1. Urutkan dulu barang dan truk dari kapasitas terbesar ke terkecil, masukan barang ke truk pertama sampai penuh lalu truk selanjutnya
   Dalam proses pemogramannya jadi di urutkan dulu sesuai kapasitas

2.

```
SELECT *

FROM users

WHERE country = 'Germany'

AND lastLogin < DATE_SUB(CURDATE(), INTERVAL 6 MONTH);
```

Pilih dulu negaranya baru cek terakhir loginnya

3. Tangki air berbentuk tabung dengan diameter 4 m, tinggi 20 m, debit masuk 5 m^3/menit, keluar 3 m^3/menit.

⍰ Langkah:

1. Volume tabung = $\pi \times r^2 \times t$
   $= 3.14 \times (2)^2 \times 20$
   $= 3.14 \times 4 \times 20 =$ **251.2 m^3**
2. Debit bersih = masuk - keluar = 5 - 3 = **2 m^3/menit**
3. Waktu = Volume / Debit = 251.2 / 2 = **125.6 menit**

□ Jawaban:

Waktu pengisian penuh = **±126 menit (≈ 2 jam 6 menit)**

4. Ambil nama pelanggan dan ID nya, jika pelanggannya melakukan transaksi lebih dari 3 kali maka itu yang di tampilkan. Id pesanan bisa jadi acuan pembelian ke berapa kali nya dengan ID dan nama pelanggan yang sama
5. Prinsipnya adalah harga jual dan harga modal, lakukan perhitungan untuk mencari rata2 margin, kemudian lakukan pencarian data dengan margin yang di atas rata-rata
6. Hitung jumlah pesanan dan total nilai penjualan

```
SELECT
   MONTH(o.orderDate) AS bulan,
   COUNT(DISTINCT o.orderID) AS jumlah_pesanan,
   SUM(od.quantity * od.unitPrice) AS total_penjualan
FROM orders o
JOIN order_details od ON o.orderID = od.orderID
WHERE YEAR(o.orderDate) = YEAR(CURDATE())
GROUP BY MONTH(o.orderDate)
ORDER BY bulan;
```

7. Tampilkan ketersediaan stok

```
SELECT
 p.productID,
 p.productName,
c.categoryName,
 p.unitsInStock
FROM products p
JOIN categories c ON p.categoryID = c.categoryID
ORDER BY p.productName;
```

```
SELECT
    p.productID,
    p.productName,
    (p.unitPrice - p.costPrice) AS margin
FROM products p
WHERE (p.unitPrice - p.costPrice) > (
    SELECT AVG(unitPrice - costPrice) FROM products
);
```

```
SELECT

    MONTH(o.orderDate) AS bulan,

    COUNT(DISTINCT o.orderID) AS jumlah_pesanan,

    SUM(od.quantity * od.unitPrice) AS total_penjualan

FROM orders o

JOIN order_details od ON o.orderID = od.orderID

WHERE YEAR(o.orderDate) = YEAR(CURDATE())

GROUP BY MONTH(o.orderDate)

ORDER BY bulan;
```

```sql
SELECT
    p.productID,
    p.productName,
    c.categoryName,
    p.unitsInStock
FROM products p
JOIN categories c ON p.categoryID = c.categoryID
ORDER BY p.productName;
```

```sql
SELECT
    o.orderID,
    c.customerName,
    o.orderDate,
    o.shippedDate,
    DATEDIFF(o.shippedDate, o.orderDate) AS waktu_pengiriman
FROM orders o
JOIN customers c ON o.customerID = c.customerID
WHERE DATEDIFF(o.shippedDate, o.orderDate) < 3
ORDER BY waktu_pengiriman ASC;
```

1. Soal satu slied 2

```
# log_processor.py

import re

from collections import defaultdict


LOG_LINE_RE = re.compile(r'\"(?P<method>[A-Z]+)\s+(?P<path>/\S*)\s+HTTP/[0-
9.]+\"\s+(?P<status>\d{3})')


def process_log_file(path):

    counts_by_method = defaultdict(int)

    counts_by_status = defaultdict(int)

    detailed = defaultdict(int)  # kombinasi (method, status)


    with open(path, 'r', encoding='utf-8') as f:

        for line in f:

            m = LOG_LINE_RE.search(line)

            if not m:

                continue

            method = m.group('method')

            status = m.group('status')


            counts_by_method[method] += 1

            counts_by_status[status] += 1

            detailed[(method, status)] += 1


    return counts_by_method, counts_by_status, detailed


if __name__ == '__main__':
    # Contoh: file 'access.log'
    methods, statuses, detail = process_log_file('access.log')

    print("Jumlah permintaan per metode:")
    for method, cnt in sorted(methods.items(), key=lambda x: (-x[1], x[0])):
        print(f"  {method}: {cnt}")

    print("\nJumlah permintaan per status code:")
    for status, cnt in sorted(statuses.items(), key=lambda x: (-int(x[0]), -x[1]) if False else (-x[1],
x[0])):
        print(f"  {status}: {cnt}")

    print("\nRincian (method, status):")
    for (method, status), cnt in sorted(detail.items(), key=lambda x: (-x[1], x[0])):
        print(f"  {method} {status}: {cnt}")
```

2. Soal 2 slide 2

```
# scheduler.py
from collections import defaultdict, deque


def schedule_tasks(tasks):
    """
    tasks: dict mapping taskID -> {'duration': int, 'deps': [taskID, ...]}
    returns: topo_order list, start_finish dict, total_time (makespan)
    """
    # build graph
    indeg = defaultdict(int)
    graph = defaultdict(list)
    for tid, meta in tasks.items():
        indeg[tid]  # ensure key exists
    for tid, meta in tasks.items():
        for dep in meta.get('deps', []):
            graph[dep].append(tid)
            indeg[tid] += 1

    # Kahn topological sort
    q = deque([tid for tid, d in indeg.items() if d == 0])
    topo = []
    while q:
        u = q.popleft()
        topo.append(u)
        for v in graph[u]:
            indeg[v] -= 1
            if indeg[v] == 0:
                q.append(v)

    if len(topo) != len(tasks):
        raise ValueError("Ada siklus dependency — tidak ada urutan valid")

    # earliest start/finish
    est = {tid: 0 for tid in tasks}  # earliest start time
    eft = {tid: 0 for tid in tasks}  # earliest finish time

    for t in topo:
        deps = tasks[t].get('deps', [])
        if deps:
            est[t] = max(eft[d] for d in deps)
        else:
            est[t] = 0
        eft[t] = est[t] + tasks[t]['duration']

    makespan = max(eft.values()) if eft else 0
    schedule = {t: (est[t], eft[t]) for t in topo}
    return topo, schedule, makespan

if __name__ == '__main__':
    # Contoh input dari soal:
    tasks = {
        1: {'duration': 3, 'deps': []},
        2: {'duration': 5, 'deps': [1]},
        3: {'duration': 2, 'deps': [1]},

        4: {'duration': 4, 'deps': [2, 3]},

    }


    order, sched, total = schedule_tasks(tasks)

    print("Topological order valid:", order)

    print("Schedule (start, finish):")

    for t in order:

        s, f = sched[t]

        print(f" Task {t}: start={s}, finish={f}")

    print("Total waktu (makespan):", total)
```

3. Soal 2 slide 2

```
# warehouse.py
class Warehouse:
  def __init__(self):
    self.items = {}  # itemID -> {name, stock, threshold}

  def add_item(self, itemID, name, stock=0, threshold=0):
    if itemID in self.items:
      raise KeyError(f"Item {itemID} sudah ada.")
    self.items[itemID] = {'name': name, 'stock': stock, 'threshold': threshold}

  def add_stock(self, itemID, qty):
    if itemID not in self.items:
      raise KeyError("Item tidak ditemukan")
    if qty < 0:
      raise ValueError("qty harus positif")
    self.items[itemID]['stock'] += qty

  def reduce_stock(self, itemID, qty):
    if itemID not in self.items:
      raise KeyError("Item tidak ditemukan")
    if qty < 0:
      raise ValueError("qty harus positif")
    if self.items[itemID]['stock'] < qty:
      raise ValueError("Stok tidak cukup")
    self.items[itemID]['stock'] -= qty

  def list_below_threshold(self):
    return {iid: data for iid, data in self.items.items() if data['stock'] <= data['threshold']}

  def __str__(self):
    lines = ["ID | Name | Stock | Threshold"]
    for iid, d in self.items.items():
      lines.append(f"{iid} | {d['name']} | {d['stock']} | {d['threshold']}")
    return "\n".join(lines)

if __name__ == '__main__':
  w = Warehouse()
  w.add_item('A001', 'Baut M4', stock=50, threshold=10)
  w.add_item('A002', 'Mur M4', stock=8, threshold=10)
  w.add_item('A003', 'Semen 40kg', stock=2, threshold=5)

  print("Semua barang:")
  print(w)
  print("\nTambah stok A002 sebanyak 10")
```

```
w.add_stock('A002', 10)
print("Kurangi stok A001 sebanyak 45")
w.reduce_stock('A001', 45)

print("\nBarang di bawah threshold:")
below = w.list_below_threshold()
for iid, d in below.items():
    print(f"{iid} - {d['name']} : stock={d['stock']}, threshold={d['threshold']}")
```