LEAKY BUCKET :

```
import time

class LeakyBucket:
    def __init__(self, capacity, leak_rate):
        self.capacity = capacity        # max bucket size
        self.leak_rate = leak_rate      # leak rate (requests per second)
        self.water = 0.0                # current water level in bucket
        self.last_check = time.monotonic()

        self.allowed = 0                # total requests accepted (added to bucket)
        self.leaked = 0.0               # total requests processed (leaked out)
        self.rejected = 0               # total requests rejected (bucket full)

    def _leak(self):
        now = time.monotonic()
        elapsed = now - self.last_check
        leaked_amount = elapsed * self.leak_rate
        leaked_now = min(self.water, leaked_amount)

        self.water = max(0.0, self.water - leaked_amount)
        self.leaked += leaked_now
        self.last_check = now

    def add_request(self):
        self._leak()
        if self.water < self.capacity:
            self.water += 1
```

```
            self.allowed += 1
            return True   # request accepted
        else:
            self.rejected += 1
            return False  # request rejected


if __name__ == "__main__":
    capacity = float(input("Enter bucket capacity: "))
    leak_rate = float(input("Enter leak rate (requests per second): "))
    num_requests = int(input("Enter number of requests to simulate: "))
    interval = float(input("Enter interval between requests (seconds): "))

    bucket = LeakyBucket(capacity, leak_rate)

    for i in range(num_requests):
        if bucket.add_request():
            print(f"Request {i+1} allowed (current water: {bucket.water:.2f})")
        else:
            print(f"Request {i+1} rejected (bucket full)")
        time.sleep(interval)

    print("\nSimulation Summary:")
    print(f"Total requests allowed (added): {bucket.allowed}")
    print(f"Total requests leaked (processed): {bucket.leaked:.2f}")
    print(f"Total requests rejected: {bucket.rejected}")
```