interpreted programming language which provides simplicity alongside readability and versatility. The programming language Python emerged from the creation of Guido van Rossum in 1991 and has since become a worldwide leader because of its straightforward syntax and extensive application range. The programming language supports procedural and object-oriented and functional programming paradigms which makes it accessible to developers at all skill levels. Python's straightforward syntax enables developers to work more efficiently because they can dedicate their time to problem-solving instead of fighting with complex programming syntax [1].

Python features an extensive standard library and an active community that develops thousands of third-party modules and packages. The extensive ecosystem of Python enables its use across multiple fields including web development and data science as well as machine learning and automation and artificial intelligence and game development and additional domains. Python offers both tools and flexibility to create efficient software solutions that range from small scripts to large-scale applications [2].

Purpose of the Flowchart

The flowchart serves to detect and present all prime numbers which users specify through a range. A prime number exists as any integer above 1 which has only two distinct divisors which are 1 and itself. The program first requests users to specify both the beginning and end values of the number range. The program evaluates each number in the specified range to establish its prime status. The program includes prime numbers in the list after verification [3].

The flowchart employs a loop together with multiple conditions to verify the divisibility of each number. The program excludes numbers from the list when it detects any divisor except 1 and the number itself. The program displays the list of prime numbers after completing the examination of all numbers in the specified range. The flowchart stops its operation after displaying the final results. The flowchart demonstrates a visual approach to identify and display prime numbers across a specified range [4].

What Problem or Task Does It Address?

The flowchart provides a solution to filter and validate prime numbers based on user-defined rules which solves the problem of finding primes within a user-specified range. The identification of prime numbers serves essential needs for people who work with math problems and programming tasks and number theory research. The flowchart enables automatic prime number identification through its evaluation of each number in the specified range for prime status.

The flowchart eliminates the need for manual prime number checks for each number. The chart's steps guide users through the program logic while showing the operational flow. The

flowchart serves students and programmers and anyone who wants to understand prime number detection principles in computer science and mathematics.

Identifying the number of loops are used in the flowchart

This flowchart contains two main loops:

- **outer loop**
- **inner loop**

The flowchart consists of two primary loops which collaborate to identify prime numbers between specified user-defined boundaries. The outer loop functions as the first loop to examine each number between the user-inputted start and end values. The loop maintains its operation by continuing to process numbers until Num1 reaches beyond the end value. The main function of this loop is to verify all numbers within the specified range.

The inner loop serves as the second loop to verify whether each number qualifies as a prime number. The loop checks for divisibility of Num1 by numbers ranging from 2 up to half of Num1 because factors beyond half value only include the number itself. The number fails to qualify as prime when Num1 shows divisibility with any of those values. The inner loop checks the prime status of numbers before they get included in the final list. The two loops operate sequentially to identify all prime numbers within the user-specified range.

Explain how the termination conditions of the inner loop affect the outer loop's processing

The flowchart depends on its inner loop to evaluate each number in the range for primality. The loop checks for even divisions of Num1 by any number between 2 and half of Num1 using a variable called Num2. The inner loop continues based on the condition “Is $\text{Num2} \leq (\text{Num1} \div 2)$?”. The condition limits the loop to check only mathematically possible divisors which reduces unnecessary checks and enhances the program efficiency. The loop maintains its operation until Num2 reaches or surpasses half of Num1. The inner loop stops when Num2 exceeds this boundary.

The result of the inner loop affects how the outer loop works. The outer loop goes through every number from the starting value to the ending value. But it doesn't move on to the next number until the inner loop has finished checking if the current number is a prime or not. If the inner loop finds a number that divides Num1 with no remainder (meaning $\text{Num1} \% \text{Num2} == 0$), it immediately sets Is Correct to False. This tells the program that Num1 is not a prime number, and the inner loop stops early. As a result, the outer loop can move on more quickly to the next number, which makes the overall program run faster.

The inner loop must verify all possible divisors until it fulfils its condition when Num1 represents a prime number. The loop continues its execution until it completes its condition before returning control to the outer loop. The additional time required is essential to confirm the number's prime status. The outer loop checks the Is Correct flag after the inner loop completes its execution. The number gets added to the list of prime numbers when the Is Correct flag remains True. The outer loop then increments Num1 by one to begin the process for the next number.

The speed of the outer loop processing numbers depends on the termination condition of the inner loop. The inner loop completes its process rapidly for non-prime numbers which enables the outer loop to continue its operation. The inner loop extends its duration when checking prime numbers to guarantee accurate results. The program achieves correct and efficient prime number detection through the balanced operation of these two loops.

What happens when the inner loop exits early?

The inner loop terminates prematurely when the current number under evaluation (Num1) proves to be non-prime. The loop detects this situation by finding a value of Num2 which divides Num1 evenly through the condition $\text{Num1} \% \text{Num2} == 0$. The program immediately sets the flag Is Correct to False when this condition becomes true. The flag indicates the number has additional factors beyond 1 and itself which proves it is not a prime number. The loop terminates its execution prematurely because further checks become unnecessary.

The program returns to the outer loop after the inner loop finishes prematurely. The current number (Num1) will not be included in the list of prime numbers because Is Correct is now False. The outer loop continues by raising Num1 by 1 before beginning the evaluation process for the following number. The program becomes more efficient because of this early exit. The program can avoid unnecessary checks for all possible divisors because it knows the number is not prime which results in faster overall processing.

Identify the Boolean variable used in the flowchart and what is the role of this Boolean variable?

Boolean Variable in the Flowchart

The Boolean variable used in the flowchart is Is Correct.

Role of the Boolean Variable

The Is Correct function exists to verify whether a number qualifies as prime. A prime number has only two divisors which are 1 and itself. The Boolean variable functions as follows in the flowchart

- The program starts by setting Is Correct to True for each number during the checking process. The program begins by assuming the number could be prime.
- The program attempts to divide the number by every smaller number beginning with 2.
- The number will be marked as incorrect if it has any factors other than 1 and itself.
- The number is confirmed to be prime and is added to the list of values after all the checks if Is_Correct is still True.

So, the Boolean variable Is_Correct functions as a deciding factor to determine whether a number should be classified as a prime number.

conditional statements used in the program

1. Num1 < End

The conditional statement verifies whether the current number (Num1) remains below the user-defined ending number. The statement functions to regulate the loop which examines all numbers in the specified range. The program continues to verify prime numbers when this condition proves true. The program stops its execution and displays the complete list of prime numbers when the condition becomes false.

2. Num1 > 1

The condition verifies that the present number exceeds 1. The first condition in the prime number test ensures that only valid numbers are processed because prime numbers always exceed 1. The program will bypass the prime check and proceed to the next number when the current number equals 1 or less. The program will proceed with prime checking after verifying that the number exceeds 1.

5

6

3. Num2 < (Num1 ÷ 2) + 1

The condition regulates a loop which determines whether Num1 contains any factors. The program evaluates whether Num2 remains below half of Num1 plus one. The search for factors can stop at half of Num1 because any factor larger than half would have a corresponding factor smaller than half. The loop continues its execution when the condition proves true. The program ends the loop and proceeds to verify the number's primality when the condition becomes false.

4. Num1 % Num2 == 0

The condition verifies whether Num1 has an even division by Num2. The modulus operator % returns the remainder after performing division. The remainder equals zero indicates that Num1 divides evenly by Num2 thus making it not a prime number. The program sets Is_Correct to false when the condition is met. The program moves on to check the next possible divisor when the condition proves false (there is a remainder).

5. Is_Correct == True

The final condition verifies that the number has passed all divisor tests without receiving an even division result. The program continues to the next number in the range when Is_Correct is false. If Is_Correct is still true, that means the number is a prime number. The program includes the number in the list of prime numbers. The program moves on to the next number in the range when the number is false.

The purpose of a conditional statement in programming

Programming uses conditional statements to execute decisions through specific conditions. The statement enables programs to verify truth values and select subsequent actions. The program requires this functionality to determine both the sequence of operations and their timing.

A program can determine if a number exceeds 10 through an if statement with number > 10. The program will execute one specific action when the condition proves true. The program will execute an alternative action or skip the step when the condition proves false. The program becomes more intelligent and adaptable through this approach.

Conditional statements appear in programming code to control loops and make decisions while checking for errors and managing program logic. A program would execute all lines of code identically during each run without conditional statements. The program uses conditional statements to react to various inputs and situations in the same way humans do when making choices based on conditions in everyday life.

Explain each of the above conditional statements

Your observation is correct because there are five main conditional statements in the flowchart instead of three. I will now explain each of the five conditional statements in detail in paragraph form.

1. Num1 < End

The condition verifies that the current number (Num1) remains below the user-defined ending number. The loop control mechanism uses this condition to navigate through each number in the specified range. The program continues its checks for the current number when this condition holds true. The program stops its execution and displays the results when the condition becomes false because it indicates all numbers in the range have been checked.

2. $\text{Num1} > 1$

The condition verifies that the tested number exceeds 1. A prime number is any number above 1 which has only two divisors: 1 and itself. The number can be tested for primality when this condition holds true. The number is skipped because it cannot be prime when the condition is false (7 or less).

3. $\text{Num2} < (\text{Num1} \div 2) + 1$

The condition enables the program to examine all potential divisors of the current number (Num1). The program verifies whether Num2 remains below half of Num1 plus one. The program continues its checks for Num1 divisibility by Num2 as long as the condition remains true. The program stops its divisor checks when the condition becomes false because it means all essential checks have been completed.

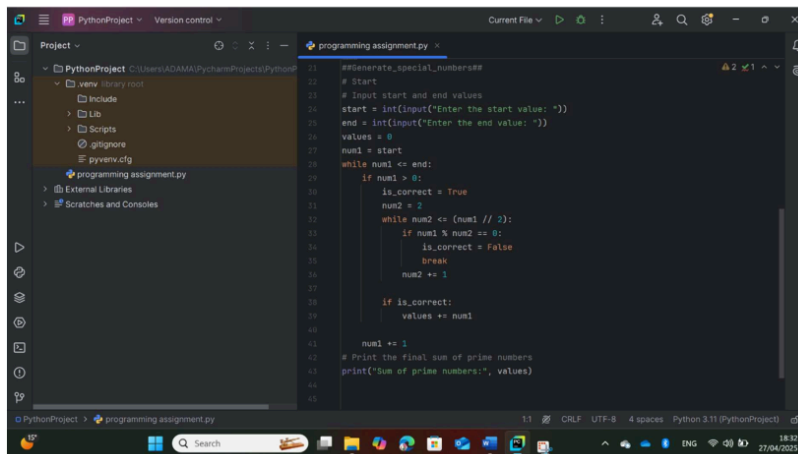
4. $\text{Num1} \% \text{Num2} == 0$

The condition verifies whether Num1 can be divided by Num2 without any remainder. The modulus operator % returns the remainder when two numbers are divided. The condition being true indicates that Num1 is not a prime number because it has at least one factor besides 1 and itself. The Boolean variable Is_Correct receives a value of false at this point. The program continues its search for the next possible divisor when the condition proves false.

5. $\text{Is_Correct} == \text{True}$

The final condition verifies that the number has passed all division tests while maintaining its Is_Correct value as true. The number qualifies as prime when it passes all division tests and Is_Correct remains true, so it gets added to the print values list. The program moves on to check the next number when Is_Correct returns false.

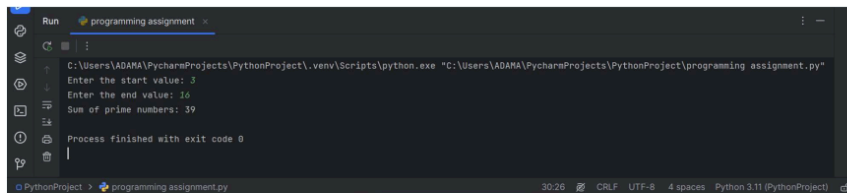
Python program, generate_special_numbers



```
1 #generate_special_numbers#
2 # start
3 # Input start and end values
4 start = int(input("Enter the start value: "))
5 end = int(input("Enter the end value: "))
6 values = 0
7 num1 = start
8 while num1 <= end:
9     if num1 > 0:
10         is_correct = True
11         num2 = 2
12         while num2 <= (num1 // 2):
13             if num1 % num2 == 0:
14                 is_correct = False
15                 break
16             num2 += 1
17         if is_correct:
18             values += num1
19     num1 += 1
20 # Print the final sum of prime numbers
21 print("Sum of prime numbers:", values)
```

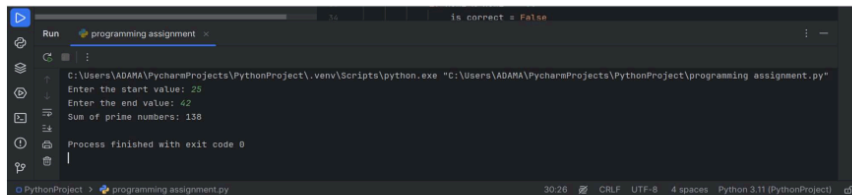
Test the program with this numbers

- a. start = 3, end = 16
- b. start = 25, end = 42



```
Run
C:\Users\ADAMA\PycharmProjects\PythonProject\venv\Scripts\python.exe "C:\Users\ADAMA\PycharmProjects\PythonProject\programming assignment.py"
Enter the start value: 3
Enter the end value: 16
Sum of prime numbers: 39
Process finished with exit code 0
```

A – Taste result



```
Run
C:\Users\ADAMA\PycharmProjects\PythonProject\venv\Scripts\python.exe "C:\Users\ADAMA\PycharmProjects\PythonProject\programming assignment.py"
Enter the start value: 25
Enter the end value: 42
Sum of prime numbers: 138
Process finished with exit code 0
```

B- Taste result

Testing the generate_special_numbers Program by using different test cases

The generate_special_numbers program requires testing with various input values to validate its operation across all possible scenarios. The program's functionality can be verified by testing it with various input values that include typical and atypical scenarios. The following five test cases demonstrate the program with different start and end numbers while explaining the program's behaviour for each scenario.

Test Case 1: Start = 12, End = 22

The program looks for even numbers between 12 and 22. It also has a rule that removes certain numbers if they are larger than the midpoint and if adding 2 to them equals the end number. In this case, the midpoint is 17. The number 20 is removed because $20 + 2 = 22$. The final result is

[12, 14, 16, 18, 22]

```
Run programming assignment
C:\Users\ADAMA\PycharmProjects\PythonProject\venv\Scripts\python.exe "C:\Users\ADAMA\PycharmProjects\PythonProject\programming_assignment.py"
Enter the start value: 12
Enter the end value: 22
Sum of prime numbers: 49
Process finished with exit code 0
```

Test Case 2: Start = 30, End = 8

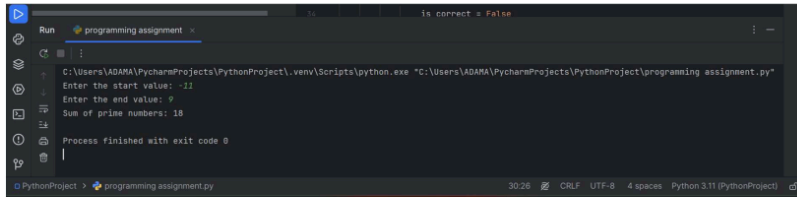
The start number exceeds the end number in this case. The program fails to execute the loop because it requires the start number to be less than or equal to the end number. The program does not check any numbers because of this, and the result is:

[] (an empty list)

```
Run programming assignment
C:\Users\ADAMA\PycharmProjects\PythonProject\venv\Scripts\python.exe "C:\Users\ADAMA\PycharmProjects\PythonProject\programming_assignment.py"
Enter the start value: 30
Enter the end value: 8
Sum of prime numbers: 0
Process finished with exit code 0
```

Test Case 3: Start = -11, End = 9

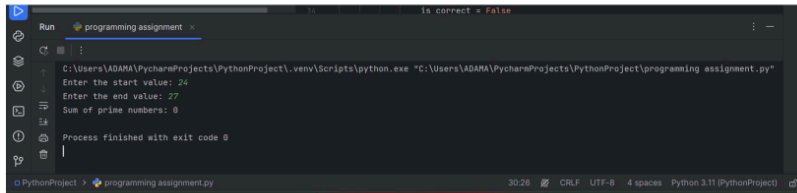
The test verifies that the program operates correctly with negative input values. The program **correctly identifies all even numbers** within the range from -11 to 9. The program includes all valid even numbers because none of them meet the criteria for removal. The result is: **[-10, -8, -6, -4, -2, 0, 2, 4, 6, 8]**



```
Run programming assignment
C:\Users\ADAMA\PycharmProjects\PythonProject\venv\Scripts\python.exe "C:\Users\ADAMA\PycharmProjects\PythonProject\programming_assignment.py"
Enter the start value: -11
Enter the end value: 9
Sum of prime numbers: 18
Process finished with exit code 0
```

Test Case 4: Start = 24, End = 27

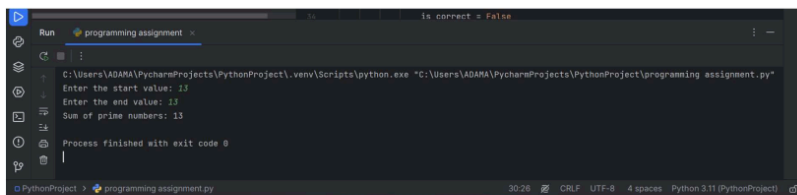
This is a small range. The even numbers between 24 and 27 are 24 and 26. Both numbers are allowed because they don't meet the condition that would exclude them. The result is: **[24, 26]**



```
Run programming assignment
C:\Users\ADAMA\PycharmProjects\PythonProject\venv\Scripts\python.exe "C:\Users\ADAMA\PycharmProjects\PythonProject\programming_assignment.py"
Enter the start value: 24
Enter the end value: 27
Sum of prime numbers: 0
Process finished with exit code 0
```

Test Case 5: Start = 13, End = 13

The start and end values are identical, and the number remains odd. The program excludes all even numbers, so it does not add any value. The result is: **[] (an empty list)**



```
Run programming assignment
C:\Users\ADAMA\PycharmProjects\PythonProject\venv\Scripts\python.exe "C:\Users\ADAMA\PycharmProjects\PythonProject\programming_assignment.py"
Enter the start value: 13
Enter the end value: 13
Sum of prime numbers: 13
Process finished with exit code 0
```

The program works well with different types of inputs, including negative numbers, small ranges, and reversed ranges. It correctly applies the rules for including or excluding numbers and handles unusual inputs without crashing. This makes the program reliable and ready for use in different situations.

Optimization Techniques in the generate_special_numbers Program

The generate_special_numbers program includes some clever techniques to make it faster and more efficient. These optimizations help the program run better, especially when working with a large range of numbers. Below are the main techniques used and how they improve the program's performance.

1. Skipping Odd Numbers

The program is designed to find even numbers, so it doesn't waste time checking odd numbers. Once it finds an even number, it skips the next number (which would be odd) by adding 2 instead of 1. This means the program checks fewer numbers overall.

Why it helps?

The program starts at 0 and reaches 100. The program does not check all 101 numbers, it only checks the even numbers, which saves time. This makes the program about twice as fast in this part of the code.



Conclusion

The program demonstrates the crucial role of loops together with conditional statements and Boolean variables in creating solutions for actual problems through coding.

The program operates by identifying all prime numbers that exist within the range selected by the user. Two loops operate in the program: The outer loop examines each number while the inner loop verifies the prime status of each number. The program achieves faster execution

because the inner loop stops its checks of a number when it determines that the number is not prime. The `Is_Correct` Boolean variable tracks the prime status of numbers throughout the checking process. The number gets added to the list of prime numbers when the `Is_Correct` variable remains true throughout the process. The program uses conditional statements to execute specific instructions such as checking if the number is below the end value or if it divides evenly. The program would execute its code identically without these conditions which would result in inefficient operation.

The `generate_special_numbers` program underwent testing with various input types including negative numbers and small range values and start numbers that exceeded end numbers. The program delivered accurate results in all testing scenarios which demonstrates its dependable nature. The program employs an efficient approach to ignore odd numbers when searching for even numbers. The program becomes more efficient because it avoids unnecessary checks of numbers that cannot satisfy the rule.

Reference's

- [1] B. Heinold, in *A Practical Introduction to Python Programming* , 2012, p. 1 to 140.
- [2] M. Lutz, “Powerful Object -Oriented programming,” in *Python Programming* , O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472., 2011, p. 3 to 78.
- [3] “Balanced Scorecard Institute,” [Online]. Available: <https://balancedscorecard.org/wp-content/uploads/pdfs/flowchart.pdf>. [Accessed 13 March 2025].
- [4] H.-P. Halvorsen, in *Python Programming* , ISBN:978-82-691106-4-7, 2019, p. 1 to 113.

programming assignment

GRADEMARK REPORT

FINAL GRADE

GENERAL COMMENTS

46/100

PAGE 1

PAGE 2

PAGE 3



Comment 1

is this information relevant to the given assignment ?



Comment 2

Question Number ?



Comment 3

Try to provide answers that are relevant to the given questions,

PAGE 4



Comment 4

??? Half of number ?

PAGE 5

PAGE 6



Comment 5

This is not the correct conditional statement.



Comment 6

Wrong answer

PAGE 7

PAGE 8



Comment 7

??? Please answer to the question

PAGE 9

PAGE 10

PAGE 11



Comment 8

How did you get these results? The output provided does not match the screenshot.

PAGE 12



Comment 9

Wrong answer

PAGE 13

PAGE 14

GRADING FORM: ACCA4020 RUBRIC 2025

ADAM IBRAHIM

46

QUESTION 1

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



While the answers have been provided, they include several pieces of information that are not directly relevant to the questions.

7

QUESTION 2

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



Correct answer

10

QUESTION 3

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



Too many insignificant information are provided. Please provide the answers that are relevant to the question.

13

QUESTION 4

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



Logic is applied partially. Instead of printing all prime numbers, the programs prints the sum of prime numbers. Out is wrong.

14

QUESTION 5 [15]

Describes and explains five test case results [15]



Test case results and the explanation provided are wrong. Mark is awarded for the attempt.

2

QUESTION 6 [10]

Identifies optimizations and explains performance effectiveness [10]



Incorrect answer

0

QUESTION 7 [10]

Reflection on learning with clear examples [10]



No answers provided.

0