

Direct Manipulation

No screens



Prof. Lydia Chilton
COMS 4170
5 February 2018

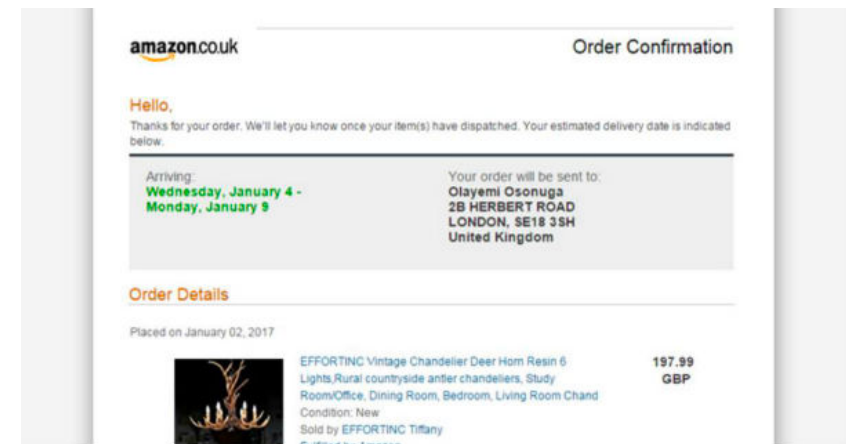
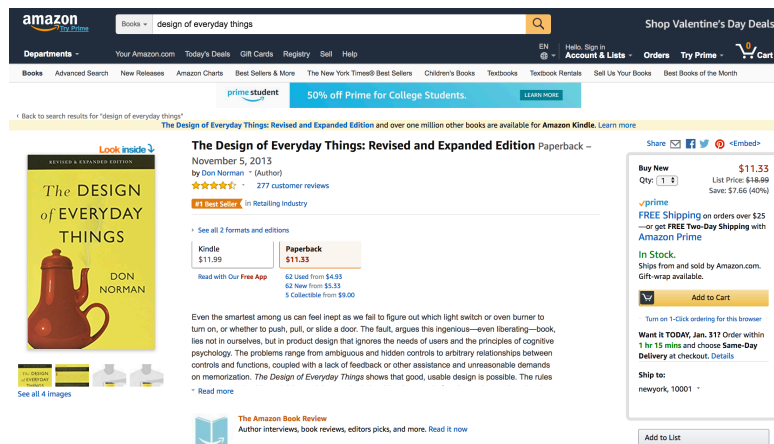
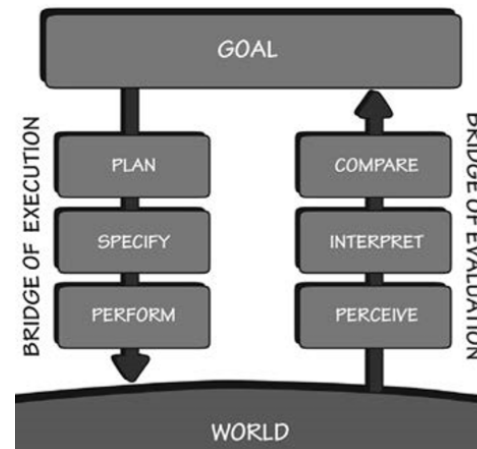
Say your name



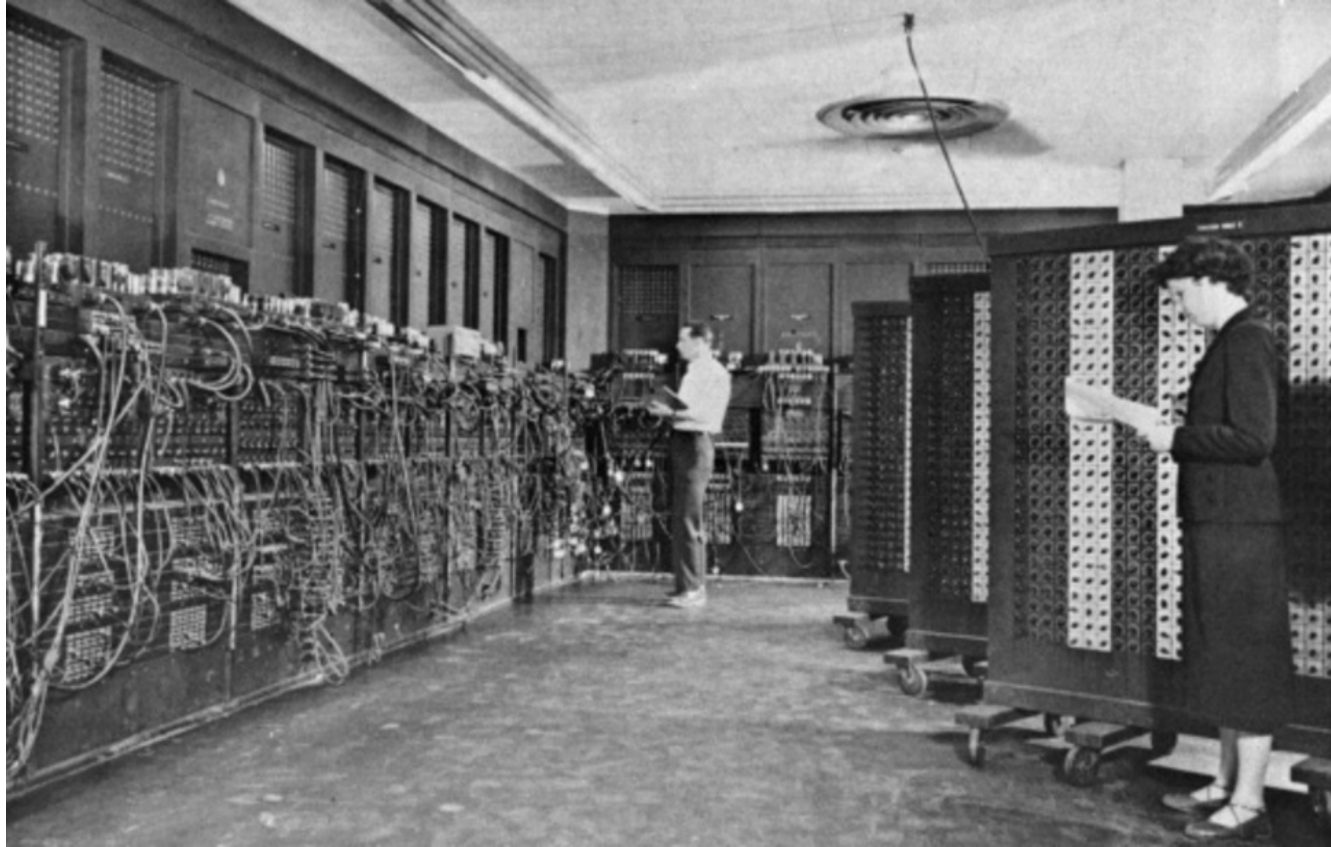
Goal 1

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

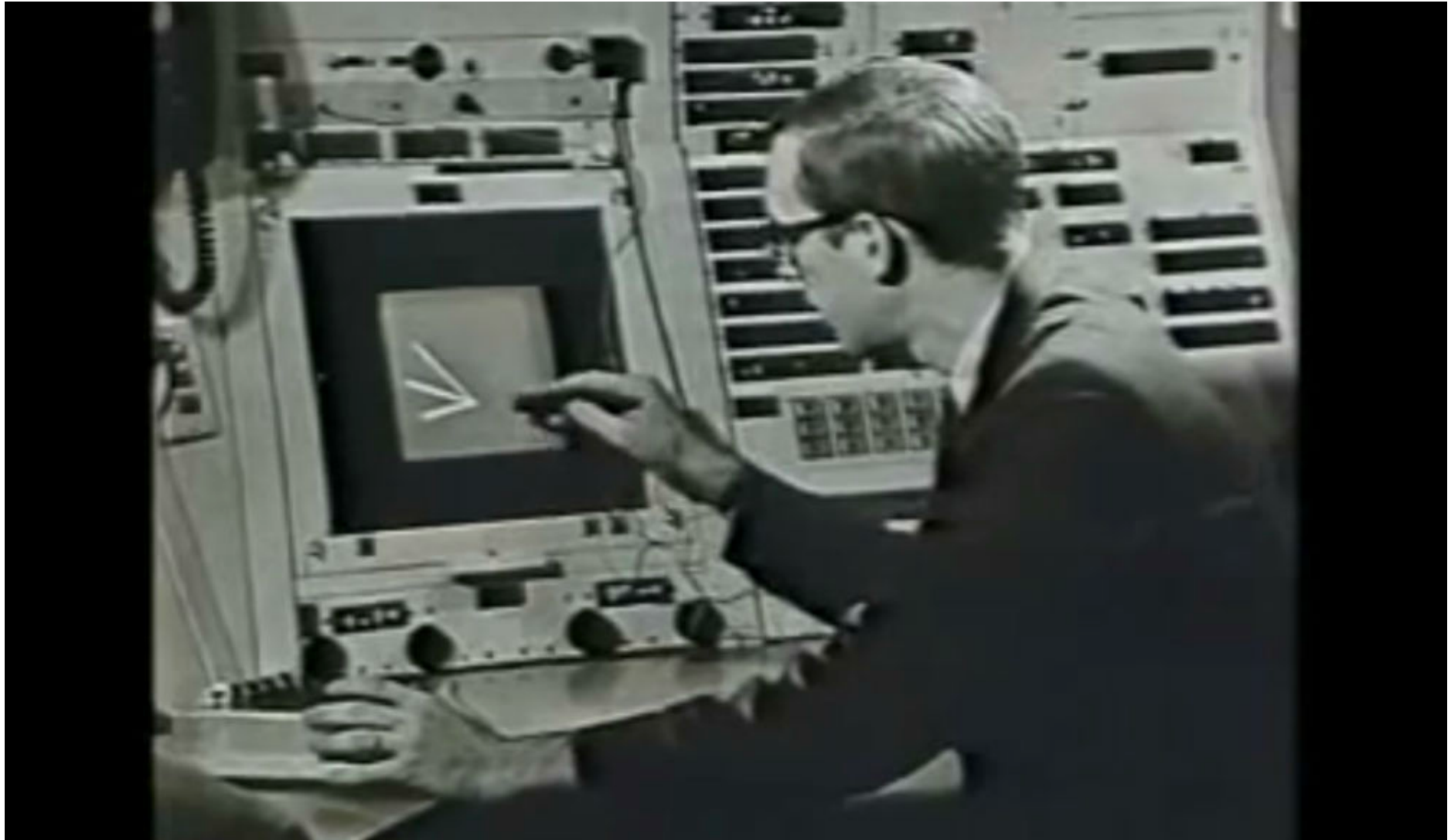


```
ASSUME CS:CODE,DS:DATA
DATA SEGMENT
LIST DW 2579H,0A500H,0C009H,0159H,0B900H
COUNT EQU 05H
DATA ENDS
CODE SEGMENT
START:      XOR BX,BX
            XOR DX,DX
            MOV AX,DATA
            MOV DS,AX
            MOV CL,COUNT
            MOV SI,OFFSET LIST

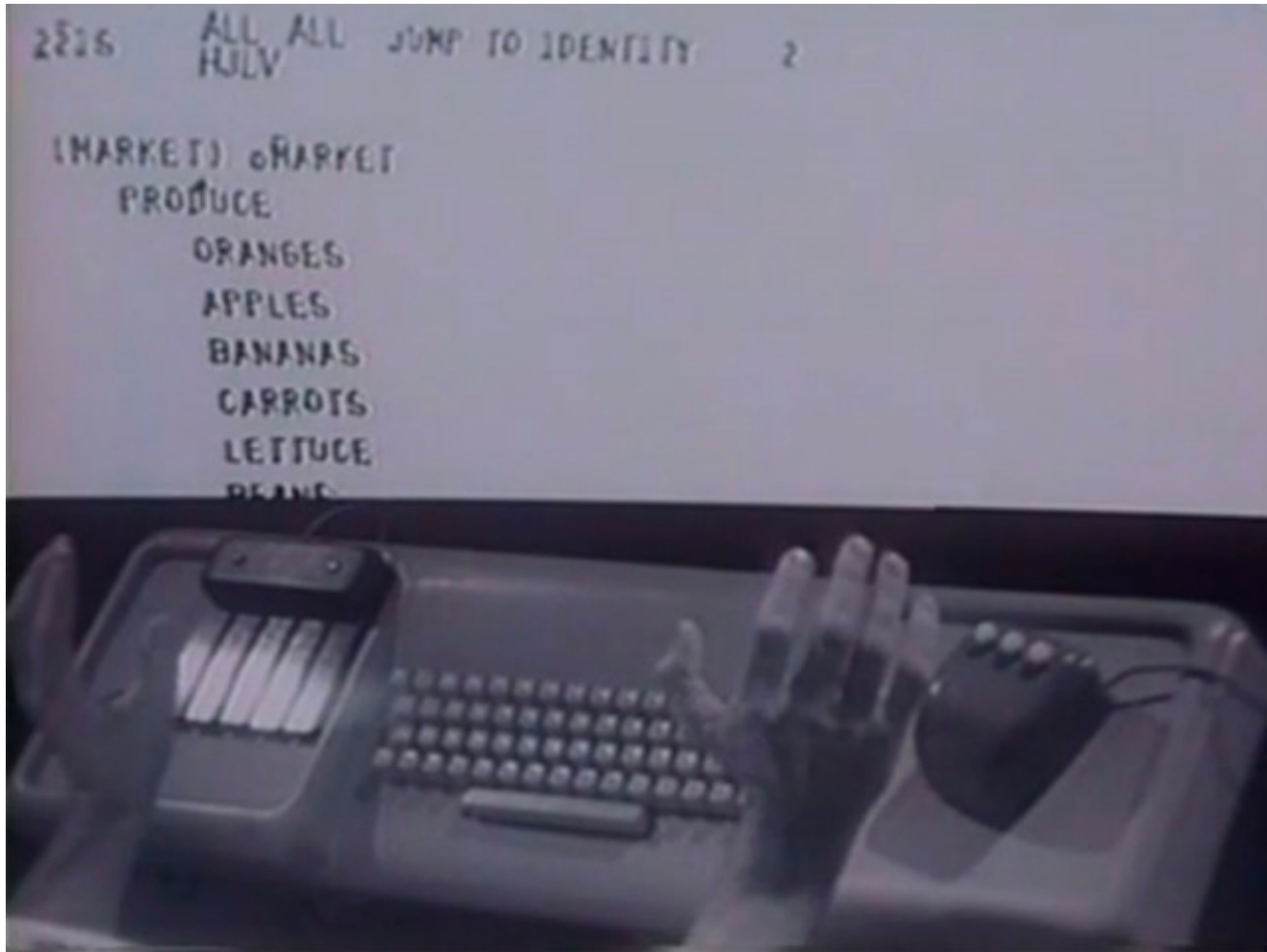
AGAIN:     MOV AX,[SI]
            SHL AX,01
            JC NEG
            INC BX
            JMP NEXT

NEG:      INC DX
NEXT:     ADD SI,02
            DEC CL
            JNZ AGAIN
            MOV AH,4CH
            INT 21H
CODE ENDS
END START
```

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

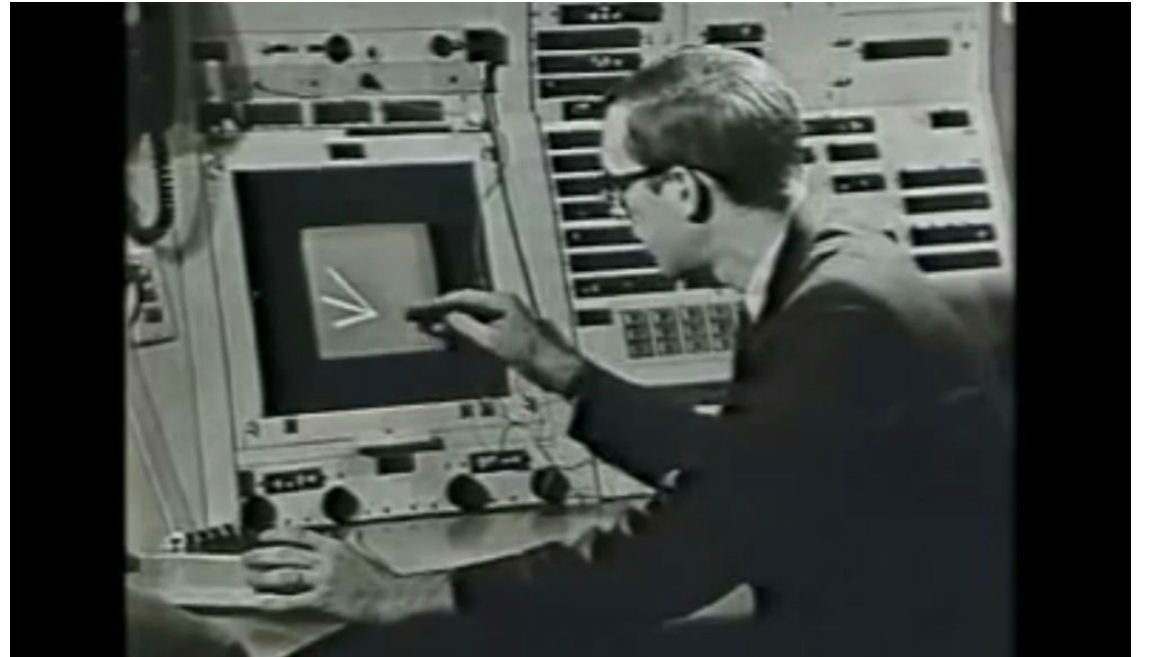


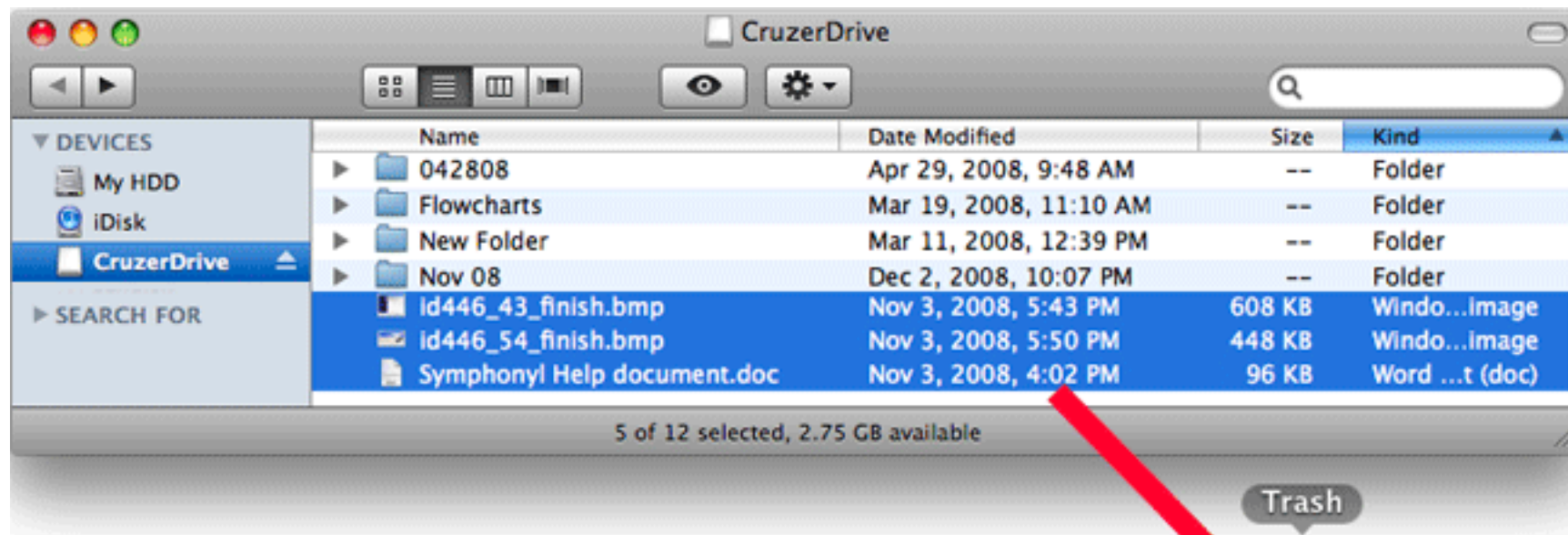
1968: Interaction devices for computer use. Douglas Engelbart's mouse



Direct Manipulation Properties

1. **Objects** are represented visually
2. **Actions** are rapid, incremental and reversible
3. User interacts **directly with object representations**





Calibri (Body) 18 A A A

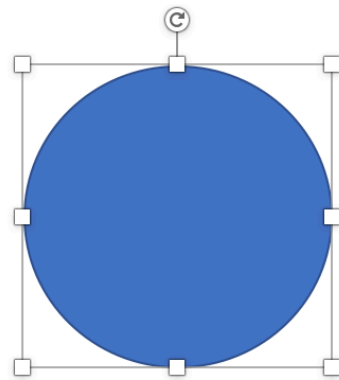
B *I* U abc X^2 X_2 AV Aa A

Convert to SmartArt

Picture Shapes Text Box

Arrange Quick Styles

Shape Fill Shape Outline



Presentation2 - PowerPoint

File Home Insert Design Transitions Animations Slide Show Review View Mix ACROBAT Format Tell me... Heather A... Share

Shape Fill

Theme Colors

Standard Colors

No Fill

More Fill Colors...

Eyedropper

Picture...

Gradient

Texture

Bring Forward

Send Backward

Selection Pane

Align

Group

Rotate

WordArt Styles

Arrange

Size

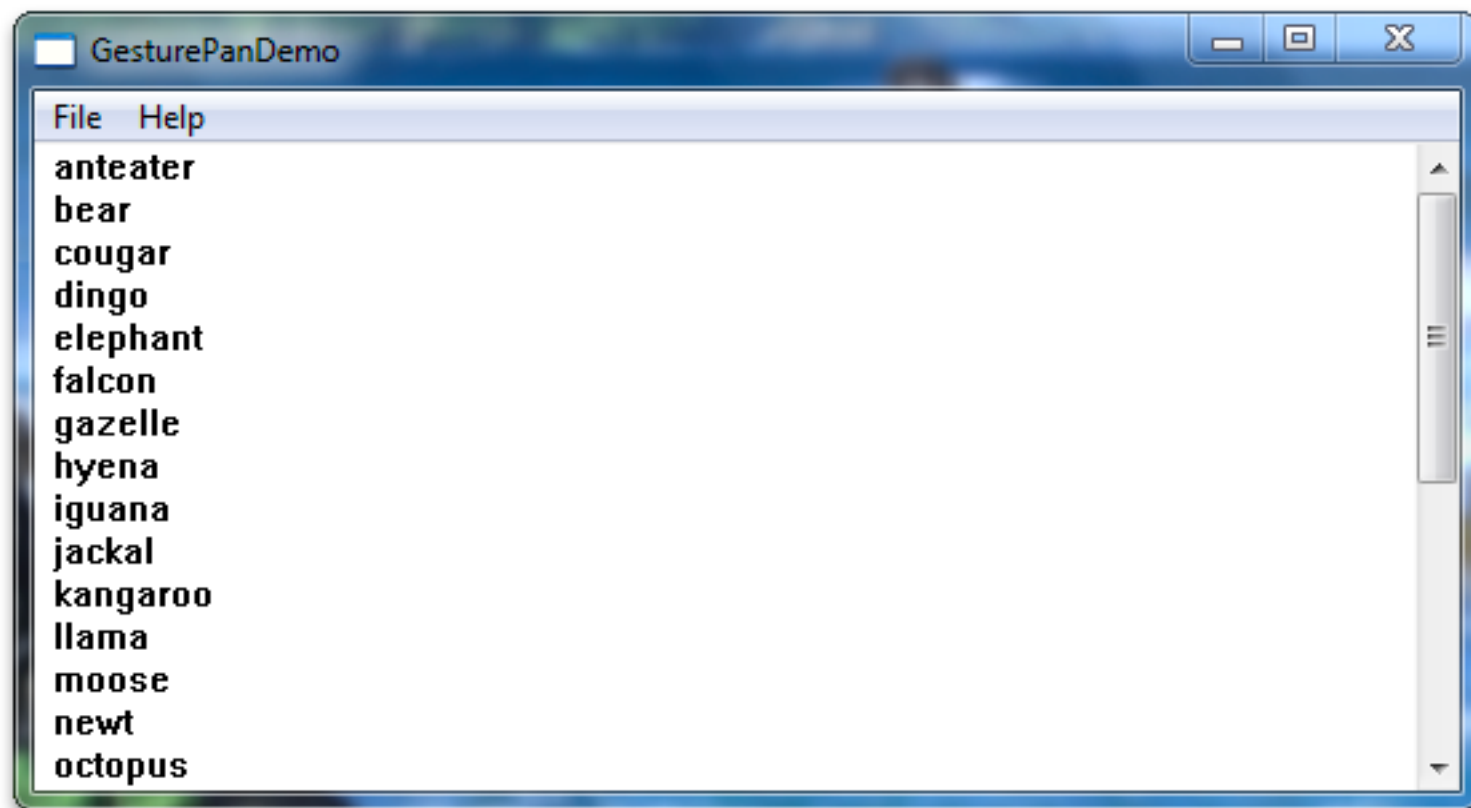
2.49"

2.6"

Slide 5 of 5

Notes Comments

63%



The image shows a screenshot of the Microsoft Excel application. At the top, there is a dark green ribbon with tabs for 'Home', 'Insert', 'Page Layout', 'Formulas', 'Data', and 'Review'. The 'Home' tab is active, displaying various editing and formatting options. Below the ribbon is the formula bar, which shows the text 'SUM' on the left and a formula '=B2*C2' on the right. The spreadsheet grid below has columns labeled A through F and rows numbered 1 through 11. The data in the grid is as follows:

| | A | B | C | D | E | F |
|----|----------|-------|----------|--------|---|---|
| 1 | | price | Quantity | | | |
| 2 | Cookies | 2 | 215 | =B2*C2 | | |
| 3 | Icecream | 4 | 860 | | | |
| 4 | Twinkies | 3 | 350 | | | |
| 5 | Cake | 6 | 133 | | | |
| 6 | Donuts | 1 | 458 | | | |
| 7 | | | | | | |
| 8 | | | | | | |
| 9 | | | | | | |
| 10 | | | | | | |
| 11 | | | | | | |

The image shows a screenshot of the Microsoft Excel application. The top ribbon is set to 'Home', with sub-ribbons for 'Insert', 'Page Layout', and 'Formulas'. The 'Home' ribbon is active, showing options for 'Paste', 'Cut', 'Copy', and 'Format'. The font settings are 'Calibri (Body)' and size '12'. The formula bar shows the active cell 'D2' with the formula $=B2*C2$. The spreadsheet data is as follows:

| | A | B | C | D | E |
|----|----------|-------|----------|-----|---|
| 1 | | price | Quantity | | |
| 2 | Cookies | 2 | 215 | 430 | |
| 3 | Icecream | 4 | 860 | | |
| 4 | Twinkies | 3 | 350 | | |
| 5 | Cake | 6 | 133 | | |
| 6 | Donuts | 1 | 458 | | |
| 7 | | | | | |
| 8 | | | | | |
| 9 | | | | | |
| 10 | | | | | |

The image shows a screenshot of the Microsoft Excel application. The top ribbon is set to 'Home', displaying various editing and formatting options. Below the ribbon, the formula bar shows the active cell (D7) containing the formula `=sum(`. The spreadsheet grid shows a table with columns A through F and rows 1 through 11. The data in the table is as follows:

| | A | B | C | D | E | F |
|----|----------|-------|----------|--------------------|---|---|
| 1 | | price | Quantity | | | |
| 2 | Cookies | 2 | 215 | 430 | | |
| 3 | Icecream | 4 | 860 | 3440 | | |
| 4 | Twinkies | 3 | 350 | 1050 | | |
| 5 | Cake | 6 | 133 | 798 | | |
| 6 | Donuts | 1 | 458 | 458 | | |
| 7 | | | | <code>=sum(</code> | | |
| 8 | | | | | | |
| 9 | | | | | | |
| 10 | | | | | | |
| 11 | | | | | | |

A tooltip for the SUM function is visible, showing the syntax: `SUM(number1, [number2], ...)`. The cell D2 is also highlighted with a 'D2' label.

Direct Manipulation Properties

1. **Objects** are represented visually



Move to trash



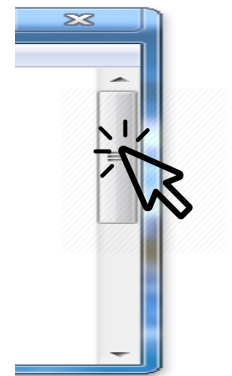
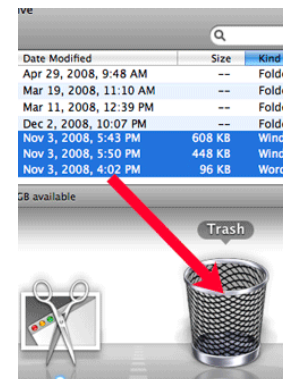
Resize



Move viewport

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

3. User interacts
directly with object representations



Is this direct manipulation?

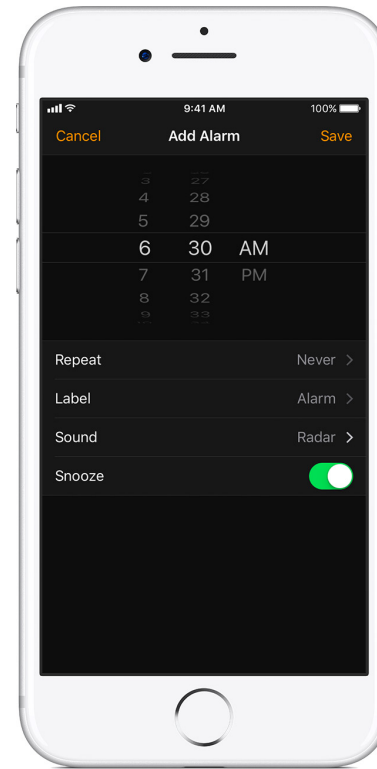


Is this direct manipulation?

Not direct manipulation



Direct manipulation



Why can Direct Manipulation be good?

1. **Objects** are represented visually



Move to trash



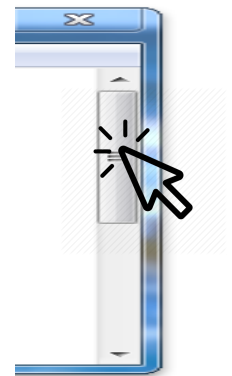
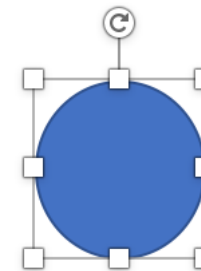
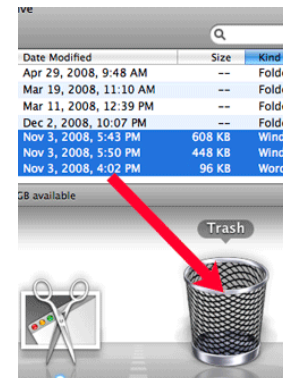
Resize



Move viewport

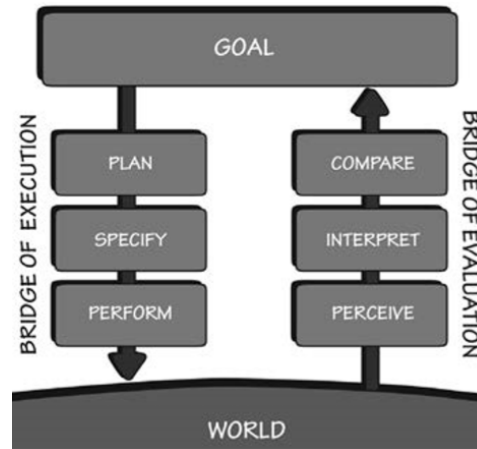
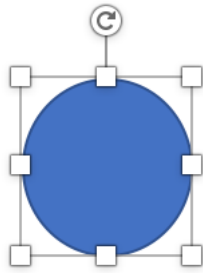
2. **Actions** are rapid,
incremental and reversible

3. User interacts
directly with object representations

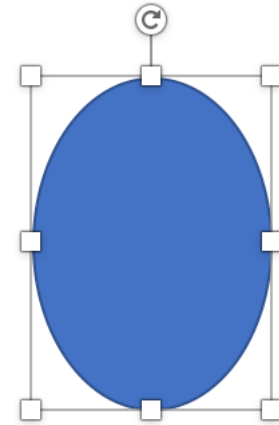


Why can Direct Manipulation be good?

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



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

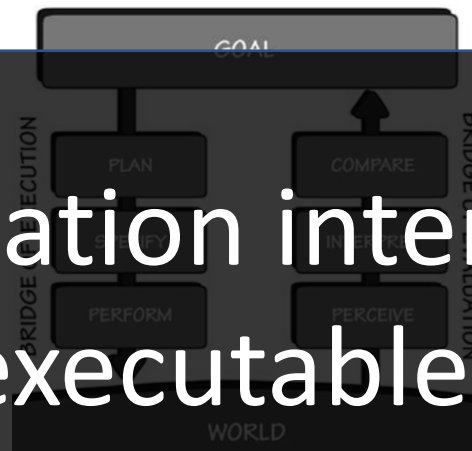


Why can Direct Manipulation be good?

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

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

Direct manipulation interfaces are usable
because they have executable actions and evaluate-
able feedback to achieve a goal



Affordances

Affordance: What can you do 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

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 your weight onto
And push

Feedback

Pushing it depresses the handle

Affordance

Push

Affordance: What should do you with this?



Perceived Affordance

Put paper in it

Signifier

Paper sized hole

Feedback

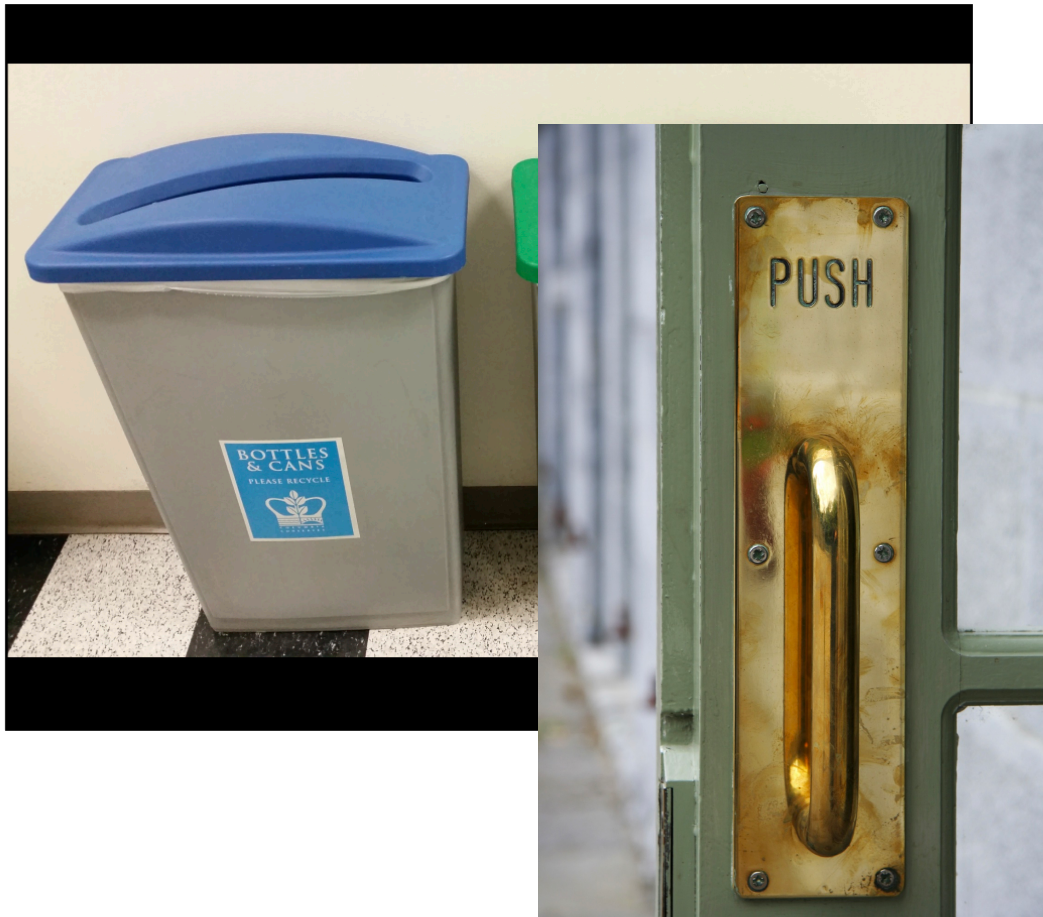
None.

Affordance

Bottles and cans

Design direct manipulation interfaces by signaling affordances with signifiers users can perceive

Bad signifiers / wrong perceived affordances

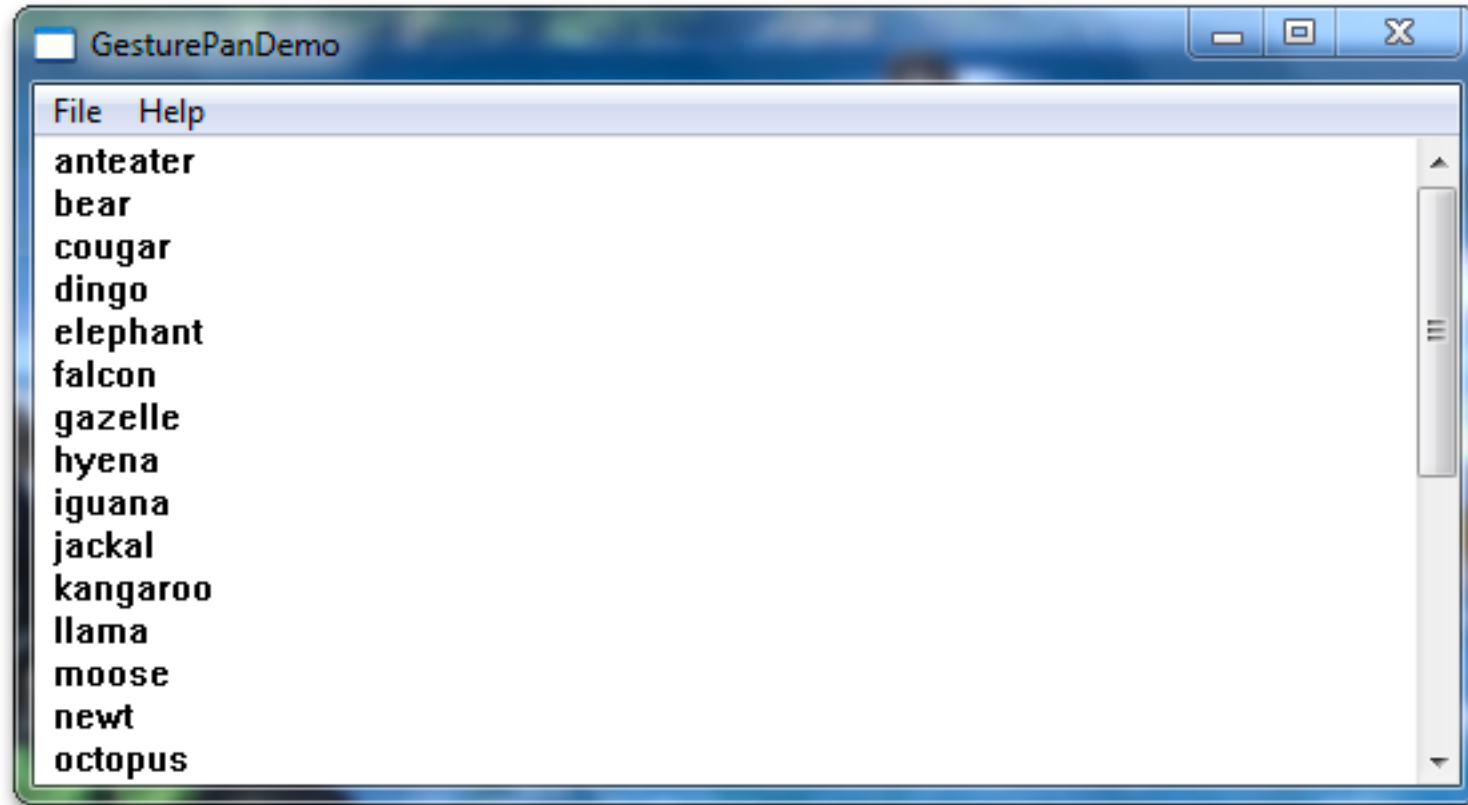


Good signifiers / correct perceived affordances

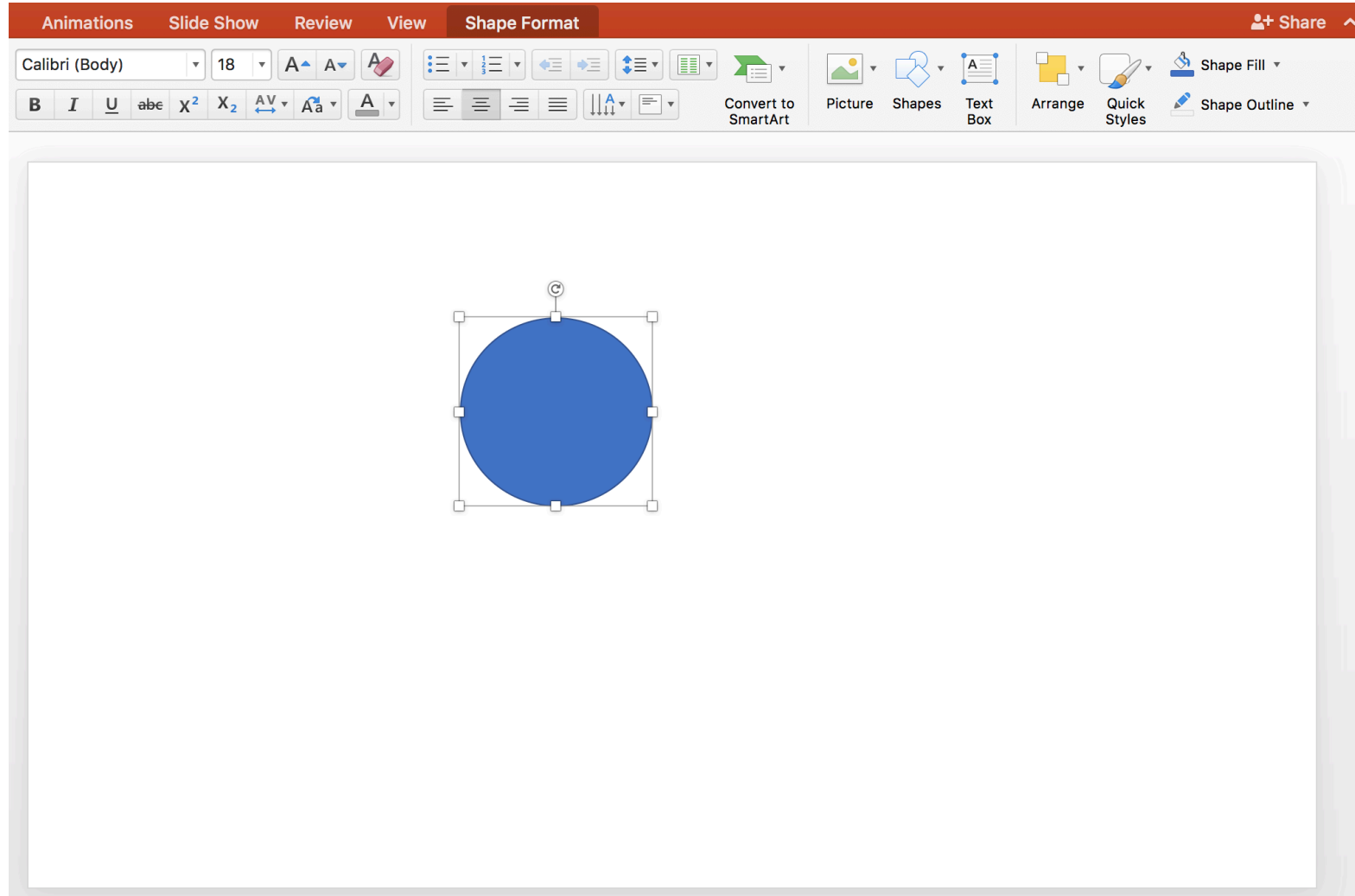


What signifiers do these UIs use to signal affordances?

What are the signifiers of affordances?



What are the signifiers of affordances?

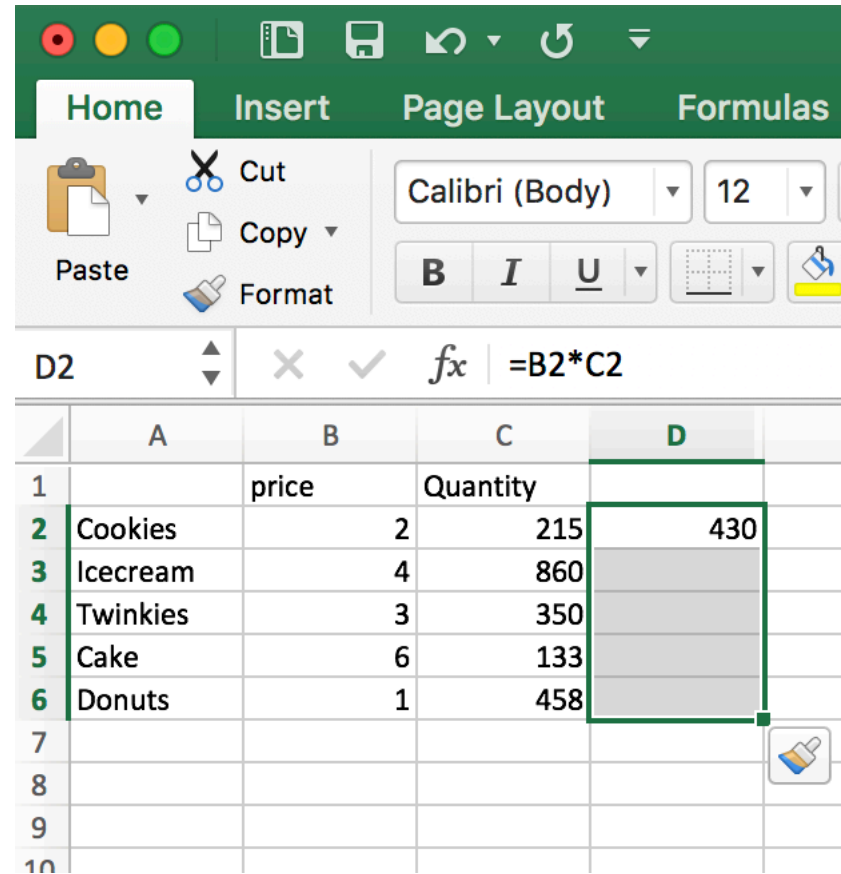


What are the signifiers of affordances?

The image shows a screenshot of the Microsoft Excel interface. The ribbon is set to 'Home', and the formula bar shows the start of a SUM formula: '=sum('. A tooltip for the SUM function is displayed, showing the syntax: `SUM(number1, [number2], ...)`. The spreadsheet data is as follows:

| | A | B | C | D | E | F |
|----|----------|-------|----------|-------|---|---|
| 1 | | price | Quantity | | | |
| 2 | Cookies | 2 | 215 | 430 | | |
| 3 | Icecream | 4 | 860 | 3440 | | |
| 4 | Twinkies | 3 | 350 | 1050 | | |
| 5 | Cake | 6 | 133 | 798 | | |
| 6 | Donuts | 1 | 458 | 458 | | |
| 7 | | | | =sum(| | |
| 8 | | | | | | |
| 9 | | | | | | |
| 10 | | | | | | |
| 11 | | | | | | |

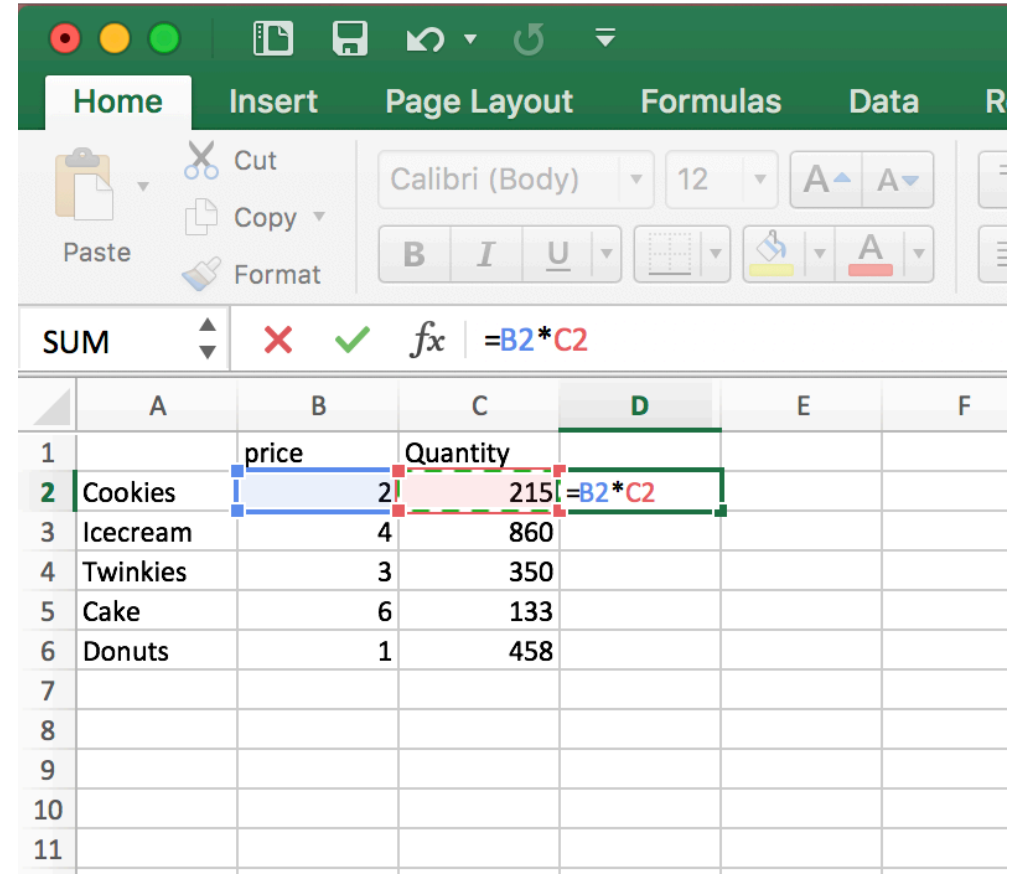
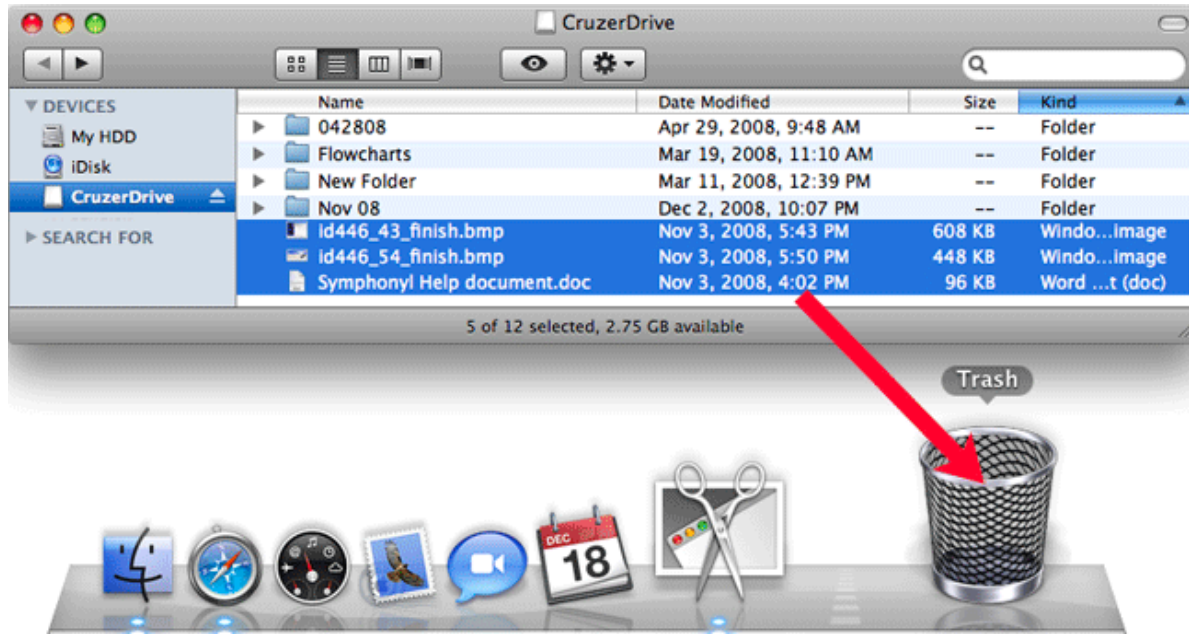
What are the signifiers of affordances?



The image shows a screenshot of the Microsoft Excel application. The ribbon is set to 'Home', and the 'Paste' dropdown menu is open. The formula bar shows the formula $=B2*C2$ in cell D2. The spreadsheet contains the following data:

| | A | B | C | D |
|----|----------|-------|----------|-----|
| 1 | | price | Quantity | |
| 2 | Cookies | 2 | 215 | 430 |
| 3 | Icecream | 4 | 860 | |
| 4 | Twinkies | 3 | 350 | |
| 5 | Cake | 6 | 133 | |
| 6 | Donuts | 1 | 458 | |
| 7 | | | | |
| 8 | | | | |
| 9 | | | | |
| 10 | | | | |

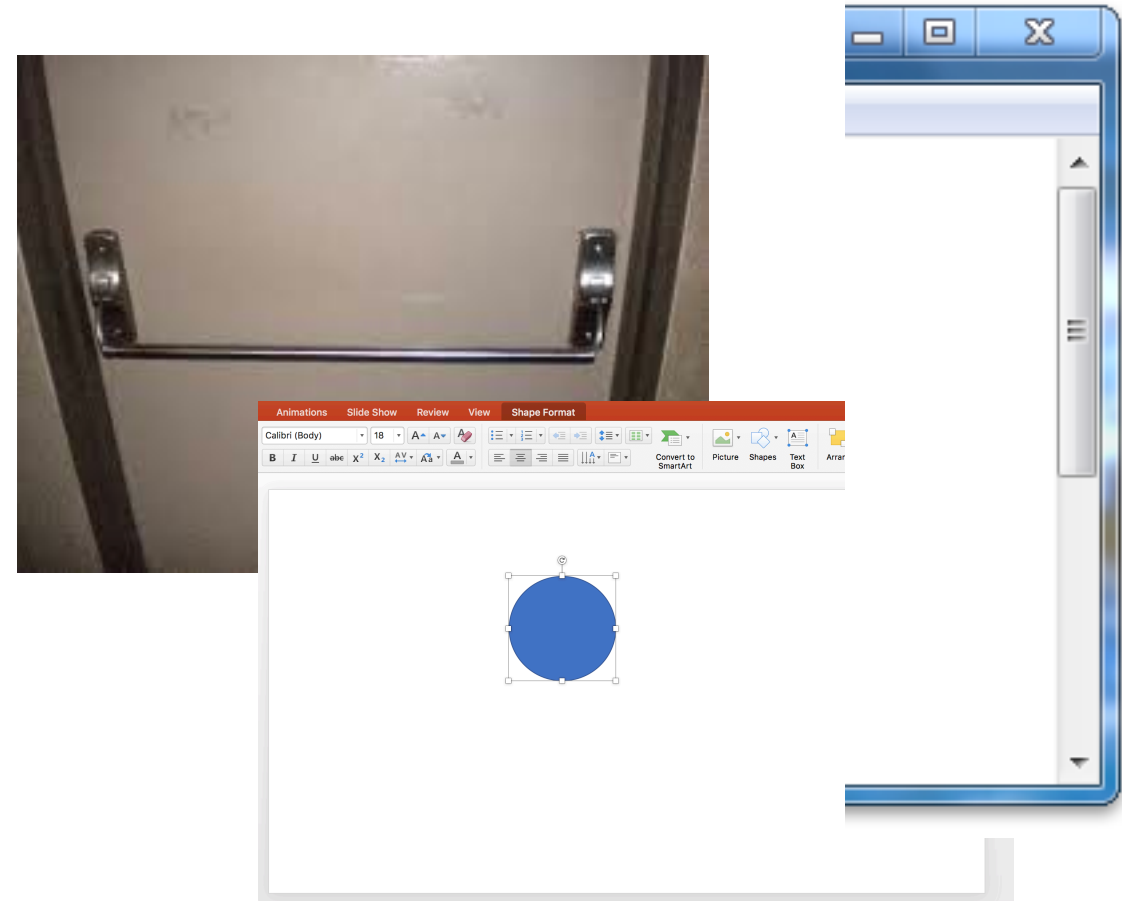
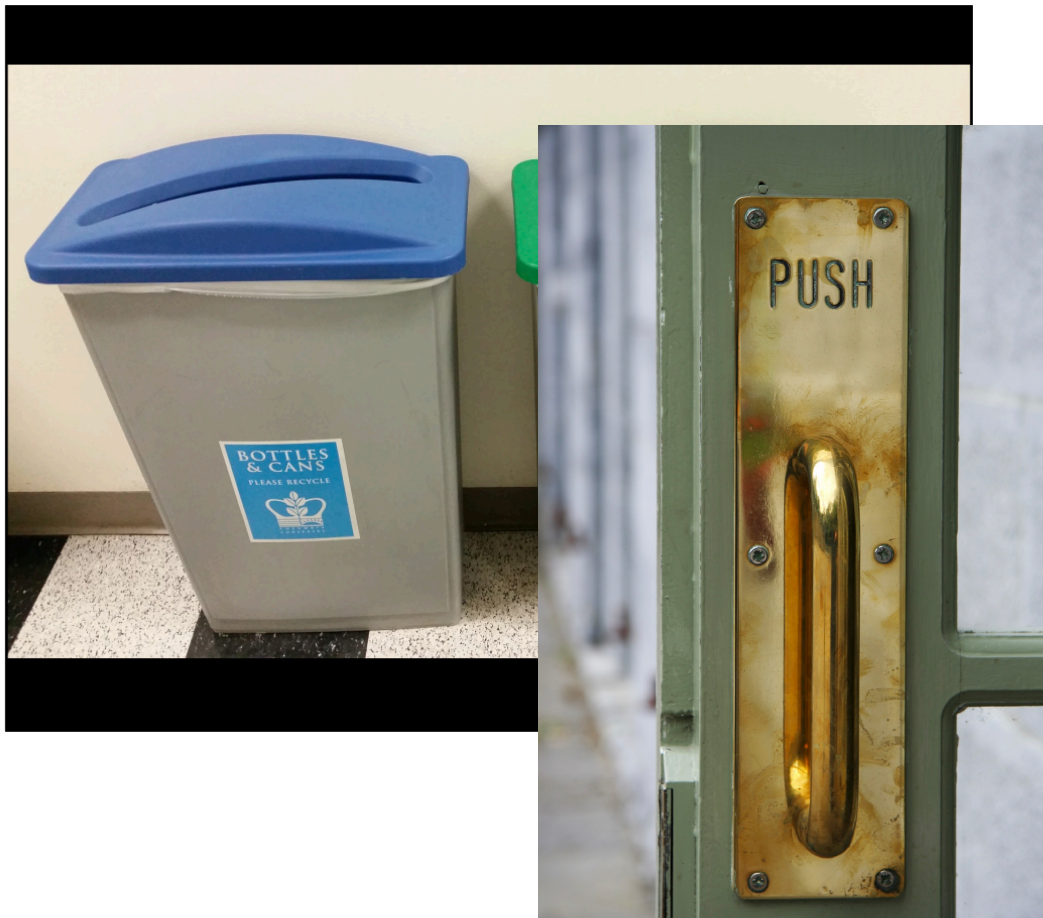
What are the signifiers of affordances?



Design direct manipulation interfaces by signaling affordances with signifiers users can perceive

Bad signifiers / wrong perceived affordances

Good signifiers / correct perceived affordances



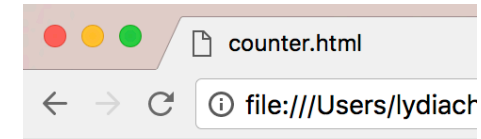
Implementing Direct Manipulation Interfaces

Direct Manipulation Properties

1. **Objects** are represented visually

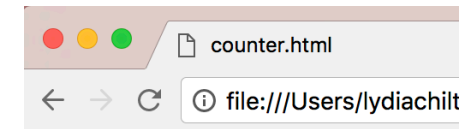
2. **Actions** are rapid,
incremental and reversible

3. User interacts
directly with object representations



Counter (0)

Click

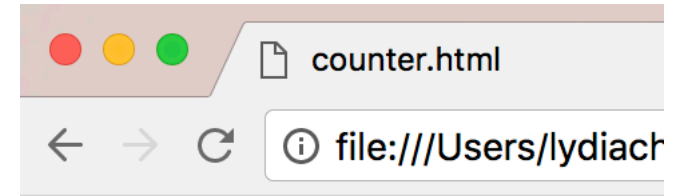


Counter (1)

What will this do when you click it?

HTML

```
30  
31 <body>  
32   <button id="counter" class="btn btn-primary">Counter (0)</button>  
33  
34 </body>  
35  
36
```

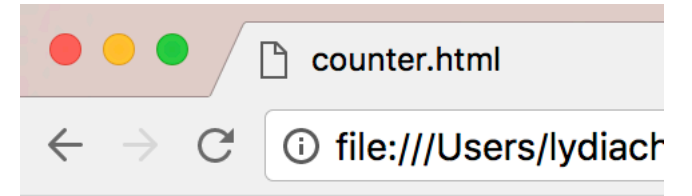


Counter (0)

Add Click events with JQuery

HTML

```
30
31 <body>
32
33   <button id="counter" class="btn btn-primary">Counter (0)</button>
34
35 </body>
36
```



Counter (0)

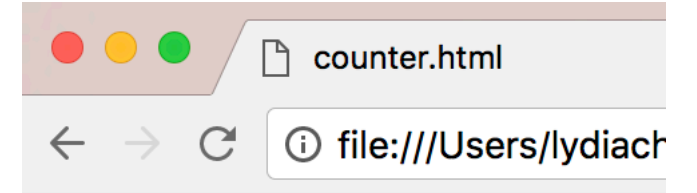
JavaScript

```
14
15 <head>
16   <!-- bootstrap -->
17   <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css">
18   <!-- JQuery -->
19   <script src="http://code.jquery.com/jquery-3.3.1.min.js"></script>
20   <script src="http://code.jquery.com/ui/1.11.4/jquery-ui.min.js"></script>
21 </head>
22
```

How do we attach an action to the button?

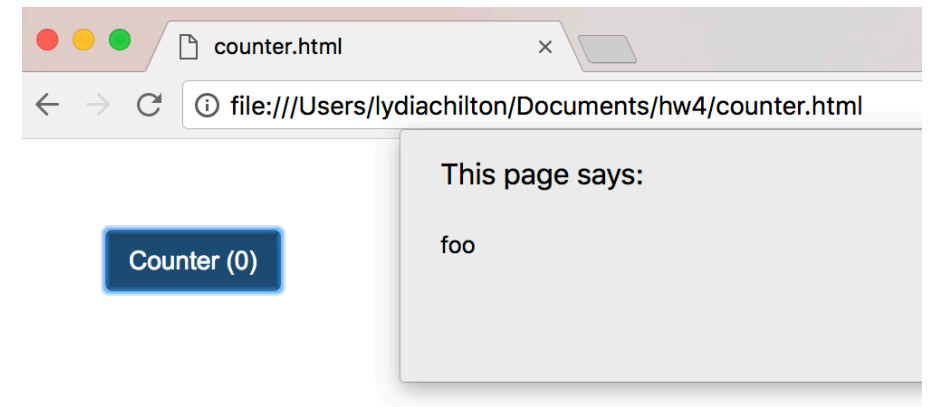
HTML

```
30  
31 <body>  
32  
33   <button id="counter" class="btn btn-primary">Counter (0)</button>  
34  
35 </body>  
36
```



JavaScript

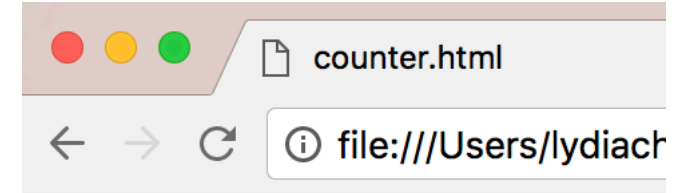
```
25  
26 $(document).ready(function(){  
27   $("#counter").click(function(){  
28     alert("foo")  
29   })  
30 })  
31
```



How do we increment the counter????

HTML

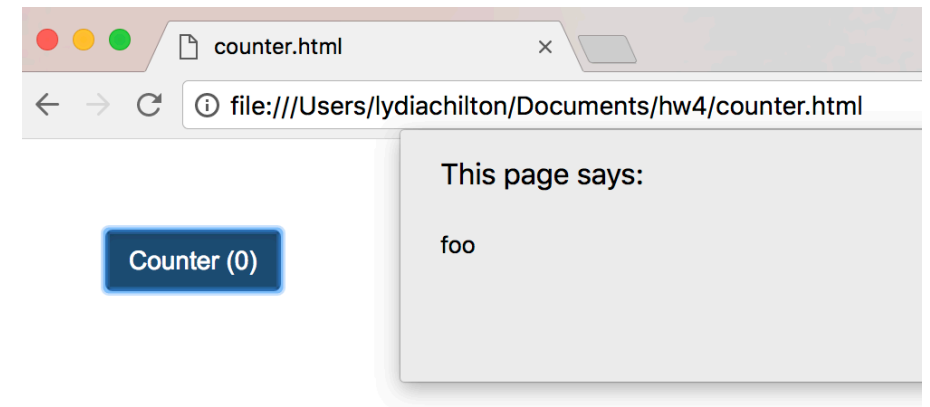
```
30  
31 <body>  
32  
33   <button id="counter" class="btn btn-primary">Counter (0)</button>  
34  
35 </body>  
36
```



Counter (0)

JavaScript

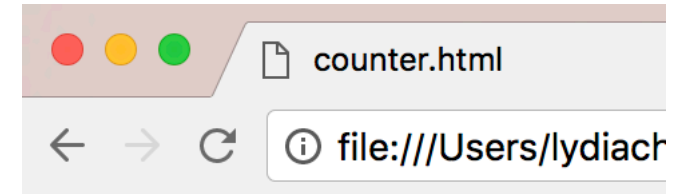
```
25  
26 $(document).ready(function(){  
27   $("#counter").click(function(){  
28     alert("foo")  
29   })  
30 })  
31
```



How NOT to increment the count?

HTML

```
30  
31 <body>  
32  
33 <button id="counter" class="btn btn-primary">Counter (0)</button>  
34  
35 </body>  
36
```



Counter (0)

JavaScript

```
25  
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
```

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

Extract the data from from the text

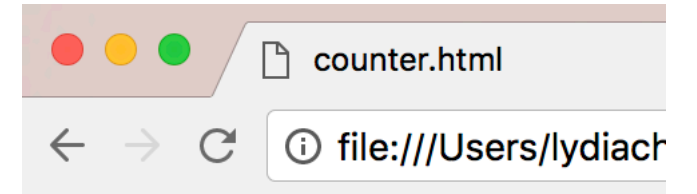
Cast to a number and add one

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

How TO increment the count?

HTML

```
61 <body>
62
63   <button id="counter" class="btn btn-primary"></button>
64
65 </body>
66
```



Counter (0)

JavaScript

```
41
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 })
56
```

Create a model of the data separate from the HTML

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

Bad JavaScript

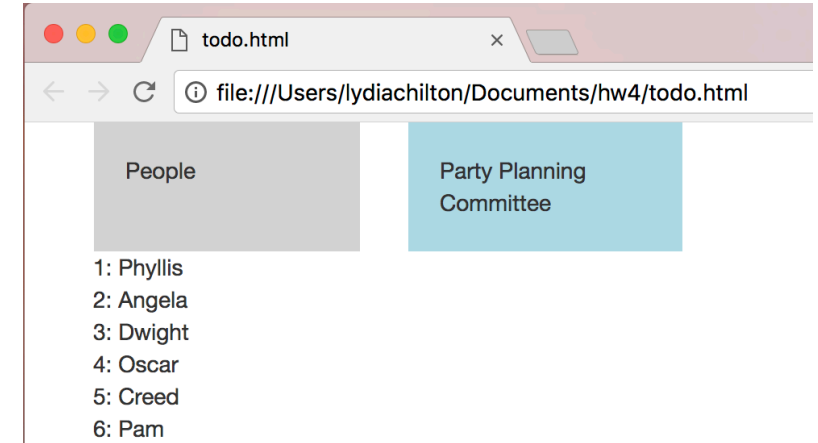
```
25
26 $(document).ready(function(){
27     $("#counter").click(function(){
28         var html = $(this).html()
29
30         var regexp = /\(([^\)]+)\)/;
31         var matches = regexp.exec(html);
32
33         var number = 1*matches[1]
34         var incremented_number = number +1
35
36         $(this).html("Counter (" + incremented_number + ")")
37     })
38 })
39
```

Good JavaScript

```
41
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 })
56
```

Direct Manipulation Properties

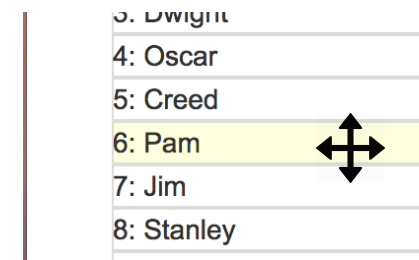
1. **Objects** are represented visually



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

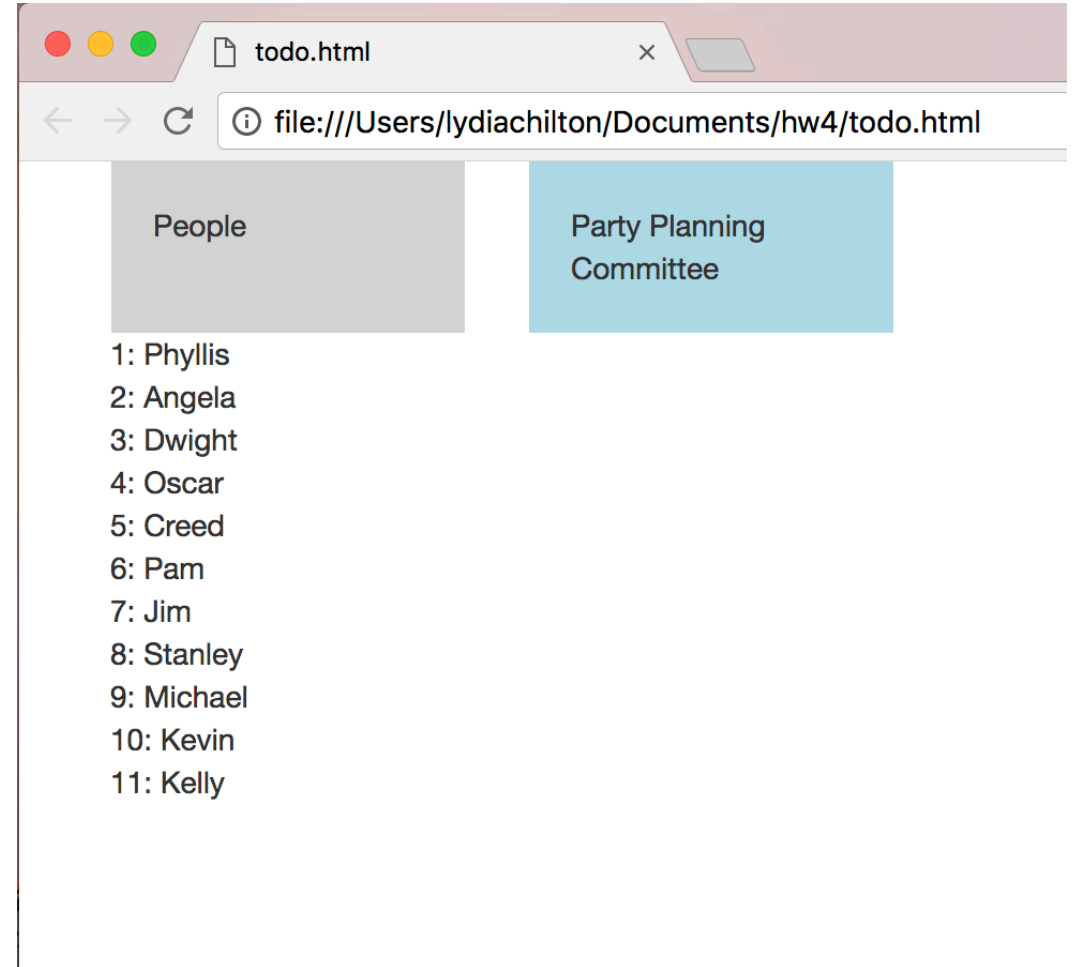
Drag and Drop

3. User interacts
directly with object representations



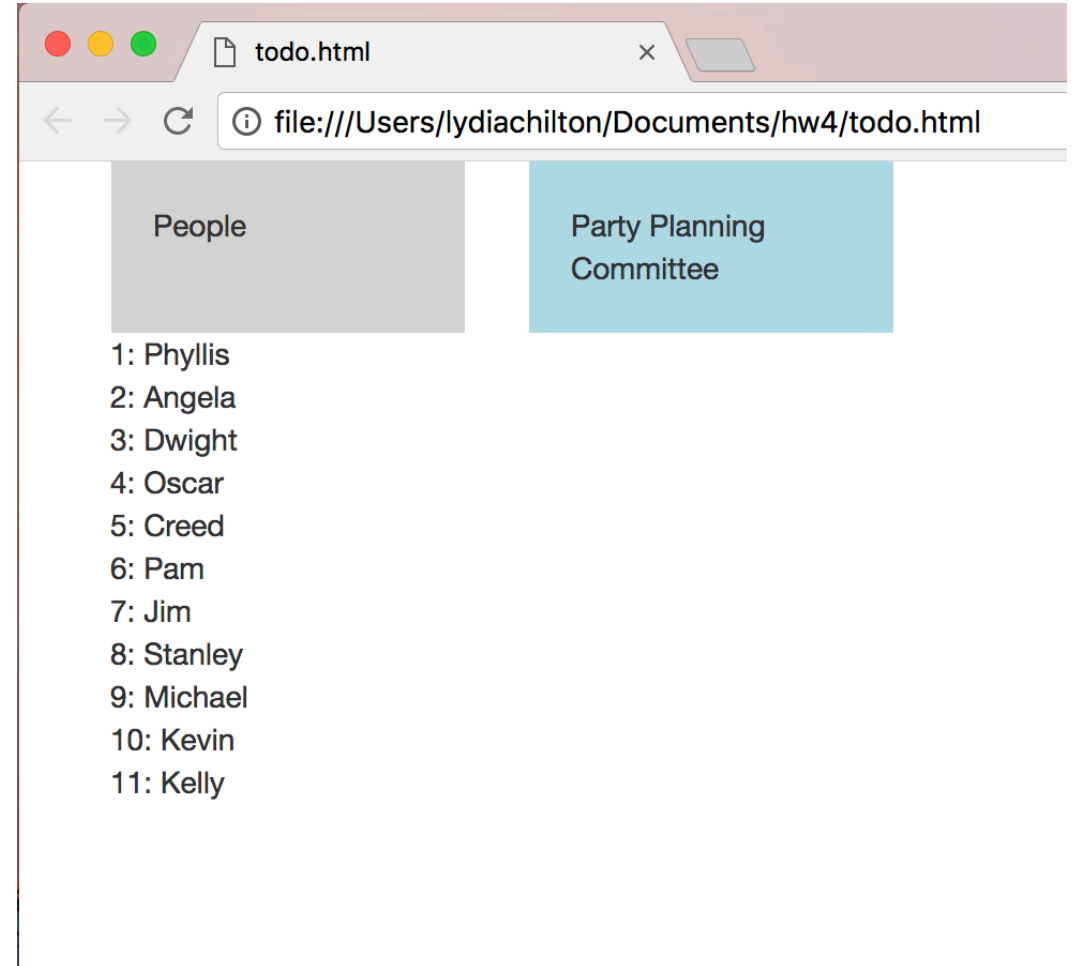
How to NOT implement this?

```
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
```



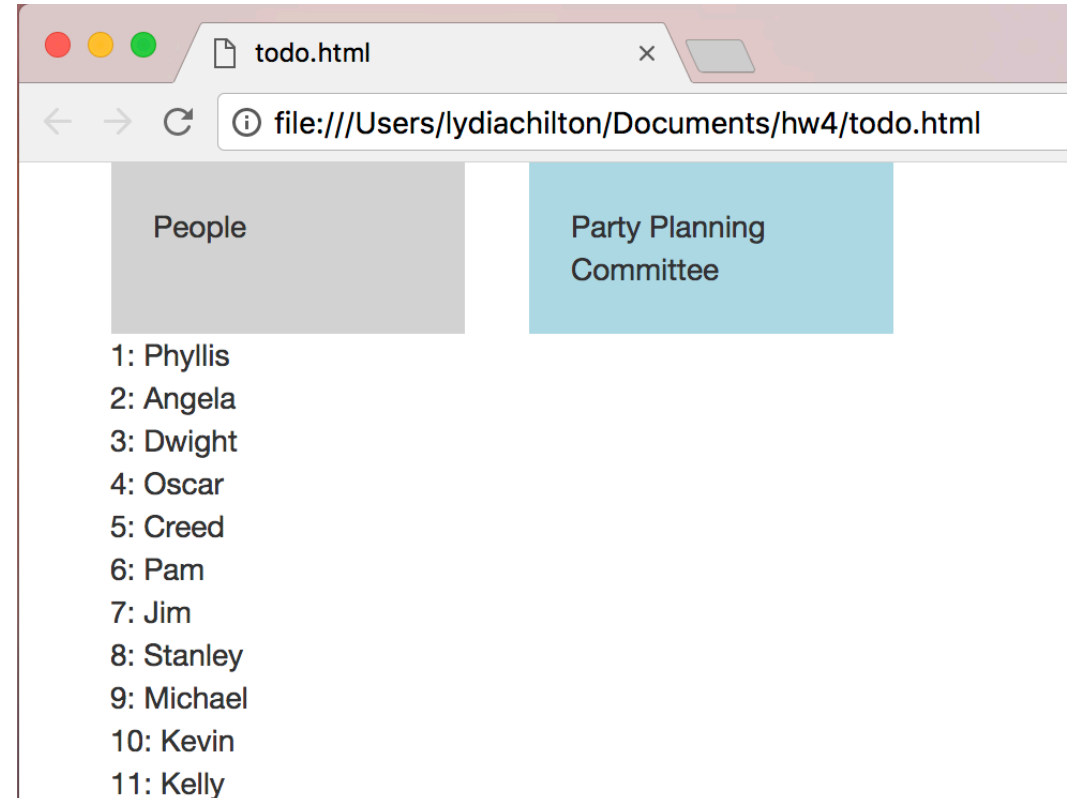
Step 1. Create the Data Model

```
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
```



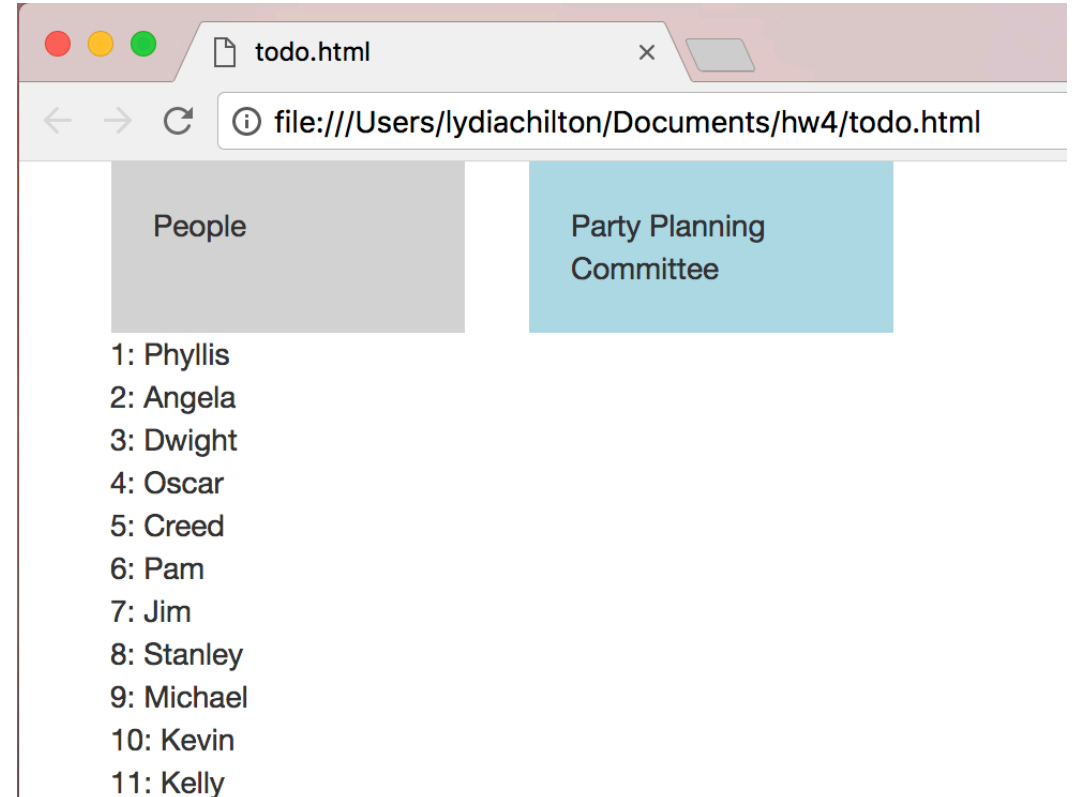
Step 2. Create a function that updates the view with new data

```
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
```



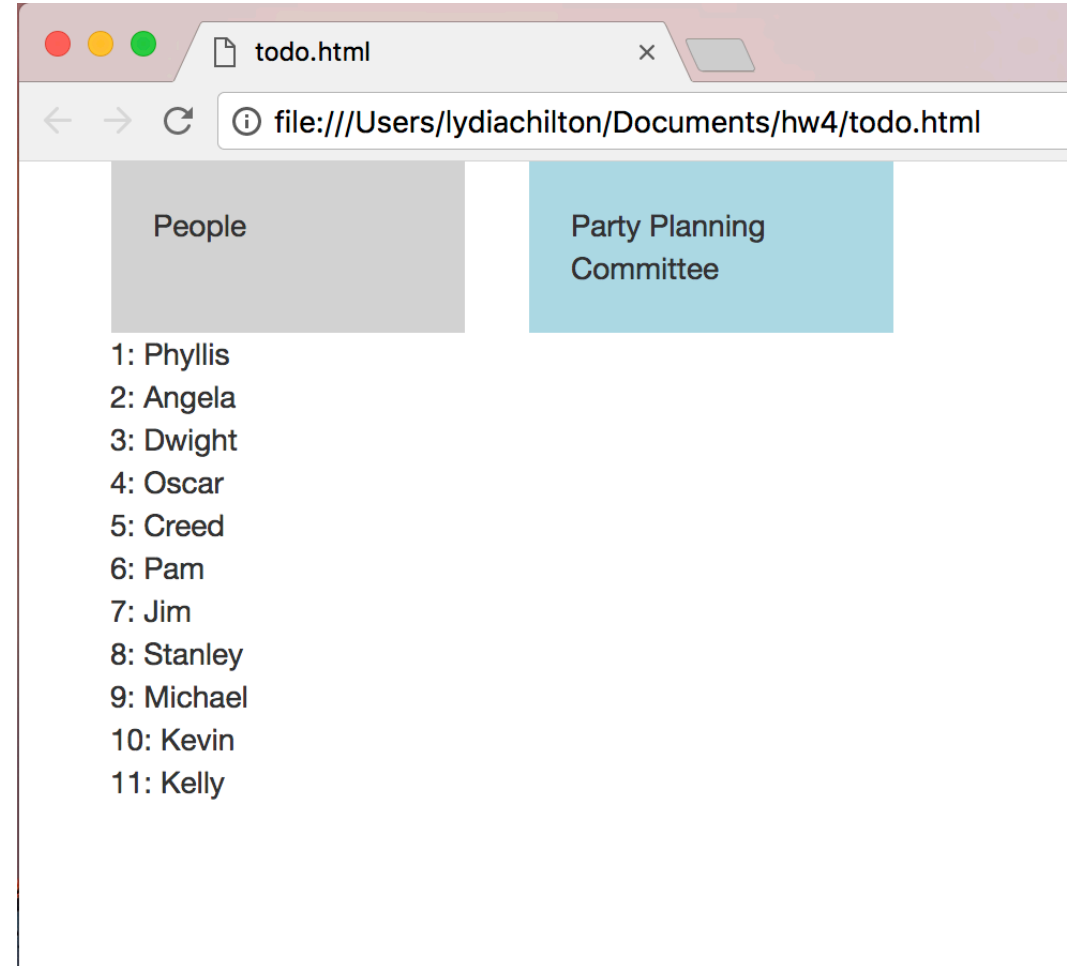
Step 3. On page load, create view

```
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 event to the object.
It should update the data, then update the view

```
188
189
190 $(document).ready(function(){
191     makeNames(names)
192
193     $("#team1_label").droppable({
194         drop: function(event, ui) {
195             //get dropped name
196
197             //update names array
198
199             //update list1 array
200
201             //update the interface to display the new lists
202         }
203     });
204 })
205
```




```

41
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 })
56

```

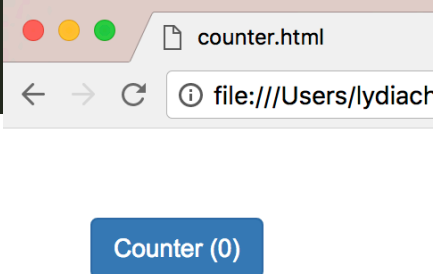
```

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
193     $( "#team1_label" ).droppable({
194         drop: function( event, ui ) {
195             //get dropped name
196
197             //update names array
198
199             //update list1 array
200
201             //update the interface to display the new lists
202         }
203     });
204 })
205
206

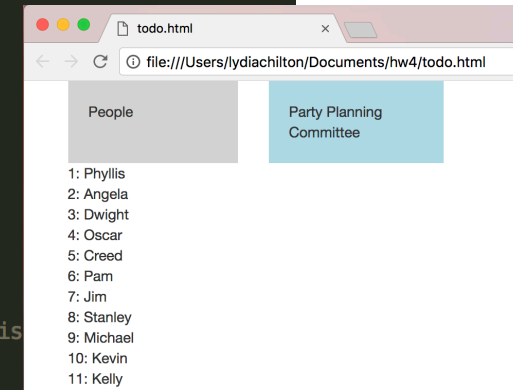
```

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

```
41
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 })
56
```

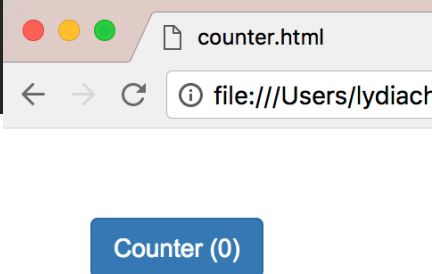


```
170
171 var names = [
172     "Phyllis",
173     "Angela",
174     "Dwight",
175     "Oscar",
176     "Creed",
177     "Pam"
178 ]
179 var list1 = []
180
181
182 function makeNames(names){
183     $("#names").empty()
184     $.each(names, function( index, value ) {
185         //make the draggable name object
186     });
187 }
188
189
190 $(document).ready(function(){
191     makeNames(names)
192
193     $( "#team1_label" ).droppable({
194         drop: function( event, ui ) {
195             //get dropped name
196
197             //update names array
198
199             //update list1 array
200
201             //update the interface to dis
202         }
203     });
204 })
205
206
```

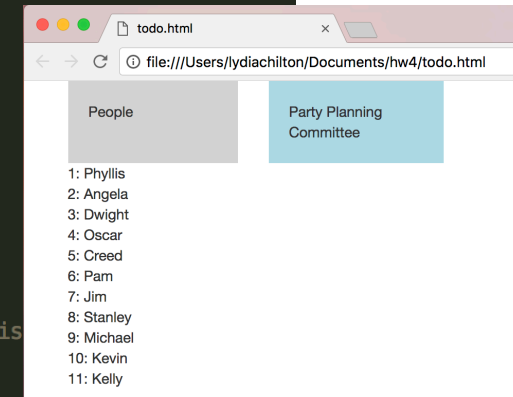


Good UI programming separates the **data** model from the **view** and **controller**

```
41
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 })
56
```

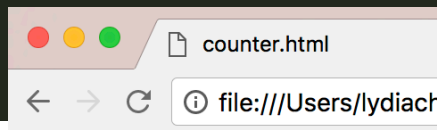


```
170
171 var names = [
172     "Phyllis",
173     "Angela",
174     "Dwight",
175     "Oscar",
176     "Creed",
177     "Pam"
178 ]
179 var list1 = []
180
181
182 function makeNames(names){
183     $("#names").empty()
184     $.each(names, function( index, value ) {
185         //make the draggable name object
186     });
187 }
188
189
190 $(document).ready(function(){
191     makeNames(names)
192
193     $( "#team1_label" ).droppable({
194         drop: function( event, ui ) {
195             //get dropped name
196
197             //update names array
198
199             //update list1 array
200
201             //update the interface to dis
202         }
203     });
204 })
205
206
```



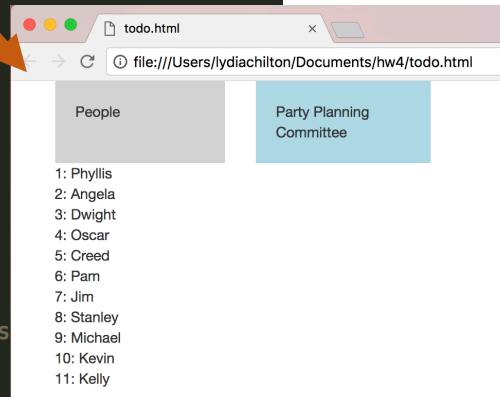
Good UI programming separates the **data** model from the **view** and **controller**

```
41
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 })
56
```



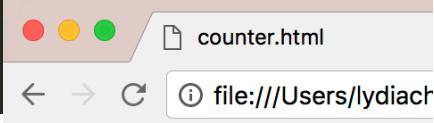
Counter (0)

```
170
171 var names = [
172     "Phyllis",
173     "Angela",
174     "Dwight",
175     "Oscar",
176     "Creed",
177     "Pam"
178 ]
179 var list1 = []
180
181 function makeNames(names){
182     $("#names").empty()
183     $.each(names, function(index, value) {
184         //make the draggable name object
185     });
186 }
187
188
189
190 $(document).ready(function(){
191     makeNames(names)
192
193     $("#team1_label").droppable({
194         drop: function(event, ui) {
195             //get dropped name
196
197             //update names array
198
199             //update list1 array
200
201             //update the interface to dis
202         }
203     });
204 }
205
206
```



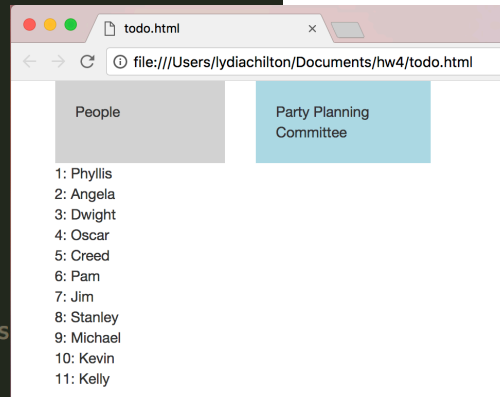
Good UI programming separates the **data** model from the **view** and **controller**

```
41
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 })
56
```



Counter (0)

```
170
171 var names = [
172     "Phyllis",
173     "Angela",
174     "Dwight",
175     "Oscar",
176     "Creed",
177     "Pam"
178 ]
179 var list1 = []
180
181 function makeNames(names){
182     $("#names").empty()
183     $.each(names, function( index, value ) {
184         //make the draggable name object
185     });
186 }
187
188
189
190 $(document).ready(function(){
191     makeNames(names)
192
193     $("#team1_label").droppable({
194         drop: function( event, ui ) {
195             //get dropped name
196
197             //update names array
198
199             //update list1 array
200
201             //update the interface to dis
202         }
203     });
204 }
205
206
```



Summary

Direct Manipulation Properties

1. **Objects** are represented visually



Move to trash



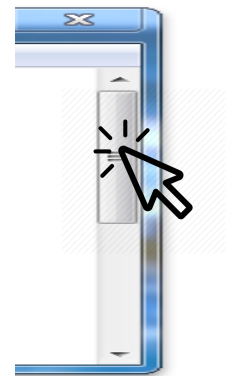
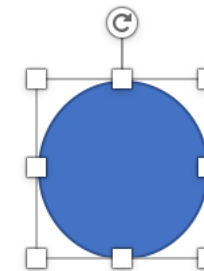
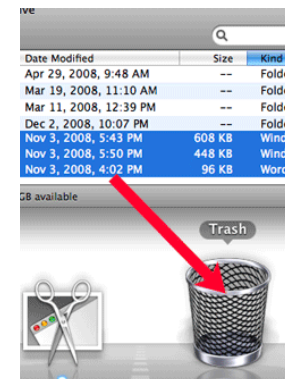
Resize



Move viewport

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

3. User interacts
directly with object representations



Signal affordances with signifiers the user can perceive

NOT this



Signifier Handle that can be yanked toward you

Perceived affordance **Pull**

Affordance **Push**

THIS



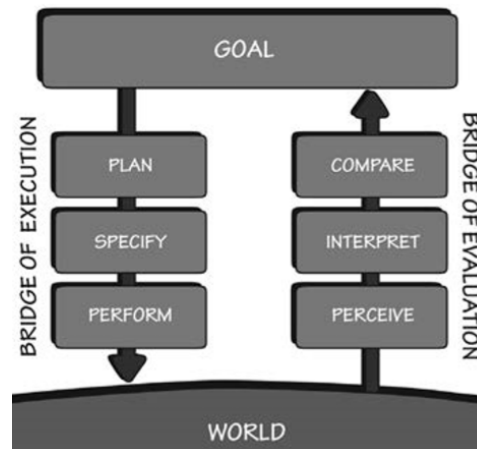
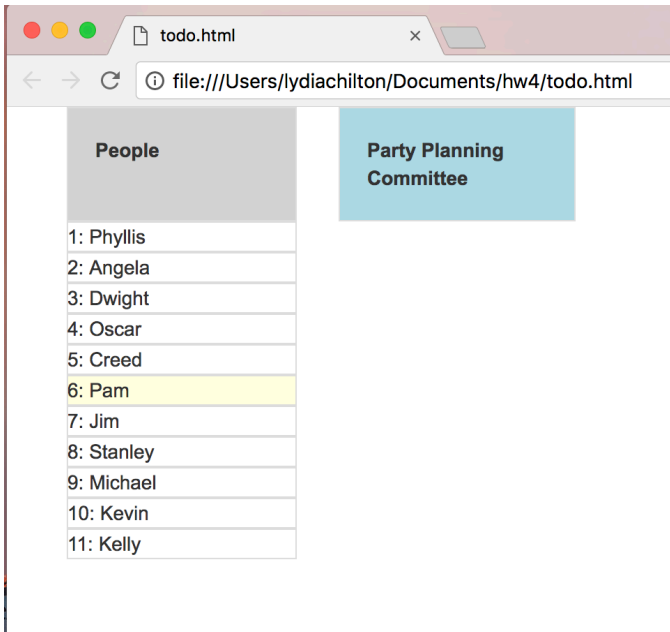
Signifier Handle that can be leaned on

Perceived affordance **Push**

Affordance **Push**

Direct manipulation interfaces are usable because they suit the 7 stages of action

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

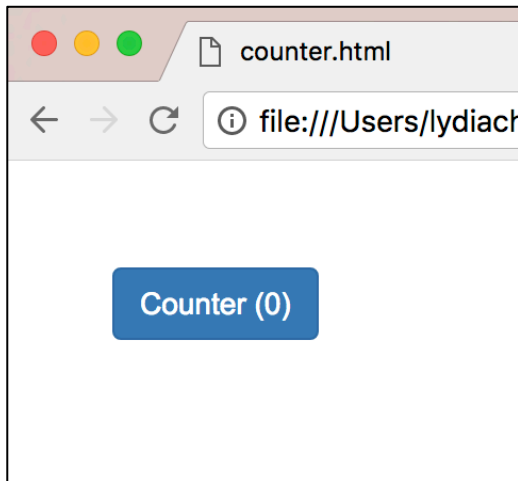


There is visible **feedback**
the user can **evaluate**

| People | Party Planning Committee |
|------------|--------------------------|
| 1: Angela | 1: Phyllis |
| 2: Dwight | |
| 3: Oscar | |
| 4: Creed | |
| 5: Pam | |
| 6: Jim | |
| 7: Stanley | |
| 8: Michael | |
| 9: Kevin | |
| 10: Kelly | |

When implementing Direct Manipulation:

Create an **object**
in the view



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

```
25
26 $(document).ready(function(){
27     $("#counter").click(function(){
28         alert("foo")
29     })
30 })
31
```

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

```
41
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 })
56
```

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