ETHAN LAJEUNESSE PRESENTS:

ETHANET SYNTAX

NODE TYPES

- ProgramNode – Root node of the entire program
  - statements[] – All the statements in the program
- ExpressionStatementNode – Represents an expression
  - Expression: The expression
- BlockStatementNode – Root node of a block
  - statements[] – All the statements in the block

NODE TYPES - STATEMENTS

- VariableDeclarationNode – Declare a new variable
  - Identifier: The variable to declare
  - Expression: What to assign to the variable
- AssignmentStatementNode – Assign a value to a variable
  - Left: The variable to assign the value to
  - Right: What to assign to the variable
- ReturnStatementNode – Return a value in a function
  - Expression: The expression to return

NODE TYPES - STATEMENTS

- ForLoopNode – A for loop
  - Initializer – Declare or assign a value to a variable as the iterator
  - Condition – Run the body until this condition is false
  - Update – After each run of the body, run this, like updating the variable in the initializer
  - Body – What is being ran until the condition is false
- WhileLoopNode – A while loop
  - Condition – The condition to check for before running another iteration of the block
  - Body – What is being ran each time condition is true
- ContinueStatementNode – Skip to the next iteration of the loop
  - This is literally just the “continue” keyword
- BreakStatementNode – Exit the loop
  - This is literally just the “break” keyword

NODE TYPES - STATEMENTS

- IfStatementNode – The classic if/else if/else
  - Condition – The condition of the if statement
  - Consequent – Is taken if the condition is true
  - Alternate – Is taken if the condition is false
- FunctionDeclarationNode – Declare a new function
  - Name – The name of the function
  - Params – An array of parameter names for the function
  - Body – The function body, aka the code ran each time function is called

NODE TYPES - EXPRESSIONS

- BinaryExpressionNode - Two expression and an operator
  - Left – The expression on the left
  - Operator – The operator to apply/use on the left and right expressions
  - Right – The expression on the right
  - Possible operators: &&, ||, ==, !=, <, <=, >, >=, +, -, *, /, %,
- UnaryExpressionNode – An expression with one operator
  - Operator – The operator to apply to the identifier
  - Argument – The factor to apply the operator to
  - Possible operators: !, -, ++, --
- ArrayLiteralNode – Creates a new array
  - Elements – An array of elements in the array
  - Ex: [1, 2, 3, 4]

NODE TYPES - FACTORS

- LiteralNode – Represents a number, string literal, or a boolean
  - Value – The value to represent
- IdentifierNode – Represents a variable
  - Name – The name of a variable
  - Example: print(x);, x would be the identifier node.
- FunctionCallNode – Represents a function call
  - Callee – The name of the function to call
  - Args – The arguments to pass into the function's scope
- ArrayAccessNode - Access an element of an array
  - Identifier – The IdentifierNode where the array is stored
  - Index

PARSER GENERATOR

- Parser Generator: Jison (JavaScript)
- Jison Features
  - Lexer AND Parser generation
  - Define grammar
- To use jison, you can 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

DEMO

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


add(2, 5);
----------------------------------------------------------------------------------------------------
laj factorial(n) {
        if (n == 0) {
                return 1;
        } else {
                return n * factorial(n - 1);
        }
}
```

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;
ethan subResult = 10 - 5;
ethan multResult = 10 * 5;
ethan divResult = 10 / 5;
ethan modResult = 10 % 5;

addResult = addResult + 2;
subResult = subResult - 2;
multResult = multResult * 2;
divResult = divResult / 2;
```

DEMO- COMPARISONS

```
ethan gt = 10 > 5;
ethan ls = 10 < 5;
ethan gte = 10 >= 5;
ethan gte2 = 10 >= 10;
ethan lte = 10 <= 5;
ethan lte2 = 10 <= 10;
ethan eq = 10 == 5;
ethan ne = 10 != 5;
```

DEMO- LOGICAL

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

DEMO – ASSIGNMENTS + MORE

```
ethan x = 5;
x = x + 100;

if (x > 100) {
  return yes;
}
```