JISP

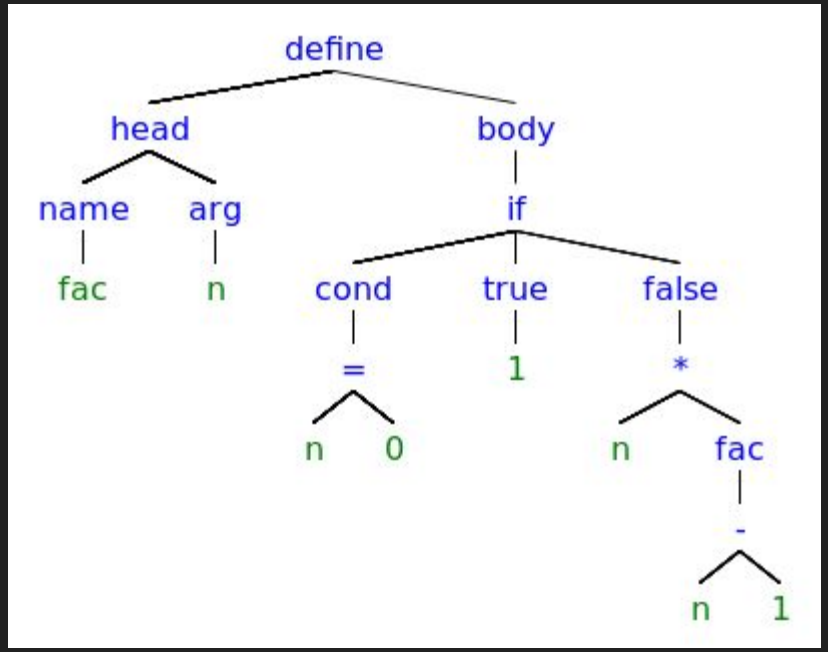
Eine funktionale Spielzeugsprache selbst gemacht

In drei einfachen Schritten zur Programmiersprache

1. Syntax erfinden
2. Parser implementieren
3. Interpreter implementieren

Syntax ~~erfinden~~ stehlen

```
(define (fac n)
  (if (= n 0)
    1
    (*
      n
      (fac (- n 1)))))
```



Syntax ~~erfinden~~ stehlen

```
(define (fac n)
  (if (= n 0)
    1
    (*
      n
      (fac (- n 1)))))
```

Scheme

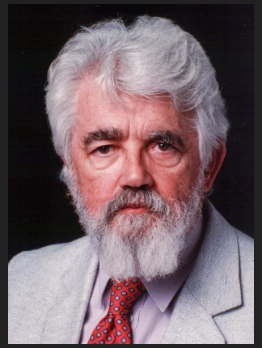
```
['lambda', ['n'],
 ['if', ['=', 'n', 0],
  1,
  ['*',
   'n',
   ['recur', ['-', 'n', 1]]]]]
```

JISP

Syntax ~~erfinden~~ stehlen

```
(define (fac n)
  (if (= n 0)
    1
    (*
      n
      (fac (- n 1)))))
```

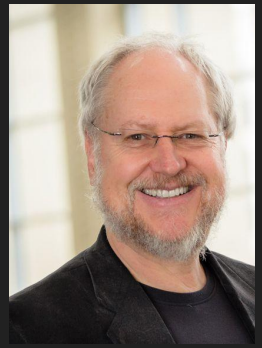
S-Expression



John McCarthy

```
['lambda', ['n'],
 ['if', ['=', 'n', 0],
  1,
  ['*',
   'n',
   ['recur', ['-', 'n', 1]]]]]
```

JSON



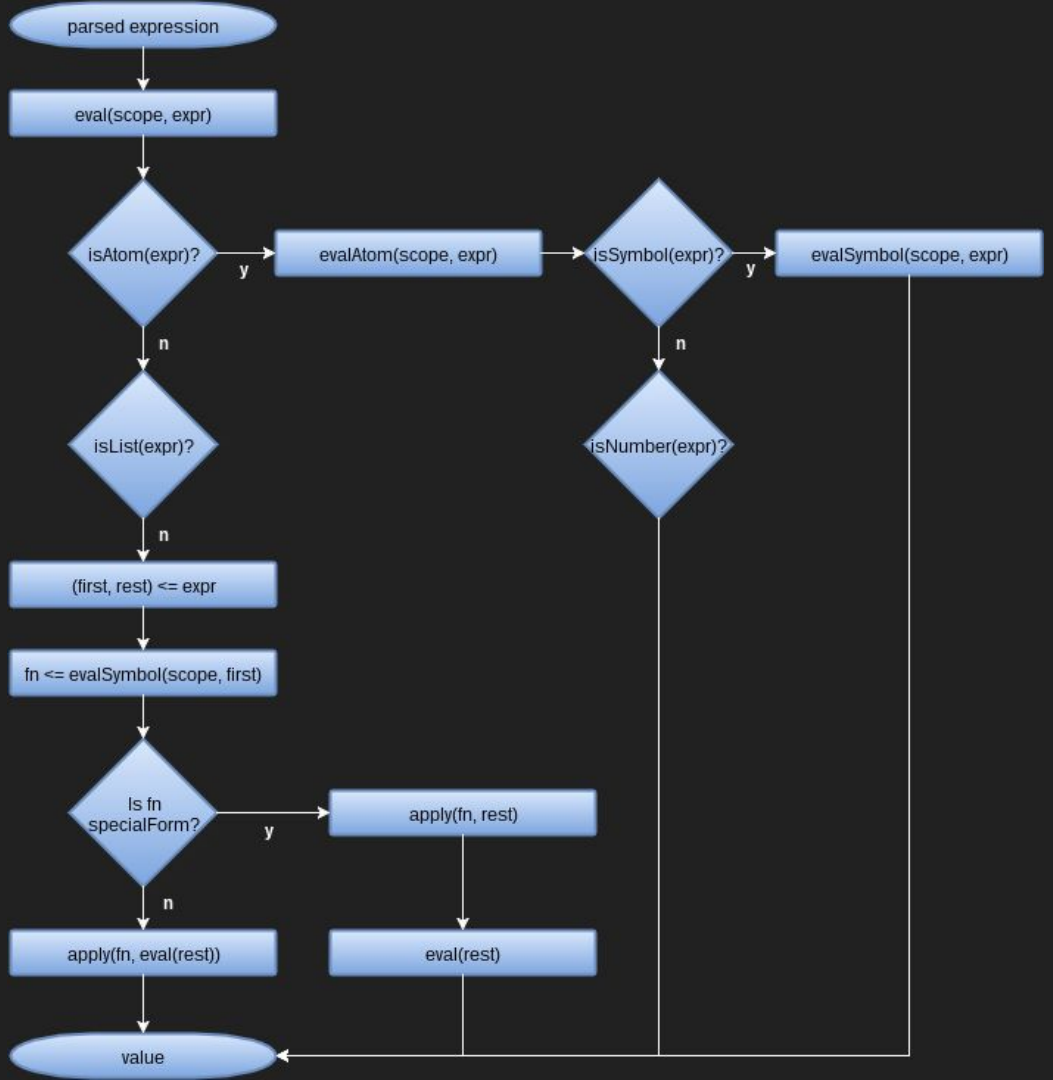
Douglas Crockford

In drei einfachen Schritten zur Programmiersprache

1. ~~Syntax erfinden~~ **JSON**
2. ~~Parser implementieren~~ **JSON.parse(src)**
3. Interpreter implementieren
    a. Host-Language → JavaScript
    b. Datentypen:
        i. Symbole (JS-Strings)
        ii. Zahlen (JS-Numbers)
        iii. Listen (JS-Arrays)
        iv. Lambdas (JS-Functions)
    c. Arithmetik
    d. Funktionen
    e. Variablen
    f. Kontrollstrukturen (if, ~~Schleifen~~ Rekursion)

Der JISP-Interpreter

- verwaltet eine Symboltabelle *scope*
  - runtime
  - benutzerdefinierte Symbole
- parst Ausdruck *expr* rekursiv
- Listenausdrücke der Form [‘fn-name’, ‘arg1’, ‘arg2’, ...] werden als Funktionsaufruf verstanden
- “spezielle” Funktionen sind Kontrollstrukturen



Der JISP-Interpreter

- verwaltet eine Symboltabelle *scope*
  - runtime
  - benutzerdefinierte Symbole
- parst Ausdruck *expr* rekursiv
- Listenausdrücke der Form [‘fn-name’, ‘arg1’, ‘arg2’, ...] werden als Funktionsaufruf verstanden
- “spezielle” Funktionen sind Kontrollstrukturen

```
function compileExpression(scope, expr) {
  var compileStack = [expr],
      thunk = new Thunk(),
      first, rest, fn, returnVal;

  while (compileStack.length > 0) {
    expr = compileStack.pop();

    if (isAtom(expr)) {
      first = evalAtom(scope, expr);
      thunk.pushArgument(first);
    } else if (isList(expr)) {
      first = expr[0];
      rest = expr.slice(1);
      fn = evalSymbol(scope, first);
      if (fn.macro) {
        compileStack.push(fn.apply(scope, rest));
      } else if (fn.special) {
        thunk.merge(fn.apply(scope, rest));
      } else {
        thunk.pushFunction(fn, rest.length, expr);
        Array.prototype.push.apply(compileStack, rest);
      }
    }
  }

  return thunk;
}
```

JISP = Interpreter + Runtime

```
['+', 1, 2, 3] // 6

['+',
 ['*', 2, 2],
 ['*', 3, 3]] // 13

// Summe von 1 bis 100??
// Das wäre cool:

['apply', '+',
 ['list:ton', 100]]
```

```
JISP.Runtime = {
  '+': function () {
    var len = arguments.length, sum = 0, i, n;

    for (i = 0; i < len; i += 1) {
      n = arguments[i];
      if (isNumber(n)) {
        sum += n;
      } else {
        return error('not a number');
      }
    }

    return sum;
  }
};
```

JISP-Funktionen

```
['lambda', ['n'],
 ['if', ['=', 'n', 0],
  1,
  ['*',
   'n',
   ['recur', ['-', 'n', 1]]]]]
```

JISP

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

JavaScript

JISP-Funktionen

```
JISP.Runtime = {
  'lambda': special(function (symbols, body) {
    var scope = this,
        childScope, i,
        fn = function () {
          var argLen = arguments.length;

          childScope = Object.create(scope);

          childScope.recur = fn;

          for (i = 0; i < argLen; i += 1) {
            childScope[symbols[i]] = arguments[i];
          }

          return Interpreter.compileExpression(childScope, body);
        };

    return fn;
  })
};
```

JISP-Variablen

```
[['lambda', ['a', 'b'],
  ['+',
   ['*', 'a', 'a'],
   ['*', 'b', 'b']]], 2, 3]


// 4 + 9 = 13
```

```
['let', [['a', 2],
         ['b', 3]],
 ['+',
  ['*', 'a', 'a'],
  ['*', 'b', 'b']]]

// 4 + 9 = 13
```

JISP-Variablen

```
JISP.Runtime = {
  'let': macro(function (declarations, body) {
    var vars = declarations.map(function (declaration) {
          return declaration[0];
        }),
        initForms = declarations.map(function (declaration) {
          return declaration[1];
        });

    return [['lambda', vars,
             body]].concat(initForms);
  })
};
```

JISP-Variablen

```
['let', [['a', 2],
         ['b', 3]],
 ['let', [['a**2',
           ['*', 'a', 'a']],
          ['b**2',
           ['*', 'b', 'b']]],
  ['+', 'a**2', 'b**2']]]

// 4 + 9 = 13
```

```
['let*', [['a', 2],
          ['a**2',
           ['*', 'a', 'a']],
          ['b', 3],
          ['b**2',
           ['*', 'b', 'b']]],
 ['+', 'a**2', 'b**2']]

// 4 + 9 = 13
```

JISP-Variablen

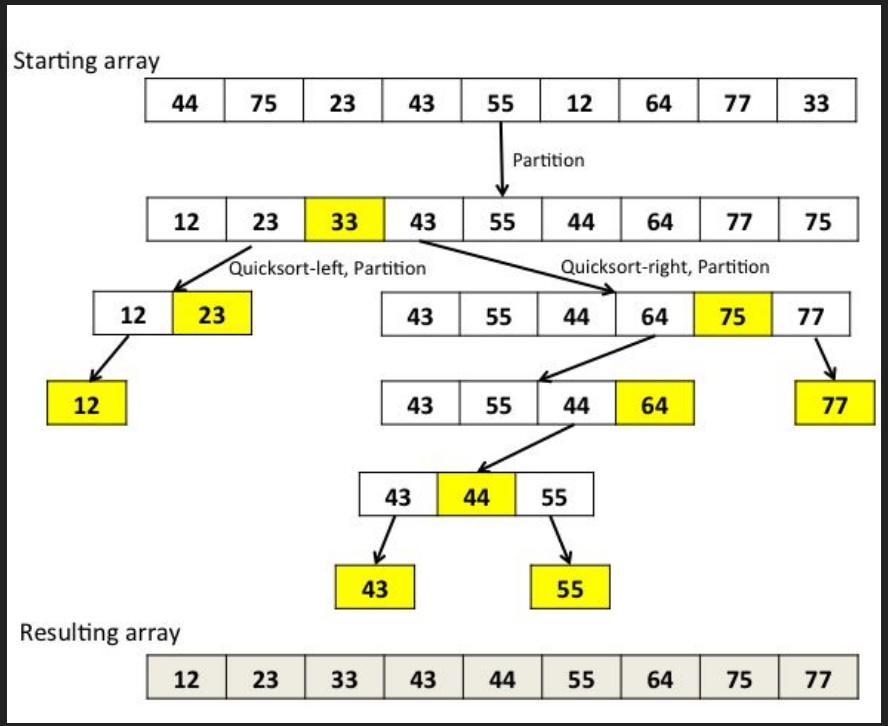
```
JISP.Runtime = {
  'let*': macro(function (declarations, body) {
    var vars = declarations.map(function (declaration) {
          return declaration[0];
        }),
        initForms = declarations.map(function (declaration) {
          return declaration[1];
        });

    return buildLets();

    function buildLets() {
      var oneVar = vars.shift(),
          oneInitForm = initForms.shift();

      if (oneVar) {
        return ['let', [[oneVar, oneInitForm]],
                buildLets()];
      } else {
        return body;
      }
    }
  })
};
```

Beispiel-Programm: QuickSort

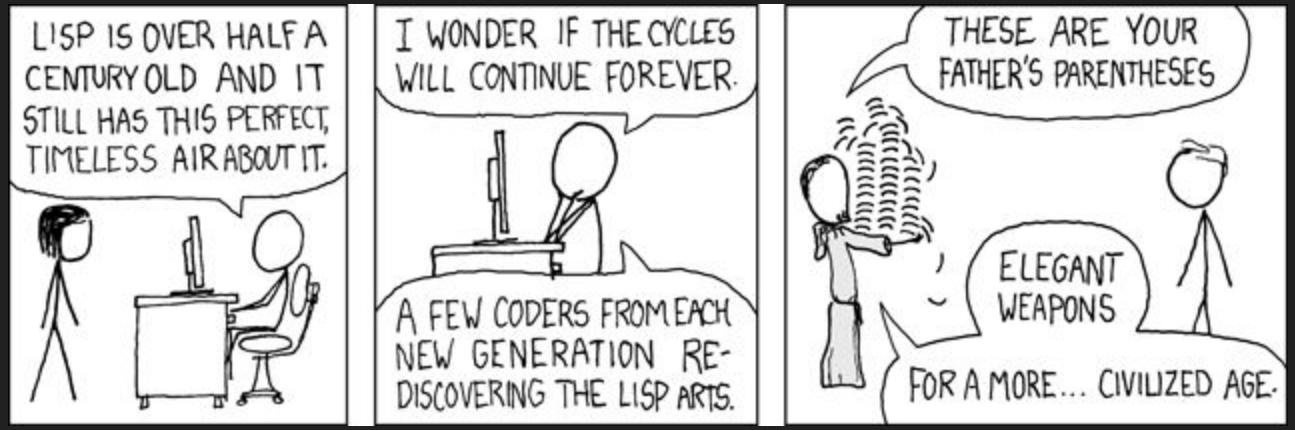


http://www.cs.swarthmore.edu/~soni/cs35/f13/Labs/images/06/quickSort.png

```
['lambda', ['cmp', 'list'],
 ['if', ['>', ['length', 'list'], 1],
  ['let*', [['pivot', ['car', 'list']],
            ['st-p', ['lambda', ['elt'],
             ['=', -1, ['cmp', 'elt', 'pivot']]]],
            ['ge-p', ['lambda', ['elt'],
             ['or',
              ['=', 1, ['cmp', 'elt', 'pivot']],
              ['=', 0, ['cmp', 'elt', 'pivot']]]]],
            ['greater-or-equal',
             ['list:filter', 'ge-p', ['cdr', 'list']]],
            ['smaller',
             ['list:filter', 'st-p', ['cdr', 'list']]]],
   ['cons',
    ['recur', 'cmp', 'smaller'],
    ['cons', 'pivot',
     ['recur', 'cmp', 'greater-or-equal']]]],
  'list']]
```

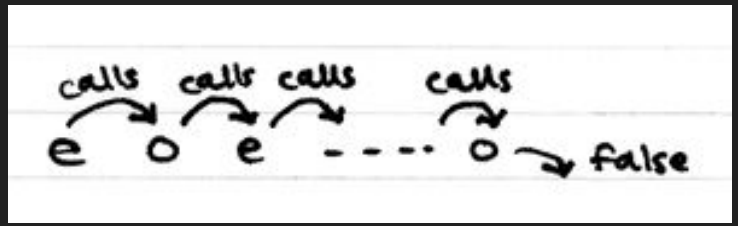
Live-Demo

Danke für Eure Aufmerksamkeit!



https://xkcd.com/297

Trampolining



http://blog.richdougherty.com/2009/04/tail-calls-tailrec-and-trampolines.html