javascript, good & bad

Daniel Jackson
# dynamic vs. static languages

<table>
<thead>
<tr>
<th>conventional phasing</th>
<th>conventional languages</th>
<th>dynamic languages</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>compile time</strong></td>
<td>Fortran</td>
<td>Smalltalk</td>
</tr>
<tr>
<td>process decls</td>
<td>Algol</td>
<td>LISP</td>
</tr>
<tr>
<td>layout code &amp; data</td>
<td>Pascal</td>
<td>Scheme</td>
</tr>
<tr>
<td><strong>runtime</strong></td>
<td>CLU</td>
<td>Perl</td>
</tr>
<tr>
<td>run code</td>
<td>Ada</td>
<td>Lua</td>
</tr>
<tr>
<td>modify data</td>
<td>ML</td>
<td>Ruby</td>
</tr>
<tr>
<td></td>
<td>C/C++</td>
<td>Javascript</td>
</tr>
<tr>
<td></td>
<td>Java</td>
<td>Python</td>
</tr>
<tr>
<td></td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
## types of types

<table>
<thead>
<tr>
<th>approach</th>
<th>examples</th>
<th>catch errors at compile-time?</th>
<th>prevent memory corruption?</th>
<th>check #function arguments?</th>
<th>no implicit conversions?</th>
</tr>
</thead>
<tbody>
<tr>
<td>safe, static</td>
<td>Ada, Java, Pascal, Scala</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>safe, dynamic</td>
<td>Scheme, Smalltalk</td>
<td>✗</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>weakly typed dynamic</td>
<td>Javascript, Python</td>
<td>✗</td>
<td>✔</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>unsafe</td>
<td>C, C++</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✔</td>
</tr>
</tbody>
</table>

- ✔: True
- ✗: False
a subjective view of JavaScript

**good parts**
- first-class functions
- lexical scope & closures
- simple, uniform semantics

**debateable parts**
- prototypes
- dynamic types
- extensible objects

**bad: can work around**
- default global scope
- new and this
- semicolon insertion

**awful: stuck with these**
- implicit conversions
- no packaging
why I love JavaScript

lexical scope & closures
› Java: no standalone functions
› Ruby: complex & messy closures
› Python: no real closures (vars are readonly)

simple clean syntax
› Python: indentation rules
› Ruby: whitespace sensitivity
› LISP: tons of parens
› Java: so many characters to type...

uniform semantics
› at least for the core
why I hate JavaScript

the bad parts
› funny things (==, NaN, etc)
› silent failures
› implicit conversions

no packaging constructs
› to organize a large program
› cf. Java’s package namespace

no access control
› to implement data abstraction, eg
› can use closures, but not for two objects
› cf. Java’s private fields
data abstraction example

```javascript
var Set = function () {
  var elts = []; // invariant: no duplicates
  var that = createObject(Set.prototype);
  that.each = function (body) {
    elts.forEach(function (e) {body(e);});
  }
  that.insert = function (e) {
    if (!that.contains(e)) elts.push(e);
  }
  that.size = function () {
    return elts.length;
  }
  that.contains = function (e) {
    var result = false;
    that.each(function (x) {
      if (x === e) result = true;
    });
    return result;
  }
  that.union = function (s) {
    s.each(function (e) {that.insert(e);});
  }
  Object.freeze(that); // prevent slot mods
  return that;
}
```

s must be treated as abstract by “method”
implicit conversions & solvents

isopropyl alcohol
useful for cleaning things?

hydrofluoric acid
useful for dissolving rocks?

distinguish language features
› risky but helpful: dynamic types? growable objects?
› more trouble than they’re worth: implicit conversions
strategy for bad parts

two strategies for dealing with language features
› learn all the details and use as many as possible
› identify a core of reliable, strong features & ignore rest
define your own subset!

It is rarely possible for standards committees to remove imperfections from a language... But you have the power to define your own subset.

Douglas Crockford, in *Javascript: The Good Parts*

*The Skater’s Principle of Language Use:*
Stay in the middle, where the ice is thicker.

Michael Jackson, in *Software Requirements & Specifications*

There is danger and misery at the edges.

Douglas Crockford, in *Javascript: The Good Parts*