ICP 2_ EXECUTION OUTPUTS

```python
# List of student ages
stu_ages = [19, 22, 19, 24, 20, 25, 26, 24, 25, 24]

# Sort the list
stu_ages.sort()

# Find the min and max age
min_age = stu_ages[0]
max_age = stu_ages[-1]

# Adding minimum and maximum ages to the list
stu_ages.append(min_age)
stu_ages.append(max_age)

# the median age
n = len(stu_ages)
if n % 2 == 0:
    middle1 = stu_ages[n // 2 - 1]
    middle2 = stu_ages[n // 2]
    median_age = (middle1 + middle2) / 2
else:
    median_age = stu_ages[n // 2]

#  the average age
average_age = sum(stu_ages) / len(stu_ages)

#  the range of ages
age_range = max_age - min_age

# Print the results
print("Sorted Ages:", stu_ages)
print("Minimum Age:", min_age)
print("Maximum Age:", max_age)
print("Median of Age:", median_age)
print("Average of Age:", average_age)
print("Rangeof ages:", age_range)
```

```
Sorted Ages: [19, 19, 20, 22, 24, 24, 24, 25, 25, 26, 19, 26]
Minimum Age: 19
Maximum Age: 26
Median of Age: 24.0
Average of Age: 22.75
Rangeof ages: 7
```

```python
[2] # Add key-value pairs to the 'dog' dictionary
    dog['name'] = 'rodo'
    dog['color'] = 'white'
    dog['breed'] = 'pitbull'
    dog['legs'] = 4
    dog['age'] = 2

    # Create a 'student' dictionary with key-value pairs
    student = {
        'first_name': 'vin',
        'last_name': 'alex',
        'gender': 'Male',
        'age': 23,
        'marital_status': 'Single',
        'skills': ['java', 'react'],
        'country': 'india',
        'city': 'hyderabad',
        'address': 'miyapur'
    }

    # Get the length of the 'student' dictionary
    stu_length = len(student)

    # Get the value of 'skills' and check the data type (should be a list)
    skills_value = student['skills']
    skills_data_type = type(skills_value)
```

```python
# Modify the 'skills' values by adding one or two skills
student['skills'].extend(['HTML', 'CSS'])

# Get the dictionary keys as a list
student_keys = list(student.keys())

# Get the dictionary values as a list
student_values = list(student.values())

# Print the results
print("Dog Dictionary:")
print(dog)
print("\nStudent Dictionary Length:", stu_length)
print("Skills Data Type:", skills_data_type)
print("Modified Skills:")
print(student['skills'])
print("\nDictionary Keys:")
print(student_keys)
print("\nDictionary Values:")
print(student_values)
```

```
[2]
Dog Dictionary:
{'name': 'rodo', 'color': 'white', 'breed': 'pitbull', 'legs': 4, 'age': 2}

Student Dictionary Length: 9
Skills Data Type: <class 'list'>
Modified Skills:
['java', 'react', 'HTML', 'CSS']

Dictionary Keys:
['first_name', 'last_name', 'gender', 'age', 'marital_status', 'skills', 'country', 'city', 'address']

Dictionary Values:
['vin', 'alex', 'Male', 23, 'Single', ['java', 'react', 'HTML', 'CSS'], 'india', 'hyderabad', 'miyapur']
```

```python
# Define the sets and list
it_companies = {'Facebook', 'Google', 'Microsoft', 'Apple', 'IBM', 'Oracle', 'Amazon'}
A = {19, 22, 24, 20, 25, 26}
B = {19, 22, 20, 25, 26, 24, 28, 27}
age = [22, 19, 24, 25, 26, 24, 25, 24]

# Find the length of the set it_companies
it_comp_length = len(it_companies)

# Add 'Twitter' to it_companies
it_companies.add('Twitter')

# Insert multiple IT companies at once to the set it_companies
it_companies.update(['LinkedIn', 'Netflix', 'Adobe'])

# Remove one of the companies from the set it_companies
it_companies.remove('IBM')

# Difference between remove and discard:
# - remove() raises an error if the element is not found, - discard() does nothing if the element is not found.You can choose the appropriate method based on your requirements.

# Join A and B
joined_set = A.union(B)

# Find A intersection B
intersection_set = A.intersection(B)

# Checking A is a subset of B
is_A_subset_of_B = A.issubset(B)

# Checking A and B are disjoint sets
are_disjoint = A.isdisjoint(B)

# Joining A with B and B with A
A.update(B)
B.update(A)
```

```python
# the symmetric difference between A and B
symmetric_difference = A.symmetric_difference(B)



# Converting the ages to a set and compare the length of the list and the set
ages_set = set(age)
ages_list_length = len(age)
ages_set_length = len(ages_set)

# Print the results
print("Length of it_companies:", it_comp_length)
print("Updated it_companies:", it_companies)  # This will raise an error because 'it_companies' was deleted
print("Joined Set A and B:", joined_set)
print("Intersection of A and B:", intersection_set)
print("Is A a Subset of B:", is_A_subset_of_B)
print("Are A and B Disjoint Sets:", are_disjoint)
print("Updated Set A:", A)
print("Updated Set B:", B)
print("Symmetric Difference between A and B:", symmetric_difference)
print("Length of Age List:", ages_list_length)
print("Length of Age Set:", ages_set_length)

# Delete the all sets c
del it_companies
del A
del B

print(it_companies)
```

```
Length of it_companies: 7
Updated it_companies: {'Microsoft', 'Netflix', 'Oracle', 'Facebook', 'Google', 'Amazon', 'Apple', 'Twitter', 'LinkedIn', 'Adobe'}
Joined Set A and B: {19, 20, 22, 24, 25, 26, 27, 28}
Intersection of A and B: {19, 20, 22, 24, 25, 26}
Is A a Subset of B: True
Are A and B Disjoint Sets: False
Updated Set A: {19, 20, 22, 24, 25, 26, 27, 28}
Updated Set B: {19, 20, 22, 24, 25, 26, 27, 28}
Symmetric Difference between A and B: set()
Length of Age List: 8
Length of Age Set: 5
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-7-29289ae0ea56> in <cell line: 66>()
     64 del B
     65
---> 66 print(it_companies)

NameError: name 'it_companies' is not defined

SEARCH STACK OVERFLOW
```

```python
[3] class Employee:
        # Class variable to count the number of Employees
        num_employees = 0

        def __init__(self, name, family, salary, department):
            # Initialize instance variables
            self.name = name
            self.family = family
            self.salary = salary
            self.department = department

            # Increment the count of Employees
            Employee.num_employees += 1

        @staticmethod
        def average_salary(employees):
            # Calculate and return the average salary of a list of employees
            total_salary = sum(emp.salary for emp in employees)
            return total_salary / len(employees)


    class FulltimeEmployee(Employee):
        def __init__(self, name, family, salary, department):
            # Call the constructor of the parent class (Employee)
            super().__init__(name, family, salary, department)


    # Create instances of Employee and FulltimeEmployee
    employee1 = Employee("sirisha", "Spouse", 600000, "developer")
    employee2 = Employee("rana prathap", "Child", 550000, "Finance")
    fulltime_employee = FulltimeEmployee(" John", "father", 755000, "Engineering")

    # Call the average_salary function on a list of employees
    employees_list = [employee1, employee2, fulltime_employee]
    average_salary = Employee.average_salary(employees_list)

    # Print the average salary and the number of employees
    print("Average Salary of Employees:", average_salary)
    print("Total Number of Employees:", Employee.num_employees)
```

```
Average Salary of Employees: 635000.0
Total Number of Employees: 3
```

Github Repo Link: https://github.com/sxk7912/Bigdata

Youtube Video Link : https://youtu.be/dxL8qyqhyUE