



**ETHAN
LAJEUNESSE
PRESENTS:**

ETHANET

LANGUAGE DETAILS

- Name: Ethanet (named after me)
- JavaScript-Like Syntax
- Uses open and closed braces to define a code block:
- Semicolons are required at the end of each statement
- Ex:

```
for (ethan i = 0; i < 10; i++) {  
    if (i == 5) {  
        continue; // Skips the current iteration when i is 5  
    }  
    // Operations to execute if i is not 5  
}
```



TOKEN TYPES - GENERAL SYNTAX

- STRING_LITERAL - "string" or 'string'
- VAR - ethan
- IDENTIFIER - Variable names
- ASSIGN - =
- NUMBER - 1, 2, 3.5, 400, 5, 6500, 7, 8, 900, etc.
- LBRACE - {
- RBRACE - }
- LPAREN - (
- RPAREN -)



TOKEN TYPES - GENERAL SYNTAX

- LBRACKET - [
- RBRACKET -]
- COMMA - ,
- SEMI - ;
- RETURN - return
- NULL - null - Deliberately empty
- UNDEFINED - undefined - Denotes an undefined value
- EOF - Denotes the end of the file
- INVALID - Denotes an invalid token



TOKEN TYPES - BOOLEANS

- TRUE - yes
- FALSE - no



TOKEN TYPES - CONDITIONALS

- IF - if
- ELSE - else



TOKEN TYPES - MATH

- ADD - +
- SUBTRACT - -
- MULT - *
- DIV - /
- MOD - %
- INCREMENT - ++
- DECREMENT - --



TOKEN TYPES - EQUALITY

- EQUAL - ==
- NOTEQUAL - !=
- LESSTHAN - <
- LESSTHANEQUAL - <=
- GREATERTHAN - >
- GREATERTHANEQUAL - >=



TOKEN TYPES - LOGICAL

- AND - &&
- OR - ||
- NOT - !



TOKEN TYPES - LOOPS

- FOR - for
- WHILE - while
- BREAK - break
- CONTINUE - continue



TOKEN TYPES - FUNCTIONS

- FUNCTION - laj



DATA TYPES?

- For now, Ethanet will allow variables to be anything. This means a number can be assigned to a variable originally defined as a string.
- Therefore, variables can hold:
 - Numbers
 - Strings
 - Arrays
 - Null
 - Undefined
- This may change if it's easier to create the language around enforcing types. If this is the case, we will have implicit typing, where once a variable is assigned a number, it can only be a number.



LEXER GENERATOR

- Lexer Generator: Jison
- Jison Features
 - Use regular expressions to define tokens
 - 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
- For purposes of this presentation, I separated the lexer and parser so that it returned tokens



UNIQUE FEATURES

- Define variables with the **ethan** keyword
- Define functions with the **laj** keyword
- Boolean values - **yes** is true, **no** is false
- The **+-** and **-+** operators do nothing
- More to come!



D E M O

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

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



D E M O

```
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);  
    }  
}
```

