Equal — the Esoteric Programming Language Masquerading as HTML

Table of Contents

Links

- Online interpreter: https://equal-lang.github.io/equal/
- Code repository: https://github.com/equal-lang/equal
- My Github: https://github.com/hliu23

Using Equal

Usage

- The online interpreter is recommended

Building from source

Equal and CLI

```
npm run test
npm run build-cli
npm run start-cli
```

Deprecated GUI

```
npm run build-gui
npm run dist-gui
npm run start-gui
```

Online interpreter

```
npm run build-cli
npm run build-website
npm run start-website
npm run watch-website
```

API for online interpreter

```
npm run build-api
npm run setup-api
npm run start-api | npm run start-api-prod
```

All

```
npm run make
```

The Equal Language

Rules of thumb

- Tagnames
  - `span` is used in print statements
  - `a` is used for variables
  - `form` is used for reserved operators or user-defined functions
    - `input` is used to indicate parameters and return statements in functions
    - `label` is used to pass arguments to operators or functions
  - `h1` to `h6` is used for if/else statements
  - `p` is used for loops
  - `div` is used for code blocks that have their own scope
- Attributes
  - `id` is usually used when defining variables or functions
  - `href` and `title` are used to refer to variables and functions that are already defined

Features

IO

Output

```
<span>
<!-- output expressions here -->
</span>
```

Variables

Declaration or assignment

```
<a id="name_of_variable" class="global">
  <!-- set class value to global to modify global value -->
  value
</a>
```

Reference

```
<!-- refer to variable -->
<a href="name_of_variable">
</a>
```

- variables must be initialized
- only href considered in expression
- types
  - dynamically typed
  - three types available: string, number, boolean

- if `!isNaN(Number(expression))` is true, the variable is a number
- else if `expression` matches true or false exactly, the variable is a boolean
- else if the variable is a string
- mismatched types and operator will throw error

Operators

Arithmetic

```
<form title="+">
  <label for="name_of_param1">num1</label>
  <label for="name_of_param2">num2</label>
  <!-- more args possible -->
</form>
<!-- possible titles: "+" "-" "*" "/" -->
```

- no division by zero
- plus does not work on strings

Comparsion

```
<form title="==">
  <label for="name_of_param1">expression1</label>
  <label for="name_of_param2">expression2</label>
  <!-- only two args -->
</form>
<!-- possible titles: "==" "!=" ">" "<" -->
```

- equal and not equal are implemented strictly, i.e. "==="

Logic

```
<!-- (1) -->
<form title="!">
  <label for="name_of_param1">expression1</label>
  <!-- only one arg -->
</form>
<!-- (2) -->
<form title="&&">
  <label for="name_of_param1">expression1</label>
  <label for="name_of_param2">expression2</label>
  <!-- more args possible -->
</form>
<!-- possible titles for (1): "!" -->
<!-- possible titles for (2): "&&" "||" -->
<!-- &&: return true if all evaluate to true -->
<!-- ||: return true if one evaluates to true -->
```

- only operators that cannot be easily constructed with other operators are provided

Reserved function names

- + , - , * , /, !, ==, !=, >, <, &&, ||

User-defined functions

Definition

```
<form id="name_of_function">
  <input id="name_of_param1">
  <input id="name_of_param2">
  <input id="name_of_param3">
  <!-- etc -->
  <div>
    <!-- code to be executed -->
    <!-- optional, return value defaults to 0 -->
    <input type="submit">
    <!-- an expression that is the return value -->
  </div>
</form>
```

- only global functions supported
- functions must be declared (and initialized) before being used

Reference

```
<!-- for is optional and is for readability purposes only-->
<form title="name_of_function">
  <label for="name_of_param1">
    arg1
  </label>
  <label for="name_of_param2">
    arg2
  </label>
  <label for="name_of_param3">
    arg3
  </label>
  <!-- etc -->
</form>
```

Control Flow

If/else statements

```
<h1>
  <!-- expression -->
  <!-- statements (execute if expression evaluates to true) -->
</h1>
<h2>
  <!-- optional else if statements -->
  <!-- expression -->
  <!-- statements -->
</h2>
<h6>
  <!-- optional else statement -->
</h6>
```

- from h1 to h5, the elements must go in order
- h1, or h1 and h6 can appear alone

While loops

```
<p>
  <!-- expression -->
  <!-- statements (execute until expression evaluates to false) -->
</p>
```

Misc

Comments

```
<!-- comments about the code -->
```

Code blocks

```
<div>
  <!-- code inside -->
</div>
```

- variable outside the current div block can be accessed, variable in the global scope can be modified
- variable in children div blocks cannot be accessed or modified

Examples

Calculating the first twenty Fibonacci numbers in JS

```
function fib(n) {
  if (n < 2) return n;
  return (fib(n-2) + fib(n-1));
}

var i = 0;
while (i < 20) {
  console.log(fib(i));
  i = i + 1;
}
```

Calculating the first twenty Fibonacci numbers in Equal

```
<form id="fib">
  <!-- function fib(n) -->
  <input id="n">
  <div>
    <h1>
      <!--  if (n < 2)    -->
      <form title="<">
        <label><a href="n"></a></label>
        <label>2</label>
      </form>
      <!-- return n -->
      <input type="submit">
      <a href="n"></a>
    </h1>

<!--   return fib(n-2) + fib(n-1) -->
    <input type="submit">
    <form title="+">
      <label>
        <form title="fib">
          <label>
            <form title="-">
              <label>
                <a href="n"></a>
              </label>
              <label>2</label>
            </form>
          </label>
        </form>
```

```
      </label>

      <label>
        <form title="fib">
          <label>
            <form title="-">
              <label>
                <a href="n"></a>
              </label>
              <label>1</label>
            </form>
          </label>
        </form>
      </label>
    </form>

  </div>
</form>

<!-- var i = ? -->
<a id="i">0</a>

<!-- while (i < ?)  -->
<p>
  <form title="<">
    <label>
      <a href="i"></a>
    </label>
    <label>20</label>
  </form>

<!-- console.log(fib(i)) -->
  <span>
    <form title="fib">
      <label>
        <a href="i"></a>
      </label>
    </form>
  </span>

<!-- i = i + ? -->
  <a id="i" class="global">
    <form title="+">
      <label>
        <a href="i"></a>
      </label>
```

```
      <label>1</label>
    </form>
  </a>
</p>

<!-- much longer than JS, but same functions and same ideas -->
```

Implementation Details

Formal definitions in Backus-Naur Form

```
PROGRAM -> STATEMENT*

STATEMENT -> SCOPE

SCOPE ->
((<div>
  ASSIGNMENT
</div>)*
| ASSIGNMENT*)

ASSIGNMENT ->
(<a id="ID" (class="global")?>
  EXPRESSION
</a>
| FUNCTION_DECLARATION)

FUNCTION_DECLARATION ->
(<form id="ID">
  (<input id="ID">)*
  <div>
    (STATEMENT)*
  </div>
</form>
| RETURN_STATEMENT)

RETURN_STATEMENT ->
(<input type="submit" value="ID">
EXPRESSION
| LOOP)

LOOP ->
(<p>
  EXPRESSION
  STATEMENT*
</p>
| CONDITIONAL_STATEMENT)

CONDITIONAL_STATEMENT ->
(<h1>
  EXPRESSION
  STATEMENT*
</h1>
```

```
(<h2>
  EXPRESSION
  STATEMENT*
<h2>)?
(<h3>
  EXPRESSION
  STATEMENT*
<h3>)?
(<h4>
  EXPRESSION
  STATEMENT*
<h4>)?
(<h5>
  EXPRESSION
  STATEMENT*
<h5>)?
(<h6>
  STATEMENT*
</h6>)?
| PRINT_STATEMENT)

PRINT_STATEMENT ->
(<span>
  EXPRESSION*
</span>
| EXPRESSION_STATEMENT)

EXPRESSION_STATEMENT -> EXPRESSION

EXPRESSION -> LOGIC

LOGIC ->
(<form title="(&& | ||)">
  <label>EXPRESSION</label>
  (<label>EXPRESSION</label>)+
</form>
| EQUALITY)

EQUALITY ->
(<form title="(== | !=)">
  <label>EXPRESSION</label>
  <label>EXPRESSION</label>
</form>
| COMPARSION)

COMPARSION ->
```

```
(<form title="(> | <)">
  <label>EXPRESSION</label>
  <label>EXPRESSION</label>
</form>
| ADDITION)

ADDITION ->
(<form title="(+ | -)">
  <label>EXPRESSION</label>
  (<label>EXPRESSION</label>)+
</form>
| MULTIPLICATION)

MULTIPLICATION ->
(<form title="(* | /)">
  <label>EXPRESSION</label>
  (<label>EXPRESSION</label>)+
</form>
| UNARY)

UNARY ->
(<form title="!">
  <label>EXPRESSION</label>
</form>
| CALL)

CALL ->
(<form title="ID">
  (<label>EXPRESSION</label>)*
</form>
| PRIMARY)

PRIMARY ->
(string | number | boolean | IDENTIFIER)

IDENTIFIER ->
(<a href="ID">
</a>)

ID -> string
```

- Non-terminals: CAPS
- Terminals: string, number, boolean (corresponding to JS types)

References

- The vast majority of code in this repository is written by me and the errors are mine alone
- Credit to Crafting Interpreters for inspiring much of the structure of the interpreter
- Other references:
  - HTML Standards
  - The Python Language Reference