Low-Level Design (LLD) – E-Commerce Sales Analysis
Difficulty Level: Medium | **Total Marks:** 15
Standards Followed: 6 Functions | 5 Visible Test Cases | 3 Hidden Test Cases

——

□ **Summary of Corrections** (Based on SME Feedback)

- □ All operations encapsulated as free-standing functions (no classes)
- □ Strict separation: each function does exactly one thing, returns data for testing
- □ Used NumPy (`np.array`, `np.unique`, `np.sum`, `np.mean`, `np.argmax`, etc.)
- □ No side-effects—input arrays never mutated
- □ Return types and shapes match driver's expectations

□ **Concepts Tested**

- □ NumPy array creation & validation
- □ Built-in aggregations (`sum`, `mean`, `max`, `min`)
- □ Conditional labeling & clipping
- □ Iteration and boolean masking
- □ Array formatting (string conversion)

□ **Problem Statement**
Build a set of utility functions for analyzing daily sales figures stored in NumPy arrays. You must implement:

1. **create_sales_array** — convert list → `np.ndarray`
2. **validate_sales_array** — ensure all values ≥0, non-empty
3. **compute_sales_metrics** — total, average, maximum
4. **categorize_demand_levels** — label each day "Low"/"Moderate"/"High" demand
5. **longest_growth_streak** — longest strictly increasing run
6. **format_sales_data** — convert numbers to comma-formatted strings

All functions should return new data (never print), matching exactly the types and shapes the `driver.py` tests expect.

OPERATIONS (Structured Format)

□ Operation 1: `create_sales_array`

- □ **Purpose**:
  Convert a Python list of daily sales into a NumPy array.
- □ **Input**:
  - `sales_data`: A list of integers or floats representing sales.
- □ **Output**:
  - A NumPy array containing the same values.
- □ **Logic**:
  1. Accept the `sales_data` list.
  2. Use `np.array()` to convert it into a NumPy array.
  3. Return the resulting array.
- □ **Example**:

```
create_sales_array([150, 220, 90, 300]) → array([150, 220,  90, 300])
```

□ Operation 2: `validate_sales_array`

- □ **Purpose**:
  Check if the array is valid—non-empty, numeric, and non-negative.
- □ **Input**:
  - `sales_array`: A NumPy array.
- □ **Output**:
  - `True` if valid, `False` otherwise.
- □ **Logic**:
  1. Check that the array is not empty using `.size`.
  2. Confirm all values are numeric types.
  3. Ensure all sales are ≥ 0 using `np.all()`.
  4. Return the final Boolean result.
- □ **Example**:

```
validate_sales_array(np.array([100, 200, -50])) → False
```

□ Operation 3: `compute_sales_metrics`

- □ **Purpose**:
  Calculate the total, average (rounded), and highest sale.
- □ **Input**:
  - `sales_array`: A validated NumPy array.
- □ **Output**:
  - A tuple: (`total`, `average`, `maximum`) as `(int/float, float, int/float).`
- □ **Logic**:
  1. Compute total sales using `.sum()`.
  2. Calculate average using `.mean()` and round to 1 decimal.
  3. Find maximum sale using `.max()`.
  4. Return the 3 values as a tuple.
- □ **Example**:

```
compute_sales_metrics(np.array([150, 200, 250])) ➞ (600, 200.0, 250)
```

□ Operation 4: `categorize_demand_levels`

- □ **Purpose**:
  Label each day's demand as "Low", "Moderate", or "High".
- □ **Input**:
  - `sales_array`: A NumPy array of sales data.
- □ **Output**:
  - A NumPy array of strings with labels for each day.
- □ **Logic**:
  1. Initialize an empty list `labels`.
  2. For each sale value:
     - `<100` → "Low Demand"
     - `100-250` → "Moderate Demand"
     - `>250` → "High Demand"
  3. Append the corresponding label to `labels`.
  4. Convert `labels` to a NumPy array and return.
- □ **Example**:

```
categorize_demand_levels(np.array([50, 150, 300])) ➞ array(['Low
Demand', 'Moderate Demand', 'High Demand'])
```

□ Operation 5: `longest_growth_streak`

- □ **Purpose**:
  Find the length of the longest strictly increasing streak.
- □ **Input**:
  - `sales_array`: A NumPy array of sales data.
- □ **Output**:
  - Integer representing the maximum streak length.
- □ **Logic**:
  1. Initialize `max_streak = 1`, `current_streak = 1`.
  2. Iterate through the array starting from index 1.
  3. If current value > previous → increment `current_streak`.
  4. Else → reset `current_streak` to 1.
  5. Update `max_streak` accordingly and return it.
- □ **Example**:

```
longest_growth_streak(np.array([100, 120, 140, 130, 150])) ➞ 3
```

□ Operation 6: `format_sales_data`

- □ **Purpose**:
  Format each sales number into a human-readable string with commas.
- □ **Input**:
  - `sales_array`: A NumPy array of integers or floats.
- □ **Output**:
  - A NumPy array of strings with comma separators.
- □ **Logic**:
  1. Initialize an empty list `formatted`.
  2. For each number in the array:
     - Format using `f"{num:,}"`
  3. Store each formatted string in the list.
  4. Convert and return as a NumPy array.
- □ **Example**:

```
format_sales_data(np.array([1000, 24500])) ➞ array(['1,000',
'24,500'])
```

□ Test Cases & Marks Allocation

Test Case ID	Description	Function	Marks
TC1	Create sales array	`create_sales_array()`	2.5
TC2	Validate with negative values	`validate_sales_array()`	2.5
TC3	Compute total, average, max	`compute_sales_metrics()`	2.5
TC4	Categorize demand levels	`categorize_demand_levels()`	2.5
TC5	Longest growth streak	`longest_growth_streak()`	2.5
HTC1	Format with commas	`format_sales_data()`	2.5
HTC2	100% high demand boundary (e.g. 250→High)	`categorize_demand_levels()`	2.5
HTC3	Empty array validation in `validate_sales_array()`	`validate_sales_array()`	2.5
TOTAL			**20**

□ Visible Test Cases (5)

1. TC1:

```
create_sales_array([150,220,90,300,175])  # → array([150,220, 90,300,175])
```

2. TC2:

```
validate_sales_array(np.array([150,220,-5]))  # → False
```

3. TC3:

```
compute_sales_metrics(np.array([150,220,90,300,175]))  # → (935,187.0,300)
```

4. TC4:

```
categorize_demand_levels(np.array([99,150,275]))  # → ['Low','Moderate','High']
```

5. TC5:

```
longest_growth_streak(np.array([100,120,140,130,150,160,170,140,145,150,155]))  # → 4
```

□ Hidden Test Cases (3)

- HTC1:

  ```
  format_sales_data(np.array([1000,24500]))  # → ['1,000','24,500']
  ```

- HTC2:

  ```
  categorize_demand_levels(np.array([100,250]))  # →
  ['Moderate','Moderate']
  ```

- HTC3:

  ```
  validate_sales_array(np.array([]))  # → False
  ```
