Direct Manipulation

No screens

Prof. Lydia Chilton
COMS 4170
18 February 2019

Say your name

Homework 4 Review

Goal: Recreate Twitter UI to Post a tweet

Write a tweet

Call me Ishmael. Some years ago—
never mind how long precisely—having

-37 Post Tweet

POSTS

chilton	Most recent post
chilton	More ambitious third post
chilton	Second post
chilton	First post

HTML: Why do I have the tweet_length_feedback div?

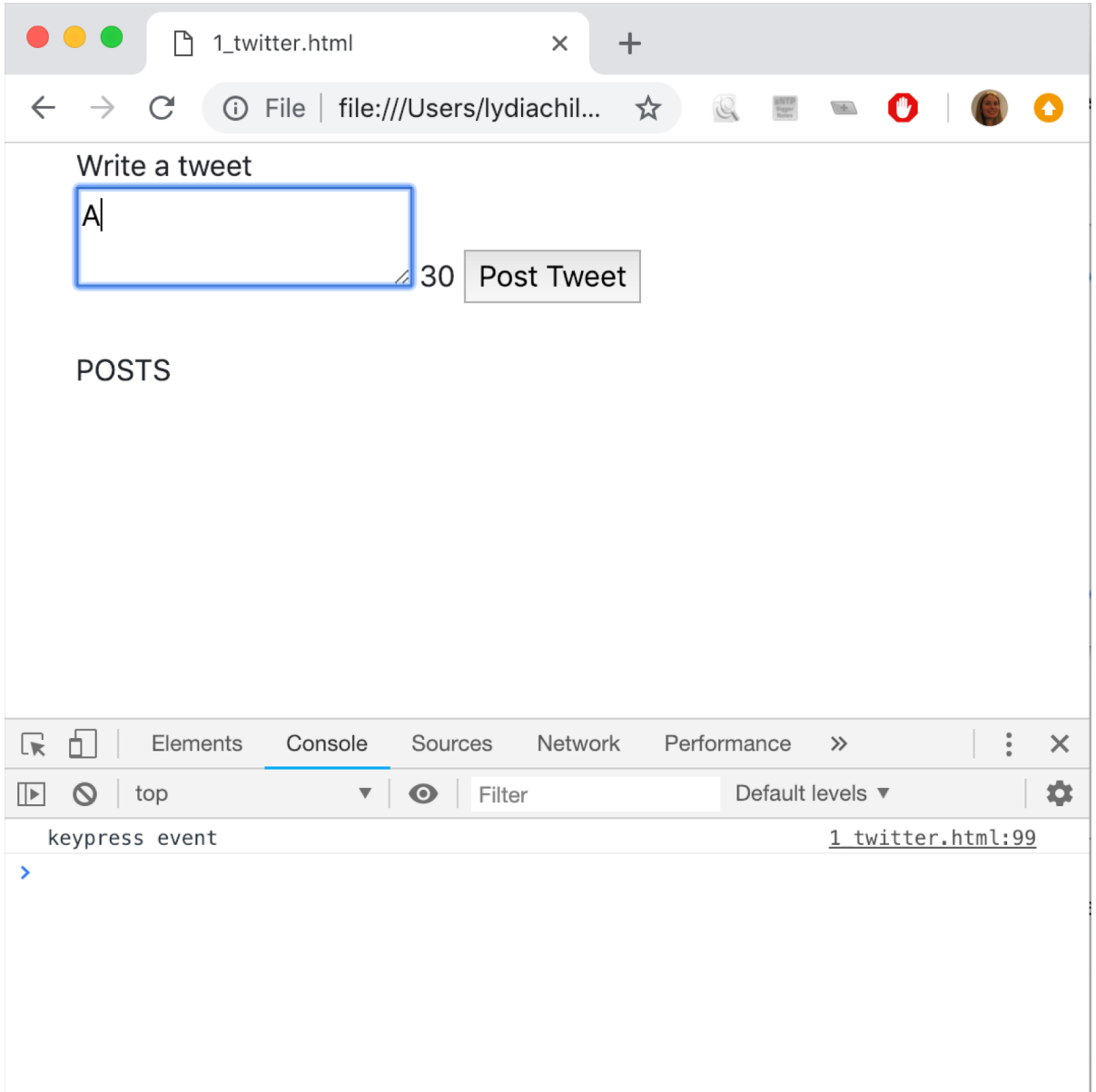
```
17    #tweet_length_feedback{
18      width: 20px;
19      display: inline-block;
20    }
```


```
100   <div>Write a tweet</div>
101   <textarea id="write_tweet"></textarea>
102   <div id="tweet_length_feedback"></div>
103   <button id="post_tweet"> Post Tweet</button>
```

JS: What does this do?

```
95
96      $(document).ready(function(){
97        $("#write_tweet").keypress(function(){
98          var tweet_text = $(this).val()
99          console.log("keypress event")
100       })
101     })
102
```

Line 97 attaches a “keypress” event listener to the textarea

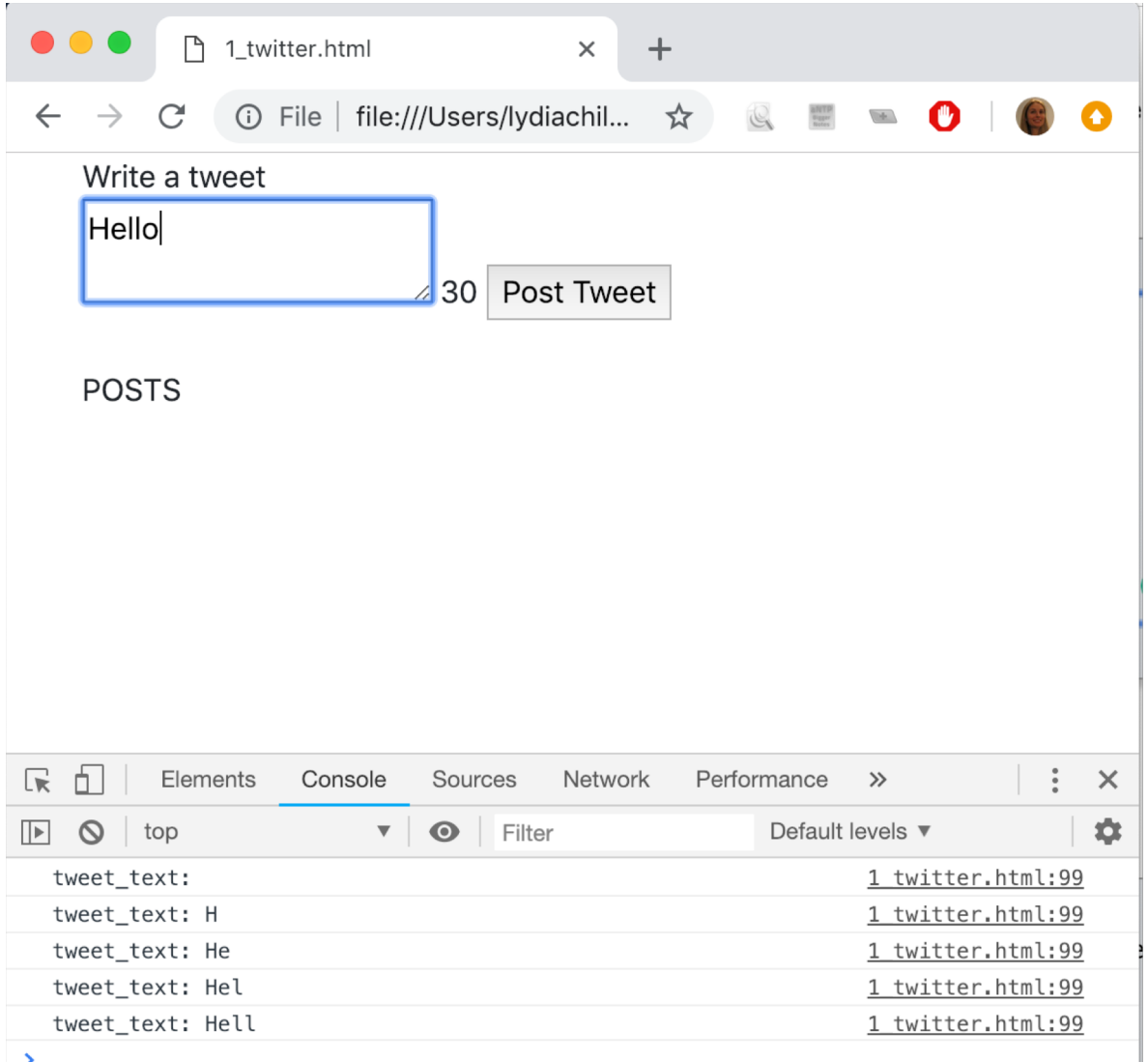


What does this do?
Why is this a good next step?

```
96
97      $("#write_tweet").keypress(function(){
98        var tweet_text = $(this).val()
99        console.log("tweet_text: "+tweet_text)
100
101
102     })
103
```

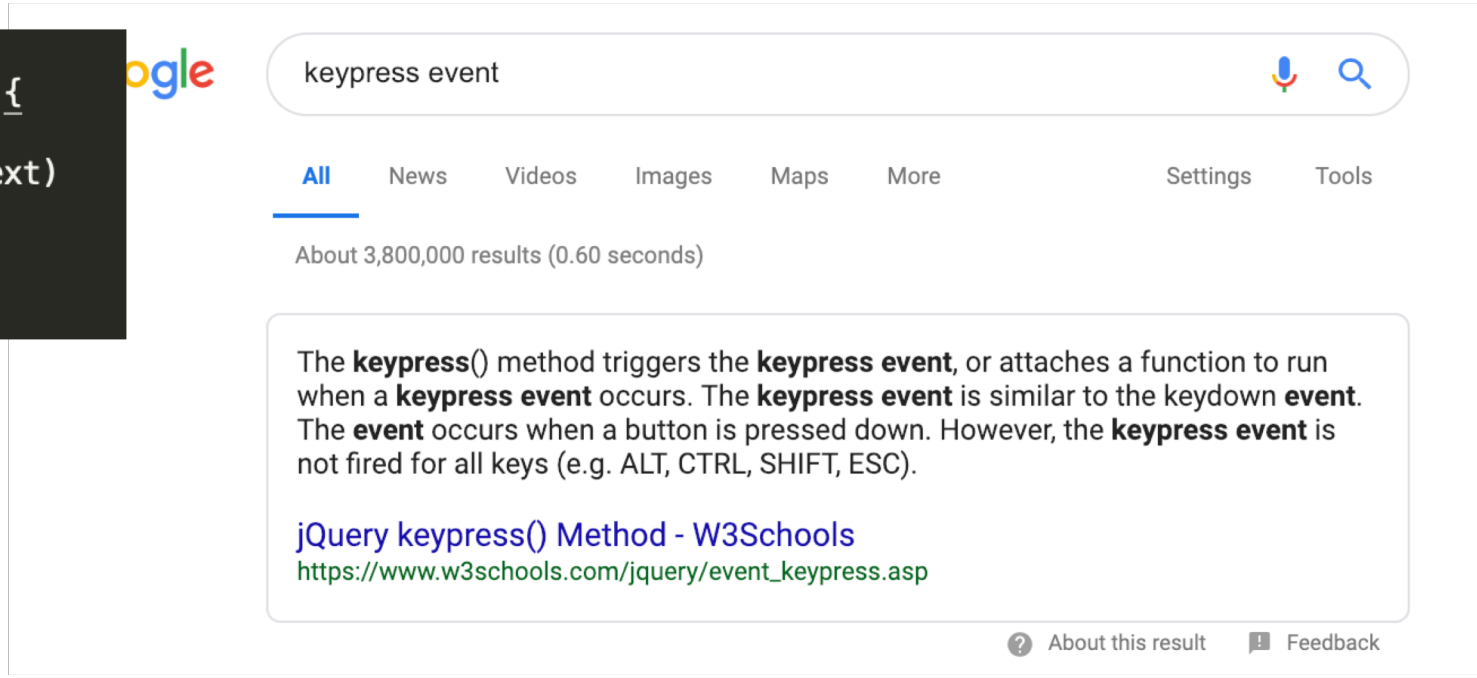
It allows us to check if line 98 does what we expect.

Does it?



Not quite. What might be the problem?

```
96
97          $("#write_tweet").keypress(function(){
98            var tweet_text = $(this).val()
99            console.log("tweet_text: "+tweet_text)
100
101
102         })
```

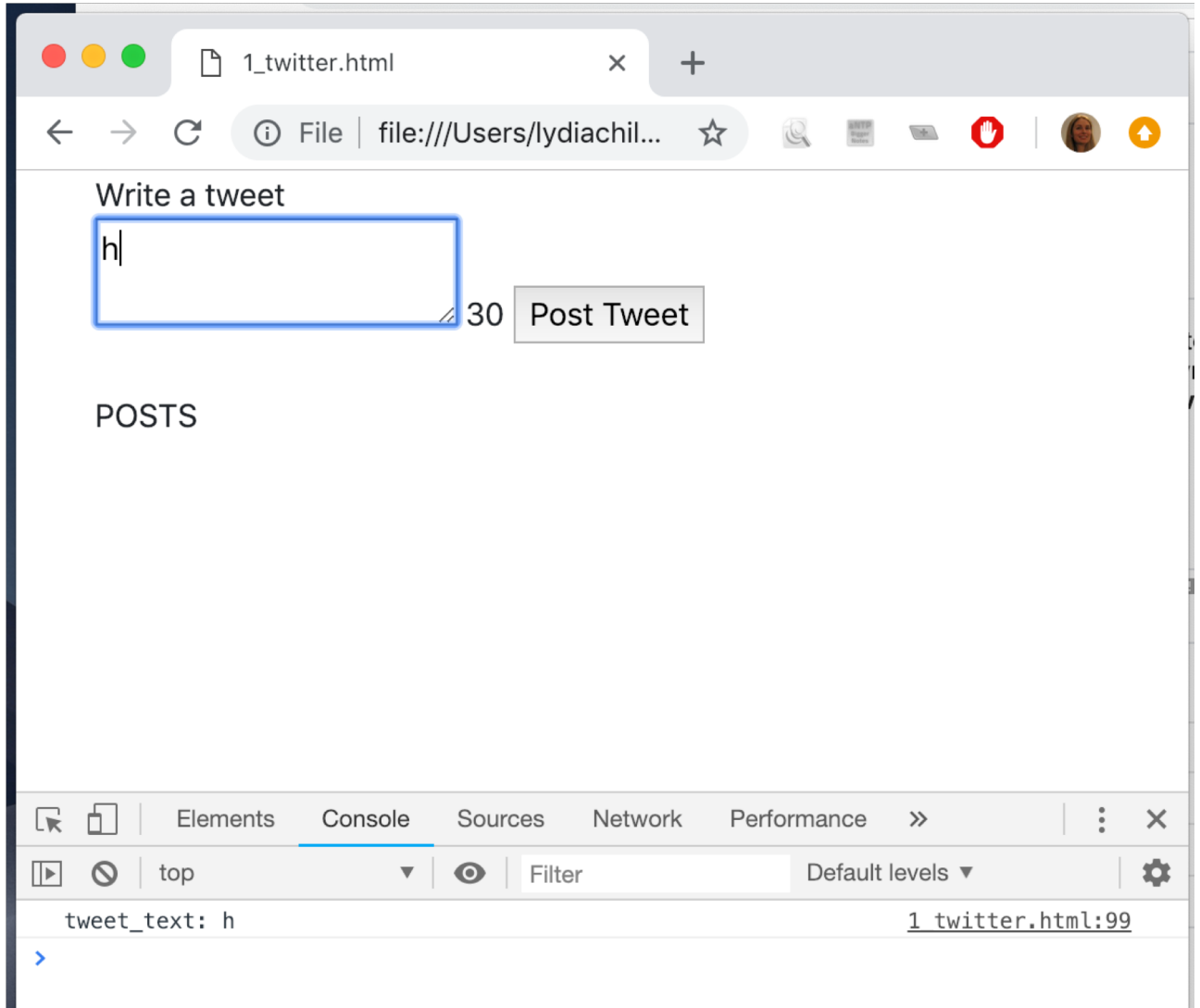


What should we try to fix?

Keyup event seems better. Does it work?

```
96
97          $("#write_tweet").keypress(function(){
98            var tweet_text = $(this).val()
99            console.log("tweet_text: "+tweet_text)
100
101
102         })
```

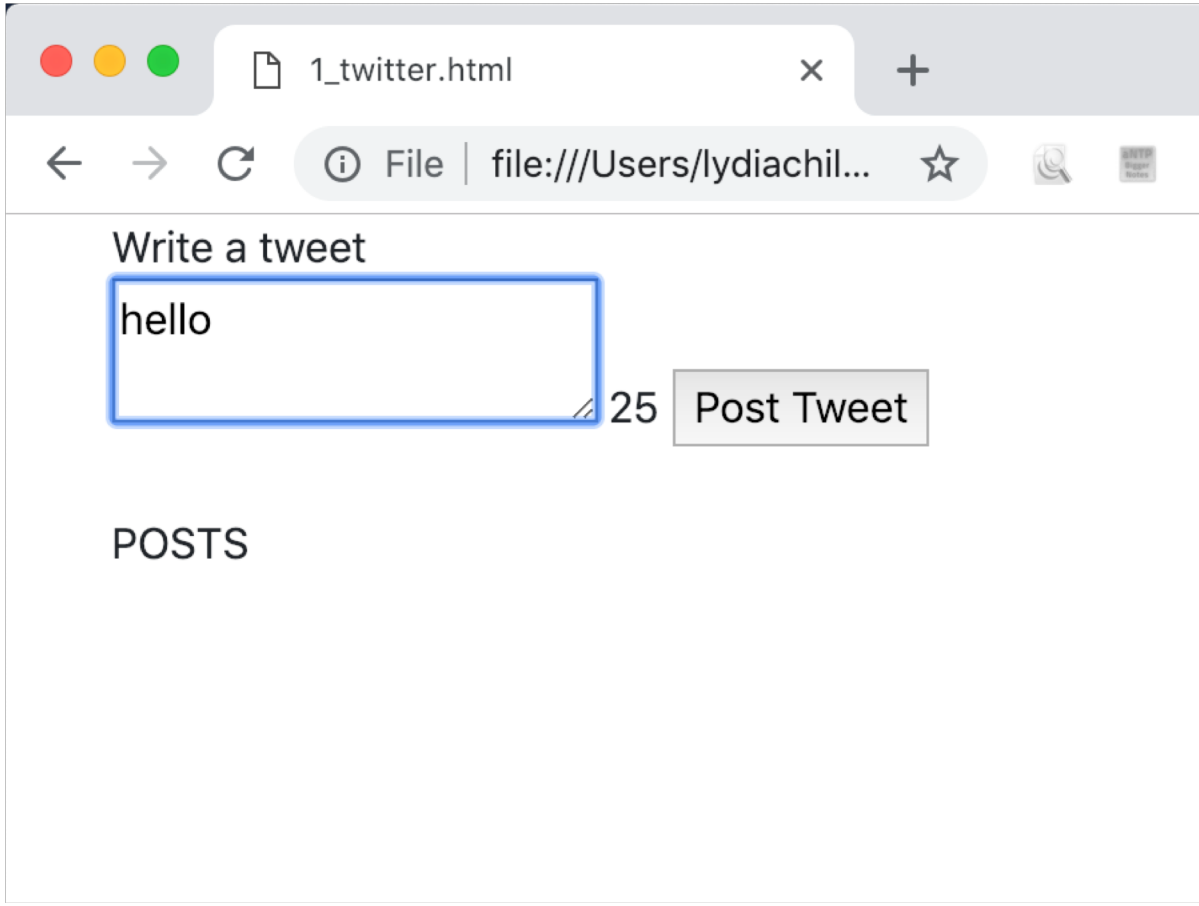
```
96
97          $("#write_tweet").keyup(function(){
98            var tweet_text = $(this).val()
99            console.log("tweet_text: "+tweet_text)
100
101
102         })
103
```



Now what?

Display the characters left

```
97    $("#write_tweet").keyup(function(){
98      var tweet_text = $(this).val()
99      console.log("tweet_text: "+tweet_text)
100
101     var tweet_length = tweet_text.length
102     var chars_left = max_tweet_length-tweet_length
103
104     $("#tweet_length_feedback").html(chars_left)
105
106   })
```



Now what?

Turn negative characters red. Like this?

```
96
97          $("#write_tweet").keyup(function(){
98            var tweet_text = $(this).val()
99            console.log("tweet_text: "+tweet_text)
100
101           var tweet_length = tweet_text.length
102           var chars_left = max_tweet_length-tweet_length
103
104
105           if(chars_left<0){
106
107             $("#tweet_length_feedback").css("color", "red").css("font-weight", "bold")
108           }else{
109             // ???
110
111           }
112
113           $("#tweet_length_feedback").html(chars_left)
114
115         })
116
```

Now what?

Add a class

```
97        $("#write_tweet").keyup(function(){
98          var tweet_text = $(this).val()
99          console.log("tweet_text: "+tweet_text)
100
101         var tweet_length = tweet_text.length
102         var chars_left = max_tweet_length-tweet_length
103
104
105         if(chars_left<0){
106           //turn characters red
107           // NO
108           //$("#tweet_length_feedback").css("color", "red").css("font-weight", "bold")
109
110           // YES
111           $("#tweet_length_feedback").addClass("length_warning")
112
113         }else{
114           // ???
115
116         }
117
118         $("#tweet_length_feedback").html(chars_left)
119
120       })
121
```

```
.length_warning{
  color: red;
  font-weight: bold;
}
```

Now what?

Next - remove the class. When?

```
96
97          $("#write_tweet").keyup(function(){
98            var tweet_text = $(this).val()
99            console.log("tweet_text: "+tweet_text)
100
101           var tweet_length = tweet_text.length
102           var chars_left = max_tweet_length-tweet_length
103
104
105           if(chars_left<0){
106             //turn characters red
107             // NO
108             //$("#tweet_length_feedback").css("color", "red").css("font-weight", "bold")
109
110             // YES
111             $("#tweet_length_feedback").addClass("length_warning")
112
113           }else{
114             $("#tweet_length_feedback").removeClass("length_warning")
115
116           }
117
118           $("#tweet_length_feedback").html(chars_left)
119
120         })
121
```

What next?

Write a tweet

Call me Ishmael. Some years ago—never mind how long precisely—having

-37 Post Tweet

POSTS

chilton	Most recent post
chilton	More ambitious third post
chilton	Second post
chilton	First post

DO THIS

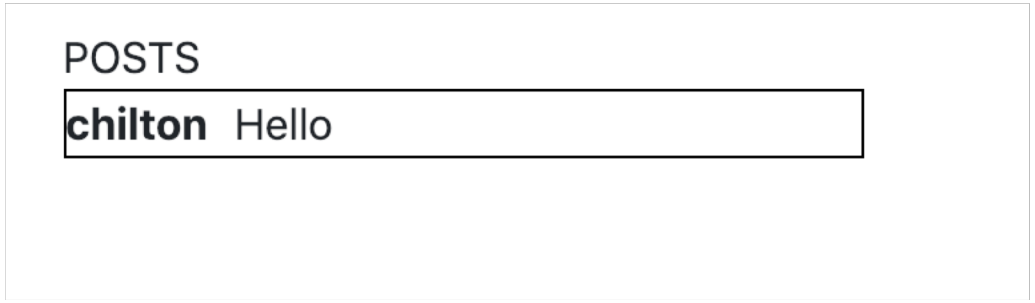
How do we start?

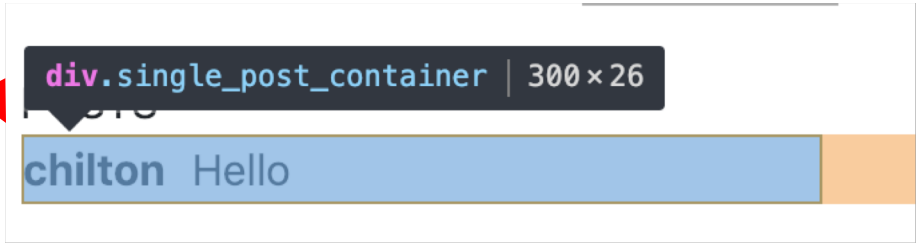
```
34      <script>
35        var username = "chilton"
36        var max_tweet_length = 30
37
38        $(document).ready(function(){
39          $("#post_tweet").click(function(){
40            var tweet_text = $("#write_tweet").val()
41            create_new_post(username, tweet_text)
42          })
43
```

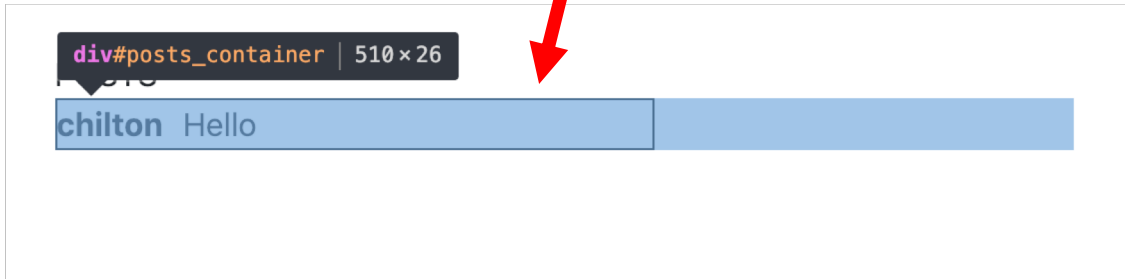
Add a click event listener (handler) to the post_tweet button

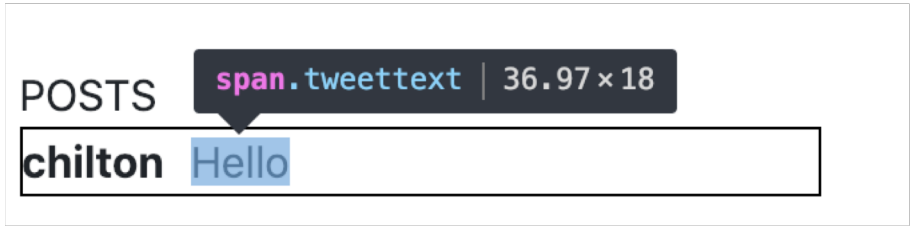
What does create_post do?

```
96
97      var create_new_post = function(username, tweet_text){
98        var newDiv = $("<div class='single_post_container'>")
99        var nameSpan = $("<span class='post_username'>")
100       nameSpan.html(username)
101
102       var tweetDiv = $("<span class='tweetText'>")
103       tweetDiv.html(tweet_text)
104
105       newDiv.append(nameSpan)
106       newDiv.append(tweetDiv)
107       $("#posts_container").prepend(newDiv)
108     }
```









What else does it need to do?

```
<script>
  var username = "chilton"
  var max_tweet_length = 30

  $(document).ready(function(){
    $("#post_tweet").click(function(){
      var tweet_text = $("#write_tweet").val()
      create_new_post(username, tweet_text)
    })

    // clear text
    $("#write_tweet").val("")
```

a. When the user presses the “post tweet” button the following things must happen:
   i. The post must appear with the poster’s username (not an image). The username can be hard coded in JavaScript (but not in HTML). The post text cannot be hard coded (obviously).
   ii. New posts must appear at the top of the list of posts, so that users see the latest tweets at the top of the list.
   iii. The text in the box where users write tweets must disappear.
   iv. The number that counts the number of characters remaining must return to the maximum number.
   v. The cursor must return to the box where the user writes tweets (so that they can immediately start writing another tweet!)

After the text clears, then what?

```
34      <script>
35        var username = "chilton"
36        var max_tweet_length = 30
37
38        $(document).ready(function(){
39          $("#post_tweet").click(function(){
40            var tweet_text = $("#write_tweet").val()
41            create_new_post(username, tweet_text)
42          })
43
```

```
// clear text
$("#write_tweet").val("")
```

```
//clear counter
$("#tweet_length_feedback").html(max_tweet_length)
$("#tweet_length_feedback").removeClass("length_warning")
```

a. When the user presses the “post tweet” button the following things must happen:
   i. The post must appear with the poster’s username (not an image). The username can be hard coded in JavaScript (but not in HTML). The post text cannot be hard coded (obviously).
   ii. New posts must appear at the top of the list of posts, so that users see the latest tweets at the top of the list.
   iii. The text in the box where users write tweets must disappear.
   iv. The number that counts the number of characters remaining must return to the maximum number.
   v. The cursor must return to the box where the user writes tweets (so that they can immediately start writing another tweet!)

After the tweet length = 30m then what?

```
34      <script>
35        var username = "chilton"
36        var max_tweet_length = 30
37
38        $(document).ready(function(){
39          $("#post_tweet").click(function(){
40            var tweet_text = $("#write_tweet").val()
41            create_new_post(username, tweet_text)
42          })
43
            // clear text
            $("#write_tweet").val("")

            //clear counter
            $("#tweet_length_feedback").html(max_tweet_length)
            $("#tweet_length_feedback").removeClass("length_warning")

            //return focus to textarea
            $("#write_tweet").focus()
```

a. When the user presses the “post tweet” button the following things must happen:
   i. The post must appear with the poster’s username (not an image). The username can be hard coded in JavaScript (but not in HTML). The post text cannot be hard coded (obviously).
   ii. New posts must appear at the top of the list of posts, so that users see the latest tweets at the top of the list.
   iii. The text in the box where users write tweets must disappear.
   iv. The number that counts the number of characters remaining must return to the maximum number.
   v. The cursor must return to the box where the user writes tweets (so that they can immediately start writing another tweet!)

There’s one more thing I do. Why?

```
34    <script>
35      var username = "chilton"
36      var max_tweet_length = 30
37
38      $(document).ready(function(){
39        $("#write_tweet").focus()
40        $("#tweet_length_feedback").html(max_tweet_length)
41
42
```


```
100   <div>Write a tweet</div>
101   <textarea id="write_tweet"></textarea>
102   <div id="tweet_length_feedback"></div>
103   <button id="post_tweet"> Post Tweet</button>
```

Direct Manipulation

No screens

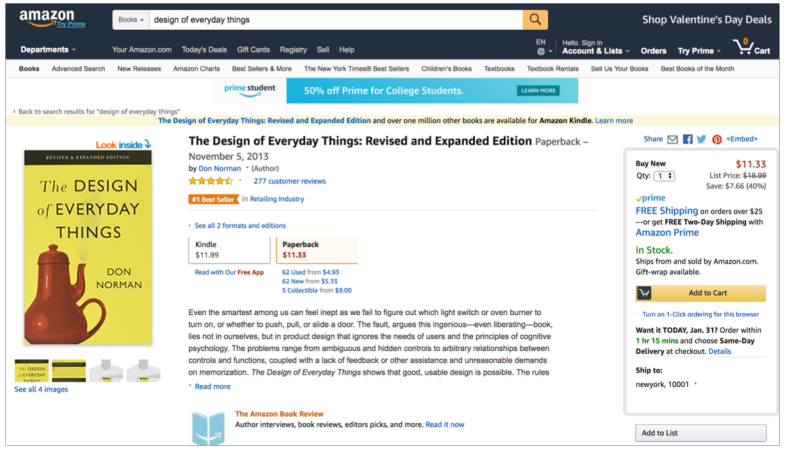
Prof. Lydia Chilton
COMS 4170
18 February 2019

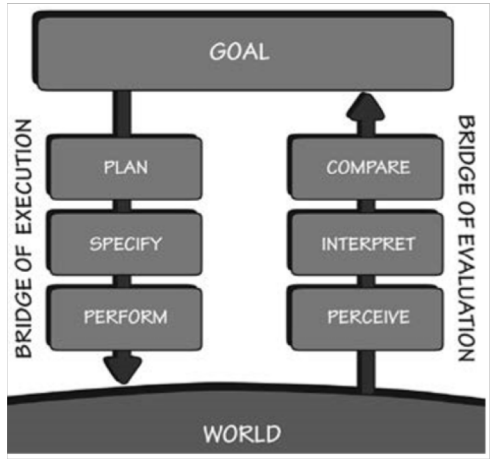
Say your name

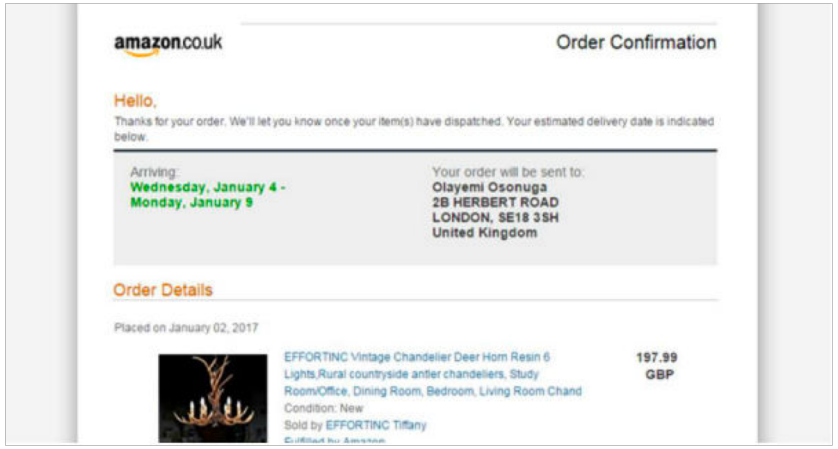
Goal 1

Build websites that suit the needs and abilities of users

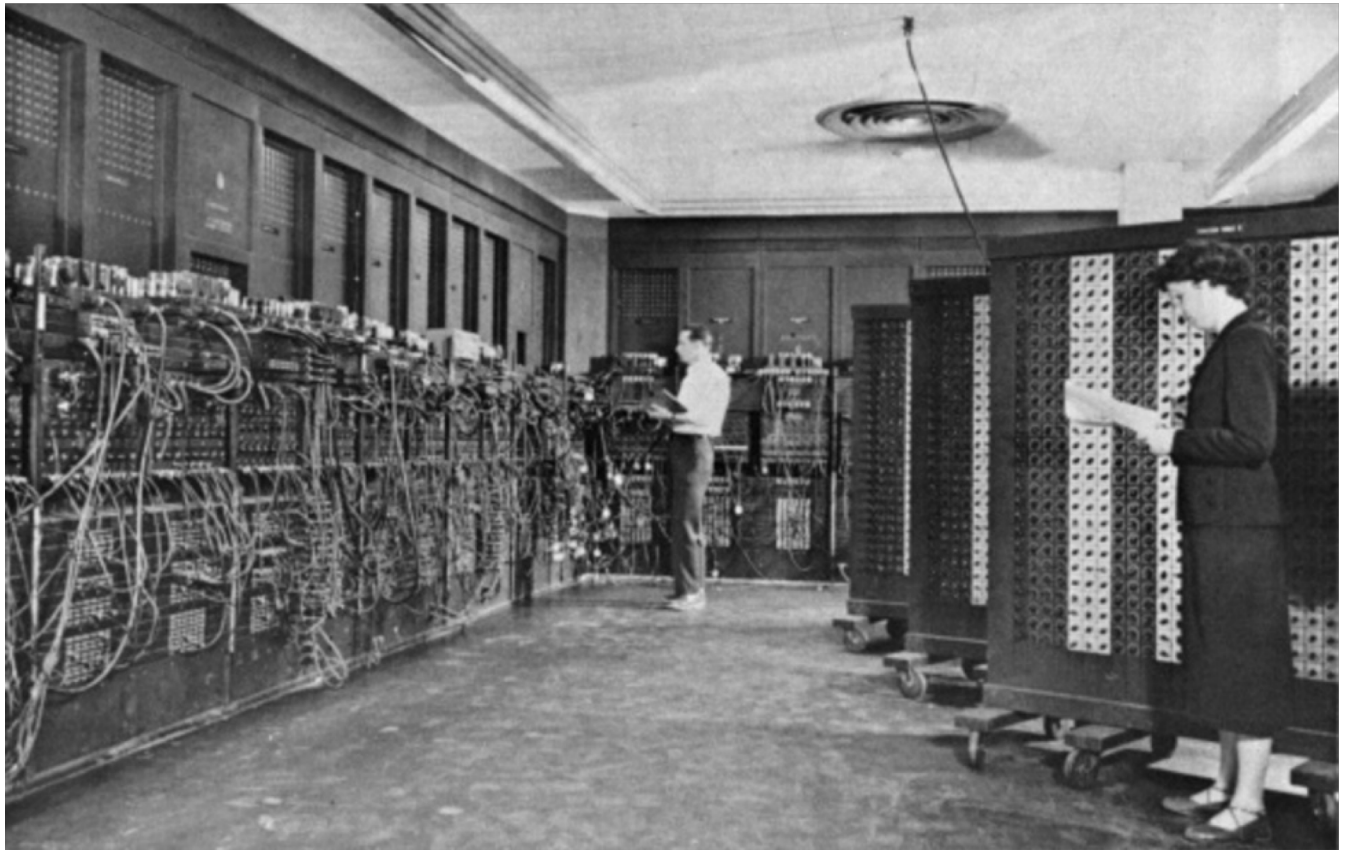
To accomplish a **goal**, users must
execute an operation and
evaluate the result





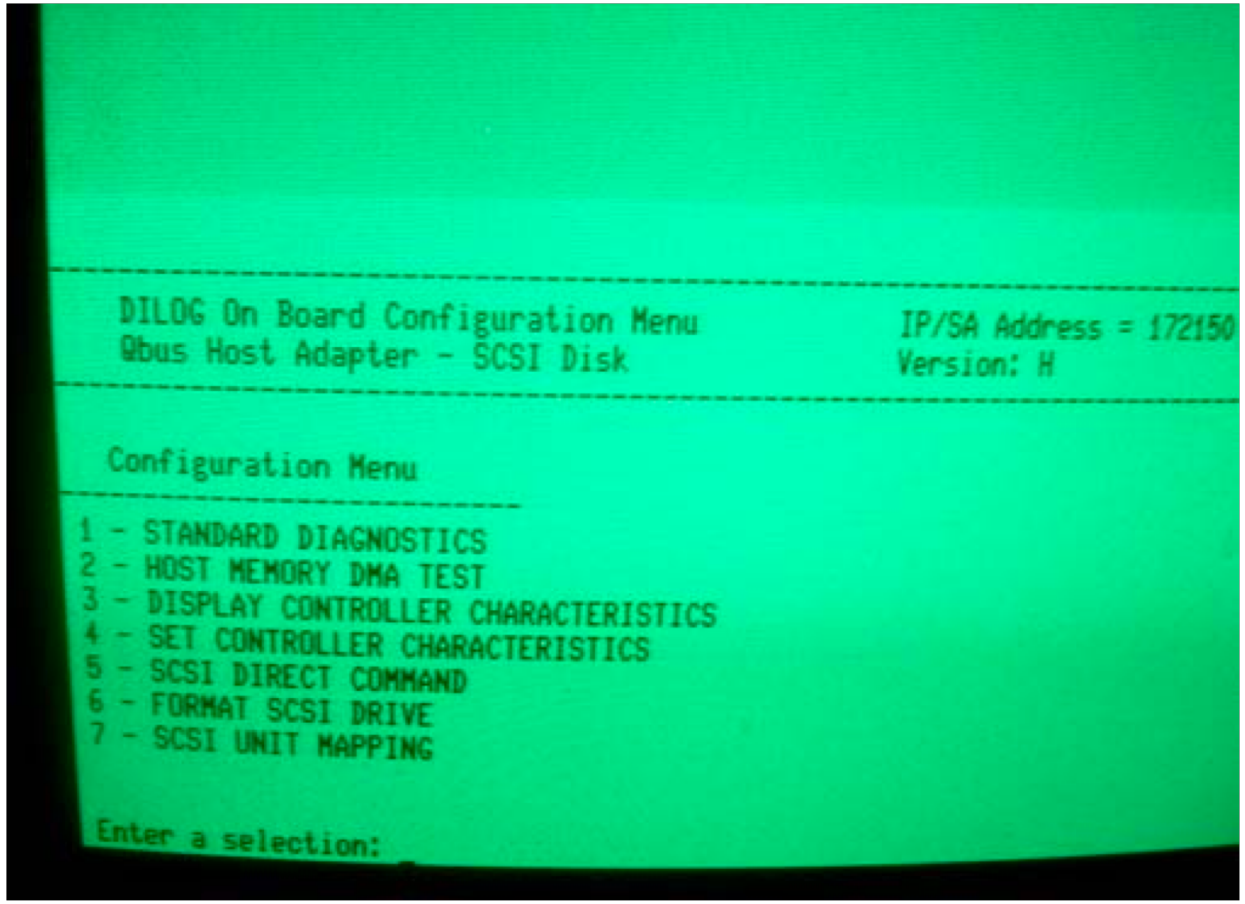


Computers: Tools for Calculation



```
ASSUME CS:CODE,DS:DATA
DATA SEGMENT
LIST DW 2579H,0A500H,0C009H,0159H,0B900H
COUNT EQU 05H
DATA ENDS
CODE SEGMENT
START:      XOR BX,BX
            XOR DX,DX
            MOV AX,DATA
            MOV DS,AX
            MOV CL,COUNT
            MOV SI,OFFSET LIST
AGAIN:      MOV AX,[SI]
            SHL AX,01
            JC NEG
            INC BX
            JMP NEXT
NEG:        INC DX
NEXT:       ADD SI,02
            DEC CL
            JNZ AGAIN
            MOV AH,4CH
            INT 21H
            CODE ENDS
            END START
```

Interaction: through Textual Commands

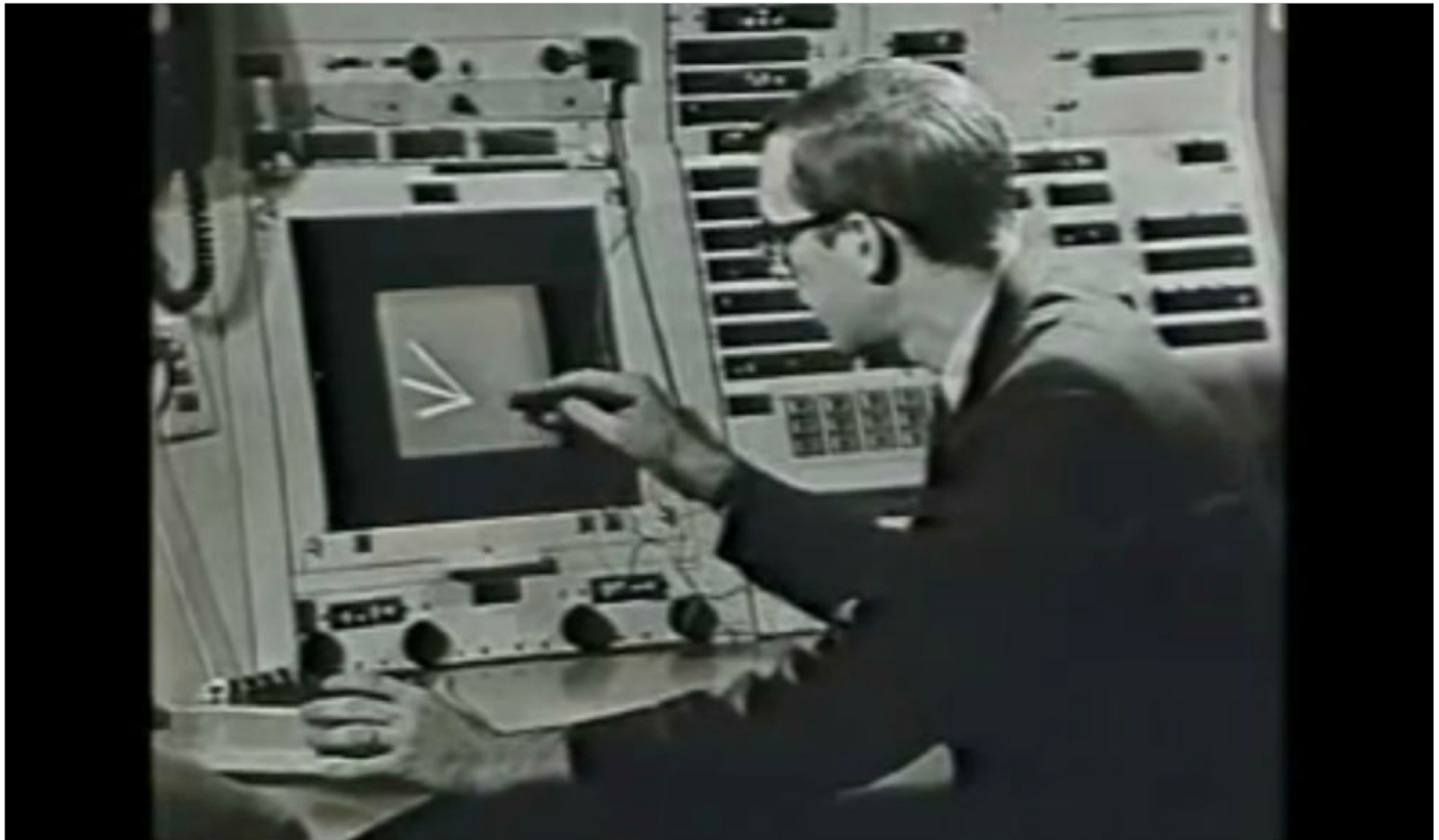


1963: First Graphical User Interface (GUI) Ivan Sutherland's CAD software, Sketchpad

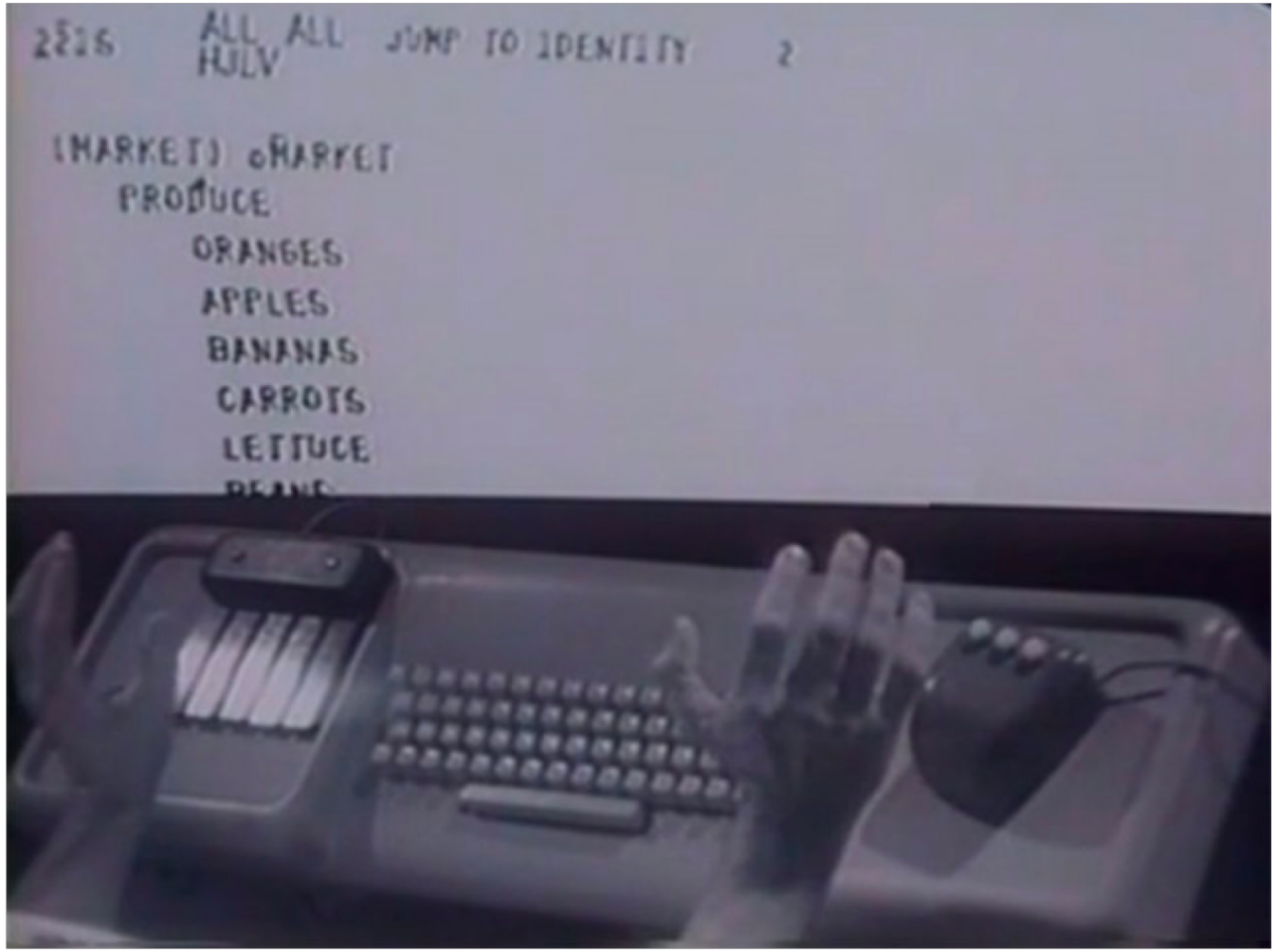
The terminal way

```
drawLine((0,0), (1,1))
```

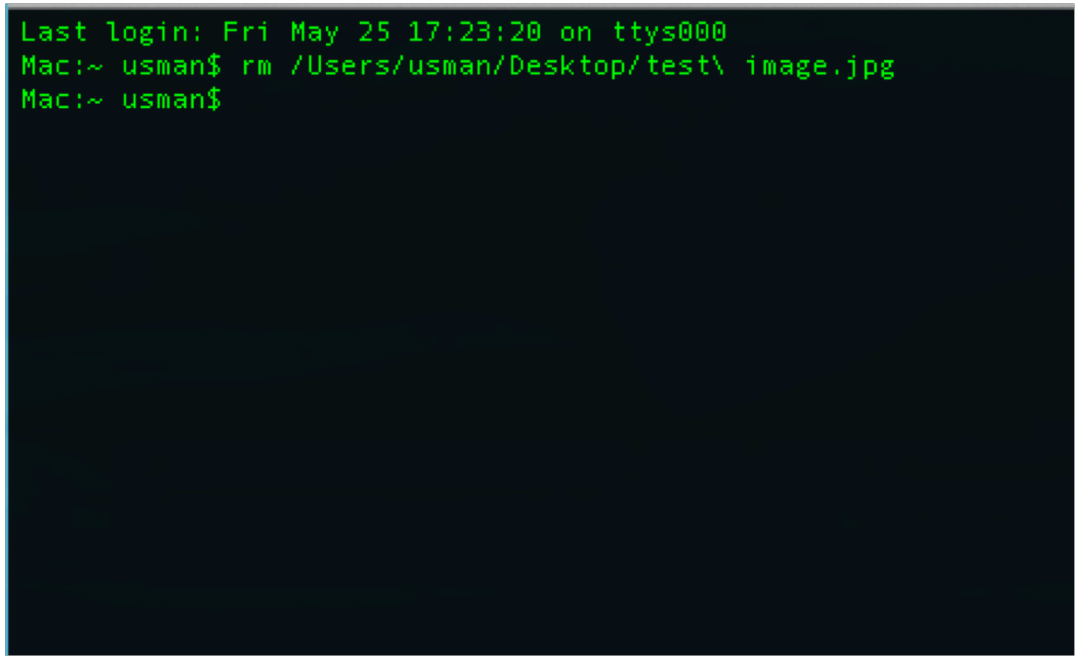
The GUI way



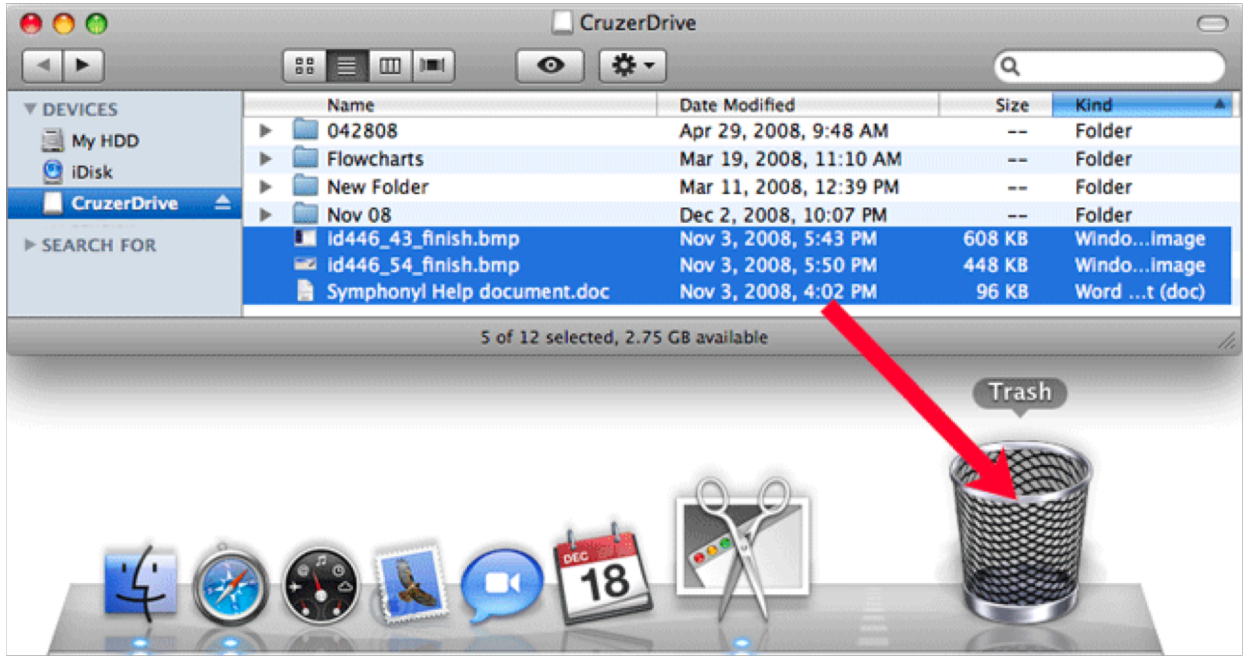
1968: Interaction devices for computer use. Douglas Engelbart's mouse



Then:
Textual commands



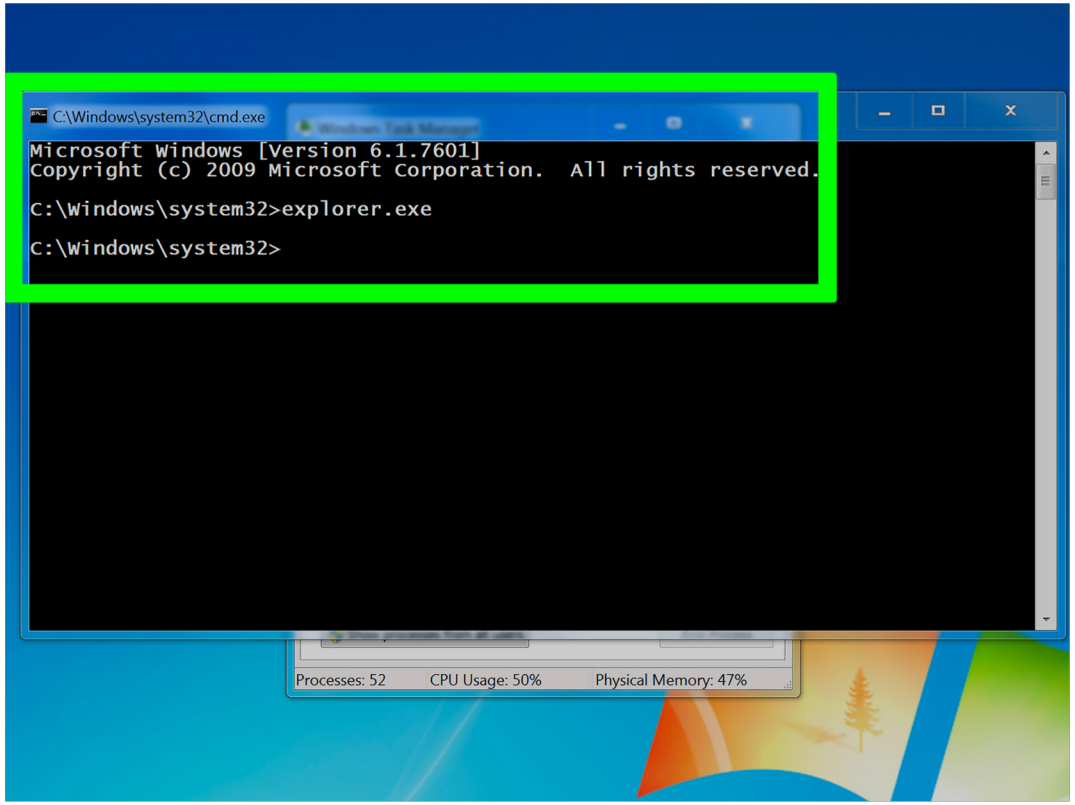
Now:
Graphical User Interfaces

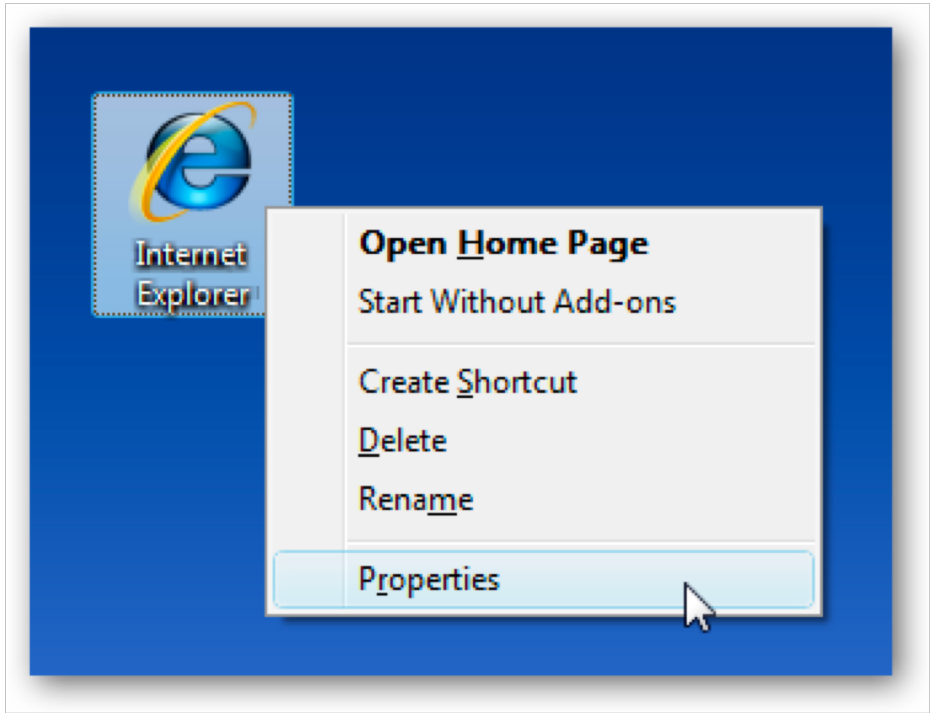


What usability heuristic is this?

6. Recognition rather than recall

Minimize the user's memory load by making objects, actions, and options visible. The user should not have to remember information from one part of the dialogue to another.



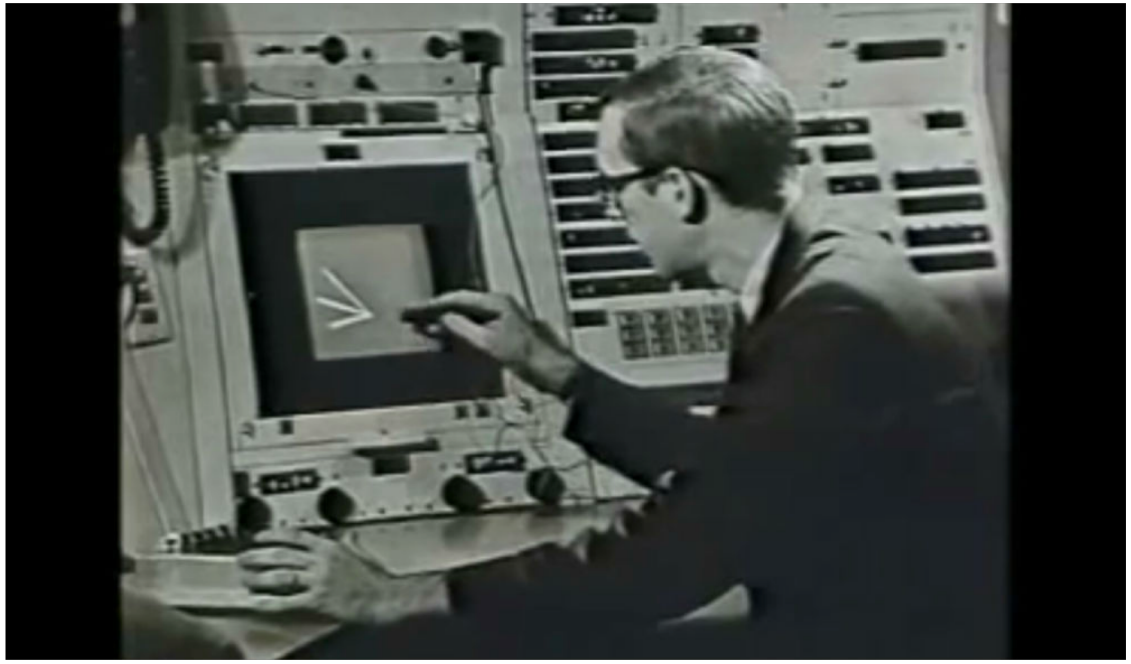


Direct Manipulation Properties

1. **Objects** are represented visually

2. **Actions** are rapid, incremental and reversible

3. User interacts **directly with object representations**

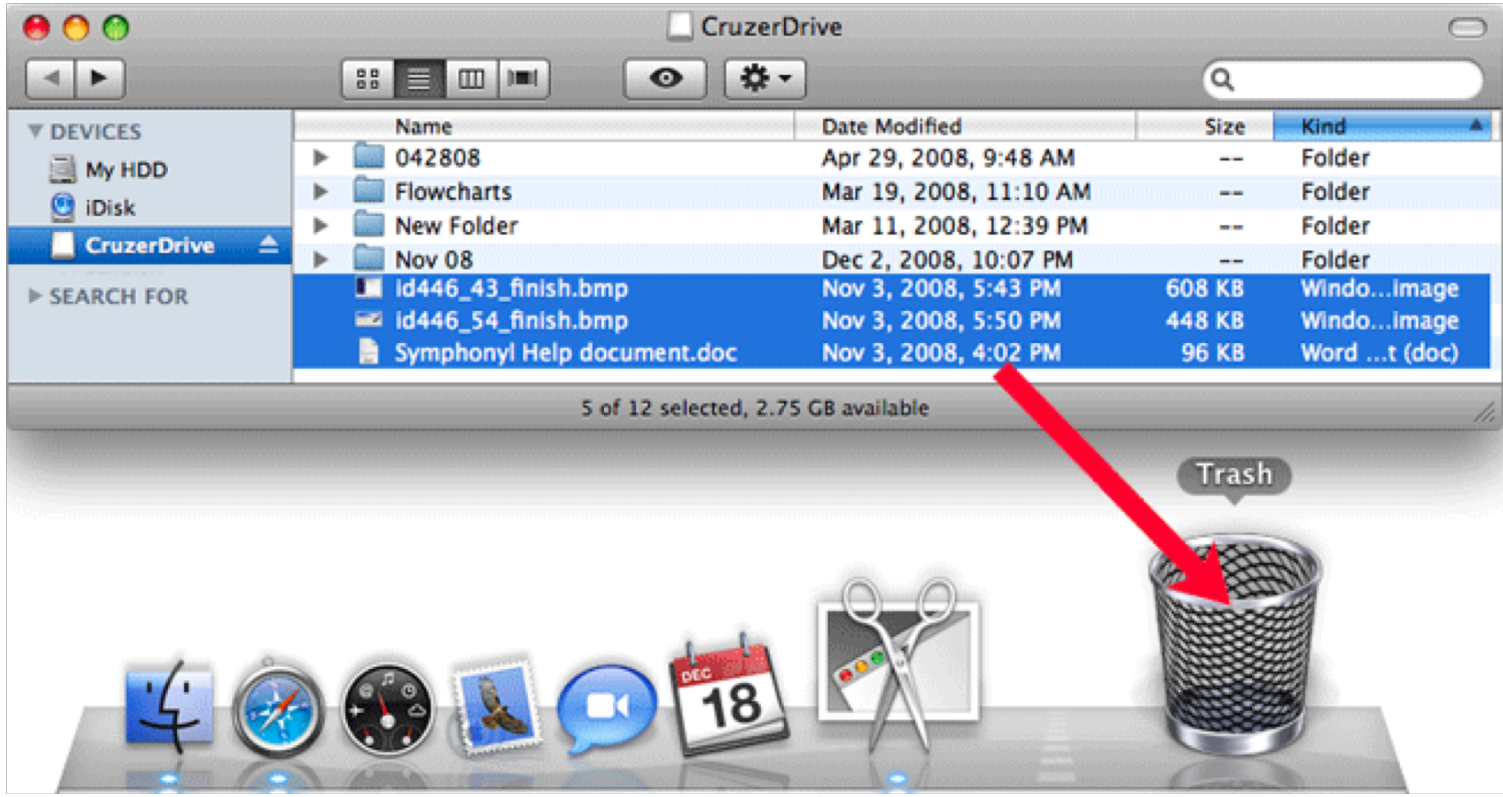


Goal: Move file to trash.

1. What **Objects** are represented visually?

2. What **Actions** are rapid, incremental and reversible?

3. How do user interacts **directly with object representations**

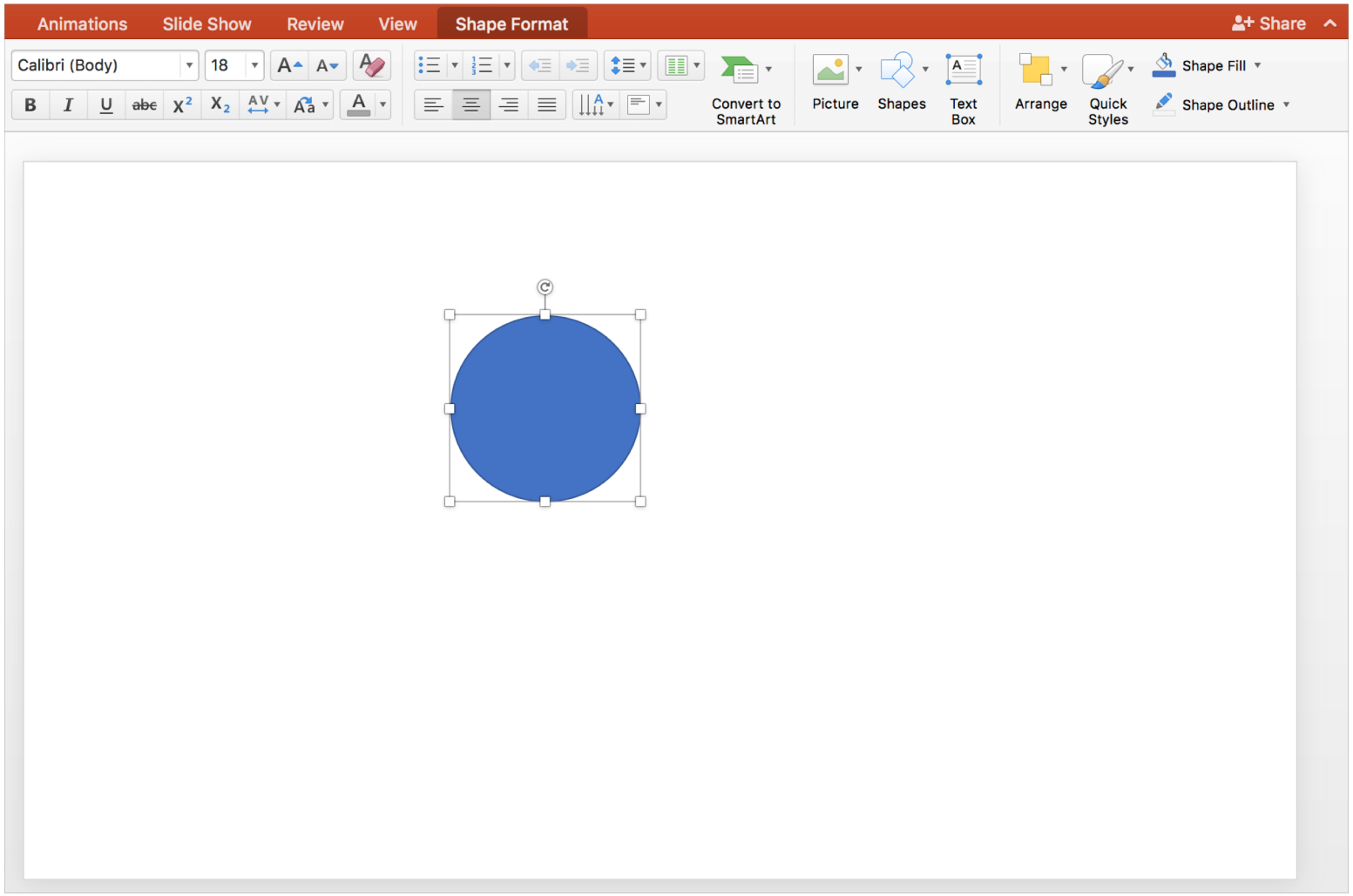


Goal: Make circle bigger.

1. What **Objects** are represented visually?

2. What **Actions** are rapid, incremental and reversible?

3. How do user interacts **directly with object representations**

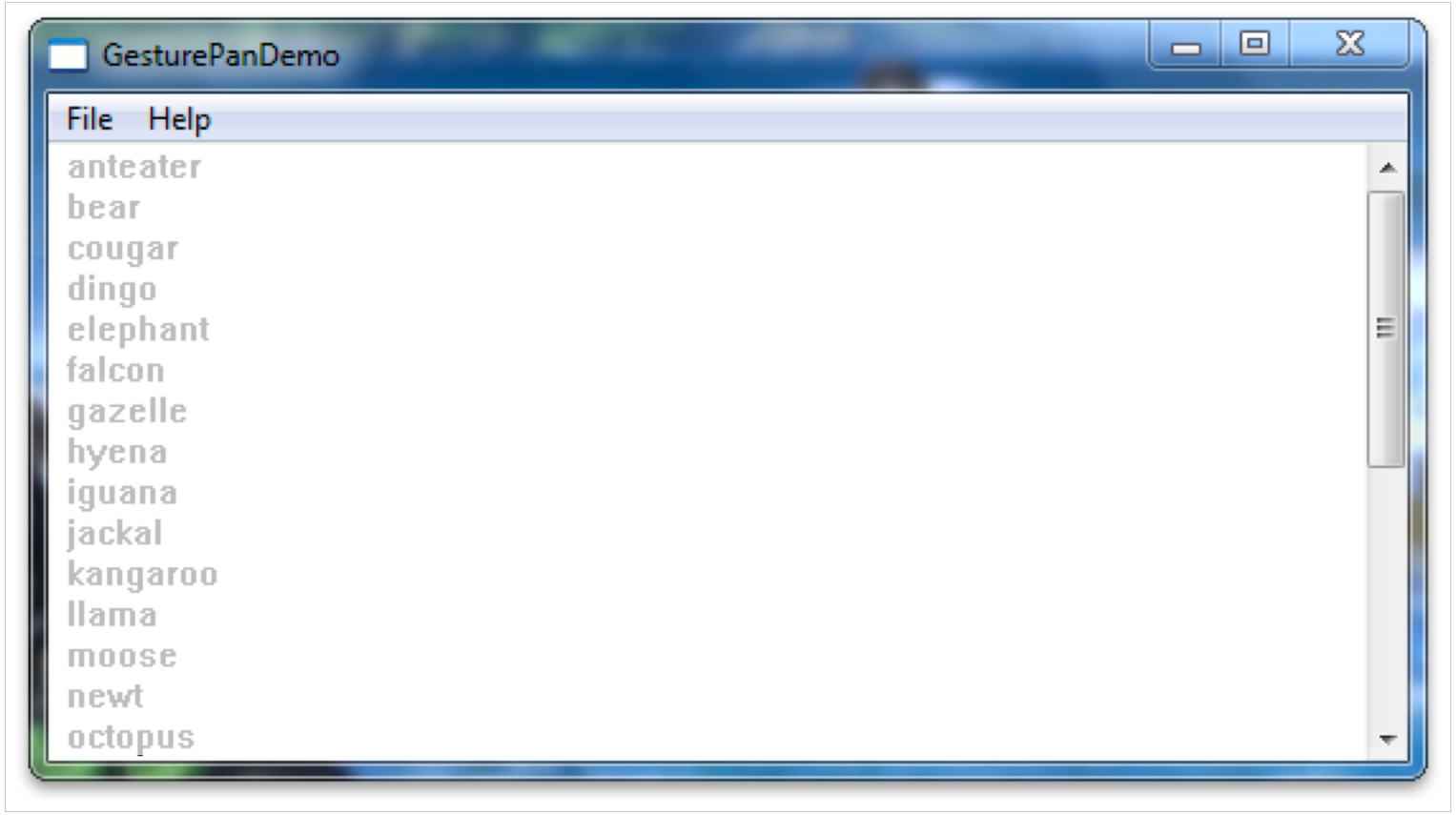


Goal: See stuff at the bottom on the list.

1. What **Objects** are represented visually?

2. What **Actions** are rapid, incremental and reversible?

3. How do user interacts **directly with object representations**

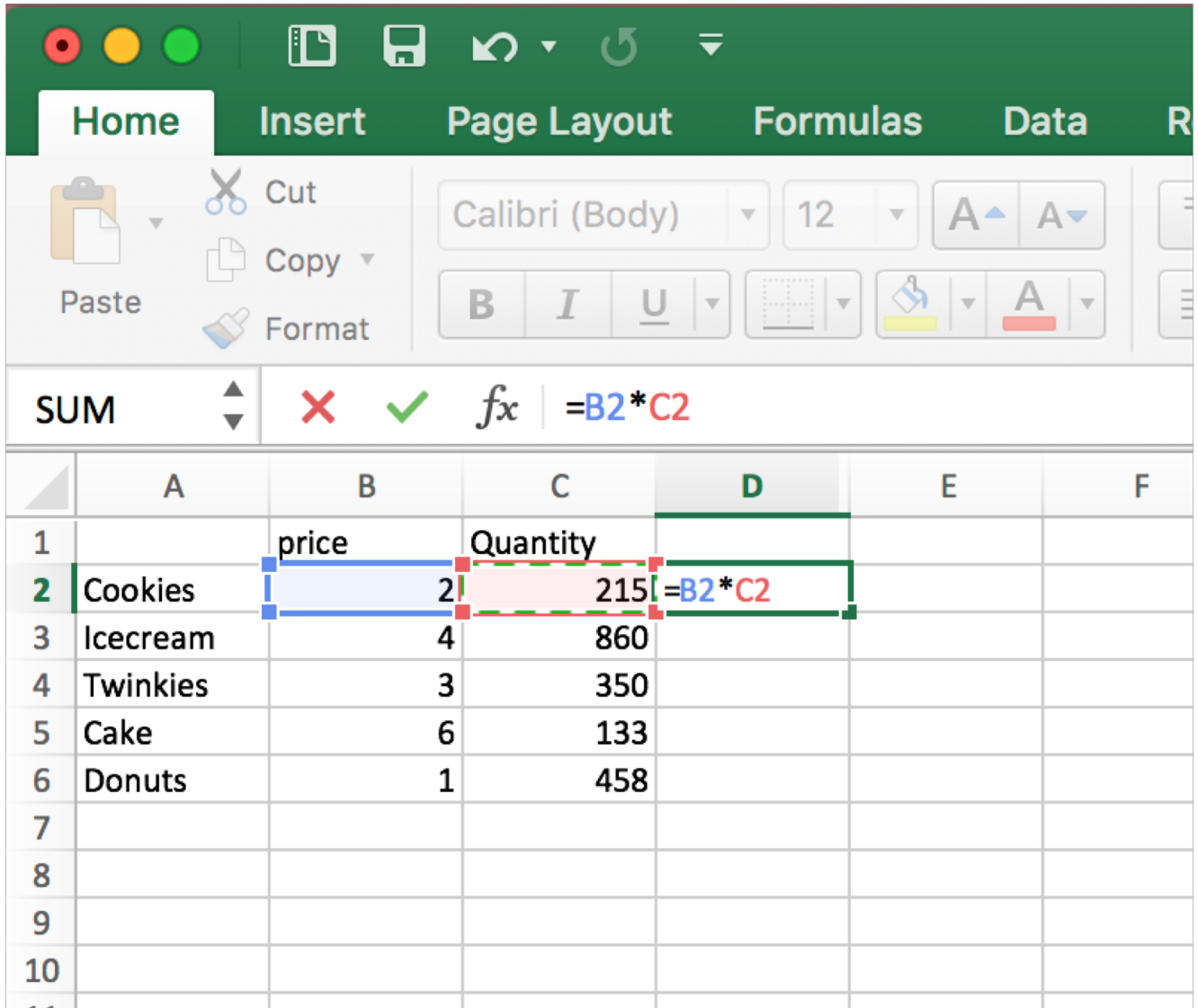


Goal: Multiply numbers in a Spreadsheet.

1. What **Objects** are represented visually?

2. What **Actions** are rapid, incremental and reversible?

3. How do user interacts **directly with object representations**



Direct Manipulation Properties

1. **Objects** are represented visually

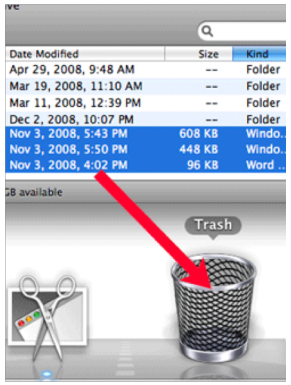
2. **Actions** are rapid, incremental and reversible

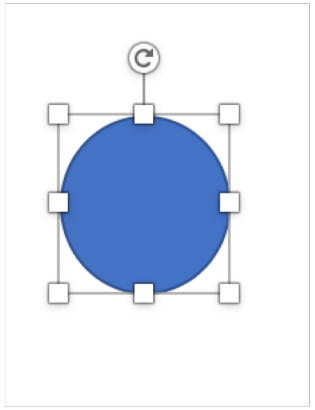
Move file to…

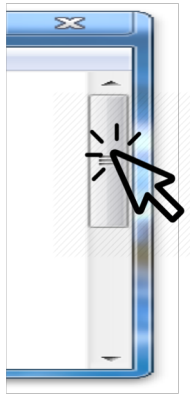
Resize

Move viewport

3. User interacts **directly with object representations**







Goal: Multiply two numbers on the phone calculator.

Is this direct manipulation? No.

1. Are **Objects** are represented visually? Yes.

2. Are **Actions** are rapid, incremental and reversible? Sorta.

3. Do user interacts **directly with object representations?** No.

2. Match between system and the real world

The system should speak the users' language, with words, phrases and concepts familiar to the user, rather than system-oriented terms.

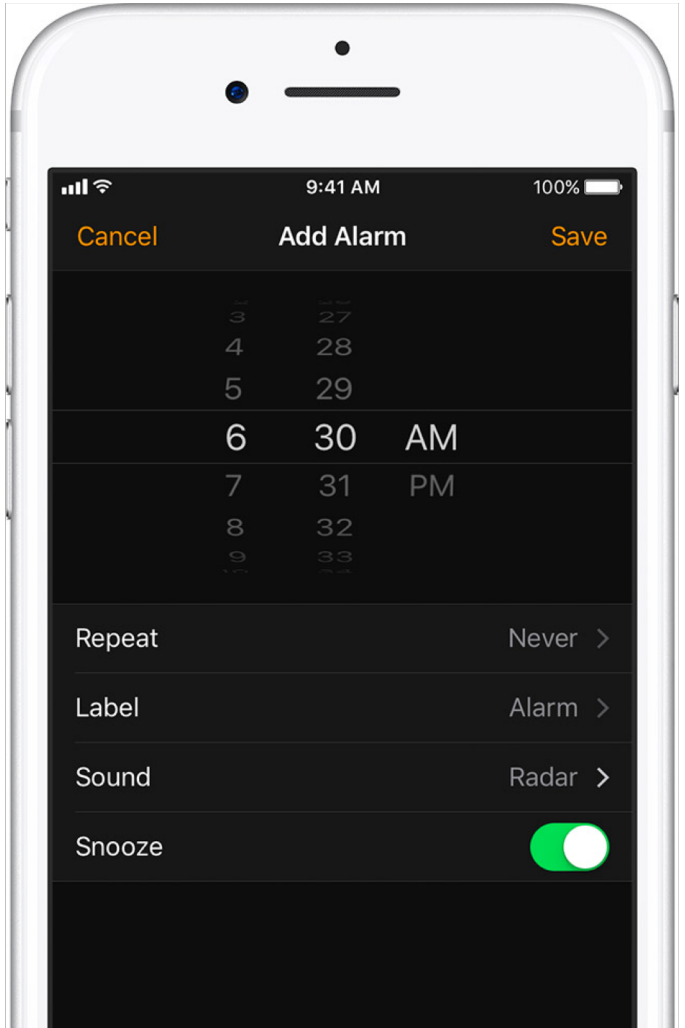
Goal: Set an alarm clock with number selection wheel.

Is this direct manipulation? Yes.

1. Are **Objects** are represented visually? Yes.

2. Are **Actions** are rapid, incremental and reversible? Yes.

3. Do user interacts **directly with object representations?** Yes.



Goal: Set an alarm clock with Siri.

Is this direct manipulation? No. But it's awesome!

1. Are **Objects** are represented visually? Yes.

2. Are **Actions** are rapid, incremental and reversible? No.

3. Do user interacts **directly with object representations?** No.

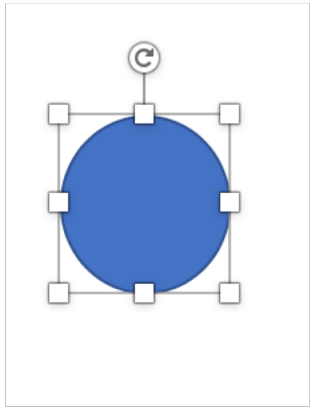
Direct manipulation requires directly interacting with object representation.

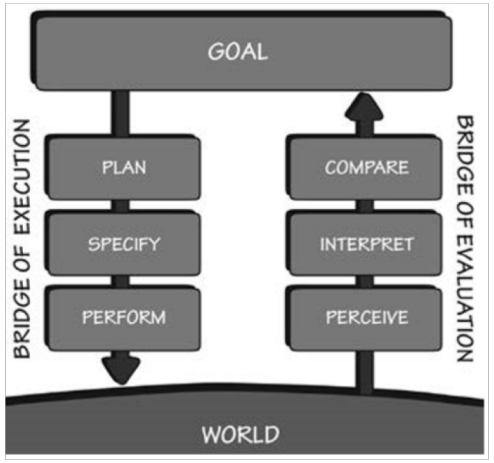
Not direct manipulation

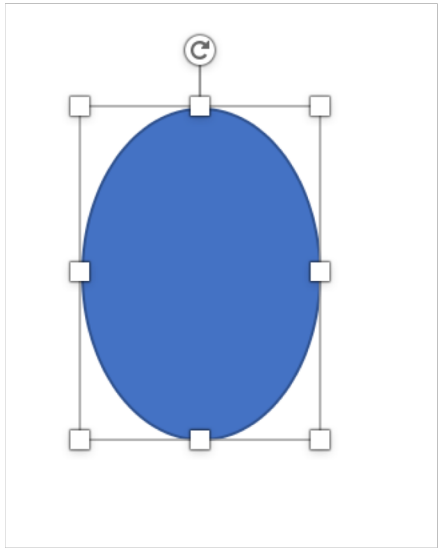
Direct manipulation

Not direct manipulation

Why can Direct Manipulation be good?



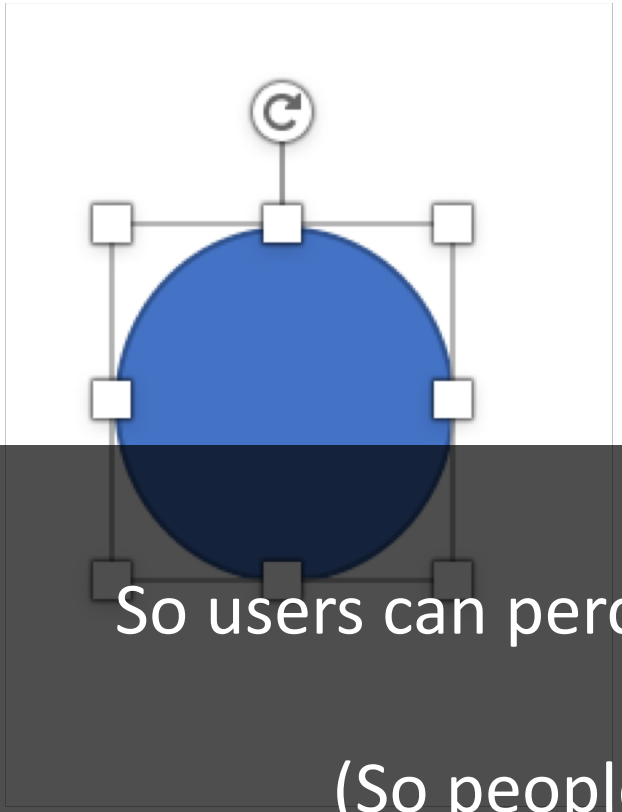


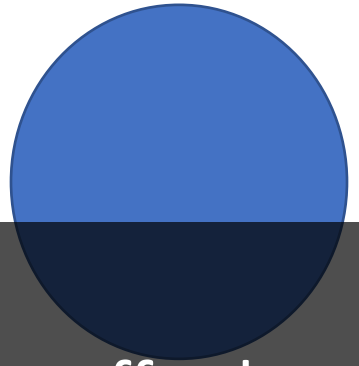


There are visible **actions** the user can **execute**

There is visible **feedback** the user can **evaluate**

Why is it important for the circle to have the resize handles?





So users can perceive the correct affordances.

(So people can see what to do.)

Affordances

Affordance: What can do you with this?

Perceived Affordance	Sitting
Signifier	Flat part at knee-height Back panel for support Sturdy wood Butt indentation
Feedback	Test sitting on it.
Affordance	Sitting

Affordance: What can do you with this?

Perceived Affordance	Sitting
Signifier	Flat part at knee-height Back panel for support Possibly sturdy cans?
Feedback	Test sitting on it.
Affordance	NOT sitting. Looking awesome.

Affordance: What can do you with this?

Perceived Affordance	Pull
Signifier	A handle you can grasp and yank
Feedback	Yanking it
Affordance	NOT pull push

Affordance: What can do you with this?

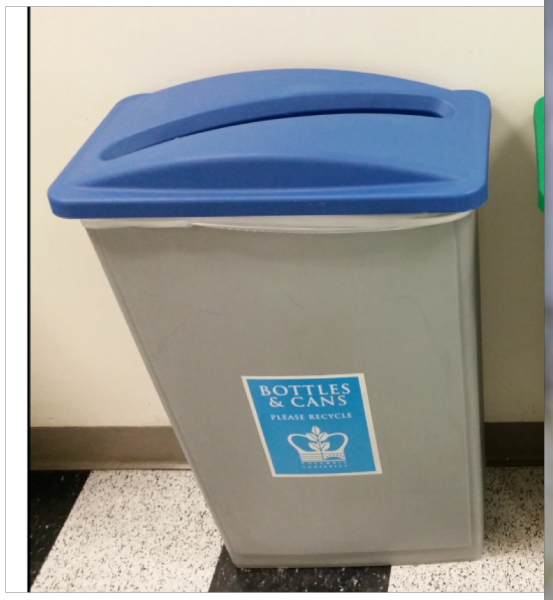
Perceived Affordance	Push
Signifier	A handle you can lean on and push
Feedback	Push, depress handle
Affordance	Push

Affordance: What should do you with this?

Perceived Affordance	Put paper in it
Signifier	Paper sized hole
Feedback	None.
Affordance	Bottles and cans

Design direct manipulation interfaces with good *perceived* affordances.

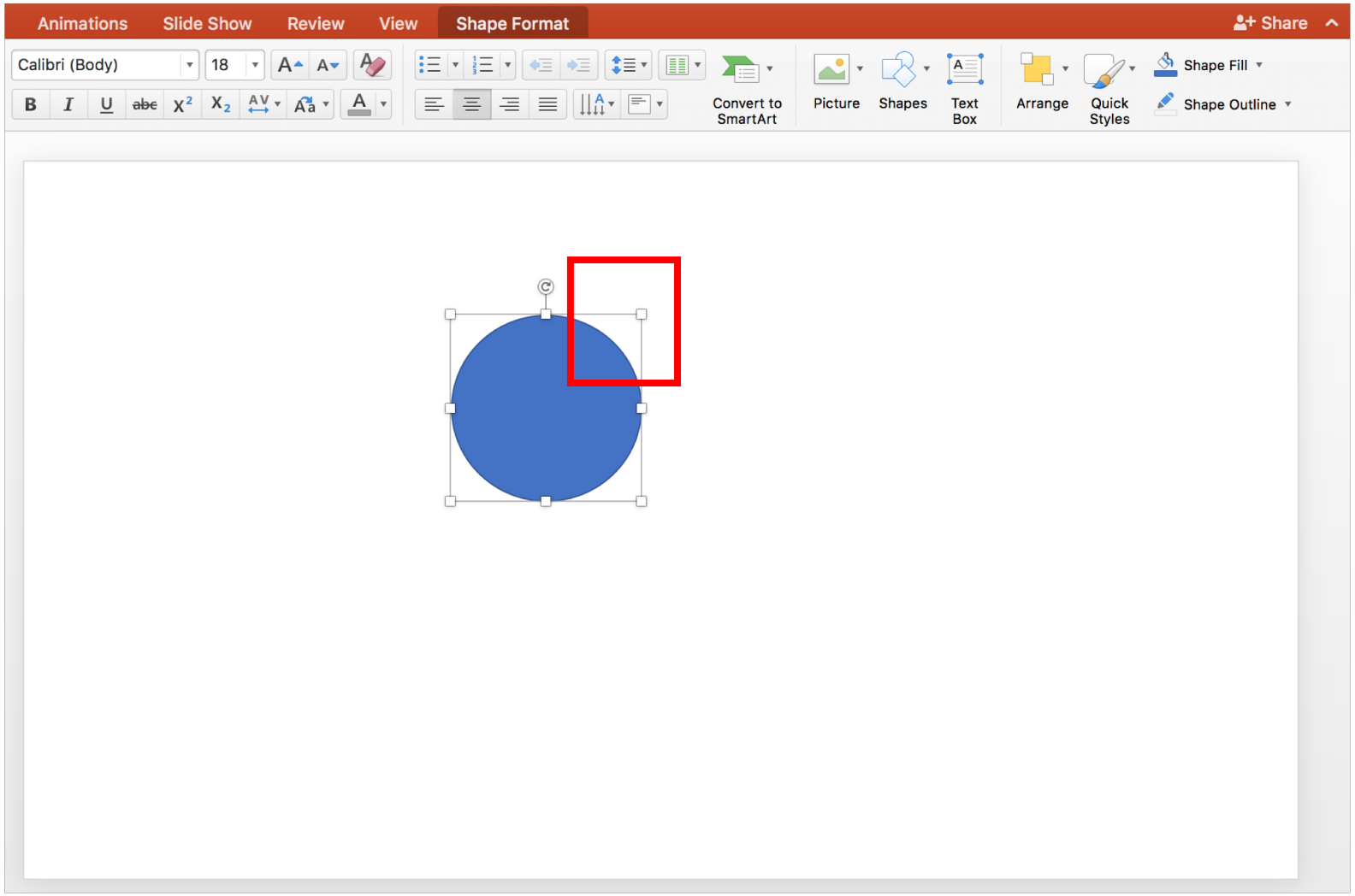
Bad signifiers / wrong perceived affordances



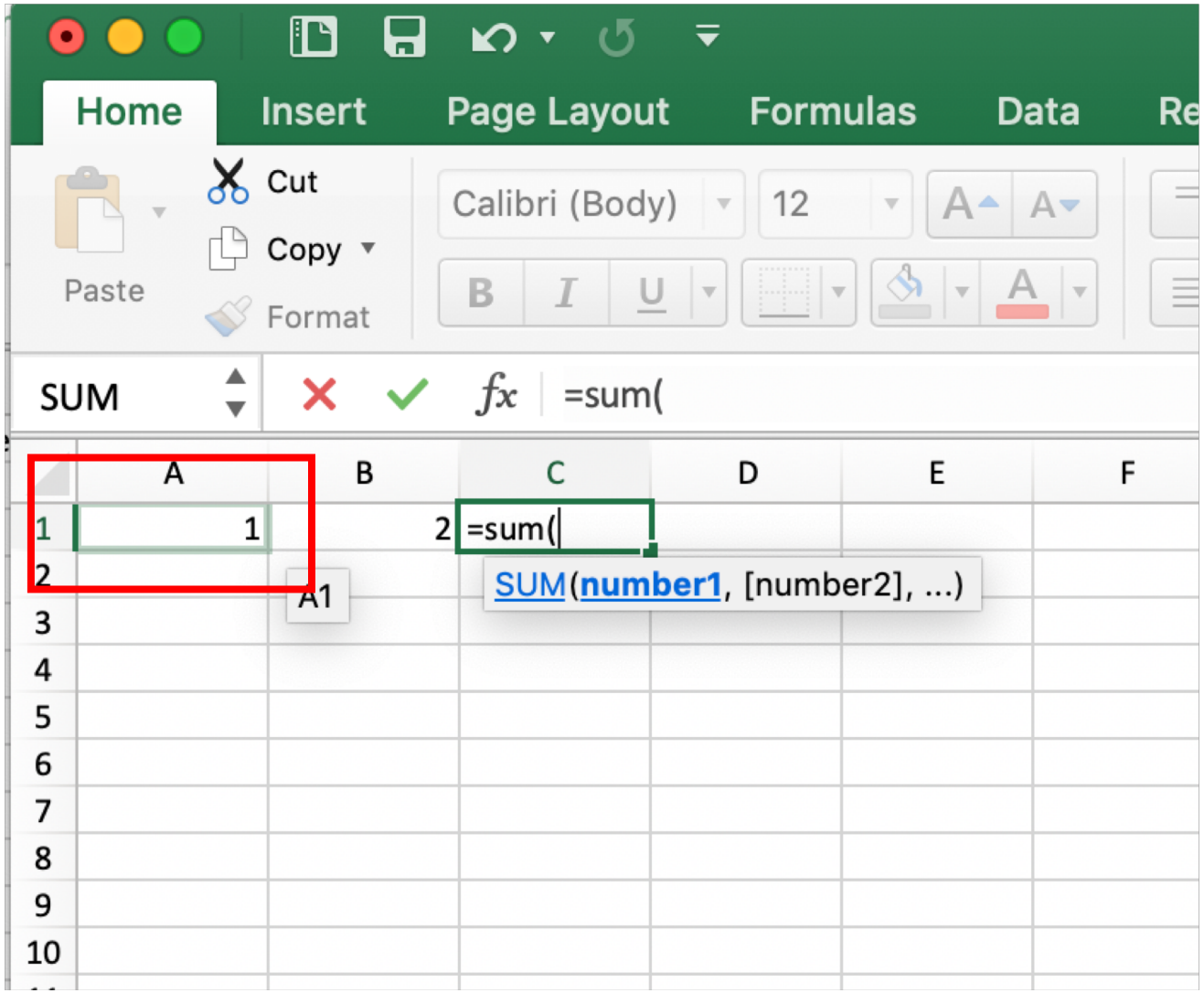
Good signifiers / correct perceived affordances

What signifiers do these UIs use to signal affordances?

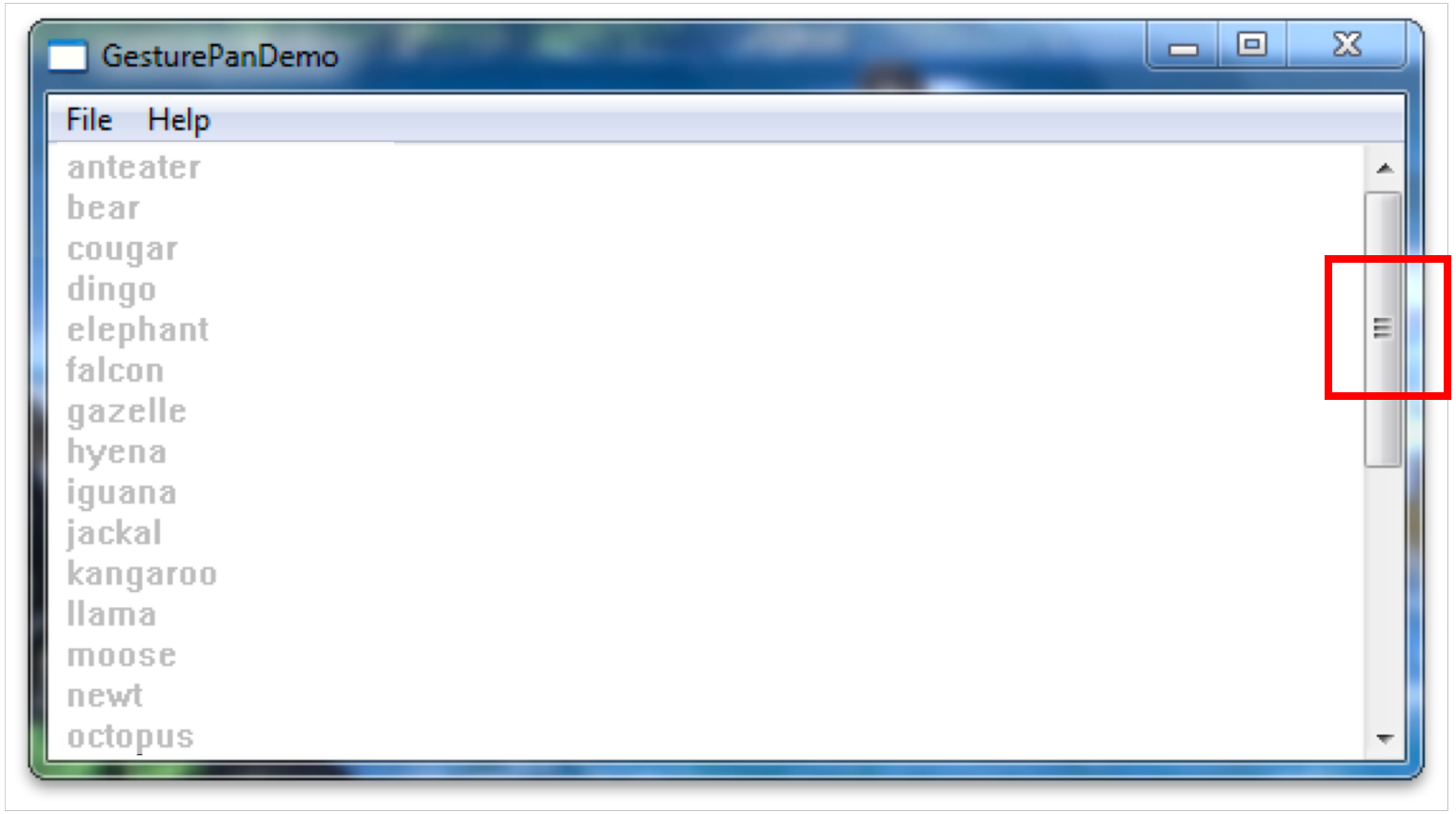
What are the signifiers of affordances?



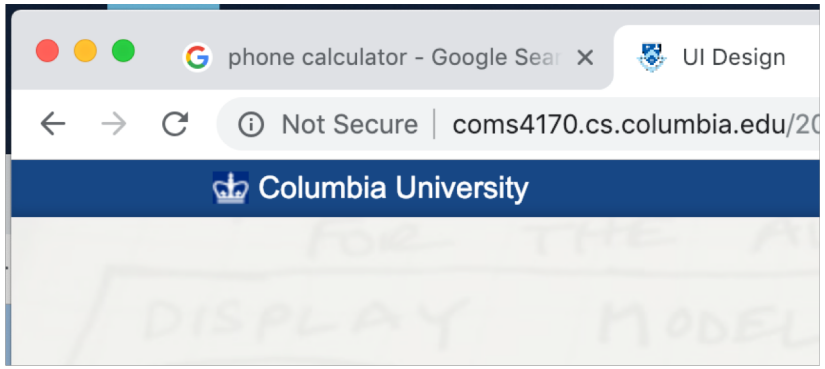
What are the signifiers of affordances?

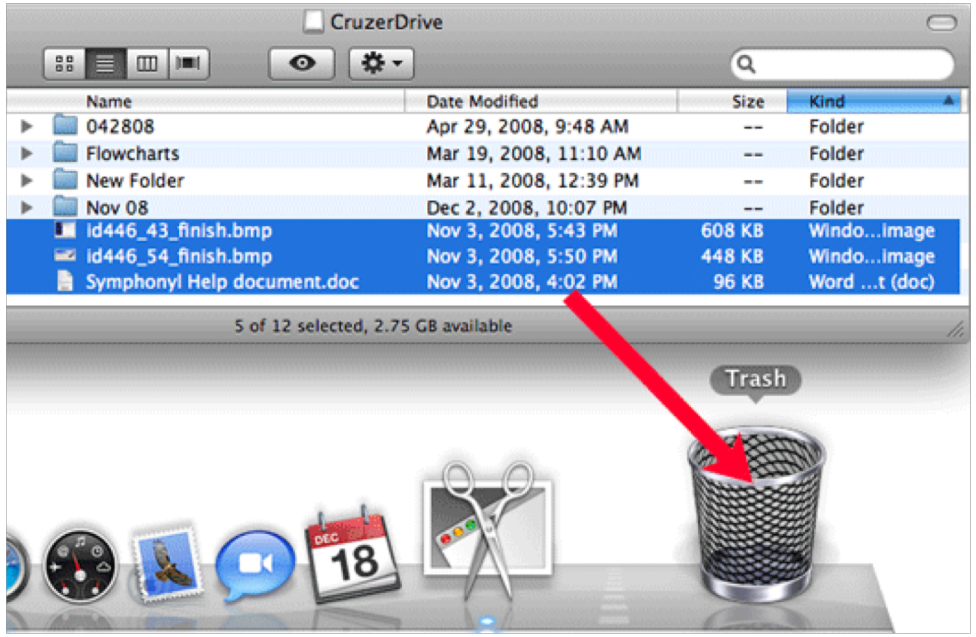


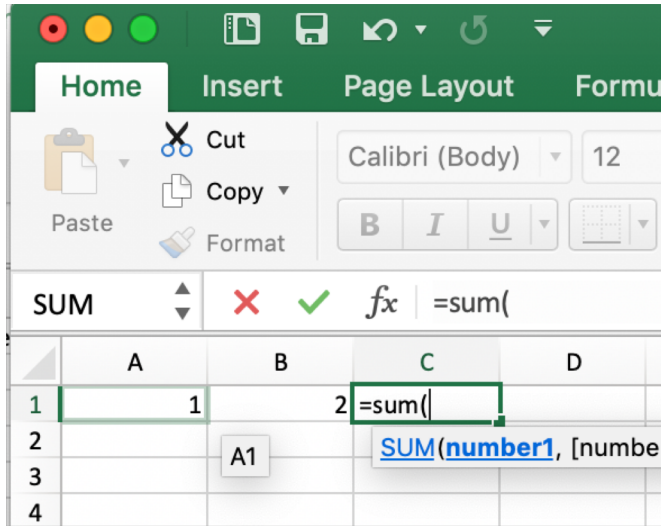
What are the signifiers of affordances?



What are the signifiers of affordances?



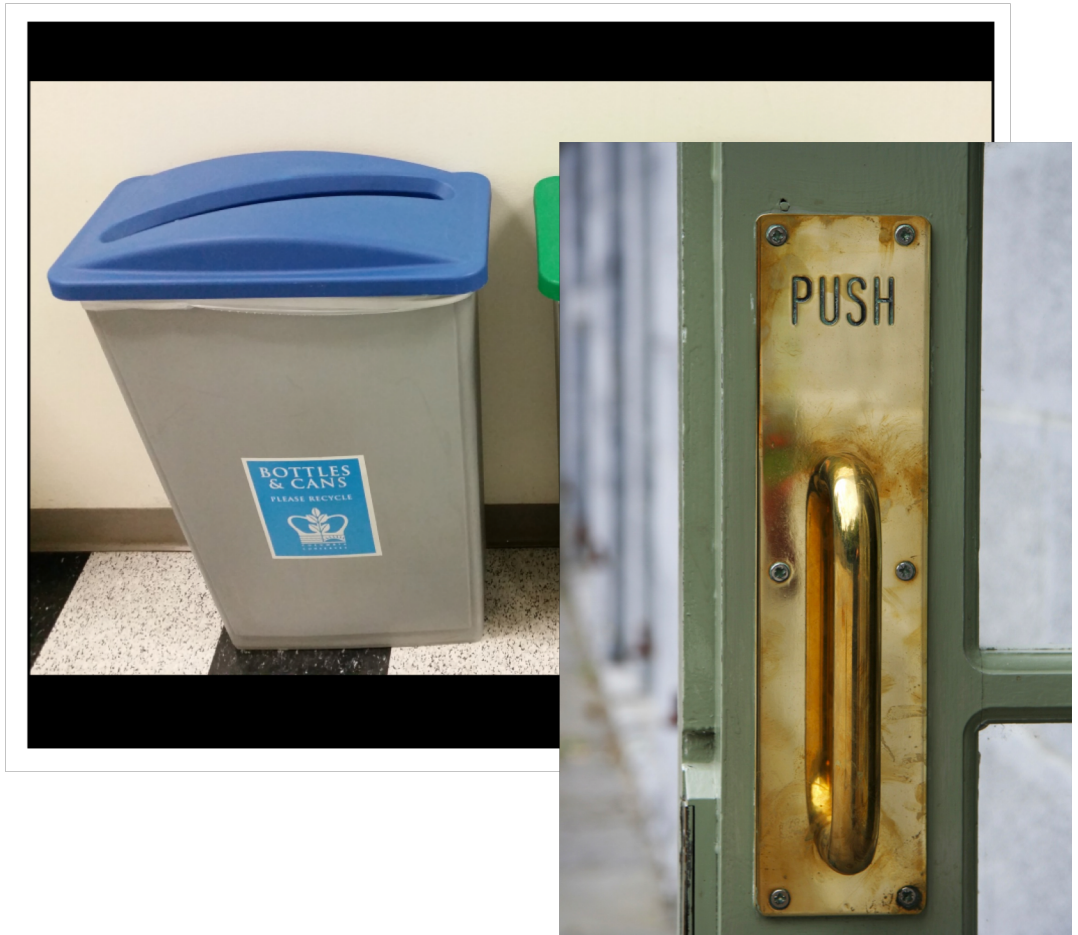




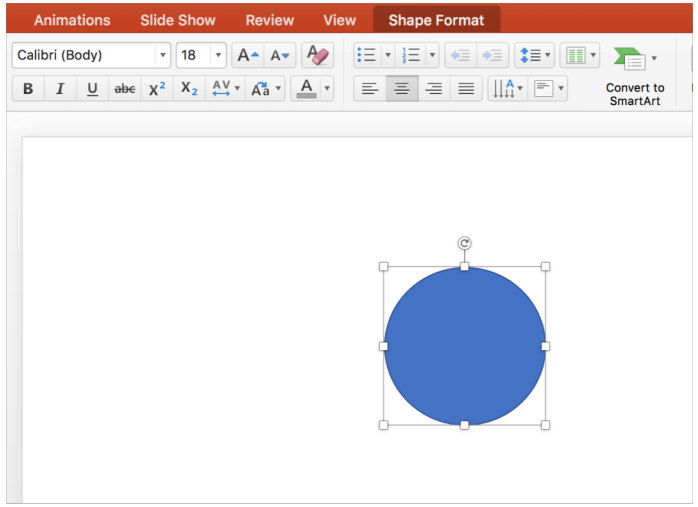
Hover event changes (like highlighting) often signify direct manipulation

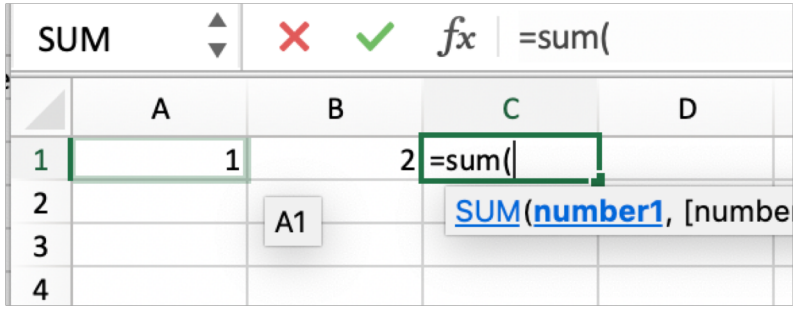
Design direct manipulation interfaces with good *perceived* affordances.

Bad signifiers / wrong perceived affordances

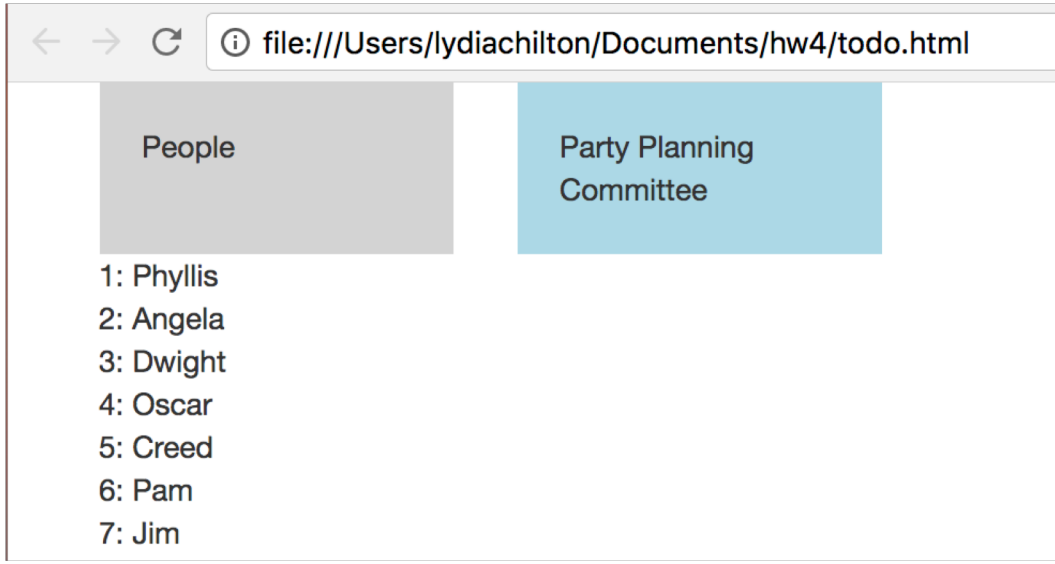


Good signifiers / correct perceived affordances





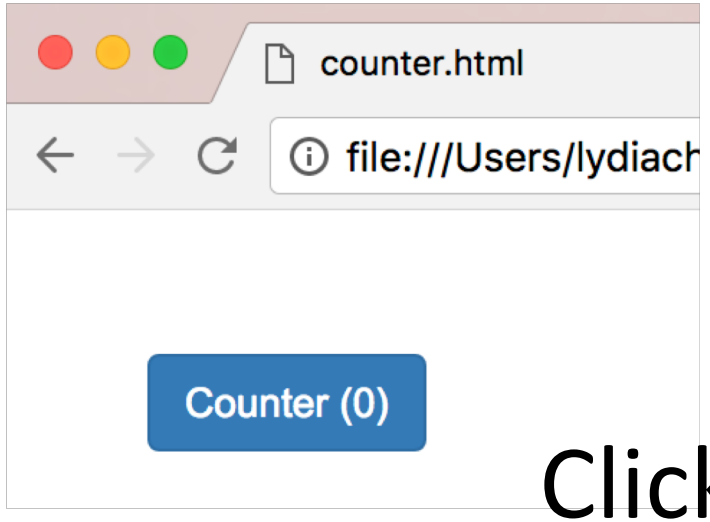
Implementing Direct Manipulation Interfaces



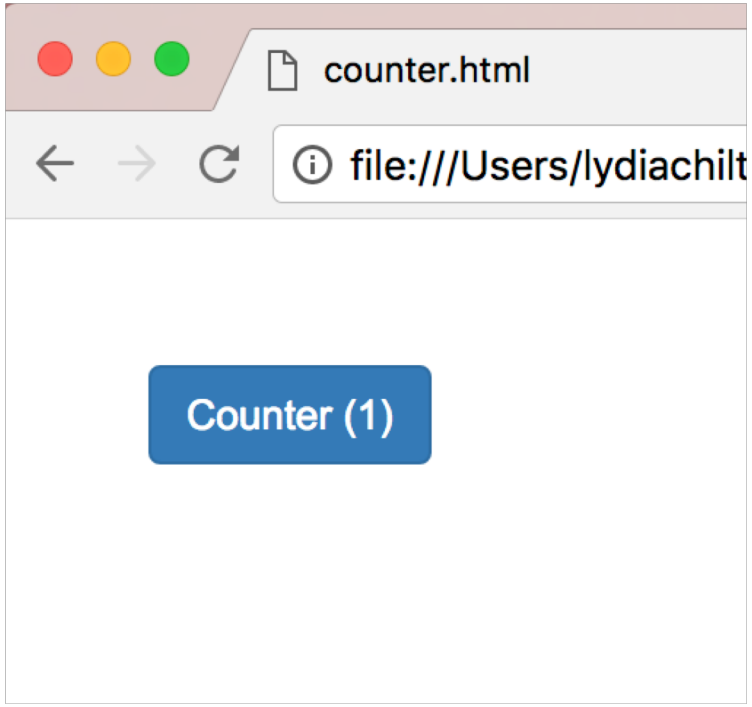
BUT FIRST…

Model, View, Controller (MVC) Style Programming

When users interact with data,
How do we update the database?



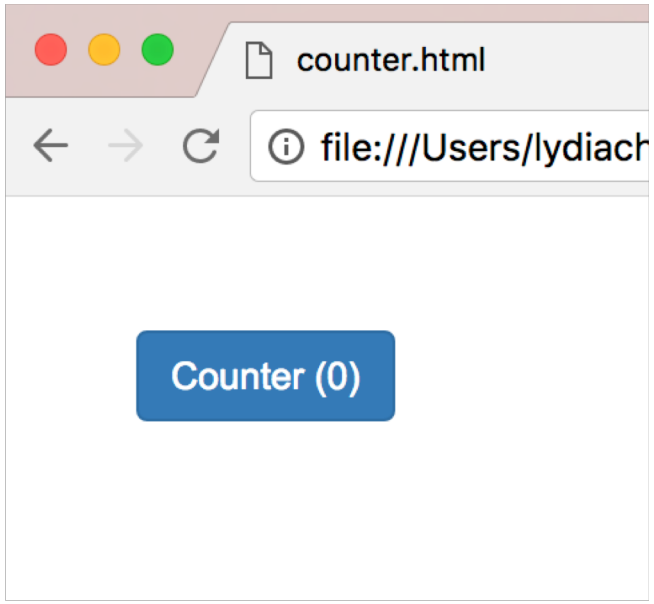
Click



Create a Button in HTML

HTML

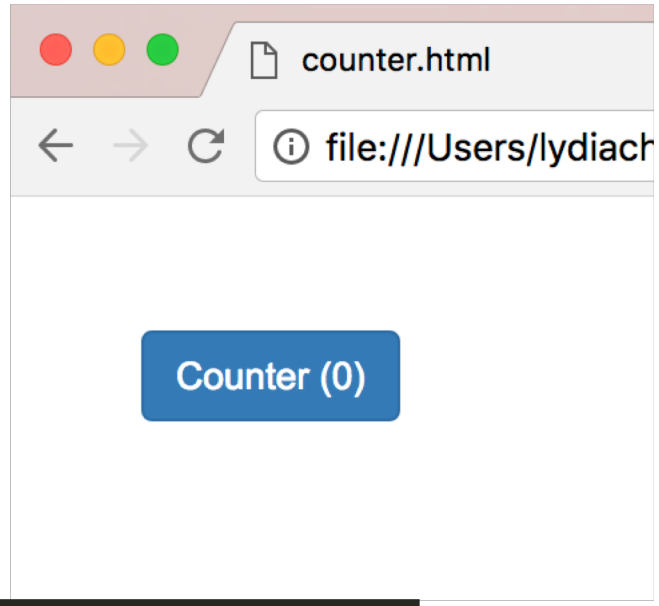
```html
<body>

    <button id="counter" class="btn btn-primary">Counter (0)</button>

</body>
```



Add JQuery and Bootstrap “libraries”

HTML

```html
<body>

    <button id="counter" class="btn btn-primary">Counter (0)</button>

</body>
```



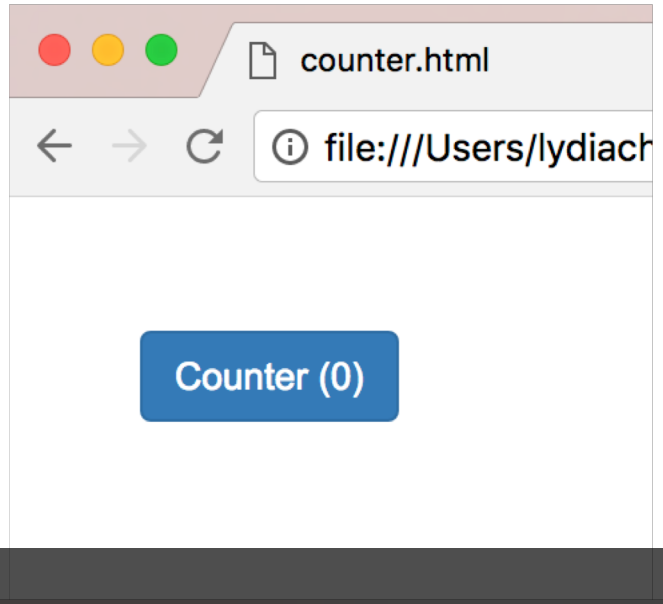
JavaScript

```html

<head>
    <!-- bootstrap -->
    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css">
    <!-- JQuery -->
    <script src="http://code.jquery.com/jquery-3.3.1.min.js"></script>
    <script src="http://code.jquery.com/ui/1.11.4/jquery-ui.min.js"></script>
</head>
```

We attach a click handler

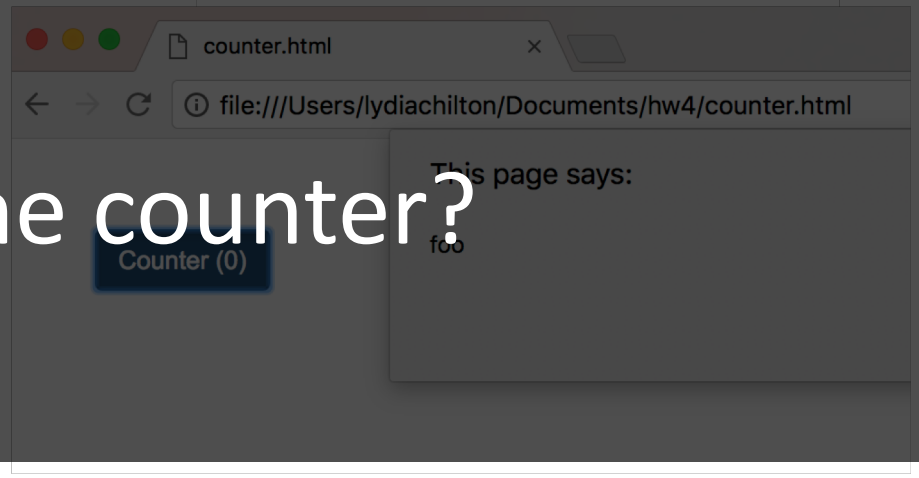
HTML

```html
<body>

    <button id="counter" class="btn btn-primary">Counter (0)</button>

</body>
```



JavaScript

```javascript
$(document).ready(function(){
    $("#counter").click(function(){
        alert("foo")
    })
})
```

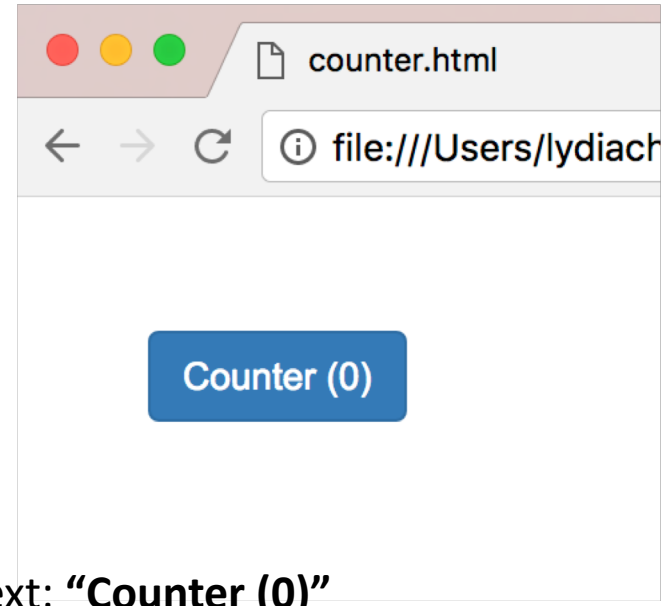


How do we increment the counter?

How **NOT** to increment the count?

HTML

```html
<body>

    <button id="counter" class="btn btn-primary">Counter (0)</button>

</body>
```



JavaScript

```javascript
$(document).ready(function(){
    $("#counter").click(function(){
        var html = $(this).html()

        var regExp = /\(([^)]+)\)/;
        var matches = regExp.exec(html);

        var number = 1*matches[1]
        var incremented_number = number +1

        $(this).html("Counter ("+incremented_number+")")
    })
})
```

Get the button text: **“Counter (0)”**

Extract the data from from the text

Cast to a number and add one

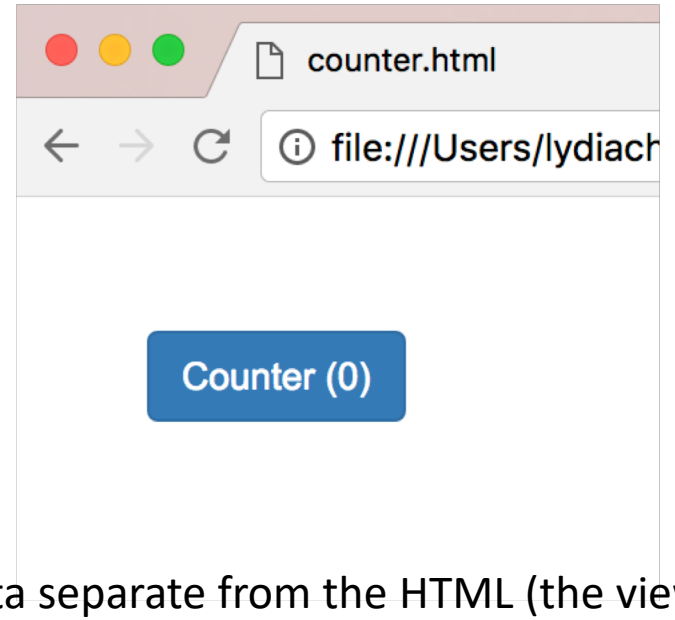
Replace the button text: **“Counter (1)”**

Problem? State is stored ONLY in the UI.

How TO increment the count?

HTML

```html
<body>

    <button id="counter" class="btn btn-primary"></button>

</body>
```



JavaScript

```javascript
var count = 0

function setCount(count){
    $("#counter").html("Counter ("+count+")")
}

$(document).ready(function(){
    setCount(count)

    $("#counter").click(function(){
        count = count +1
        setCount(count)
    })
})
```

Create a model of the data separate from the HTML (the view)

Create a function that can set the counter data to the view

When the page first loads, set the counter to 0

When the counter is clicked,
modify the data,
then update the view

Good UI programming separates the **data** model from the **view** and controller

Not MVC: data stored in UI

```
31  <body>
32
33      <button id="counter" class="btn btn-primary">Counter (0)</button>
34
35  </body>
36
26  $(document).ready(function(){
27      $("#counter").click(function(){
28          var html = $(this).html()
29
30          var regExp = /\(([^)]+)\)/;
31          var matches = regExp.exec(html);
32
33          var number = 1*matches[1]
34          var incremented_number = number +1
35
36          $(this).html("Counter ("+incremented_number+")")
37      })
38  })
39
```

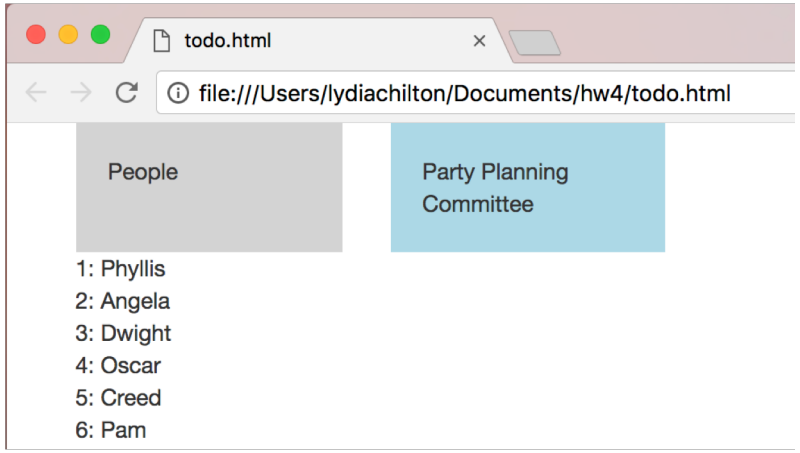
Good (MVC): Data stored as a variable. UI generated from data

```
42  var count = 0
43
44  function setCount(count){
45      $("#counter").html("Counter ("+count+")")
46  }
47
48  $(document).ready(function(){
49      setCount(count)
50
51      $("#counter").click(function(){
52          count = count +1
53          setCount(count)
54      })
55  })
```

Implementing Direct Manipulation

Direct Manipulation Properties

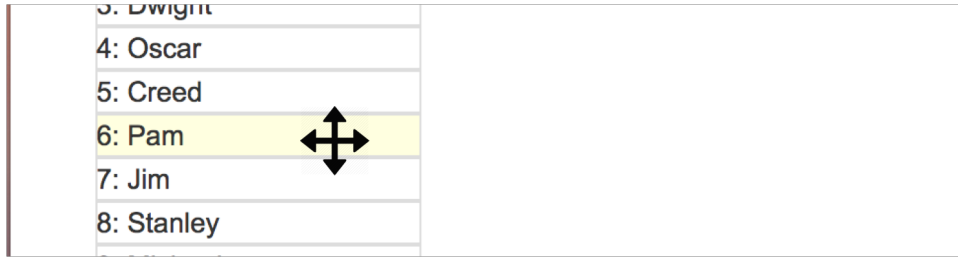
1. **Objects** are represented visually



2. **Actions** are rapid, incremental and reversible

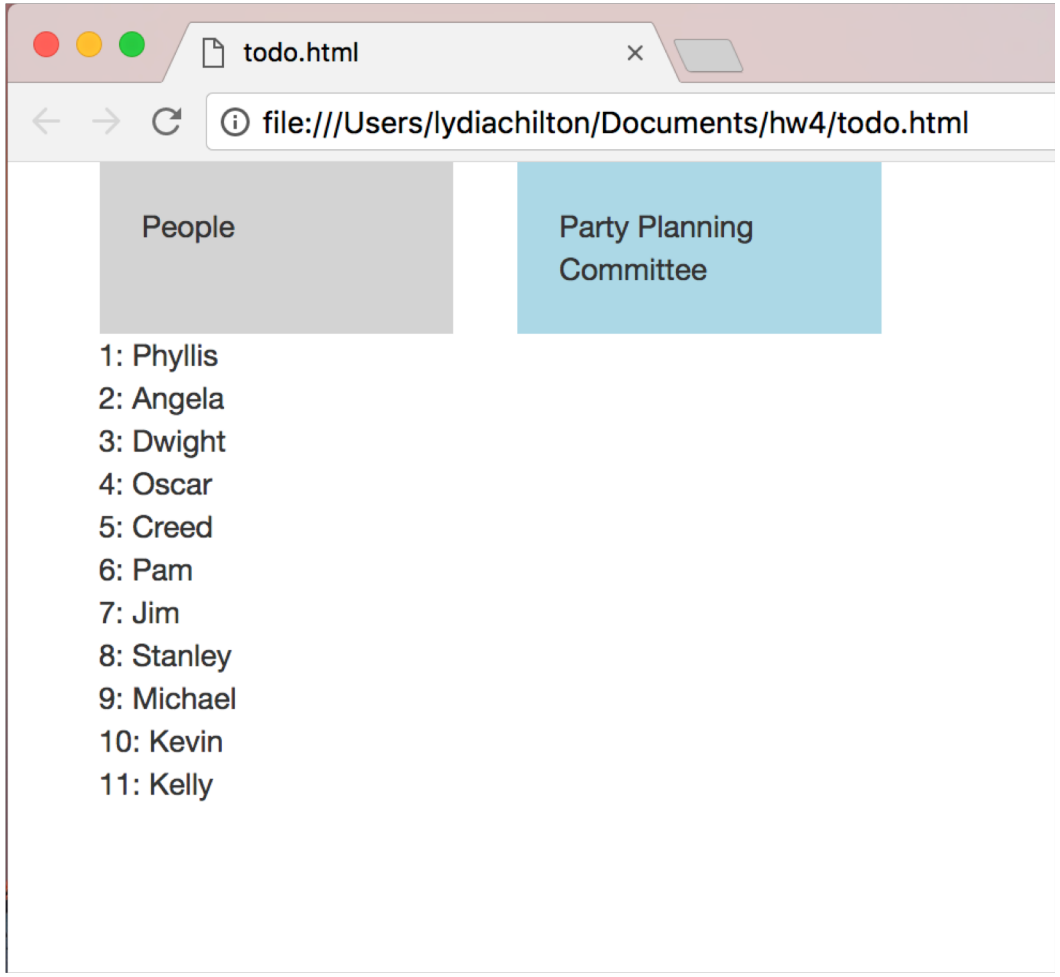
Drag and Drop

3. User interacts **directly with object representations**



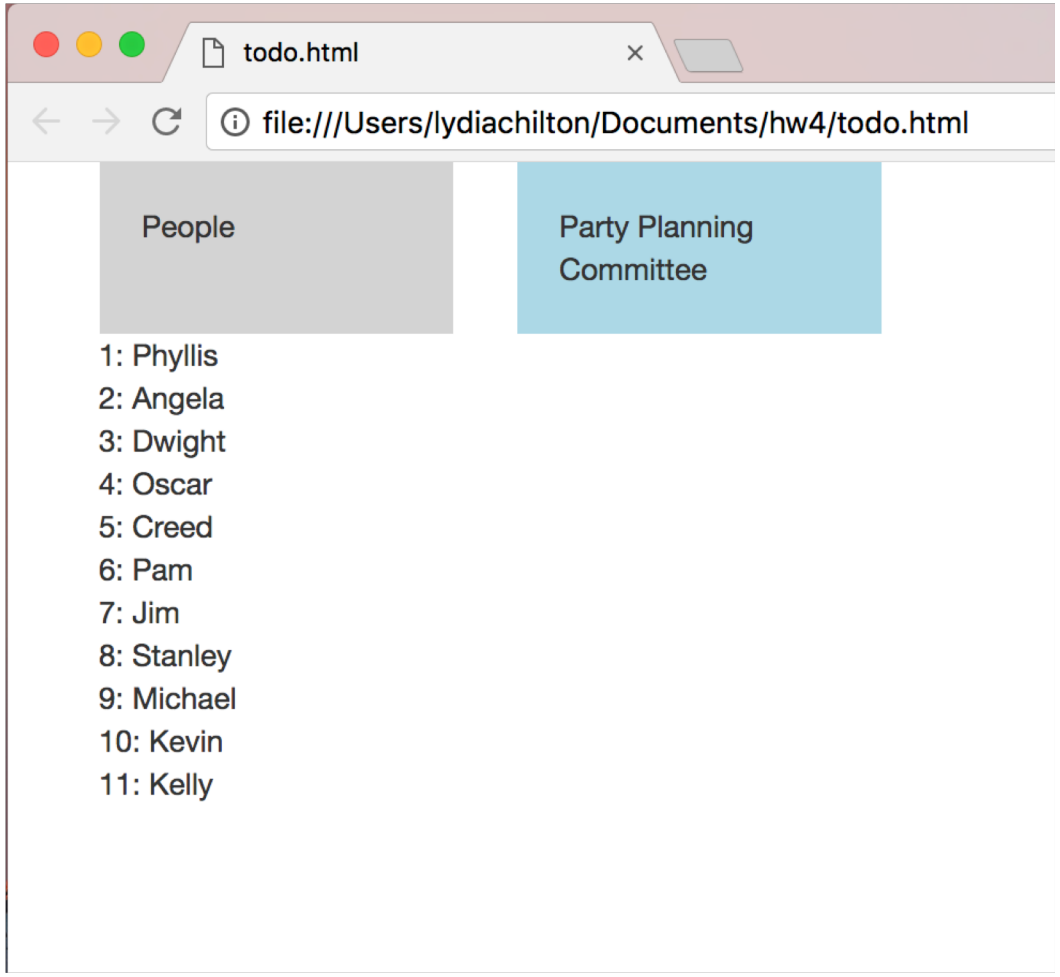
How to NOT implement this?

```
27
28      <div class="header"> NAMES </div>
29      <div>Phyllis</div>
30      <div>Angela</div>
31      <div>Dwight</div>
32      <div>Oscar</div>
33      <div>Creed</div>
34      <div>Stanley</div>
35
36
```



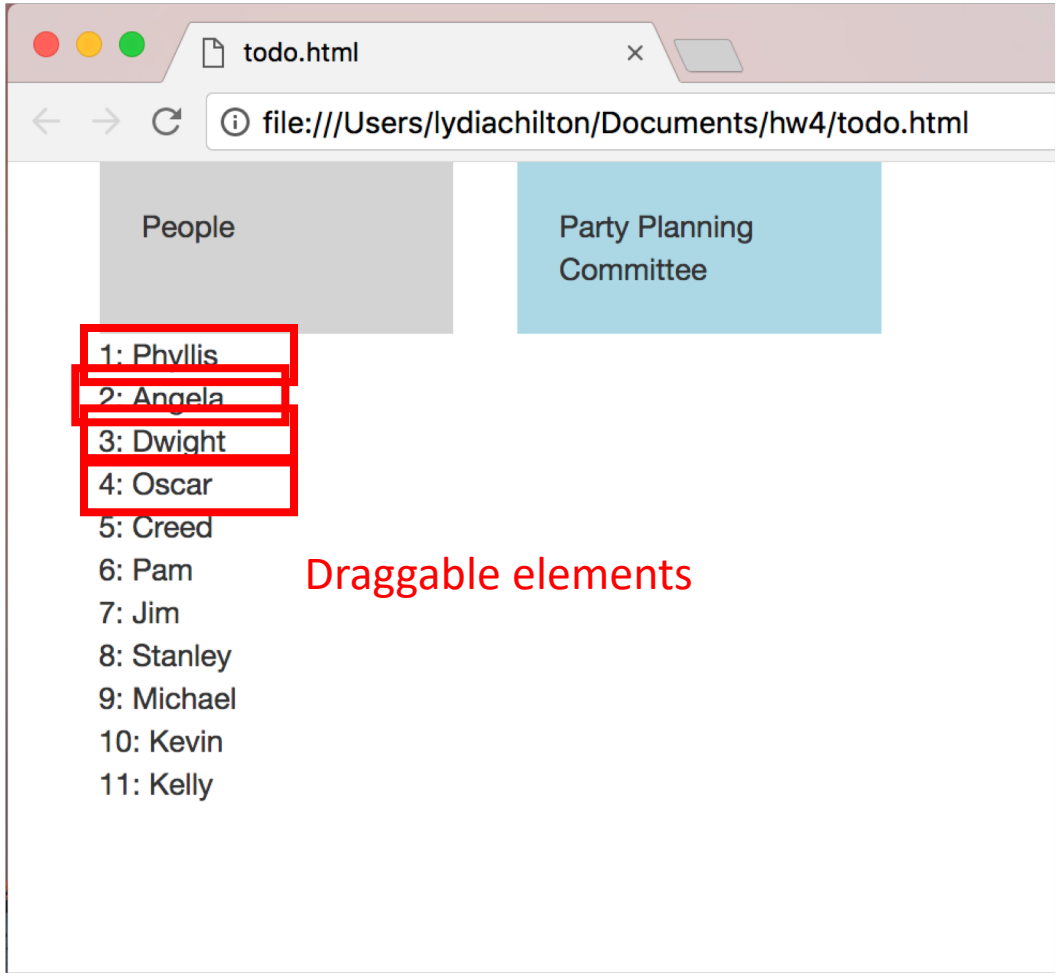
Step 1. Create the Data Model

```
1   var names = [
2   "Phyllis",
3   "Angela",
4   "Dwight",
5   "Oscar",
6   "Creed",
7   "Pam",
8   "Jim",
9   "Stanley",
10  "Michael",
11  "Kevin",
12  "Kelly"
13  ]
14
15  var list1 = []
16
```



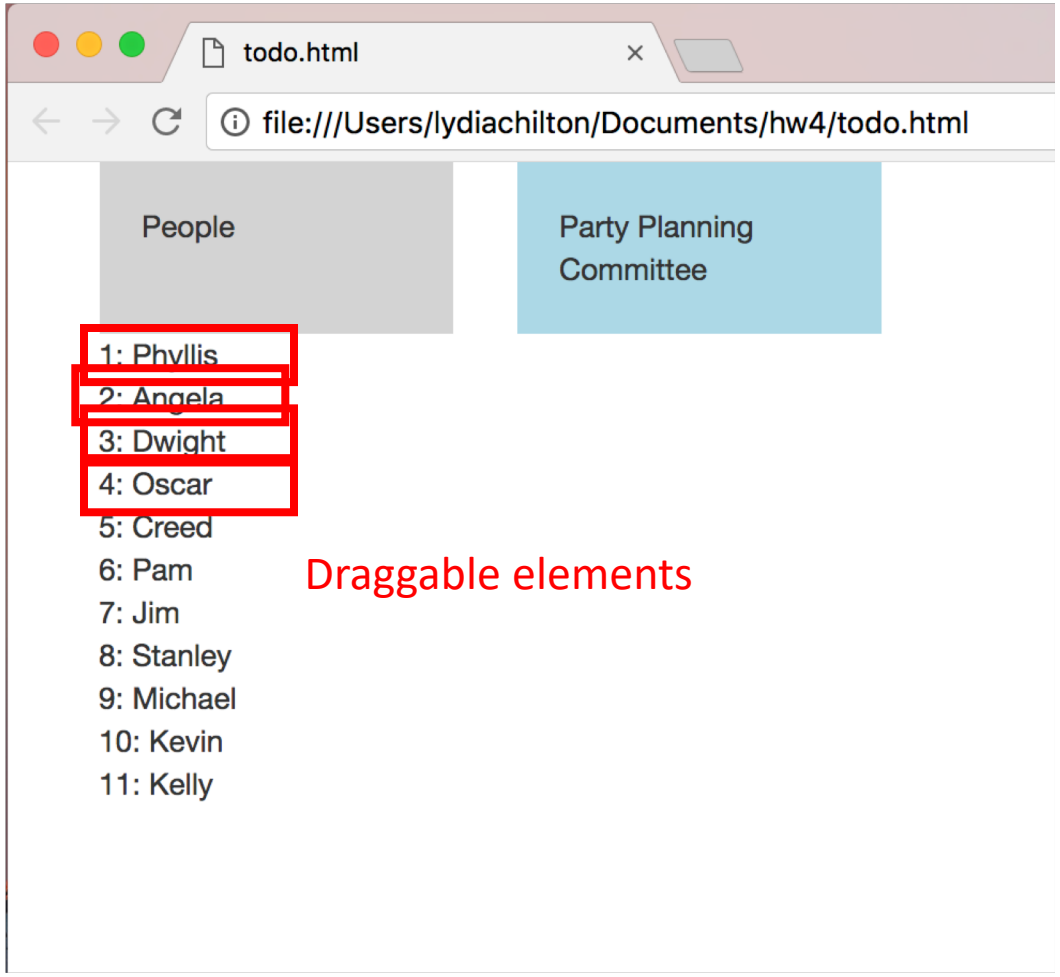
Step 2. Create a function that updates the view with new data

```
170
171  var names = [
172  "Phyllis",
173  "Angela",
174  "Dwight",
175  "Oscar",
176  "Creed",
177  "Pam"
178  ]
179  var list1 = []
180
181
182  function makeNames(names){
183      $("#names").empty()
184      $.each(names, function( index, value ) {
185          //make the draggable name object
186      });
187  }
188
```



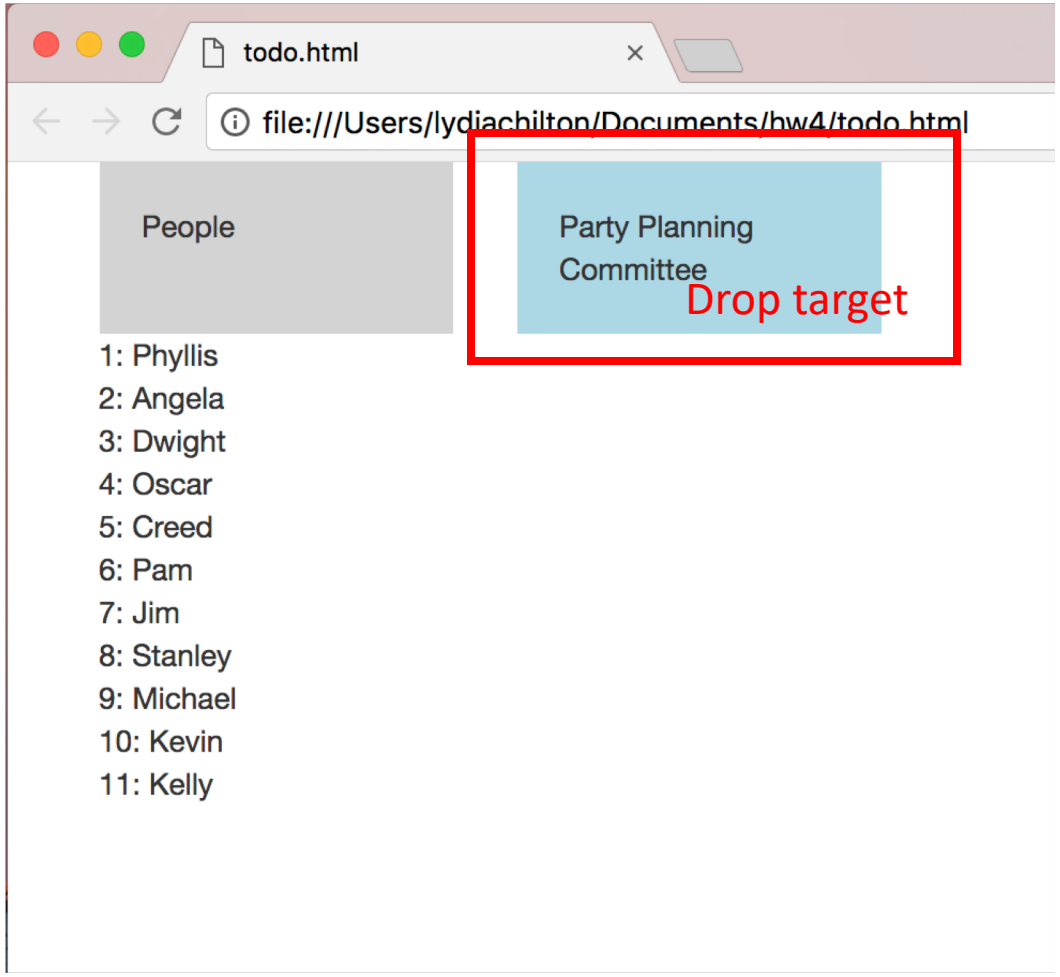
Step 3. On page load, create view.

```
170
171  var names = [
172  "Phyllis",
173  "Angela",
174  "Dwight",
175  "Oscar",
176  "Creed",
177  "Pam"
178  ]
179  var list1 = []
180
181
182  function makeNames(names){
183      $("#names").empty()
184      $.each(names, function( index, value ) {
185          //make the draggable name object
186      });
187  }
188
189
190  $(document).ready(function(){
191      makeNames(names)
192
```

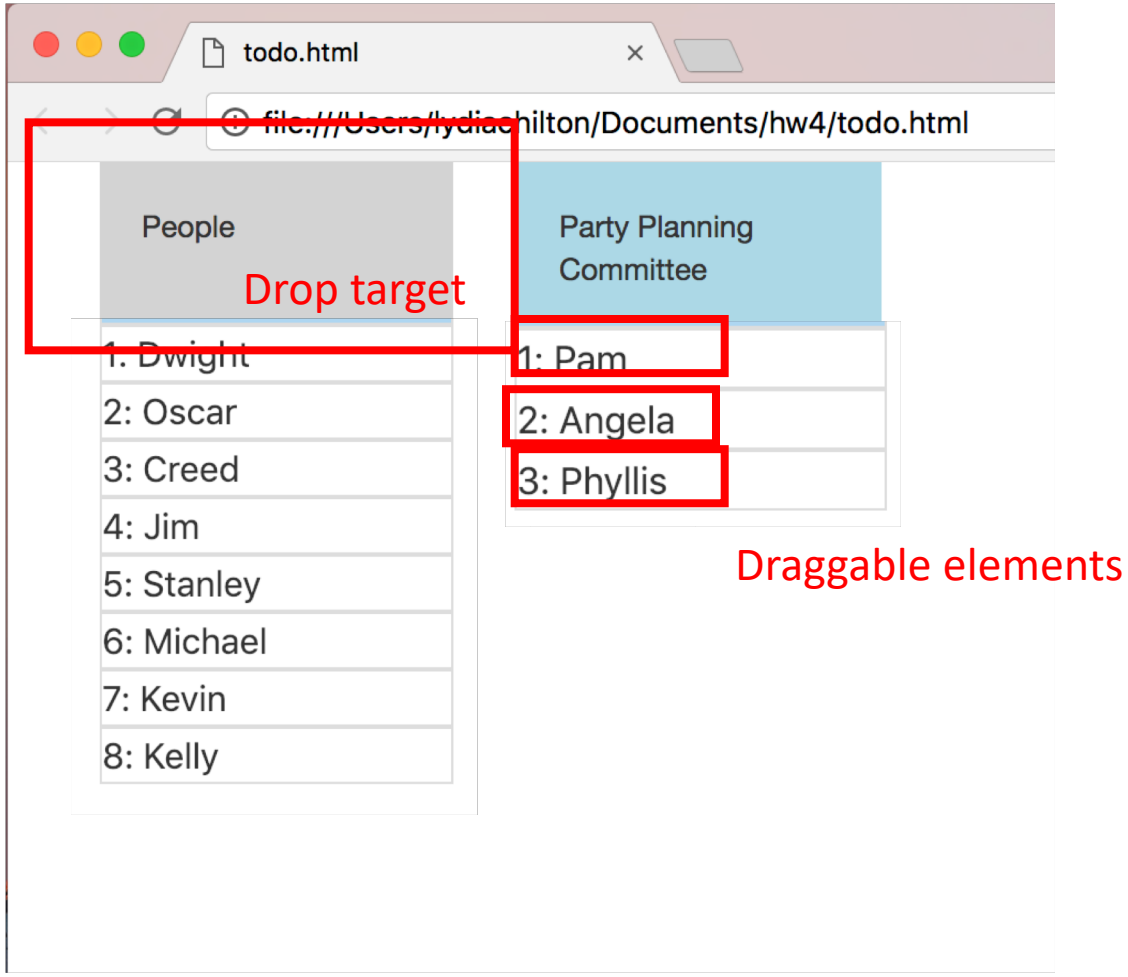


Step 4. Attach an **drop event** to the **drop target**. It should update the data, then update the view

```
$(document).ready(function(){
    makeNames(names)

    $( "#ppc_label" ).droppable({
        drop: function( event, ui ) {
            //get dropped name

            //update names array

            //update list1 array

            //update the interface to display the new lists
        }
    });
})
```

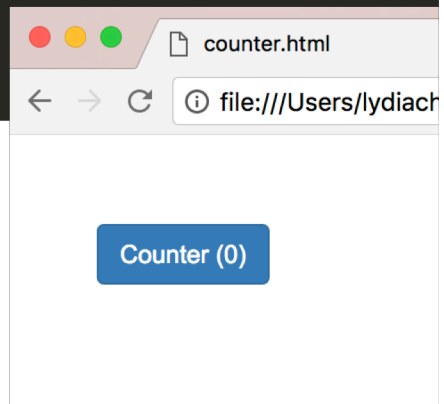


Step 5. What else do we need to do?

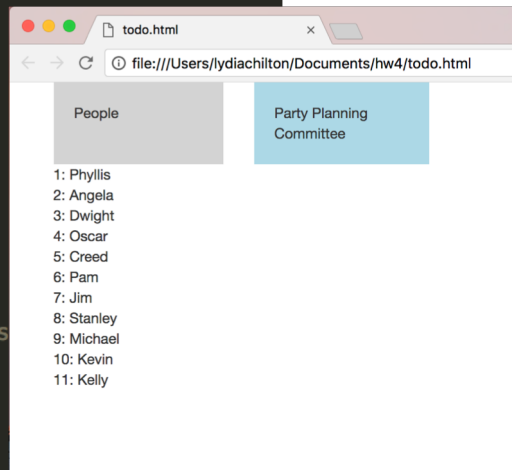


Good UI programming separates the data model from the view and controller

```
var count = 0

function setCount(count){
    $("#counter").html("Counter ("+count+")")
}

$(document).ready(function(){
    setCount(count)

    $("#counter").click(function(){
        count = count +1
        setCount(count)
    })
})
```

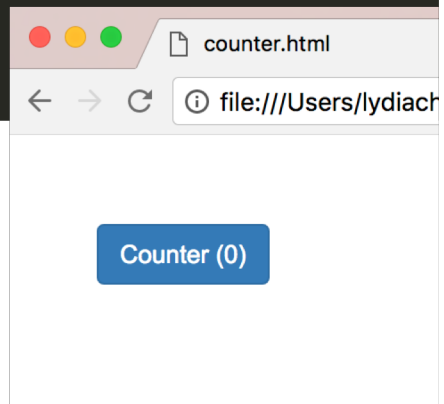



```
var names = [
"Phyllis",
"Angela",
"Dwight",
"Oscar",
"Creed",
"Pam"
]
var list1 = []


function makeNames(names){
    $("#names").empty()
    $.each(names, function( index, value ) {
        //make the draggable name object
    });
}


$(document).ready(function(){
    makeNames(names)
    $( "#ppc_label" ).droppable({
        drop: function( event, ui ) {
            //get dropped name

            //update names array

            //update list1 array

            //update the interface to dis
        }
    });
})
```

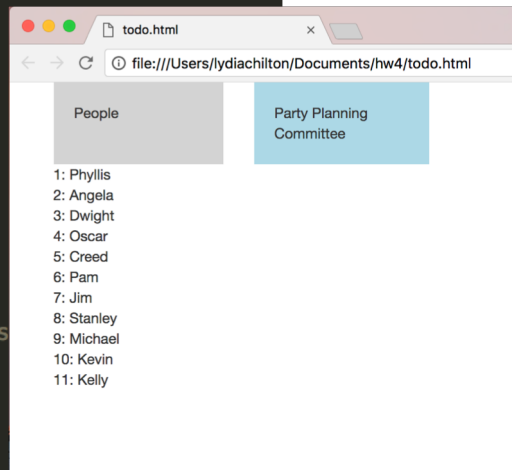


Good UI programming separates the data model from the view and controller

```
var count = 0

function setCount(count){
    $("#counter").html("Counter ("+count+")")
}

$(document).ready(function(){
    setCount(count)

    $("#counter").click(function(){
        count = count +1
        setCount(count)
    })
})
```

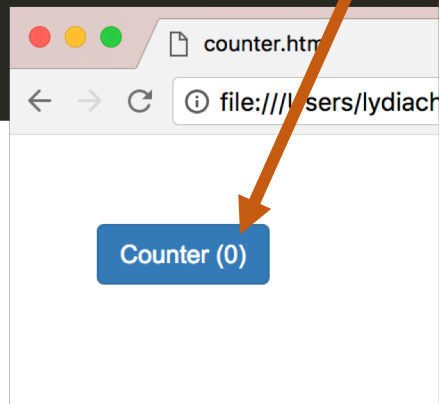



```
var names = [
"Phyllis",
"Angela",
"Dwight",
"Oscar",
"Creed",
"Pam"
]
var list1 = []


function makeNames(names){
    $("#names").empty()
    $.each(names, function( index, value ) {
        //make the draggable name object
    });
}


$(document).ready(function(){
    makeNames(names)
    $( "#ppc_label" ).droppable({
        drop: function( event, ui ) {
            //get dropped name

            //update names array

            //update list1 array

            //update the interface to dis
        }
    });
})
```

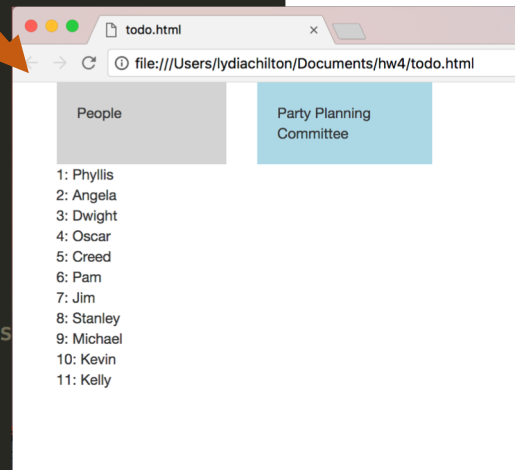


Good UI programming separates the data model from the view and controller

```
var count = 0

function setCount(count){
    $("#counter").html("Counter ("+count+")")
}

$(document).ready(function(){
    setCount(count)

    $("#counter").click(function(){
        count = count +1
        setCount(count)
    })
})
```

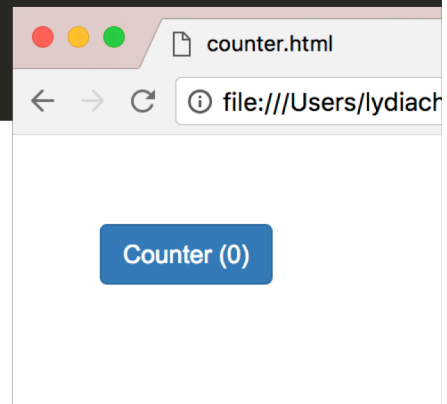



```
var names = [
"Phyllis",
"Angela",
"Dwight",
"Oscar",
"Creed",
"Pam"
]
var list1 = []


function makeNames(names){
    $("#names").empty()
    $.each(names, function( index, value ) {
        //make the draggable name object
    });
}


$(document).ready(function(){
    makeNames(names)
    $( "#ppc_label" ).droppable({
        drop: function( event, ui ) {
            //get dropped name

            //update names array

            //update list1 array

            //update the interface to dis
        }
    });
})
```

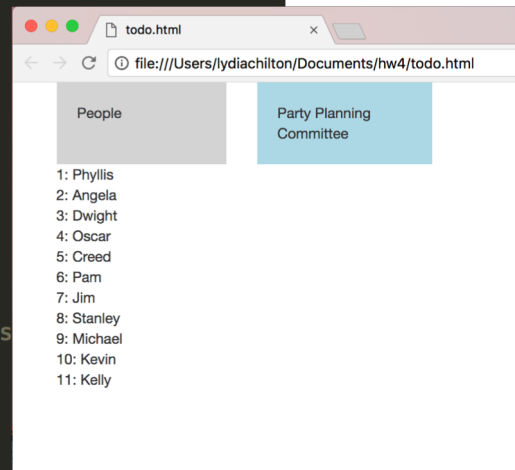


Good UI programming separates the data model from the view and controller

```
var count = 0

function setCount(count){
    $("#counter").html("Counter ("+count+")")
}

$(document).ready(function(){
    setCount(count)

    $("#counter").click(function(){
        count = count +1
        setCount(count)
    })
})
```




```
var names = [
"Phyllis",
"Angela",
"Dwight",
"Oscar",
"Creed",
"Pam"
]
var list1 = []


function makeNames(names){
    $("#names").empty()
    $.each(names, function( index, value ) {
        //make the draggable name object
    });
}


$(document).ready(function(){
    makeNames(names)
    $( "#ppc_label" ).droppable({
        drop: function( event, ui ) {
            //get dropped name

            //update names array

            //update list1 array

            //update the interface to dis
        }
    });
})
```



Summary

Direct Manipulation Properties

1. **Objects** are represented visually

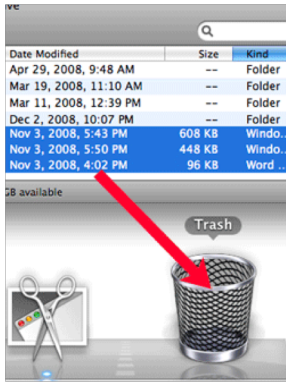
2. **Actions** are rapid, incremental and reversible

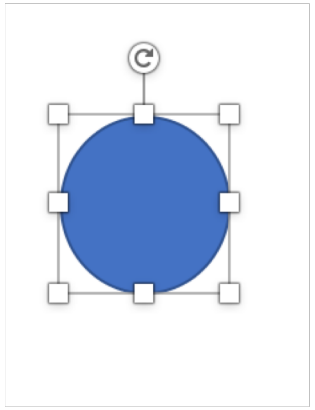
Move to trash

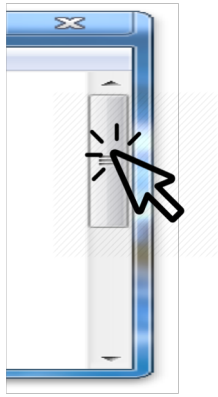
Resize

Move viewport

3. User interacts **directly with object representations**







Signifiers help users perceive affordances

Bad signifiers

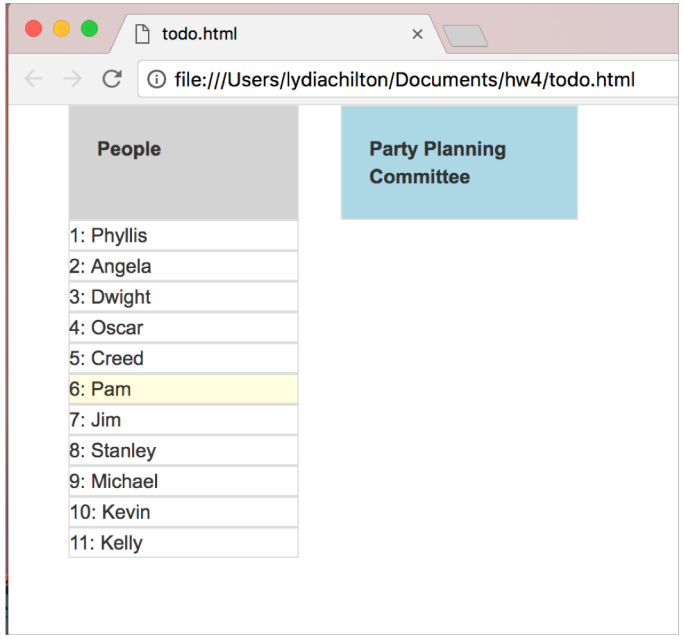
Signifier	Handle that can be yanked toward you
Perceived affordance	**Pull**
Affordance	**Push**

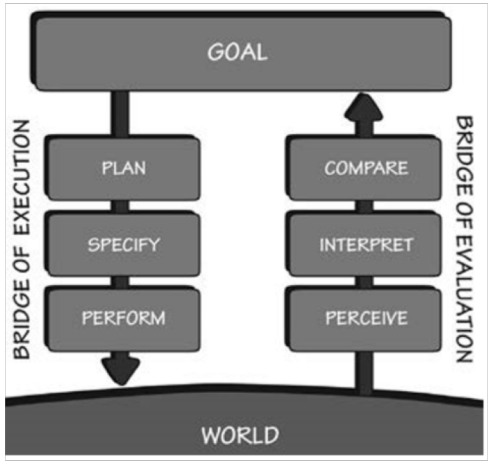
Good signifiers

Signifier	Handle that can be leaned on
Perceived affordance	**Push**
Affordance	**Push**

Direct manipulation interfaces suit the 7 stages of action

There are visible **actions** the user can **execute**

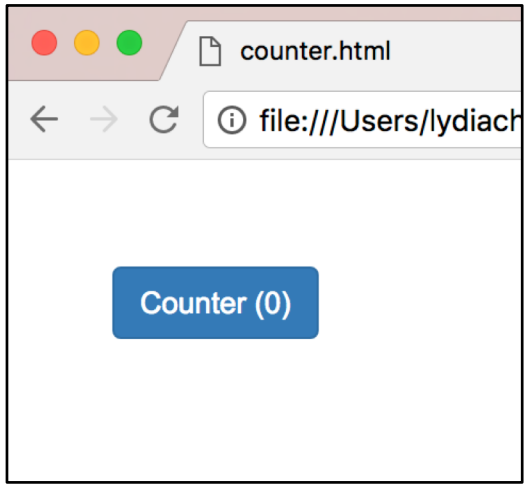




There is visible **feedback** the user can **evaluate**

When implementing Direct Manipulation:

Create an **object** in the view



Add an **event handler** to respond to user's actions

```
$(document).ready(function(){
    $("#counter").click(function(){
        alert("foo")
    })
})
```

Modify the **data**, then update the **view**

```
var count = 0

function setCount(count){
    $("#counter").html("Counter ("+count+")")
}

$(document).ready(function(){
    setCount(count)

    $("#counter").click(function(){
        count = count +1
        setCount(count)
    })
})
```

It is important to separate the data (or model) from the view and the controller.

Announcements

- Class Wednesday is canceled
- My office hours Wednesday are canceled
  - I will reschedule these
- Homework 5 is due Friday
- Don’t forget participation