Design of CFG For Python

⇒ CFG For Variable Initialization and Variable declaration:-

In python, there is no need to declare a Variable. at a Run time Compiler automatically decides for which variable integer, float, character etc.

Sample Statement:-

```
X=10
name= "Ali"
balance = 5000
Pins = [ 1111 ,2222, 3333]
Y= X+5
```

∴ Non-Terminals:-

```
INIT        # Complete Initialization statement
VAR         # Variable name
EXPR        # Expression / value assigned
VAL         # Value (numbers or String)
DIGIT       # Single digit
CHAR        # Single character
LIST        # Python list
ELEMENTS    # Elements in List
```

∴ Terminals:-

```
identifier      # a-z, A-Z
0 1 2 3 4 5 6 7 8 9     # Digits
" "             # for string Literals
[ ] ,           # List Syntax
+, -, * /       # Arithmetic operators
=               # Assignment operator
```

1:- Initialization Statement:-

INTT → VAR = EXPR

2:- Variable Name:-

VAR → CHAR VAR | CHAR

CHAR → a|b|c| ..... |z

CHAR → A|B|C|D|E ..... |Z

3:- Expression:-

EXPR → VAL | VAR | EXPR OP EXPR | LIST

OP → + | - | * | /

4:- Value:-

VAL → DIGIT VAL | DIGIT

| "CHAR_SEQ"

DIGIT → 0|1|2 3|4|5|6| 7|8|9

CHAR_SEQ → CHAR CHAR_SEQ | CHAR

5:- LIST:-

LIST → [ELEMENTS]

ELEMENTS → EXPR. ELEMENTS | EXPR

=> Examples:-

1:- Simple Numbex Assignment

INIT

VAR = EXPR

↳ X = VAL

X = 10

2:- List Assignment

VAR = EXPR

Pins = LIST

Pins = [EXPR. ELE

Pins = [111, 22
3333]

For Decision Making:-

Sample Statement:-

```
amount = 2000
balance = 1500
if (amount <=balance):
    Print("withdraw successful")
elif amount >balance:
    Print("Insufficient balance")
else:
    Print("Invalid amount")
```

:- Non- Terminals:-

DESION	#Represent a decision making statement
IF-STMT	# If statemt
ELIF-STMT	#elseif statement
ELSE-STMT	#else statement
COND	# represents a condition
BODY	# Represent the body of an if, elseif or else
STATEMENT	#inside a body
VAR	# variable name
VAL	# variables values
REL-OP	# relational operators
DIGIT	# represent a single digit
CHAR	# Character

Terminals:

```
if , elif, else :        # statements
identifier               # a-z A-Z
Print                    # Print keywork
( )                      # Parenthesis
" "                      # Quotes
= == <> <=> = != # relational operators
+ - * /                  # operations
0 1 2 3 4 5 6 7 8 9 # Digits
```

⇒ Production Rules:-

1:- Decision Making Statement:-

DECISION ⟶ IF_STMT

2:- If statement:-

IF_STMT ⟶ if COND : BODY
| if COND : BODY ELSE-STMT
| if COND : BODY ELIF_STM

3:- Else if statement:-

ELIF-STMT ⟶ elif COND : BODY
| elif COND : BODY ELSE_S

4:- Else statement:-

ELSE_STMT ⟶ else : BODY

5:- Condition:-

COND ⟶ VAR REL-OP VAL

REL-OP ⟶ < | > | == | <= | >= | !=

Body:-

BODY → STATEMENT
| IF-STMT

Statement:-

STATEMENT → VAR=VAL
| VAR = VAR+VAL
| VAR = VAR-VAL
| Print (VAL)

Variable:-

VAR → CHAR VAR | CHAR
CHAR → a|b|c|....|z|A|B|C|....|Z

Value:-

VAL → DIGIT VALIDIGIT
| "CHAR_SEQ"
DIGIT → 0|1|2|3|4|5|6|7|8|9
CHAR_SEQ → CHAR CHAR_SEQ | CHAR

Examples:-

1:- Simple if:-

DECISION → IF_STMT
IF_STMT → if COND : BODY
if balance > 0 : Print("Account is Active")

2:- If-else:-

DESION → IF_STMT

IF-STMT → if enteredPin= Pin : BODY ELS

if entered-pin ==Pin : access = 1 else : access

3:- If-else-if:-

DECISION → IF_STMT

if amount <= balance : BODY ELIF-STMT

if amount <= balance : status =1

elif amount > balance : status =0

else : status = -1

4:- Nested If:-

DECISION → IF_STMT

IF—STMT → if entered Pin ==Pin : IF

if entered-Pin ==Pin :

if amount < = balance: balance=bala

am

> CFG For loops:-

:- For Loop:-

for variable in range(start, end):
    #body statements
        statement 1
        statement 2

:- Terminals:-

for
in
range
(
)
:
,
identifier,
0 1 2 3 4 5 6 7 8 9,
+ - * / =

Non-Terminals:-

FOR_LOOP # represents a 'for' loop using rang
VAR # variable for loop counter
START # Starting value of the loop
END # Ending value of the loop
BODY # statements inside the loop

STATEMENT # Single executable statement inside the loop
EXPR # Expressen used in assignment or arith.
VAL # values used in EXPR
OP # operators
DIGIT # Single digit
CHAR # character

⇒ Production Rules:-

1:- FOR_LOOP statement:-

FOR_LOOP → for VAR in range(START,END)
BODY

2:- Body:-

BODY → STATEMENT | FOR_LOOP #Nested lo
supporte

3:- Statement:-

STATEMENT → VAR = EXPR

4:- Expression:-

EXPR → VAR OP VAL | VAL

5:-Operators:-

OP → +|-|*|/

6:-Variable:-

VAR → CHAR VAR | CHAR
CHAR → a|b|c|....|z
CHAR → A|B|C|....|Z

7:- Valve:-

VAL → DIGIT VAL | DIGIT
DIGIT → 0|1|2|3|4|5|6|7|8|9

⇒ Example Derivation:-

FOR_LOOP
for VAR in range(START, END) : BODY
for i in range(0, 5) : STATEMENT
for i in range(0, 5) : balance = balance+

- While Loop:-

General Syntax:-

```
while Condition:
    #body statements
      statement 1
      statement 2
```

Terminals:-

while, :, identifier,
0 1 2 3 4 5 6 7 8 9
+ - * / = < > <= >= == !=

Non-Terminals:-

WHILE_LOOP	# represents a while loop
COND	# condition
BODY	# Statements inside the loop
STATEMENT	# Single executable statement
VAR	# variable name
VAL	# value
EXPR	# expression
OP	# arithmetic operator
REL_OP	# relational operator
DIGIT	# single digit
CHAR	# character

⇒ Production Rules:-

1:- WHILE_LOOP Statement:-

WHILE_LOOP → while COND: BODY

2:- Body:-

BODY → STATEMENT | WHILE_LOOP #nested loops

3:- Statement:-

STATEMENT → VAR = EXPR

4:- Expression:-

EXPR → VAR OP VAL | VAL

5:- Condition:-

COND → VAR REL_OP-VAL

6:- Relational Operators:-

REL-OP → < | > | == | <= | >= | != | ==

7:- Variable:-

VAR → CHAR VAR | CHAR

CHAR → a|b|c|.... |z

CHAR → A|B|C|.... |Z

8:- Value:-

VAL → DIGIT VAL | DIGIT

DIGIT → 0|1|2|3|4|5|6|7|8|9

Example Derivation:-

WHILE-LOOP

while COND: BODY

while balance < 1000 : STATEMENT

while balance < 1000 : balance = balance + 50

- FOREACH Loop:-

General Syntax:-

```
for variable in list-variable:
    # body statements
    statement1
    statement2
```

- Terminals:-

for, in, :, identifier

[ ]

0 1 2 3 4 5 6 7 8 9

+ - * / = " "

- NON-TErminals:-

FOREACH-LOOP	# Represents a for-each loop
VAR	# Loop variable
LIST	# Python list to iterate
ELEMENTS	# elements of the list
BODY	# statement inside loop
STATEMENT	# Single executable statement

VAL # valve in statement or list

EXPR # Expression for assignment

OP # Arithmetic operator

DIGIT # Single digit

CHAR # Single character

CHAR-SEQ # Sequence of character for string

⇒ Production Rules:-

1:- For each loop statement:-

FOREACH_LOOP → for VAR in LIST : BODY

2:- Body:-

BODY → STATEMENT | FOREACH_LOOP # n
lo

3:- Statement:-

STATEMENT → VAR= EXPR | Print(VAL)

4:- Expression:-

EXPR → VAR OP VAL | VAL

5:- List:-

LIST → [ELEMENTS]

ELEMENTS → VAL , ELEMENTS | VAL

Variable:-

VAR → CHAR VAR | CHAR

CHAR → a|b|....|z

CHAR → A|B|C|....|Z

Value:-

VAL → DIGIT VAL | DIGIT | "CHAR-SEQ"

DIGIT → 0|1|2|3|4|5|6|7|8|9

CHAR-SEQ → CHAR CHAR-SEQ | CHAR

Example Derivation:-

FOREACH-LOOP

for VAR in LIST : BODY

for Pin in [1111, 2222, 3333] : STATEMENT

for Pin in [1111, 2222, 3333] : print(Pin)

Nested list

```
numbers = [10, 20, 30, 40]
#Access elements
print(numbers[0])

numbers[2] = 35
matrix = [ [1,2,3], [4, 5, 6 ], [7,8,9] ]
// Nested list
```

1:- Terminals:-

identifier, = , [ ]

0 1 2 3 4 5 6 7 8 9

" " +

2:- Non-Terminals:-

ARRAY # represents a complete list

VAR # Variable name

LIST # Python list

ELEMENTS # Elements inside the list

VAL # Single Value (number or stri

EXPR # Expression for assignment or Calculation

BODY # for nested lists

DIGIT # Single digit (0-9)

CHAR # Single character

CHAR-SEQ # Sequence of characters (for String

INDEX # Index for accessing array elements

=> Production Rules:-

:- Array declaration and Initialization:-

ARRAY → VAR = LIST

:- List:-

LIST → [ELEMENTS]

ELEMENTS → VAL, ELEMENTS
| VAL | LIST, ELEMENTS

ELEMENTS → LIST # nested list
supported by comma

:- Value:-

VAL → DIGTT VAL | DIGIT | "CHAR-SEQ"

:- Variable:-

VAR → CHAR VAR | CHAR

CHAR → a | b | c | ..... | z | A | B | C | ... | Z

- Indexing:-

VAR [INDEX] = VAL

INDEX → DIGIT INDEX | DIGIT

Expression:-

EXPR → VAR OP VAL | VAL

OP → + | - | * | /

=> Example Derivations :-

1 :- Simple array :-

VAR = LIST

numbers = [ELEMENTS]

numbers = [10, ELEMENTS]

numbers = [10, 20, 30, 40]

2 :- Nested Array :-

ARRAY

VAR = LIST

matrix = [ELEMENTS]

matrix = [[1,2,3], ELEMENTS]

matrix = [[1,2,3], [4,5,6]]

3 :- Access and update :-

STATEMENT → VAR[INDEX] = VAL

numbers[2] = 35

⇒ CFG For Functions:-

- **Syntax Sample:-**

```
def greet():
    Print("Hello world")
def add (a, b):
    result = a+b
    return result
sum = add (5, 10)
```

:- **Terminals:-**

def ,
return
( )
identifier
0 1 2 3 4 5 6 7 8 9
+ - * / = " "

Non-Terminals:-

FUNC # represents a complete function
FUNC_NAME # function name
PARAMS # Function paramaters list
PARAM # Single Parameter
BODY # Statements inside the fn

STATEMENT # Single executable statement

VAR # Variable name

VAL # Value assigned

EXPR # Expression for assignment or arithmetic

OP # arithmetic operator

DIGIT # single character

CHAR-SEQ # sequence of characters

RETURN_STMT # optional return statement

CALL # Call Statement

ARGUMENTS # Arguments Passed to fn call

=> **Production Rules:-**

1:- **Function Declaration:-**

FUNC → def FUNC_NAME (PARAMS): BODY

RETURN_STMT

2:- **Function Name:-**

FUNC_NAME → CHAR FUNC_NAME|CHAR

CHAR → a|b|c|....|z|A|B|...|Z

3:- Parameters:-

PARAMS → PARAM , PARAMS | PARAM | ∧

PARAM → VAR

4:- Body:-

BODY → STATEMENT | STATEMENT BODY
| FUNC # nested calls

:- statement:-

STATEMENT → VAR = EXPR
| Print(VAL)
| CALL

Expression:-

EXPR → VAR OP VAL
| VAL

P → + | - | * | /

lue:-

L → DIGIT VAL | DIGIT
| "CHAR-SEQ"

T → 0|1|2|3|4| 5|6|7|8|9

CHAR_SEQ → CHAR CHAR_SEQ | CHAR

8:- Return Statement:-

RETURN_STMT → return EXPR | ε

9:- Function Call:-

CALL → FUNC_NAME (ARGUMENTS)

ARGUMENTS → EXPR , ARGUMENTS | EXPR | ε

=> Example:-

```
def FUNC_NAME(PARAMS) : BODY RETURN_STMT
def greet() : print("Hello World")
```

=> CFG For nested Loop and Nested Conditions:-

CFG for nested loop and conditions are implemented in loops and decision making Section.

THE END