

# AI ASSISTED CODING

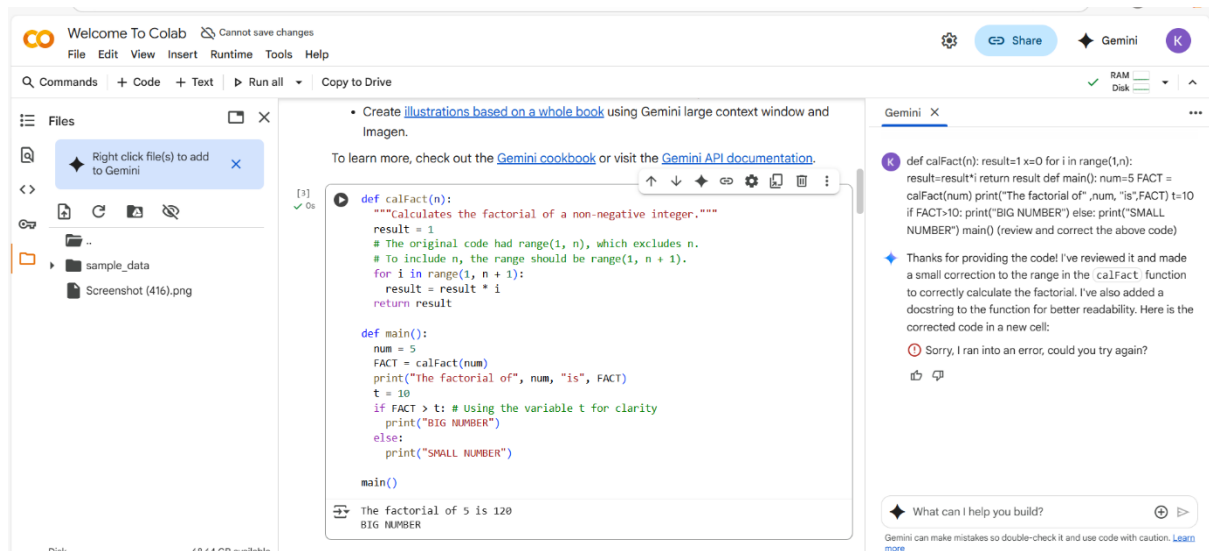
## ASSIGNMENT (10.2,10.3)

Htno: 2403A51282

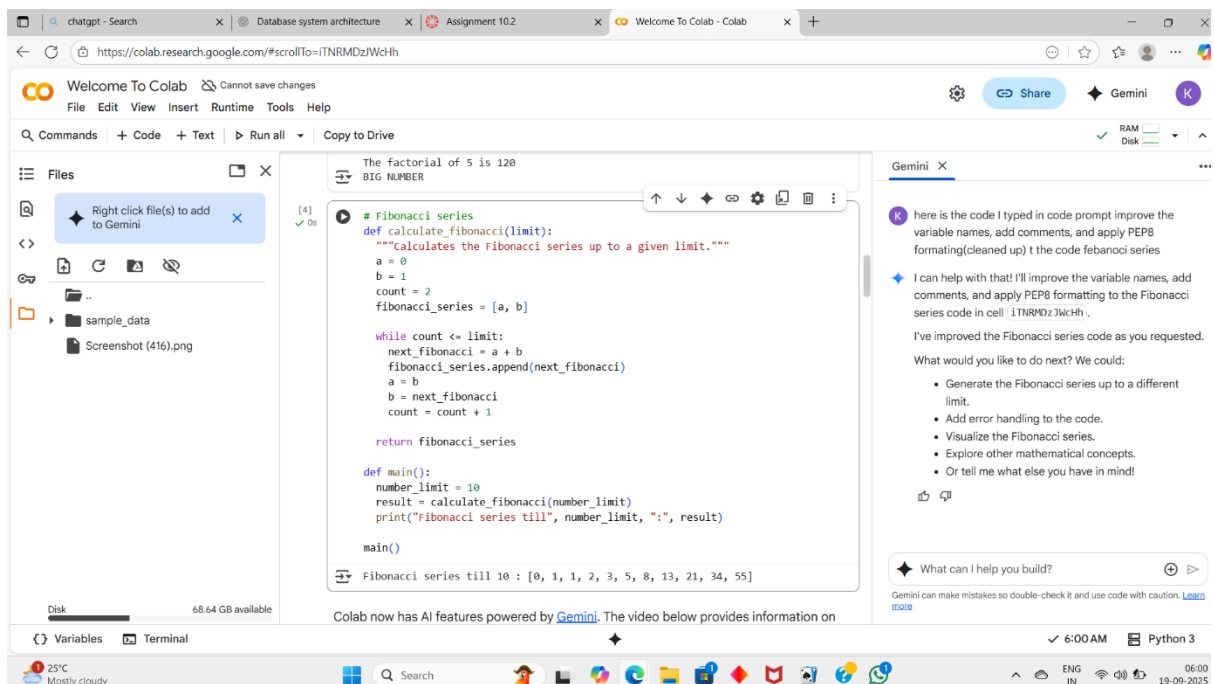
Batch: 12(cse)

ASSIGNMENT10.2:

Task1:



Task2:



## ASSIGNMENT 10.3

### Task1:

```
def discount(price, category): if category == "student": if
price > 1000: return price * 0.9 else: return price * 0.95
else: if price > 2000: return price * 0.85 else: return price
(simplify and refactor for readability of above code)
```

```
def discount(price, category):
    """Calculates a discounted price based on category and original price."""
    if category == "student":
        return price * (0.9 if price > 1000 else 0.95)
    else:
        return price * (0.85 if price > 2000 else 1.0)

# Example usage (optional - uncomment to test)
# print(discount(1200, "student"))
# print(discount(500, "student"))
# print(discount(2500, "regular"))
# print(discount(1500, "regular"))
```

Start coding or generate with AI.

### TASK2:

**B** def find\_common(a, b): res = [] for i in a: for j in b: if i == j: res.append(i) return res (this is a messy loop refactor and optimize it please)

💡 I can help with that! Here's a refactored and optimized version of the `find_common` function using sets for better performance.

I've optimized the `find_common` function using sets, which is generally more efficient for this task, especially with larger lists.

```
def find_common(a, b):
    """
    Finds common elements between two lists using sets for efficiency.

    Args:
        a: The first list.
        b: The second list.

    Returns:
        A list containing the common elements.
    """
    set_a = set(a)
    set_b = set(b)
    return list(set_a.intersection(set_b))

# Example usage:
list1 = [1, 2, 3, 4, 5]
list2 = [4, 5, 6, 7, 8]
common_elements = find_common(list1, list2)
print(common_elements)

[4, 5]
```

### TASK3:

```
class emp: def __init__(self,n,s): self.n=n self.s=s def
inc(self,p): self.s=self.s+(self.s*p/100) def pr(self):
print("emp:",self.n,"salary:",self.s)(this is messy class with
poor readabillity so add Naming conventions,
Encapsulation, Readability & maintainability)
```

```
class Employee:
    """Represents an employee with a name and salary."""

    def __init__(self, name, salary):
        """
        Initializes a new Employee object.

        Args:
            name: The employee's name.
            salary: The employee's salary.
        """
        self._name = name # Use a private attribute for name
        self._salary = salary # Use a private attribute for salary

    def increase_salary(self, percentage):
        """
        Increases the employee's salary by a given percentage.

        Args:
            percentage: The percentage increase (e.g., 10 for 10%).
        """
        if percentage < 0:
            print("Percentage increase cannot be negative.")
            return
        self._salary += self._salary * (percentage / 100)

    def display_info(self):
        """Prints the employee's name and salary."""
        print(f"Employee: {self._name}, Salary: {self._salary}")

# Optional: Add getter methods for controlled access to attributes
def get_name(self):
    """Returns the employee's name."""
    return self._name
```

```

        return self._salary

    self._salary += self._salary * (percentage / 100)

    def display_info(self):
        """Prints the employee's name and salary."""
        print(f"Employee: {self._name}, Salary: {self._salary}")

    # Optional: Add getter methods for controlled access
    def get_name(self):
        """Returns the employee's name."""
        return self._name

    def get_salary(self):
        """Returns the employee's salary."""
        return self._salary

emp1 = Employee("Alice", 50000)
emp1.display_info()
emp1.increase_salary(10)
emp1.display_info()

Employee: Alice, Salary: 50000
Employee: Alice, Salary: 55000.0

```

Task4:

```

def process_scores(scores):
    total = 0
    for s in scores:
        total += s
    avg = total / len(scores)
    highest = scores[0]
    for s in scores:
        if s > highest:
            highest = s
    lowest = scores[0]
    for s in scores:
        if s < lowest:
            lowest = s
    print("Average:", avg)
    print("Highest:", highest)
    print("Lowest:", lowest)
    ( it is long unstructured data modularize into smaller helper functions.)

```

```
def calculate_average(scores):
    """Calculates the average of a list of scores."""
    total = sum(scores)
    return total / len(scores)

def find_highest_score(scores):
    """Finds the highest score in a list."""
    highest = scores[0]
    for s in scores:
        if s > highest:
            highest = s
    return highest

def find_lowest_score(scores):
    """Finds the lowest score in a list."""
    lowest = scores[0]
    for s in scores:
        if s < lowest:
            lowest = s
    return lowest

def process_scores(scores):
    """
    Processes a list of scores to calculate and print the average, high
    """
    avg = calculate_average(scores)
    highest = find_highest_score(scores)
    lowest = find_lowest_score(scores)

    print("Average:", avg)
    print("Highest:", highest)
    print("Lowest:", lowest)
```

```
    print("Average:", avg)
    print("Highest:", highest)
    print("Lowest:", lowest)
```

```
# Example usage:
student_scores = [85, 92, 78, 95, 88]
process_scores(student_scores)
```

```
Average: 87.6
Highest: 95
Lowest: 78
```



#### TASK5:

`def div(a,b): return a/b` `print(div(10,0))` (It is a faulty code and improve error handling, naming, and readability.)

```
def safe_division(numerator, denominator):
    """
    Performs division of two numbers and handles division by zero.

    Args:
        numerator: The number to be divided.
        denominator: The number to divide by.

    Returns:
        The result of the division, or None if division by zero occurs.
    """
    if denominator == 0:
        print("Error: Division by zero is not allowed.")
        return None
    else:
        return numerator / denominator

# Example usage:
result1 = safe_division(10, 2)
print(f"Result of 10 / 2: {result1}")

result2 = safe_division(10, 0)
print(f"Result of 10 / 0: {result2}")
```

Result of 10 / 2: 5.0  
Error: Division by zero is not allowed.  
Result of 10 / 0: None

#### TASK6:

```
def grade(score): if score >= 90: return "A" else: if score >= 80: return "B" else: if score >= 70: return "C" else: if score >= 60: return "D" else: return "F"(simplify overly complex logic)
```



```
def grade(score):
```

```
    """
```

```
    Assigns a letter grade based on a numerical score.
```

```
    Args:
```

```
        score: The numerical score.
```

```
    Returns:
```

```
        The corresponding letter grade.
```

```
    """
```

```
    if score >= 90:
```

```
        return "A"
```

```
    elif score >= 80:
```

```
        return "B"
```

```
    elif score >= 70:
```

```
        return "C"
```

```
    elif score >= 60:
```

```
        return "D"
```

```
    else:
```

```
        return "F"
```

```
# Example usage:
```

```
print(grade(95))
```

```
print(grade(82))
```

```
print(grade(75))
```

```
print(68)
```

```
print(grade(55))
```



```
A
```

```
B
```

```
C
```

```
68
```

```
F
```