Context-Free Languages & Parsing Assignment

Introduction

This assignment will test your understanding of Context-Free Languages (CFLs) and your ability to implement parsers for them using Python. Unlike Regular Expressions, CFLs often require **memory** (stacks) or **recursion** to handle nested structures.

Each question will have a set of test cases, and your goal is to write the correct logic to pass all provided test cases.

Submission Instructions

- **File Naming:** Name your Python file with your roll number, e.g., 2321cs12.py.
- **Function Naming:** For each question, you should define a function with the exact signature as provided in the problem statement.
- Each function should return True if the input matches the language/grammar pattern and False otherwise.
- **Submission:** Submit your Python file (.py) containing all the required functions.

Questions

Question 1: Balanced Parentheses (The Stack)

Context-Free Grammars are essential for defining nested structures. Write a function is_balanced (s: str) -> bool that checks if a string containing parentheses (), brackets [], and braces {} is balanced.

- A string is balanced if every opening bracket has a corresponding closing bracket of the same type, properly nested.
- The string may contain other characters (letters, numbers), which should be ignored.

Function Signature:

```
def is_balanced(s: str) -> bool:
    pass
```

Test Cases:

- "{[()]}" → True

- "def foo(): return [x]" → True
- "{[(])}" → False (Mismatched nesting)
- "((()))" → False (Unclosed)

Question 2: Valid Arithmetic Expression (Recursive Descent)

The presentation defined expressions like E -> E+E | E*E | (E) | id . Write a function is_valid_expr(s: str) -> bool to validate a simple arithmetic expression.

- Allowed characters: Lowercase letters (variables), digits, +, *, (,).
- Rules:
  - Variables/numbers are operands.
  - Operators + and * must be between operands.
  - Parentheses must be balanced and contain a valid expression.
  - Example of invalid: a+, (*b), a**b.

Function Signature:

```
def is_valid_expr(s: str) -> bool:
    pass
```

Test Cases:

- "(a+b)*c" → True
- "a+(b*3)" → True
- "a+*b" → False
- "(a+b" → False
- "a" → True

Question 3: Palindromes (Recursive Structure)

Palindromes are a classic Context-Free Language S -> xSx | x | Ɛ . Write a function is_palindrome(s: str) -> bool that checks if the input string is a palindrome.

- The check should be case-sensitive.
- You must treat the string literally (do not ignore whitespace for this specific problem).
- Hint: This models the production S-> aSa | bSb …

Function Signature:

```
def is_palindrome(s: str) -> bool:
```

```
    pass
```

Test Cases:

- "radar" → True
- "abba" → True
- "step on no pets" → True
- "hello" → False
- "Racecar" → False (Case sensitive)