Question 1

Question 1: Consider the following code snippet:

```
[1] class Counter:
      count = 0

      def __init__(self):
        self._count = 0

      def increment(self):
        self._count += 1
        Counter.count += 1

      def get_counts(self):
        return f"instance count: {self._count}, class count: {Counter.count}"


    a = Counter()
    b = Counter()

    a.increment()
    a.increment()
    b.increment()

    print(a.get_counts())
    print(b.get_counts())
```

```
instance count: 2, class count: 3
instance count: 1, class count: 3
```

- The difference between Counter.count and self._count is that Counter.count increments as a **global variable,** where self._count is **not a global variable.** The important difference is that Counter.count counts **how many times the class has been used**, where self._count counts **how many times a specific object has been counted.**
- As seen above, the output from a.get_counts() is "instance count: 2, class count: 3", where b.get_counts() is "instance count: 1, class count: 3"
- As mentioned in the first bullet, the increment method interacts with the method through maintaining a global variabl that permeates through uses. Ideally, its output will always be the same after all "increments" have been executed. The instance variables are like local variables, and they are separated from one another. The instance count of "a" and "b" are two totally different pieces of data, thus, printed seperately.

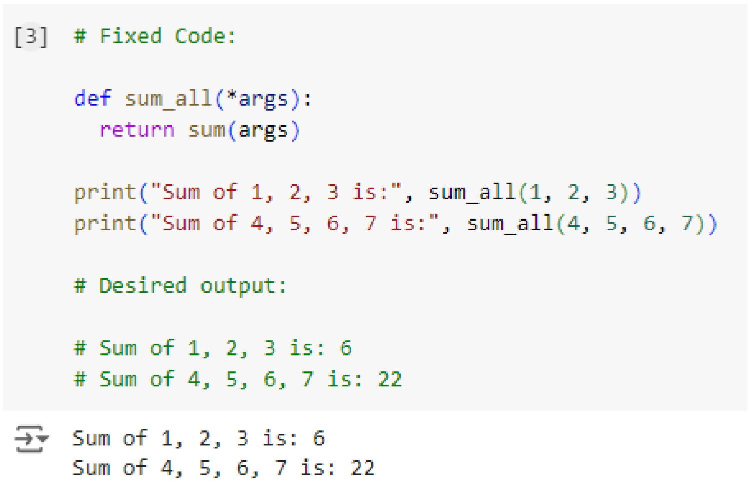
Question 2

Question 2: Find and remove the bug form the code to obtain the given output.

```
[2] # "Broken" Code:

    def sum_all(args):
      return sum(args)

    print("Sum of 1, 2, 3 is:", sum_all(1, 2, 3))
    print("Sum of 4, 5, 6, 7 is:", sum_all(4, 5, 6, 7))

    # Desired output:

    # Sum of 1, 2, 3 is: 6
    # Sum of 4, 5, 6, 7 is: 22
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-2-65ca1fadbe93> in <cell line: 6>()
      4   return sum(args)
      5
----> 6 print("Sum of 1, 2, 3 is:", sum_all(1, 2, 3))
      7 print("Sum of 4, 5, 6, 7 is:", sum_all(4, 5, 6, 7))

TypeError: sum_all() takes 1 positional argument but 3 were given
```

The code that is fixed is turning the input of the “sum_all()” function into a list. Before, it was only taking in a single variable. Now, it can take in a list of variables.

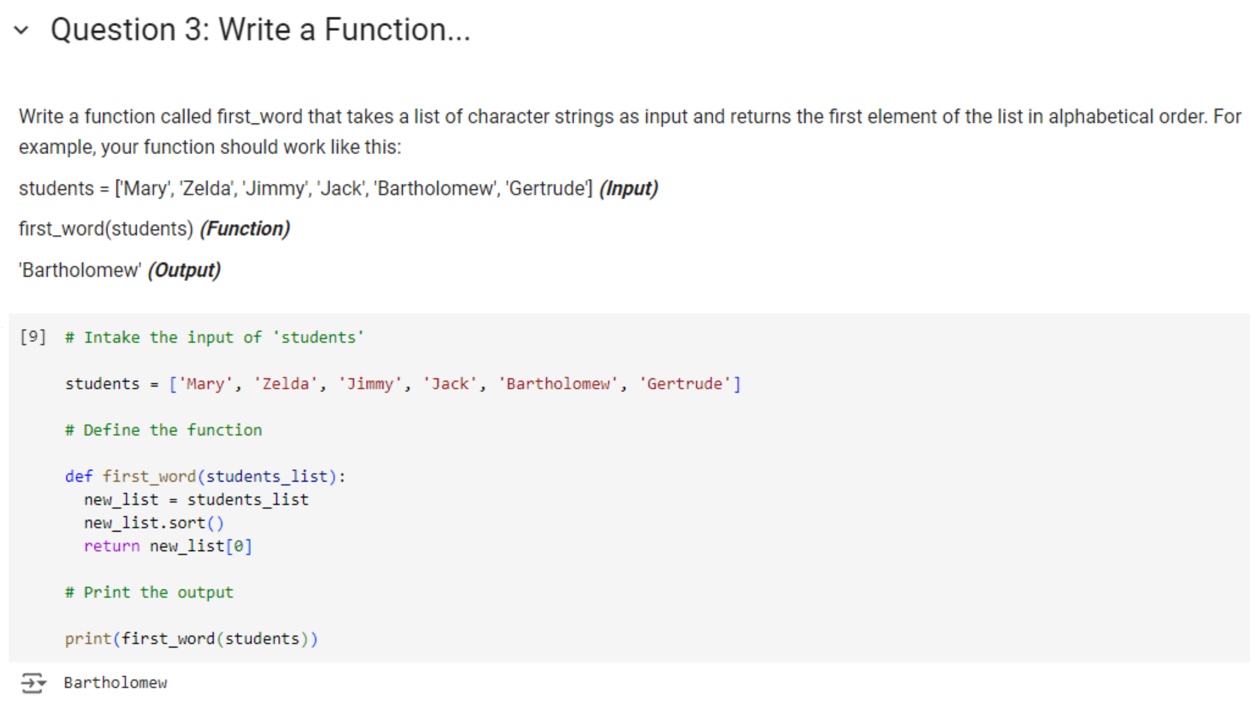


```
[3] # Fixed Code:

    def sum_all(*args):
      return sum(args)

    print("Sum of 1, 2, 3 is:", sum_all(1, 2, 3))
    print("Sum of 4, 5, 6, 7 is:", sum_all(4, 5, 6, 7))

    # Desired output:

    # Sum of 1, 2, 3 is: 6
    # Sum of 4, 5, 6, 7 is: 22
```

```
Sum of 1, 2, 3 is: 6
Sum of 4, 5, 6, 7 is: 22
```

Question 3

To make the function, I simply duplicated the list (to maintain the original list and prevent messing it up, like if it were ordered a certain way for another function), and sorted that duplicate. I simply pass through the 1st instance (index 0) of that new list.



Question 3: Write a Function...

Write a function called first_word that takes a list of character strings as input and returns the first element of the list in alphabetical order. For example, your function should work like this:

students = ['Mary', 'Zelda', 'Jimmy', 'Jack', 'Bartholomew', 'Gertrude'] ***(Input)***

first_word(students) ***(Function)***

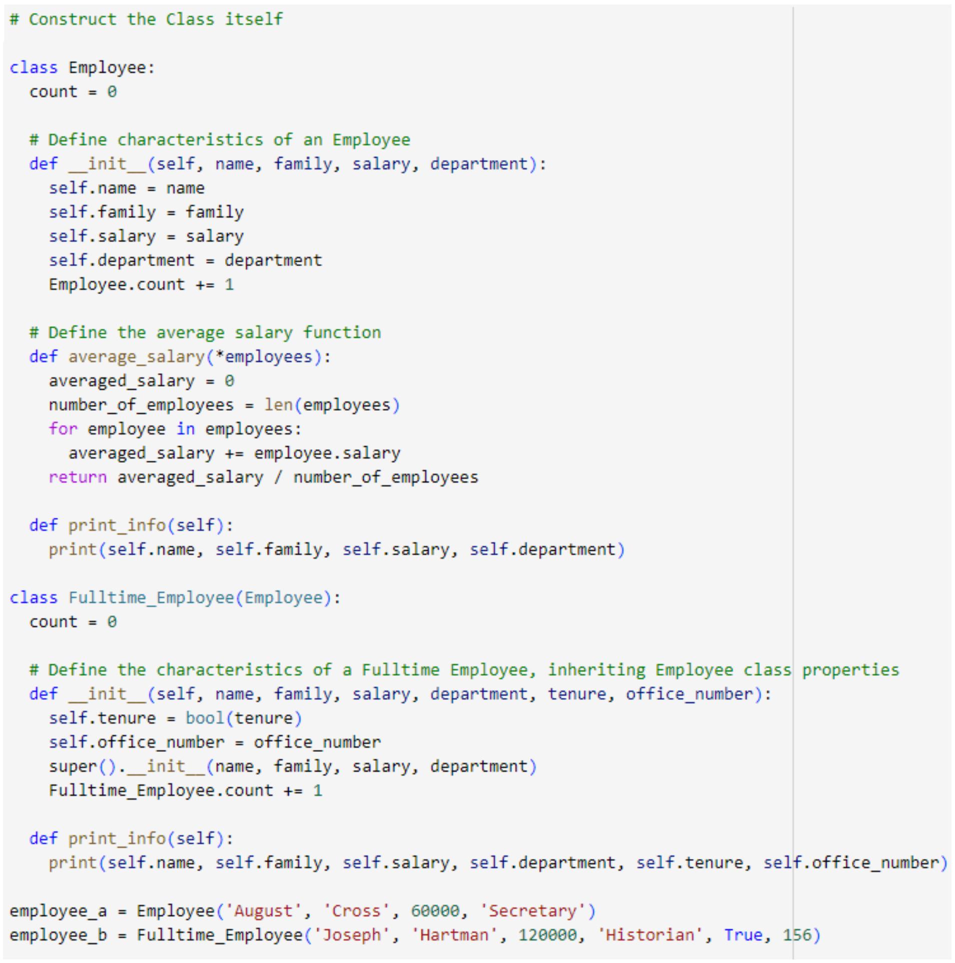
'Bartholomew' ***(Output)***

```
[9] # Intake the input of 'students'

    students = ['Mary', 'Zelda', 'Jimmy', 'Jack', 'Bartholomew', 'Gertrude']

    # Define the function

    def first_word(students_list):
      new_list = students_list
      new_list.sort()
      return new_list[0]

    # Print the output

    print(first_word(students))
```

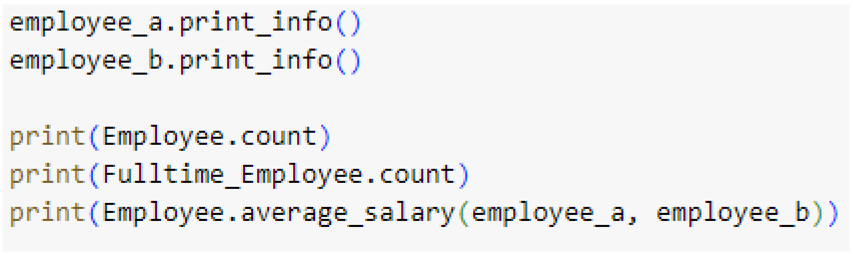
```
Bartholomew
```

Question 4

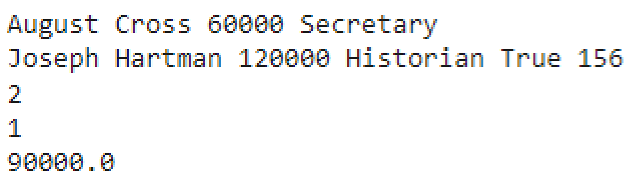
As discussed in class, these employee classes both inherit and create original contents to mock future use and to help differentiate the functions from one another. The main difference is they count separately. Since Fulltime Employees are still Employees (or are a **subset** of the Employee set), Employee class counts *both* regular and Fulltime employees.



```python
# Construct the Class itself

class Employee:
  count = 0

  # Define characteristics of an Employee
  def __init__(self, name, family, salary, department):
    self.name = name
    self.family = family
    self.salary = salary
    self.department = department
    Employee.count += 1

  # Define the average salary function
  def average_salary(*employees):
    averaged_salary = 0
    number_of_employees = len(employees)
    for employee in employees:
      averaged_salary += employee.salary
    return averaged_salary / number_of_employees

  def print_info(self):
    print(self.name, self.family, self.salary, self.department)

class Fulltime_Employee(Employee):
  count = 0

  # Define the characteristics of a Fulltime Employee, inheriting Employee class properties
  def __init__(self, name, family, salary, department, tenure, office_number):
    self.tenure = bool(tenure)
    self.office_number = office_number
    super().__init__(name, family, salary, department)
    Fulltime_Employee.count += 1

  def print_info(self):
    print(self.name, self.family, self.salary, self.department, self.tenure, self.office_number)

employee_a = Employee('August', 'Cross', 60000, 'Secretary')
employee_b = Fulltime_Employee('Joseph', 'Hartman', 120000, 'Historian', True, 156)
```



```python
employee_a.print_info()
employee_b.print_info()

print(Employee.count)
print(Fulltime_Employee.count)
print(Employee.average_salary(employee_a, employee_b))
```



```
August Cross 60000 Secretary
Joseph Hartman 120000 Historian True 156
2
1
90000.0
```