Question1. Write a function find_average(student) that takes student tuple as input and print student rollno, name, marks and average marks as output. Test Cases:

1. stud1 = (1, “rex”, 60, 85, 70) find_average(stud1) Modify the above function find_average(student) so that it processes a tuple of tuples.
2. stud2 = (2, “rex”, (80, 75, 90)) find_average(stud2)

In [1]:
```
def find_average(student):
    rollno, name, *marks = student
    if len(marks) == 1 and isinstance(marks[0], tuple):
        marks = marks[0]
    total_marks = sum(marks)
    average_marks = total_marks / len(marks)
    print(f"Roll No: {rollno}")
    print (f"Name: {name}")
    print("Marks: {marks}")
    print("Average Marks: {average_marks: .2f}")
stud1 = (1, "rex", 60, 85, 70)
find_average(stud1)
stud2 = (2, "rex", (80, 75, 90))
find_average(stud2)
```

```
Roll No: 1
Name: rex
Marks: {marks}
Average Marks: {average_marks: .2f}
Roll No: 2
Name: rex
Marks: {marks}
Average Marks: {average_marks: .2f}
```

Question2. Write a weight management program that prompts the user to enter in 7 days of their body weight values as float numbers. Store them in list. Then print first day weight, last day weight, 4th day weight, highest weight, lowest weight and average weight. Finally, print if average weight < lowest weight, then print “Your weight management is excellent”. Otherwise print “Your weight management is not good. Please take care of your diet”.

In [9]:
```
def  weight_management_program():
    weight_values = []
    for day in range(1, 8):
        while True:
            try:
                weight = float(input("Enter weight for Day {day}: "))
                break
            except ValueError:
                print("Invalid input. Please enter a valid weight as a float nu
        weight_values.append(weight)
    print("\nWeight Management Summary:")
    print (f"First day weight: {weight_values[0]}")
    print("Last day weight: {weight_values[-1]}")
    print("4th day weight: {weight_values[3]}")
    print("Highest weight: {max (weight_values)}")
    print("Lowest weight: {min(weight_values)}")
    average_weight=sum(weight_values) / len(weight_values)
    print (f"Average weight: {average_weight:.2f}")
    if average_weight < min(weight_values):
          print("your weight management is excellent.")
    else:
           print("your weight management is not good. please take care of your
if __name__ == "__main__":
     weight_management_program()
```

```
Enter weight for Day {day}: 43
Enter weight for Day {day}: 68
Enter weight for Day {day}: 96
Enter weight for Day {day}: 54
Enter weight for Day {day}: 98
Enter weight for Day {day}: 23
Enter weight for Day {day}: 12

Weight Management Summary:
First day weight: 43.0
Last day weight: {weight_values[-1]}
4th day weight: {weight_values[3]}
Highest weight: {max (weight_values)}
Lowest weight: {min(weight_values)}
Average weight: 56.29
your weight management is not good. please take care of your diet.
```

Question3. Write a function lastN(lst, n) that takes a list of integers and n and returns n largest numbers. How many numbers you want to enter?: 6 Enter a number: 12 Enter a number: 32 Enter a number: 10 Enter a number: 9 Enter a number: 52 Enter a number: 45 How many largest numbers you want to find?: 3 Largest numbers are: 52, 45, 32

In [21]:
```
def lastN(lst, n):
    lst.sort(reverse=True)
    return lst[:n]
def main():
    num_of_inputs = int(input("How many numbers you want to enter?: "))
    numbers = []
    for i in range(num_of_inputs):
        while True:
            try:
                number = int(input("Enter a number: "))
                break
            except ValueError:
                print("Invalid input. Please enter a valid integer number.")
        numbers.append(number)
    n = int(input("How many largest numbers you want to find?: "))
    largest_numbers = lastN(numbers, n)
    print("Largest numbers are:", ", ".join(map(str, largest_numbers)))
if __name__ == "__main__":
    main()
```

```
How many numbers you want to enter?: 3
Enter a number: 1
Enter a number: 7
Enter a number: 8
How many largest numbers you want to find?: 4
Largest numbers are: 8, 7, 1
```

Question4. Given a list of strings, return a list with the strings in sorted order, except group all the strings that begin with 'x' first. Hint: this can be done by making 2 lists and sorting each of them before combining them. Test Cases:

1. Input: ['mix', 'xyz', 'apple', 'xanadu', 'aardvark'] Output: ['xanadu', 'xyz', 'aardvark', 'apple', 'mix']
2. Input: *'ccc','bbb','aaa','xcc','xaa'+ Output: *'xaa','xcc','aaa','bbb','ccc'+
3. Input: *'bbb','ccc','axx','xzz','xaa'+ Output: *'xaa','xzz','axx','bbb','ccc'+

In [46]:
```
def custom_sort(strings_list):
    x_strings = sorted([s for s in strings_list if s.startswith('x')])
    other_strings = sorted([s for s in strings_list if not s.startswith('x')])
    return x_strings + other_strings
def main():
    test_cases = [
        ['mix', 'xyz', 'apple', 'xanadu', 'aardvark'],
        ['ccc', 'bbb', 'aaa', 'xcc', 'xaa'],
        ['bbb', 'ccc', 'axx', 'xzz', 'xaa']
    ]
    for i, test_case in enumerate(test_cases, start=1):
        sorted_strings = custom_sort (test_case)
        print(f"Test Case {i}: Input: {test_case}, Output: {sorted_strings}")
if  __name__ == "__main__":
     main()
```

```
Test Case 1: Input: ['mix', 'xyz', 'apple', 'xanadu', 'aardvark'], Output:
['xanadu', 'xyz', 'aardvark', 'apple', 'mix']
Test Case 2: Input: ['ccc', 'bbb', 'aaa', 'xcc', 'xaa'], Output: ['xaa', 'xc
c', 'aaa', 'bbb', 'ccc']
Test Case 3: Input: ['bbb', 'ccc', 'axx', 'xzz', 'xaa'], Output: ['xaa', 'xz
z', 'axx', 'bbb', 'ccc']
```

In [ ]:
```
Question5. Develop a function sort_last(). Given a list of non-empty tuples, r
sorted in increasing order by the last element in each tuple. Hint: use a cust
function to extract the last element form each tuple.
Test Cases:
1. Input: [(1, 7), (1, 3), (3, 4, 5), (2, 2)]
Output: [(2, 2), (1, 3), (3, 4, 5), (1, 7)]
2. Input: [(1,3),(3,2),(2,1)]
Output: [(2,1),(3,2),(1,3)]
3. Input: [(2,3),(1,2),(3,1)]
Output: [(3,1),(1,2),(2,3)]
```

In [25]:
```
def sort_last(tuples_list):
    return sorted(tuples_list, key=lambda tup: tup[-1])
def main():
    test_cases = [
        [(1, 7), (1, 3), (3, 4, 5), (2, 2)],
        [(1, 3), (3, 2), (2, 1)],
        [(2, 3), (1, 2), (3, 1)]
    ]
    for i, test_case in enumerate(test_cases, start=1):
        sorted_tuples = sort_last(test_case)
        print(f"Test Case {i}: Input: {test_case}, Output: {sorted_tuples}")
if __name__ == "__main__":
    main()
```

```
Test Case 1: Input: [(1, 7), (1, 3), (3, 4, 5), (2, 2)], Output: [(2, 2), (1,
3), (3, 4, 5), (1, 7)]
Test Case 2: Input: [(1, 3), (3, 2), (2, 1)], Output: [(2, 1), (3, 2), (1,
3)]
Test Case 3: Input: [(2, 3), (1, 2), (3, 1)], Output: [(3, 1), (1, 2), (2,
3)]
```

Question6. Other String Functions a) Define a function first() that receives a tuple and returns its first element b) Define a function sort_first() that receives a list of tuples and returns the sorted c) Print lists in sorted order d) Define a function middle() that receives a a tuple and returns its middle element e) Define a functino sort_middle() that receives a list of tuples and returns it sorted using the key middle f) Print the list [(1,2,3), (2,1,4), (10,7,15), (20,4,50), (30, 6, 40)] in sorted order. Output should be: [(2, 1, 4), (1, 2, 3), (20, 4, 50), (30, 6, 40), (10, 7, 15)]

In [26]:
```
def first(tup):
    return tup[0]
def sort_first(tuples_list):
    return sorted(tuples_list, key=first)
def middle(tup):
    return tup[len(tup) // 2]
def sort_middle(tuples_list):
    return sorted(tuples_list, key=middle)
def main():
    tuples_list = [(1, 2, 3), (2, 1, 4), (10, 7, 15), (28, 4, 50), (38, 6, 40)
    print("Sorted by the first element:")
    print(sort_first(tuples_list))
    print("\nSorted by the middle element:")
    print(sort_middle(tuples_list))
if __name__ == "__main__":
    main()
```

```
Sorted by the first element:
[(1, 2, 3), (2, 1, 4), (10, 7, 15), (28, 4, 50), (38, 6, 40)]

Sorted by the middle element:
[(2, 1, 4), (1, 2, 3), (28, 4, 50), (38, 6, 40), (10, 7, 15)]
```

Question7. Develop a function remove_adjacent(). Given a list of numbers, return a list where all adjacent same elements have been reduced to a single element. You may create a new list or modify the passed in list. Test Cases:

1. Input: [1, 2, 2, 3] and output: [1, 2, 3]
2. Input: [2, 2, 3, 3, 3] and output: [2, 3]
3. Input: [ ]. Output: [ ].
4. Input: [2,5,5,6,6,7] Output: [2,5,6,7]
5. Input: [6,7,7,8,9,9] Output: [6,7,8,9]

In [41]:
```
def remove_adjacent(nums):
    result = []
    for num in nums:
        if not result or num != result[-1]:
            result.append(num)
    return result
def main():
    test_cases = [
        [1, 2, 2, 3],
        [2, 2, 3, 3, 3],
        [],
        [2, 5, 5, 6, 6, 7],
        [6, 7, 7, 8, 9, 9]
    ]
    for i, test_case in enumerate(test_cases, start=1):
            result = remove_adjacent(test_case)
            print(f"Test Case {i}: Input: {test_case}, Output: {result}")
if __name__ == "__main__":
    main()
```

```
Test Case 1: Input: [1, 2, 2, 3], Output: [1, 2, 3]
Test Case 2: Input: [2, 2, 3, 3, 3], Output: [2, 3]
Test Case 3: Input: [], Output: []
Test Case 4: Input: [2, 5, 5, 6, 6, 7], Output: [2, 5, 6, 7]
Test Case 5: Input: [6, 7, 7, 8, 9, 9], Output: [6, 7, 8, 9]
```

Question8. Write a function verbing(). Given a string, if its length is at least 3, add 'ing' to its end. Unless it already ends in 'ing', in which case add 'ly' instead. If the string length is less than 3, leave it unchanged. Return the resulting string. So ‘hail’ yields: hailing; ‘swimming’ yields: swimmingly; ‘do’ yields: do.

In [42]:
```
def verbing(s):
    if len(s) < 3:
        return s
    if s[-3:] == 'ing':
        return s + 'ly'
    else:
        return s + 'ing'
def main():
    test_cases = [
        "hail",
        "swimming",
        "do"
    ]
    for i, test_case in enumerate(test_cases, start=1):
        result = verbing(test_case)
        print (f"Test Case {i}: Input: '{test_case}', Output: '{result}'")
if __name__ == "__main__":
    main()
```

```
Test Case 1: Input: 'hail', Output: 'hailing'
Test Case 2: Input: 'swimming', Output: 'swimmingly'
Test Case 3: Input: 'do', Output: 'do'
```

Question9. Develop a function not_bad(). Given a string, find the first appearance of the substring 'not' and 'bad'. If the 'bad' follows the 'not', replace the whole 'not'...'bad' substring with 'good'. Return the resulting string. So 'This dinner is not that bad!' yields: This dinner is good!

In [45]:
```
def not_bad(s):
    not_index = s.find('not')
    bad_index = s.find('bad')
    if not_index != -1 and bad_index != -1 and bad_index > not_index:
        return s[:not_index] + 'good' + s[bad_index+3:]
    else:
        return s
def main():
    test_cases = [
        "This dinner is not that bad!",
        "This movie is bad, but not that bad!",
        "The weather is great!"
    ]
    for i, test_case in enumerate(test_cases, start=1):
        result = not_bad(test_case)
        print(f"Test Case {i}: Input: '{test_case}, Output: '{result}'")
if __name__ == "__main__":
    main()
```

```
Test Case 1: Input: 'This dinner is not that bad!, Output: 'This dinner is go
od!'
Test Case 2: Input: 'This movie is bad, but not that bad!, Output: 'This movi
e is bad, but not that bad!'
Test Case 3: Input: 'The weather is great!, Output: 'The weather is great!'
```

```
In [ ]:
```