Take-Home Assignment

NLP → DSL → Strategy Execution Prototype

Overview

This assignment evaluates your ability to design a small domain-specific language (DSL), parse natural-language instructions, generate executable Python logic, and run a simplified time-series simulation.

You will build a **mini NL → DSL → AST → Code → Execution pipeline** involving stock-style rules (price, volume, indicators like SMA/RSI, etc.).

This is a research-engineering task focused on language design, parsing, and code generation.

Deliverables

You must submit:

1. A short **DSL design document**
2. Code for **NL → structured representation**
3. Code for **DSL parser + AST builder**
4. Code for **AST → Python strategy generator**
5. A **simple backtest-style simulator**
6. An **end-to-end demo script** showing input → DSL → AST → execution → results
7. A README with instructions to run your solution

All code must be in **Python**.

Part 1 — Natural-Language Parsing

Your system must accept rules written in simple natural language, for example:

- “Buy when the close price is above the 20-day moving average and volume is above 1 million.”
- “Enter when price crosses above yesterday’s high.”
- “Exit when RSI(14) is below 30.”
- “Trigger entry when volume increases by more than 30 percent compared to last week.”

You do **not** need a full NLP engine; a combination of regex, prompt-based LLM usage, or your own logic is acceptable.

Your output must be a **structured JSON representation**, such as:

```
{
  "entry": [
    {
      "left": "close",
      "operator": ">",
      "right": "sma(close,20)"
    },
    {
      "left": "volume",
      "operator": ">",
      "right": 1000000
    }
  ],
  "exit": [
    {
      "left": "rsi(close,14)",
      "operator": "<",
      "right": 30
    }
  ]
}
```

The structure is up to you; consistency and clarity are key.

Part 2 — DSL Design

Design a small DSL capable of expressing:

- Entry rules
- Exit rules
- Boolean logic (AND / OR)
- Comparisons (> < >= <= ==)
- Time-based lookbacks (yesterday, N-day window, last week)
- Indicators such as SMA, RSI
- Cross events (“price crosses above X”)

Example DSL format (you may design your own):

```
ENTRY:
  close > SMA(close,20) AND volume > 1000000

EXIT:
  RSI(close,14) < 30
```

Or a more structured variant:

```
ENTRY:
  RULE1: close > sma(close,20)
  RULE2: volume > 1000000

EXIT:
  RULE1: rsi(close,14) < 30
```

The DSL must:

- Have a consistent grammar
- Validate field/indicator names
- Support nested logic (parentheses)

- Represent strategy rules unambiguously

Provide a **written DSL specification** describing:

- Grammar
- Operator support
- Indicator syntax
- Examples
- Any assumptions you made

Part 3 — DSL Parser + AST Construction

Write a parser (Lark, PLY, ANTLR, or your custom tokenizer) that:

1. Takes DSL text as input
2. Validates the syntax
3. Builds an **AST (Abstract Syntax Tree)** or equivalent structured format
4. Raises informative errors for invalid syntax

Example AST (illustrative):

```
{
  "entry": [
    {
      "type": "binary_op",
      "left": {"type": "series", "value": "close"},
      "op": ">",
      "right": {"type": "indicator", "name": "sma", "params":
["close", 20]}
    }
  ],
  "exit": [...]
}
```

Part 4 — Code Generator (AST → Python)

Implement a code generator that converts the AST into a Python function that evaluates entry/exit conditions over a pandas DataFrame containing OHLCV data.

For example, convert the AST into logic such as:

```
signals['entry'] = (df['close'] > df['close'].rolling(20).mean()) &
(df['volume'] > 1000000)
signals['exit'] = (df['rsi_14'] < 30)
```

You may implement indicators such as SMA/RSI manually or using simple helper functions.

Your generated function must:

- Evaluate the rules row-by-row
- Return a list or DataFrame of entry/exit boolean values

Part 5 — Simple Backtest-Style Execution

Using the signals produced by your code:

Implement a basic simulator:

- Start with no position
- Enter when the `entry` condition becomes true
- Exit when the `exit` condition becomes true
- Record:

- Entry date
- Exit date
- Entry price / exit price
- Profit/loss
- Total return
- Max drawdown
- Number of trades

You may use synthetic OHLCV data or the example dataset below.

Example dataset:

```
date,open,high,low,close,volume
2023-01-01,100,105,99,103,900000
2023-01-02,103,108,101,107,1200000
2023-01-03,107,110,106,109,1300000
...
```

Part 6 — End-to-End Demonstration

Your final script must:

1. Accept natural language input
2. Generate DSL text
3. Parse DSL into AST
4. Convert AST into Python code
5. Execute the generated code over sample data
6. Run the backtest simulator
7. Print a final report such as:

```
Natural Language Input:
"Buy when price closes above the 20-day moving average and volume is
above 1M."

Generated DSL:
ENTRY: close > sma(close,20) AND volume > 1000000
EXIT: rsi(close,14) < 30

Parsed AST: {...}

Backtest Result:
Total Return: 12.3%
Max Drawdown: -3.1%
Trades: 4
Entry/Exit Log:
 - Enter: 2023-01-02 at 107
 - Exit: 2023-01-10 at 115
...
```

Expectations & Evaluation Criteria

Each section will be evaluated on:

1. DSL Design Quality

- Clarity, structure, future extensibility
- Thoughtfulness of grammar choices

2. Parsing & AST Construction

- Accuracy of parsing
- Handling Boolean logic
- Error handling

3. NL → DSL Reasoning

- How well natural language is mapped into structured rules
- Consistency and robustness

4. Code Generation

- Clean, correct Python code generation
- Indicator correctness
- Readable logic

5. Backtest Execution

- Correct implementation of entry/exit lifecycle
- Correct calculations of PnL, Drawdown, etc.

6. Architecture & Modularity

- Separation of parser, generator, execution
- Proper file structure

7. Documentation

- Clear DSL description
- Explanation of design choices
- Examples

8. Overall Completeness

- Does the end-to-end pipeline run successfully?

Submission Guidelines

Please submit:

- A GitHub/GitLab repo link or ZIP file
- README explaining how to run tests and demo
- A short document describing your DSL grammar with examples

Notes

- You do not need to support every possible English phrasing; support the examples and similar patterns.
- You may use open-source libraries (Lark, PLY, TaLib, Pandas).
- Focus on correctness, clarity, and thoughtful design, not production optimization.
