Expression parsing and evaluation

Expressions can be parsed and evaluated in various ways:

- Using the function `math.eval(expr [,scope])`.
- Using the function `math.compile(expr)`.
- Using the function `math.parse(expr)`.
- By creating a parser, `math.parser()`, which contains a function `eval` and keeps a scope with assigned variables in memory.

Eval

Math.js comes with a function `math.eval` to evaluate expressions. Syntax:

```
math.eval(expr)
math.eval(expr, scope)
math.eval([expr1, expr2, expr3, ...])
math.eval([expr1, expr2, expr3, ...], scope)
```

Function `eval` accepts a single expression or an array with expressions as the first argument and has an optional second argument containing a scope with variables and functions. The scope is a regular JavaScript Object. The scope will be used to resolve symbols, and to write assigned variables or function.

The following code demonstrates how to evaluate expressions.

```
// evaluate expressions
math.eval('sqrt(3^2 + 4^2)')            // 5
math.eval('sqrt(-4)')                   // 2i
math.eval('2 inch to cm')               // 5.08 cm
math.eval('cos(45 deg)')                // 0.7071067811865476

// provide a scope
let scope = {
    a: 3,
    b: 4
}
math.eval('a * b', scope)               // 12
```

```
math.eval('c = 2.3 + 4.5', scope)       // 6.8
scope.c                                 // 6.8
```

Compile

Math.js contains a function `math.compile` which compiles expressions into JavaScript code. This is a shortcut for first parsing and then compiling an expression. The syntax is:

```
math.compile(expr)
math.compile([expr1, expr2, expr3, ...])
```

Function `compile` accepts a single expression or an array with expressions as the argument. Function `compile` returns an object with a function `eval([scope])`, which can be executed to evaluate the expression against an (optional) scope:

```
const code = math.compile(expr)     // compile an expression
const result = code.eval([scope])   // evaluate the code with an optional scope
```

An expression needs to be compiled only once, after which the expression can be evaluated repeatedly and against different scopes. The optional scope is used to resolve symbols and to write assigned variables or functions. Parameter `scope` is a regular Object.

Example usage:

```
// parse an expression into a node, and evaluate the node
const code1 = math.compile('sqrt(3^2 + 4^2)')
code1.eval()  // 5
```

Parse

Math.js contains a function `math.parse` to parse expressions into an expression tree. The syntax is:

```
math.parse(expr)
math.parse([expr1, expr2, expr3, ...])
```

Function `parse` accepts a single expression or an array with expressions as the argument. Function `parse` returns a the root node of the tree, which can be successively compiled and evaluated:

```
const node = math.parse(expr)       // parse expression into a node tree
const code = node.compile()         // compile the node tree
const result = code.eval([scope])   // evaluate the code with an optional scope
```

The API of nodes is described in detail on the page Expression trees.

An expression needs to be parsed and compiled only once, after which the expression can be evaluated repeatedly. On evaluation, an optional scope can be provided, which is used to resolve symbols and to write assigned variables or functions. Parameter `scope` is a regular Object.

Example usage:

```
// parse an expression into a node, and evaluate the node
const node1 = math.parse('sqrt(3^2 + 4^2)')
```

```
const code1 = node1.compile()
code1.eval()  // 5

// provide a scope
const node2 = math.parse('x^a')
const code2 = node2.compile()
let scope = {
    x: 3,
    a: 2
}
code2.eval(scope)  // 9

// change a value in the scope and re-evaluate the node
scope.a = 3
code2.eval(scope)  // 27
```

Parsed expressions can be exported to text using `node.toString()`, and can be exported to LaTeX using `node.toTex()`. The LaTeX export can be used to pretty print an expression in the browser with a library like MathJax. Example usage:

```
// parse an expression
const node = math.parse('sqrt(x/x+1)')
node.toString()   // returns 'sqrt((x / x) + 1)'
node.toTex()      // returns '\sqrt{ {\frac{x}{x} }+{1} }'
```

Parser

In addition to the static functions `math.eval` and `math.parse`, math.js contains a parser with functions `eval` and `parse`, which automatically keeps a scope with assigned variables in memory. The parser also contains some convenience functions to get, set, and remove variables from memory.

A parser can be created by:

```
const parser = math.parser()
```

The parser contains the following functions:

- `clear()` Completely clear the parser's scope.
- `eval(expr)` Evaluate an expression. Returns the result of the expression.
- `get(name)` Retrieve a variable or function from the parser's scope.
- `getAll()` Retrieve a map with all defined a variables from the parser's scope.
- `remove(name)` Remove a variable or function from the parser's scope.
- `set(name, value)` Set a variable or function in the parser's scope.

The following code shows how to create and use a parser.

```
// create a parser
const parser = math.parser()

// evaluate expressions
parser.eval('sqrt(3^2 + 4^2)')          // 5
parser.eval('sqrt(-4)')                 // 2i
parser.eval('2 inch to cm')             // 5.08 cm
parser.eval('cos(45 deg)')              // 0.7071067811865476
```

```
// define variables and functions
parser.eval('x = 7 / 2')                // 3.5
parser.eval('x + 3')                    // 6.5
parser.eval('f(x, y) = x^y')            // f(x, y)
parser.eval('f(2, 3)')                  // 8

// get and set variables and functions
const x = parser.get('x')               // x = 7
const f = parser.get('f')               // function
const g = f(3, 3)                       // g = 27
parser.set('h', 500)
parser.eval('h / 2')                    // 250
parser.set('hello', function (name) {
    return 'hello, ' + name + '!'
})
parser.eval('hello("user")')            // "hello, user!"

// clear defined functions and variables
parser.clear()
```