ASSIGNMENT-6.1

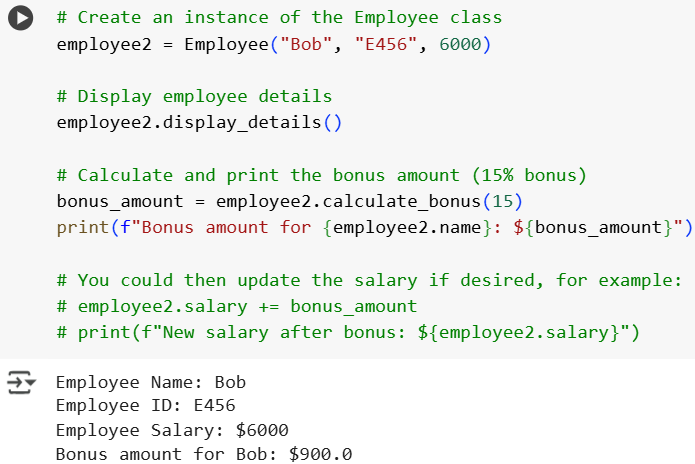
NAME:SRIJA GATTU

ROLL NO:2403A51320

BATCH:13

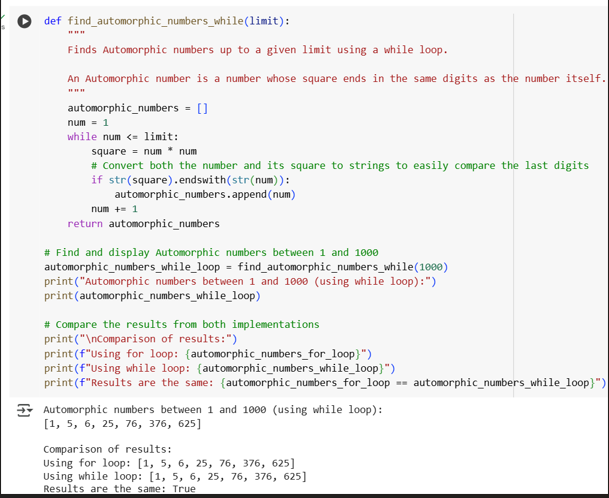
**TASK-1:**

Use AI to create an Employee class with attributes (name,  
id, salary) and a method to calculate yearly salary.  
• Instructions:  
o Prompt AI to generate the Employee class.  
o Analyze the generated code for correctness and structure.  
o Ask AI to add a method to give a bonus and recalculate  
salary.  
 Output:   
• A class with constructor, display\_details(), and calculate\_bonus()  
methods.



**TASK:2**

Prompt AI to generate a function that displays all  
Automorphic numbers between 1 and 1000 using a for loop.  
• Instructions:  
o Get AI-generated code to list Automorphic numbers using  
a for loop.  
o Analyze the correctness and efficiency of the generated  
logic.  
o Ask AI to regenerate using a while loop and compare both  
implementations.  
Output :  
• Correct implementation that lists Automorphic numbers using  
both loop types, with explanation.



**TASK-3**:

Ask AI to write nested if-elif-else conditions to classify  
online shopping feedback as Positive, Neutral, or Negative based  
on a numerical rating (1–5).  
• Instructions:  
o Generate initial code using nested if-elif-else.  
o Analyze correctness and readability.  
o Ask AI to rewrite using dictionary-based or match-case  
structure.  
Expected Output #3:  
• Feedback classification function with explanation and an  
alternative approach.





**Explanation:**

**1. classify\_feedback\_nested\_if (using nested if-elif-else):**

This function uses a series of if, elif (else if), and else statements, with one set of conditionals nested inside another.

* **Outer if 1 <= rating <= 5::** This is the first check. It verifies if the input rating is within the valid range of 1 to 5 (inclusive).
  + If the rating is *not* within this range (e.g., 0 or 6), the condition is false, and the code skips the nested block and goes directly to the final else statement, returning "Invalid Rating".
  + If the rating *is* within the range (1, 2, 3, 4, or 5), the code proceeds to the nested if-elif-else block.
* **Nested if rating >= 4::** This check is performed only if the rating was between 1 and 5. It checks if the rating is 4 or 5.
  + If true, it means the feedback is positive, and the function returns "Positive".
* **Nested elif rating == 3::** This check is performed only if the rating was between 1 and 5 and was *not* 4 or 5. It specifically checks if the rating is exactly 3.
  + If true, it means the feedback is neutral, and the function returns "Neutral".
* **Nested else::** This is the final case within the nested block, reached only if the rating was between 1 and 5 but was neither 4, 5, nor 3. This means the rating must be 1 or 2.
  + In this case, the feedback is considered negative, and the function returns "Negative".

In summary, the nested if-elif-else approach checks the validity of the input range first and then uses further conditions to categorize the valid ratings.

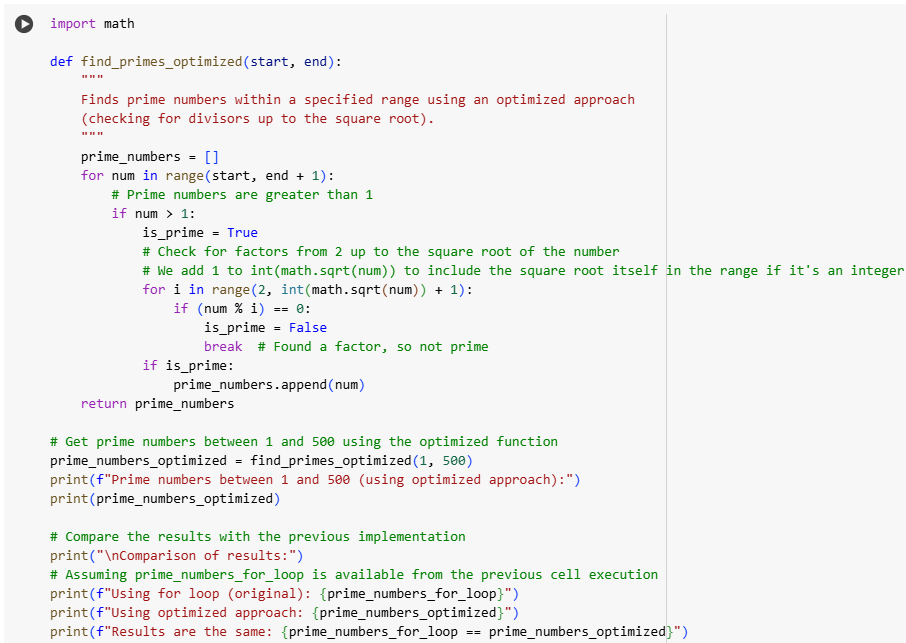
**2. classify\_feedback\_dictionary (using a dictionary):**

This function uses a dictionary to map the numerical ratings directly to their corresponding classification strings.

* **classification\_map = { ... }:** A dictionary named classification\_map is created. The keys of the dictionary are the valid integer ratings (1, 2, 3, 4, 5), and the values are the corresponding classification strings ("Negative", "Neutral", "Positive"). Notice that ratings 4 and 5 both map to "Positive", and ratings 1 and 2 both map to "Negative".
* **return classification\_map.get(rating, "Invalid Rating"):** This is the core of this function.
  + classification\_map.get(rating, ...) is a dictionary method. It tries to find the rating as a key in the classification\_map dictionary.
  + If the rating is found as a key (i.e., if the input rating is 1, 2, 3, 4, or 5), the method returns the value associated with that key (e.g., for a rating of 4, it returns "Positive").
  + If the rating is *not* found as a key in the dictionary (e.g., if the input rating is 0, 6, or any other number not in the dictionary), the get() method returns the second argument provided, which is the default value. In this case, the default value is "Invalid Rating".

**TASK-4:**

Generate a function using AI that displays all prime  
numbers within a user-specified range (e.g., 1 to 500).  
• Instructions:  
o Get AI-generated code to list all primes using a for loop.  
o Analyze the correctness and efficiency of the prime-  
checking logic.  
o Ask AI to regenerate an optimized version (e.g., using the  
square root method).  
Expected Output #4:  
• Python program that lists all prime numbers within a given range,  
with an optimized version and explanation.



**OUTPUT:**

Prime numbers between 1 and 500 (using optimized approach):

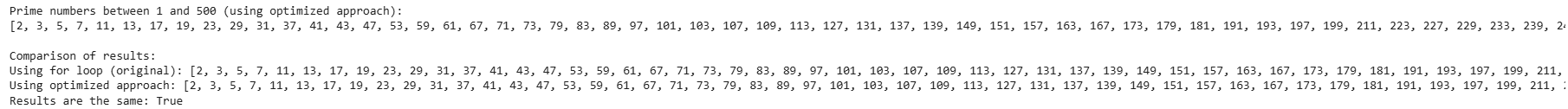
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97, 101, 103, 107, 109, 113, 127, 131, 137, 139, 149, 151, 157, 163, 167, 173, 179, 181, 191, 193, 197, 199, 211, 223, 227, 229, 233, 239, 241, 251, 257, 263, 269, 271, 277, 281, 283, 293, 307, 311, 313, 317, 331, 337, 347, 349, 353, 359, 367, 373, 379, 383, 389, 397, 401, 409, 419, 421, 431, 433, 439, 443, 449, 457, 461, 463, 467, 479, 487, 491, 499]

Comparison of results:

Using for loop (original): [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97, 101, 103, 107, 109, 113, 127, 131, 137, 139, 149, 151, 157, 163, 167, 173, 179, 181, 191, 193, 197, 199, 211, 223, 227, 229, 233, 239, 241, 251, 257, 263, 269, 271, 277, 281, 283, 293, 307, 311, 313, 317, 331, 337, 347, 349, 353, 359, 367, 373, 379, 383, 389, 397, 401, 409, 419, 421, 431, 433, 439, 443, 449, 457, 461, 463, 467, 479, 487, 491, 499]

Using optimized approach: [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97, 101, 103, 107, 109, 113, 127, 131, 137, 139, 149, 151, 157, 163, 167, 173, 179, 181, 191, 193, 197, 199, 211, 223, 227, 229, 233, 239, 241, 251, 257, 263, 269, 271, 277, 281, 283, 293, 307, 311, 313, 317, 331, 337, 347, 349, 353, 359, 367, 373, 379, 383, 389, 397, 401, 409, 419, 421, 431, 433, 439, 443, 449, 457, 461, 463, 467, 479, 487, 491, 499]

Results are the same: True



**EXPLANATION:**

**• The Optimization: A number num is prime if it is not divisible by any number other than 1 and itself. To check for primality, we traditionally check for divisibility by all numbers from 2 up to num - 1. If a number num has a divisor greater than its square root, it must also have a divisor smaller than its square root. Therefore, we only need to check for divisibility by numbers from 2 up to the integer part of the square root of num.**

**• import math: This line imports the math module, which provides mathematical functions like sqrt() for calculating the square root.**

**• int(math.sqrt(num)) + 1: Inside the inner for loop, this calculates the integer part of the square root of the current number (num) and adds 1 to include the square root itself in the range of numbers to check for divisibility.**

**• for i in range(2, int(math.sqrt(num)) + 1):: This loop iterates through potential divisors starting from 2 up to the calculated square root.**

**• if (num % i) == 0:: If num is divisible by any number i in this range, it means num is not prime, so we break out of the inner loop.**

**• else:: The else block associated with the inner for loop is executed only if the loop completes without finding any divisors (i.e., the break statement was not hit). This indicates that the number is prime, and it is appended to the prime\_numbers list.**

**TASK-5:**

• Use AI to build a Library class with methods to  
add\_book(), issue\_book(), and display\_books().  
• Instructions:  
o Generate Library class code using AI.  
o Analyze if methods handle edge cases (e.g., issuing  
unavailable books).  
o Ask AI to add comments and documentation.  
Expected Output #5:  
• Library class with all methods, inline comments, and explanation.

Note: Report should be submitted a word document for all tasks in a  
single document with prompts, comments & code explanation, and  
output and if required, screenshots



**EXPLANATION:**

1. **my\_library = Library()**: This line creates an instance (an object) of the Library class and assigns it to the variable my\_library. This essentially sets up a new, empty library.
2. **my\_library.add\_book(...)**: These lines call the add\_book method of the my\_library object to add several book titles to the library's collection. You can see the output messages confirming that each book has been added. Notice that adding "Pride and Prejudice" twice results in two copies of that title in the library's list.
3. **my\_library.display\_books()**: This line calls the display\_books method. This method checks if there are any books in the library and, if so, prints a list of the available book titles. The output shows all the books that were added.
4. **my\_library.issue\_book("1984")**: This line calls the issue\_book method to try and issue the book "1984". Since "1984" was added to the library, the method finds it, removes it from the library's collection, and prints a message indicating that the book has been issued.
5. **my\_library.issue\_book("Brave New World")**: This line calls the issue\_book method for a book that was *not* added to the library ("Brave New World"). The issue\_book method checks if the book is available and, finding that it is not, prints a message indicating that the book is not available. The library's collection remains unchanged by this call.
6. **my\_library.display\_books()**: This final line calls display\_books again. The output shows the list of books remaining in the library *after* "1984" was issued. You can see that "1984" is no longer in the list, while the other books (including the duplicate "Pride and Prejudice") are still there.