16. Mixed Delimiters in a Single Line

- **Description**: A single line with different delimiters separating data segments.
- **Example**: `5;10,15/20:25`
- **Explanation**: Each segment is separated by different delimiters (`;`, `,`, `/`, `:`). You would need to handle each delimiter to split and process the values correctly.

17. Nested Input with Multiple Levels

- **Description**: Input includes multiple levels of nested lists or arrays.
- **Example**: `[[1, [2, 3]], [4, [5, 6, [7, 8]]]]`
- **Explanation**: This nested list format requires handling multiple levels of lists or arrays. You may need recursive parsing to access inner elements.

18. Key-Value Pairs (Dictionary-Like Input)

- **Description**: Each input line contains a key-value pair.
- **Example**: `Name: John        Age: 25        Score: 89`
- **Explanation**: The input represents a dictionary format, where each line is a key-value pair. You need to parse and store these pairs, often useful in JSON-like data structures.

19. Bracketed Expressions in a Single Line

- **Description**: Values enclosed in different types of brackets (e.g., round `()`, square `[]`, curly `{}`).
- **Example**: `(1, 2), [3, 4], {5, 6}`
- **Explanation**: Each segment uses a different type of bracket, often representing different data types or levels. Parsing involves recognizing and separating these types.

20. Complex Delimited Structure

- **Description**: Each data point has both a prefix and a delimiter.
- **Example**: `A-1 B-2 C-3`
- **Explanation**: Each data point (`A`, `B`, `C`) has a prefix and a corresponding value separated by a dash. You need to parse both prefixes and values separately.

21. Single Line with Condition-Specific Delimiters

- **Description**: A delimiter varies based on the value type.
- **Example**: `5.2#3,4&8.9*7`

- **Explanation**: Here, decimal numbers use #, integers use &, and float integers use `*`. The parsing approach varies based on the delimiter type.

22. Patterned Matrix Input with Mixed Delimiters

- **Description**: Matrix input where each row is separated by newlines and each column by a different delimiter.
- **Example**: `1,2;3      4,5;6`
- **Explanation**: This matrix format uses commas and semicolons to separate elements. Parsing requires handling both types of delimiters per row.

23. Input with Headers and Subsections

- **Description**: Data is grouped under specific headers with subsections.
- **Example**: `Students:      Name: Alice, Age: 20      Name: Bob, Age: 22      Courses:      Math: 90      Science: 85`
- **Explanation**: This format mimics structured documents where headers (`Students`, `Courses`) separate data groups.

24. Range Specification in a Single Line

- **Description**: A range is specified with start and end values, sometimes with a step.
- **Example**: `1-10 by 2`
- **Explanation**: This input represents a range from 1 to 10 with a step of 2. Parsing should generate numbers `[1, 3, 5, 7, 9]` based on the range and step.

25. Multidimensional Array with Custom Delimiters

- **Description**: Multidimensional arrays with unique delimiters per dimension.
- **Example**: `[1,2;3,4];[5,6;7,8]`
- **Explanation**: Here, `;` separates rows and `,` separates elements within rows. Parsing requires handling nested delimiters correctly.

26. Time-Series Data with Timestamps

- **Description**: Each data point has an associated timestamp.
- **Example**: `12:00-5 12:30-8 13:00-10`
- **Explanation**: The input combines time and values. The task involves parsing timestamps with corresponding data for operations like time-based aggregation.

27. Sparse Matrix Representation

- **Description**: Only non-zero elements with positions are listed.

- **Example**: `1 1 5        2 3 10`
- **Explanation**: Each row represents a position (row, column) and value. This format is common for efficient representation of sparse data.

28. Encoded String with Special Symbols

- **Description**: Encoded text where each symbol represents a certain value.
- **Example**: `@#% 2 $!`
- **Explanation**: The format uses symbols to denote certain values or types, such as `@` for uppercase, # for numeric. Decoding is required to interpret the symbols.

29. Boolean Expressions

- **Description**: Each line contains a Boolean expression in text.
- **Example**: `(A AND B) OR (C AND NOT D)`
- **Explanation**: Parsing involves interpreting Boolean expressions, evaluating conditions, or building a logical tree.

30. Tab-Separated Multiple Column Data

- **Description**: Each value is separated by tabs.
- **Example**: `1\t2\t3\t4`
- **Explanation**: Values are separated by a tab character. Parsing involves handling tabs rather than spaces, common in spreadsheet-like data.

31. Graph Edges in a Line (Adjacency List)

- **Description**: Graph input in edge-list format.
- **Example**: `1->2, 1->3, 2->4`
- **Explanation**: Each edge is listed as `node1->node2`. This format is useful for constructing graphs using adjacency lists.

32. Input with Comments or Annotations

- **Description**: Comments or irrelevant information are present and need to be ignored.
- **Example**: `# Input starts here        1,2,3       # End of input`
- **Explanation**: The # symbol indicates comments that should be ignored. Parsing involves discarding these lines.

33. Keyed Multi-List with Group Indicators

- **Description**: Data is presented in groups identified by a key.
- **Example**: `Group1: 1,2,3        Group2: 4,5,6`
- **Explanation**: Each group has a label, and elements are grouped by this label. Parsing involves separating groups and processing them independently.

34. JSON-Like Data in Plain Text

- **Description**: Structured data resembling JSON but in plain text format.
- **Example**: `{"name": "Alice", "age": 25, "scores": [85, 90, 95]}`
- **Explanation**: The data is in JSON-like structure but may not use true JSON encoding, requiring manual parsing.

35. Fixed-Length Substring Data

- **Description**: A long string divided into fixed-length substrings.
- **Example**: `1234567890`
- **Explanation**: Every two characters represent a distinct piece of data (`12`, `34`, `56`, etc.), commonly used in binary or hexadecimal parsing.

36. Pairwise Key-Value with Delimiter

- **Description**: Each key-value pair is separated by `=`.
- **Example**: `a=1 b=2 c=3`
- **Explanation**: This structure uses `=` to assign values to keys, resembling environment variable syntax.