import time

class SimpleCache:

def \_\_init\_\_(self, capacity):

self.cache = {}

self.capacity = capacity

self.order = [] # To simulate LRU or similar eviction policy

def get(self, key):

if key in self.cache:

# Simulate moving to front for LRU

if key in self.order:

self.order.remove(key)

self.order.append(key)

print(f"Cache Hit for key: {key}")

return self.cache[key]

print(f"Cache Miss for key: {key}")

return None

def put(self, key, value):

if len(self.cache) >= self.capacity:

# Evict the least recently used item (first in order list)

evicted\_key = self.order.pop(0)

del self.cache[evicted\_key]

print(f"Cache full, evicted key: {evicted\_key}")

self.cache[key] = value

self.order.append(key)

print(f"Added to cache: {key} -> {value}")

def expensive\_computation(data):

"""Simulates a time-consuming operation."""

print(f"Performing expensive computation for: {data}")

time.sleep(1) # Simulate delay

return data \* 2

# --- Simulation ---

cache\_size = 3

my\_cache = SimpleCache(cache\_size)

# First access - will be a cache miss and trigger computation

result1 = my\_cache.get("item\_A")

if result1 is None:

result1 = expensive\_computation(10)

my\_cache.put("item\_A", result1)

# Second access - will be a cache miss and trigger computation

result2 = my\_cache.get("item\_B")

if result2 is None:

result2 = expensive\_computation(20)

my\_cache.put("item\_B", result2)

# Third access - will be a cache miss and trigger computation

result3 = my\_cache.get("item\_C")

if result3 is None:

result3 = expensive\_computation(30)

my\_cache.put("item\_C", result3)

# Fourth access - will be a cache hit

result4 = my\_cache.get("item\_A")

# Fifth access - will be a cache miss and cause eviction

result5 = my\_cache.get("item\_D")

if result5 is None:

result5 = expensive\_computation(40)

my\_cache.put("item\_D", result5)

# Sixth access - will be a cache miss (item\_B was evicted)

result6 = my\_cache.get("item\_B")

if result6 is None:

result6 = expensive\_computation(20)

my\_cache.put("item\_B", result6)