Usability

- Summarize
- Organize
- Write compactly
- Don’t be too creative!
Navigation and links

- Menus:
  - Horizontal
  - Vertical
  - Flyout

- Efficient forms
  - Proper input elements
  - Min number of fields
  - Javascript/PHP for validation
Visual Effects
Scriptacularous overview

Scriptacularous : a JavaScript library, built on top of Prototype, that adds:

- visual effects (animation, fade in/out, highlighting)
- drag and drop
- some DOM enhancements
- other stuff (unit testing, etc.)
Downloading and using Scriptaculous

- documentation available on their [wiki](#)
- [Scriptaculous Effects Cheat Sheet](#)
Visual Effects

- appear
- blindDown
- grow
- slideDown  (appearing)

- blindUp
- dropOut
- fade
- fold
- puff
- shrink
- slideUp
- squish
- switchOff  (disappearing)

- highlight
- pulsate
- shake
- morph

- Effect.Move
- Effect.Scale
- Effect.toggle (blind)  (Getting attention)

Click effects above
Adding effects to an element

- the effect will begin to animate on screen (asynchronously) the moment you call it
- six core effects are used to implement all effects on the previous slides:

```javascript
$("sidebar").shake();
var buttons = $$("results > button");
for (var i = 0; i < buttons.length; i++) {
    buttons[i].fade();
}
```

```javascript
element.effectName(); // for most effects
// some effects must be run the following way:
new Effect.name(element or id);
```
Adding effects to an element

- many effects can be customized by passing additional options (note the `{}`)  
- options (wiki): delay, direction, duration, fps, from, queue, sync, to, transition
Adding effects to an element

- all effects have the following events that you can handle:
  - beforeStart, beforeUpdate, afterUpdate, afterFinish

- the afterFinish event fires once the effect is done animating
  - useful do something to the element (style, remove, etc.) when effect is done

```javascript
$("my_element").fade(
  {
    duration: 3.0,
    afterFinish: displayMessage
  }));

function displayMessage(effect) {
  alert(effect.element + " is done fading now!");
}
```
Adding effects to an element

- each of these events receives the Effect object as its parameter

- its properties: `element`, `options`, `currentFrame`, `startOn`, `finishOn`

- some effects (e.g. Shrink) are technically "parallel effects", so to access the modified element, you write `effect.effects[0].element` rather than just `effect.element`
Scriptaculous provides several objects for supporting drag-and-drop functionality:

- **Draggable** : an element that can be dragged
- **Draggables** : manages all Draggable objects on the page
- **Droppables** : elements on which a Draggable can be dropped
- **Sortable** : a list of items that can be reordered
- **Puzzle Game demo**
Draggable

```javascript
new Draggable(element or id, { options });
```

- specifies an element as being able to be dragged
- **options**: `handle`, `revert`, `snap`, `zindex`, `constraint`, `ghosting`, `starteffect`, `reverteffect`, `endeffect`
- **event options**: `onStart`, `onDrag`, `onEnd`
  - each handler function accepts two parameters: the Draggable object, and the mouse event
Draggable Example

```html
<div id="draggabledemo1">Draggable demo. Default options.</div>
<div id="draggabledemo2">Draggable demo. {snap: [40,40], revert: true}</div>
```

```javascript
document.observe("dom:loaded", function() {
    new Draggable("draggabledemo1");
    new Draggable("draggabledemo2", {revert: true, snap: [40, 40]});
});
```
Draggables

- a global helper for accessing/managing all Draggable objects on a page

- **properties**: drags, observers

- **methods**: register, unregister, activate, deactivate, updateDrag, endDrag, keyPress, addObserver, removeObserver, notify
Droppables

Droppables.add(element or id, {
  options
});

- To make an element react when a Draggable is dropped onto it, you’ll add it to the Droppables of the page

- **options**: accept, containment, hoverclass, overlap, greedy

- **event options**: onHover, onDrop
  
  - each callback accepts three parameters: the Draggable, the Droppable, and the event
Draggable Example

```html
<img id="product1" src="images/shirt.png" alt="shirt" />
<img id="product2" src="images/cup.png" alt="cup" />
<div id="droptarget"></div>
```

```js
document.observe("dom:loaded", function() {
    new Draggable("product1");
    new Draggable("product2");
    Droppables.add("droptarget", {onDrop: productDrop});
});

function productDrop(drag, drop, event) {
    alert("You dropped " + drag.id);
}
```
Sortable

```js
Sortable.create(element or id of list,
    { options }
);
```

- specifies a list (ul, ol) as being able to be dragged into any order
- implemented internally using Draggables and Droppables
- options: `tag`, `only`, `overlap`, `constraint`, `containment`, `format`, `handle`, `hoverclass`, `ghosting`, `dropOnEmpty`, `scroll`, `scrollSensitivity`, `scrollSpeed`, `tree`, `treeTag`

- to make a list un-sortable again, call `Sortable.destroy` on it
Sortable demo

```html
<ol id="simpsons">
  <li id="simpsons_0">Homer</li>
  <li id="simpsons_1">Marge</li>
  <li id="simpsons_2">Bart</li>
  <li id="simpsons_3">Lisa</li>
  <li id="simpsons_4">Maggie</li>
</ol>
```

```javascript
document.observe("dom:loaded", function() {
  Sortable.create("simpsons");
});
```
# Sortable demo

<table>
<thead>
<tr>
<th>event</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>onChange</td>
<td>when any list item hovers over a new position while dragging</td>
</tr>
<tr>
<td>onUpdate</td>
<td>when a list item is dropped into a new position (more useful)</td>
</tr>
</tbody>
</table>

```javascript
Sortable.create("simpsons", {
  onChange: listUpdate,
  onUpdate: listUpdate
});
```
Sortable list events example

document.observe("dom:loaded", function() {
  Sortable.create("simpsons", {
    onUpdate: listUpdate
  });
});

function listUpdate(list) {
  // can do anything I want here; effects, an Ajax request, etc.
  list.shake();
}

JS
Auto-completing text fields

- Scriptaculous offers ways to make a text box that auto-completes based on prefix strings:
  - Autocompleter.Local: auto-completes from an array of choices
  - Ajax.Autocompleter: fetches and displays list of choices using Ajax
Using Autocompleter.Local

```js
new Autocompleter.Local(
    element or id of text box,
    element or id of div to show completions,
    array of choices,
    { options }
);
```

- you must create an (initially empty) div to store the auto-completion matches
  - it will be inserted as a ul that you can style with CSS
  - the user can select items by pressing Up/Down arrows; selected item is given a class of selected
- pass the choices as an array of strings
- pass any extra options as a fourth parameter between {}
  - options: choices, partialSearch, fullSearch, partialChars, ignoreCase
Using Autocompleter.Local

```html
<input id="bands70s" size="40" type="text" />
<div id="bandlistarea"></div>
```

```js
document.observe("dom:loaded", function() {
    new Autocompleter.Local(
        "bands70s",
        "bandlistarea",
        ["ABBA", "AC/DC", "Aerosmith", "America",
        "Bay City Rollers", ...],
        {});
});
```
Using Autocompleteer.Local

```html
<input id="bands70s" size="40" type="text" />
<div id="bandlistarea"></div>
```

```css
#bandlistarea {
    border: 2px solid gray;
}
/* 'selected' class is given to the autocomplete item currently chosen */
#bandlistarea .selected {
    background-color: pink;
}
```
Using Ajax.Autocompleter

- when you have too many choices to hold them all in an array, you can instead fetch subsets of choices from the server using Ajax
- instead of passing choices as an array, pass a URL from which to fetch them
  - the choices are sent back from the server as an HTML ul with li elements in it
- options: paramName, tokens, frequency, minChars, indicator, updateElement, afterUpdateElement, callback, parameters

```javascript
new Ajax.Autocompleter(
  element or id of text box,
  element or id of div to show completions,
  url,
  { options }
);
```

JS
Playing sounds (API)

<table>
<thead>
<tr>
<th>method</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sound.play(&quot;url&quot;);</td>
<td>plays a sound/music file</td>
</tr>
<tr>
<td>Sound.disable();</td>
<td>stops future sounds from playing (doesn't mute any sound in progress)</td>
</tr>
<tr>
<td>Sound.enable();</td>
<td>re-enables sounds to be playable after a call to Sound.disable()</td>
</tr>
</tbody>
</table>

```javascript
Sound.play("music/java_rap.mp3");
Sound.play("music/wazzaaaaaap.wav");
```

- to silence a sound playing in progress, use
  ```javascript
  Sound.play('', {replace: true});
  ```
- cannot play sounds from a local computer (must be uploaded to a web site)
new Ajax.InPlaceEditor(element or id, url, { options });

- **options**: okButton, okText, cancelLink, cancelText, savingText, clickToEditText, formId, externalControl, rows, onComplete, onFailure, cols, size, highlightcolor, highlightendcolor, formClassName, hoverClassName, loadTextURL, loadingText, callback, submitOnBlur, ajaxOptions

- **event options**: onEnterHover, onLeaveHover, onEnterEditMode, onLeaveEditMode
Ajax.InPlaceEditor

new Ajax.InPlaceCollectionEditor(element or id, url, {
    collection: array of choices,
    options
});

- a variation of Ajax.InPlaceEditor that gives a collection of choices
- requires collection option whose value is an array of strings to choose from
- all other options are the same as Ajax.InPlaceEditor
Ajax.InPlaceEditor

- **slider control**:

```javascript
new Control.Slider("id of knob", "id of track", {options});
```

- **Builder** - convenience class to replace `document.createElement`:

```javascript
var img = Builder.node("img", {
  src: "images/lolcat.jpg",
  width: 100, height: 100,
  alt: "I can haz Scriptaculous?"
});

\$('main').appendChild(img);
```

- **Tabbed UIs**