CSAL: Automata and Formal Languages Theory **Lecturer:** James Grimaldo

Sari

Context-Free Grammar

Oblea, Matthew Adelbert R.

Table of Contents

1. Basic Arithmetic, I/O, and String Concatenation

- Context-Free Grammar

$\mathit{program} \rightarrow \mathit{statement\_list}$

$\mathit{statement\_list} \rightarrow \mathit{statement} \mid \mathit{statement\_list}\ \mathit{statement}$

$\mathit{statement} \rightarrow \mathit{assignment} \mid \mathit{input\_statement} \mid \mathit{print\_statement} \mid \mathit{if\_statement} \mid \mathit{while\_loop} \mid \mathit{for\_loop} \mid \mathit{return\_statement} \mid \mathit{function\_definition} \mid \mathit{function\_call}$

$\mathit{if\_statement} \rightarrow \mathit{kung}\ \mathit{expression}\ :\ \mathit{block} \mid \mathit{kung}\ \mathit{expression} :\ \mathit{block}\ \mathit{else\_if\_list}\ \mathit{else}$

$\mathit{expression} \rightarrow \mathit{term} \mid \mathit{expression}\ \mathit{operator}\ \mathit{term}$

$\mathit{block} \rightarrow \mathit{statement\_list}$

$\mathit{operator} \rightarrow + \mid - \mid * \mid / \mid \mathit{at} \mid o \mid == \mid \neq \mid < \mid > \mid \leq \mid \geq$

$\mathit{term} \rightarrow \mathit{identifier} \mid \mathit{number} \mid \mathit{string} \mid (\ \mathit{expression}\ ) \mid \mathit{function\_call} \mid \mathit{method\_chain} \mid \mathit{identifier}\ (\ \mathit{expression}\ ) \mid \mathit{input\_statement} \mid \mathit{identifier}\ (\ \mathit{input\_statement}\ ) \mid \mathit{print\_statement} \mid \mathit{expression} \mid \mathit{list}$

$\mathit{assignment} \rightarrow \mathit{identifier}\ =\ \mathit{expression} \mid \mathit{identifier}\ =\ (\ \mathit{expression}\ ) \mid \mathit{identifier}\ =\ (\ \mathit{expression}\ )\ \mathit{method\_chain} \mid \mathit{identifier}\ =\ \mathit{list} \mid \mathit{identifier}\ \mathit{assign\_op}\ \mathit{expression}$

$\mathit{string} \rightarrow \backslash\ .*?\ \backslash$

$\mathit{identifier} \rightarrow [a-zA-Z\_][a-zA-Z0-9\_]*$

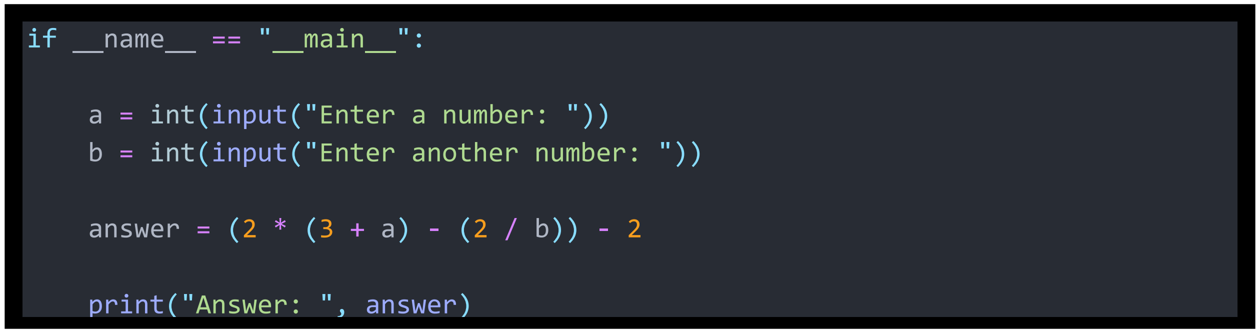
$\mathit{input\_statement} \rightarrow \mathit{basahin}\ (\ \mathit{identifier}\ ) \mid \mathit{basahin}\ (\ \mathit{expression}\ ) \mid \mathit{identifier}\ =\ \mathit{basahin}\ (\ \mathit{string}\ )$

$\mathit{number} \rightarrow [0-9]+ \mid [0-9]+.[0-9]*$

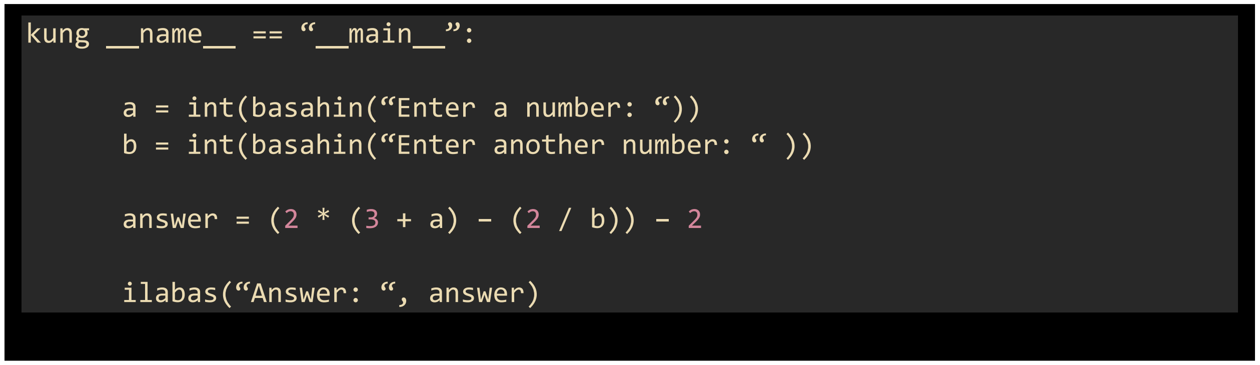
$\mathit{print\_statement} \rightarrow \mathit{ilabas}\ (\ \mathit{f\_string}\ ) \mid \mathit{ilabas}\ (\ \mathit{expression\_list}\ ) \mid \mathit{ilabas}\ (\ \mathit{identifier}\ )$

$\mathit{expression\_list} \rightarrow \mathit{expression} \mid \mathit{expression}\ ,\mathit{expression\_list}$

- Language Name Sample Code

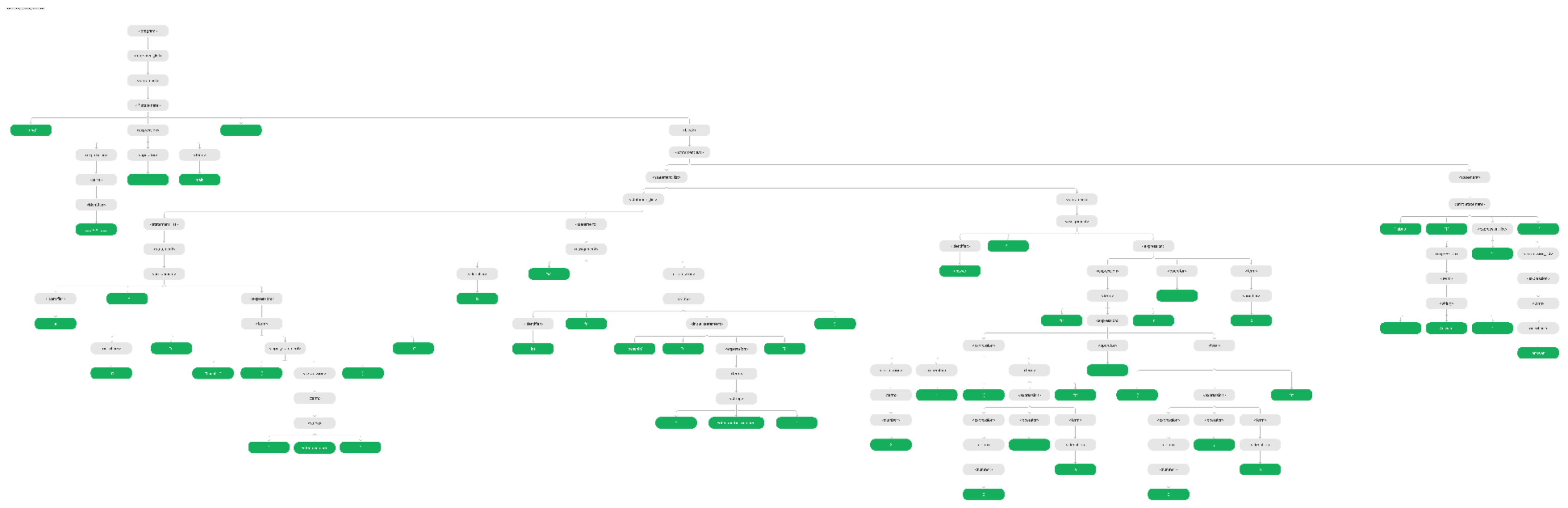


```
if __name__ == "__main__":

    a = int(input("Enter a number: "))
    b = int(input("Enter another number: "))

    answer = (2 * (3 + a) - (2 / b)) - 2

    print("Answer: ", answer)
```



```
kung __name__ == “__main__”:

    a = int(basahin(“Enter a number: “))
    b = int(basahin(“Enter another number: “ ))

    answer = (2 * (3 + a) – (2 / b)) – 2

    ilabas(“Answer: “, answer)
```

- Parse Tree



To enhance your understanding of the CFG Parse Tree, scan the QR code. This will direct you to a digital version of the tree, complete with clearer visuals.

2. Conditional Statement

- Context-Free Grammar

$\mathit{program} \rightarrow \mathit{statement\_list}$

$\mathit{statement\_list} \rightarrow \mathit{statement} \mid \mathit{statement\_list}\ \mathit{statement}$

$\mathit{statement}$
$\rightarrow \mathit{assignment} \mid \mathit{input\_statement} \mid \mathit{print\_statement} \mid \mathit{if\_statement} \mid \mathit{while\_loop} \mid \mathit{for\_loop} \mid \mathit{return\_statement} \mid \mathit{function\_definition} \mid \mathit{function\_call}$

$\mathit{if\_statement} \rightarrow \mathit{kung}\ \mathit{expression}\ :\ \mathit{block} \mid \mathit{kung}\ \mathit{expression} :\ \mathit{block}\ \mathit{else\_if\_list}\ \mathit{else}$

$\mathit{expression} \rightarrow \mathit{term} \mid \mathit{expression}\ \mathit{operator}\ \mathit{term}$

$\mathit{block} \rightarrow \mathit{statement\_list}$

$\mathit{operator} \rightarrow + \mid - \mid * \mid / \mid \mathit{at} \mid o \mid == \mid \neq \mid < \mid > \mid \leq \mid \geq$

term
$\rightarrow \mathit{identifier} \mid \mathit{number} \mid \mathit{string} \mid (\ \mathit{expression}\ ) \mid \mathit{function\_call} \mid \mathit{method\_chain} \mid \mathit{identifier}\ (\ \mathit{expression}\ ) \mid \mathit{input\_statement} \mid \mathit{identifier}\ (\ \mathit{input\_statement}\ ) \mid \mathit{print\_statement} \mid \mathit{expression} \mid \mathit{list}$

$\mathit{assignment} \rightarrow \mathit{identifier}\ =\ \mathit{expression} \mid \mathit{identifier}\ =\ (\ \mathit{expression}\ ) \mid \mathit{identifier}$
$=\ (\ \mathit{expression}\ )\ \mathit{method\_chain} \mid \mathit{identifier}\ =\ \mathit{list} \mid \mathit{identifier}\ \mathit{assign\_op}\ \mathit{expression}$

$\mathit{string} \rightarrow \backslash\ .*?\ \backslash$

$\mathit{identifier} \rightarrow [a - zA - Z\_][a - zA - Z0 - 9\_]*$

$\mathit{input\_statement} \rightarrow \mathit{basahin}\ (\ \mathit{identifier}\ ) \mid \mathit{basahin}\ (\ \mathit{expression}\ ) \mid \mathit{identifier}\ =\ \mathit{basahin}\ (\ \mathit{string}\ )$

$\mathit{number} \rightarrow [0 - 9] + \mid [0 - 9]+.\ [0 - 9]*$

$\mathit{print\_statement} \rightarrow \mathit{ilabas}\ (\ \mathit{f\_string}\ ) \mid \mathit{ilabas}\ (\ \mathit{expression\_list}\ ) \mid \mathit{ilabas}\ (\ \mathit{identifier}\ )$

$\mathit{expression\_list} \rightarrow \mathit{expression} \mid \mathit{expression}\ ,\mathit{expression\_list}$

$\mathit{method\_chain} \rightarrow .\ \mathit{function\_call} \mid \mathit{method\_chain}\ .\ \mathit{function\_call}$

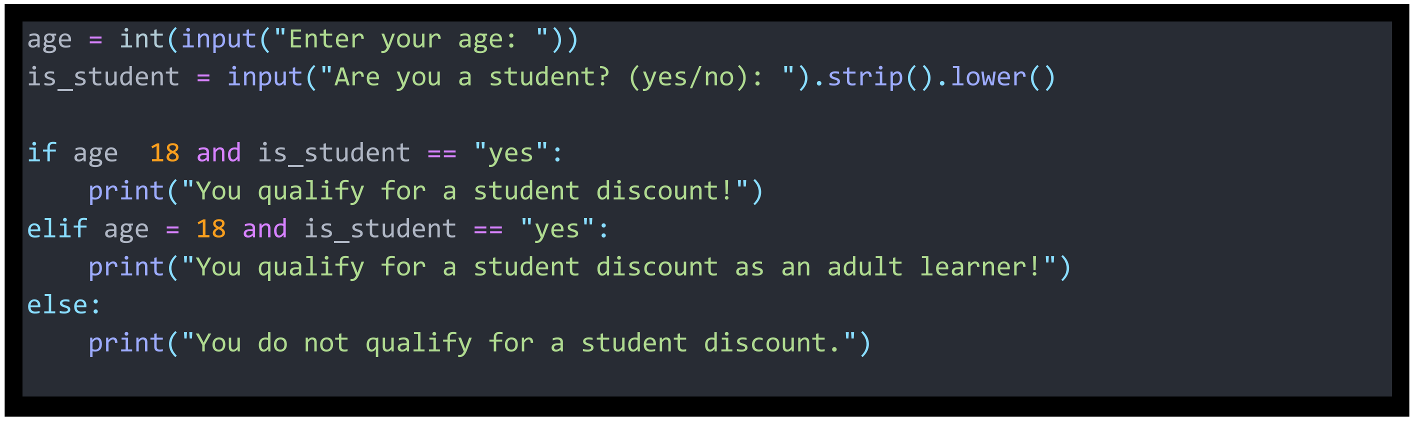
$\mathit{function\_call} \rightarrow \mathit{identifier}\ (\ \mathit{arguments}\ )$

$\mathit{arguments} \rightarrow \mathit{expression} \mid \mathit{expression}\ ,\mathit{arguments} \mid \varepsilon$

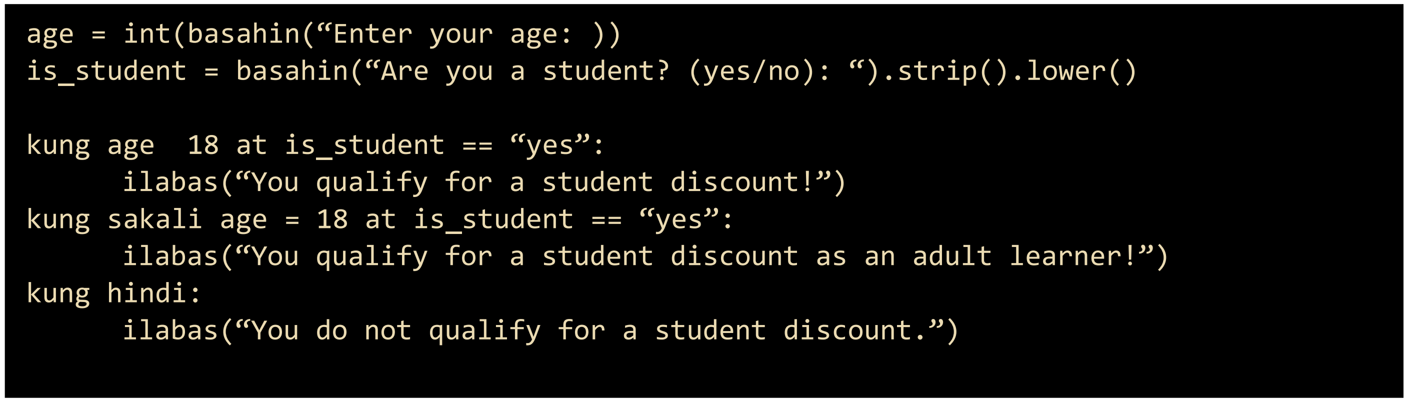
else_list → *kung sakali expression* : *block* | *ε*

else → *kung hindi* : *block*

- Language Name Sample Code

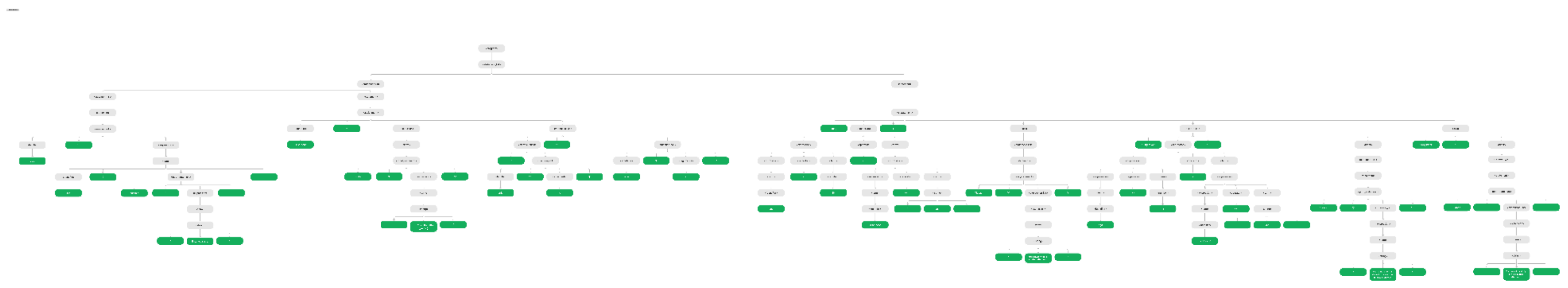


```
age = int(input("Enter your age: "))
is_student = input("Are you a student? (yes/no): ").strip().lower()

if age  18 and is_student == "yes":
    print("You qualify for a student discount!")
elif age = 18 and is_student == "yes":
    print("You qualify for a student discount as an adult learner!")
else:
    print("You do not qualify for a student discount.")
```



```
age = int(basahin("Enter your age: ))
is_student = basahin("Are you a student? (yes/no): ").strip().lower()

kung age  18 at is_student == "yes":
      ilabas("You qualify for a student discount!")
kung sakali age = 18 at is_student == "yes":
      ilabas("You qualify for a student discount as an adult learner!")
kung hindi:
      ilabas("You do not qualify for a student discount.")
```

- Parse Tree



To enhance your understanding of the CFG Parse Tree, scan the QR code. This will direct you to a digital version of the tree, complete with clearer visuals.

3. Loops – For Loops

- Context-Free Grammar

$\mathit{program} \rightarrow \mathit{statement\_list}$

$\mathit{statement\_list} \rightarrow \mathit{statement} \mid \mathit{statement\_list}\ \mathit{statement}$

$\mathit{statement} \rightarrow \mathit{assignment} \mid \mathit{input\_statement} \mid \mathit{print\_statement} \mid \mathit{if\_statement} \mid \mathit{while\_loop} \mid \mathit{for\_loop} \mid \mathit{return\_statement} \mid \mathit{function\_definition} \mid \mathit{function\_call}$

$\mathit{expression} \rightarrow \mathit{term} \mid \mathit{expression}\ \mathit{operator}\ \mathit{term}$

$\mathit{block} \rightarrow \mathit{statement\_list}$

$\mathit{term} \rightarrow \mathit{identifier} \mid \mathit{number} \mid \mathit{string} \mid (\ \mathit{expression}\ ) \mid \mathit{function\_call} \mid \mathit{method\_chain} \mid \mathit{identifier}\ (\ \mathit{expression}\ ) \mid \mathit{input\_statement} \mid \mathit{identifier}\ (\ \mathit{input\_statement}\ ) \mid \mathit{print\_statement} \mid \mathit{expression} \mid \mathit{list}$

$\mathit{assignment} \rightarrow \mathit{identifier}\ =\ \mathit{expression} \mid \mathit{identifier}\ =\ (\ \mathit{expression}\ ) \mid \mathit{identifier}\ =\ (\ \mathit{expression}\ )\ \mathit{method\_chain} \mid \mathit{identifier}\ =\ \mathit{list} \mid \mathit{identifier}\ \mathit{assign\_op}\ \mathit{expression}$

$\mathit{identifier} \rightarrow [a - zA - Z\_][a - zA - Z0 - 9\_] *$

$\mathit{number} \rightarrow [0 - 9] + \mid [0 - 9]+.[0 - 9] *$

$\mathit{print\_statement} \rightarrow \mathit{ilabas}\ (\ \mathit{f\_string}\ ) \mid \mathit{ilabas}\ (\ \mathit{expression\_list}\ ) \mid \mathit{ilabas}\ (\ \mathit{identifier}\ )$

$\mathit{list} \rightarrow [\ \mathit{elements}\ ]$

$\mathit{for\_loop} \rightarrow \mathit{para}\ \mathit{identifier}\ \mathit{sa}\ \mathit{expression}\ :\ \mathit{block}$

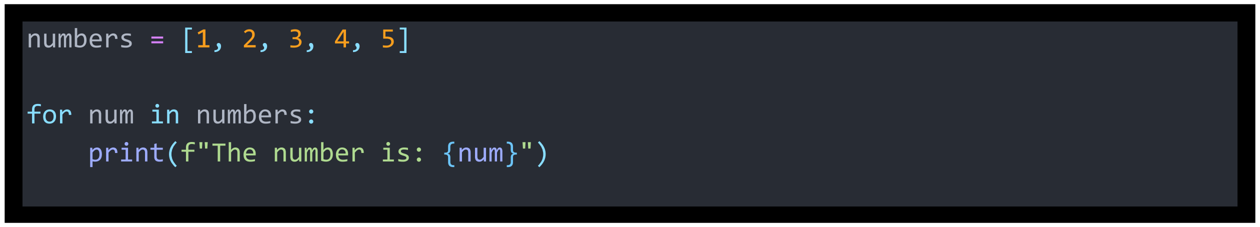
$\mathit{f\_string} \rightarrow f\ “\ \mathit{f\_string\_parts}\ ”$

$\mathit{f\_string\_parts} \rightarrow \mathit{f\_string\_part} \mid \mathit{f\_string\_part}\ \mathit{f\_string\_parts}$

$\mathit{f\_string\_part} \rightarrow \mathit{text}\ \{\ \mathit{expression}\ \} \mid \mathit{text}$

$\mathit{text} \rightarrow \backslash\ .*?\ \backslash$

- Language Name Sample Code
    - For loop



```
numbers = [1, 2, 3, 4, 5]

for num in numbers:
    print(f"The number is: {num}")
```

```
numbers = [1, 2, 3, 4, 5]

para num sa numbers:
      ilabas(f”The number is: {num}”)
```

3. Loops – While Loops

- Context-Free Grammar

$program \rightarrow statement\_list$

$statement\_list \rightarrow statement \mid statement\_list\ statement$

$statement$
$\rightarrow assignment \mid input\_statement \mid print\_statement \mid if\_statement \mid while\_loop \mid for\_loop \mid return\_statement \mid function\_definition \mid function\_call$

$expression \rightarrow term \mid expression\ operator\ term$

$block \rightarrow statement\_list$

$term$
$\rightarrow identifier \mid number \mid string \mid (\ expression\ ) \mid function\_call \mid method\_chain \mid identifier\ (\ expression\ ) \mid input\_statement \mid identifier\ (\ input\_statement\ ) \mid print\_statement \mid expression \mid list$

$assignment \rightarrow identifier\ =\ expression \mid identifier\ =\ (\ expression\ ) \mid identifier$
$=\ (\ expression\ )\ method\_chain \mid identifier\ =\ list \mid identifier\ assign\_op\ expression$

$identifier \rightarrow [a-zA-Z\_][a-zA-Z0-9\_]*$

$number \rightarrow [0-9]+ \mid [0-9]+.[0-9]*$

$print\_statement \rightarrow ilabas\ (\ f\_string\ ) \mid ilabas\ (\ expression\_list\ ) \mid ilabas\ (\ identifier\ )$

$list \rightarrow [\ elements\ ]$

$while\_loop \rightarrow habang\ expression\ :\ block$

$f\_string \rightarrow f\ “\ f\_string\_parts\ ”$

$f\_string\_parts \rightarrow f\_string\_part \mid f\_string\_part\ f\_string\_parts$

$f\_string\_part \rightarrow text\ \{\ expression\ \} \mid text$

$text \rightarrow \backslash\ .*?\ \backslash$

$operator \rightarrow + \mid - \mid * \mid / \mid at \mid o \mid == \mid \neq \mid < \mid > \mid \leq \mid \geq$

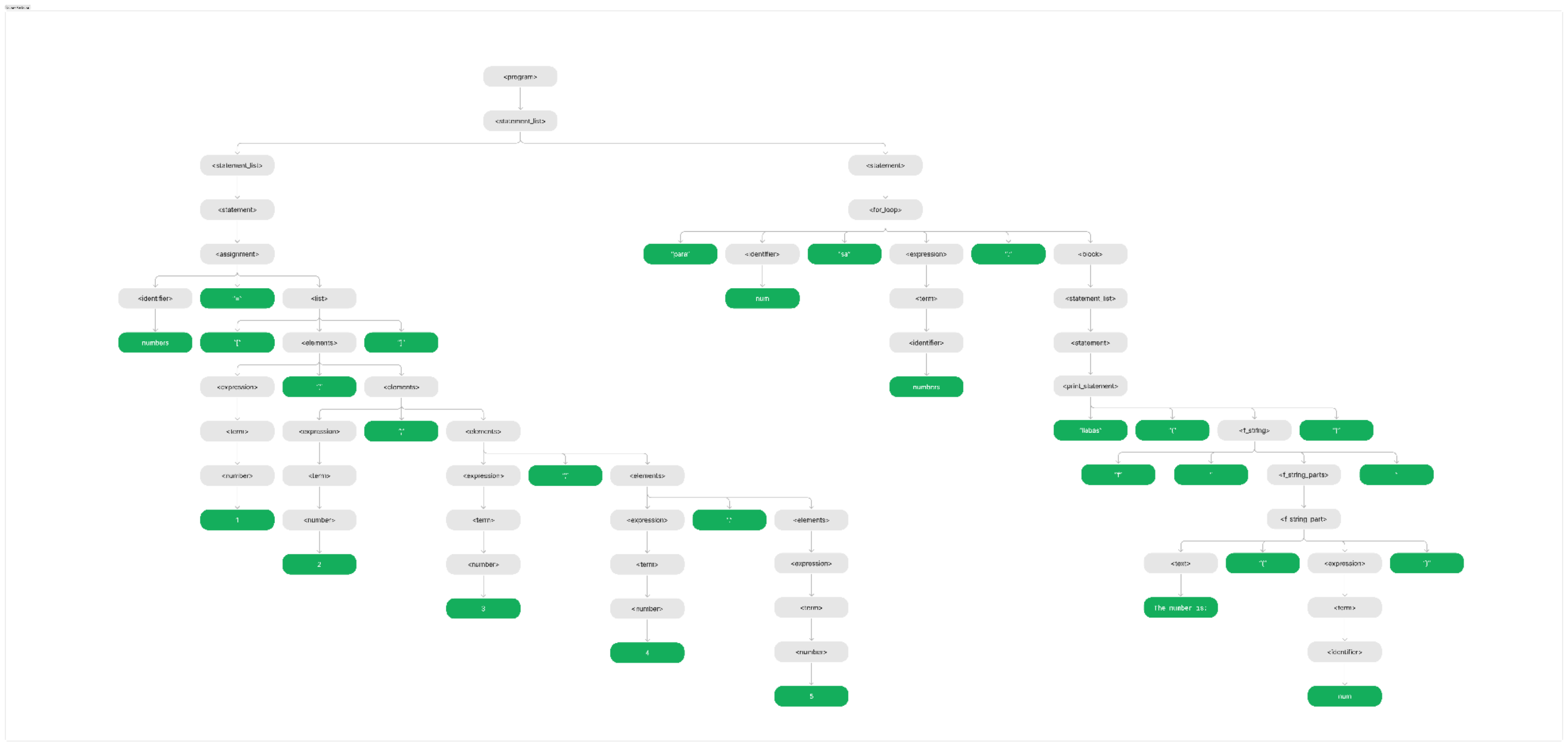
$assign\_op \rightarrow += \mid -=$

- While loop

```
counter = 1

while counter = 5:
    print(f"The counter is: {counter}")
    counter += 1
```

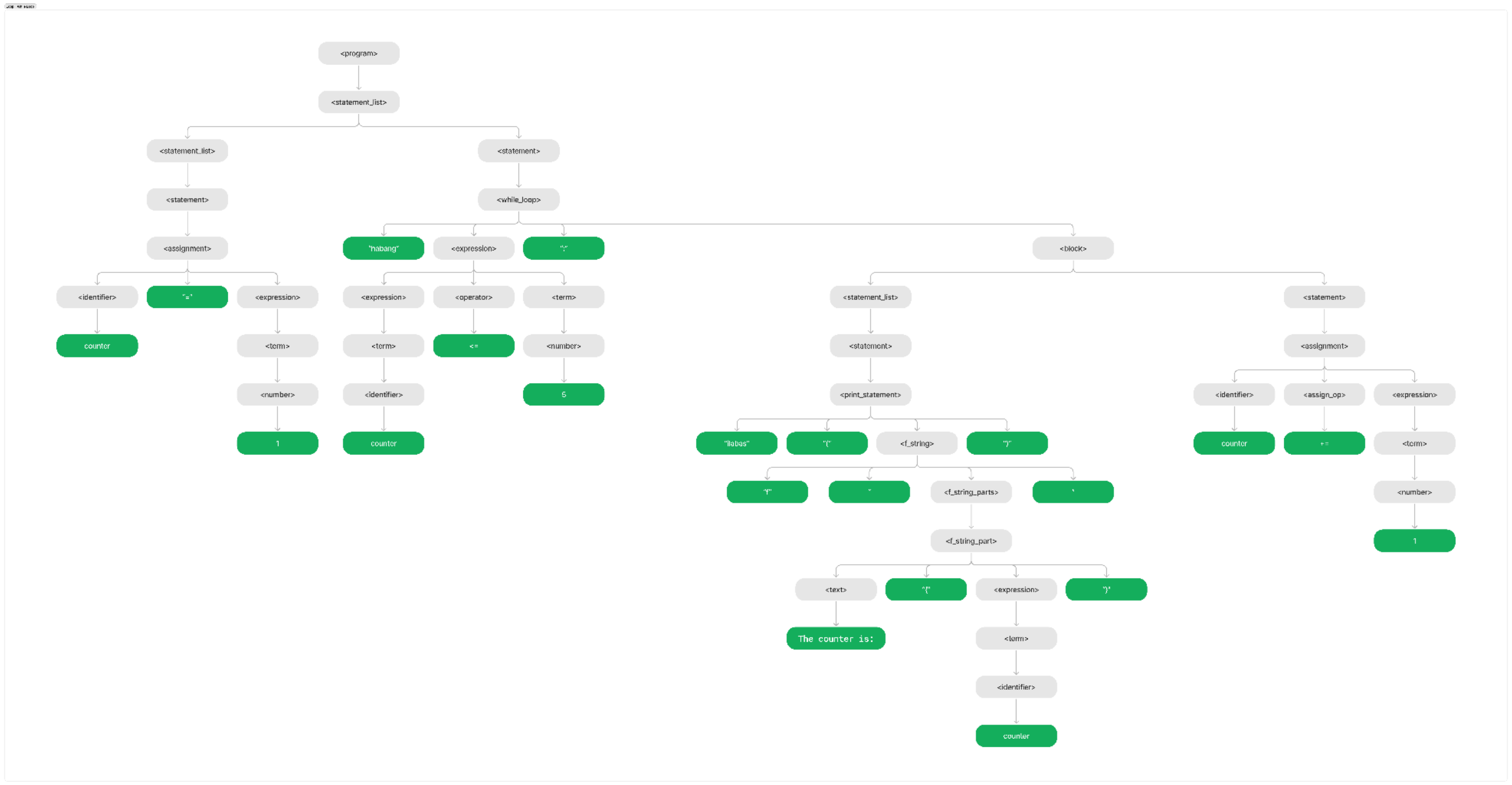
```
counter = 1

habang counter = 5:
      ilabas(f”The counter is: {counter}”)
      counter += 1
```

- For Loop Parse Tree



To enhance your understanding of the CFG Parse Tree, scan the QR code. This will direct you to a digital version of the tree, complete with clearer visuals.

- While Parse Tree



To enhance your understanding of the CFG Parse Tree, scan the QR code. This will direct you to a digital version of the tree, complete with clearer visuals.

4. Function

- Context-Free Grammar

program → statement_list

statement_list → statement | statement_list statement

statement
→ assignment | input_statement| print_statement | if_statement | while_loop | for_loop | return_statement | function_definition | function_call

if_statement → kung expression : block | kung expression : block else_if_list else

expression → term | expression operator term

block → statement_list

operator → + | − | ∗ | / | at | o | == | ≠ | < | > | ≤ | ≥

term
→ identifier | number | string | (expression) | function_call | method_chain | identifier (expression) | input_statement | identifier (input_statement) | print_statement | expression | list

string → \ .∗? \

identifier → [a − zA − Z_][a − zA − Z0 − 9_] ∗

print_statement → ilabas (f_string) | ilabas (expression_list) | ilabas (identifier)

expression_list → expression | expression , expression_list

function_call → identifier (arguments)

arguments → expression | expression , arguments | ε

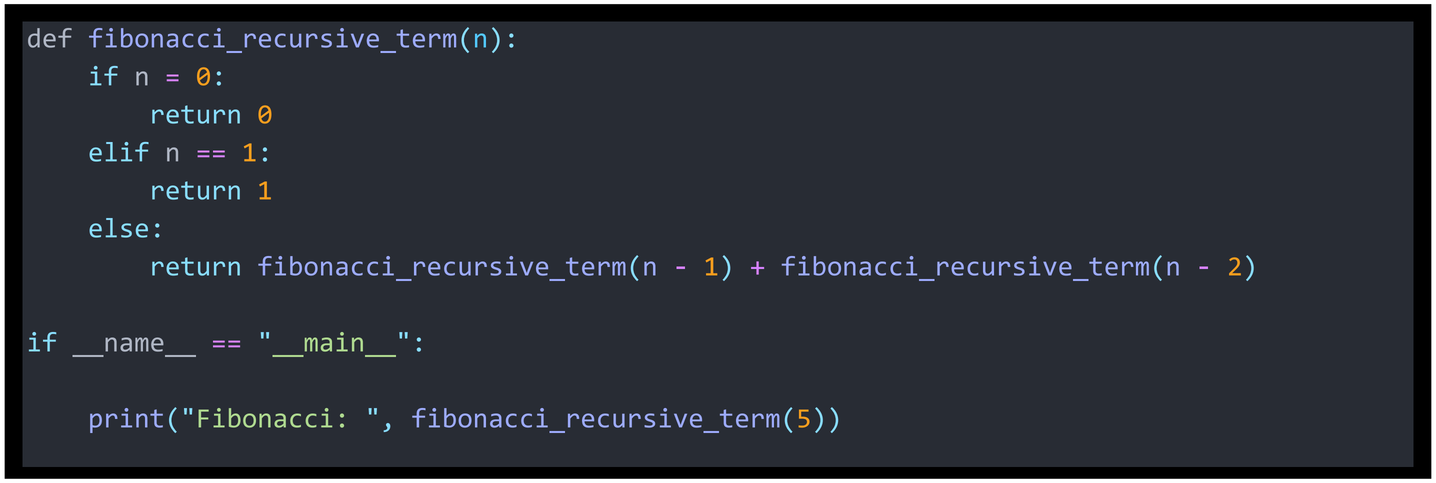
function_definition → itakda identifier (parameters) : block

parameters → parameter | parameter , parameters | ε

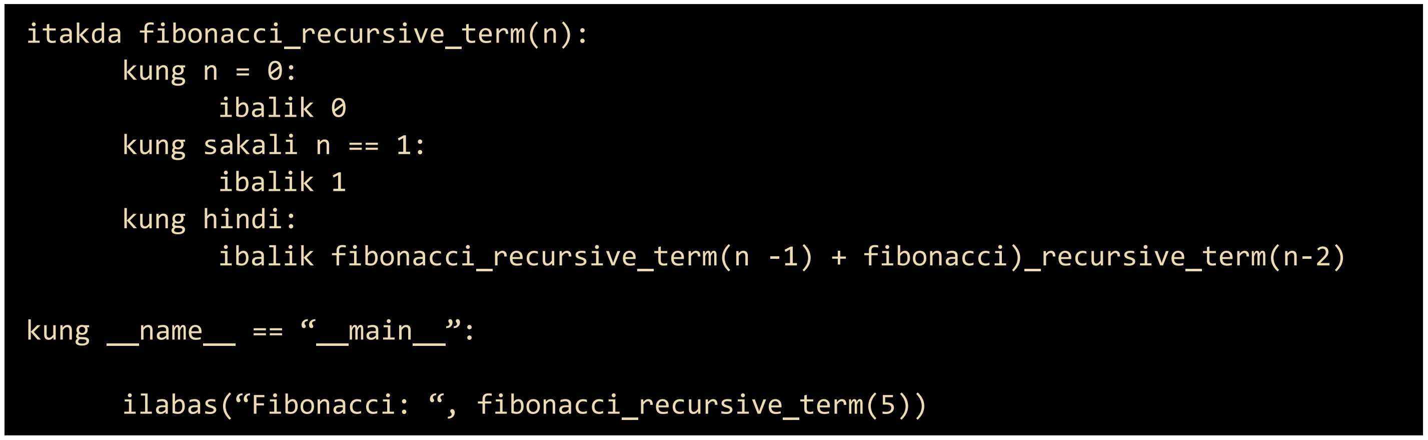
parameter → identifier

return_statement → ibalik expression

- Language Name Sample Code

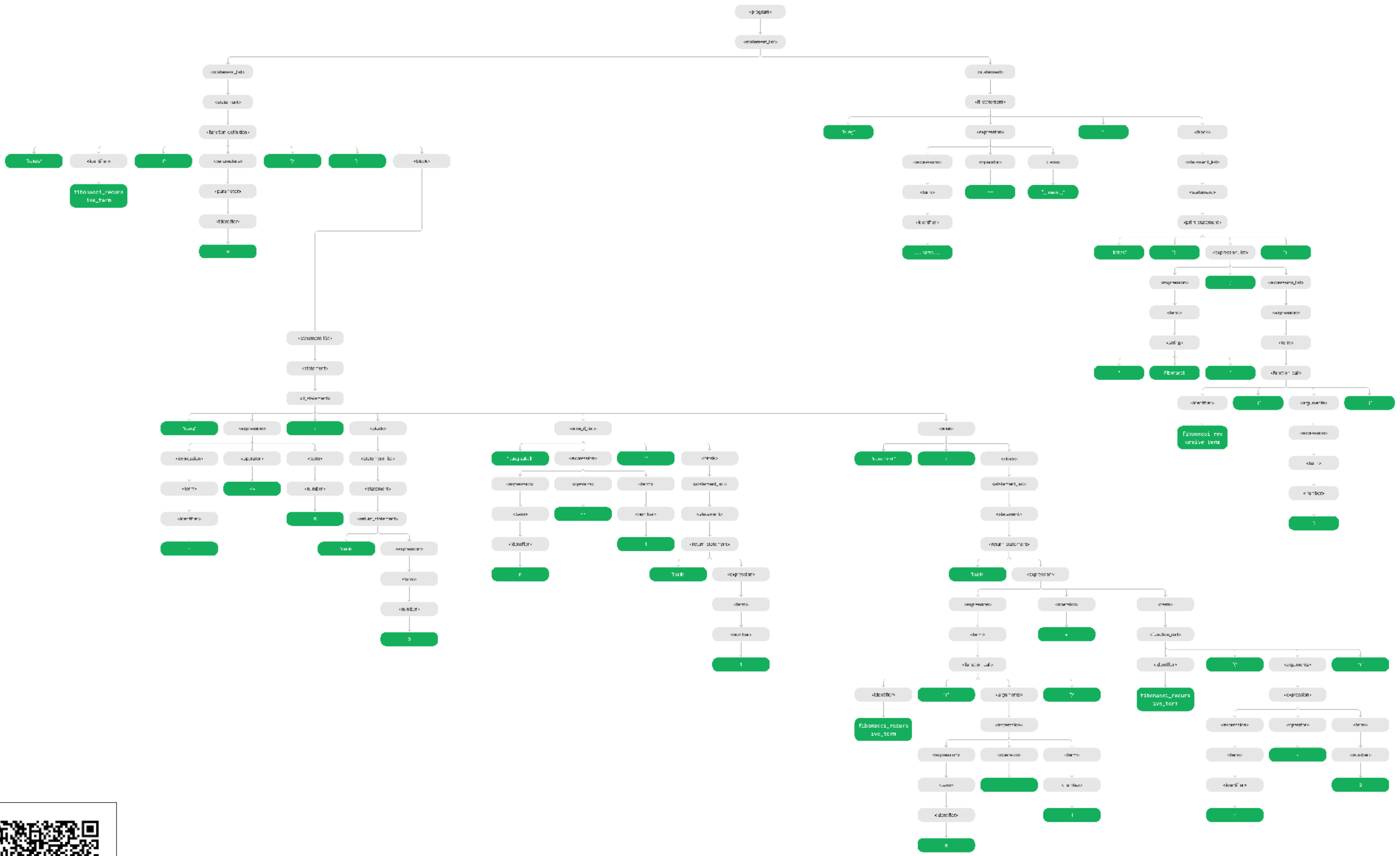


```python
def fibonacci_recursive_term(n):
    if n = 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fibonacci_recursive_term(n - 1) + fibonacci_recursive_term(n - 2)

if __name__ == "__main__":

    print("Fibonacci: ", fibonacci_recursive_term(5))
```



```
itakda fibonacci_recursive_term(n):
      kung n = 0:
            ibalik 0
      kung sakali n == 1:
            ibalik 1
      kung hindi:
            ibalik fibonacci_recursive_term(n -1) + fibonacci)_recursive_term(n-2)

kung __name__ == “__main__”:

      ilabas(“Fibonacci: “, fibonacci_recursive_term(5))
```

■ Parse Tree



To enhance your understanding of the CFG Parse Tree, scan the QR code. This will direct you to a digital version of the tree, complete with clearer visuals.