UNIVERSIDAD DE PUERTO RICO

RECINTO UNIVERSITARIO DE MAYAGÜEZ

COMP 4036-070

PARSERTONGUE

Carlos Lopez Sosa

Alejandro Proskauer

Aryam Yulfo Soto

Prof. Alcibiades Bustillo

May 14, 2021

INTRODUCTION

¨Modern problems require modern solutions¨ is a quote we hear often and it holds many truths. In this rapidly changing world we must come up with solutions that not only are fast but efficient and accurate as well. This was the motivation behind our programming language Parsertongue.

Like described before, our language will be capable of solving a system of equations as fast and efficiently as possible. Using Python Programming as our base language we created our own unique language that will help us solve multivariable matrices.

The website for Parsertongue can be found here:

https://amprosk.github.io/COMP4036-Project-Proskauer-Yulfo-Lopez/

LANGUAGE TUTORIAL

In order to get the results of the matrices inputted by the user, the user must correctly input the system of equations. The user must include the symbol ¨&¨ after each equation. When the user is done inputting the equations, the user must include the ¨#¨ symbol at the end of his equation. After this, our language will give out the correct answer in seconds.

A correct user input would look like this:

```
Parsertongue > 2x +0y -  z = 4 & 3x - 2y + 3z = 4 & 7x - 7y + 9z = 8 #
```

An incorrect user input would look like this:

```
Parsertongue > 2x + 0y - z = 4 3x - 2y + 3z = 4 & 7x - 7y + 9z = 8
```

This user input will give a syntax error since it is missing an ¨&¨ symbol.

In situations where the system is unsolvable the language will be able to tell this and tell us that the system cannot be solved.

Example:

```
Parsertongue > 4x+0y=4 & 5x+0y=3 #
Equation 1: (4)x + (0)y = 4
Equation 2: (5)x + (0)y = 3
ERROR OCCURRED, THIS SYSTEM OF EQUATIONS IS NOT SOLVABLE
```

```
Parsertongue > 5x-3y+0z = 1 & -x + 1.2y -0z = -2.4 & 8.6x - 69y +0z = 32#
Equation 1: (5.0)x + (-3.0)y + (0.0)z = 1.0
Equation 2: (-1.0)x + (1.2)y + (0.0)z = -2.4
Equation 3: (8.6)x + (-69.0)y + (0.0)z = 32.0
ERROR OCCURRED, THIS SYSTEM OF EQUATIONS IS NOT SOLVABLE
```

LANGUAGE REFERENCE MANUAL

Our language has certain limitations that can be improved on in the future. One of these limitations is that it can't recognize empty values.

For example, if we have the following system of equations:

```
Parsertongue > 2x + z = 4 & 3x - 2y + 3z = 4 & 7x - 7y + 9z = 8 #
```

The language cannot identify the missing value of y in the first equation. We can fix this by inputting a 0 next to the empty variable we want.

Example:

```
Parsertongue > 2x + 0y + z = 4
```

This way the language can correctly compute the values.

Parsertongue cannot identify the specific values if they are out of order.
For example, if the user inputs:

```
Parsertongue > 3x - 2z + 3y = 4 & 2x + y + z = 5.2 & -x + 9y - 4.2z = 8 #
```

The program will output a syntax error since the first equation is out of order.

Instead the input of the first equation should be:

```
Parsertongue > 3x + 3y - 2z = 4
```

This will output the following results:

```
Equation 1: (3.0)x + (3.0)y + (-2.0)z = 4.0

Equation 2: (2.0)x + (1.0)y + (1.0)z = 5.2

Equation 3: (-1.0)x + (9.0)y + (-4.2)z = 8.0

Solutions:

x = 0.7379061371841154

y = 1.846931407942239

z = 1.8772563176895314
```

Each leading x term of each equation also needs some type of coefficient or +, - for it to work.

For example, if we have the input:

```
Parsertongue > 3x + 3y - 2z = 4 & x + y + z = 5.2 & -x + 9y - 4.2z = 8 #
```

Will generate a syntax error, instead we need the input:

```
Parsertongue > 3x + 3y - 2z = 4 & +x + y + z = 5.2 & -x + 9y - 4.2z = 8 #
```

or

```
Parsertongue > 3x + 3y - 2z = 4 & 1x + y + z = 5.2 & -x + 9y - 4.2z = 8 #.
```

The output for both cases is:

```
Equation 1: (3.0)x + (3.0)y + (-2.0)z = 4.0

Equation 2: (1.0)x + (1.0)y + (1.0)z = 5.2

Equation 3: (-1.0)x + (9.0)y + (-4.2)z = 8.0

Solutions:

x = 0.8176

y = 2.0624000000000002

z = 2.3200000000000003
```

LANGUAGE DEVELOPMENT

Parsertongue was developed in Python. It uses the SLY (Sly Lex Yacc) library to build the Lexer and Parser for a language. Our language also utilizes modules from the NumPy library.

Translator Architecture

Lexer

The lexer is built as a derived class ptLexer of the built in class Lexer from SLY. The following tokens are defined:

`TOKEN_INT = r'\d+’`

(This is used to identify integers)

`TOKEN_PERIOD = r'\.'`

(This is used to identify a floating point to later interpret decimal numbers)

`TOKEN_SUM = r'\+'`

(This is used to identify when the linear equation involves a positive coefficient or constant)

`TOKEN_MINUS = r'-'`

(This is used to identify when the linear equation involves a negative coefficient or constant)

`TOKEN_IGUAL = r'='`

(This is used to mark when the linear combination of variables is equal to the constant)

`TOKEN_X = r'[xX]'`

(This is used to identify the 1st variable of the linear equation)

`TOKEN_Y = r'[yY]'`

(This is used to identify the 2nd variable of the linear equation)

`TOKEN_Z = r'[zZ]'`

(This is used to identify the 3rd variable of the linear equation)

`NEXT_EQUATION = r'&'`

(This is used to separate between equations in the system)

`END_SYSTEM = r'#'`

(This is used at the end of the system of equations)

Parser

As with the lexer, the parser is built as a derived class ptParser of the built in class Parser from SLY. Before the language can be parsed, arrays to store the coefficients and constants of the system of equations are initialized. The following grammar rules are defined, and each has a related action within the language:

```
S' -> number
number -> TOKEN_INT
```

Returns the value of the integer

```
number -> TOKEN_INT TOKEN_PERIOD TOKEN_INT
```

Returns value of the interpreted floating point number

```
number -> TOKEN_MINUS number
```

Returns value of the number times -1

```
number -> TOKEN_SUM number
```

Returns value of the number

```
number -> x_term
```

Returns value of the coefficient of x

```
number -> y_term
```

Returns value of the coefficient of y

```
number -> z_term
```

Returns value of the coefficient of z

```
number -> system
```

Uses numpy.linalg.solve and the coefficient/constant arrays that have been filled to return a tuple of the coefficients, constants, and solutions. Also checks whether the system is solvable or not. if not, returns array with single element

```
x_term -> TOKEN_MINUS TOKEN_X
```

Assign the coefficient -1 to x

```
x_term -> TOKEN_SUM TOKEN_X
```

Assign the coefficient 1 to x

```
x_term -> number TOKEN_X
```

Assign the coefficient *number* to x

```
y_term -> TOKEN_MINUS TOKEN_Y
```

Assign the coefficient -1 to y

```
y_term -> TOKEN_SUM TOKEN_Y
```

Assign the coefficient 1 to y

```
y_term -> number TOKEN_Y
```

Assign the coefficient *number* to x

```
z_term -> TOKEN_MINUS TOKEN_Z
```

Assign the coefficient -1 to z

```
z_term -> TOKEN_SUM TOKEN_Z
```

Assign the coefficient 1 to z

```
z_term -> number TOKEN_Z
```

Assign the coefficient *number* to z

```
system -> x_term y_term TOKEN_IGUAL number NEXT_EQUATION
          x_term y_term TOKEN_IGUAL number END_SYSTEM
```

Identify a system of 2 equations and assign the corresponding coefficients and constants to the arrays stored by the language.

```
system -> x_term y_term z_term TOKEN_IGUAL number
          NEXT_EQUATION x_term y_term z_term TOKEN_IGUAL
          number NEXT_EQUATION x_term y_term z_term
          TOKEN_IGUAL number END_SYSTEM
```

Identify a system of 3 equations and assign the corresponding coefficients and constants to the arrays stored by the language.

In addition to these grammar rules, the following precedence levels were established to resolve shift/reduce conflicts

```
Level 1 (left): TOKEN_X, TOKEN_Y, TOKEN_Z
Level 2 (left): TOKEN_SUM, TOKEN_MINUS
Level 3 (left): TOKEN_INT, TOKEN_PERIOD
```

Shell

The shell of the language is within the main function of the python file in which the language is written. It asks for input from the user. tokenizes it, then parses it. Then it checks if the output is a tuple. If it is not a tuple, that means it was a test of the number or x/y/z_term grammars so it simply returns the result. If it is a tuple, it unpacks it and looks at the solutions array to determine if a system of 2 or 3 variables was solved, or if the system was unsolvable. Then it prints the system of equations and the corresponding solutions, or an error message if it was unsolvable.

Interfaces Between Modules

The modules interact within the shell component. After receiving the input from the user, the shell uses the built in function from the *Lexer* class from SLY to tokenize the text. Then in the same line the tokenized text is taken as input in the built in function from the *Parser* class from SLY to parse it. Then the relevant result is printed.

Software Development Environment

The programming language was developed using the Anaconda distribution of Python 3.8 using the Spyder IDE. As previously mentioned, the language is built on modules from the SLY library and uses NumPy modules (linalg module and arrays) to solve the system of equations inputted by the user.

Test Methodology

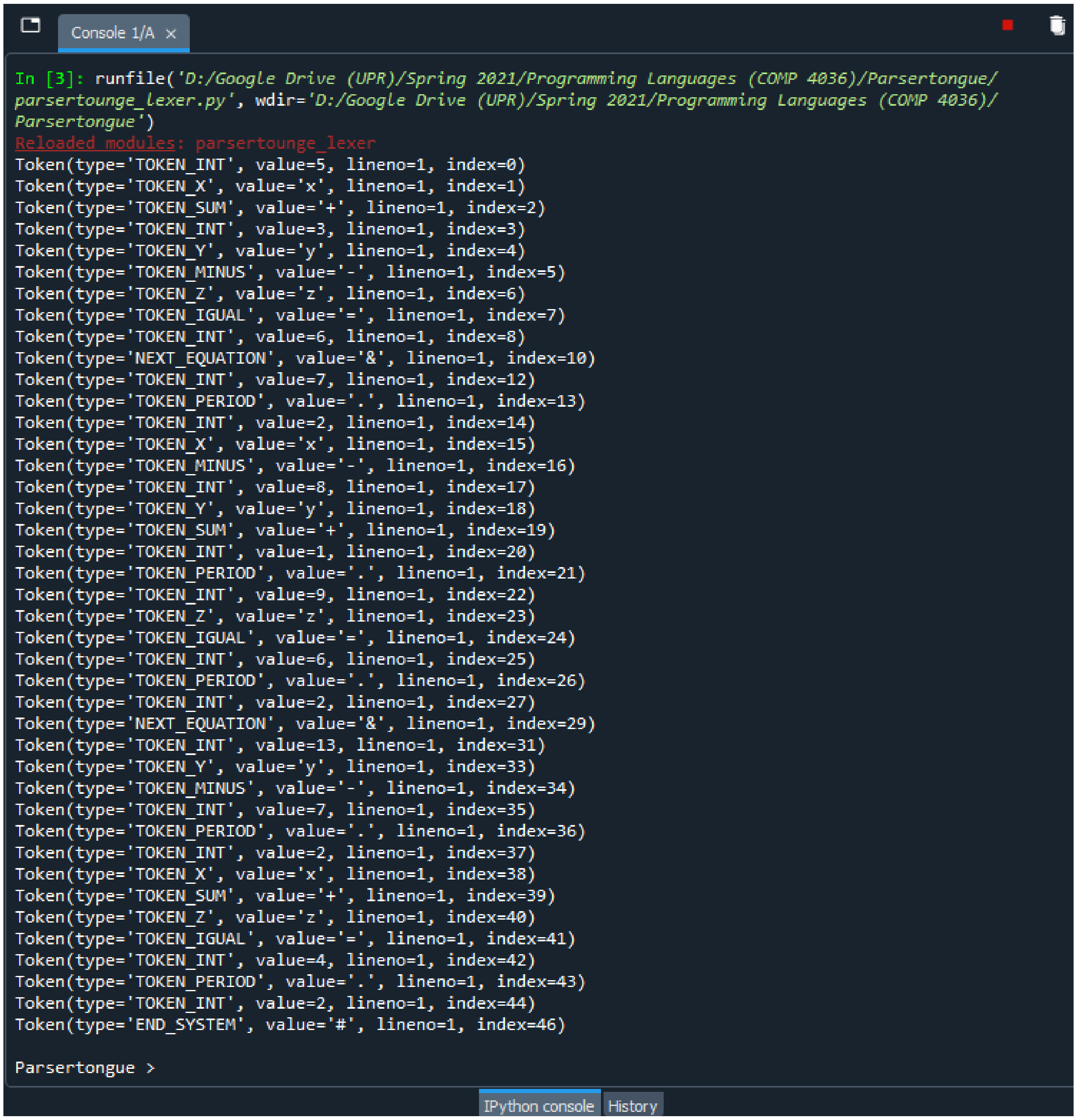
To test the modules of Parsertongue, first the Lexer was tested by itself. This was done by checking if the tokens of a predetermined test input were properly identified, then checking if various user inputs were tokenized correctly. Once the Lexer was verified to work, we focused on testing all the grammar rules involving numbers. We needed to make sure that it was parsing floating point numbers, integers, and negative numbers correctly. Then, the grammar rules involving x/y/z terms were tested to make sure the coefficients were being parsed correctly. Next, the system grammar rules were tested to ensure the systems of equations were being solved and the output was tested first as a simple tuple of arrays and then as a formal statement. Finally, the functionality of displaying whether a system is unsolvable was tested.

Tests

Lexer

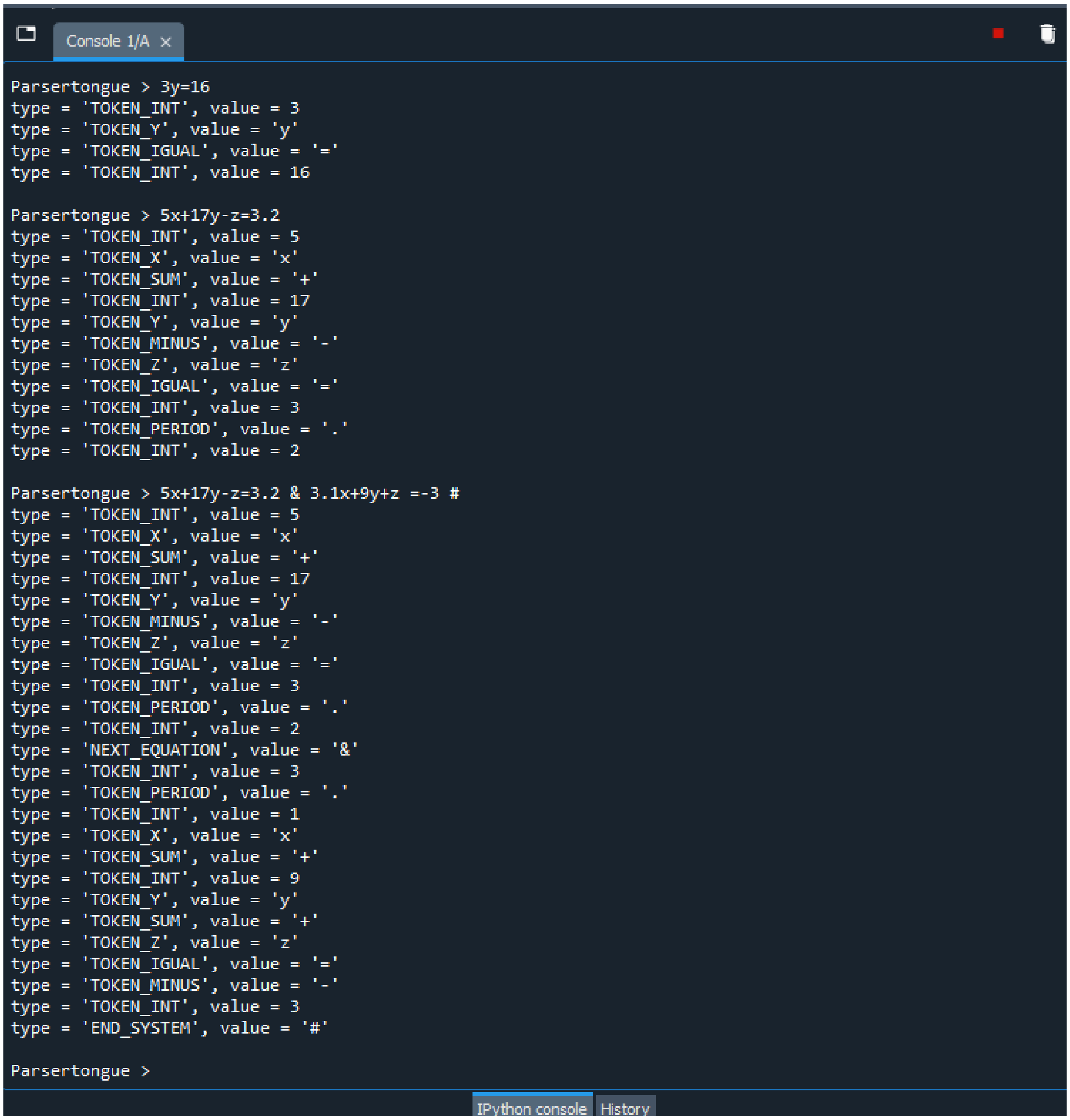
Test input: “5x+3y-z=6 & 7.2x-8y+1.9z=6.2 & 13y-7.2x+z=4.2 #"

Test output:



```
In [3]: runfile('D:/Google Drive (UPR)/Spring 2021/Programming Languages (COMP 4036)/Parsertongue/
parsertounge_lexer.py', wdir='D:/Google Drive (UPR)/Spring 2021/Programming Languages (COMP 4036)/
Parsertongue')
Reloaded modules: parsertounge_lexer
Token(type='TOKEN_INT', value=5, lineno=1, index=0)
Token(type='TOKEN_X', value='x', lineno=1, index=1)
Token(type='TOKEN_SUM', value='+', lineno=1, index=2)
Token(type='TOKEN_INT', value=3, lineno=1, index=3)
Token(type='TOKEN_Y', value='y', lineno=1, index=4)
Token(type='TOKEN_MINUS', value='-', lineno=1, index=5)
Token(type='TOKEN_Z', value='z', lineno=1, index=6)
Token(type='TOKEN_IGUAL', value='=', lineno=1, index=7)
Token(type='TOKEN_INT', value=6, lineno=1, index=8)
Token(type='NEXT_EQUATION', value='&', lineno=1, index=10)
Token(type='TOKEN_INT', value=7, lineno=1, index=12)
Token(type='TOKEN_PERIOD', value='.', lineno=1, index=13)
Token(type='TOKEN_INT', value=2, lineno=1, index=14)
Token(type='TOKEN_X', value='x', lineno=1, index=15)
Token(type='TOKEN_MINUS', value='-', lineno=1, index=16)
Token(type='TOKEN_INT', value=8, lineno=1, index=17)
Token(type='TOKEN_Y', value='y', lineno=1, index=18)
Token(type='TOKEN_SUM', value='+', lineno=1, index=19)
Token(type='TOKEN_INT', value=1, lineno=1, index=20)
Token(type='TOKEN_PERIOD', value='.', lineno=1, index=21)
Token(type='TOKEN_INT', value=9, lineno=1, index=22)
Token(type='TOKEN_Z', value='z', lineno=1, index=23)
Token(type='TOKEN_IGUAL', value='=', lineno=1, index=24)
Token(type='TOKEN_INT', value=6, lineno=1, index=25)
Token(type='TOKEN_PERIOD', value='.', lineno=1, index=26)
Token(type='TOKEN_INT', value=2, lineno=1, index=27)
Token(type='NEXT_EQUATION', value='&', lineno=1, index=29)
Token(type='TOKEN_INT', value=13, lineno=1, index=31)
Token(type='TOKEN_Y', value='y', lineno=1, index=33)
Token(type='TOKEN_MINUS', value='-', lineno=1, index=34)
Token(type='TOKEN_INT', value=7, lineno=1, index=35)
Token(type='TOKEN_PERIOD', value='.', lineno=1, index=36)
Token(type='TOKEN_INT', value=2, lineno=1, index=37)
Token(type='TOKEN_X', value='x', lineno=1, index=38)
Token(type='TOKEN_SUM', value='+', lineno=1, index=39)
Token(type='TOKEN_Z', value='z', lineno=1, index=40)
Token(type='TOKEN_IGUAL', value='=', lineno=1, index=41)
Token(type='TOKEN_INT', value=4, lineno=1, index=42)
Token(type='TOKEN_PERIOD', value='.', lineno=1, index=43)
Token(type='TOKEN_INT', value=2, lineno=1, index=44)
Token(type='END_SYSTEM', value='#', lineno=1, index=46)

Parsertongue >
```

User input:

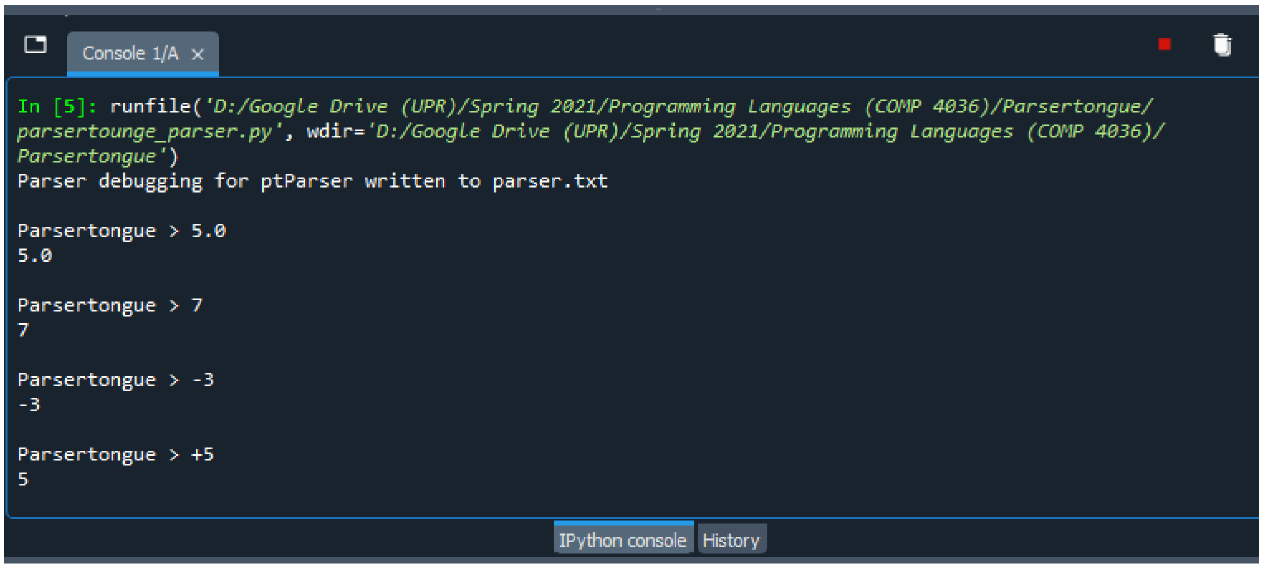


Console 1/A

```
Parsertongue > 3y=16
type = 'TOKEN_INT', value = 3
type = 'TOKEN_Y', value = 'y'
type = 'TOKEN_IGUAL', value = '='
type = 'TOKEN_INT', value = 16

Parsertongue > 5x+17y-z=3.2
type = 'TOKEN_INT', value = 5
type = 'TOKEN_X', value = 'x'
type = 'TOKEN_SUM', value = '+'
type = 'TOKEN_INT', value = 17
type = 'TOKEN_Y', value = 'y'
type = 'TOKEN_MINUS', value = '-'
type = 'TOKEN_Z', value = 'z'
type = 'TOKEN_IGUAL', value = '='
type = 'TOKEN_INT', value = 3
type = 'TOKEN_PERIOD', value = '.'
type = 'TOKEN_INT', value = 2

Parsertongue > 5x+17y-z=3.2 & 3.1x+9y+z =-3 #
type = 'TOKEN_INT', value = 5
type = 'TOKEN_X', value = 'x'
type = 'TOKEN_SUM', value = '+'
type = 'TOKEN_INT', value = 17
type = 'TOKEN_Y', value = 'y'
type = 'TOKEN_MINUS', value = '-'
type = 'TOKEN_Z', value = 'z'
type = 'TOKEN_IGUAL', value = '='
type = 'TOKEN_INT', value = 3
type = 'TOKEN_PERIOD', value = '.'
type = 'TOKEN_INT', value = 2
type = 'NEXT_EQUATION', value = '&'
type = 'TOKEN_INT', value = 3
type = 'TOKEN_PERIOD', value = '.'
type = 'TOKEN_INT', value = 1
type = 'TOKEN_X', value = 'x'
type = 'TOKEN_SUM', value = '+'
type = 'TOKEN_INT', value = 9
type = 'TOKEN_Y', value = 'y'
type = 'TOKEN_SUM', value = '+'
type = 'TOKEN_Z', value = 'z'
type = 'TOKEN_IGUAL', value = '='
type = 'TOKEN_MINUS', value = '-'
type = 'TOKEN_INT', value = 3
type = 'END_SYSTEM', value = '#'

Parsertongue >
```

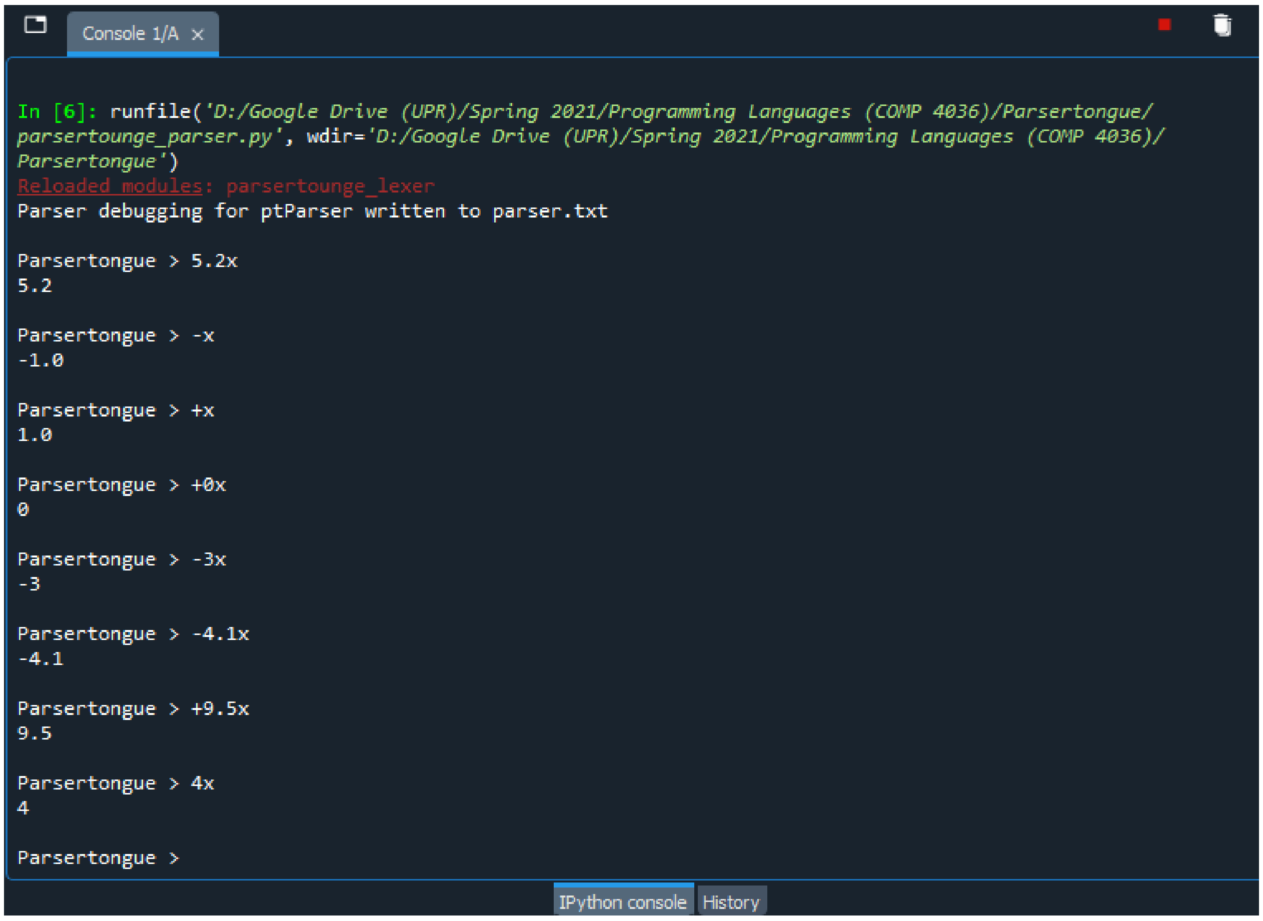
Parser:

Testing *number* grammars



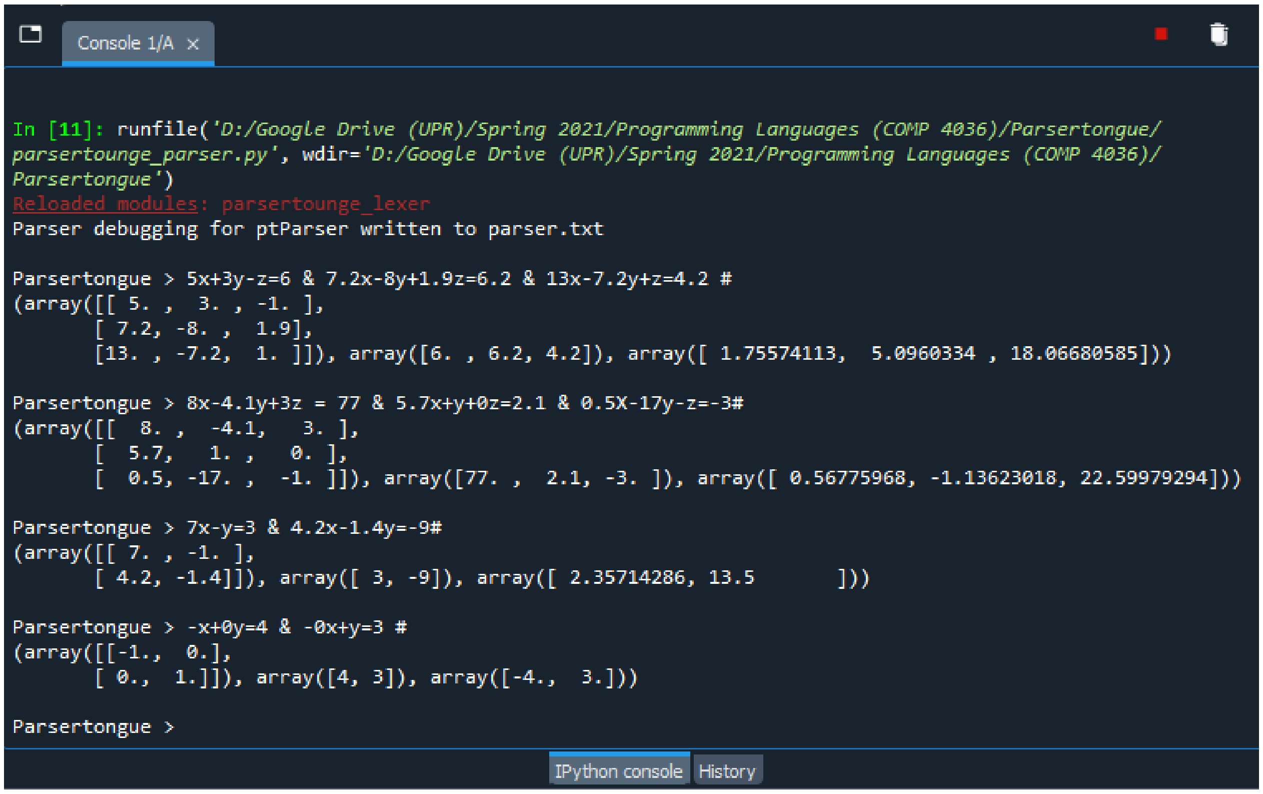
```
In [5]: runfile('D:/Google Drive (UPR)/Spring 2021/Programming Languages (COMP 4036)/Parsertongue/
parsertounge_parser.py', wdir='D:/Google Drive (UPR)/Spring 2021/Programming Languages (COMP 4036)/
Parsertongue')
Parser debugging for ptParser written to parser.txt

Parsertongue > 5.0
5.0

Parsertongue > 7
7

Parsertongue > -3
-3

Parsertongue > +5
5
```

x/y/z_term grammars:



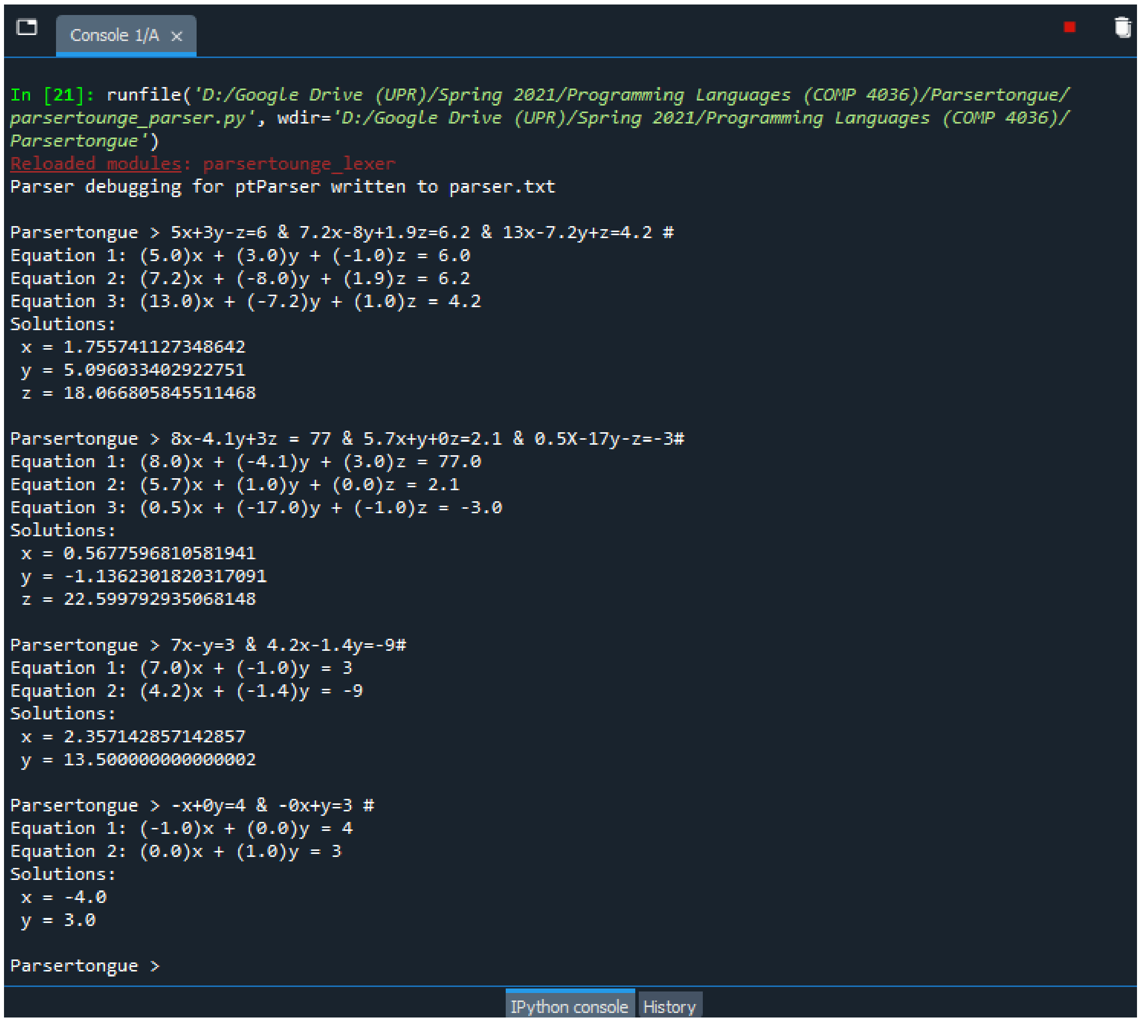
```
In [6]: runfile('D:/Google Drive (UPR)/Spring 2021/Programming Languages (COMP 4036)/Parsertongue/
parsertounge_parser.py', wdir='D:/Google Drive (UPR)/Spring 2021/Programming Languages (COMP 4036)/
Parsertongue')
Reloaded modules: parsertounge_lexer
Parser debugging for ptParser written to parser.txt

Parsertongue > 5.2x
5.2

Parsertongue > -x
-1.0

Parsertongue > +x
1.0

Parsertongue > +0x
0

Parsertongue > -3x
-3

Parsertongue > -4.1x
-4.1

Parsertongue > +9.5x
9.5

Parsertongue > 4x
4

Parsertongue >
```

solutions to systems (outputting as tuples):



```
In [11]: runfile('D:/Google Drive (UPR)/Spring 2021/Programming Languages (COMP 4036)/Parsertongue/
parsertounge_parser.py', wdir='D:/Google Drive (UPR)/Spring 2021/Programming Languages (COMP 4036)/
Parsertongue')
Reloaded modules: parsertounge_lexer
Parser debugging for ptParser written to parser.txt

Parsertongue > 5x+3y-z=6 & 7.2x-8y+1.9z=6.2 & 13x-7.2y+z=4.2 #
(array([[ 5. ,  3. , -1. ],
       [ 7.2, -8. ,  1.9],
       [13. , -7.2,  1. ]]), array([6. , 6.2, 4.2]), array([ 1.75574113,  5.0960334 , 18.06680585]))

Parsertongue > 8x-4.1y+3z = 77 & 5.7x+y+0z=2.1 & 0.5X-17y-z=-3#
(array([[  8. ,  -4.1,   3. ],
       [  5.7,   1. ,   0. ],
       [  0.5, -17. ,  -1. ]]), array([77. ,  2.1, -3. ]), array([ 0.56775968, -1.13623018, 22.59979294]))

Parsertongue > 7x-y=3 & 4.2x-1.4y=-9#
(array([[ 7. , -1. ],
       [ 4.2, -1.4]]), array([ 3, -9]), array([ 2.35714286, 13.5       ]))

Parsertongue > -x+0y=4 & -0x+y=3 #
(array([[-1.,  0.],
       [ 0.,  1.]]), array([4, 3]), array([-4.,  3.]))

Parsertongue >
```

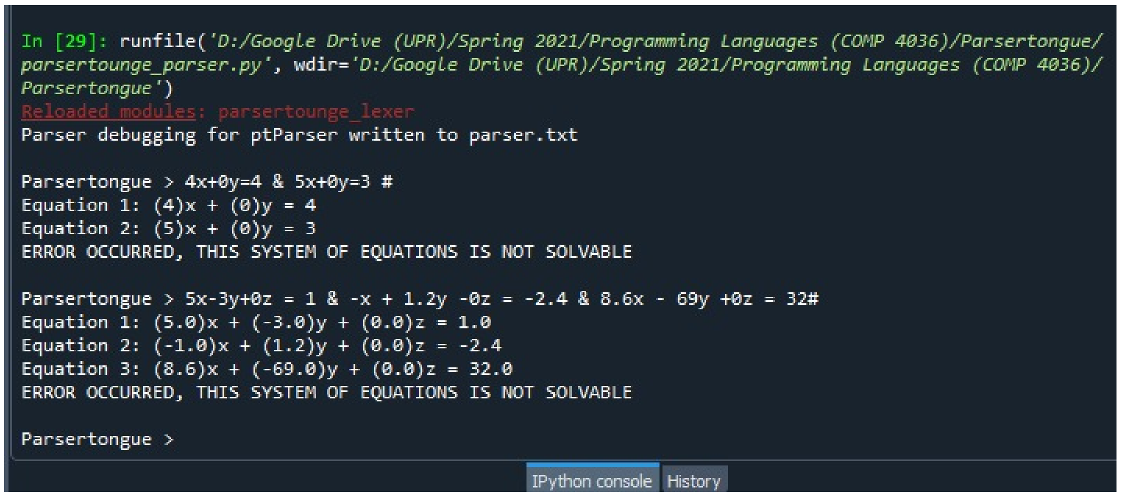
solutions to systems formally written:



Console 1/A

```
In [21]: runfile('D:/Google Drive (UPR)/Spring 2021/Programming Languages (COMP 4036)/Parsertongue/
parsertounge_parser.py', wdir='D:/Google Drive (UPR)/Spring 2021/Programming Languages (COMP 4036)/
Parsertongue')
Reloaded modules: parsertounge_lexer
Parser debugging for ptParser written to parser.txt

Parsertongue > 5x+3y-z=6 & 7.2x-8y+1.9z=6.2 & 13x-7.2y+z=4.2 #
Equation 1: (5.0)x + (3.0)y + (-1.0)z = 6.0
Equation 2: (7.2)x + (-8.0)y + (1.9)z = 6.2
Equation 3: (13.0)x + (-7.2)y + (1.0)z = 4.2
Solutions:
 x = 1.755741127348642
 y = 5.096033402922751
 z = 18.066805845511468

Parsertongue > 8x-4.1y+3z = 77 & 5.7x+y+0z=2.1 & 0.5X-17y-z=-3#
Equation 1: (8.0)x + (-4.1)y + (3.0)z = 77.0
Equation 2: (5.7)x + (1.0)y + (0.0)z = 2.1
Equation 3: (0.5)x + (-17.0)y + (-1.0)z = -3.0
Solutions:
 x = 0.5677596810581941
 y = -1.1362301820317091
 z = 22.599792935068148

Parsertongue > 7x-y=3 & 4.2x-1.4y=-9#
Equation 1: (7.0)x + (-1.0)y = 3
Equation 2: (4.2)x + (-1.4)y = -9
Solutions:
 x = 2.357142857142857
 y = 13.500000000000002

Parsertongue > -x+0y=4 & -0x+y=3 #
Equation 1: (-1.0)x + (0.0)y = 4
Equation 2: (0.0)x + (1.0)y = 3
Solutions:
 x = -4.0
 y = 3.0

Parsertongue >
```

Unsolvable equations:



```
In [29]: runfile('D:/Google Drive (UPR)/Spring 2021/Programming Languages (COMP 4036)/Parsertongue/
parsertounge_parser.py', wdir='D:/Google Drive (UPR)/Spring 2021/Programming Languages (COMP 4036)/
Parsertongue')
Reloaded modules: parsertounge_lexer
Parser debugging for ptParser written to parser.txt

Parsertongue > 4x+0y=4 & 5x+0y=3 #
Equation 1: (4)x + (0)y = 4
Equation 2: (5)x + (0)y = 3
ERROR OCCURRED, THIS SYSTEM OF EQUATIONS IS NOT SOLVABLE

Parsertongue > 5x-3y+0z = 1 & -x + 1.2y -0z = -2.4 & 8.6x - 69y +0z = 32#
Equation 1: (5.0)x + (-3.0)y + (0.0)z = 1.0
Equation 2: (-1.0)x + (1.2)y + (0.0)z = -2.4
Equation 3: (8.6)x + (-69.0)y + (0.0)z = 32.0
ERROR OCCURRED, THIS SYSTEM OF EQUATIONS IS NOT SOLVABLE

Parsertongue >
```

CONCLUSION

Creating a programming language is a challenging project. You need to know how Lexers and Parsers are defined in order to make them work together. It was interesting to learn about all the aspects a Lexer and a Parser have so they can essentially create a new language. Once we got the Lexer to work with the Parser the rest was easier to manage. Documenting about Parsertongue was an essential part of this project. It will help others understand our work and hopefully find it useful for their own projects. As a group each individual's strong suits were tested for this project and at the end the results were great.

USED LIBRARIES

https://docs.anaconda.com/anaconda/

https://numpy.org/doc/

https://sly.readthedocs.io/en/latest/sly.html