1. It is similar to SQL case When, executes sequence of expressions until it matches the condition and returns a value when match.

Example-1:

from pyspark.sql import \*  
from pyspark.sql.functions import \*  
  
spark = SparkSession.builder.appName('when\_otherwise\_function').getOrCreate()  
  
myData = [(1,'Harsha','M',2000),  
 (2,'Mokshit','M',4000),  
 (3,'Harika','F',6000),  
 (4,'abcd','', 3000)]  
  
mySchema = ['id','name','gender','salary']  
  
df = spark.createDataFrame(myData,mySchema)  
  
df1 = df.select(df.id,\  
 df.name,\  
 when(df.gender=='M','Male').\  
 when(df.gender=='F','Female').\  
 otherwise('unknown').  
 alias('gender'))  
  
df1.show()

**Output:**

+---+-------+-------+

| id| name| gender|

+---+-------+-------+

| 1| Harsha| Male|

| 2|Mokshit| Male|

| 3| Harika| Female|

| 4| abcd|unknown|

+---+-------+-------+

*# Scenario 1: Categorizing Data  
# Let's say you have a DataFrame containing information about products and their prices.  
# You want to add a new column that categorizes each product based on its price range.*from pyspark.sql import SparkSession  
from pyspark.sql.functions import col, when  
  
spark = SparkSession.builder.appName("Example").getOrCreate()  
  
data = [(1, "Product A", 150),  
 (2, "Product B", 50),  
 (3, "Product C", 300),  
 (4, "Product D", 80)]  
  
columns = ["product\_id", "product\_name", "price"]  
df = spark.createDataFrame(data, columns)  
  
df\_categorized = df.withColumn("price\_category",\  
 when(col("price") < 100, "Low")\  
 .when(col("price") < 200, "Medium")\  
 .otherwise("High"))  
  
df\_categorized.show()

**Output:**

+----------+------------+-----+--------------+

|product\_id|product\_name|price|price\_category|

+----------+------------+-----+--------------+

| 1| Product A| 150| Medium|

| 2| Product B| 50| Low|

| 3| Product C| 300| High|

| 4| Product D| 80| Low|

+----------+------------+-----+--------------+

*# Scenario 2: Handling Missing Values  
# Suppose you have a DataFrame with sales data, and some of the rows have missing values in the "quantity\_sold" column. You want to replace those missing values with 0.*from pyspark.sql import SparkSession  
from pyspark.sql.functions import col, when  
  
spark = SparkSession.builder.appName("Example").getOrCreate()  
  
data = [(1, "Product A", 100),  
 (2, "Product B", None),  
 (3, "Product C", 50),  
 (4, "Product D", None)]  
  
columns = ["product\_id", "product\_name", "quantity\_sold"]  
df = spark.createDataFrame(data, columns)  
  
df\_filled = df.withColumn('quantity\_sold',\  
 when(col('quantity\_sold')\  
 .isNull(),0)\  
 .otherwise(col('quantity\_sold')))  
  
df\_filled.show()

**Output:**

+----------+------------+-------------+

|product\_id|product\_name|quantity\_sold|

+----------+------------+-------------+

| 1| Product A| 100|

| 2| Product B| 0|

| 3| Product C| 50|

| 4| Product D| 0|

+----------+------------+-------------+

*# Scenario 3: Calculating Discounts  
# Imagine you have a DataFrame containing information about orders and their total amounts.  
# You want to calculate a discounted amount for each order based on the order's total amount and a discount rate.*from pyspark.sql import SparkSession  
from pyspark.sql.functions import col, when  
  
spark = SparkSession.builder.appName("Example").getOrCreate()  
  
data = [(1, "Order A", 200),  
 (2, "Order B", 500),  
 (3, "Order C", 1000),  
 (4, "Order D", 150)]  
  
columns = ["order\_id", "order\_name", "total\_amount"]  
df = spark.createDataFrame(data, columns)  
  
discount\_rate = 0.1 *# 10% discount*df\_discounted = df.select(col("order\_id"),  
 col("order\_name"),  
 col("total\_amount"),  
 (col("total\_amount") - when(col("total\_amount") > 500, col("total\_amount") \* discount\_rate).otherwise(0)).alias("discounted\_amount")  
 )  
  
df\_discounted.show()

**Output:**

+--------+----------+------------+-----------------+

|order\_id|order\_name|total\_amount|discounted\_amount|

+--------+----------+------------+-----------------+

| 1| Order A| 200| 200.0|

| 2| Order B| 500| 500.0|

| 3| Order C| 1000| 900.0|

| 4| Order D| 150| 150.0|

+--------+----------+------------+-----------------+