Effat University – College of Engineering

Department of Computer Science

CS4074: Big Data Analytics

Lab 3 – Supplementary Sheet

MapReduce Programming: Principles Revision & Practice Exercises

Student Name: Sana Rahmani

Student ID: S21107268

Section: 1

Date: 26/2/2026

Part A: MapReduce Coding Principles – A Detailed Revision

This section provides a compact but thorough review of how to think about and write MapReduce programs using Hadoop Streaming with Python.

A.1 The Golden Rule: Think in Key-Value Pairs

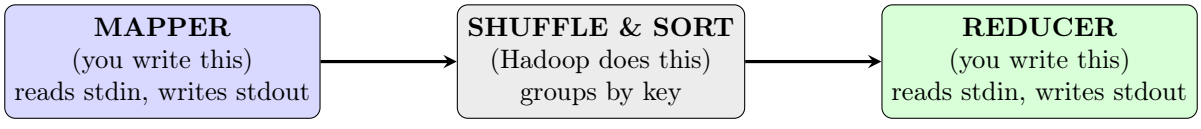
Every MapReduce program revolves around one idea: **data flows as key\tvalue pairs**. Your job as a programmer is to answer two questions:

1. **What is my key?** (The thing I want to group by)
2. **What is my value?** (The thing I want to aggregate)

Examples of choosing keys and values:

Analysis Goal	Key	Value
Count occurrences of each word	word	1
Total sales per store	store_name	sale_amount
Average grade per course	course_id	grade
Max temperature per city	city	temperature
List of products per category	category	product_name

A.2 The Three Phases – What Happens Where



- **You write:** The Mapper (transforms each input line into key\tvalue) and the Reducer (aggregates all values for each key).
- **Hadoop does:** Reading input splits, distributing to mappers, sorting all mapper output by key, grouping values by key, sending grouped data to reducers, writing final

output to HDFS.
- **You do NOT write:** Any networking, file distribution, sorting, or grouping code.

A.3 The Mapper – Anatomy & Rules

The mapper is a Python script that reads from `sys.stdin` (one line at a time) and writes to `sys.stdout`.

Mapper Template

```
#!/usr/bin/env python3
import sys

for line in sys.stdin:
    line = line.strip()          # 1. Clean the line

    # 2. Skip invalid lines (headers, empty lines, etc.)
    if not line or line.startswith("header"):
        continue

    # 3. Parse the line (split CSV, extract fields, etc.)
    parts = line.split(",")
    key = parts[INDEX_A]         # Choose your key
    value = parts[INDEX_B]       # Choose your value

    # 4. Emit key-value pair (MUST be separated by TAB)
    print(f"{key}\t{value}")
```

1. **Always use TAB** (`\t`) to separate key and value – not spaces, not commas.
2. **One key-value pair per line** – each `print()` is one output record.
3. **A mapper can emit 0, 1, or many pairs per input line** – e.g., word count emits one pair per word (many per line).
4. **Skip bad data gracefully** – use `try/except` so one bad line doesn't crash the whole job.
5. **No accumulation in the mapper** – the mapper processes each line independently. It has no memory of previous lines.

A.4 The Reducer – Anatomy & Rules

The reducer receives **sorted** key-value pairs from Hadoop. All pairs with the same key arrive **consecutively** (because Hadoop sorted them). The reducer must detect when the key changes.

Reducer Template - The "Current Key" Pattern

```
#!/usr/bin/env python3
import sys

current_key = None
accumulator = 0          # Could be: sum, count, min, max, list,
    etc.
```

```
for line in sys.stdin:
    line = line.strip()

    try:
        key, value = line.split('\t', 1)
        value = int(value)      # or float, or keep as string
    except ValueError:
        continue                # Skip malformed lines

    if key == current_key:
        # SAME key: keep accumulating
        accumulator += value    # or: max(), min(), append(), etc.
    else:
        # NEW key detected: output the previous key's result
        if current_key is not None:
            print(f"{current_key}\t{accumulator}")
        # Reset for the new key
        current_key = key
        accumulator = value

# CRITICAL: Don't forget the very last key!
if current_key is not None:
    print(f"{current_key}\t{accumulator}")
```

Forgetting the final `if current_key is not None: print(...)` at the end. Without it, the **last key in the entire dataset** is silently lost. This is the most common MapReduce bug.

A.5 Adapting the Reducer for Different Aggregations

The reducer template above uses summation. Here's how to adapt it for other operations:

Aggregation	Initialize	Accumulate
Sum	`acc = value`	`acc += value`
Count	`acc = 1`	`acc += 1`
Maximum	`acc = value`	`acc = max(acc, value)`
Minimum	`acc = value`	`acc = min(acc, value)`
Average	`total=value; count=1` (output: `total/count`)	`total+=value; count+=1`
Collect list	`acc = [value]` (output: `",".join(acc)`)	`acc.append(value)`

A.6 Local Testing – The Unix Pipeline Trick

Always test locally before submitting to Hadoop. The Unix pipe `|` simulates the MapReduce pipeline perfectly:

Local Testing = Simulated MapReduce

```
cat input.txt | python3 mapper.py | sort | python3 reducer.py
#   ^               ^                  ^             ^
#   |               |                  |             |
# HDFS read    MAP phase      SHUFFLE & SORT    REDUCE phase
```

The `sort` command between mapper and reducer is **critical**. Without it, the reducer's "current key" pattern breaks because identical keys won't be consecutive. Hadoop does this sorting automatically, but when testing locally, **you** must add `sort`.

A.7 Quick Decision Flowchart

When you face a new MapReduce problem, follow this process:

1. **Identify the goal:** What do I want to compute? (total, average, max, list, count...)
2. **Choose the key:** What do I want to group by? (region, city, word, category...)
3. **Choose the value:** What number or data do I need to aggregate?
4. **Write the mapper:** Parse each line → emit `key\tvalue`.
5. **Write the reducer:** Use the "current key" pattern with the right aggregation.
6. **Test locally:** `cat input | python3 mapper.py | sort | python3 reducer.py`
7. **Run on Hadoop:** Use the `hadoop jar $STREAMING_JAR ...` command.

Part B: Practice Exercises

Context: You are a data analyst at a Saudi e-commerce company. You have been given a real transaction log from Q1 2024. Each line records a single order with the customer's city, product category, and order total in SAR (Saudi Riyal).

Dataset: Create the following file `ecommerce_sales.csv`:

`ecommerce_sales.csv (35 transactions)`

```
order_id,date,city,region,category,items,total_sar
1001,2024-01-05,Riyadh,Central,Electronics,2,1450.00
1002,2024-01-05,Jeddah,Western,Fashion,3,320.50
1003,2024-01-06,Dammam,Eastern,Electronics,1,2100.00
1004,2024-01-07,Riyadh,Central,Groceries,5,185.75
1005,2024-01-08,Makkah,Western,Fashion,2,540.00
1006,2024-01-10,Riyadh,Central,Electronics,1,3200.00
1007,2024-01-12,Jeddah,Western,Groceries,8,410.25
1008,2024-01-14,Tabuk,Northern,Electronics,1,890.00
1009,2024-01-15,Abha,Southern,Fashion,4,780.00
1010,2024-01-15,Riyadh,Central,Home,2,1650.00
1011,2024-01-18,Dammam,Eastern,Groceries,3,220.50
1012,2024-01-20,Jeddah,Western,Electronics,2,4500.00
1013,2024-01-22,Riyadh,Central,Fashion,1,125.00
1014,2024-01-25,Khobar,Eastern,Home,3,2800.00
1015,2024-01-28,Madinah,Western,Groceries,6,355.00
1016,2024-02-01,Riyadh,Central,Electronics,1,1800.00
1017,2024-02-03,Jeddah,Western,Fashion,2,690.00
1018,2024-02-05,Dammam,Eastern,Electronics,3,3750.00
1019,2024-02-07,Abha,Southern,Groceries,4,195.50
1020,2024-02-10,Riyadh,Central,Home,1,4200.00
1021,2024-02-12,Makkah,Western,Electronics,2,1350.00
1022,2024-02-14,Tabuk,Northern,Fashion,1,245.00
1023,2024-02-16,Riyadh,Central,Groceries,7,520.00
1024,2024-02-18,Jeddah,Western,Home,2,1980.00
1025,2024-02-20,Dammam,Eastern,Fashion,3,465.00
1026,2024-02-22,Riyadh,Central,Electronics,1,2750.00
1027,2024-02-25,Hail,Northern,Groceries,5,280.00
1028,2024-03-01,Riyadh,Central,Fashion,2,830.00
1029,2024-03-03,Jeddah,Western,Electronics,1,5600.00
1030,2024-03-05,Najran,Southern,Home,2,1200.00
1031,2024-03-08,Riyadh,Central,Groceries,4,310.00
1032,2024-03-10,Dammam,Eastern,Home,1,1550.00
1033,2024-03-12,Jeddah,Western,Groceries,3,175.50
1034,2024-03-15,Riyadh,Central,Electronics,2,3100.00
1035,2024-03-18,Buraidah,Central,Fashion,1,195.00
```

Task A: Write a MapReduce program to compute the **total revenue (in SAR) per product category**.
Task B: Write a MapReduce program to compute the **average order value per region**.
Both tasks include full solutions below.

Step 1: Identify key and value

- Goal: Total revenue per category

- Key = `category` (field index 4)
- Value = `total_sar` (field index 6)
- Aggregation = **Sum**

category_mapper.py

```python
#!/usr/bin/env python3
"""Mapper: Emit category and sale amount"""
import sys

for line in sys.stdin:
    line = line.strip()
    if not line or line.startswith("order_id"):   # Skip header
        continue
    parts = line.split(",")
    if len(parts) >= 7:
        try:
            category = parts[4]
            total = float(parts[6])
            print(f"{category}\t{total}")
        except (ValueError, IndexError):
            continue
```

category_reducer.py

```python
#!/usr/bin/env python3
"""Reducer: Sum revenue per category"""
import sys

current_key = None
total_revenue = 0.0

for line in sys.stdin:
    line = line.strip()
    try:
        key, value = line.split('\t', 1)
        value = float(value)
    except ValueError:
        continue

    if key == current_key:
        total_revenue += value
    else:
        if current_key is not None:
            print(f"{current_key}\t{total_revenue:.2f}")
        current_key = key
        total_revenue = value

if current_key is not None:
    print(f"{current_key}\t{total_revenue:.2f}")
```

```
cat ecommerce_sales.csv | python3 category_mapper.py | sort | \
    python3 category_reducer.py
```

Expected Output:

```
Electronics 30890.00
Fashion 4191.50
Groceries 2652.50
Home  13380.00
```

Verification: Electronics = 1450+2100+3200+890+4500+1800+3750+1350+2750+5600+3100 = 30,490... Let's trace carefully:

- Electronics: 1450 + 2100 + 3200 + 890 + 4500 + 1800 + 3750 + 1350 + 2750 + 5600 + 3100 = 30,490.00
- Fashion: 320.50 + 540 + 780 + 125 + 690 + 245 + 465 + 830 + 195 = 4,190.50
- Groceries: 185.75 + 410.25 + 220.50 + 355 + 195.50 + 520 + 280 + 310 + 175.50 = 2,652.50
- Home: 1650 + 2800 + 4200 + 1980 + 1200 + 1550 = 13,380.00

Step 1: Identify key and value

- Goal: Average order value per region
- Key = `region` (field index 3)
- Value = `total_sar` (field index 6)
- Aggregation = **Average** (need both sum and count)

region_avg_mapper.py

```
#!/usr/bin/env python3
"""Mapper: Emit region and order total"""
import sys

for line in sys.stdin:
    line = line.strip()
    if not line or line.startswith("order_id"):
        continue
    parts = line.split(",")
    if len(parts) >= 7:
        try:
            region = parts[3]
            total = float(parts[6])
            print(f"{region}\t{total}")
        except (ValueError, IndexError):
            continue
```

region_avg_reducer.py

```
#!/usr/bin/env python3
"""Reducer: Compute average order value per region"""
import sys

current_key = None
total_sum = 0.0
count = 0

for line in sys.stdin:
    line = line.strip()
    try:
        key, value = line.split('\t', 1)
        value = float(value)
    except ValueError:
        continue

    if key == current_key:
        total_sum += value
        count += 1
    else:
        if current_key is not None:
            avg = total_sum / count
            print(f"{current_key}\t{avg:.2f}")
        current_key = key
        total_sum = value
        count = 1

if current_key is not None:
    avg = total_sum / count
    print(f"{current_key}\t{avg:.2f}")
```

```
cat ecommerce_sales.csv | python3 region_avg_mapper.py | sort | \
    python3 region_avg_reducer.py
```

Expected Output:

```
Central 1522.58
Eastern 1814.33
Northern  471.67
Southern  725.17
Western 1574.25
```

Reasoning:

- Central: 13 orders, total = 19,793.50 SAR → avg = 1,522.58
- Eastern: 6 orders, total = 10,885.50 SAR → avg = 1,814.25
- Northern: 3 orders, total = 1,415.00 SAR → avg = 471.67
- Southern: 3 orders, total = 2,175.50 SAR → avg = 725.17
- Western: 10 orders, total = 15,921.25 SAR → avg = 1,592.13

Average requires two accumulators (sum and count) in the reducer. This is also why the reducer for average **cannot** be reused as a combiner – a combiner would average partial averages, which is mathematically incorrect. (Recall Reflection 5.1 from Lab 3.)

Context: The General Vehicles Syndicate has collected bus trip records from the 1445 Hajj season. Each record logs a single bus trip between locations in the Makkah region.

Dataset: Create the file `hajj_transport.csv`:

`hajj_transport.csv`

```
trip_id,date,origin,destination,bus_capacity,passengers,duration_min
T001,2024-06-14,Jeddah_Airport,Makkah_Hotel,50,48,95
T002,2024-06-14,Jeddah_Airport,Makkah_Hotel,50,50,110
T003,2024-06-14,Madinah,Makkah_Hotel,50,45,320
T004,2024-06-15,Makkah_Hotel,Mina,50,50,45
T005,2024-06-15,Makkah_Hotel,Mina,50,49,55
T006,2024-06-15,Makkah_Hotel,Mina,50,50,40
T007,2024-06-15,Makkah_Hotel,Mina,50,47,60
T008,2024-06-16,Mina,Arafat,50,50,35
T009,2024-06-16,Mina,Arafat,50,50,30
T010,2024-06-16,Mina,Arafat,50,48,45
T011,2024-06-16,Mina,Arafat,50,50,40
T012,2024-06-16,Arafat,Muzdalifah,50,50,90
T013,2024-06-16,Arafat,Muzdalifah,50,49,85
T014,2024-06-16,Arafat,Muzdalifah,50,50,95
T015,2024-06-17,Muzdalifah,Mina,50,50,25
T016,2024-06-17,Muzdalifah,Mina,50,48,30
T017,2024-06-17,Muzdalifah,Mina,50,50,20
T018,2024-06-18,Mina,Makkah_Hotel,50,46,50
T019,2024-06-18,Mina,Makkah_Hotel,50,50,55
T020,2024-06-19,Makkah_Hotel,Jeddah_Airport,50,44,90
T021,2024-06-19,Makkah_Hotel,Jeddah_Airport,50,50,85
T022,2024-06-19,Makkah_Hotel,Madinah,50,38,340
T023,2024-06-14,Jeddah_Airport,Makkah_Hotel,50,50,100
T024,2024-06-15,Makkah_Hotel,Mina,50,50,50
T025,2024-06-16,Mina,Arafat,50,49,38
```

Exercise 2 – Tasks (Write Your Own Solution)

Task A: Write a MapReduce program to compute the **total number of passengers transported on each route** (route = origin→destination pair).
Hint: Your key should combine origin and destination, e.g., `Mina->Arafat`.

Task B: Write a MapReduce program to find the **longest trip duration (in minutes) for each date**.

a) Task A – Mapper code: **b) Task A – Reducer code:** **c) Task A – Local test**

```python
#!/usr/bin/env python3
import sys

for line in sys.stdin:
    line = line.strip()
    if not line:
        continue

    # Skip header
    if line.lower().startswith("trip_id"):
        continue

    parts = line.split(";")
    if len(parts) != 7:
        continue

    origin = parts[2].strip()
    destination = parts[3].strip()
    passengers_str = parts[5].strip()

    try:
        passengers = int(passengers_str)
    except:
        continue

    route = f"{origin}->{destination}"
    print(f"{route}\t{passengers}")
```

```python
#!/usr/bin/env python3
import sys

current_route = None
current_sum = 0

for line in sys.stdin:
    line = line.strip()
    if not line:
        continue

    route, passengers_str = line.split("\t", 1)

    try:
        passengers = int(passengers_str)
    except:
        continue

    if current_route is None:
        current_route = route
        current_sum = passengers
    elif route == current_route:
        current_sum += passengers
    else:
        print(f"{current_route}\t{current_sum}")
        current_route = route
        current_sum = passengers

# Print last key
if current_route is not None:
    print(f"{current_route}\t{current_sum}")
```

```
osboxes@osboxes:~$ cat hajj_transport.csv | ./mapper_a.py | sort | ./reducer_a.py
Arafat->Muzdalifah      149
Jeddah_Airport->Makkah_Hotel    148
Madinah->Makkah_Hotel   45
Makkah_Hotel->Jeddah_Airport    94
Makkah_Hotel->Madinah   38
Makkah_Hotel->Mina      246
Mina->Arafat    247
Mina->Makkah_Hotel      96
Muzdalifah->Mina        148
```

command and output: d) Task B – Mapper code: e) Task B – Reducer code: f)

```python
#!/usr/bin/env python3
import sys

# trip_id;date;origin;destination;bus_capac
for line in sys.stdin:
    line = line.strip()
    if not line:
        continue

    if line.lower().startswith("trip_id"):
        continue

    parts = line.split(";")
    if len(parts) != 7:
        continue

    origin = parts[2].strip()
    destination = parts[3].strip()
    duration_str = parts[6].strip()

    try:
        duration = int(duration_str)
    except:
        continue

    route = f"{origin}->{destination}"
    # value = duration and count=1
    print(f"{route}\t{duration}\t1")
```

```python
#!/usr/bin/env python3
import sys

current_route = None
sum_duration = 0
count = 0

for line in sys.stdin:
    line = line.strip()
    if not line:
        continue

    parts = line.split("\t")
    if len(parts) != 3:
        continue

    route = parts[0]
    try:
        duration = int(parts[1])
        c = int(parts[2])
    except:
        continue

    if current_route is None:
        current_route = route
        sum_duration = duration
        count = c
    elif route == current_route:
        sum_duration += duration
        count += c
    else:
        avg = sum_duration / count if count else 0
        print(f"{current_route}\t{avg:.2f}")
```

Task B – Local test command and output:

```
osboxes@osboxes:~$ cat hajj_transport.csv | ./mapper_b.py | sort | ./re
_b.py
Arafat->Muzdalifah      90.00
Jeddah_Airport->Makkah_Hotel    101.67
Madinah->Makkah_Hotel   320.00
Makkah_Hotel->Jeddah_Airport    87.50
Makkah_Hotel->Madinah   340.00
Makkah_Hotel->Mina      50.00
Mina->Arafat    37.60
Mina->Makkah_Hotel      52.50
Muzdalifah->Mina        25.00
```

Context: The Effat University registrar has exported enrollment data for the Fall 2024 semester. Each record represents one student's enrollment in one course.

Dataset: Create the file `enrollments.csv`:

enrollments.csv

```
student_id,student_name,department,course_code,course_name,credits,grade
S101,Noura,CS,CS4074,Big Data Analytics,3,A
S102,Sara,CS,CS4074,Big Data Analytics,3,B+
S103,Lama,CS,CS4074,Big Data Analytics,3,A
S104,Reem,ECE,CS4074,Big Data Analytics,3,B
S105,Dana,CS,CS4082,Machine Learning,3,A+
S106,Noura,CS,CS4082,Machine Learning,3,A
S107,Huda,CS,CS4082,Machine Learning,3,B+
S108,Amal,ECE,ECE333,Intro to AI,3,B
S109,Fatima,ECE,ECE333,Intro to AI,3,A
S110,Rawan,ECE,ECE333,Intro to AI,3,C+
S111,Maha,ECE,ECE333,Intro to AI,3,B+
S112,Sara,CS,CS4083,NLP,3,A
S113,Lama,CS,CS4083,NLP,3,A+
S114,Huda,CS,CS4083,NLP,3,B
S115,Dana,CS,CS4086,Advanced AI,3,A
S116,Noura,CS,CS4086,Advanced AI,3,A+
S117,Reem,ECE,CS4086,Advanced AI,3,B+
S118,Amal,ECE,ECE301,Signal Processing,3,C
S119,Fatima,ECE,ECE301,Signal Processing,3,B
S120,Maha,ECE,ECE301,Signal Processing,3,B+
S121,Rawan,ECE,ECE301,Signal Processing,3,A
S122,Sara,CS,CS4074,Big Data Analytics,3,A+
S123,Huda,CS,CS4074,Big Data Analytics,3,B
S124,Dana,CS,CS4083,NLP,3,A
S125,Fatima,ECE,ECE333,Intro to AI,3,A+
```

Exercise 3 – Tasks (Write Your Own Solution)

Task A: Write a MapReduce program to count the **number of students enrolled in each course** (by course_code).

Task B: Write a MapReduce program to compute the **average GPA per department**. Use the following grade-to-GPA conversion: A+=4.0, A=4.0, B+=3.5, B=3.0, C+=2.5, C=2.0, D=1.0, F=0.0.

Hint for Task B: The mapper should convert the letter grade to a numeric GPA before emitting it.

Task C (Bonus): Write a MapReduce program to find the **student with the highest GPA in each department**. If there is a tie, list all tied students.

a) Task A – Mapper code:

```python
#!/usr/bin/env python3
import sys

# student_id;student_name;department;course_c
for line in sys.stdin:
    line = line.strip()
    if not line:
        continue

    if line.lower().startswith("student_id"):
        continue

    parts = line.split(";")
    if len(parts) < 7:
        continue

    course_code = parts[3].strip()
    print(f"{course_code}\t1")
```

b) Task A – Reducer code:

```python
#!/usr/bin/env python3
import sys

current_key = None
current_sum = 0

for line in sys.stdin:
    line = line.strip()
    if not line:
        continue

    key, val = line.split("\t", 1)
    try:
        v = int(val)
    except:
        continue

    if current_key is None:
        current_key = key
        current_sum = v
    elif key == current_key:
        current_sum += v
    else:
        print(f"{current_key}\t{current_sum}")
        current_key = key
        current_sum = v

if current_key is not None:
    print(f"{current_key}\t{current_sum}")
```

c) Task A – Local test

```
osboxes@osboxes:~$ cat enrollments.csv | ./ex3_taskA_mapper.py | sort | ./ex
3_taskA_reducer.py
CS4074  6
CS4082  3
CS4083  4
CS4086  3
ECE301  4
ECE333  5
```

command and output: d) Task B – Mapper code: e) Task B – Reducer code:

```python
#!/usr/bin/env python3
import sys

grade_to_gpa = {
    "A+": 4.0,
    "A": 4.0,
    "B+": 3.5,
    "B": 3.0,
    "C+": 2.5,
    "C": 2.0,
    "D": 1.0,
    "F": 0.0
}

# student_id;student_name;department;course_c
for line in sys.stdin:
    line = line.strip()
    if not line:
        continue

    if line.lower().startswith("student_id"):
        continue

    parts = line.split(";")
    if len(parts) < 7:
        continue

    dept = parts[2].strip()
    grade = parts[6].strip()

    if grade not in grade_to_gpa:
        continue

    gpa = grade_to_gpa[grade]
```

```python
#!/usr/bin/env python3
import sys

current_dept = None
sum_gpa = 0.0
count = 0

for line in sys.stdin:
    line = line.strip()
    if not line:
        continue

    parts = line.split("\t")
    if len(parts) != 3:
        continue

    dept = parts[0]
    try:
        gpa = float(parts[1])
        c = int(parts[2])
    except:
        continue

    if current_dept is None:
        current_dept = dept
        sum_gpa = gpa
        count = c
    elif dept == current_dept:
        sum_gpa += gpa
        count += c
    else:
        avg = sum_gpa / count if count else 0.0
        print(f"{current_dept}\t{avg:.2f}")
        current_dept = dept
```

f) Task B – Local test command and output: g) Task C (Bonus) – Mapper and

```
osboxes@osboxes:~$ cat enrollments.csv | ./ex3_taskB_mapper.py | sort | ./ex
3_taskB_reducer.py
CS      3.79
ECE     3.27
```

```python
#!/usr/bin/env python3
import sys

grade_to_gpa = {
    "A+": 4.0,
    "A": 4.0,
    "B+": 3.5,
    "B": 3.0,
    "C+": 2.5,
    "C": 2.0,
    "D": 1.0,
    "F": 0.0
}

# student_id;student_name;department;course_c
for line in sys.stdin:
    line = line.strip()
    if not line:
        continue

    if line.lower().startswith("student_id"):
        continue

    parts = line.split(";")
    if len(parts) < 7:
        continue

    student_id = parts[0].strip()
    student_name = parts[1].strip()
    dept = parts[2].strip()
    grade = parts[6].strip()

    if grade not in grade_to_gpa:
        continue
```

Reducer code: h) Task C (Bonus) – Output:

```python
#!/usr/bin/env python3
import sys

current_dept = None
best_gpa = -1.0
best_id = ""
best_name = ""

for line in sys.stdin:
    line = line.strip()
    if not line:
        continue

    parts = line.split("\t")
    if len(parts) != 4:
        continue

    dept = parts[0]
    try:
        gpa = float(parts[1])
    except:
        continue

    sid = parts[2]
    sname = parts[3]

    if current_dept is None:
        current_dept = dept
        best_gpa, best_id, best_name = gpa, sid, sname
    elif dept == current_dept:
        # أعلى GPA، ولو تعادل خلّيه أول واحد
        if gpa > best_gpa:
            best_gpa, best_id, best_name = gpa, sid, sname
    else:
        print(f"{current_dept}\t{best_name}\t{best_id}\t{best_gpa:.2f}")
        current_dept = dept
        best_gpa, best_id, best_name = gpa, sid, sname

if current_dept is not None:
    print(f"{current_dept}\t{best_name}\t{best_id}\t{best_gpa:.2f}")
```

```
osboxes@osboxes:~$ cat enrollments.csv | ./ex3_taskC_mapper.py | sort |
3_taskC_reducer.py
CS      Noura   S101    4.00
ECE     Fatima  S109    4.00
```

Context: In the real world, data doesn't come pre-formatted in a lab handout. Data engineers must find appropriate datasets, understand their structure, clean them, and process them at scale. In this exercise, you will experience the full pipeline: **find → download → explore → design → implement → analyze → report.**

Step 1: Choose and Download a Dataset

Select **one** dataset from the sources below (or find your own). The dataset must be:

- In CSV or text format (or convertible to CSV)
- At least **1,000 rows**
- Containing at least **one categorical column** (to group by) and at least **one numerical column** (to aggregate)

Source	URL	Suggested Datasets
Kaggle	https://www.kaggle.com/datasets	Global Superstore Sales, World University Rankings, FIFA Players
Saudi Open Data	https://data.gov.sa	Health facilities, Education statistics, Traffic data
UCI ML Repository	https://archive.ics.uci.edu	Adult Census Income, Online Retail, Air Quality
data.gov	https://data.gov	US Baby Names, Airport Delays, Food Inspections
World Bank	https://data.worldbank.org	GDP by country, Population growth, Education indicators

If you're unsure what to choose, download the **Global Superstore Sales** dataset from Kaggle (∼10,000 rows). It has clear categorical columns (Country, Category, Segment) and numerical columns (Sales, Profit, Quantity) — perfect for MapReduce.

Step 2: Upload to HDFS

Download and Upload

```
# Download your dataset (example: from Kaggle after login)
# Move the downloaded file to your home directory
mv ~/Downloads/your_dataset.csv ~/dataset.csv

# Inspect the file
head -5 ~/dataset.csv          # See the first 5 lines
wc -l ~/dataset.csv            # Count total rows
du -h ~/dataset.csv             # Check file size

# Upload to HDFS
hdfs dfs -mkdir -p /user/student/lab3/exercise4_input
hdfs dfs -put ~/dataset.csv /user/student/lab3/exercise4_input/
```

Step 3: Design and Implement Two MapReduce Analyses

You must design and implement **two different MapReduce programs** on your chosen dataset. Each program must use a different aggregation type. Choose from:

Aggregation Type	Example Question
Sum	What is the total sales per country?
Count	How many records per category?
Average	What is the average score per department?
Max / Min	What is the highest temperature per city?
Collect / List	Which products belong to each category?

Your two analyses must use **different keys** and **different aggregation types**. For example, if Analysis 1 computes "total sales per country" (key=country, aggregation=sum), then Analysis 2 could be "average profit per category" (key=category, aggregation=average).

Step 4: Write a Processing Report

After implementing and running your MapReduce programs, write a **1–2 page report** that covers the following sections:

Exercise 4 – Report Structure

Your report must include the following sections:

1. **Dataset Description** (5 lines minimum)
   - Name of the dataset and source URL
   - Number of rows and columns
   - Description of each column (name, type, meaning)
   - Any data quality issues observed (missing values, inconsistent formats, etc.)

2. **Analysis Design** (for each of the 2 analyses)
   - The question you are answering
   - What you chose as the key and why
   - What you chose as the value and why
   - What aggregation type you used

3. **Implementation**
   - Full mapper and reducer code for both analyses
   - Any data cleaning steps performed in the mapper (e.g., skipping headers, handling missing values, type conversions)
   - The local test command and its output
   - The Hadoop execution command

4. Results & Discussion

- Output of both MapReduce programs (paste the actual results)
- A brief interpretation: What did you learn from the data? Were there any surprising findings?
- Manually verify at least 2 output rows by tracing through the original data

5. Challenges & Lessons Learned (5 lines minimum)

- What difficulties did you encounter? (e.g., encoding issues, delimiter problems, dirty data)
- How did you solve them?
- What would you do differently next time?

Exercise 4 – Deliverables Checklist

Submit the following items:

#	Item	Included?
1	Dataset file (or link if >10 MB)	☐
2	Screenshot: `head -5` and `wc -l` of your dataset	☐
3	Analysis 1: mapper.py + reducer.py code	☐
4	Analysis 1: local test output screenshot	☐
5	Analysis 1: Hadoop output (first 15 lines)	☐
6	Analysis 2: mapper.py + reducer.py code	☐
7	Analysis 2: local test output screenshot	☐
8	Analysis 2: Hadoop output (first 15 lines)	☐
9	Screenshot: YARN Web UI showing completed jobs	☐
10	Written report (1–2 pages covering all 5 sections above)	☐

In industry, no one gives you a clean 15-row CSV. Real datasets are messy, large, and undocumented. The skills you practice here — exploring unfamiliar data, handling edge cases in your mapper, verifying results manually, and communicating findings clearly — are exactly what employers look for in data engineers and data scientists. This exercise bridges the gap between **lab exercises** and **real-world data processing**.

End of Supplementary Sheet