CFGS OF LOOP AND CONDITIONAL STATEMENT:

<loop> -> iter(<initialize_statement> ;<condition>;<counter>) { <body> }

<initialize_statement> ->DT ID = <expression>

<condition> -> digit <cond_operator> ID | ID <cond_operator> digit |ID <cond_operator> ID

<counter> -> ID=ID+1 | ID=ID-1

<cond_operator> -> == | '<' | '<=' | '>' | '>=' | not | and | or

<conditional_statement> -> if (<condition>) { <body>}| elif (<condition>) { <body>}
| else { <body>}

<body> -> <statement> | <body> <statement>
<statement> -> <initialize_statement> <statement> | <counter> <statement>| <loop> <statement> |
<conditional_statement> <statement> | <print_statement> <statement>| <input_statement> <statement> |
null

<input_statement> -> input (ID)

<print_statement> -> print(ID) | print(digit)

<expression> -> ID <new> | digit<new>
<new> -> <arithmetic_operator> ID <new> | <arithmetic_operator> digit <new> | null

<arithmetic_operator> -> '+' | '-' |'*' | '/'

Example:

```
# Define an iterative loop with initialization, condition, and counter
iter(num #i = 0; #i < 5; #i = #i + 1) {
   # Inside the loop body
   # Initialize a variable j
   DT #j = #i * 2
   # Check a condition
   if (#j >= 5) {
      print(#j)
   }
}
```

LEFT MOST DERIVATION :

<loop> -> iter(<initialize_statement> ; <condition>; <counter>) { <body> }
<initialize_statement> ->iter(num #i = <expression> ; <condition>; <counter>) { <body> }
<expression> -> iter(num #i = 0 <new> ; <condition>; <counter>) { <body> }

<new> -> iter(num #i = 0; <condition>; <counter>) { <body> }
<condition> -> iter(num #i = 0; #i <cond_operator> digit ; <counter>) { <body> }
<cond_operator> -> iter(num #i = 0; #i < digit ; <counter>) { <body> }
<condition> -> iter(num #i = 0; #i < 5; <counter>) { <body> }
<counter> -> iter(num #i = 0; #i < 5; #i = #i + 1){<body> }
<body> -> iter(num #i = 0; #i < 5; #i = #i + 1){<statement> }
<statement> -> iter(num #i = 0; #i < 5; #i = #i + 1){ <initialize_statement> <statement> }
<initialize_statement> -> iter(num #i = 0; #i < 5; #i = #i + 1){ num #j = <expression> <statement> }
<expression> -> iter(num #i = 0; #i < 5; #i = #i + 1){ num #j = #i <new> <statement> }
<new> ->iter(num #i = 0; #i < 5; #i = #i + 1){ num #j = #i <arithmetic_operator> digit <new> <statement> }
<arithmetic_operator> -> iter(num #i = 0; #i < 5; #i = #i + 1){ num #j = #i * 2 <statement> }
<statement>-> iter(num #i = 0; #i < 5; #i = #i + 1){ num #j = #i * 2 <conditional_statement> <statement> }
<conditional_statement> -> iter(num #i = 0; #i < 5; #i = #i + 1){ num #j = #i * 2 if (<condition>) { <body>}
<statement> }
<condition> -> iter(num #i = 0; #i < 5; #i = #i + 1){ num #j = #i * 2 if (#j <cond_operator> digit) { <body>}
<statement> }
<cond_operator> -> iter(num #i = 0; #i < 5; #i = #i + 1){ num #j = #i * 2 if (#j >= 5) { <body> } <statement> }
<body> -> <cond_operator> == iter(num #i = 0; #i < 5; #i = #i + 1){ num #j = #i * 2 if (#j >= 5) { <statement>
} <statement> }
<statement> -> iter(num #i = 0; #i < 5; #i = #i + 1){ num #j = #i * 2 if (#j >= 5) { <print_statement>
<statement> } <statement> }
<print_statement> -> iter(num #i = 0; #i < 5; #i = #i + 1){ num #j = #i * 2 if (#j >= 5) { print(#j) } }

RIGHT MOST DERIVATION:

<loop> -> iter(<initialize_statement> ; <condition>; <counter>) { <body> }
<body> -> iter(<initialize_statement> ; <condition>; <counter>) { <statement> }
<statement> ->iter(<initialize_statement> ; <condition>; <counter>) { <initialize_statement> <statement> }
<statement> -> iter(<initialize_statement> ; <condition>; <counter>) { <initialize_statement>
<conditional_statement>}
<conditional_statement> -> iter(<initialize_statement> ; <condition>; <counter>) { <initialize_statement>if (
<condition>) { <body>} }
<body> -> iter(<initialize_statement> ; <condition>; <counter>) { <initialize_statement> if (<condition>) {
<statement>} }
<statement> -> iter(<initialize_statement> ; <condition>; <counter>) { <initialize_statement> if (
<condition>) {<print_statement>}}
<print_statement> -> iter(<initialize_statement> ; <condition>; <counter>) { <initialize_statement> if (
<condition>) {print(#j)}}
<condition> -> iter(<initialize_statement> ; <condition>; <counter>) { <initialize_statement> if (ID
<cond_operator> 5) {print(#j)}}
<cond_operator> ->iter(<initialize_statement> ; <condition>; <counter>) { <initialize_statement> if (#j >= 5)
{print(#j)}}
<initialize_statement> ->iter(<initialize_statement> ; <condition>; <counter>) { DT ID = <expression> if (#j
>= 5) {print(#j)}}
<expression> ->iter(<initialize_statement> ; <condition>; <counter>) { DT ID = ID <new> if (#j >= 5)
{print(#j)}}
<new> -> iter(<initialize_statement> ; <condition>; <counter>) { DT ID = ID <arithmetic_operator> 2 if (#j >=
5) {print(#j)}}
<arithmetic_operator> -> iter(<initialize_statement> ; <condition>; <counter>) { num #j = #i * 2 if (#j >= 5)
{print(#j)}}

<counter> -> iter(<initialize_statement> ; <condition>; #i=#i+1) { num #j = #i * 2 if (#j >= 5) {print(#j)}}
<condition> -> iter(<initialize_statement> ;ID <cond_operator> 5 ; #i = #i +1) { num #j = #i * 2 if (#j >= 5) {print(#j)}}
<cond_operator> ->iter(<initialize_statement> ; #i < 5 ; #i = #i +1) { num #j = #i * 2 if (#j >= 5) {print(#j)}}
<initialize_statement> -> iter(DT ID = <expression> ; #i < 5 ; #i = #i +1) { num #j = #i * 2 if (#j >= 5) {print(#j)}}
<expression> -> iter(DT ID = 0 ; #i < 5 ; #i = #i +1) { num #j = #i * 2 if (#j >= 5) {print(#j)}}

Final->iter(num #i = 0 ; #i < 5 ; #i = #i +1) { num #j = #i * 2 if (#j >= 5) {print(#j)}}

CFG OF INPUT STATEMENT:

<input_statement> -> input (ID)

Example:
input(#d)

<input_statement> -> input (#d)

CFGS OF EXPRESSION:

<initialize_statement> -> DT ID = <expression>
<expression> -> ID <new> | digit<new>
<new> -> <arithmetic_operator> ID <new> | <arithmetic_operator> digit <new> | null
<arithmetic_operator> -> '+' | '-' |'*' | '/'

Example:
num #a = 4+5

LEFT DERIVATION:

<initialize_statement> -> DT ID = <expression>
<expression> -> num #a = digit <new>
<new> -> num #a =4 <arithmetic_operator> digit <new>
<arithmetic_operator> -> num #a =4 + 5

RIGHT DERIVATION:

<initialize_statement> -> DT ID = <expression>
<expression> -> DT ID = digit<new>
<new> -> DT ID = digit <arithmetic_operator> digit <new>
<new> -> DT ID = digit <arithmetic_operator> digit
<arithmetic_operator> -> DT ID = digit + 5

Final-> num #a = 4 +5

CFGS OF VARIABLE DECLARATION:

<variable_decl> ->DT # letter <letter> <digit> <underscore> I DT # letter <letter> <digit> I DT # letter <letter>
<letter> -> letter<letter> I <digit> I null
<digit> -> digit<digit> I <letter> I null
<underscore> -> _ <letter> <digit> I _ <digit> <letter>

Example:

num #abc_1d

LEFT DERIVATION:

<variable_decl> ->DT # letter <letter> <digit> <underscore>
<letter> -> num #a letter<letter> <digit> <underscore>
<letter> -> num #ab letter<letter> <digit> <underscore>
<underscore> -> num #abc _ <digit> <letter>
<digit> -> num #abc_digit<digit><letter>
<letter> -> num #abc_1 letter<letter>
<letter> -> num #abc_1d

RIGHT DERIVATION:

<variable_decl> ->DT # letter <letter> <digit> <underscore>
<underscore> -> DT # letter <letter> <digit> _ <digit> <letter>
<letter> ->D T # letter <letter> <digit> _ <digit> letter<letter>
<letter> ->D T # letter <letter> <digit> _ <digit> d
<digit> ->D T # letter <letter> <digit> _ digit<digit> d
<digit> ->D T # letter <letter> <digit> _1d
<digit> ->D T # letter <letter> _1d
<letter> -> DT # letter letter <letter>_1d
 <letter> -> DT # letter letter letter <letter>_1d
<letter> -> DT #letter letter letter _1d

Final -> num #abc_1d

CFGS OF VARIABLE INITIALIZATION:

<variable_initial> -> <variable_decl> = <variable_decl> I <variable_decl> = <value>
<value> -> <sign> digit <digit> <decimal> <exponent>
<exponent> -> e <value> I E <value> I null
<sign> -> + I – I null
<decimal> -> . digit <digit> I null
<variable_decl> ->DT # letter <letter> <digit> <underscore> I DT # letter <letter> <digit> I DT # letter <letter>
<letter> -> letter<letter> I <digit> I null
<digit> -> digit<digit> I <letter> I null
<underscore> -> _ <letter> <digit> I _ <digit> <letter>

Example:
num #ab = 9.5 e -10

LEFT DERIVATION:

<variable_initial> -> <variable_decl> = <value>
<variable_decl> -> DT # letter <letter> = <value>
<letter> -> num # a letter<letter> = <value>
<letter> -> num # ab = <value>
<value> -> num # ab = 9 <decimal> <exponent>
<decimal> -> num # ab = 9. digit <digit> <exponent>
<decimal> -> num # ab = 9.5<exponent>
<exponent> -> num # ab = 9.5 e <value>
<value> -> num # ab = 9.5 e <sign> digit <digit> <decimal> <exponent>
<sign> -> num # ab = 9.5 e – 1 <digit> <decimal> <exponent>
<digit>-> num # ab = 9.5 e – 1 digit<digit> <decimal> <exponent>
<digit>-> num # ab = 9.5 e – 10

RIGHT DERIVATION:

<variable_initial> -> <variable_decl> = <value>
<value> -> <variable_decl> =<sign> digit <digit> <decimal> <exponent>
<exponent> -> <variable_decl>= <sign> digit <digit> <decimal> e <value>
 <value> -> <variable_decl>= <sign> digit <digit> <decimal> e <sign> digit <digit>
 <digit> -> <variable_decl>= <sign> digit <digit> <decimal> e <sign>10
<sign> -> <variable_decl>= <sign> digit <digit> <decimal> e-10
<decimal> -> <variable_decl>= <sign> digit <digit>. digit e-10
<digit> -> <variable_decl>= 9.5 e-10
<variable_decl> -> DT # letter <letter> = 9.5 e-10
<letter> -> DT # letter<letter> = 9.5 e-10
<letter> -> DT # letter letter= 9.5 e-10

Final -> num #ab = 9.5e-10