```
#Name: Tasmim Noshrat Jahan Charu
#ID: 0242220005101844
#Section: 63_M1

# Task 1: Define the list
a = [1, 3, 5, 7, 9]

# 1) Access a[-2] and a[2], find the length and type of 'a'
print("a[-2]:", a[-2])  # Access the second last element
print("a[2]:", a[2])    # Access the third element
print("Length of a:", len(a))  # Length of the list
print("Type of a:", type(a))   # Type of the variable

# 2) Change a[3] = 50, and change a[2] = 49
a[3] = 50
a[2] = 49
print("Updated list a:", a)

# 3) Add 100 in the last index and 200 at index 2
a.append(100)           # Add 100 at the end
a.insert(2, 200)        # Add 200 at index 2
print("After adding 100 and 200:", a)

# 4) Remove the last element and the element at index 1
a.pop()                 # Remove the last element
a.pop(1)                # Remove the element at index 1
print("After removals:", a)

# 5) Join a new list [2, 4, 6] with 'a'
new_list = [2, 4, 6]
a.extend(new_list)      # Extend 'a' with 'new_list'
print("After joining new list:", a)

# 6) Copy all values in a new list 'b'
b = a.copy()
print("New list b:", b)

# 7) Sort the elements of 'b'
b.sort()
print("Sorted list b:", b)

# 8) Print all the elements using a loop and break if it gets 5
print("Elements of b (break at 5):")
for element in b:
```

```
    if element == 5:
        print(element)
        break
    print(element)

# 9) Find the largest number in 'a'
largest_number = max(a)
print("Largest number in a:", largest_number)
```

```
a[-2]: 7
a[2]: 5
Length of a: 5
Type of a: <class 'list'>
Updated list a: [1, 3, 49, 50, 9]
After adding 100 and 200: [1, 3, 200, 49, 50, 9, 100]
After removals: [1, 200, 49, 50, 9]
After joining new list: [1, 200, 49, 50, 9, 2, 4, 6]
New list b: [1, 200, 49, 50, 9, 2, 4, 6]
Sorted list b: [1, 2, 4, 6, 9, 49, 50, 200]
Elements of b (break at 5):
1
2
4
6
9
49
50
200
Largest number in a: 200
```

```
# Task 2: Define the tuple
a = (1, 3, 5, 7, 4)

# a) Find the sum of all odd numbers in 'a'
odd_sum = sum(x for x in a if x % 2 != 0)
print("Sum of all odd numbers:", odd_sum)

# b) Find the 2nd index element in 'a'
second_index_element = a[2]
print("Element at index 2:", second_index_element)

# c) Count the number of odd and even numbers separately
odd_count = sum(1 for x in a if x % 2 != 0)
even_count = sum(1 for x in a if x % 2 == 0)
print("Count of odd numbers:", odd_count)
print("Count of even numbers:", even_count)

# d) Extend the tuple with (2, 4, 6)
extended_tuple = a + (2, 4, 6)
print("Extended tuple:", extended_tuple)

# e) Add a new item (400) in index 2
# Tuples are immutable; to modify, convert to a list and back
```

```
temp_list = list(a)
temp_list.insert(2, 400)
modified_tuple = tuple(temp_list)
print("Tuple after adding 400 at index 2:", modified_tuple)

# f) Remove the last element
temp_list.pop()  # Remove the last element
modified_tuple = tuple(temp_list)
print("Tuple after removing the last element:", modified_tuple)

# g) Perform slicing [-4:-1]
sliced_tuple = a[-4:-1]
print("Sliced tuple [-4:-1]:", sliced_tuple)

# h) Print the tuple using a loop and use continue if it gets 5
print("Printing tuple with continue if value is 5:")
for value in a:
    if value == 5:
        continue
    print(value)
```

```
Sum of all odd numbers: 16
Element at index 2: 5
Count of odd numbers: 4
Count of even numbers: 1
Extended tuple: (1, 3, 5, 7, 4, 2, 4, 6)
Tuple after adding 400 at index 2: (1, 3, 400, 5, 7, 4)
Tuple after removing the last element: (1, 3, 400, 5, 7)
Sliced tuple [-4:-1]: (3, 5, 7)
Printing tuple with continue if value is 5:
1
3
7
4
```

```
# Define the sets
a = {1, 3, 5, 8, 3, 7}
b = {0, False, 1, 5}

# 1. Print a and b
print("Set a:", a)
print("Set b:", b)

# 2. Print length and their type
print("Length of set a:", len(a), "and type:", type(a))
print("Length of set b:", len(b), "and type:", type(b))

# 3. Add a new element 10 to set a
a.add(10)
print("Set a after adding 10:", a)
```

```
# 4. Remove 8 from set a
a.discard(8)
print("Set a after removing 8:", a)

# 5. Perform union, intersection, difference, symmetric difference, and
print("Union of a and b:", a.union(b))
print("Intersection of a and b:", a.intersection(b))
print("Difference of a and b:", a.difference(b))
print("Symmetric difference of a and b:", a.symmetric_difference(b))
print("Is set b a subset of set a?", b.issubset(a))

# 6. Join a new list [2, 3, 4] with set a
new_list = [2, 3, 4]
joined_set = a.union(new_list)
print("Set a after joining with [2, 3, 4]:", joined_set)
```

```
Set a: {1, 3, 5, 7, 8}
Set b: {0, 1, 5}
Length of set a: 5 and type: <class 'set'>
Length of set b: 3 and type: <class 'set'>
Set a after adding 10: {1, 3, 5, 7, 8, 10}
Set a after removing 8: {1, 3, 5, 7, 10}
Union of a and b: {0, 1, 3, 5, 7, 10}
Intersection of a and b: {1, 5}
Difference of a and b: {10, 3, 7}
Symmetric difference of a and b: {0, 3, 7, 10}
Is set b a subset of set a? False
Set a after joining with [2, 3, 4]: {1, 2, 3, 4, 5, 7, 10}
```

```
# Define the dictionary
employee = {
    "name": "Ali",
    "age": 40,
    "type": {"developer": ["iOS", "android"]},
    "permanent": True,
    "salary": 50000,
    100: (1, 2, 3),
    4.5: {5, 6, True, 7, 1}
}

# 1. Print length and type of the employee dictionary
print("Length of dictionary:", len(employee))
print("Type of dictionary:", type(employee))

# 2. Access the key "type" and its nested value "developer"
print("Accessing 'developer' under 'type':", employee["type"]["developer

# 3. Change the value of "permanent" to False
employee["permanent"] = False
```

```
print("Updated dictionary after changing 'permanent':", employee)

# 4. Add a new key "gender" with value "male"
employee["gender"] = "male"
print("Updated dictionary after adding 'gender':", employee)

# 5. Remove the "age" key from the dictionary
employee.pop("age", None)  # Using pop to safely remove the key
print("Updated dictionary after removing 'age':", employee)

# 6. Use keys(), values(), and items() methods
print("Keys of the dictionary:", employee.keys())
print("Values of the dictionary:", employee.values())
print("Items of the dictionary:", employee.items())

# 7. Iterate the dictionary using a loop
print("Iterating over the dictionary:")
for key, value in employee.items():
    print(f"{key}: {value}")
```

```
Length of dictionary: 7
Type of dictionary: <class 'dict'>
Accessing 'developer' under 'type': ['iOS', 'android']
Updated dictionary after changing 'permanent': {'name': 'Ali', 'age': 40, 'type': {'developer': ['iOS', 'android']}, 'permanent': F
Updated dictionary after adding 'gender': {'name': 'Ali', 'age': 40, 'type': {'developer': ['iOS', 'android']}, 'permanent': False,
Updated dictionary after removing 'age': {'name': 'Ali', 'type': {'developer': ['iOS', 'android']}, 'permanent': False, 'salary': 5
Keys of the dictionary: dict_keys(['name', 'type', 'permanent', 'salary', 100, 4.5, 'gender'])
Values of the dictionary: dict_values(['Ali', {'developer': ['iOS', 'android']}, False, 50000, (1, 2, 3), {True, 5, 6, 7}, 'male'])
Items of the dictionary: dict_items([('name', 'Ali'), ('type', {'developer': ['iOS', 'android']}), ('permanent', False), ('salary',
Iterating over the dictionary:
name: Ali
type: {'developer': ['iOS', 'android']}
permanent: False
salary: 50000
100: (1, 2, 3)
4.5: {True, 5, 6, 7}
gender: male
```

```
# Define the strings
a = "hello"
b = "b2b2b2 "
c = " gg3g. "

# 1. Declare a new variable `d` and concatenate `a`, `b`, and `c`
d = a + b + c
print("Concatenated string (d):", d)

# 2. Find the length of `d` and print `d[::-1]`
print("Length of d:", len(d))
print("Reversed string (d[::-1]):", d[::-1])

# 3. Check if "a2" is present in `d`
```

```
print("Is 'a2' present in d?:", "a2" in d)

# 4. Perform the following operations
# a. Convert to uppercase
print("Uppercase:", d.upper())

# b. Convert to lowercase
print("Lowercase:", d.lower())

# c. Convert to title case
print("Title case:", d.title())

# d. Strip leading and trailing spaces
print("Stripped string:", d.strip())

# e. Check if the string is digit
print("Is digit:", d.isdigit())

# f. Find the index of "3g"
print("Index of '3g':", d.find("3g"))

# g. Capitalize the string
print("Capitalized string:", d.capitalize())

# h. Check if the string is alphanumeric
print("Is alphanumeric:", d.isalnum())

# i. Count occurrences of "b2"
print("Count of 'b2':", d.count("b2"))

# j. Split the string into a list
print("Split string:", d.split())

# k. Swap the case
print("Swapcase:", d.swapcase())

# l. Strip leading spaces
print("Leading stripped string:", d.lstrip())

# m. Replace "hello" with "python"
print("Replaced string:", d.replace("hello", "python"))
```

```
Concatenated string (d): hellob2b2b2  gg3g.
Length of d: 19
Reversed string (d[::-1]):  .g3gg  2b2b2bolleh
Is 'a2' present in d?: False
Uppercase: HELLOB2B2B2  GG3G.
```

```
Lowercase: hellob2b2b2  gg3g.
Title case: Hellob2B2B2  Gg3G.
Stripped string: hellob2b2b2  gg3g.
Is digit: False
Index of '3g': 15
Capitalized string: Hellob2b2b2  gg3g.
Is alphanumeric: False
Count of 'b2': 3
Split string: ['hellob2b2b2', 'gg3g.']
Swapcase: HELLOB2B2B2  GG3G.
Leading stripped string: hellob2b2b2  gg3g.
Replaced string: pythonb2b2b2  gg3g.
```

```
import numpy as np

# Original score array
score = np.array([85, 90, 78, 92, 88])

# a) Convert the data type into float
score_float = score.astype(float)

# b) Create a copy of "score" named "a_score" and add 5 points to it
#    Note that the original score will be unchanged
a_score = score.copy()
a_score += 5

# c) Find shape, ndim, size, itemsize, dtype, sort
shape = score.shape
ndim = score.ndim
size = score.size
itemsize = score.itemsize
dtype = score.dtype
sorted_scores = np.sort(score)

# d) Find the index of scores who got 80+
indexes_80_plus = np.where(score > 80)

# e) Find min, max, std, var, sum, mean, axis-wise mean
#    (since it's 1D, axis-wise mean is just the mean)
min_score = np.min(score)
max_score = np.max(score)
std_dev = np.std(score)
variance = np.var(score)
total_sum = np.sum(score)
mean_score = np.mean(score)

# f) Print:
#    - score[:2]
#    - score[-3:-1]
#    - score[:4]
```

```
slice1 = score[:2]
slice2 = score[-3:-1]
slice3 = score[:4]

# Print results
print("Original Scores:", score)
print("Scores as Float:", score_float)
print("Modified Scores (a_score):", a_score)
print("Shape:", shape)
print("Ndim:", ndim)
print("Size:", size)
print("Itemsize:", itemsize)
print("Dtype:", dtype)
print("Sorted Scores:", sorted_scores)
print("Indexes of Scores > 80:", indexes_80_plus[0])
print("Min Score:", min_score)
print("Max Score:", max_score)
print("Standard Deviation:", std_dev)
print("Variance:", variance)
print("Total Sum:", total_sum)
print("Mean Score:", mean_score)
print("Slice [:2]:", slice1)
print("Slice [-3:-1]:", slice2)
print("Slice [:4]:", slice3)
```

```
Original Scores: [85 90 78 92 88]
Scores as Float: [85. 90. 78. 92. 88.]
Modified Scores (a_score): [90 95 83 97 93]
Shape: (5,)
Ndim: 1
Size: 5
Itemsize: 8
Dtype: int64
Sorted Scores: [78 85 88 90 92]
Indexes of Scores > 80: [0 1 3 4]
Min Score: 78
Max Score: 92
Standard Deviation: 4.882622246293481
Variance: 23.84
Total Sum: 433
Mean Score: 86.6
Slice [:2]: [85 90]
Slice [-3:-1]: [78 92]
Slice [:4]: [85 90 78 92]
```

```
#Custom Exception for Age Validation (InvalidVoterException)
# Custom Exception for Invalid Voter
class InvalidVoterException(Exception):
    pass

# Function to check age
def check_voter_age(age):
    if age < 18:
```

```
            raise InvalidVoterException("Age must be 18 or above to vote.")
        else:
            print("You are eligible to vote.")

    # Example usage
    try:
        age = int(input("Enter your age: "))
        check_voter_age(age)
    except InvalidVoterException as e:
        print(e)
```

```
Enter your age: 25
You are eligible to vote.
```

```
    #Custom Exception for Salary Validation (SalaryNotInRange)
    # Custom Exception for Salary Range
    class SalaryNotInRange(Exception):
        pass

    # Employee Class
    class Employee:
        def __init__(self, name, salary):
            self.name = name
            self.salary = salary
            self.validate_salary()

        def validate_salary(self):
            if self.salary < 10000 or self.salary > 50000:
                raise SalaryNotInRange("Salary must be between 10,000 and 50

        def display_salary(self):
            print(f"Employee Name: {self.name}, Salary: {self.salary}")

    # Example usage
    try:
        name = input("Enter employee name: ")
        salary = int(input("Enter employee salary: "))
        emp = Employee(name, salary)
        emp.display_salary()
    except SalaryNotInRange as e:
        print(e)
```

```
Enter employee name: Charu
Enter employee salary: 45000
Employee Name: Charu, Salary: 45000
```

```
#Division Operation with a Custom Array
# Division operation
arr = [10, 5, 15, 20]

try:
    divisor = int(input("Enter a divisor: "))
    for i, value in enumerate(arr):
        print(f"arr[{i}] / {divisor} = {value / divisor}")
except ZeroDivisionError:
    print("Error: Division by zero is not allowed.")
except ValueError:
    print("Error: Please enter a valid integer as the divisor.")
```

```
Enter a divisor: 22
arr[0] / 22 = 0.45454545454545453
arr[1] / 22 = 0.22727272727272727
arr[2] / 22 = 0.6818181818181818
arr[3] / 22 = 0.9090909090909091
```

```
#Handle Various Exceptions with try, except, else, finally
try:
    # Example operations to trigger exceptions
    a = int(input("Enter a number: "))  # Could raise ValueError
    b = int(input("Enter another number: "))  # Could raise ValueError
    result = a / b  # Could raise ZeroDivisionError
    print(f"Result of division: {result}")

    lst = [1, 2, 3]
    print(lst[5])  # Could raise IndexError

    file = open("nonexistent_file.txt", "r")  # Could raise FileNotFoun
except ZeroDivisionError:
    print("Error: Division by zero is not allowed.")
except ValueError:
    print("Error: Invalid input. Please enter a valid integer.")
except NameError:
    print("Error: Variable not defined.")
except TypeError:
    print("Error: Unsupported operation between incompatible types.")
except IndexError:
    print("Error: List index out of range.")
except AttributeError:
    print("Error: Invalid attribute access.")
except FileNotFoundError:
    print("Error: File not found.")
else:
    print("Operations completed successfully without any exceptions.")
```

```
finally:
    print("Execution of the try block is complete.")
```

```
Enter a number: 500
Enter another number: 499
Result of division: 1.002004008016032
Error: List index out of range.
Execution of the try block is complete.
```

```
#Custom Exception for Insufficient Balance (InsufficientBalance)
# Custom Exception for Insufficient Balance
class InsufficientBalance(Exception):
    pass

# BankAccount Class
class BankAccount:
    def __init__(self, balance):
        self.balance = balance

    def withdraw(self, amount):
        try:
            if amount > self.balance:
                raise InsufficientBalance("Withdrawal amount exceeds ava
            self.balance -= amount
            print(f"Withdrawal successful. Remaining balance: {self.bala
        except InsufficientBalance as e:
            print(e)

# Example usage
try:
    account = BankAccount(balance=5000)  # Initial balance
    amount = int(input("Enter amount to withdraw: "))
    account.withdraw(amount)
except ValueError:
    print("Error: Please enter a valid integer amount.")
```

```
Enter amount to withdraw: 22500
Withdrawal amount exceeds available balance.
```