Home

01 — Welcome
JavaScript basics
Design as communication
Homework

02 — Paperwork
Some more basics
Closures
Introducing Paper.js
Homework

03 — Paperworked
Introducing Skip.js
Homework

04 — Paper Squares
Homework

05 — Powers of Three
06 — Code is data
07 — Midterm critiques
08 — Observe and report
09 — Poster break
10 — Patterning
11 — TBD
12 — TBD
13 — Last chances
14 — Final critiques

Skip.js

Skip is a tiny JavaScript library that focuses on the language itself, rather than browser-related tasks like searching the DOM or handling Ajax requests. It adds little helper functions and properties to JavaScript through prototypal inheritance. Skip's goal is to add a few bits that should have been there already, and to make code more legible from left-to-right like English. (This is primarily accomplished through "chaining" as we'll see below.)

I've included Skip in your template packages and here in the course notes site. (So if you open up your browser's console right now you'll have access to Skip's functionality.) You can also visit Skip's GitHub page to download the latest copy. Full disclosure: This is my own personal library that I created to make my life easier and in sharing it with you I hope it will do the same.

Fun with dates

Dealing with time in JavaScript can often be annoying. The only time units JavaScript can speak in is milliseconds. There's no built-in units for seconds, minutes, hours, etc. so you must multiply out milliseconds × 60 × 60 × 24... Skip let's you skip this tedious task by adding the following constants to the global scope:

```
SECOND  = 1000
MINUTE  = SECOND *  60
HOUR    = MINUTE *  60
DAY     = HOUR   *  24
WEEK    = DAY    *   7
MONTH   = DAY    *  30.4368499
YEAR    = DAY    * 365.242199
DECADE  = YEAR   *  10
CENTURY = YEAR   * 100
```

That's convenient, sure, but not altogether exciting. So we take it a step further. Skip augments the Number prototype with functions that multiply numbers by time units, like so:

```
var two = 2
undefined
two.seconds()
```

```
2000
```

But this applies to *all* Number objects, including number literals. We just have to wrap the number literal in parenthesis, otherwise JavaScript will think our dot is supposed to be a decimal point and fail when it reaches something other than another numerical digit after the decimal.

```
(2).seconds()
2000
(3).weeks()
1814400000
(4).days() + (6).hours() + MINUTE
367260000
```

Function chaining

When a function returns a useful value you can immediately chain another method on to it, as we'll do below. We often speak in *relative* time—how long ago something happened, or how far into the future from now something might happen.

```
now()
1349221607833
(4).days().ago()
1348876007833
(4).days().fromNow()
1349567207833
```

But that would be a lot more helpful if we could quickly check that those abstract numbers make human sense.

```
now().toDate()
Tue Oct 02 2012 19:46:47 GMT-0400 (EDT)
(4).days().ago().toDate()
Fri Sep 28 2012 19:46:47 GMT-0400 (EDT)
(4).days().fromNow().toDate()
Sat Oct 06 2012 19:46:47 GMT-0400 (EDT)
```

More easy math

When you need to check that a value `n` is contained in a range from `a` to `b` you might write something like this:

```
var a = 0, b = 2, n = 1
```

```
undefined
a <= n && n <= b
true
```

Skip reads a little closer to English.

```
var a = 0, b = 2, n = 1
undefined
n.isBetween( a, b )
true
```

And remember, dates are also just numbers.

```
var
yesterday = now() - DAY,
tomorrow  = now() + DAY,
today     = now()
undefined
today.isBetween( yesterday, tomorrow )
true
```

These are just a few features that Skip adds to JavaScript. Let's open up Skip's source code and peer a little deeper. What other constants does it add to the global scope? What other functions does it add to the Number object? Strings? Arrays? Here's a good example to end on as we can discuss the Array-like `arguments` property of functions as well as polymorphism:

```
[ 0 ].distanceTo( 7 )
7
[ 0, 0 ].distanceTo( 0, 7 )
7
[ 0, 0, 0 ].distanceTo( 0, 0, 7 )
7
[ 0, 0, 0 ].distanceTo([ 0, 0, 7 ])
7
```