Rules For Naming A Variable :

A variable name is a word that consist of following,

1) English letters A-Z and a-z.
2) A dollar Sign ($)
3) No other character is permited.
4) Space not permited.
5) Variable must not start with a digit or underscore.(0-9/_) only a character is necessary after that.
6) Double underscore are not permited.

e.g:

1a	✗
$my	✗
M2	✓
num	✓

- ## Loops :

·) for →

- Syntex :

```
for (<initilize>, <conditional statement>, <increment/decre>)
[
        <Body>
]
```

e.g:

```
for(num val = 0, val<10, Val++)
[
   Show (val)
]
```

- ## Operators :-

·) Arithmetics (+ , - , * , /).
·) Increment (++), Decrement (--).
·) Comparison (< , > , <= , >= , == , !=)

·) Assigment (=)

- Input And Output :-

- Input : take( ).
- Output : Show( ).

- Keywords :-

	Token	Class Part
num	(	(
letter	)	)
either	=	=
or	+	+
for	-	-
defult	÷	/
return	×	*
lable	{ }	[ ]
define	;	,
take	>, <, >=, <=, =,	>, <, >=, <=, ==, !=
show	!=	

~~Langua~~ Variable Initilization & Declearation :-

< Data type > < variable name > < assignment operator > < value >.

e.g:

num value1 = 30.

letter abc

LANGUAGE SPECIFICATION

To define the language we have to give the details about the following:

- Data types.
- Operators.
- Keywords.
- Condition Statements.
- Loops.

- DATA TYPES:

- Integer ⟶ num.
- Float ⟶ decimal.
- Characters ⟶ letters.

- CONDITIONAL STATEMENT:

- either - or

Syntax:

```
either (<Condition>)
[
    <body>
]
or
[
    <body>
]
```

e.g:

```
either (a == 4)
[
    Show ('four')
]
or
[
Show ('not four')
].
```

CamScanner

For Loop :-

* CFG :

<For_Loop> for (<initialize>, <condition>, <increment>)

<initialize> → <datatype> <id> = <value>.

<datatype> → num | letter | decimal

<id> → ~~i | j | k |~~ [a-z]

<value> → <number>

<number> → ~~<digits> +~~ [0 - 9]

~~<digits> → [0-9]~~

<Condition> → <id> <log-op> <number>

<log-op> → < | > | == | <= | >= | !=

< increment > → <id> <inc-op>.

< inc-op > → ++ | -- .

1) Derivation:-

String = for (num i = 1, i < 9, i++)

<for-loop>

→ for (<initialize>, <condition>, <increment>)

→ for (<datatype> <id> = <value>, <conditional>, <increment>)

→ for (num <id> = <value>, <conditional>, <increment>)

→ for (num i = <number>, <condition>, < increment>)

→ for (num i = 1, <condition>, < increment>)

→ for (num i = 1, <id> <log-op> <number>, <increment>)

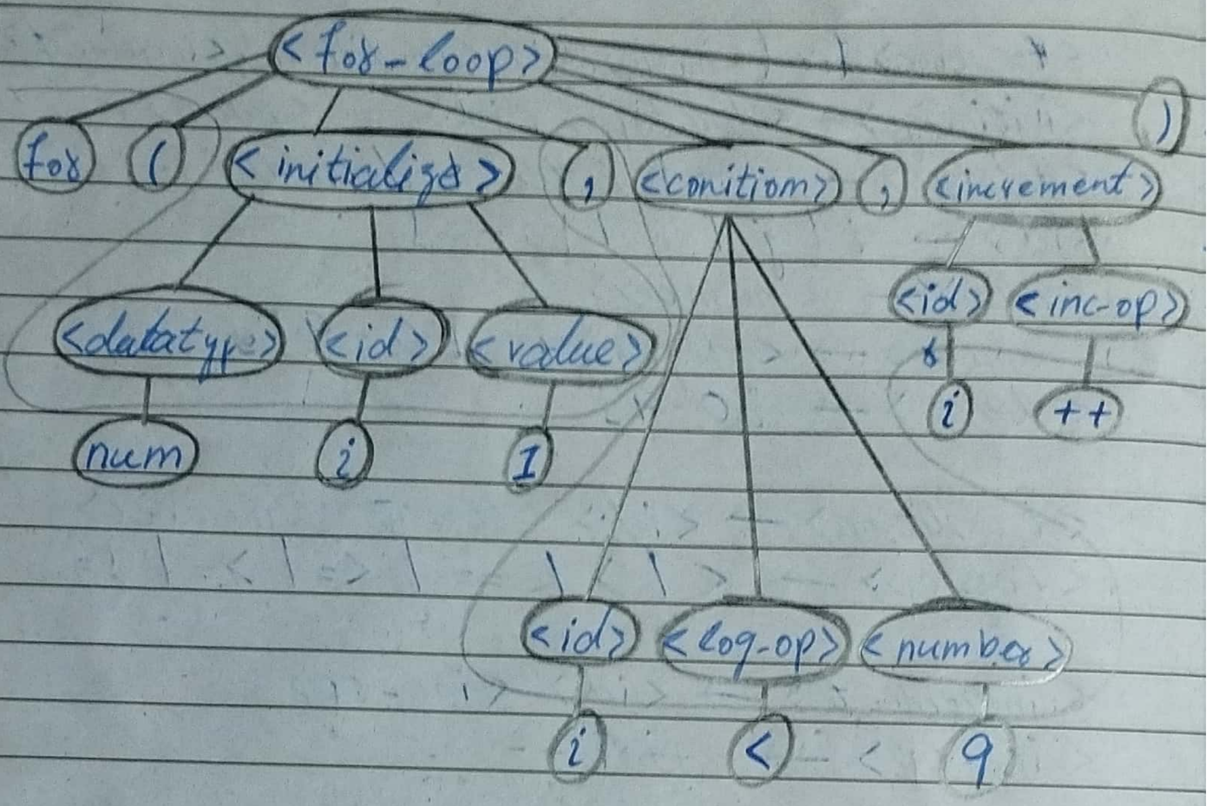
→ for (num i = 1, i < 9, < increment >)

→ for (num i = 1, i < 9, <id> <inc-op>)

→ for (num i = 1, i < 9, i++)

1) PARSE TREE :-

string = for(num i=1; i<9 , i++).



VARIABLE DECLARATION :-

- CFG :-

→ <declaration> → <type> <var-list>
→ <type> → num | decimal | letters .
→ <var-list> → <letters> <id-tail>
→ <id-tail> → <letters> <id-tail> | <digits> <idtail> | _ <id_tail> | ϕ
→ <letters> → [a-z] <letters> | [A-Z] <letters> | ϕ
→ <digit> → [0-9] <digits> | ϕ .

- DERIVATION :-

~~Decl~~ ~~Deriv~~ String => 'num x'.

→ <declaration>

→ <type> <var-list>

→ num <letter> <id-tail>

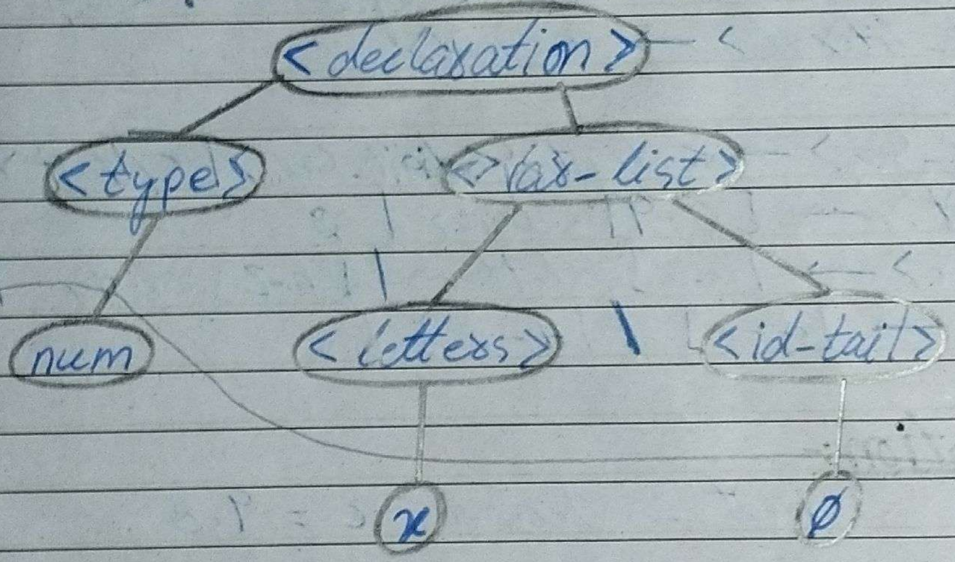
→ num x <id-tail>

→ num x ∅

→ num x

- Parse Tree :-

String => num x



VARIABLE INTTALIZATION

- CFG :-

→ <init> → <type> <identifier> = <value> .

→ <type> → num | decimal | letters

→ <identifier> → <letters> <id-tail>

→ <id-tail> → <letters> <id-tail> | <digits> <id-tail>
| ø .

→ <values> → <int-literal> | <decimal-literal>
| <char-literal>

→ <int-literal> → [-] <digit-Seq> .

→ <decimal-literal> → [-] <digit-Seq> . <digit-Seq>

→ <char_literal> → <char>

→ <digit-Seq> → <digit> <digit-Seq> | <digit>

→ <digit> → [0-9] <digit> | ø

→ <letter> → [a-z] <letter> | [A-Z] <letter> | ø

→ <Char> → [a-z] | [A-Z]

- Derivation :-

input => decimal c = 9.8 .

→ <init>

→ <type> <identifier> = <value>

→ decimal <identifier> = <value>

→ decimal c = <value>

→ decimal c = <decimal-literal>

→ decimal c = <digit-Seq> . <digit.Seq>

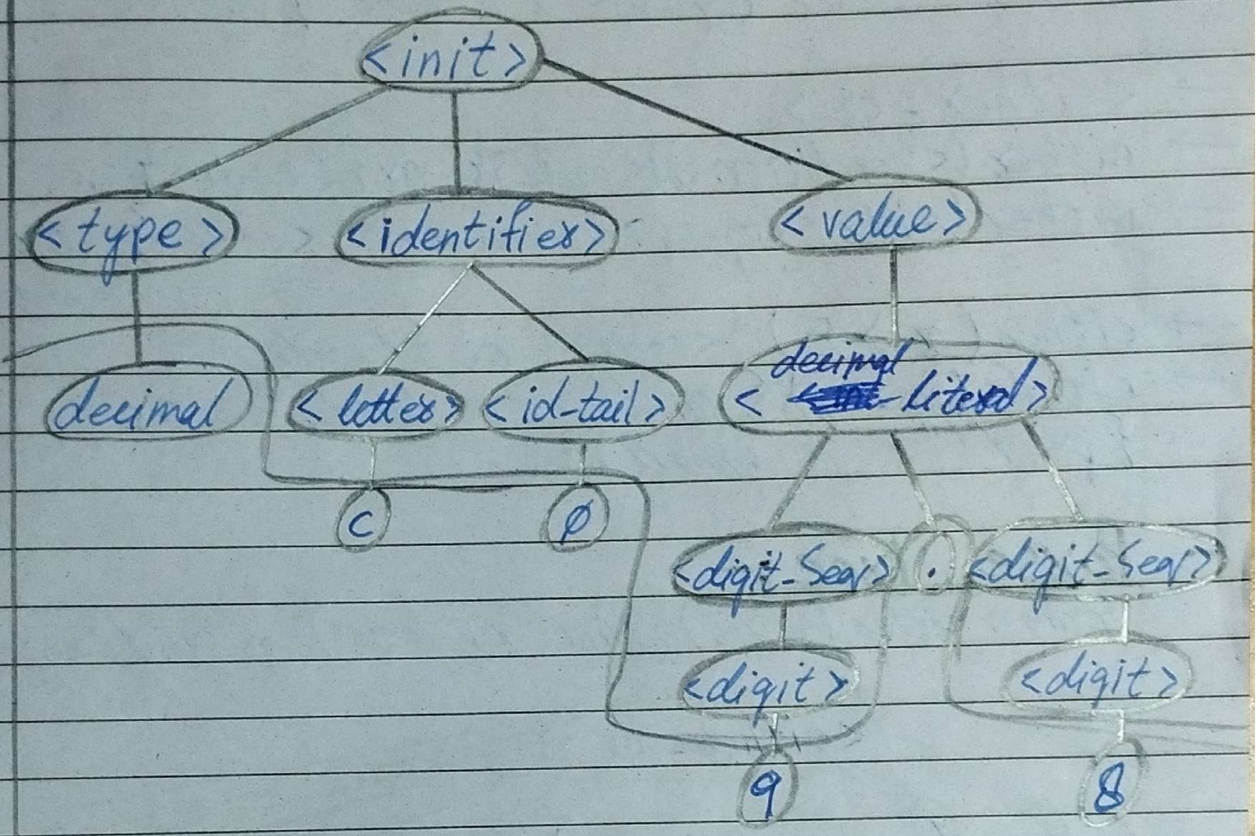
→ decimal c = <digit> . <digit>

→ decimal c = 9.8 .

CS CamScanner

- Parse Tree :-

String => decimal c = 9.8 .



★ Either - Or :-

- CFG :-

→ <Either-Or> → either (<condition>) {<code>} or {<code>}.

→ <condition> → <id> < log-op > < number > .

→ <id> → [a-z] <id> | [A-Z] <id> | [A-Z] | [a-z].

→ <log-op> → < | > | == | <= | >= | != .

→ <number> → [0-9] <number> | [0-9] .

→ <code> → any text .

- Derivation :-

String => either (x>5) { print("x is large"} or { print(" x is small ")}.

→ <either - or >

→ either (<condition>){<code>} or{< code>} .

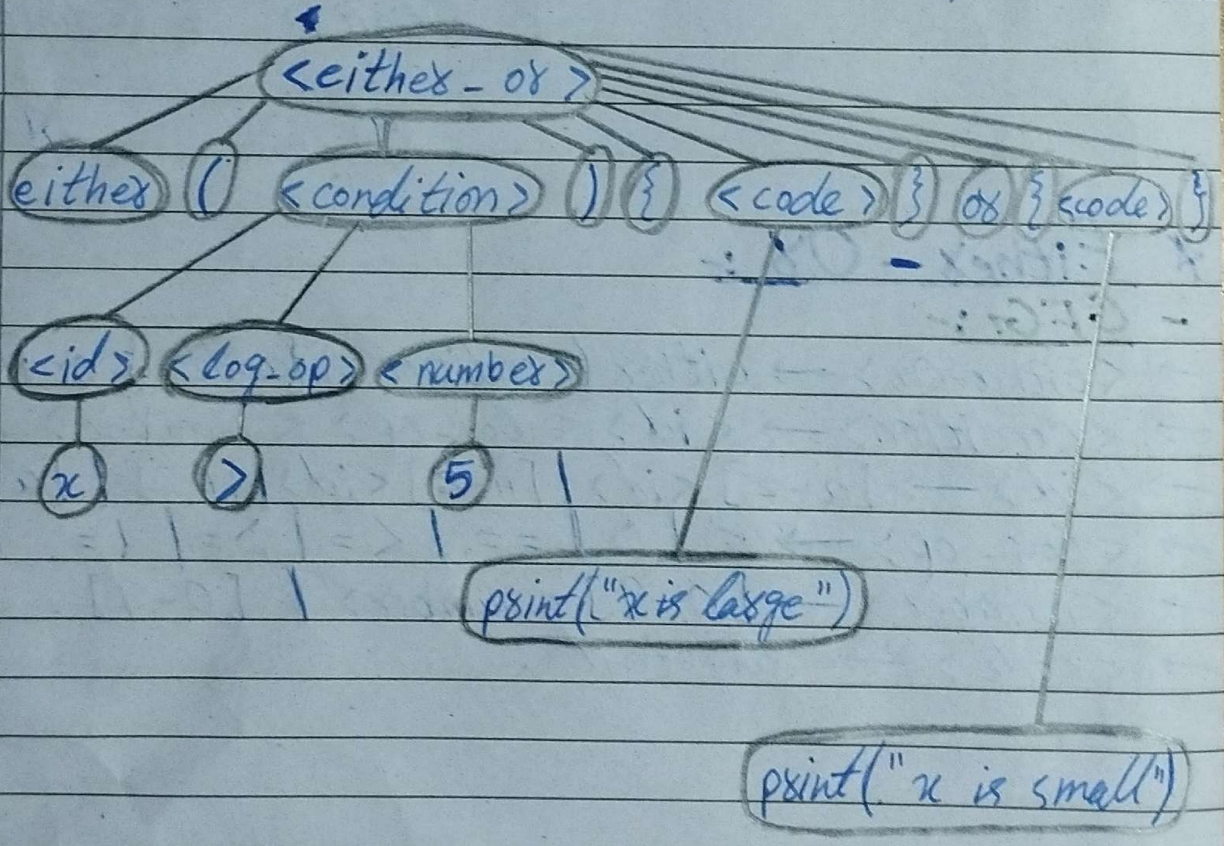
→ either (<id> <log-op><number>) {<code>} or {<code>}.

→ either (x > 5) {<code>} or {<code>}

→ either (x >5) {print("x is large")} or {print("x is small)}.

- Parse Tree :-

String=> either (x >5) {print("x is large")} or {print("x is small")}.



<either_or>
either
(
<condition>
)
{
<code>
}
or
{
<code>
}
<id>
<log-op>
<number>
x
>
5
print("x is large")
print(" x is small")