ETHAN LAJEUNESSE PRESENTS:

ETHANET FINAL

LEXER AND PARSER GENERATORS

- Jison (JavaScript)
- Jison Features
  - Lexer AND Parser generation
  - Define regular expressions for Jison to find tokens with
  - Then define grammar for Jison to generate an AST with
- To use jison, create a .jison file and define the lexer and parser rules.
  - Then use the Jison command line tools to generate the parser and lexer in one javascript file

DATA TYPES

- Strings
- Numbers – Can be integers or decimal numbers
- Booleans – Yes or no values
- Arrays – Can hold anything

TOKEN TYPES

STRING_LITERAL	"string", 'string'	TRUE	yes	GREATERTHANEQUAL	>=
VAR	ethan	FALSE	no	AND	&&
IDENTIFIER	Variables	IF	if	OR	||
NUMBER	1, 2, 3.5, …	ELSE	else	NOT	!
LBRACE	{	ADD	+	FOR	for
RBRACE	}	SUBTRACT	-	WHILE	while
LPAREN	(	MULT	*	BREAK	break
RPAREN	)	DIV	/	CONTINUE	continue
LBRACKET	[	MOD	%	FUNCTION	laj
RBRACKET	]	INCREMENT	++		
COMMA	,	DECREMENT	--		
SEMI	;	EQUAL	==		
RETURN	return	NOTEQUAL	!=		
NULL	null	LESSTHAN	<		
UNDEFINED	undefined	LESSTHANEQUAL	<=		
EOF	End of file	GREATERTHAN	>		

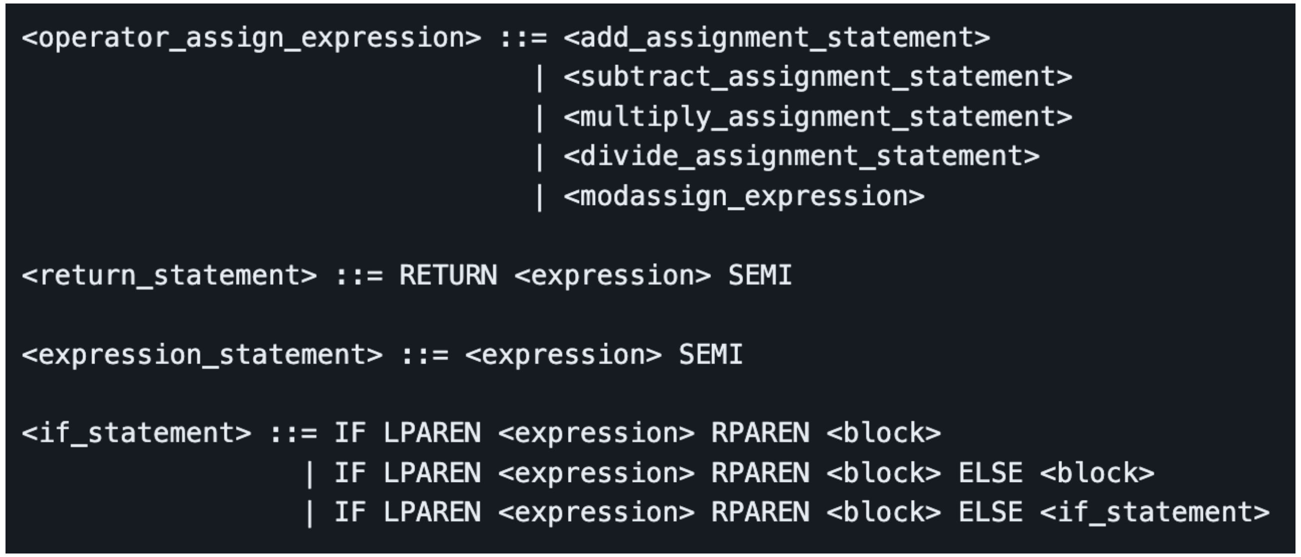
INTRODUCING…

ADDASSIGN	+=
SUBTRACTASSIGN	-=
MULTASSIGN	*=
DIVASSIGN	/=
MODASSIGN	%=
APPROXEQUAL	~=

GRAMMAR

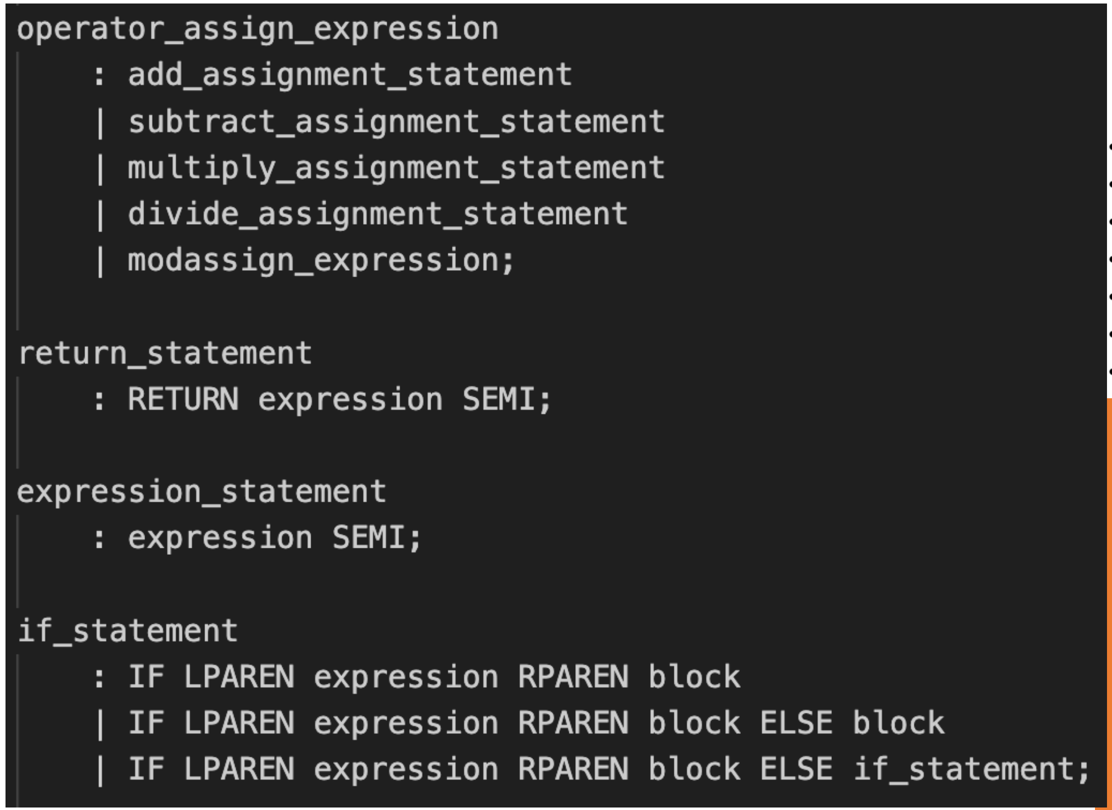
- Language grammar is defined in a BNF-like format

BNF



```
<operator_assign_expression> ::= <add_assignment_statement>
                               | <subtract_assignment_statement>
                               | <multiply_assignment_statement>
                               | <divide_assignment_statement>
                               | <modassign_expression>

<return_statement> ::= RETURN <expression> SEMI

<expression_statement> ::= <expression> SEMI

<if_statement> ::= IF LPAREN <expression> RPAREN <block>
                 | IF LPAREN <expression> RPAREN <block> ELSE <block>
                 | IF LPAREN <expression> RPAREN <block> ELSE <if_statement>
```

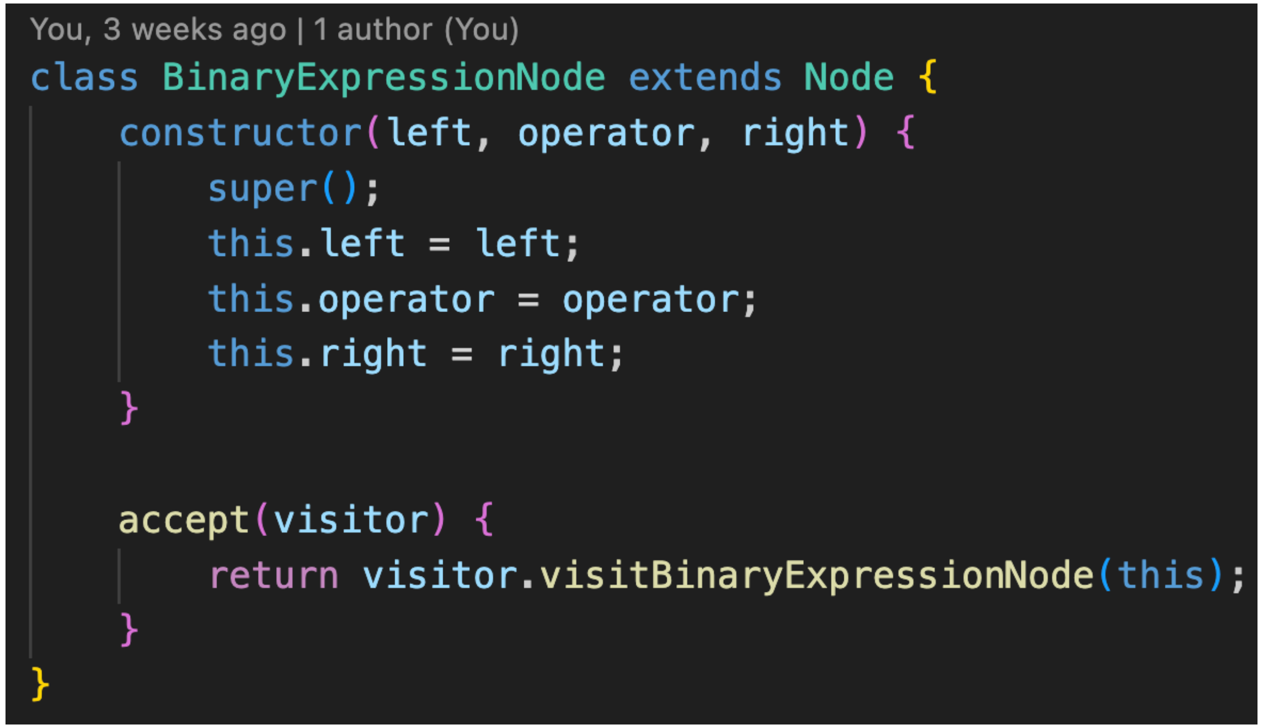
Code



```
operator_assign_expression
    : add_assignment_statement
    | subtract_assignment_statement
    | multiply_assignment_statement
    | divide_assignment_statement
    | modassign_expression;

return_statement
    : RETURN expression SEMI;

expression_statement
    : expression SEMI;

if_statement
    : IF LPAREN expression RPAREN block
    | IF LPAREN expression RPAREN block ELSE block
    | IF LPAREN expression RPAREN block ELSE if_statement;
```

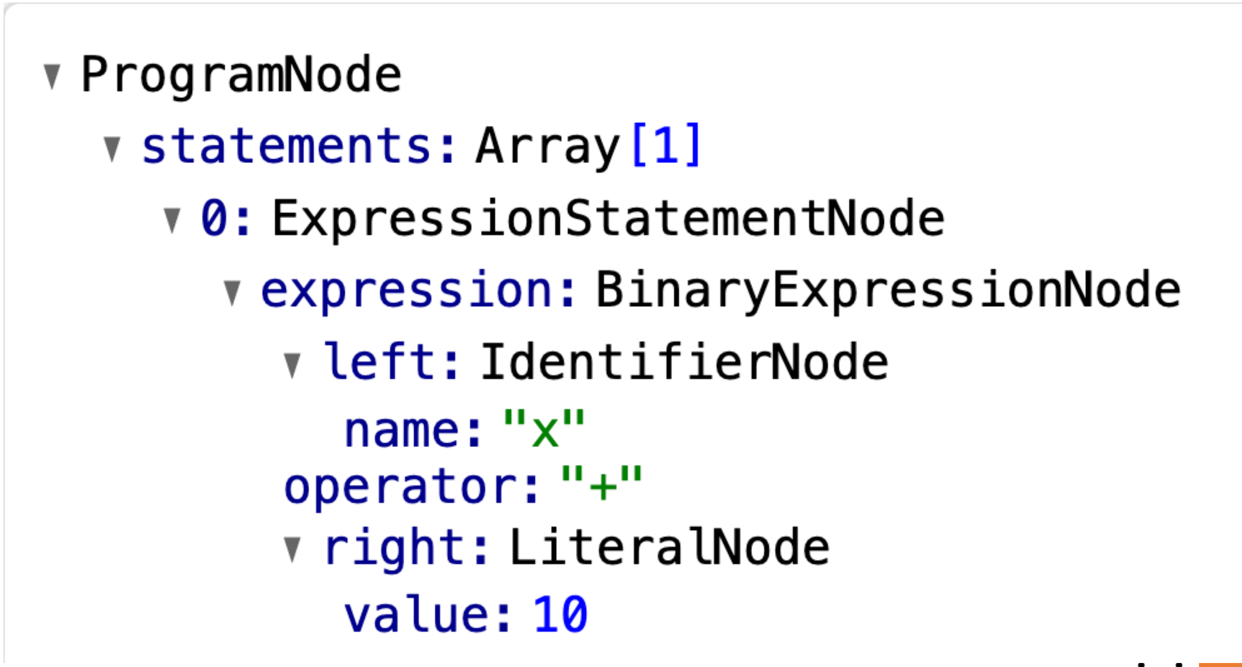
AST - EXAMPLE

Node



```
You, 3 weeks ago | 1 author (You)
class BinaryExpressionNode extends Node {
    constructor(left, operator, right) {
        super();
        this.left = left;
        this.operator = operator;
        this.right = right;
    }

    accept(visitor) {
        return visitor.visitBinaryExpressionNode(this);
    }
}
```

AST

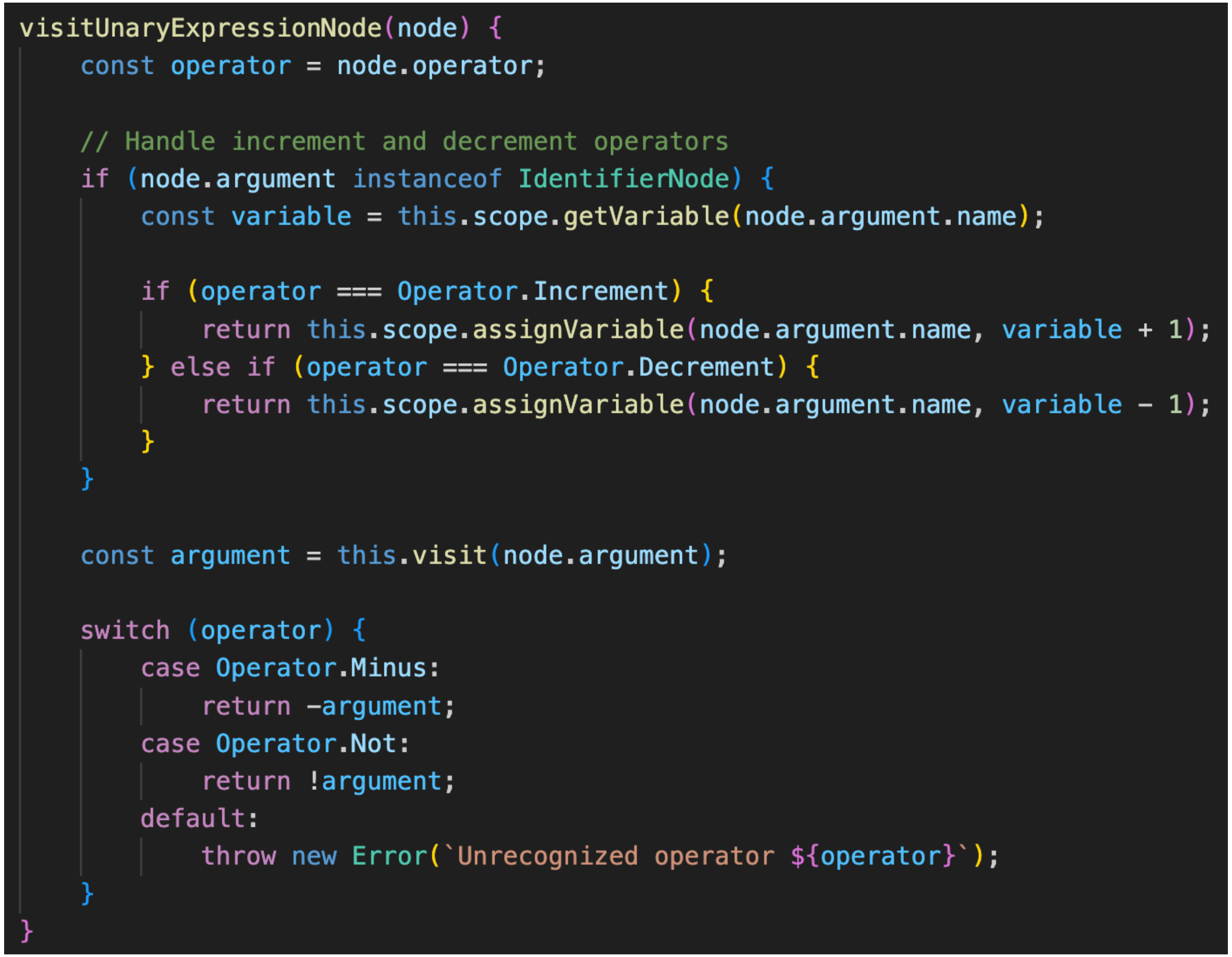


```
▼ ProgramNode
  ▼ statements: Array[1]
    ▼ 0: ExpressionStatementNode
      ▼ expression: BinaryExpressionNode
        ▼ left: IdentifierNode
            name: "x"
          operator: "+"
        ▼ right: LiteralNode
            value: 10
```

INTERPRETER

- Parser was programmed to build an abstract syntax tree (AST) of custom node classes
- Each Node class contains a method named accept
  - The accept method will call the appropriate visit method that is defined
  - For example, calling the accept method on the ProgramNode (root node) will call the visitProgramNode function in the Interpreter
- Each visit function in the interpreter is uniquely defined based on the node properties.

INTERPRETER - EXAMPLE



```
visitUnaryExpressionNode(node) {
    const operator = node.operator;

    // Handle increment and decrement operators
    if (node.argument instanceof IdentifierNode) {
        const variable = this.scope.getVariable(node.argument.name);

        if (operator === Operator.Increment) {
            return this.scope.assignVariable(node.argument.name, variable + 1);
        } else if (operator === Operator.Decrement) {
            return this.scope.assignVariable(node.argument.name, variable - 1);
        }
    }

    const argument = this.visit(node.argument);

    switch (operator) {
        case Operator.Minus:
            return -argument;
        case Operator.Not:
            return !argument;
        default:
            throw new Error(`Unrecognized operator ${operator}`);
    }
}
```

I N T R O D U C I N G …

print(…args)	Prints to the console
ethanify(str, delim = " ")	Takes in a string and a delimiter, splits the string, and returns a new string with each split replaced with “ethan”
isFizz(num)	Takes in a number and returns a boolean: number mod 3 == 0
isBuzz(num)	Takes in a number and returns a boolean: number mod 5 == 0
isFizzBuzz(num)	Takes in a number and returns a boolean: isFizz && isBuzz
gcd(num1, num2)	Takes in two numbers, returns the greatest common denominator
pow(base, exp)	Takes in a base number and exponent, raises the base to the power of the exponent

INTRODUCING…

contains(arr, value)	Takes in an array and a value, returns a boolean that equals whether the array contains the value
sum(arr)	Takes in an array of numbers, returns the sum of the array elements.
average(arr)	Takes in an array of numbers, returns the average of the array elements.
push(arr, value)	Takes in an array and value, adds the value to the array.
remove(arr, value)	Takes in an array and value, removes the value from the array.
findIndex(arr, value)	Takes in an array and value, returns the index where the value is located at.
removeAtIndex(arr, index)	Takes in an array and index, removes the element at specified index.
length(value)	Takes in an array or string value and returns the length.
reverse(value)	Takes in an array or string value and reverses it.

CHALLENGES FACED

- Learning how to use Jison
- Nested arrays
  - [1, 2, 3, [], 4];
- Print command customization
- The Website
  - Needed to properly learn how to use webpack to ensure successful presentations
  - Website needed to use npm packages, which cannot be normally imported in a browser environment.

DEMO

```
laj add(x, y) {
        return x + y;
}

ethan res = add(2, 5);
print(res);

ethan petFoxWebsite = undefined;
print(petFoxWebsite);
print("The above was undefined because it doesn't exist");
```

DEMO

```
laj factorial(n) {
        if (n == 0) {
                return 1;
        } else {
                return n * factorial(n - 1);
        }
}
print(factorial(3));
```

DEMO

```
for (ethan i = 1; i <= 100; i++) {
        if (i % 3 == 0 && i % 5 == 0) {
                // comment to ignore
                print('fizzbuzz');
        } else if (i % 3 == 0) {
                print('fizz');
        } else if (i % 5 == 0) {
                print('buzz');
        } else {
                print(i);
        }
}
```

DEMO- ARITHMETIC

```
ethan addResult = 10 + 5;
print(addResult);
ethan subResult = 10 - 5;
print(subResult);
ethan multResult = 10 * 5;
print(multResult);
ethan divResult = 10 / 5;
print(divResult);
ethan modResult = 10 % 5;
print(modResult);
```

DEMO- COMPARISONS

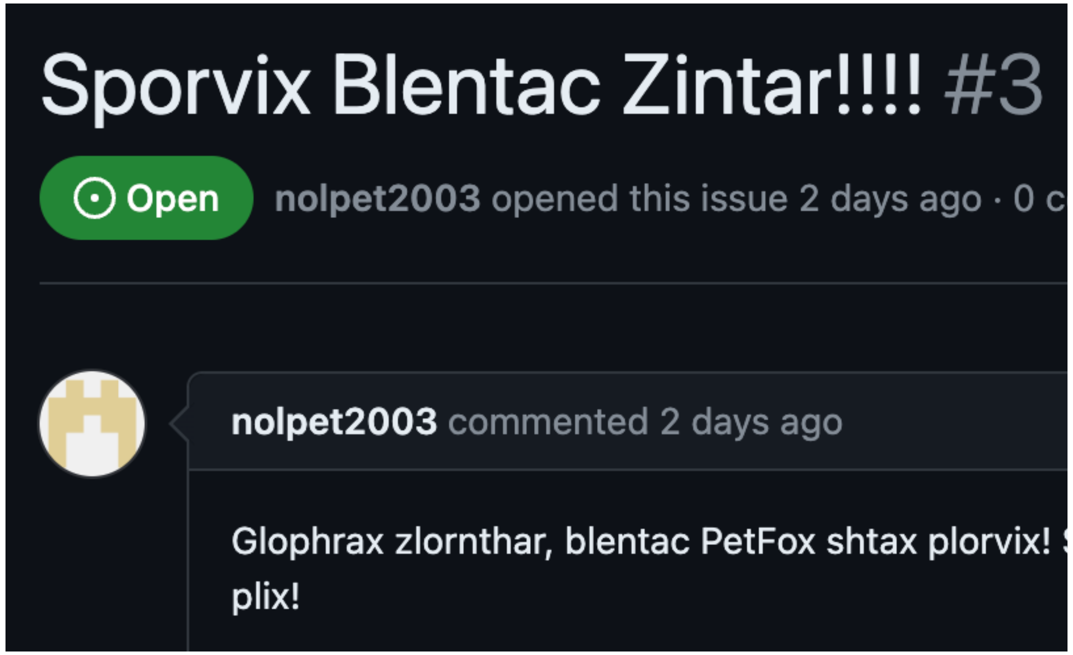
```
ethan gt = 10 > 5;
print(gt);
ethan ls = 10 < 5;
print(ls);
ethan gte = 10 >= 5;
print(gte);
ethan lte = 10 <= 5;
print(lte);
ethan eq = 10 == 5;
print(eq);
ethan ne = 10 != 5;
print(ne);
```

DEMO- LOGICAL

```
ethan x = no;
if (!x) {
   print("true 1");
}
if (x || yes) {
   print("true 2");
}
if (x && yes) {
   print("true 3");
}
```

DEMO – ASSIGNMENTS + MORE

```
ethan str = "The Martian from PetFox was just Nolan";
print(ethanify(str));

ethan x = 5;
x = x + 100;
if (x > 100) {
  print(yes);
  return yes;
}

print("Does return work?");
```



Sporvix Blentac Zintar!!!! #3

Open nolpet2003 opened this issue 2 days ago · 0 c

nolpet2003 commented 2 days ago

Glophrax zlornthar, blentac PetFox shtax plorvix!
plix!