Day 14: Yield Statement

Problem 1: Infinite Sequence of Prime Numbers

Write a generator function that yields an infinite sequence of prime numbers. Test the function by printing the first 10 prime numbers.

Example:

```
# Expected Output:
# 2, 3, 5, 7, 11, 13, 17, 19, 23, 29
```

Problem 2: Daily Task Scheduler

Imagine you are building a simple task scheduler for a week. Write a generator function that yields daily tasks from a list of tasks. After reaching the end of the week (7 tasks), it should cycle back to the first task again.

Example:

```
# Example task list: ['Coding', 'Exercise', 'Reading', 'Wor
k', 'Grocery Shopping', 'Cooking', 'Relaxing']
```

```
# Expected Output: Should loop through tasks one at a time
for multiple weeks when called repeatedly.
```

Problem 3: Traffic Signal Simulation

Simulate a traffic signal using a generator function. The generator should yield the traffic light colors in the sequence: "Green", "Yellow", and "Red". The traffic signal should cycle through the sequence indefinitely.

Example:

```
# Expected Output:
# Green
# Yellow
# Red
# Green
# ...
```

Problem 4: Movie Ticket Booking System

You are designing a simple movie ticket booking system. Write a generator function that yields available seat numbers for a theater (e.g., seat numbers 1 to 100). Once all seats are booked, the generator should stop.

Example:

```
# Expected Output: A sequence of seat numbers from 1 to 100
when booking tickets. Stop when all are booked.
```

Problem 5: Countdown Timer

Write a generator function that takes an input number and counts down to 0, yielding each number. This can be used to simulate a countdown timer.

Example:

```
# Expected Output:
# For countdown(5):
```

```
# 5
# 4
# 3
# 2
# 1
# 0
```

Problem 6: Weather Data Logger

You are collecting temperature data every hour from a weather station. Write a generator function that simulates yielding hourly temperature readings from a list of pre-recorded temperatures. After reaching the end of the list, it should stop.

Example:

```
# Example temperature readings: [22, 23, 21, 20, 19, 18, 1
7, 16]
# Expected Output: Should yield temperatures one at a time
for 8 hours and then stop.
```

Problem 7: Library Book Circulation

You are managing a library system where each book is returned and borrowed again. Write a generator function that keeps track of book circulation by yielding each book from a list in a circular fashion. After the last book is yielded, it should start from the first book again.

Example:

```
# Example books: ['Python Basics', 'Data Science Handbook',
'Machine Learning 101', 'AI Revolution']
# Expected Output: A cycle of books where each one is borro
wed and returned repeatedly.
```

Problem 8: Car Rental System

You are designing a car rental system where each car is rented out and then returned to the pool. Write a generator function that yields cars one by one from a list of available cars. Once all cars are rented, the system waits until one is returned before yielding the next car.

Example:

```
# Example cars: ['Sedan', 'SUV', 'Truck', 'Convertible']
# Expected Output: Each car is rented out in sequence until
none are left.
```

Problem 9: Real-Time Stock Price Monitor

You are building a stock price monitoring system. Write a generator function that yields real-time stock prices from a list of recorded stock prices. After all the prices are yielded, it should stop.

Example:

```
# Example stock prices: [100.5, 102.3, 101.8, 103.0, 104.2]
# Expected Output: Each stock price is yielded one by one u
ntil all prices are processed.
```

Problem 10: Taxi Dispatcher

You are managing a taxi dispatch system where taxis are assigned to passengers in the order they become available. Write a generator function that yields available taxis from a list of taxi IDs. Once all taxis are assigned, the system waits for a taxi to become available again before dispatching the next.

Example:

```
# Example taxis: ['Taxi1', 'Taxi2', 'Taxi3', 'Taxi4']
# Expected Output: Assign each taxi in sequence and wait fo
r them to become available again.
```
