CODE 1

```
# Class to represent an item with value and weight
class Item:
    def __init__(self, value, weight):
        self.value = value
        self.weight = weight

# Function to calculate the maximum value that can be carried
def fractional_knapsack(items, capacity):
    # Sort items by value-to-weight ratio in descending order
    items.sort(key=lambda item: item.value / item.weight, reverse=True)
    total_value = 0.0  # To store the total value

    for item in items:
        if capacity >= item.weight:
            # If the item can fit in the remaining capacity, take it all
            capacity -= item.weight
            total_value += item.value
        else:
            # Otherwise, take the fraction of the item that fits
            fraction = capacity / item.weight
            total_value += item.value * fraction
            break  # The knapsack is full

    return total_value

# Driver code
if __name__ == "__main__":
    # Taking the number of items as input
    n = int(input("Enter the number of items: "))

    # Taking item values and weights as input from the user
    items = []
    for i in range(n):
        value = float(input(f"Enter the value of item {i + 1}: "))
        weight = float(input(f"Enter the weight of item {i + 1}: "))
        items.append(Item(value, weight))

    # Taking the capacity of the knapsack as input
    capacity = float(input("Enter the capacity of the knapsack: "))

    # Calculate and print the maximum value
    max_value = fractional_knapsack(items, capacity)
```

```
    print(f"Maximum value we can obtain = {max_value:.2f}")
```

CODE 2

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

contract StudentData {
    // Structure to represent a Student
    struct Student {
        uint256 id;
        string name;
        uint8 age;
        string course;
    }

    // Array to store the list of students
    Student[] public students;

    // Event to log when a student is added
    event StudentAdded(uint256 id, string name, uint8 age, string course);

    // Function to add a new student
    function addStudent(uint256 _id, string memory _name, uint8 _age, string memory _course)
public {
        // Create a new student and push to the array
        students.push(Student(_id, _name, _age, _course));

        // Emit an event when a student is added
        emit StudentAdded(_id, _name, _age, _course);
    }

    // Fallback function
    fallback() external payable {
        revert("Fallback function called. No direct payments allowed.");
    }

    // Function to get the number of students
    function getStudentCount() public view returns (uint256) {
        return students.length;
    }

    // Function to retrieve a student by index
```

```
    function getStudent(uint256 index) public view returns (uint256, string memory, uint8, string
memory) {
        require(index < students.length, "Index out of bounds");
        Student memory student = students[index];
        return (student.id, student.name, student.age, student.course);
    }
}
```