

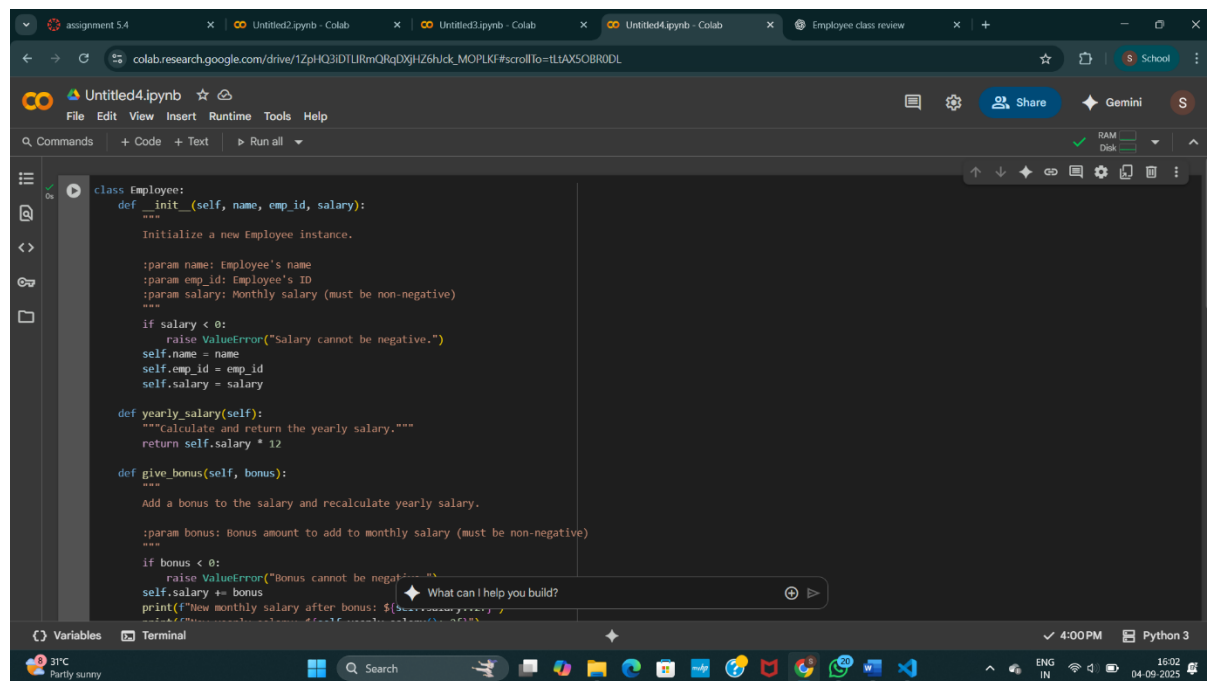
AI_ASSIGNMENT : 6

Batch:12

Hall ticket : 2403A51292

Task Description #1 (Classes – Employee Management)

- Task: Use AI to create an Employee class with attributes (name, id, salary) and a method to calculate yearly salary.
- Instructions:
 - Prompt AI to generate the Employee class.
 - Analyze the generated code for correctness and structure.
 - Ask AI to add a method to give a bonus and recalculate salary.



The screenshot shows a Google Colab notebook titled 'Untitled4.ipynb'. The code defines an 'Employee' class with the following methods:

```
class Employee:
    def __init__(self, name, emp_id, salary):
        """
        Initialize a new Employee instance.

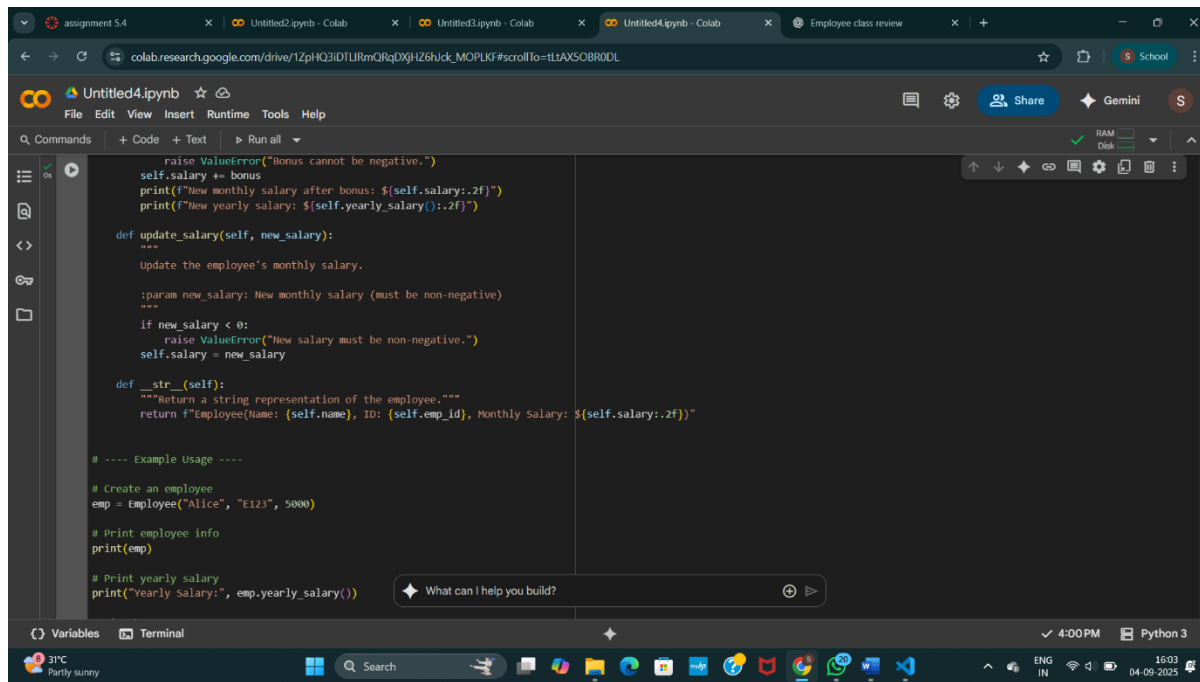
        :param name: Employee's name
        :param emp_id: Employee's ID
        :param salary: Monthly salary (must be non-negative)
        """
        if salary < 0:
            raise ValueError("Salary cannot be negative.")
        self.name = name
        self.emp_id = emp_id
        self.salary = salary

    def yearly_salary(self):
        """Calculate and return the yearly salary."""
        return self.salary * 12

    def give_bonus(self, bonus):
        """
        Add a bonus to the salary and recalculate yearly salary.

        :param bonus: Bonus amount to add to monthly salary (must be non-negative)
        """
        if bonus < 0:
            raise ValueError("Bonus cannot be negative.")
        self.salary += bonus
        print(f"New monthly salary after bonus: ${self.salary}")
        print(f"New yearly salary after bonus: ${self.yearly_salary()}")
```

The notebook interface includes a menu bar (File, Edit, View, Insert, Runtime, Tools, Help), a toolbar with icons for running, saving, and sharing, and a bottom status bar showing 'Variables', 'Terminal', '4:00 PM', and 'Python 3'. The system tray at the very bottom shows weather (31°C Partly sunny), search, and system icons.



```
raise ValueError("Bonus cannot be negative.")
self.salary += bonus
print(f"New monthly salary after bonus: ${self.salary:.2f}")
print(f"New yearly salary: ${self.yearly_salary():.2f}")

def update_salary(self, new_salary):
    """
    Update the employee's monthly salary.

    :param new_salary: New monthly salary (must be non-negative)
    """
    if new_salary < 0:
        raise ValueError("New salary must be non-negative.")
    self.salary = new_salary

def __str__(self):
    """Return a string representation of the employee."""
    return f"Employee(Name: {self.name}, ID: {self.emp_id}, Monthly Salary: ${self.salary:.2f})"

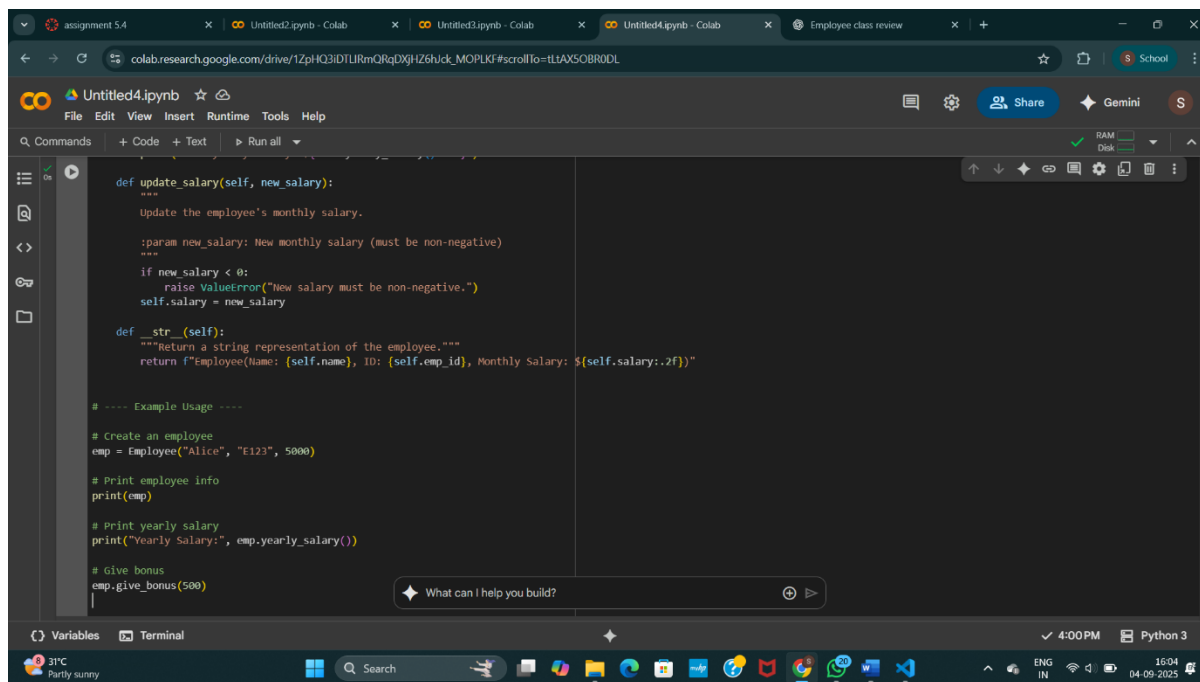
# ---- Example Usage ----

# Create an employee
emp = Employee("Alice", "E123", 5000)

# Print employee info
print(emp)

# Print yearly salary
print("Yearly salary:", emp.yearly_salary())
```

What can I help you build?



```
def update_salary(self, new_salary):
    """
    Update the employee's monthly salary.

    :param new_salary: New monthly salary (must be non-negative)
    """
    if new_salary < 0:
        raise ValueError("New salary must be non-negative.")
    self.salary = new_salary

def __str__(self):
    """Return a string representation of the employee."""
    return f"Employee(Name: {self.name}, ID: {self.emp_id}, Monthly Salary: ${self.salary:.2f})"

# ---- Example Usage ----

# Create an employee
emp = Employee("Alice", "E123", 5000)

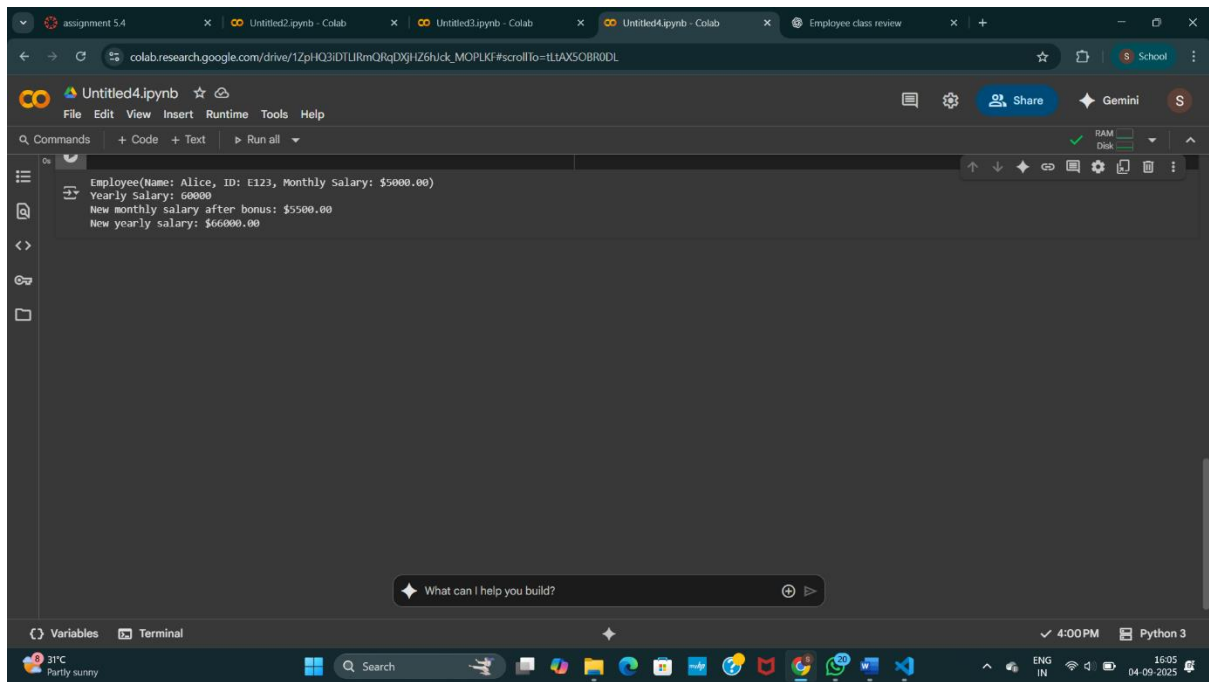
# Print employee info
print(emp)

# Print yearly salary
print("Yearly salary:", emp.yearly_salary())

# Give bonus
emp.give_bonus(500)
```

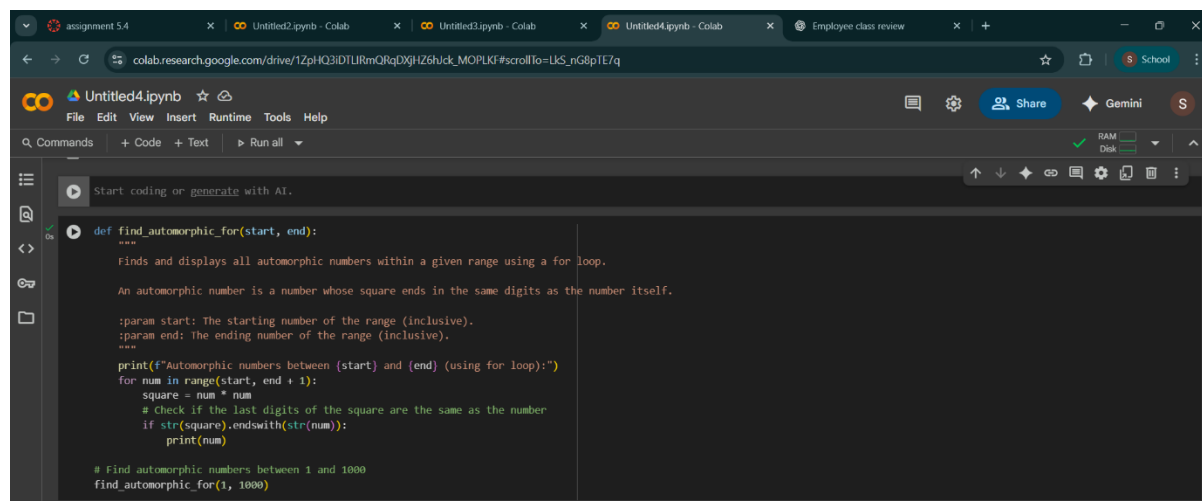
What can I help you build?

OUTPUT:



Task Description #2 (Loops – Automorphic Numbers in a Range)

- Task: Prompt AI to generate a function that displays all Automorphic numbers between 1 and 1000 using a for loop.
- Instructions:
 - Get AI-generated code to list Automorphic numbers using a for loop.
 - Analyze the correctness and efficiency of the generated logic.
 - Ask AI to regenerate using a while loop and compare both implementations.



OUTPUT:

The screenshot shows a Google Colab notebook titled 'Untitled4.ipynb'. The code cell contains the following text:

```
Automorphic numbers between 1 and 1000 (using for loop):  
1  
5  
6  
25  
76  
376  
625
```

The notebook interface includes a menu bar (File, Edit, View, Insert, Runtime, Tools, Help), a toolbar with 'Run all' and 'Share' buttons, and a status bar at the bottom showing 'Python 3' and system information.

Task Description #3 (Conditional Statements – Online Shopping Feedback Classification)

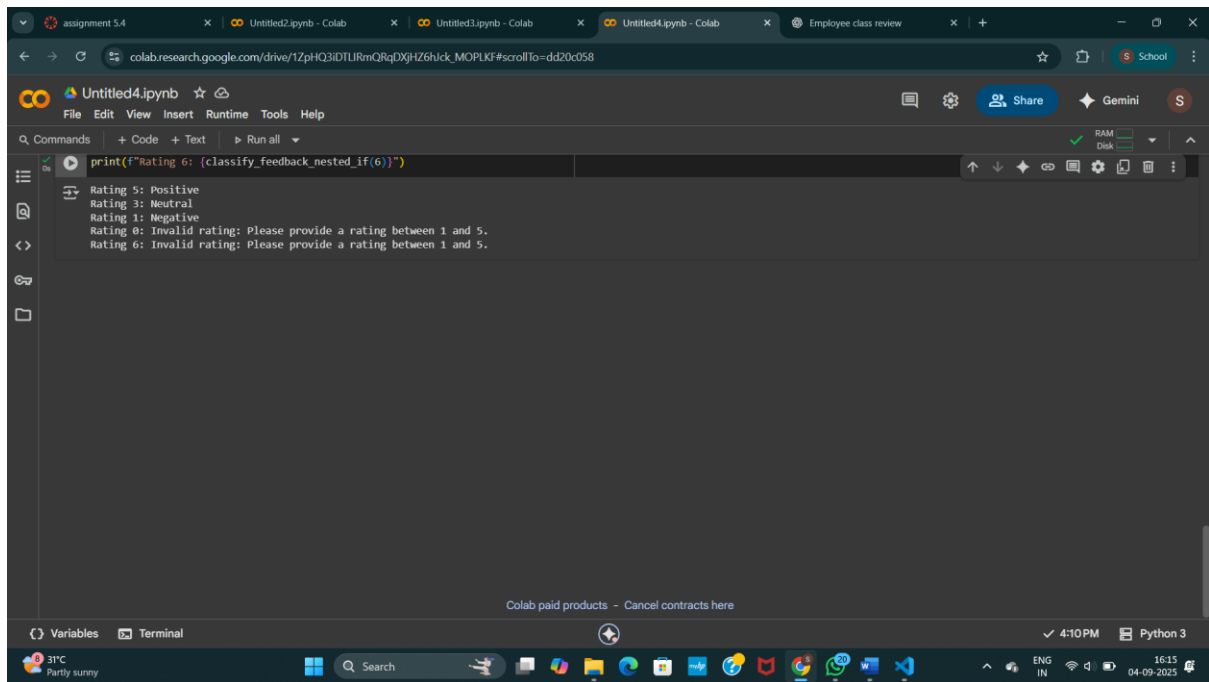
- Task: Ask AI to write nested if-elif-else conditions to classify online shopping feedback as Positive, Neutral, or Negative based on a numerical rating (1–5).
- Instructions:
 - Generate initial code using nested if-elif-else.
 - Analyze correctness and readability.
 - Ask AI to rewrite using dictionary-based or match-case structure.

The screenshot shows a Google Colab notebook titled 'Untitled4.ipynb'. The code cell contains the following Python code:

```
def classify_feedback_nested_if(rating):  
    """  
    Classifies online shopping feedback as Positive, Neutral, or Negative  
    using nested if-elif-else conditions based on a numerical rating (1-5).  
    :param rating: An integer representing the feedback rating (1 to 5).  
    :return: A string indicating the feedback classification (Positive, Neutral, or Negative).  
    """  
    if 1 <= rating <= 5:  
        if rating >= 4:  
            return "Positive"  
        elif rating == 3:  
            return "Neutral"  
        else:  
            return "Negative"  
    else:  
        return "Invalid rating: Please provide a rating between 1 and 5."  
  
# Example Usage:  
print(f"Rating 5: {classify_feedback_nested_if(5)}")  
print(f"Rating 3: {classify_feedback_nested_if(3)}")  
print(f"Rating 1: {classify_feedback_nested_if(1)}")  
print(f"Rating 0: {classify_feedback_nested_if(0)}")  
print(f"Rating 6: {classify_feedback_nested_if(6)}")
```

The notebook interface includes a menu bar (File, Edit, View, Insert, Runtime, Tools, Help), a toolbar with 'Run all' and 'Share' buttons, and a status bar at the bottom showing 'Python 3' and system information.

OUTPUT:



The screenshot shows a Google Colab notebook interface. The top bar includes tabs for 'assignment 5.4', 'Untitled2.ipynb - Colab', 'Untitled3.ipynb - Colab', 'Untitled4.ipynb - Colab', and 'Employee class review'. The address bar shows the URL 'colab.research.google.com/drive/1ZpHQ3iDTURmQRqDXgHZ6hJck_MOPLKF#scrollTo=dd20c058'. The notebook is titled 'Untitled4.ipynb' and has a menu bar with 'File', 'Edit', 'View', 'Insert', 'Runtime', 'Tools', and 'Help'. The code editor shows a single cell with the following Python code:

```
print(f"Rating 6: {classify_feedback_nested_if(6)}")
```

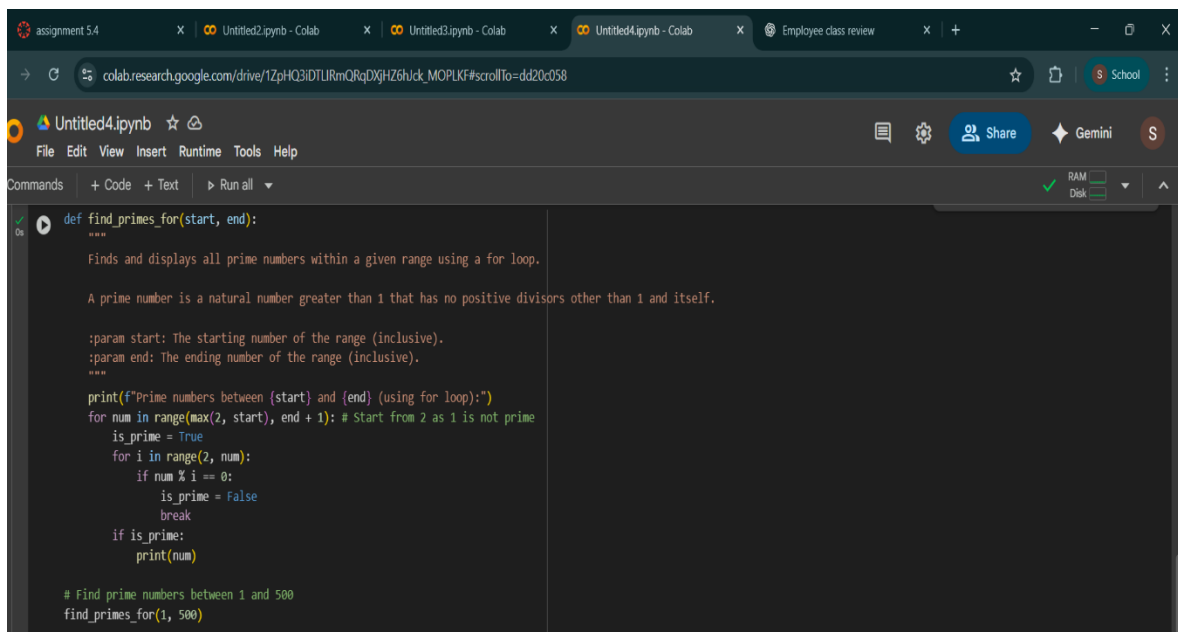
 The output of the cell is displayed below the code:

```
Rating 5: Positive
Rating 3: Neutral
Rating 1: Negative
Rating 0: Invalid rating: Please provide a rating between 1 and 5.
Rating 6: Invalid rating: Please provide a rating between 1 and 5.
```

 The bottom status bar shows 'Colab paid products - Cancel contracts here', 'Variables', 'Terminal', '4:10 PM', and 'Python 3'.

Task Description #4 (Loops – Prime Numbers in a Range)

- Task: Generate a function using AI that displays all prime numbers within a user-specified range (e.g., 1 to 500).
- Instructions:
 - Get AI-generated code to list all primes using a for loop.
 - Analyze the correctness and efficiency of the prime-checking logic.
 - Ask AI to regenerate an optimized version (e.g., using the square root method).



The screenshot shows a Google Colab notebook interface. The top bar includes tabs for 'assignment 5.4', 'Untitled2.ipynb - Colab', 'Untitled3.ipynb - Colab', 'Untitled4.ipynb - Colab', and 'Employee class review'. The address bar shows the URL 'colab.research.google.com/drive/1ZpHQ3iDTURmQRqDXgHZ6hJck_MOPLKF#scrollTo=dd20c058'. The notebook is titled 'Untitled4.ipynb' and has a menu bar with 'File', 'Edit', 'View', 'Insert', 'Runtime', 'Tools', and 'Help'. The code editor shows a single cell with the following Python code:

```
def find_primes_for(start, end):
    """
    Finds and displays all prime numbers within a given range using a for loop.

    A prime number is a natural number greater than 1 that has no positive divisors other than 1 and itself.

    :param start: The starting number of the range (inclusive).
    :param end: The ending number of the range (inclusive).
    """
    print(f"Prime numbers between {start} and {end} (using for loop):")
    for num in range(max(2, start), end + 1): # Start from 2 as 1 is not prime
        is_prime = True
        for i in range(2, num):
            if num % i == 0:
                is_prime = False
                break
        if is_prime:
            print(num)

# Find prime numbers between 1 and 500
find_primes_for(1, 500)
```

 The bottom status bar shows 'Variables', 'Terminal', '4:10 PM', and 'Python 3'.

OUTPUT:

The screenshot shows a Google Colab notebook titled 'Untitled4.ipynb'. The code in the cell is a Python script that prints prime numbers between 1 and 500 using a loop. The output of the code is a list of prime numbers: 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, and 131. The notebook interface includes a menu bar (File, Edit, View, Insert, Runtime, Tools, Help), a toolbar with icons for file operations and execution, and a status bar at the bottom showing the time as 4:16 PM and the Python version as Python 3.

```
Prime numbers between 1 and 500 (using for loop):
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
```

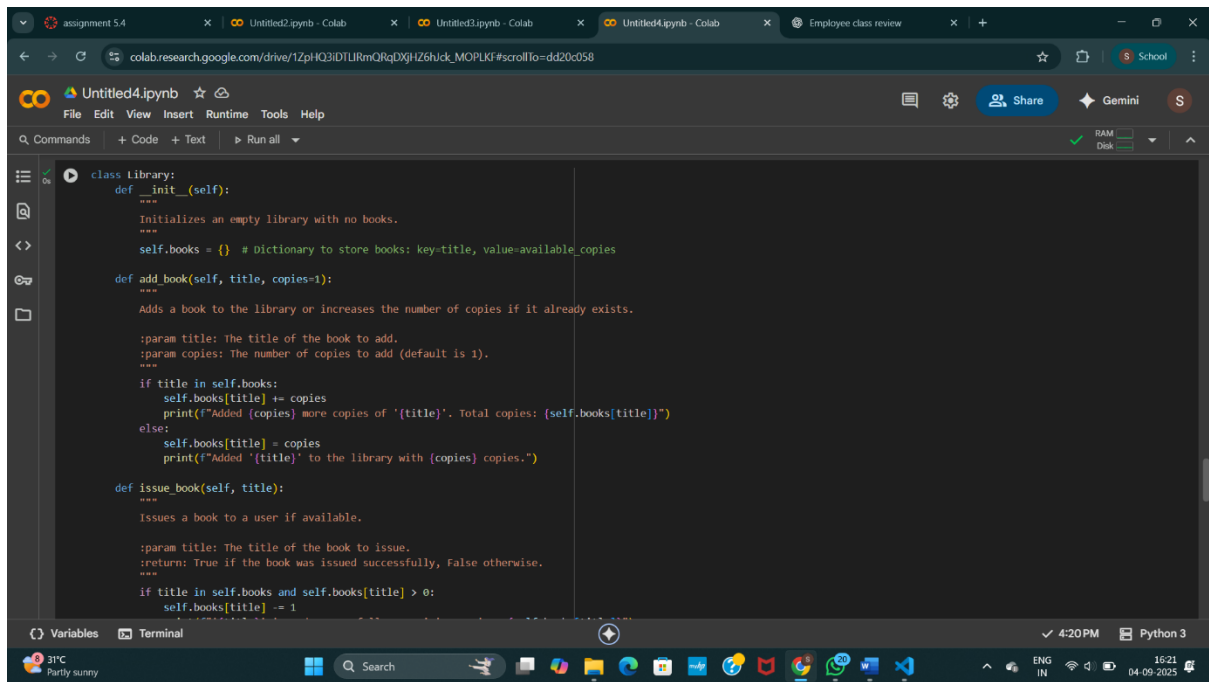
The screenshot shows a Google Colab notebook titled 'Untitled4.ipynb'. The code in the cell is a Python script that prints prime numbers between 1 and 500 using a loop. The output of the code is a list of prime numbers: 281, 283, 293, 307, 311, 313, 317, 331, 337, 347, 349, 353, 359, 367, 373, 379, 383, 389, 397, 401, 409, 419, 421, 431, 433, 439, 443, 449, 457, 461, and ... (indicating more numbers). The notebook interface includes a menu bar (File, Edit, View, Insert, Runtime, Tools, Help), a toolbar with icons for file operations and execution, and a status bar at the bottom showing the time as 4:16 PM and the Python version as Python 3.

```
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
...
```

Task Description #5 (Classes – Library System)

- Task: Use AI to build a Library class with methods to `add_book()`, `issue_book()`, and `display_books()`.
- Instructions:
 - Generate Library class code using AI.
 - Analyze if methods handle edge cases (e.g., issuing unavailable books).

Ask AI to add comments and documentation.



The screenshot shows a Google Colab notebook titled 'Untitled4.ipynb'. The code defines a `Library` class with three methods: `__init__`, `add_book`, and `issue_book`. The `__init__` method initializes an empty dictionary `self.books`. The `add_book` method adds a new book or increases the count of an existing one. The `issue_book` method checks if a book is available and issues it, returning a boolean.

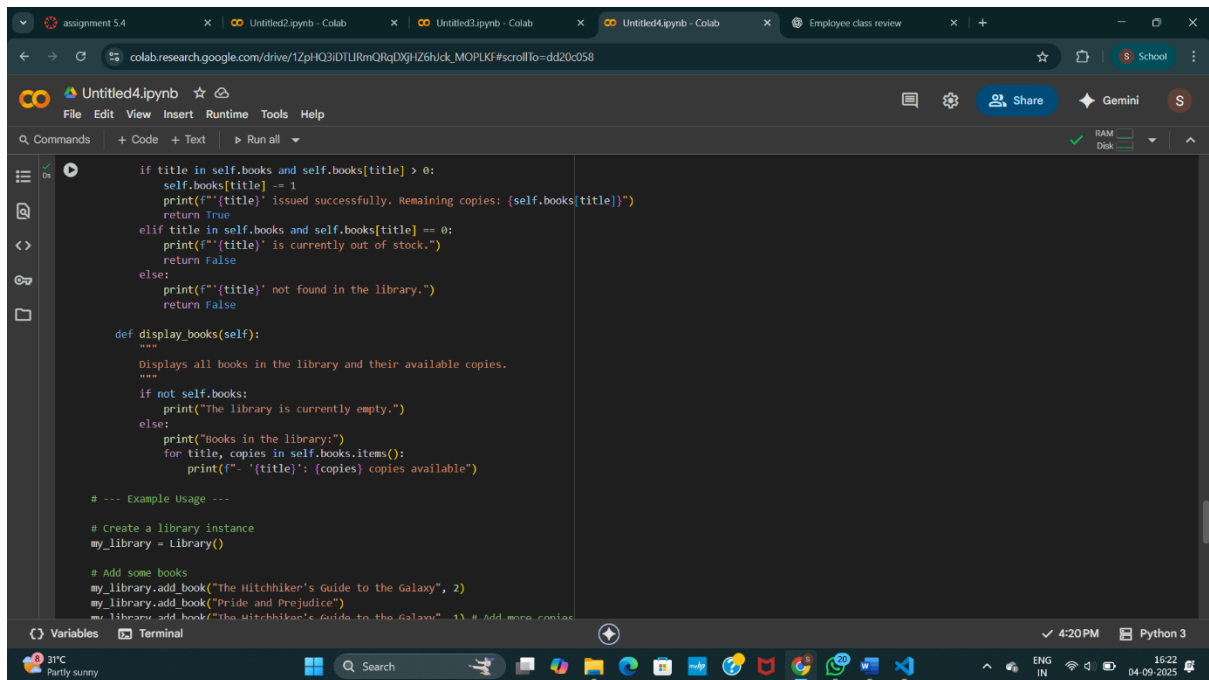
```
class Library:
    def __init__(self):
        """
        Initializes an empty library with no books.
        """
        self.books = {} # Dictionary to store books: key=title, value=available_copies

    def add_book(self, title, copies=1):
        """
        Adds a book to the library or increases the number of copies if it already exists.

        :param title: The title of the book to add.
        :param copies: The number of copies to add (default is 1).
        """
        if title in self.books:
            self.books[title] += copies
            print(f"Added {copies} more copies of '{title}'. Total copies: {self.books[title]}")
        else:
            self.books[title] = copies
            print(f"Added '{title}' to the library with {copies} copies.")

    def issue_book(self, title):
        """
        Issues a book to a user if available.

        :param title: The title of the book to issue.
        :return: True if the book was issued successfully, False otherwise.
        """
        if title in self.books and self.books[title] > 0:
            self.books[title] -= 1
```



The screenshot shows the same Google Colab notebook with the `Library` class code completed. It adds a `display_books` method to show the current state of the library. Below the class definition, there is an example usage section that creates a `my_library` instance, adds three books, and then issues one of them.

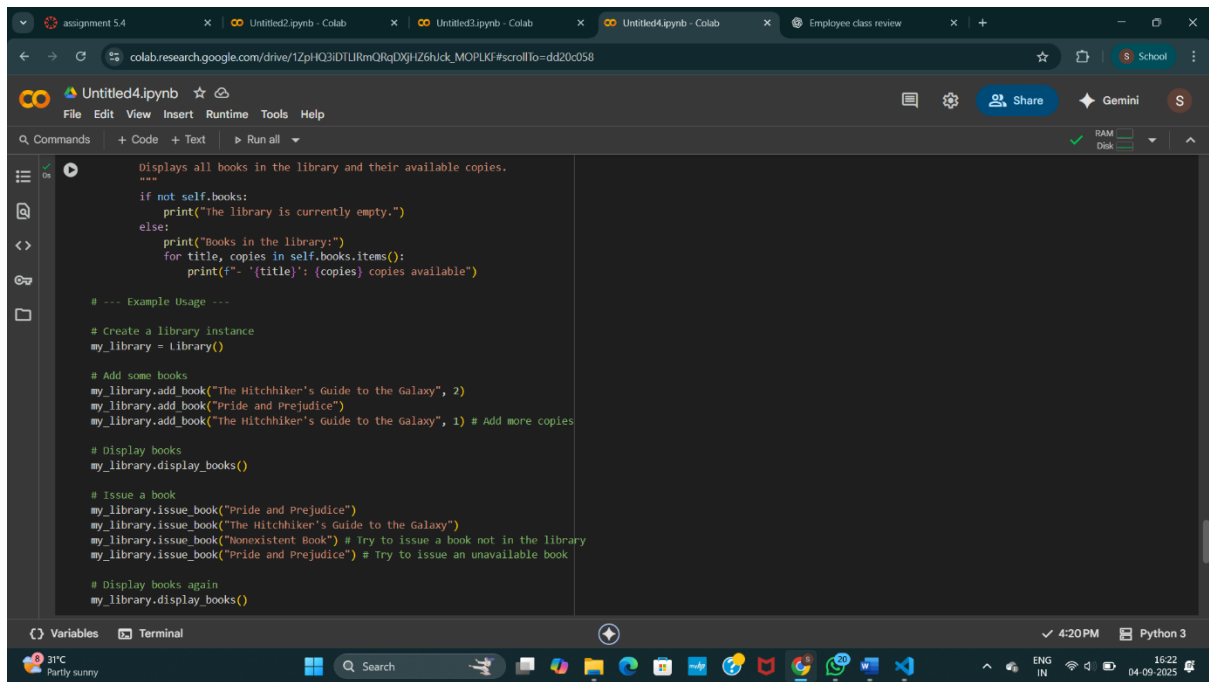
```
        if title in self.books and self.books[title] > 0:
            self.books[title] -= 1
            print(f"'{title}' issued successfully. Remaining copies: {self.books[title]}")
            return True
        elif title in self.books and self.books[title] == 0:
            print(f"'{title}' is currently out of stock.")
            return False
        else:
            print(f"'{title}' not found in the library.")
            return False

    def display_books(self):
        """
        Displays all books in the library and their available copies.
        """
        if not self.books:
            print("The library is currently empty.")
        else:
            print("Books in the library:")
            for title, copies in self.books.items():
                print(f"  '{title}': {copies} copies available")

# --- Example Usage ---

# Create a library instance
my_library = Library()

# Add some books
my_library.add_book("The Hitchhiker's Guide to the Galaxy", 2)
my_library.add_book("Pride and Prejudice")
my_library.add_book("The Hitchhiker's Guide to the Galaxy", 1) # Add more copies
```



The screenshot shows a Google Colab notebook titled 'Untitled4.ipynb'. The code defines a 'Library' class with methods to add books, issue books, and display books. The code is as follows:

```
'''
Displays all books in the library and their available copies.
'''
if not self.books:
    print("The library is currently empty.")
else:
    print("Books in the library:")
    for title, copies in self.books.items():
        print(f"- {title}: {copies} copies available")

# --- Example Usage ---

# Create a library instance
my_library = Library()

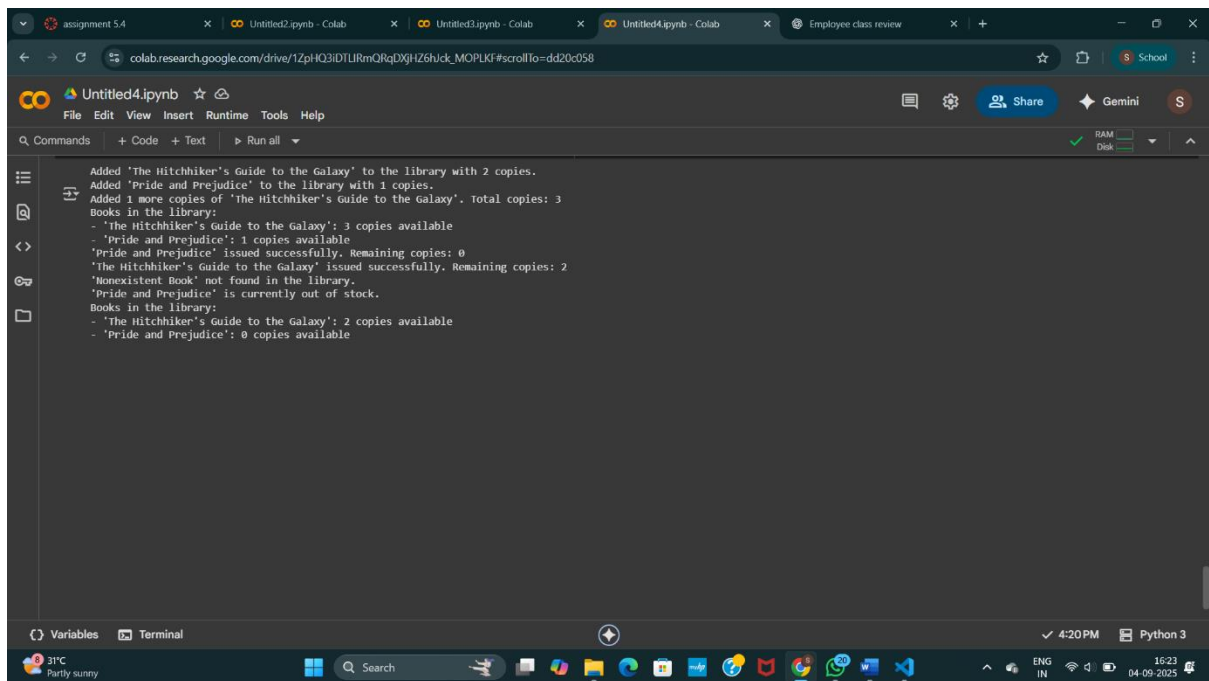
# Add some books
my_library.add_book("The Hitchhiker's Guide to the Galaxy", 2)
my_library.add_book("Pride and Prejudice")
my_library.add_book("The Hitchhiker's Guide to the Galaxy", 1) # Add more copies

# Display books
my_library.display_books()

# Issue a book
my_library.issue_book("Pride and Prejudice")
my_library.issue_book("The Hitchhiker's Guide to the Galaxy")
my_library.issue_book("Nonexistent Book") # Try to issue a book not in the library
my_library.issue_book("Pride and Prejudice") # Try to issue an unavailable book

# Display books again
my_library.display_books()
```

OUTPUT:



The screenshot shows the same Google Colab notebook, but now displaying the output of the code. The output is as follows:

```
Added 'The Hitchhiker's Guide to the Galaxy' to the library with 2 copies.
Added 'Pride and Prejudice' to the library with 1 copies.
Added 1 more copies of 'The Hitchhiker's Guide to the Galaxy'. Total copies: 3
Books in the library:
- 'The Hitchhiker's Guide to the Galaxy': 3 copies available
- 'Pride and Prejudice': 1 copies available
'Pride and Prejudice' issued successfully. Remaining copies: 0
'The Hitchhiker's Guide to the Galaxy' issued successfully. Remaining copies: 2
'Nonexistent Book' not found in the library.
'Pride and Prejudice' is currently out of stock.
Books in the library:
- 'The Hitchhiker's Guide to the Galaxy': 2 copies available
- 'Pride and Prejudice': 0 copies available
```