**FOR Loop**

○ **Iterate through a list and print elements.**

*fruits = ["Apple", "Banana", "Cherry"]*

*for fruit in fruits:*

*print(fruit)*

○ **Calculate the sum of numbers within a range.**

*total = sum(range(1, 11)) # Sum of numbers from 1 to 10*

*print("Sum:", total)*

○ **Generate a multiplication table using nested loops.**

*num = 5*

*for i in range(1, 11):*

*print(f"{num} x {i} = {num \* i}")*

**2. IF Condition**

**○ Check if a number is positive, negative, or zero.**

*num = int(input("Enter a number: "))*

*if num > 0:*

*print("Positive")*

*elif num < 0:*

*print("Negative")*

*else:*

*print("Zero")*

**○ Compare two numbers and determine the larger one.**

*a, b = 10, 20*

*print(f"Larger number: {max(a, b)}")*

**○ Categorize a score into grades using conditional statements.**

*score = int(input("Enter score: "))*

*if score >= 90:*

*print("Grade: A")*

*elif score >= 80:*

*print("Grade: B")*

*elif score >= 70:*

*print("Grade: C")*

*elif score >= 60:*

*print("Grade: D")*

*else:*

*print("Grade: F")*

**3. BREAK and PASS**

**○ Exit a loop upon encountering a specific value.**

*for num in range(1, 10):*

*if num == 5:*

*break*

*print(num)*

**○ Skip a condition within a loop while continuing execution.**

*for num in range(1, 10):*

*if num == 5:*

*pass # Placeholder, does nothing*

*else:*

*print(num)*

**○ Combine BREAK and PASS for advanced loop control.**

*for num in range(1, 10):*

*if num == 5:*

*print("Skipping 5")*

*pass*

*elif num == 7:*

*break*

*print(num)*

**4. Functions**

**○ Write a function to calculate the factorial of a number.**

*def factorial(n):*

*if n == 0:*

*return 1*

*return n \* factorial(n - 1)*

*print(factorial(5))*

**○ Reverse a string using a defined function.**

*def reverse\_string(s):*

*return s[::-1]*

*print(reverse\_string("Hello"))*

**○ Identify the maximum value in a list through a function.**

*def find\_max(lst):*

*return max(lst)*

*print(find\_max([3, 5, 9, 1, 6]))*

**5. Date in Databricks**

**○ Retrieve the current date.**

*from datetime import datetime*

*# Current Date*

*current\_date = datetime.now().date()*

*print("Current Date:", current\_date)*

**○ Extract components like day, month, and year from a date column.**

*import pandas as pd*

*# Sample DataFrame with Date column*

*data = {'date': ['2024-03-15', '2025-01-10', '2025-06-20']}*

*df = pd.DataFrame(data)*

*# Convert to datetime format*

*df['date'] = pd.to\_datetime(df['date'])*

*# Extract Year, Month, and Day*

*df['year'] = df['date'].dt.year*

*df['month'] = df['date'].dt.month*

*df['day'] = df['date'].dt.day*

*print(df)*

**○ Add days to a date column and create a new column.**

*# Adding 10 days to each date*

*df['new\_date'] = df['date'] + pd.to\_timedelta(10, unit='D')*

*print(df)*

**6. Filter**

**○ Filter rows in a DataFrame based on a single condition.**

*df\_filtered = df.filter(df["age"] > 30)*

*df\_filtered.show()*

**○ Apply multiple conditions to filter rows.**

*df\_filtered = df.filter((df["age"] > 30) & (df["salary"] > 50000))*

*df\_filtered.show()*

**○ Filter rows matching a specific string value.**

*df\_filtered = df.filter(df["city"] == "New York")*

*df\_filtered.show()*

**7. New Column**

**○ Create a column based on arithmetic operations.**

*df = df.withColumn("new\_salary", df["salary"] \* 1.1)*

**○ Categorize data into groups using conditional logic.**

*from pyspark.sql.functions import when*

*df = df.withColumn("category", when(df["age"] < 30, "Young").otherwise("Old"))*

**○ Add a column with transformed string values.**

*from pyspark.sql.functions import upper*

*df = df.withColumn("upper\_name", upper(df["name"]))*

**8. Literals ('DATAFLOW')**

**○ Create a DataFrame using a string literal.**

*from pyspark.sql import Row*

*data = [Row(name="DATAFLOW")]*

*df = spark.createDataFrame(data)*

*df.show()*

**○ Filter rows based on a specific string literal.**

*df\_filtered = df.filter(df["name"] == "DATAFLOW")*

*df\_filtered.show()*

**○ Insert a string literal as a constant value in a new column.**

*from pyspark.sql.functions import lit*

*df = df.withColumn("constant\_value", lit("DATAFLOW"))*

*df.show()*

**9. Joins**

**○ Perform inner, left, and full outer joins on DataFrames.**

*df1 = spark.createDataFrame([(1, "Alice"), (2, "Bob")], ["id", "name"])*

*df2 = spark.createDataFrame([(1, "HR"), (3, "Finance")], ["id", "dept"])*

*# Inner Join*

*df\_inner = df1.join(df2, "id", "inner")*

*# Left Join*

*df\_left = df1.join(df2, "id", "left")*

*# Full Outer Join*

*df\_full = df1.join(df2, "id", "full")*

*df\_inner.show()*

*df\_left.show()*

*df\_full.show()*

**○ Merge DataFrames while handling null values.**