Extending the CSS Mentality to Practical Rule-Based Screen Readers

Hi everybody!

This is “Extending the CSS mentality to practical rule-based screen readers”

I realize now that that’s an incredibly long name. So what am I gonna talk about?

Configurable Screen Readers

Why & How

wanna talk about the whys and the hows of making screen readers more configurable

first, a bit about me

I’m Ben Stolovitz
19 and 3/4
soph cs at WashU

for past 2-3 summers worked at GT Son Lab w/ Walker

that’s where stumbled on question about config screen readers

Configurable Screen Readers

Why & How

why should we make our screen readers more configurable, and how do we do that

let’s talk about the why

why is it worth the effort of making a screen reader more configurable?

Why?

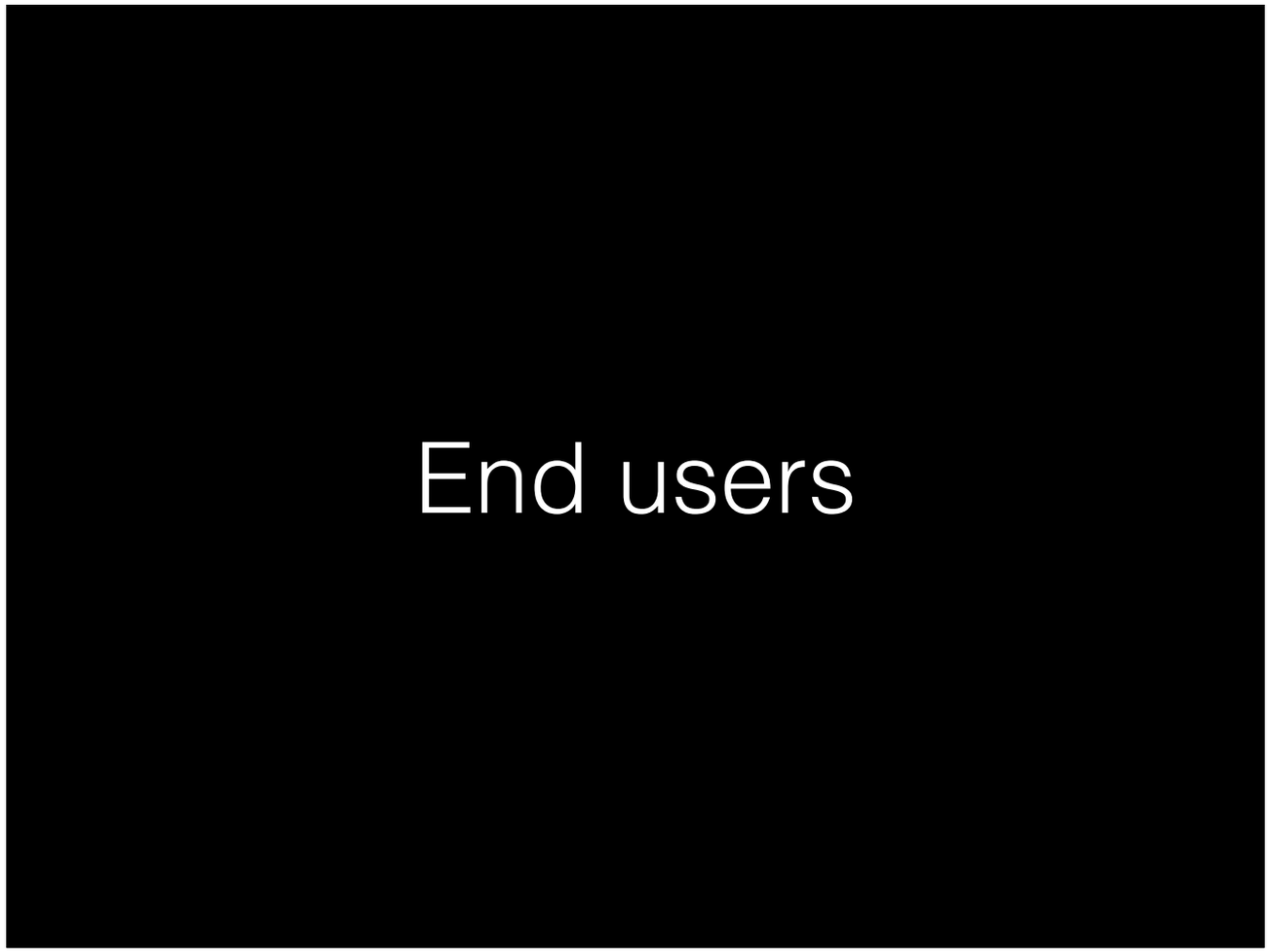
- End Users
- Researchers

what I’ve seen is

customizability improves things for two main groups

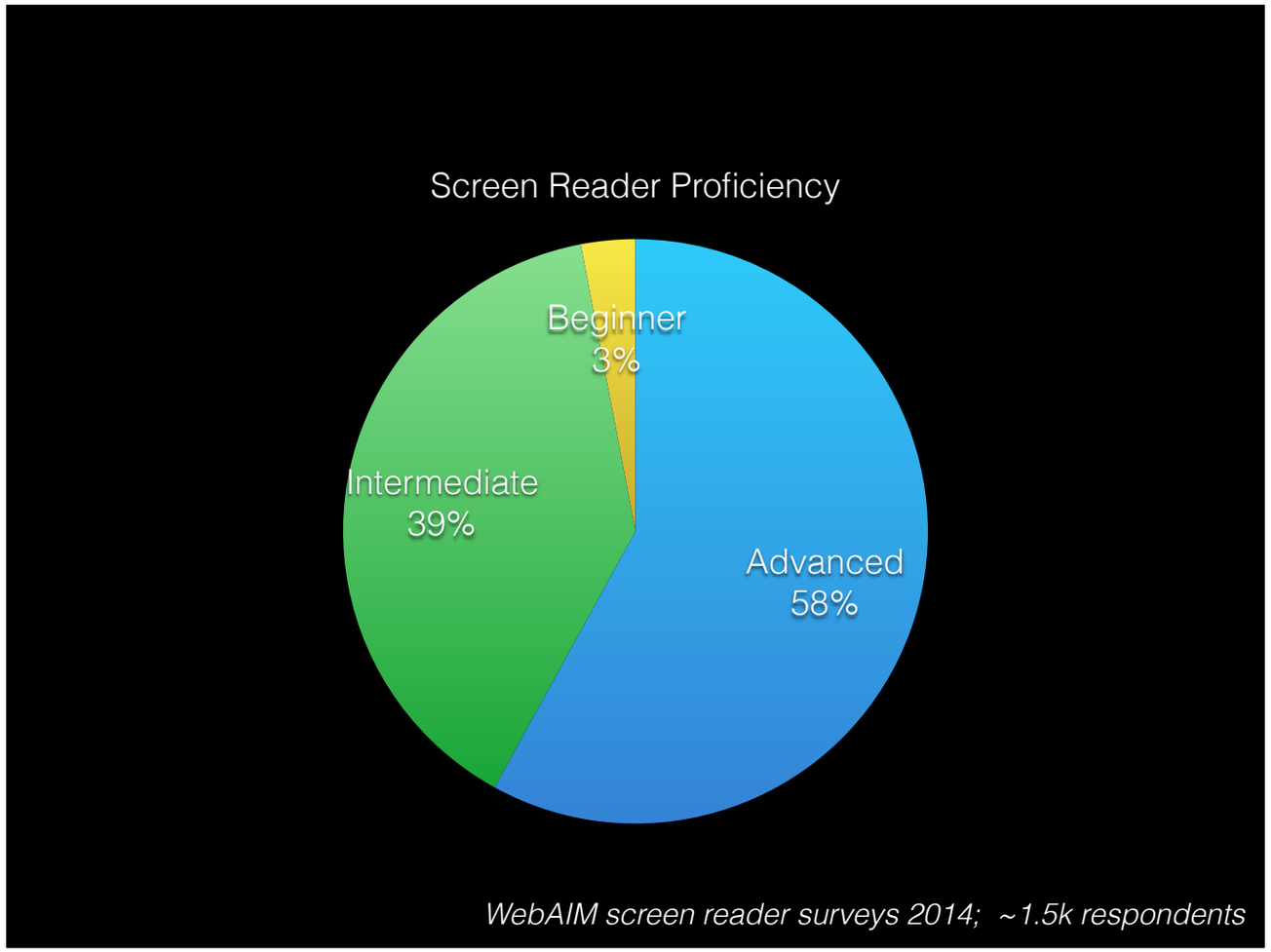
end users
and researchers

basically anyone who uses a screen reader,
and anyone who wants to improve one



of course end users benefit from more customizability

but we aren’t giving users enough



this is webaim data from January.

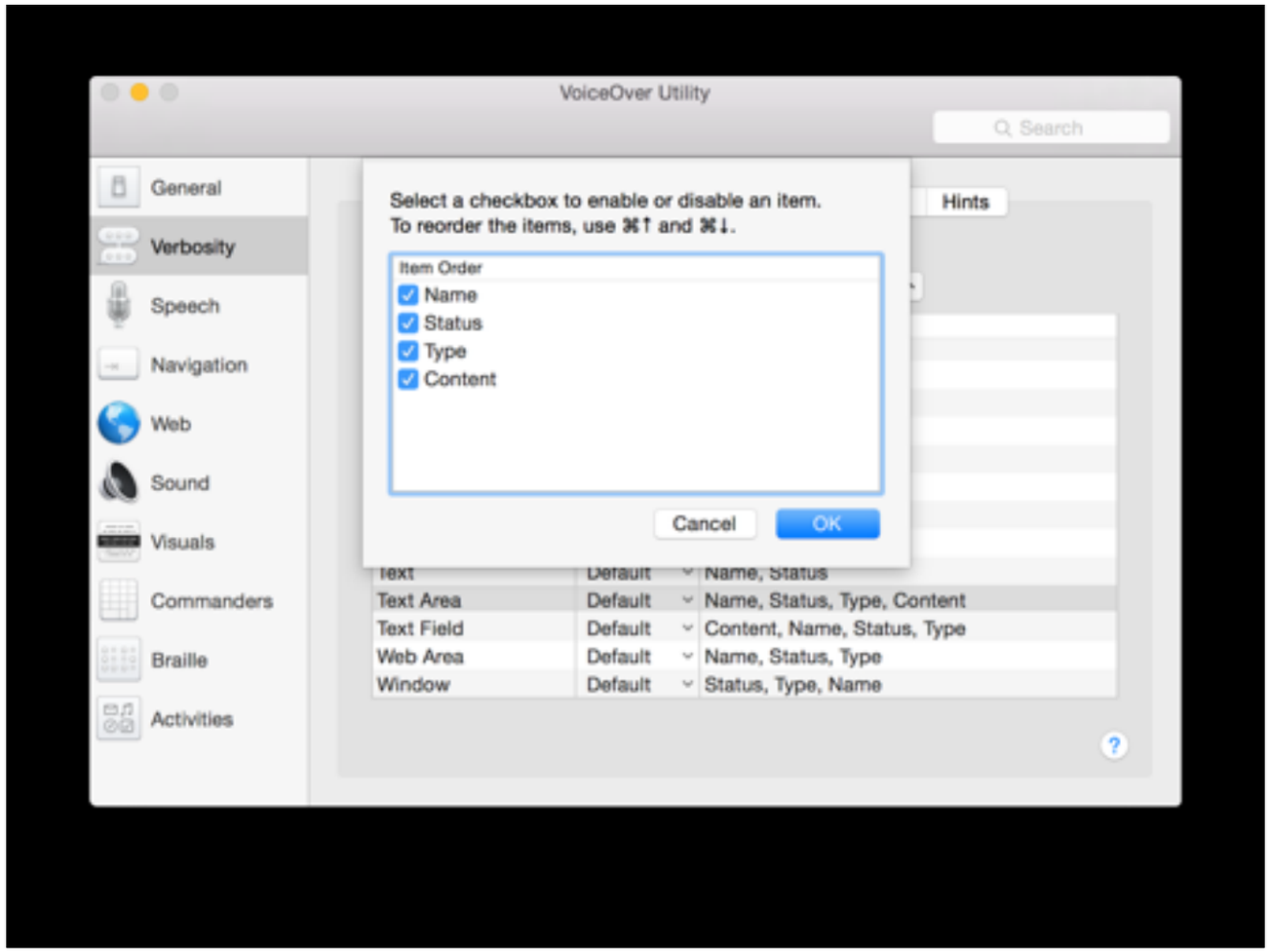
Most screen reader users rate themselves as “advanced”, or at least “intermediate”

71% customize their screen readers

WebAIM screen reader surveys 2009; ~1k respondents

now this is older WebAIM data (2009)

but 71% of visually impaired users customize their screen readers “a lot” or “somewhat”



we’re not giving them enough.

these are the customizable options for reading text boxes in Voiceover.
I can check or uncheck “name”, “status”, “type”, and “content”.

that’s useful, sure, but where are the advanced options, the custom ticks and prompts that powerusers demand?

(https://lclarkse.files.wordpress.com/2011/04/computer-tree.jpg)

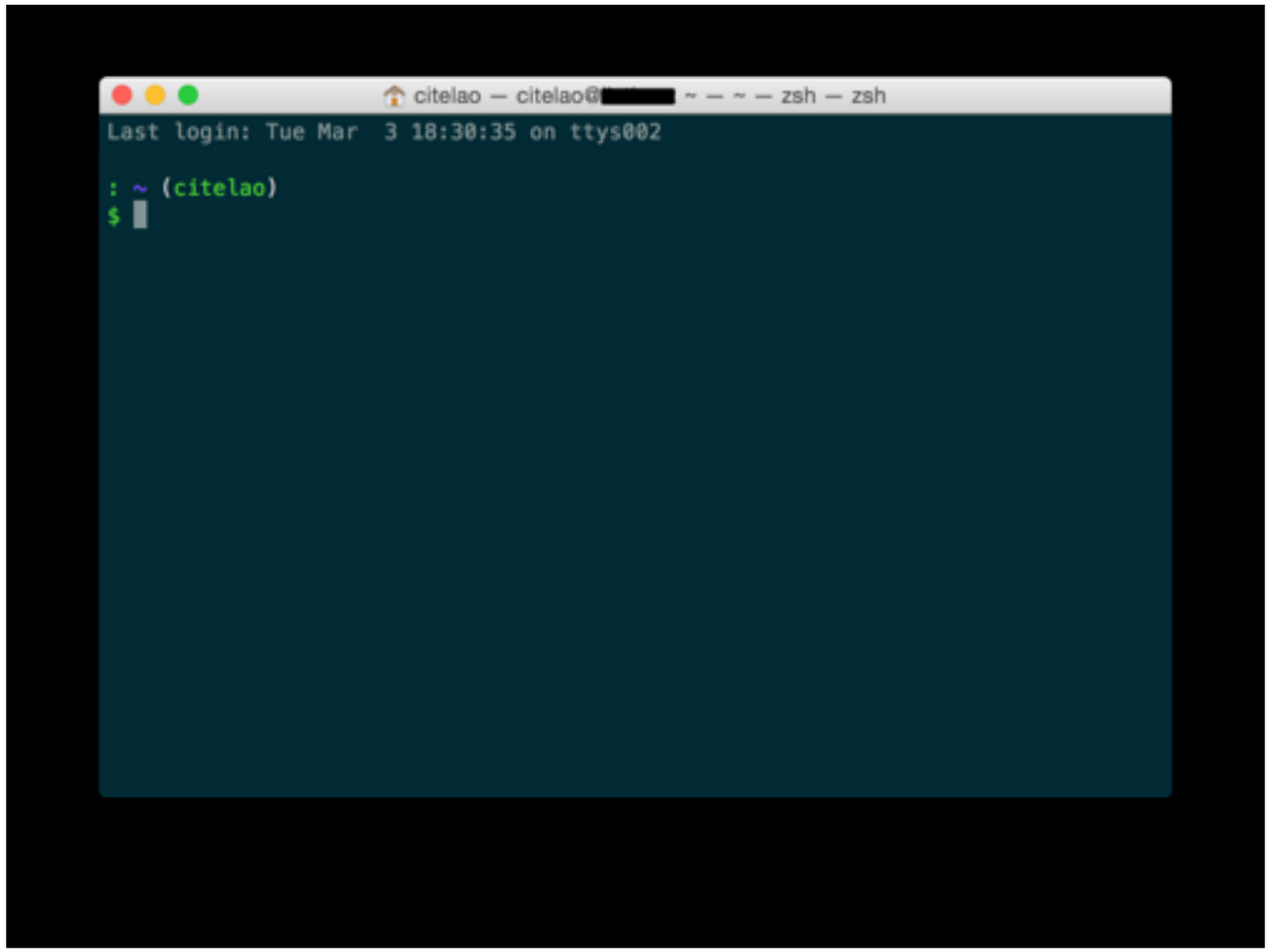
screen readers,

- voiceover
- jaws
- nvda,
- what have you

they build hierarchical trees, with nodes and branches and navigation patterns.

their prime job is transforming the screen into a hierarchical structure, using very complex rules and inferences.

yet somehow, all the end user gets to choose is whether to actually read out the content in a text box or not.



I know advanced computer users.
I am an advanced computer user.

And I *love* customization.

This is my terminal prompt.
If you are also an advanced computer user, and you are not me,
you probably hate it.

We have strong feelings about our computers. We make them *ours*.

We can’t do that if we can’t customize them.

Researchers

the kind of deep customization I would like to see would also help researchers.

research is kind of hard now.
why?

because every time we want to test a new method of navigation,
every time we want to test a new way of outputting content,

```
function key_navigate(event) {

    // left
    if (event.keyCode == 37) {

        var prev;

        // 'this' refers to the <a> tag which is in focus


        // compensate for the section spacer

        if ($(this).closest('td').prev().hasClass('sectionLeadingSpacer')) {


            // check if the destination element is the move button
            if ($(this).closest('td').prev().prev().hasClass('movebutton')) {

                prev = $(this).closest('td').prev().prev().find('input');
            } else {
                prev = $(this).closest('td').prev().prev().find('a');
            }
        } else {

            if ($(this).closest('td').prev().hasClass('movebutton')) {

                prev = $(this).closest('td').prev().find('input');
            } else {

                prev = $(this).closest('td').prev().find('a');
            }
        }

        current = prev;
        recent = $(this);
        prev.focus();

    }

    // up
    else if (event.keyCode == 38) {

        var upper;

        var classname;


        // 'this' refers to the <a> tag which is in focus

        classname = $(this).parent().attr("class");

        var temp = $(this).parent().parent().prev().children('td');
        console.log(temp);

        temp.each(function(index, value) {

            if (jQuery(value).attr('class') === classname) {
                upper = jQuery(value).find('a').first();

                return false;
            }
        });

        upper.focus();

        calcPlayer(upper);
    }

    // right
    else if (event.keyCode == 39) {

        var next;

        // 'this' refers to the <a> tag which is in focus

        if ($(this).closest('td').next().hasClass('sectionLeadingSpacer')) {
            next = $(this).closest('td').next().next().find('a');
        } else {

            if ($(this).closest('td').next().hasClass('movebutton')) {

                next = $(this).closest('td').next().find('input');
            } else {
                next = $(this).closest('td').next().find('a');

            }
        }

        next.focus();
    }

    // down
    else if (event.keyCode == 40) {

        var lower;

        var classname;

        // 'this' refers to the <a> tag which is in focus

        classname = $(this).parent().attr("class");

        var temp = $(this).parent().parent().next().children('td');
        console.log(temp);

        temp.each(function(index, value) {

            if (jQuery(value).attr('class') === classname) {
                lower = jQuery(value).find('a').first();

                return false;
            }
        });

        lower.focus();

        calcPlayer(lower);
    }

    // jump to player
    else if (event.keyCode == 74) {


        recent = $(this);

        jumpToPlayer();

    }

    // jump back to recent element in focus
    else if (event.keyCode == 75) {

        recent.focus();
    } else if (event.keyCode == 88) {
        var cur;

        cur = $(this).closest('td').find('a');
```

we have to write this.

this is a snippet of code from a project my friend
Jared Batterman,
testing accessible fantasy football
had new interaction method he wanted to test
lacking any easy route, he had to make this.

his team wrote 146 lines of code just to handle focus.

and it’s not generalizable.

everything’s hard coded.

We scaled up our model to drafting players, too— months of work.

this work has to be done every time we want to research a new interaction method.

we’re solving an already-solved problem. honestly kind of silly.

maybe doesn’t happen if you have access to JAWS or NVDA source, but we don’t have access to that. and a lot of people don’t.

How?

so how do we fix that?

how do we make our screen readers configurable enough for those new uses?

Rule-based Design

we work with rule-based design

we did some thinking, and it turns out modeling the content is largely context-free.
Most of the time, it doesn't matter what surrounds a piece of content: it just matters what that content is.

And so we thought about CSS.
    the way the web get its style.
All content is HTML
    which gives a tree-like view of the content
CSS "selects" certain elements of the tree
    (all links, all images, or that one image in the corner, etc)
and applies styles to them.

So why not audio styles?

Aural Stylesheets

Don’t work.

Aural stylesheets are a thing.

but no one supports them.
and they don’t solve the problem.

You can’t prepend text to them specifically for screenreaders
(not easily)
and you have no form of context
(which you need just a *little* of)

so no aural stylesheets

Rule-based Design

there’s power in the way you can select things via CSS

it can do almost anything.

Another standard.

so we built an engine that ran off our own standard,
our own rules

We designed it to be modular, hackable. The goal is flexibility, with sensible, understandable defaults.

We don’t do our own text-to-speech. We use an API. We don’t do our own keybindings, we use an API.

```
[selector | "word"] {
   tabbable: [without_children | force | inline | never];

   content: string with interpolated variables;
   content-before: ditto;
   content-after: ditto;

   before: ditto;
   after: ditto;

   pause: time;
   pause-before: time;
   pause-after: time;

   pitch: [+|-] float;

   contexts {
      selector {
         [altered values]
      }
   }
}
```

our standard reads rules that look kind of like this.

And that’s it.

That’s enough to do most of what we want.

```
// td
{
   "selector": "td",
   "content_before": "{col} {row}",

   "contexts": [
      {
         // Column change (same row, td to td)
         "selector": "table tr.focussed_parent th, table tr.focussed_parent td",
         "content_before": "{col}"
      },

      {
         // Row change (same col, tr to tr)
         "selector": "table.focussed_parent td, table.focussed_parent th",
         "content_before": "{row}"
      }
   ]
},

{
   "selector": "th, thead, thead tr, thead th, thead td, thead th[scope=col],
thead td[scope=col]",
   "tabbable": "never"
}
```

Here’s Jared’s specific navigation code now.

It’s just rules. Two rules, two specific contexts.

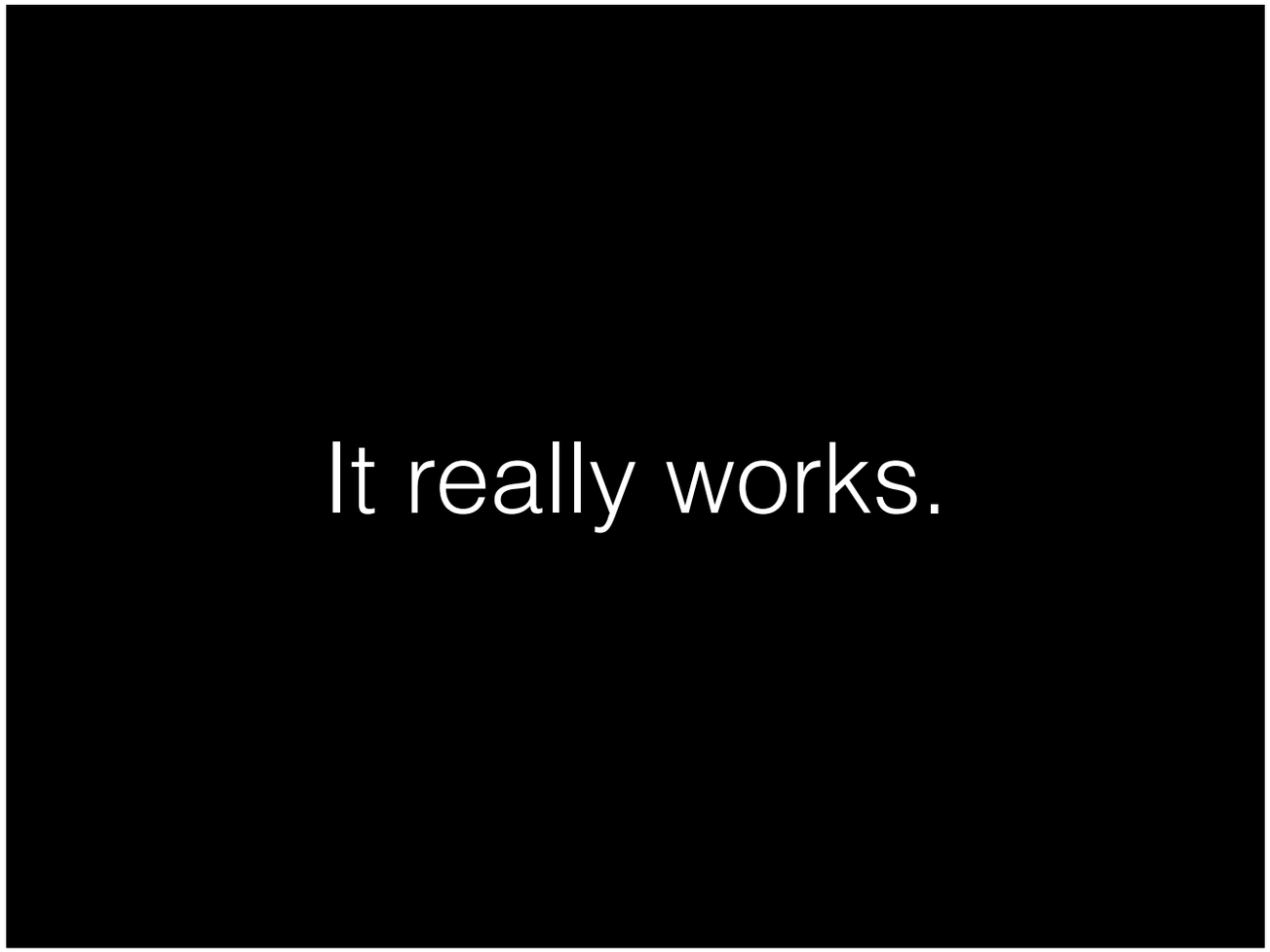
We implement our rules in JSON currently, so this code has a couple extra brackets.

```
function key_navigate(event) {

    // left
    if (event.keyCode == 37) {

        var prev;

        // 'this' refers to the <a> tag which is in focus


        // compensate for the section spacer

        if ($(this).closest('td').prev().hasClass('sectionLeadingSpacer')) {


            // check if the destination element is the move button
            if ($(this).closest('td').prev().prev().hasClass('movebutton')) {

                prev = $(this).closest('td').prev().prev().find('input');
            } else {
                prev = $(this).closest('td').prev().prev().find('a');
            }
        } else {

            if ($(this).closest('td').prev().hasClass('movebutton')) {

                prev = $(this).closest('td').prev().find('input');
            } else {

                prev = $(this).closest('td').prev().find('a');
            }
        }

        current = prev;
        recent = $(this);
        prev.focus();

    }

    // up
    else if (event.keyCode == 38) {

        var upper;

        var classname;


        // 'this' refers to the <a> tag which is in focus

        classname = $(this).parent().attr("class");

        var temp = $(this).parent().parent().prev().children('td');
        console.log(temp);

        temp.each(function(index, value) {

            if (jQuery(value).attr('class') === classname) {
                upper = jQuery(value).find('a').first();

                return false;
            }
        });

        upper.focus();

        calcPlayer(upper);
    }

    // right
    else if (event.keyCode == 39) {

        var next;

        // 'this' refers to the <a> tag which is in focus

        if ($(this).closest('td').next().hasClass('sectionLeadingSpacer')) {
            next = $(this).closest('td').next().next().find('a');
        } else {

            if ($(this).closest('td').next().hasClass('movebutton')) {

                next = $(this).closest('td').next().find('input');
            } else {
                next = $(this).closest('td').next().find('a');

            }
        }

        next.focus();
    }

    // down
    else if (event.keyCode == 40) {

        var lower;

        var classname;

        // 'this' refers to the <a> tag which is in focus

        classname = $(this).parent().attr("class");

        var temp = $(this).parent().parent().next().children('td');
        console.log(temp);

        temp.each(function(index, value) {

            if (jQuery(value).attr('class') === classname) {
                lower = jQuery(value).find('a').first();

                return false;
            }
        });

        lower.focus();

        calcPlayer(lower);
    }

    // jump to player
    else if (event.keyCode == 74) {


        recent = $(this);

        jumpToPlayer();

    }

    // jump back to recent element in focus
    else if (event.keyCode == 75) {

        recent.focus();
    } else if (event.keyCode == 88) {
        var cur;

        cur = $(this).closest('td').find('a');
```

it does all of what this does.

<25 lines

and we did it in under 25 lines



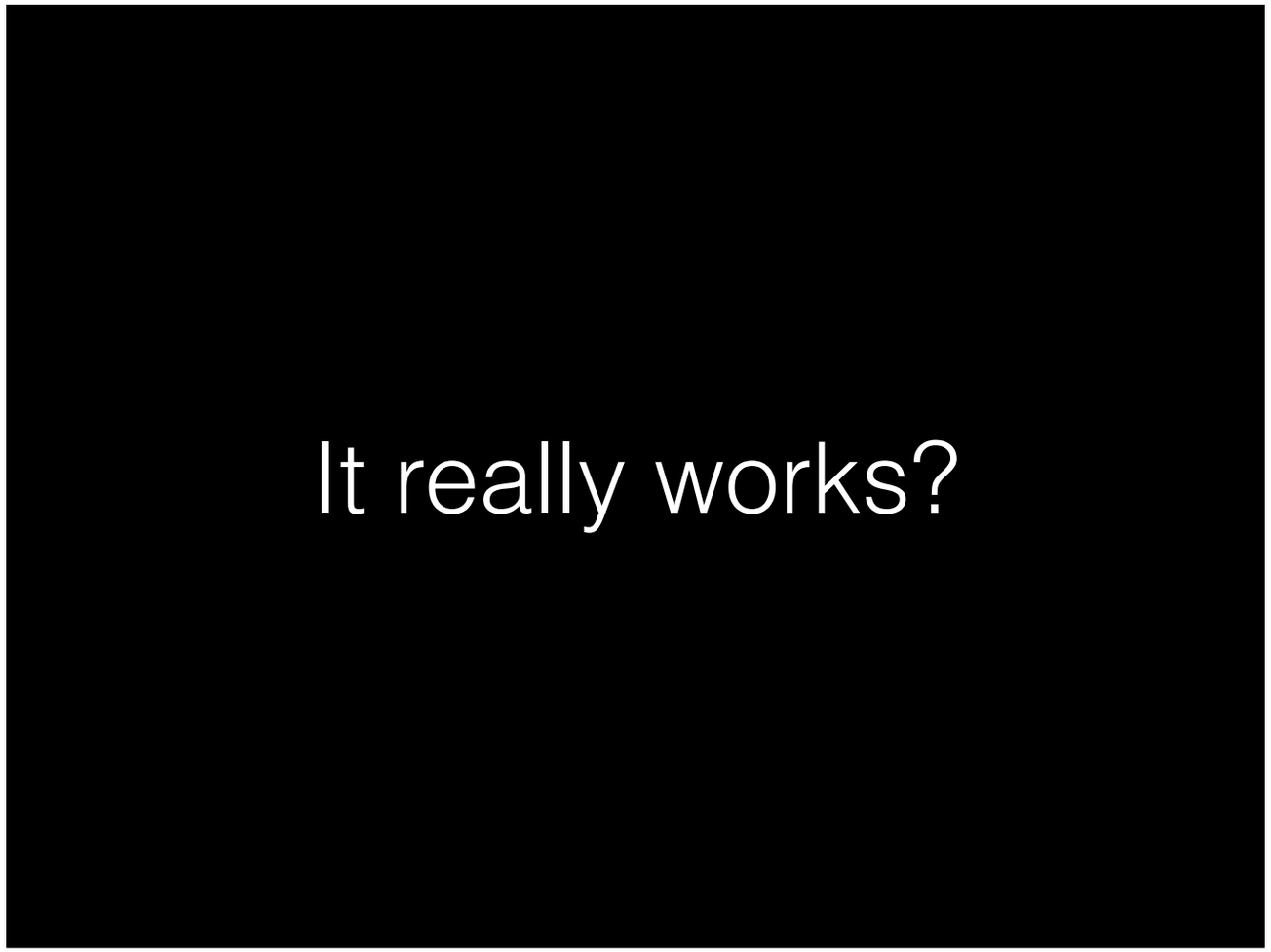
This isn’t a contrived example.

I was interested in math display after doing some research into MathJax and its awesomeness.

```
{
  "selector": "math",
  "tabbable": "force"
},

{
  "selector": "mfrac > mrow”,
  "tabbable": "force"
  "content_before": "quantity",
  "content_after": "quantity"
},

{
  "selector": "mfrac > mrow:first-of-type”,
  "tabbable": "force"
  "content_before": "quantity",
  "content_after": "all over"
}
```

So I added about 20 lines. And now it reads fractions.

Specifically, it reads fractions, and users can do navigation around the fraction, looking at the numerator and the denominator individually.



We’re really proud of it.

But it’s not the Messiah. Not yet.

We’re confident the rule-based approach can model almost every interaction, on the web, Mac, Windows, iOS, or otherwise

but that’s a tall order for me to implement by myself.

It’s just web content for now, and it doesn’t have the phonemic control or the prosody I’d like.

Right now, it doesn’t solve everything. But it can. And it already does a ton.

(demo)

So I’d like to show you!