namespaces & variables

Daniel Jackson

For now, all rights reserved. Daniel Jackson, 2012.
not a software designer...
not a software designer...

What’s in a name? That which we call a rose
By any other name would smell as sweet.
—Shakespeare; Romeo & Juliet, 2:2
namespaces
namespaces

context matters
› same name, different meaning

- Namespace
  - bindings
  - Binding
    - name
    - value
      - Name
      - Thing
namespaces

context matters
› same name, different meaning

applications of this idea
› program elements
› state components
› files & directories
› URLs & routing
› ...

Diagram:
- Namespace
  - Bindings
    - Name
    - Thing
    - value
    - name
environments

Diagram:

- Environment
  - bindings
    - Binding
      - var
      - value
        - Variable
        - Value
environments

environment

.namespace for program variables

\[
\text{Environment} \quad \text{bindings} \quad \text{Binding} \quad \text{var} \quad \text{value} \\
\text{Variable} \quad \text{Value}
\]
environments

environment
› namespace for program variables

in Javascript
› every bound variable has a value
› value may be “undefined”
environments

environment
› namespace for program variables

in Javascript
› every bound variable has a value
› value may be “undefined”

confusing
› unbound var gives ref error
› property can only be undefined
› null is a defined object value
lookup

```
> h = "hello there"
"hello there"
> escape
function escape()
{ [native code] }
> escape(h)
"hello%20there"

> foo
ReferenceError
> escape(foo)
ReferenceError
> var bar
undefined
> bar
undefined
> escape(bar)
"undefined"
> undefined
undefined
```
lookup

to evaluate an expression
 › lookup value of each var
 › apply functions to arguments

> h = "hello there"
"hello there"
> escape
default escape()
{ [native code] }
> escape(h)
"hello%20there"

> foo
ReferenceError
> escape(foo)
ReferenceError
> var bar
undefined
> bar
undefined
> escape(bar)
"undefined"
> undefined
undefined
lookup

to evaluate an expression
  › lookup value of each var
  › apply functions to arguments

how to lookup
  › just find the binding for the var

> h = "hello there"
"hello there"
> escape
function escape()
{ [native code] }
> escape(h)
"hello%20there"

> foo
ReferenceError
> escape(foo)
ReferenceError
> var bar
undefined
> bar
undefined
> escape(bar)
"undefined"
> undefined
undefined
assignment

> h = "hello there"
"hello there"
> escape(h)
"hello%20there"
> escape = function() 
  {return "gone!";}
function () {return "gone!";}
> escape(h)
"gone!"

> var foo
undefined
> foo === undefined
ture
> undefined = 3
3
> foo === undefined
false
assignment

assignment statement

\[ x = e, \text{ read } "x \text{ gets } e" \]
assignment

assignment statement
› x = e, read “x gets e”

semantics
› evaluate e to value v
› if x is bound, replace value with v
› else create new binding of x to v

> h = "hello there"
"hello there"
> escape(h)
"hello%20there"
> escape = function()
{return "gone!";}
function () {return "gone!";}
> escape(h)
"gone!"

> var foo
undefined
> foo === undefined
true
> undefined = 3
3
> foo === undefined
false
assignment

assignment statement
› x = e, read “x gets e”

domains
› evaluate e to value v
› if x is bound, replace value with v
› else create new binding of x to v

in JS, all names are vars
› function names can be reassigned
› can define “undefined” (!)

domains

> h = "hello there"
"hello there"
> escape(h)
"hello%20there"
> escape = function()
{return "gone!";}
function () {return "gone!";}
> escape(h)
"gone!"

> var foo
undefined
> foo === undefined
true
> undefined = 3
3
> foo === undefined
false
aliasing

> y = []
> []
> x = y
> []
> x.f = 1
> 1
> y.f
> 1
aliasing

after the assignment $x = y$

\[ x \text{ is bound to same value as } y \]
aliasing

after the assignment $x = y$
▷ $x$ is bound to same value as $y$

how sharing arises
▷ no implicit copying
▷ so $x$ and $y$ are names for same object

```> y = []
[]
> x = y
[]
> x.f = 1
1
> y.f
1```
**aliasing**

after the assignment \( x = y \)

› \( x \) is bound to same value as \( y \)

**how sharing arises**

› no implicit copying

› so \( x \) and \( y \) are names for **same** object

**consequence**

› change to “one” affects the “other”
**aliasing**

after the assignment $x = y$

→ $x$ is bound to same value as $y$

**how sharing arises**

→ no implicit copying

→ so $x$ and $y$ are names for *same* object

**consequence**

→ change to “one” affects the “other”

**if object is immutable**

→ no change to object possible

→ so as if value is copied

```plaintext
> y = []
[]
> x = y
[]
> x.f = 1
1
> y.f
1
```
evaluating expressions
evaluating expressions

suppose you see an expression \( e \)

\( \rightarrow \) eg, \( e \) is \( f() \)

\( \rightarrow \) what might expression do?
evaluating expressions

suppose you see an expression e
  › eg, e is f()
  › what might expression do?

evaluation can have 3 effects
  › value is returned (or exception thrown)
  › objects are modified
  › environment is updated
evaluating expressions

suppose you see an expression e
› eg, e is f()
› what might expression do?

evaluation can have 3 effects
› value is returned (or exception thrown)
› objects are modified
› environment is updated

a puzzle
› declare f so that f()==f() evals to false
solution to puzzle

```javascript
f = function () {
    f = function () {
        return 1;
    }
    return 2;
}
```
solution to puzzle

one of several possible

```javascript
function () {
    function () {
        return 1;
    }
    return 2;
}
```