**Code challenge 1**

users = [

(1, "Alice"),

(2, "Bob"),

(3, "Cathy")

]

orders = [

(1, 500, "2024-01-01"),

(2, 1500, "2024-01-01"),

(1, 1000, "2024-02-01"),

(3, 2000, "2024-01-01")

]

**Challenge**: Find users with total orders > 1000 and only include orders from January 2024.

Code challenge 2

data = [

("Electronics", "Laptop", 1000),

("Electronics", "Phone", 500),

("Furniture", "Chair", 150),

("Furniture", "Table", 300)

]

columns = ["category", "product", "revenue"]

**Challenge**: Rank products by revenue within each category using window functions.

Challenge 3

data = [

("u1", "email1", "2024-01-01"),

("u1", "email2", "2024-01-05"),

("u2", "email3", "2024-01-03")

]

columns = ["user", "email", "updated\_at"]

**Challenge**: Keep only the latest email record for each user.

Challenge 4

data = [

("u1", "2024-01-01", 100),

("u1", "2024-01-02", 200),

("u2", "2024-01-01", 300)

]

columns = ["user", "date", "points"]

**Challenge**: Compute running total of points for each user, ordered by date.

Challenge 5

data = [

("u1", "2024-01", 100),

("u1", "2024-02", 150),

("u2", "2024-01", 200)

]

columns = ["user", "month", "amount"]

**Challenge**: Pivot the data so each month becomes a column.

Challenge 6

data = [

("u1", "click", 1),

("u1", "view", 1),

("u2", "click", 1)

]

columns = ["user", "event\_type", "count"]

For each user, compute total clicks and total views in separate columns.

Challenge 7

data = [

(1, "Alice"),

(2, "Bob"),

(3, "Alice"),

(4, "David")

]

columns = ["id", "name"]

find the duplicate names

Challenge 8

data = [

("u1", "login"),

("u2", "login"),

("u1", "logout"),

("u3", "login"),

("u2", "logout"),

("u1", "login")

]

Find how many **unique users** performed each action using rdd

Challenge 9

transactions\_data = [

(1, 100),

(2, 50),

(1, 200),

(3, 300),

(2, 150)

]

users\_data = [

(1, "Alice"),

(2, "Bob"),

(3, "Charlie")

]

Join both RDDs and find **total amount spent per user**, with their names.

Challenge 10

users = {

1: "Alice",

2: "Bob",

3: "Charlie"

}

transactions = [

(1, 100),

(2, 50),

(1, 200),

(3, 300),

(2, 150)

]

Enrich each transaction with the user’s name using a **broadcast variable**.

Expected output

[('Alice', 100), ('Bob', 50), ('Alice', 200), ('Charlie', 300), ('Bob', 150)]

Challenge 11

You have a list of transactions. Count how many transactions are **invalid** (e.g., amount ≤ 0), using an **Accumulator** — but only collect the **valid** ones for further processing.

transactions = [

("u1", 100),

("u2", -50),

("u3", 0),

("u4", 200),

("u5", -10)

]

12 You have a dataset of user activities. Each activity has a list of items bought by the user. You need to **flatten** this dataset so that each item bought by a user appears as a separate row.

user\_activities = [

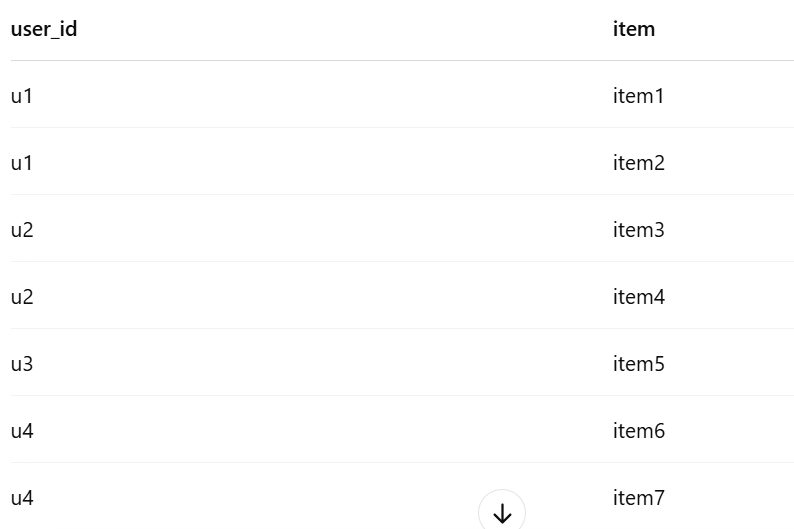
("u1", ["item1", "item2"]),

("u2", ["item3", "item4"]),

("u3", ["item5"]),

("u4", ["item6", "item7", "item8"])

]



13 You have a dataset representing the number of **page views** for different pages on a website. Your task is to **sum up the views per page** using reduceByKey.

page\_views = [

("home", 5),

("about", 2),

("home", 7),

("contact", 3),

("about", 4),

("home", 1)

]

14 Let's assume files are arriving in a directory with the following structure. Each file contains lines of text, and you need to count how many times the word PySpark appears in each file.

PySpark is awesome.

I love PySpark.

PySpark is fast.

15 You have a dataset of sales data with multiple sales representatives and their corresponding sales amounts. Your task is to **rank** the sales representatives based on their total sales, and assign a rank for each representative in descending order (highest sales get the rank 1).

data = [

("Alice", 100),

("Bob", 200),

("Alice", 300),

("Charlie", 250),

("Bob", 150),

("Charlie", 350)

]

columns = ["rep\_name", "sales\_amount"]