

Meta-circular Evaluation

Carlo Zancanaro

Programming languages
are important tools

General Purpose	Domain Specific
<p data-bbox="432 314 610 426">C#</p> <p data-bbox="202 484 838 618">Javascript</p> <p data-bbox="297 650 741 787">Clojure</p> <p data-bbox="471 904 568 929">...</p>	<p data-bbox="1232 314 1605 426">HTML</p> <p data-bbox="1277 484 1557 594">CSS</p> <p data-bbox="1099 650 1734 765">Markdown</p> <p data-bbox="1369 904 1466 929">...</p>

Rokt Specific

Custom Fields
Attribute Replacement

...

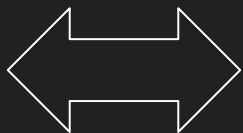
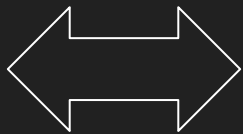
```
(match= [:raw :country]  
  ["Australia" "AU"]  
  ["New Zealand" "NZ"]  
  [:none-val nil]  
  [:else-val "Unknown"])
```

```
Hi {rokt.firstname || "there"}!
```

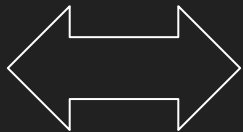
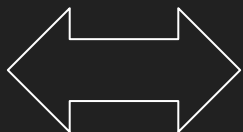
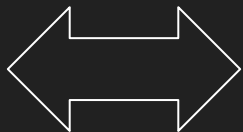
An easy way to implement
programming languages

Javascript	Rokt-Lang
booleans	booleans

Javascript	Rokt-Lang
booleans	booleans
numbers	numbers



Javascript	Rokt-Lang
booleans	booleans
numbers	numbers
functions	functions



```
const evaluate = (expression) => {  
  switch (typeof(expression)) {  
    // fill in cases here  
  }  
  throw new Error(`Can't eval ${expression}`);  
};
```

Boring values

```
    evaluate(10) // => 10  
evaluate(true)  // => true
```

```
case 'number':  
case 'boolean':  
    return expression;
```

Variables

```
evaluate("x") // => ???
```



```
const evaluate = (expression, environment) => {  
  ...  
}
```

```
evaluate("x", name => 321)  
// => 321  
evaluate("y", name => 321)  
// => 321
```

```
case 'string':  
    return environment(expression);
```

Calling functions

```
const sum = (a, b) => a + b;  
evaluate(["+", 1, 2],  
        name => sum)  
  
// => 3
```

```
case 'object':  
  if (Array.isArray(expression)) {  
    const results = expression.map(e => {  
      return evaluate(e, environment);  
    });  
    return results[0](... results.slice(1));  
  }
```

Conditionals

```
const equals = (a, b) => a === b;
evaluate(["if", ["=", 1, 1], 3, 0],
        name => equals)
// => 3
evaluate(["if", ["=", 1, 2], 3, 0],
        name => equals)
// => 0
```



```
if (Array.isArray(expression)) {  
  if (expression[0] === 'if') {  
    if (evaluate(expression[1], environment)) [  
      return evaluate(expression[2], environment);  
    } else {  
      return evaluate(expression[3], environment);  
    }  
  } else {  
    ...  
  }  
}
```

Making functions

```
const sum = (a, b) => a + b;  
const fn = ["fn", ["x"], ["+", 1, "x"]];  
evaluate([fn, 10], name => sum)  
// => 11
```

```
if (expression[0] === 'fn') {  
  const argNames = expression[1];  
  return (... argValues) => {  
    const localEnvironment = name => {  
      const index = argNames.indexOf(name);  
      if (index === -1) {  
        return environment(name);  
      } else {  
        return argValues[index];  
      }  
    };  
    return evaluate(expression[2], localEnvironment);  
  };  
}
```

Putting it all together

```
const fn = ["fn", ["x"],  
             ["if", ["=", "x", 1],  
                   0,  
                   ["+", 1, "x"]]]];  
evaluate([fn, 1], environment) // => 0  
evaluate([fn, 10], environment) // => 11
```

```
const environment = name => {  
  if (name === '+' ) {  
    return (a, b) => a + b;  
  } else if (name === '=') {  
    return (a, b) => a === b;  
  } else {  
    throw new Error(`Can't resolve ${name}`);  
  }  
}
```

A flexible way to implement
programming languages

See basic/ folder

A simple way to implement
programming languages

See document/ folder

```
Hi {rokt.firstname || "there"}!
```

Hi @(or rokt.firstname {there})!

See attribute/ folder

William Byrd on

"The Most Beautiful Program Ever Written"