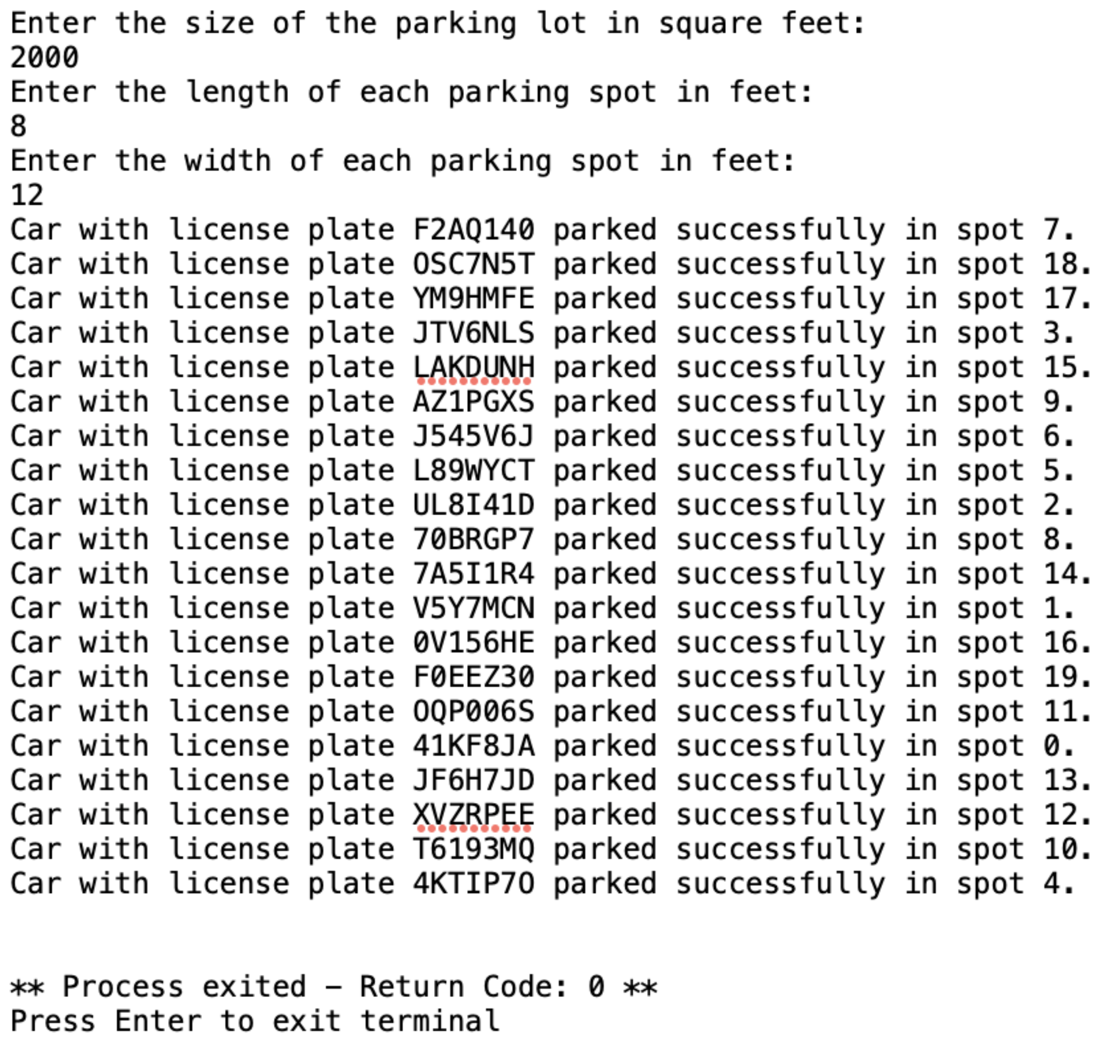
Parking Lot Challenge:
Create a parking lot class that takes in a square footage size as input and creates an array of empty values based on the input square footage size. Assume every parking spot is 8x12 (96 ft2) for this program, but have the algorithm that calculates the array size be able to account for different parking spot sizes. For example, a parking lot of size 2000ft2 can fit 20 cars, but if the parking spots were 10x12 (120 ft2), it could only fit 16 cars. The size of the array will determine how many cars can fit in the parking lot.

Parking_lot.py

```
import random

# This program simulates a parking lot where cars with 7-digit license plates can park in random spots.
# The ParkingLot class initializes a parking lot of a given size with a specified parking spot size.
# The Car class represents a car with a license plate and methods to park the car in the parking lot.
# The main function simulates the process of parking a list of cars in random spots until the parking lot is full.

class ParkingLot:
    def __init__(self, size_in_sqft, spot_length=8, spot_width=12):
        """
        Initialize the parking lot with a given size and spot dimensions.
        Calculates the number of spots based on the size and spot dimensions.
        """
        self.spot_size = spot_length * spot_width
        self.num_spots = size_in_sqft // self.spot_size
        self.spots = [None] * self.num_spots

        if self.spot_size > size_in_sqft:
            raise ValueError("spot_size cannot be more than size_in_sqft.")

    def is_full(self):
        """
        Check if the parking lot is full.
        Returns True if there are no empty spots, False otherwise.
        """
        for spot in self.spots:
            if spot is None:
                return False
        return True

    def find_random_empty_spot(self):
        """
        Find a random empty spot in the parking lot.
        Returns the index of an empty spot, or None if the lot is full.
        """
        empty_spots = [i for i, spot in enumerate(self.spots) if spot is None]
        return random.choice(empty_spots) if empty_spots else None


class Car:
    def __init__(self, license_plate):
        """
        Initialize the car with a given license plate.
        Raises a ValueError if the license plate is not a 7 digit alphanumeric string.
        """
        if len(license_plate) != 7 or not license_plate.isalnum():
            raise ValueError("License plate must be a 7 digit alphanumeric string.")
        self.license_plate = license_plate

    def __str__(self):
        """
        Return the license plate as the string representation of the car.
        """
        return self.license_plate

    def park(self, parking_lot, spot_number):
        """
        Attempt to park the car in the given spot number of the parking lot.
        Returns a tuple (success, message) indicating whether the parking was successful and a message.
        """
        if spot_number < 0 or spot_number >= parking_lot.num_spots:
            return False, f"Spot number {spot_number} is out of range."

        if parking_lot.spots[spot_number] is None:
            parking_lot.spots[spot_number] = self
            return True, f"Car with license plate {self.license_plate} parked successfully in spot {spot_number}."
        else:
            return False, f"Spot {spot_number} is already occupied."


def main(cars, parking_lot):
    """
    Simulate parking each car in the list of cars into random spots in the parking lot.
    Continues until all cars are parked or the parking lot is full.
    At the end, save the mapping of vehicles to spots in a JSON file and upload it to an S3 bucket.
    """
    for car in cars:
        if parking_lot.is_full():
            print("Parking lot is full. Exiting program.")
            break
        while True:
            spot_number = parking_lot.find_random_empty_spot()
            if spot_number is None:
                print("Parking lot is full. Exiting program.")
                break
            success, message = car.park(parking_lot, spot_number)
            print(message)
            if success:
                break


if __name__ == "__main__":
    # Example usage: Create a parking lot and a list of cars, then try to park them.
    size_sft = int(input("Enter the size of the parking lot in square feet: "))
    spot_length = int(input("Enter the length of each parking spot in feet: "))
    spot_width = int(input("Enter the width of each parking spot in feet: "))
    parking_lot = ParkingLot(size_in_sqft=size_sft, spot_length=spot_length,
spot_width=spot_width)

    cars = [Car("ABC 1234"), Car("XYZ567890"), Car("!LMN3456"),
Car("QWE7890"),Car("DBC1234"), Car("WYZ5678"), Car("OMN3456"),
Car("PWE7890"),Car("EBC1234"), Car("QYZ5678"), Car("JMN3456")]
    #cars =
[Car("".join(random.choices("ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789",
k=7))) for _ in range(parking_lot.num_spots)]

    main(cars, parking_lot)
```

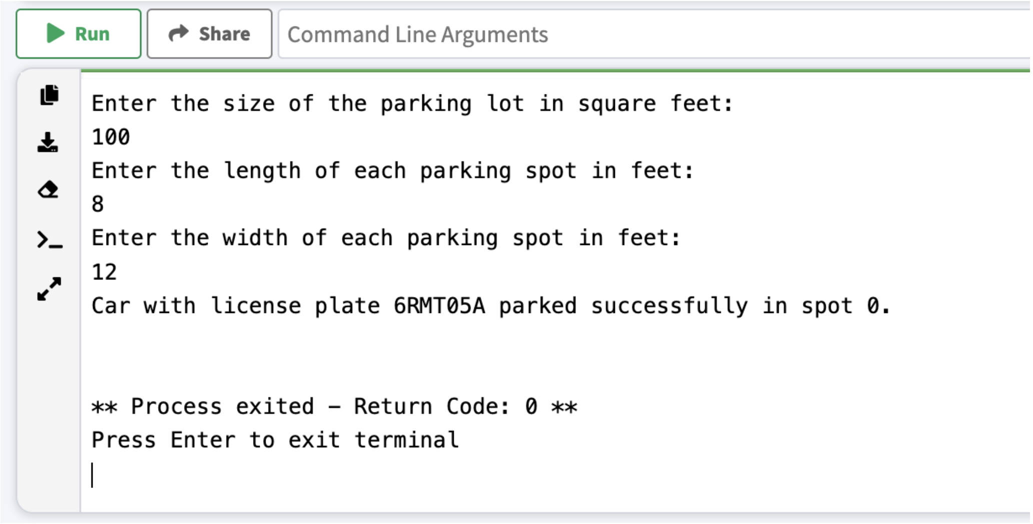
Output:

Case-1: Parked all the cars randomly



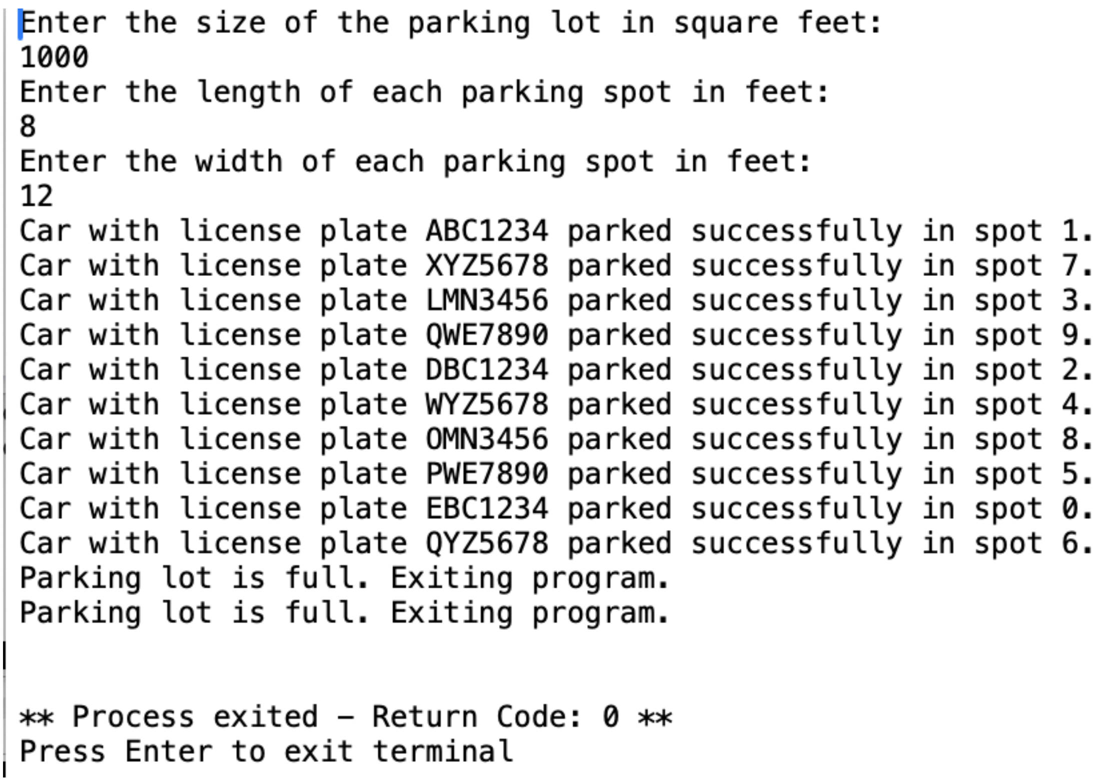
```
Enter the size of the parking lot in square feet:
2000
Enter the length of each parking spot in feet:
8
Enter the width of each parking spot in feet:
12
Car with license plate F2AQ140 parked successfully in spot 7.
Car with license plate OSC7N5T parked successfully in spot 18.
Car with license plate YM9HMFE parked successfully in spot 17.
Car with license plate JTV6NLS parked successfully in spot 3.
Car with license plate LAKDUNH parked successfully in spot 15.
Car with license plate AZ1PGXS parked successfully in spot 9.
Car with license plate J545V6J parked successfully in spot 6.
Car with license plate L89WYCT parked successfully in spot 5.
Car with license plate UL8I41D parked successfully in spot 2.
Car with license plate 70BRGP7 parked successfully in spot 8.
Car with license plate 7A5I1R4 parked successfully in spot 14.
Car with license plate V5Y7MCN parked successfully in spot 1.
Car with license plate 0V156HE parked successfully in spot 16.
Car with license plate F0EEZ30 parked successfully in spot 19.
Car with license plate OQP006S parked successfully in spot 11.
Car with license plate 41KF8JA parked successfully in spot 0.
Car with license plate JF6H7JD parked successfully in spot 13.
Car with license plate XVZRPEE parked successfully in spot 12.
Car with license plate T6193MQ parked successfully in spot 10.
Car with license plate 4KTIP7O parked successfully in spot 4.


** Process exited - Return Code: 0 **
Press Enter to exit terminal
```

Case-2: Limit the square feet area to 100



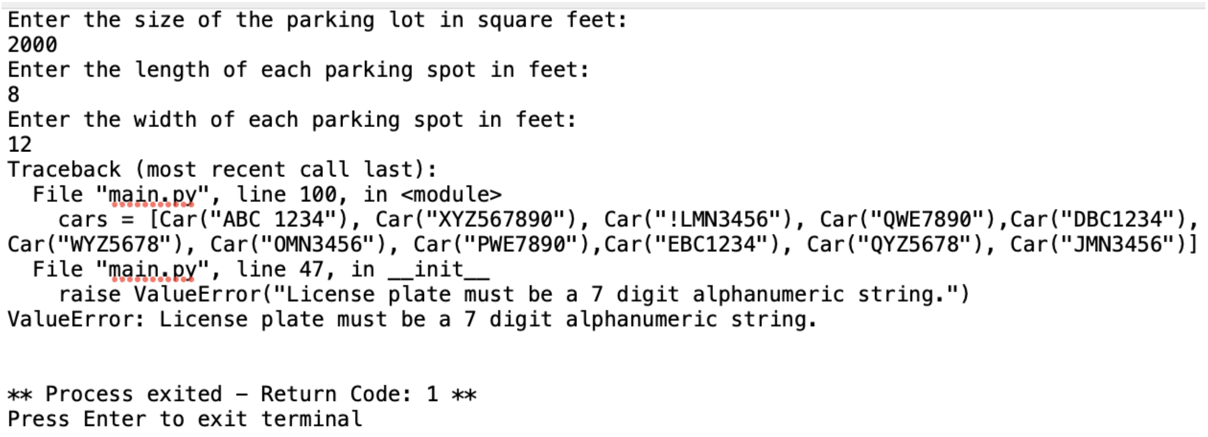
```
Enter the size of the parking lot in square feet:
100
Enter the length of each parking spot in feet:
8
Enter the width of each parking spot in feet:
12
Car with license plate 6RMT05A parked successfully in spot 0.


** Process exited - Return Code: 0 **
Press Enter to exit terminal
```

Case-3: Parking Lot Full



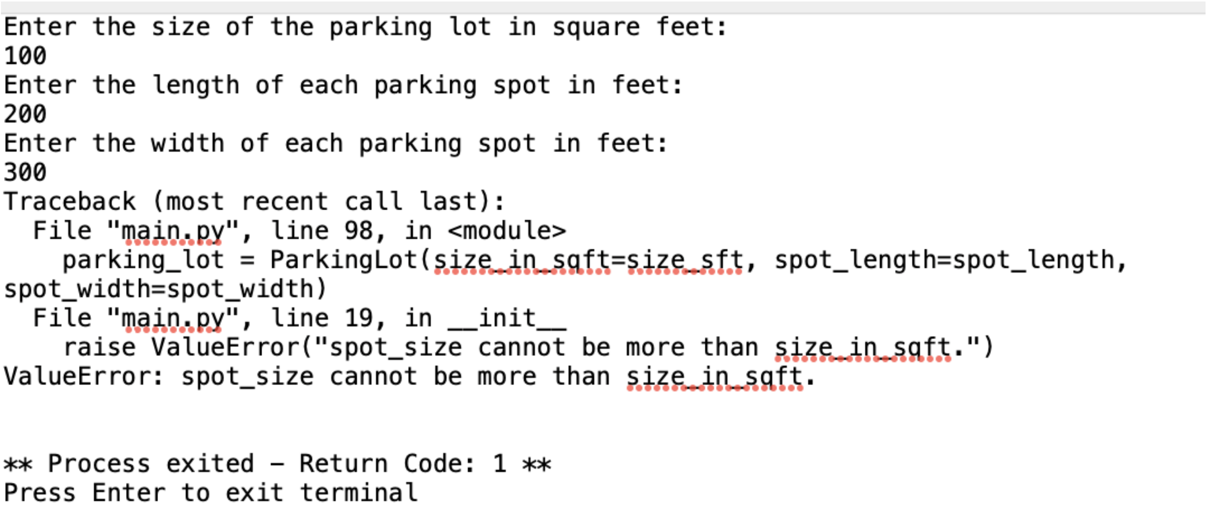
```
Enter the size of the parking lot in square feet:
1000
Enter the length of each parking spot in feet:
8
Enter the width of each parking spot in feet:
12
Car with license plate ABC1234 parked successfully in spot 1.
Car with license plate XYZ5678 parked successfully in spot 7.
Car with license plate LMN3456 parked successfully in spot 3.
Car with license plate QWE7890 parked successfully in spot 9.
Car with license plate DBC1234 parked successfully in spot 2.
Car with license plate WYZ5678 parked successfully in spot 4.
Car with license plate OMN3456 parked successfully in spot 8.
Car with license plate PWE7890 parked successfully in spot 5.
Car with license plate EBC1234 parked successfully in spot 0.
Car with license plate QYZ5678 parked successfully in spot 6.
Parking lot is full. Exiting program.
Parking lot is full. Exiting program.


** Process exited - Return Code: 0 **
Press Enter to exit terminal
```

Case-4: Invalid License Plate



```
Enter the size of the parking lot in square feet:
2000
Enter the length of each parking spot in feet:
8
Enter the width of each parking spot in feet:
12
Traceback (most recent call last):
  File "main.py", line 100, in <module>
    cars = [Car("ABC 1234"), Car("XYZ567890"), Car("!LMN3456"), Car("QWE7890"),Car("DBC1234"),
Car("WYZ5678"), Car("OMN3456"), Car("PWE7890"),Car("EBC1234"), Car("QYZ5678"), Car("JMN3456")]
  File "main.py", line 47, in __init__
    raise ValueError("License plate must be a 7 digit alphanumeric string.")
ValueError: License plate must be a 7 digit alphanumeric string.


** Process exited - Return Code: 1 **
Press Enter to exit terminal
```

Case-5: If spot size greater than total size area



```
Enter the size of the parking lot in square feet:
100
Enter the length of each parking spot in feet:
200
Enter the width of each parking spot in feet:
300
Traceback (most recent call last):
  File "main.py", line 98, in <module>
    parking_lot = ParkingLot(size_in_sqft=size_sft, spot_length=spot_length,
spot_width=spot_width)
  File "main.py", line 19, in __init__
    raise ValueError("spot_size cannot be more than size_in_sqft.")
ValueError: spot_size cannot be more than size_in_sqft.


** Process exited - Return Code: 1 **
Press Enter to exit terminal
```

Test Cases

Positive Test Cases:

Valid Parking and Mapping:
Scenario: Park cars in the parking lot and verify the vehicle-to-spot mapping.
Steps:
Initialize a parking lot and cars.
Park cars in specific spots.
Call map_vehicles_to_spots and verify the generated mapping matches the expected JSON-like dictionary.
Expected Outcome: The mapping should accurately reflect which cars are parked in which spots.

```
def test_positive_valid_parking_and_mapping():
    parking_lot = ParkingLot(size_in_sqft=480, spot_length=8, spot_width=12)
    cars = [Car("ABC1234"), Car("XYZ5678"), Car("LMN3456")]

    # Park cars in specific spots
    parking_lot.spots[0] = cars[0]  # Car "ABC1234" in spot 0
    parking_lot.spots[2] = cars[1]  # Car "XYZ5678" in spot 2
    parking_lot.spots[4] = cars[2]  # Car "LMN3456" in spot 4

    # Expected mapping
    expected_mapping = {
        "0": "ABC1234",
        "2": "XYZ5678",
        "4": "LMN3456"
    }

    # Get actual mapping
    actual_mapping = parking_lot.map_vehicles_to_spots()

    # Assert that actual mapping matches expected mapping
    assert actual_mapping == expected_mapping
```

Negative Test Cases

1. Parking Lot Full:

Scenario: Attempt to park more cars than the parking lot can accommodate.
Steps:
Initialize a parking lot with a small number of spots.
Try to park more cars than there are spots.
Verify that no additional cars can be parked once the lot is full.
Expected Outcome: Cars should not be able to park once the parking lot is full.

```
def test_negative_parking_lot_full():
    parking_lot = ParkingLot(size_in_sqft=96, spot_length=8, spot_width=12)  # Only 1 spot available
    car1 = Car("ABC1234")
    car2 = Car("XYZ5678")

    # Park the first car
    parking_lot.spots[0] = car1

    # Attempt to park the second car
    success, message = car2.park(parking_lot, 0)

    # Assert that parking is not successful
    assert not success
    assert message == "Spot 0 is already occupied."
```

----

2. Invalid License Plate:

Scenario: Attempt to create a car with an invalid license plate (not 7 characters or not alphanumeric).
Steps:
Initialize a car with an invalid license plate.
Verify that a ValueError is raised during initialization.
Expected Outcome: Initialization should fail with a ValueError due to an invalid license plate.

```
def test_negative_invalid_license_plate():
    try:
        car = Car("ABC12345")  # Invalid: More than 7 characters
        assert False  # Should not reach here
    except ValueError as e:
        assert str(e) == "License plate must be a 7 digit alphanumeric string."

    try:
        car = Car("ABC 123")  # Invalid: Contains spaces
        assert False  # Should not reach here
    except ValueError as e:
        assert str(e) == "License plate must be a 7 digit alphanumeric string."
```