Lab 7 - Code Runner - Report

Students:

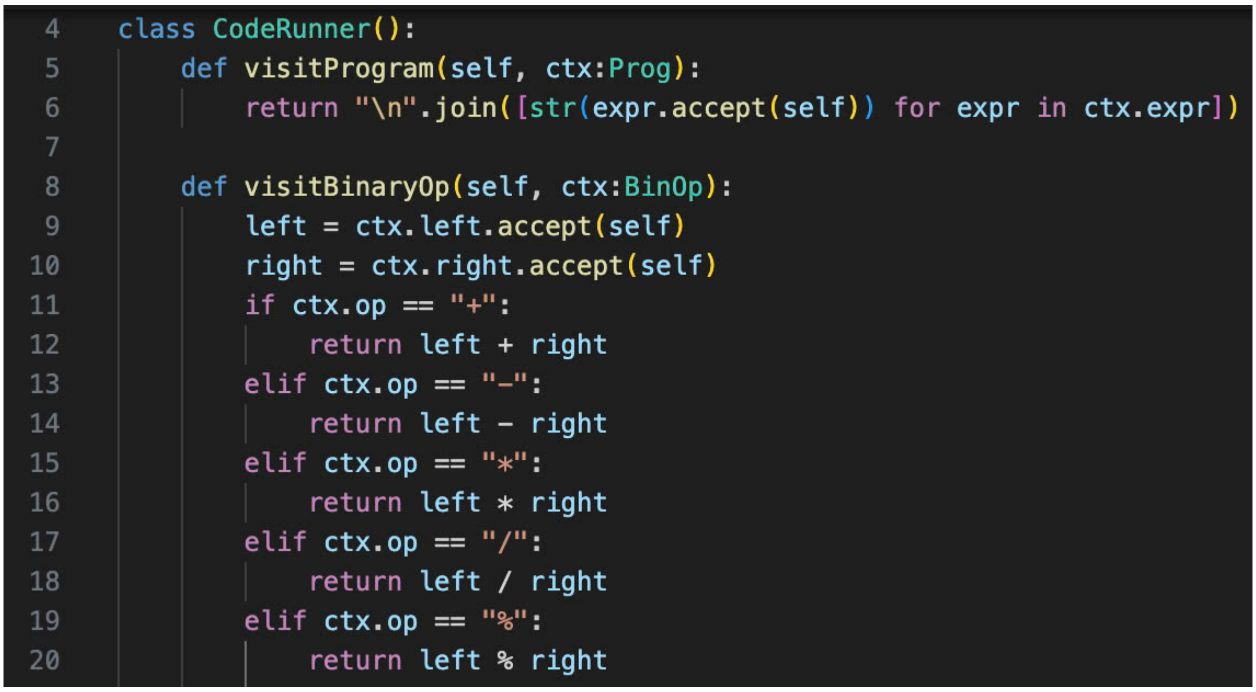
1. Nguyen Huynh Thao My - ITCSIU21204
2. Pham Duc Dat - ITITIU20184

1 Overview

2 Results

3 Exercise: Write code in "CodeRunner.py" to execute programs with the following grammars.

3.1 Exercise 1: Extend the given Sample code to perform a program that can do five operations: Add, Sub, Mul, Div, Mod



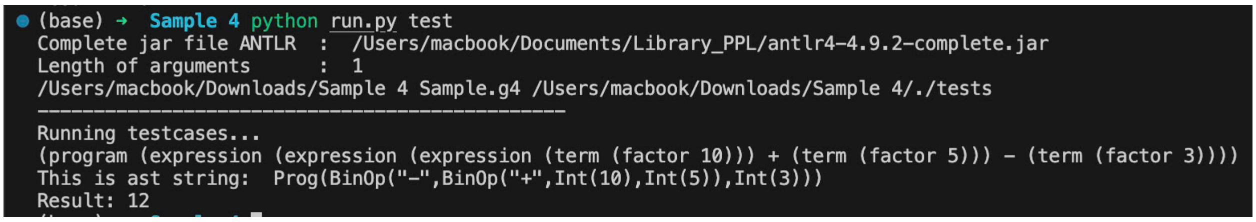
```python
class CodeRunner():
    def visitProgram(self, ctx:Prog):
        return "\n".join([str(expr.accept(self)) for expr in ctx.expr])

    def visitBinaryOp(self, ctx:BinOp):
        left = ctx.left.accept(self)
        right = ctx.right.accept(self)
        if ctx.op == "+":
            return left + right
        elif ctx.op == "-":
            return left - right
        elif ctx.op == "*":
            return left * right
        elif ctx.op == "/":
            return left / right
        elif ctx.op == "%":
            return left % right
```

Figure 1: Code runner for Ex1

3.1.1 Input 1: `10 + 5 - 3`

Explanation:

In this example, the input expression `10 + 5 - 3` is correctly parsed, and the code run-



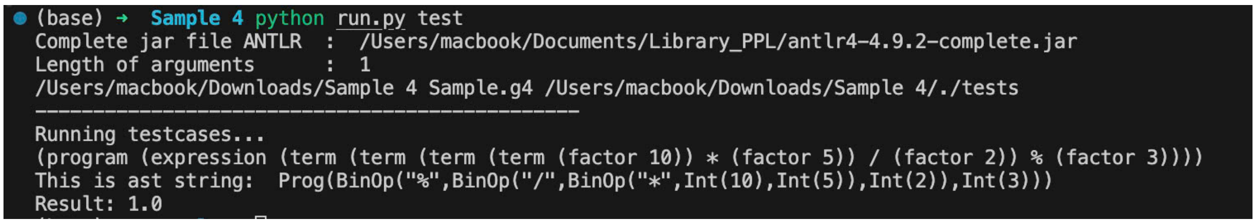
```
(base) → Sample 4 python run.py test
Complete jar file ANTLR  :  /Users/macbook/Documents/Library_PPL/antlr4-4.9.2-complete.jar
Length of arguments      :  1
/Users/macbook/Downloads/Sample 4 Sample.g4 /Users/macbook/Downloads/Sample 4/./tests
----------------------------------------------
Running testcases...
(program (expression (expression (term (factor 10))) + (term (factor 5))) - (term (factor 3))))
This is ast string:  Prog(BinOp("-",BinOp("+",Int(10),Int(5)),Int(3)))
Result: 12
```

Figure 2: Output for Input 1: `10 + 5 - 3`

ner evaluates it as follows:

- The expression is parsed into an Abstract Syntax Tree (AST) with two binary operations: addition and subtraction.
- The tree is evaluated left-to-right, first adding 10 and 5 to get 15, and then subtracting 3 to get the final result of 12.

3.1.2 Input 2: `10 * 5 / 2 % 3`



```
(base) → Sample 4 python run.py test
Complete jar file ANTLR  :  /Users/macbook/Documents/Library_PPL/antlr4-4.9.2-complete.jar
Length of arguments      :  1
/Users/macbook/Downloads/Sample 4 Sample.g4 /Users/macbook/Downloads/Sample 4/./tests
----------------------------------------------
Running testcases...
(program (expression (term (term (term (term (factor 10)) * (factor 5)) / (factor 2)) % (factor 3))))
This is ast string:  Prog(BinOp("%",BinOp("/",BinOp("*",Int(10),Int(5)),Int(2)),Int(3)))
Result: 1.0
```

Figure 3: Output for Input 2: `10 * 5 / 2 % 3`

Explanation:

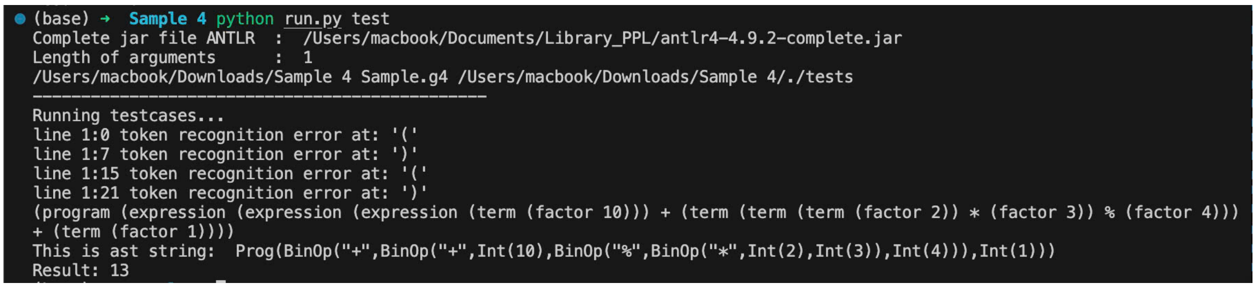
In this case, the input expression `10 * 5 / 2 % 3` involves multiplication, division, and modulus operations. The code runner processes the input as follows:

- The expression is parsed into an AST with three operations: multiplication, division, and modulus.
- The operations are evaluated left-to-right: 10 multiplied by 5 equals 50, 50 divided by 2 equals 25, and 25 modulus 3 gives a final result of 1.

3.1.3 Input 3: `(10 + 2) * 3 % (4 + 1)`

Explanation:

The input expression `(10 + 2) * 3 % (4 + 1)` is more complex, involving parentheses to group operations. The code runner handles this as follows:



```
● (base) ➜  Sample 4 python run.py test
  Complete jar file ANTLR  :  /Users/macbook/Documents/Library_PPL/antlr4-4.9.2-complete.jar
  Length of arguments      :  1
  /Users/macbook/Downloads/Sample 4 Sample.g4 /Users/macbook/Downloads/Sample 4/./tests
  ----------------------------------------------
  Running testcases...
  line 1:0 token recognition error at: '('
  line 1:7 token recognition error at: ')'
  line 1:15 token recognition error at: '('
  line 1:21 token recognition error at: ')'
  (program (expression (expression (expression (term (factor 10))) + (term (term (term (factor 2)) * (factor 3)) % (factor 4)))
  + (term (factor 1))))
  This is ast string:  Prog(BinOp("+",BinOp("+",Int(10),BinOp("%",BinOp("*",Int(2),Int(3)),Int(4))),Int(1)))
  Result: 13
```

Figure 4: Output for Input 3: `(10 + 2) * 3 % (4 + 1)`

- The expression is parsed into an AST with operations grouped according to the parentheses.
- The addition operations inside the parentheses are evaluated first: 10 + 2 equals 12, and 4 + 1 equals 5.
- The multiplication and modulus operations are then performed: 12 multiplied by 3 equals 36, and 36 modulus 5 gives a final result of 1.
- Note: The code runner may encounter issues when handling parentheses in certain cases, leading to potential errors.

3.2 Exercise 2: Extend exercise 1 to perform a program that can perform exponential function (e.g., a^n)

Input 1: `2 ^3`

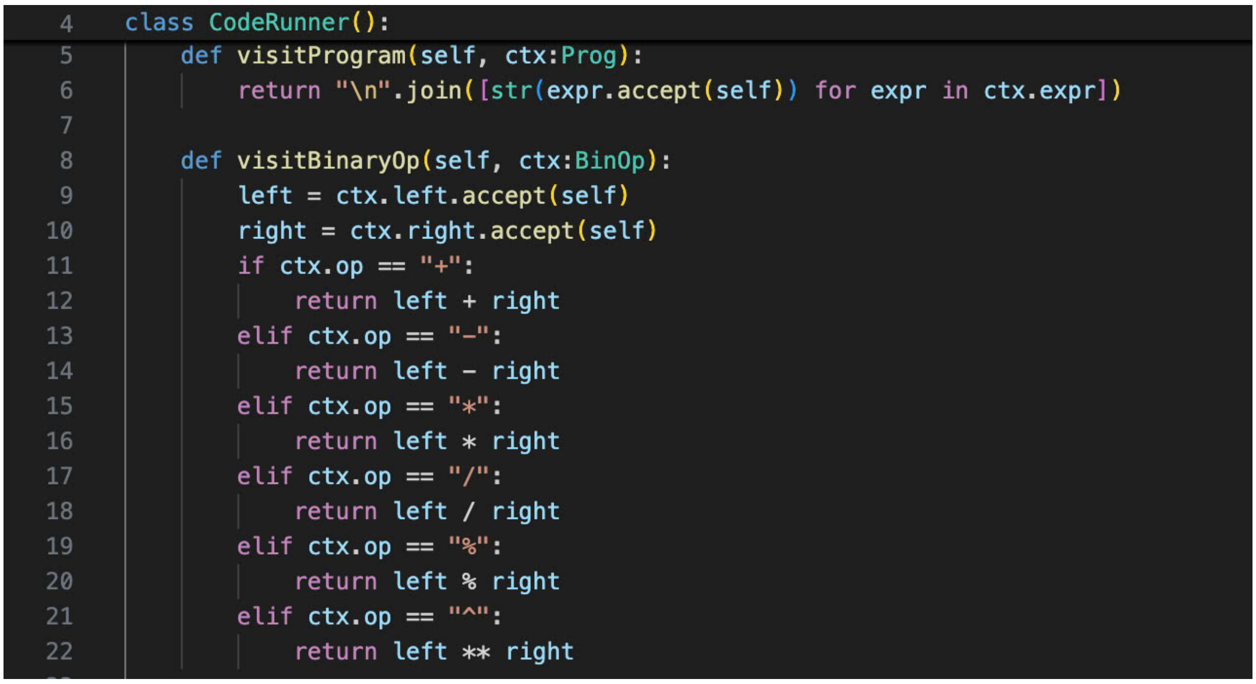
Output: Explanation: The input `2 ^3` is parsed and the exponential operation is performed, resulting in `8`.

Input 2: `2 ^3 + 4`

Output: Explanation: The input `2 ^3 + 4` is parsed, where the exponential operation is performed first, followed by the addition, resulting in `12`.

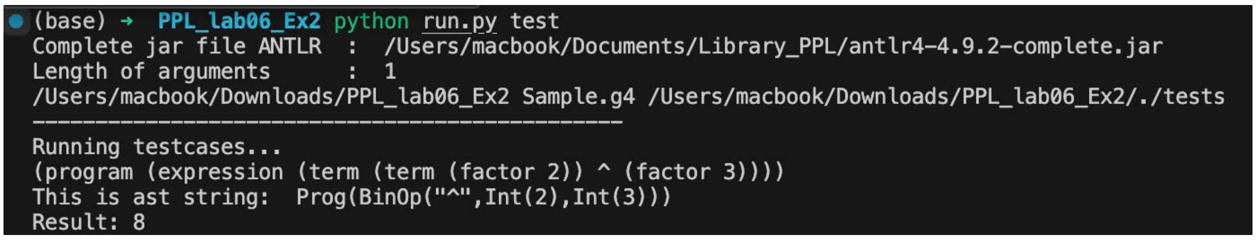
Input 3: `2 ^4 * 3`

Output: Explanation: The input `2 ^4 * 3` is parsed, where the exponential operation is performed first, followed by the multiplication, resulting in `48`.



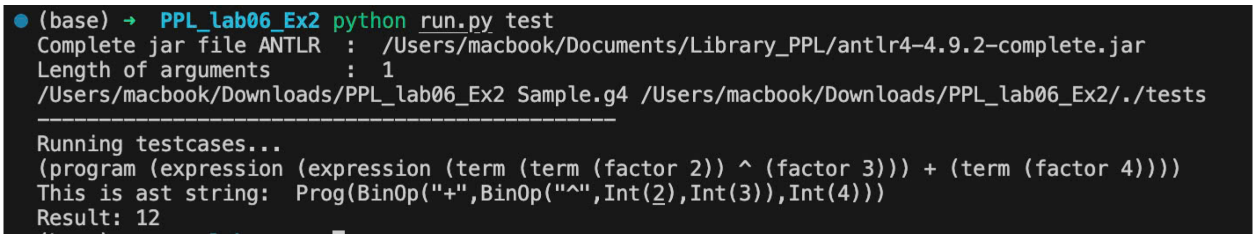
```python
class CodeRunner():
    def visitProgram(self, ctx:Prog):
        return "\n".join([str(expr.accept(self)) for expr in ctx.expr])

    def visitBinaryOp(self, ctx:BinOp):
        left = ctx.left.accept(self)
        right = ctx.right.accept(self)
        if ctx.op == "+":
            return left + right
        elif ctx.op == "-":
            return left - right
        elif ctx.op == "*":
            return left * right
        elif ctx.op == "/":
            return left / right
        elif ctx.op == "%":
            return left % right
        elif ctx.op == "^":
            return left ** right
```

Figure 5: Code runner for Ex2



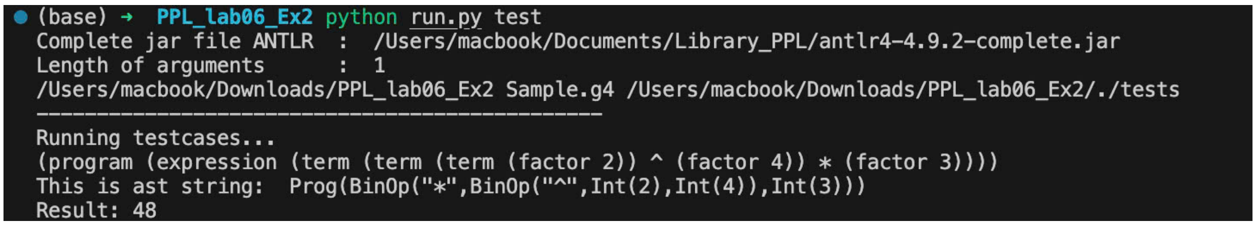
```
(base) → PPL_lab06_Ex2 python run.py test
Complete jar file ANTLR  :  /Users/macbook/Documents/Library_PPL/antlr4-4.9.2-complete.jar
Length of arguments      :  1
/Users/macbook/Downloads/PPL_lab06_Ex2 Sample.g4 /Users/macbook/Downloads/PPL_lab06_Ex2/./tests
----------------------------------------------
Running testcases...
(program (expression (term (term (factor 2)) ^ (factor 3))))
This is ast string:  Prog(BinOp("^",Int(2),Int(3)))
Result: 8
```

Figure 6: Output for Input 1



```
(base) → PPL_lab06_Ex2 python run.py test
Complete jar file ANTLR  :  /Users/macbook/Documents/Library_PPL/antlr4-4.9.2-complete.jar
Length of arguments      :  1
/Users/macbook/Downloads/PPL_lab06_Ex2 Sample.g4 /Users/macbook/Downloads/PPL_lab06_Ex2/./tests
----------------------------------------------
Running testcases...
(program (expression (expression (term (term (factor 2)) ^ (factor 3))) + (term (factor 4))))
This is ast string:  Prog(BinOp("+",BinOp("^",Int(2),Int(3)),Int(4)))
Result: 12
```

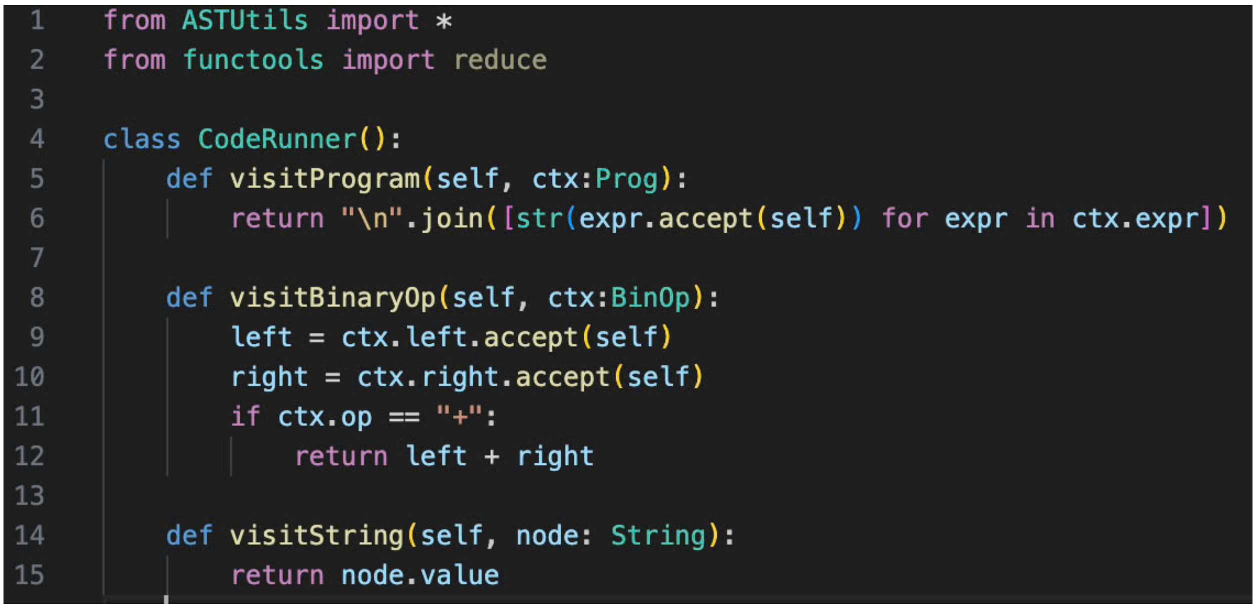
Figure 7: Output for Input 2



```
(base) → PPL_lab06_Ex2 python run.py test
Complete jar file ANTLR  :  /Users/macbook/Documents/Library_PPL/antlr4-4.9.2-complete.jar
Length of arguments      :  1
/Users/macbook/Downloads/PPL_lab06_Ex2 Sample.g4 /Users/macbook/Downloads/PPL_lab06_Ex2/./tests
----------------------------------------------
Running testcases...
(program (expression (term (term (term (factor 2)) ^ (factor 4)) * (factor 3))))
This is ast string:  Prog(BinOp("*",BinOp("^",Int(2),Int(4)),Int(3)))
Result: 48
```

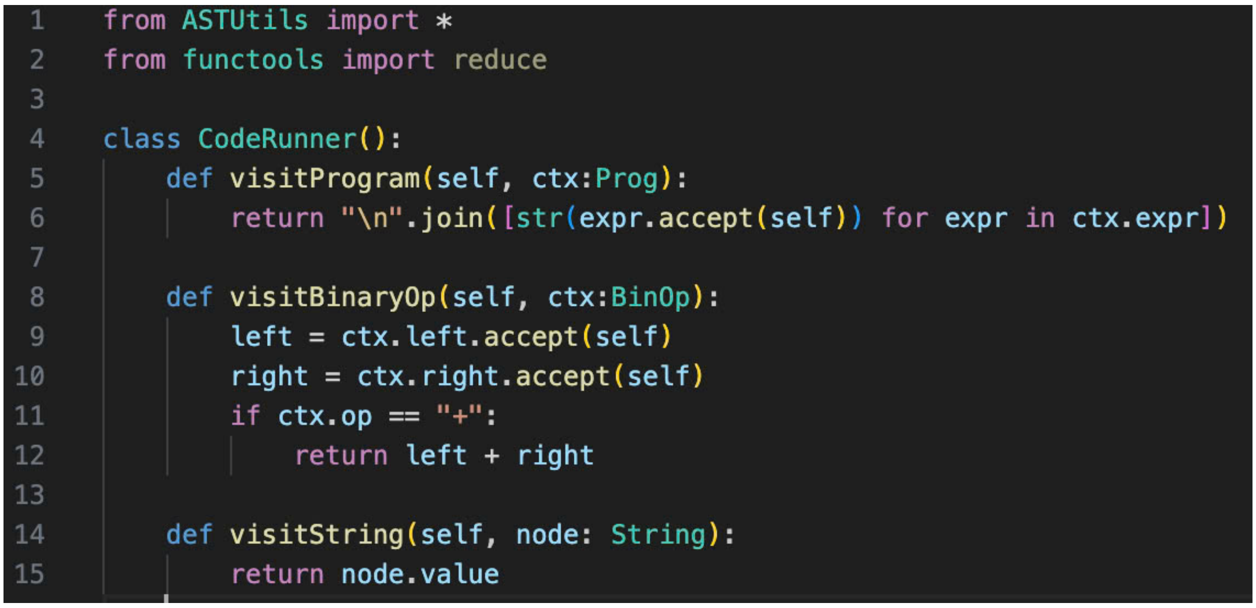
Figure 8: Output for Input 3

3.3 Exercise 3: The grammar that can perform concatenation of two strings.



```python
from ASTUtils import *
from functools import reduce

class CodeRunner():
    def visitProgram(self, ctx:Prog):
        return "\n".join([str(expr.accept(self)) for expr in ctx.expr])

    def visitBinaryOp(self, ctx:BinOp):
        left = ctx.left.accept(self)
        right = ctx.right.accept(self)
        if ctx.op == "+":
            return left + right

    def visitString(self, node: String):
        return node.value
```

Figure 9: Code runner for Ex3



```python
from ASTUtils import *
from functools import reduce

class CodeRunner():
    def visitProgram(self, ctx:Prog):
        return "\n".join([str(expr.accept(self)) for expr in ctx.expr])

    def visitBinaryOp(self, ctx:BinOp):
        left = ctx.left.accept(self)
        right = ctx.right.accept(self)
        if ctx.op == "+":
            return left + right

    def visitString(self, node: String):
        return node.value
```

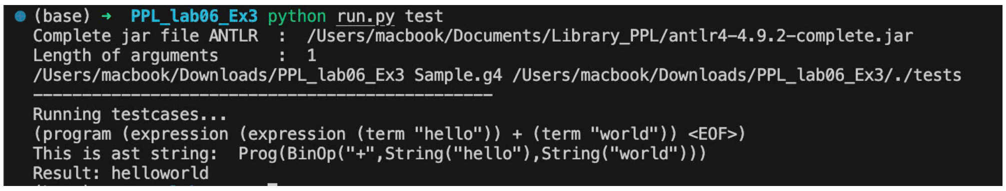
Figure 10: AST for Ex3

Input 1: `"hello" + "world"`

Output 1: `"helloworld"`

Explanation: The program concatenates the two strings ”hello” and ”world” to form ”helloworld”.

Input 2: `"a" + "b" + "c" + "d"`

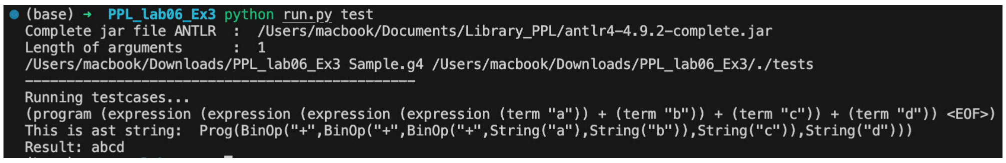


```
● (base) ➜  PPL_lab06_Ex3 python run.py test
  Complete jar file ANTLR  :  /Users/macbook/Documents/Library_PPL/antlr4-4.9.2-complete.jar
  Length of arguments      :  1
  /Users/macbook/Downloads/PPL_lab06_Ex3 Sample.g4 /Users/macbook/Downloads/PPL_lab06_Ex3/./tests
  ----------------------------------------------------
  Running testcases...
  (program (expression (expression (term "hello")) + (term "world")) <EOF>)
  This is ast string:  Prog(BinOp("+",String("hello"),String("world")))
  Result: helloworld
```

Figure 11: Output for Input 1 in Ex3

Output 2: `"abcd"`

Explanation: The program concatenates four strings "a", "b", "c", and "d" to form "abcd".



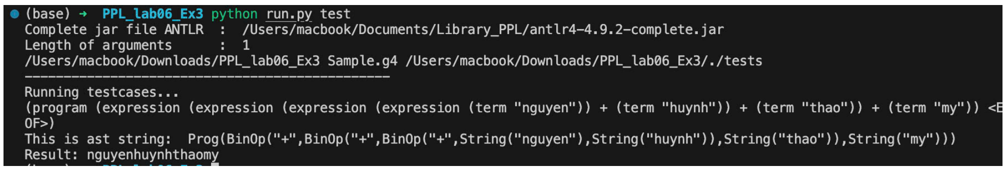
```
● (base) ➜  PPL_lab06_Ex3 python run.py test
  Complete jar file ANTLR  :  /Users/macbook/Documents/Library_PPL/antlr4-4.9.2-complete.jar
  Length of arguments      :  1
  /Users/macbook/Downloads/PPL_lab06_Ex3 Sample.g4 /Users/macbook/Downloads/PPL_lab06_Ex3/./tests
  ----------------------------------------------------
  Running testcases...
  (program (expression (expression (expression (expression (term "a")) + (term "b")) + (term "c")) + (term "d")) <EOF>)
  This is ast string:  Prog(BinOp("+",BinOp("+",BinOp("+",String("a"),String("b")),String("c")),String("d")))
  Result: abcd
```

Figure 12: Output for Input 2 in Ex3

Input 3: `"nguyen" + "huynh" + "thao" + "my"`

Output 3: `"nguyenhuynhthaomy"`

Explanation: The program concatenates four strings "nguyen", "huynh", "thao", and "my" to form "nguyenhuynhthaomy".



```
● (base) ➜  PPL_lab06_Ex3 python run.py test
  Complete jar file ANTLR  :  /Users/macbook/Documents/Library_PPL/antlr4-4.9.2-complete.jar
  Length of arguments      :  1
  /Users/macbook/Downloads/PPL_lab06_Ex3 Sample.g4 /Users/macbook/Downloads/PPL_lab06_Ex3/./tests
  ----------------------------------------------------
  Running testcases...
  (program (expression (expression (expression (expression (term "nguyen")) + (term "huynh")) + (term "thao")) + (term "my")) <E
  OF>)
  This is ast string:  Prog(BinOp("+",BinOp("+",BinOp("+",String("nguyen"),String("huynh")),String("thao")),String("my")))
  Result: nguyenhuynhthaomy
```

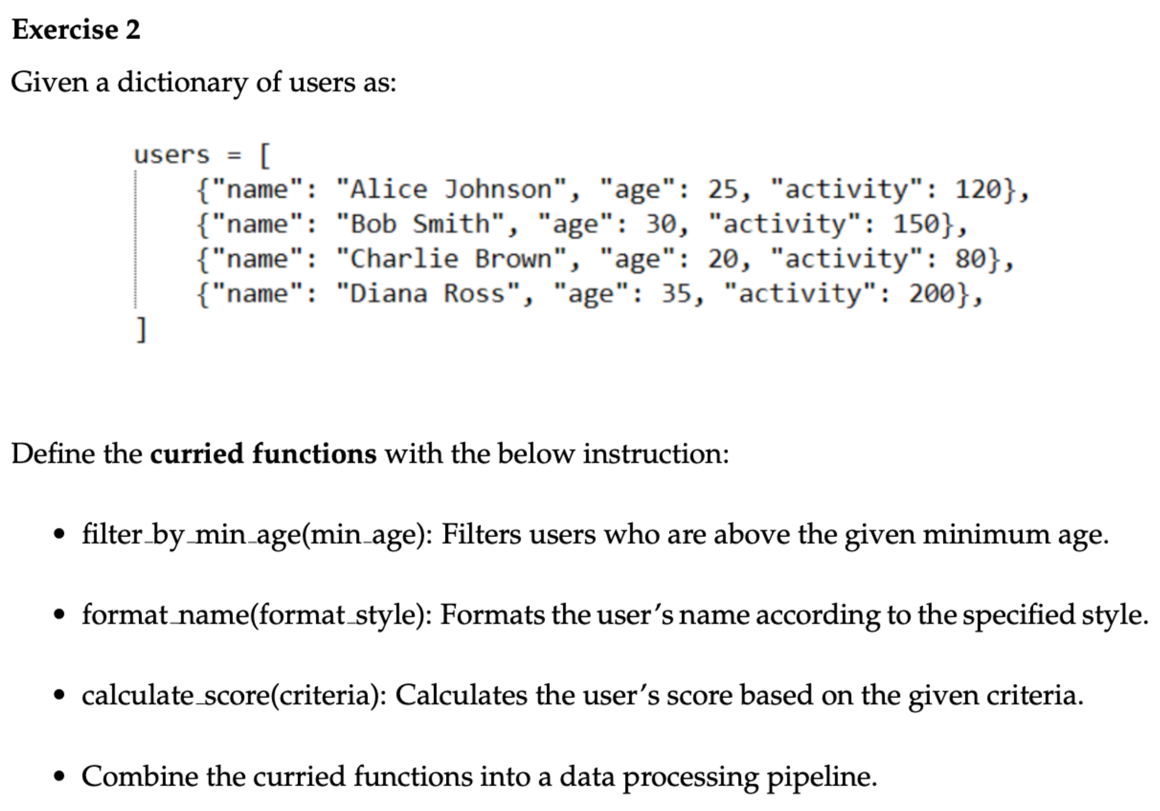
Figure 13: Output for Input 3 in Ex3

4 Exercises confident the most.

Explanation for Exercise Selection

I confidently enjoy working on these exercises because they offer a deep dive into fundamental programming concepts such as **scope**, **dynamic and static binding**, and **function utilization** in Python. These concepts are not only essential in Python but are also applicable across most programming languages.

Moreover, these exercises allow me to directly practice concepts like **currying, data processing**, and **name resolution in nested blocks**. I find the challenges related to scope



Exercise 2

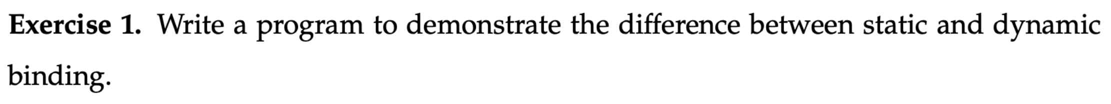
Given a dictionary of users as:

```
users = [
    {"name": "Alice Johnson", "age": 25, "activity": 120},
    {"name": "Bob Smith", "age": 30, "activity": 150},
    {"name": "Charlie Brown", "age": 20, "activity": 80},
    {"name": "Diana Ross", "age": 35, "activity": 200},
]
```

Define the **curried functions** with the below instruction:

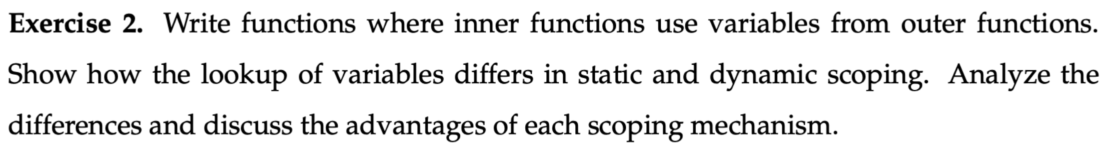
- filter_by_min_age(min_age): Filters users who are above the given minimum age.
- format_name(format_style): Formats the user's name according to the specified style.
- calculate_score(criteria): Calculates the user's score based on the given criteria.
- Combine the curried functions into a data processing pipeline.

Figure 14: Exercise 01



Exercise 1. Write a program to demonstrate the difference between static and dynamic binding.

Figure 15: Exercise 02



Exercise 2. Write functions where inner functions use variables from outer functions. Show how the lookup of variables differs in static and dynamic scoping. Analyze the differences and discuss the advantages of each scoping mechanism.

Figure 16: Exercise 03

and how different programming languages handle this issue especially intriguing. These exercises sharpen my logical thinking and programming skills by having me implement small functions and then combine them into a complete system.

Exercise 3. Write a program to demonstrate scope and name resolution in nested blocks.
Hint:
- Define variables in nested functions or blocks.
- Modify variables in inner blocks and observe their visibility and impact in outer blocks.
- Show the output and explain how Python's scope rules affect name resolution.

P/s: name resolution: the process by which a language determines the value associated with a variable (or identifier) when it is referenced.

Figure 17: Exercise 04