socrative

Name _____
Date _____

PLC21-mT3

Score _____

1. Considere os Terminais **str** (texto entre aspas), **texto** (sequência de carateres) e **id** (sequência de letras) e a seguinte Gramática Independente de Contexto (**G**)
   1. **Anota => Abre texto Fecha**
   2. **Abre => '<' id '>'**
   3. **| '<' id LstA '>'**
   4. **Fecha => '<' '/' id '>'**
   5. **LstA => Atr**
   6. **LstA => LstA Atr**
   7. **Atr => id '=' str**

   Selecione então a alínea abaixo que é uma afirmação verdadeira:

(A) a frase
<ttt="vv" o=12>bla bla</ttt>
pertence à linguagem L(G) gerada por esta gramática.

(B) a frase
<ttt a="1">bla bla</ttt>
pertence à linguagem L(G) gerada por esta gramática.

(C) a frase
<ttt a="1"><id>bla bla</id></ttt>
pertence à linguagem L(G) gerada por esta gramática.

(D) a frase
<ttt>ola ole oli</z>
não pertence à linguagem L(G) gerada por esta gramática.

2. Considere os Terminais **nint** (número inteiro), **nreal** (número decimal) e **pal** (sequência de uma ou mais letras) e a
seguinte Gramática Independente de Contexto (**G**)

Frase => '[' Elems ']'
Elems => €
Elems => Elem Elems
Elem => nint
| nreal
| pal
| Frase

Selecione então a alínea abaixo que é uma afirmação verdadeira:

(A) para a frase
9 0.2 abs
pertencer à linguagem L(G) gerada por esta gramática era obrigatório estar envolvida em parêntesis retos .

(B) a lista **vazia** (**[]**) não pertence à linguagem L(G) gerada por esta gramática.

(C) a frase **[[[ABC]]]**
não pertence à linguagem L(G) gerada por esta gramática.

(D) a frase **[9.1 [2 [a] 3 [43.1 88]**
pertence à linguagem L(G) gerada por esta gramática.

3. Considere o conjunto de Símbolos Terminais
T={ a, f, d, p, i }
e as seguintes frases válidas de uma linguagem **L**

a i f
a d p d i f
a i p i p i f
a d i p i f

Selecione então a alínea abaixo que é uma afirmação verdadeira:

(A) **L** pode ser gerada pela seguinte GIC:

```
Frase  => a Corpo f
Corpo  => Ds Is
Ds     => €
Ds     => Dmais
Dmais  => d
        | Dmais p d
Is     => i Resto
Resto  => €
        | p i Resto
```

(B) **L** pode ser gerada pela seguinte GIC:

```
Frase  => a Corpo f
Corpo  => Ds Is
Ds     => €
Ds     => Ds p d
Is     => i Resto
Resto  => €
        | p i Resto
```

(C) **L** pode ser gerada pela seguinte GIC:

```
Frase => a Corpo f
Corpo => Ds Is
Ds    => €
Ds    => Dmais
Dmais => d
       | Dmais p d
Is    => i p
       | Is i
```

(D) **L** pode ser gerada pela seguinte GIC:

```
Frase  => a Corpo f
Corpo  => Ds Is
Ds     => d
        | d p Ds
Is     => i
        | Is p i
```

4. Observe com atenção o seguinte filtro de texto implementado com o LEX do Python:

```
import ply.lex as lex
import sys
states = (('marcado','inclusive'),)
tokens = ( 'MA', 'MF', 'MARCA', 'IGN' )
def t_MA(t):
  r'\<[^>]+\>'
  t.lexer.begin('marcado')
def t_IGN(t):
  r'.|\n'
t_ignore  = ' \t\r\n'
def t_error(t):
  t.lexer.skip(1)
def t_marcado_MF(t):
  r'\<\/[^>]+\>'
  print(lexer.conteudo)
  lexer.conteudo =  ""
  t.lexer.begin('INITIAL')
def t_marcado_MARCA(t):
  r'.|\n'
  lexer.conteudo += t.value
t_marcado_ignore  = ""

lexer = lex.lex()
lexer.conteudo = ""
for linha in sys.stdin:
  lexer.input(linha)
  tok = lexer.token()
  while tok:
    tok = lexer.token()
```

Selecione então a alínea abaixo que é uma afirmação verdadeira:

(A) Se o texto de entrada for
agora <aqui>sim vai</aqui> fica <ab>123 ola 456</ab>
O resultado será formado por 2 linhas uma com "**sim vai**" e a outra com "**123 ola 456**".

(B) Se o texto de entrada for
agora <aqui>sim vai fica
O resultado será uma **frase vazia** (linha sem carateres).

(C) Se o texto de entrada for
agora <aqui>sim vai<ab> adeus</ab>fica</aqui> fecho
O resultado será formado por 3 linhas uma com "**sim vai**" , outra com " **adeus**" e a outra com "**fica**".

(D) Se o texto de entrada for
agora <aqui atr="8">sim vai fica

O resultado seria uma **frase vazia** (linha sem carateres).

5. Observe com atenção o seguinte filtro de texto implementado com o Lex do Python

```
import ply.lex as lex
import sys
states = (('marcado','inclusive'),)
tokens = ( 'MA', 'MF', 'MARCA', 'IGN' )
def t_MA(t):
  r'\<[^>]+\>'
  t.lexer.begin('marcado')
def t_IGN(t):
  r'.|\n'
t_ignore  = ' \t\r\n'
def t_error(t):
  t.lexer.skip(1)
def t_marcado_MF(t):
  r'\<\/[^>]+\>'
  print(lexer.conteudo)
  lexer.conteudo = ""
  t.lexer.begin('INITIAL')
def t_marcado_MARCA(t):
  r'.|\n'
  lexer.conteudo += t.value
t_marcado_ignore  = ''
lexer = lex.lex()
lexer.conteudo = ""
for linha in sys.stdin:
  lexer.input(linha)
  tok = lexer.token()
  while tok:
    tok = lexer.token()
```

Selecione então a alínea abaixo que é uma afirmação verdadeira:

(A) pode dizer-se que '**ply.lex**' é um processador que lê a especificação no topo do programa e gera um objeto (nesta caso '**lexer**') que é um Analisador Léxico.

(B) se na função '**t_MA**' definida a partir de '**def t_MA(t):**' a expressão regular fosse alterada para **r'\<[a-zA-Z]+\>'**
o comportamento do programa não se alterava.

(C) o compilador de Python assinala erro porque o construtor '**t_ignore**' está definido 2 vezes de forma diferente, uma para o estado inicial e outra para o estado "**marcado**".

(D) a instrução
lexer.conteudo = ""
que se encontra no corpo de
def t_marcado_MF(t):
pode ser retirada sem afetar a saída produzida pelo programa.

6. Considere o seguinte analisador léxico incompleto (não listadas as partes, prólogo e epílogo, habituais):

```
....
tokens = ( 'INI', 'FIM', 'nome', 'real', 'int', 'sinal' )

t_sinal = r'[=+\-*/()]'
def t_INI(t):
   r'(?i:begin)|\{'
   return t
def t_FIM(t):
   r'\}|[eE][nN][dD]'
   return t
def t_nome(t):
   r'[a-zA-Z][a-zA-Z0-9]*'
   return t
def t_real(t):
   r'[0-9]+\.[0-9]+'
   return t
def t_int(t):
   r'[0-9]+'
   return t
t_ignore = " \n\t"
def t_error(t):
   t.lexer.skip(1)

lexer = lex.lex()
....
```

Selecione então a alínea abaixo que é uma afirmação verdadeira:

(A) Se a sequência de símbolos retornados for:
sinal nome sinal nome sinal int sinal
o texto de entrada pode ser
(INI = end + 4)

(B) Se a sequência de símbolos retornados for:
int sinal real sinal nome
o texto de entrada pode ser **4*5.6 = FIM**

(C) Se o texto de entrada for:
{ .x == ab12-End }
a sequência de símbolos retornados é:
INI nome sinal nome int sinal FIM FIM

(D) Se o texto de entrada for.
a1.5/12-5.
a sequência de símbolos retornados é:
nome real sinal int sinal real