06 lambda , map and reduce functions (python)

lambda

A **lambda function** is a **small, anonymous function** created without using `def`.

It's used for **short, one-line operations**.

✔️ Syntax

```
lambda arguments: expression
```

✔️ Example

```
square = lambda x: x * 2
print(square(5))  # 10
```

Why are Lambda Functions used?

- For quick one-line operations
- When defining a full function using `def` is unnecessary
- Used inside **map, filter, reduce**
- Used heavily in **Spark RDDs and DataFrames**
- Functional style programming

Lambda vs Normal Function

Normal function:

```
def add_nums(x, y):
    return x + y
```

Lambda equivalent:

```
add_nums = lambda x, y: x + y
```

map and reduce

These are **higher-order functions** that come from functional programming.

They operate on sequences (lists, tuples, etc.) and help process data without writing loops.

1- map()

✔️ Purpose

Applies a function to every element in an iterable.

✔️ Syntax

```
map(function, iterable)
```

✔️ Example

```
nums = [1, 2, 3]
result = list(map(lambda x: x * 2, nums))  # [2, 4, 6]
```

✔️ Internal Working

```
[ f(1), f(2), f(3), ...
```

✔️ Key Point

map does **element-wise transformation**. No aggregation.

For N input elements it will give N output elements

2- reduce()

`reduce()` is not built-in. It comes from:

```
from functools import reduce
```

✔️ Purpose

Combines elements one-by-one into a **single value**.

✔️ Syntax

```
reduce(function, iterable)
```

✔️ Example

```
from functools import reduce
nums = [1, 2, 3, 4]

result = reduce(lambda x, y: x + y, nums)  # 10
```

✔️ Internal Working

```
(((1 + 2) + 3) + 4)
```

✔️ Key Point

reduce is used for **aggregation**, like sum, max, product, etc. it produces single value output.

split function

```
orders_row = '1,technology,200,2025-01-01'
orders_row.split(",")
```