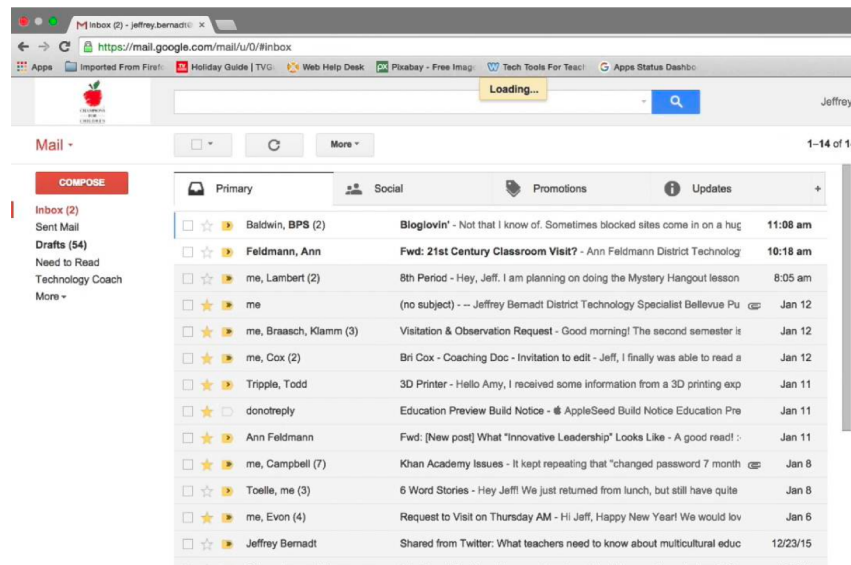




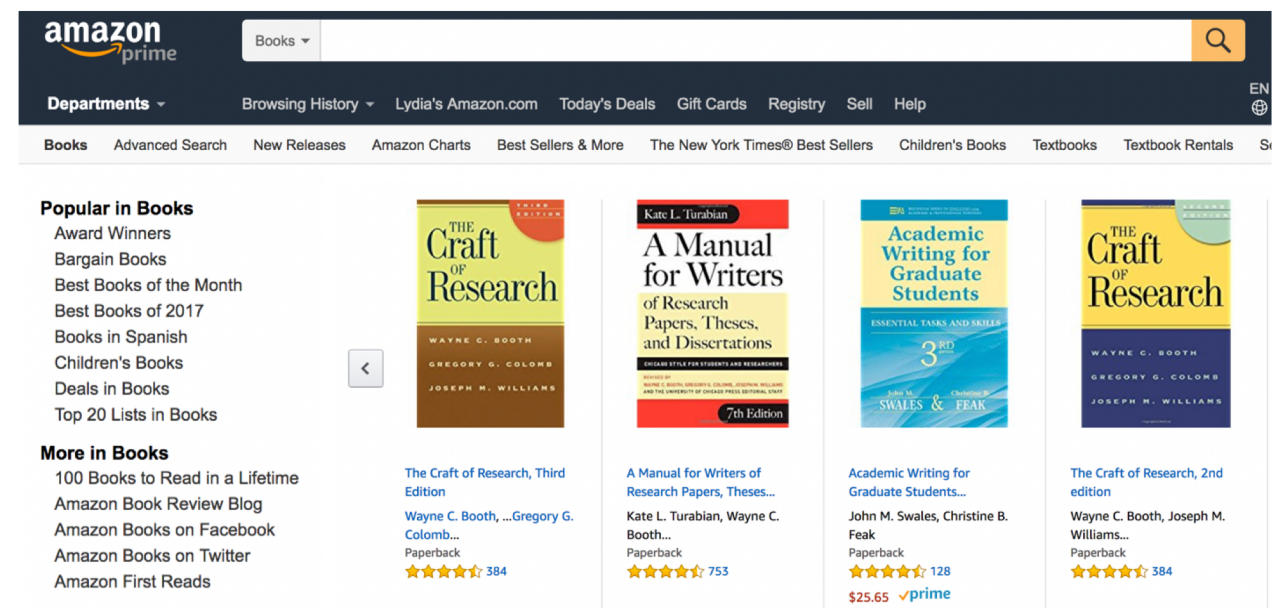
Direct Manipulation

Prof. Lydia Chilton
COMS 4170
7 February 2024

Users interact with a system to accomplish a goal

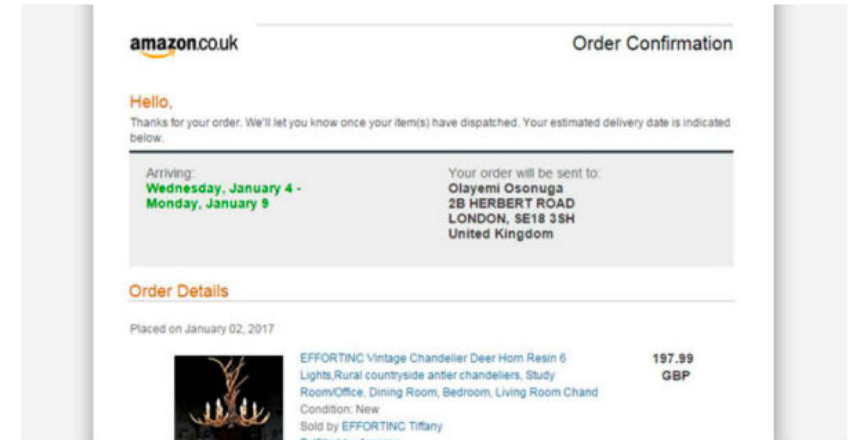
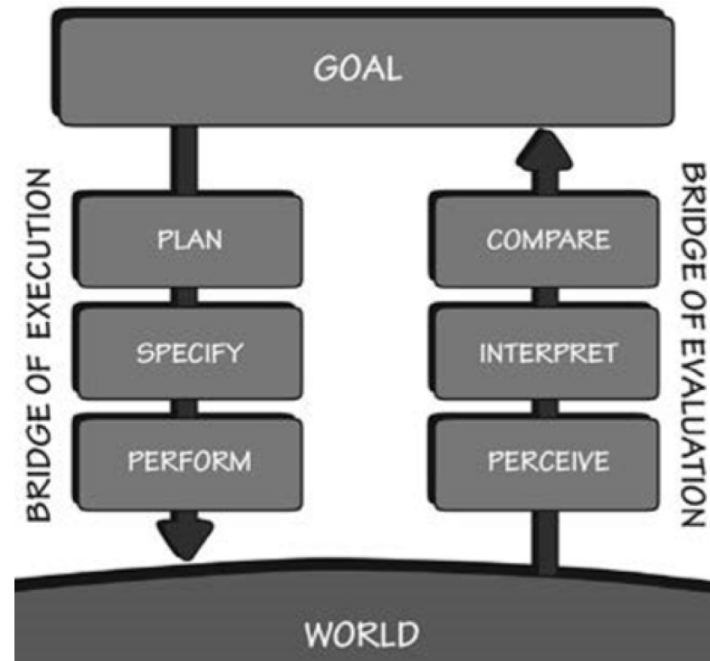
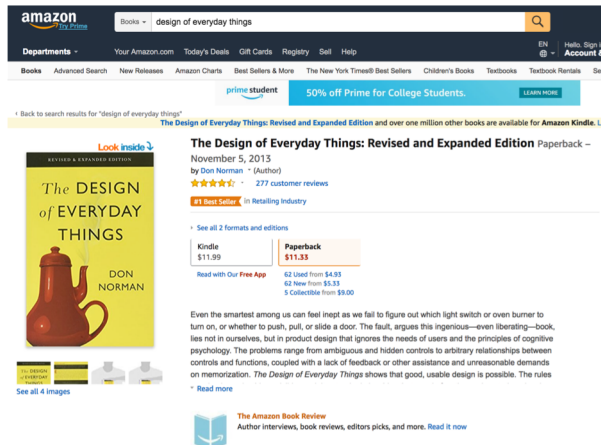


To read and respond to all email.



To buy a book

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

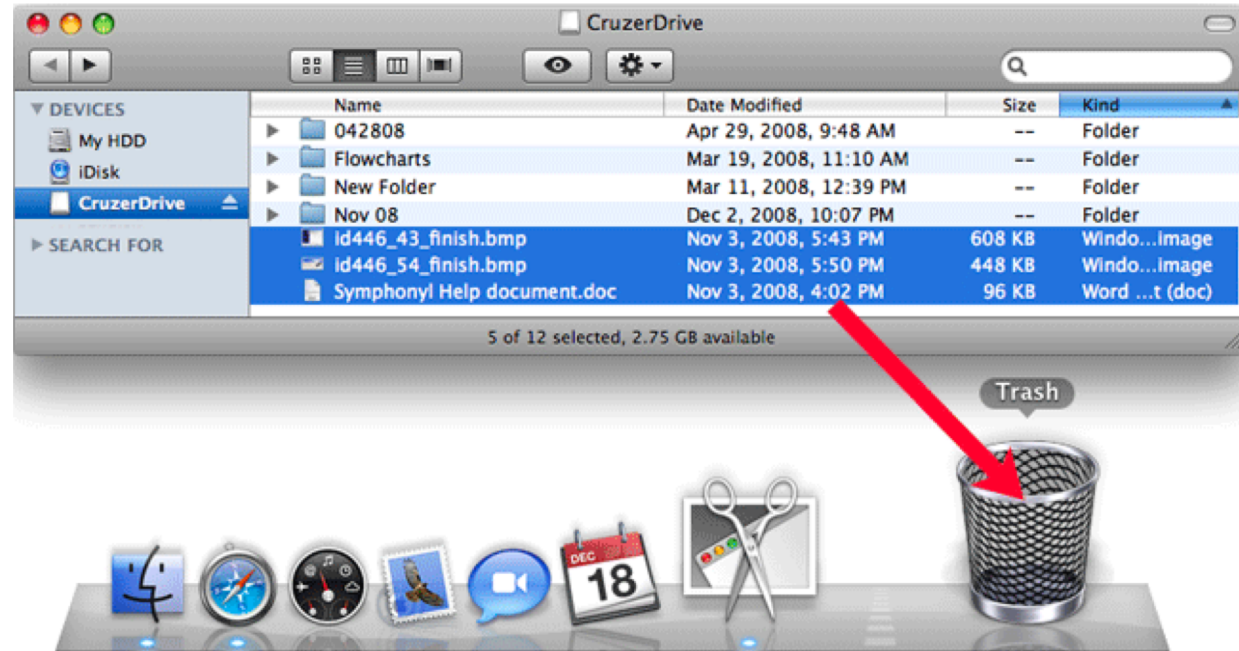


The designer must know the users' goals
and create interactions to help them execute and evaluate it.

Then:
Textual commands

```
Last login: Fri May 25 17:23:20 on ttys000
Mac:~ usman$ rm /Users/usman/Desktop/test\ image.jpg
Mac:~ usman$
```

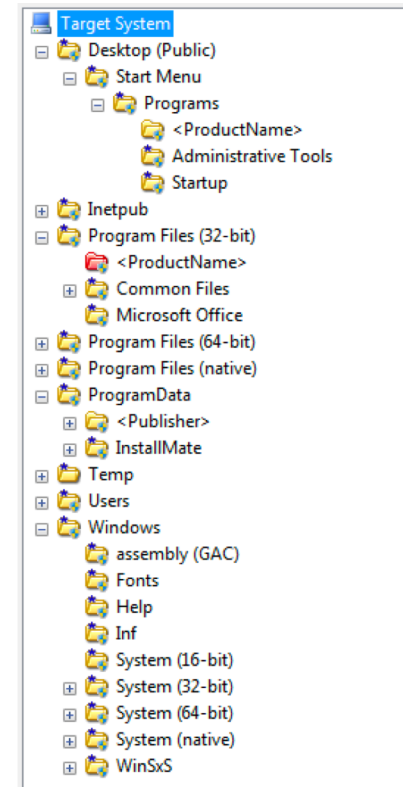
Now:
Graphical User Interfaces



Graphical interfaces have a lot of advantages

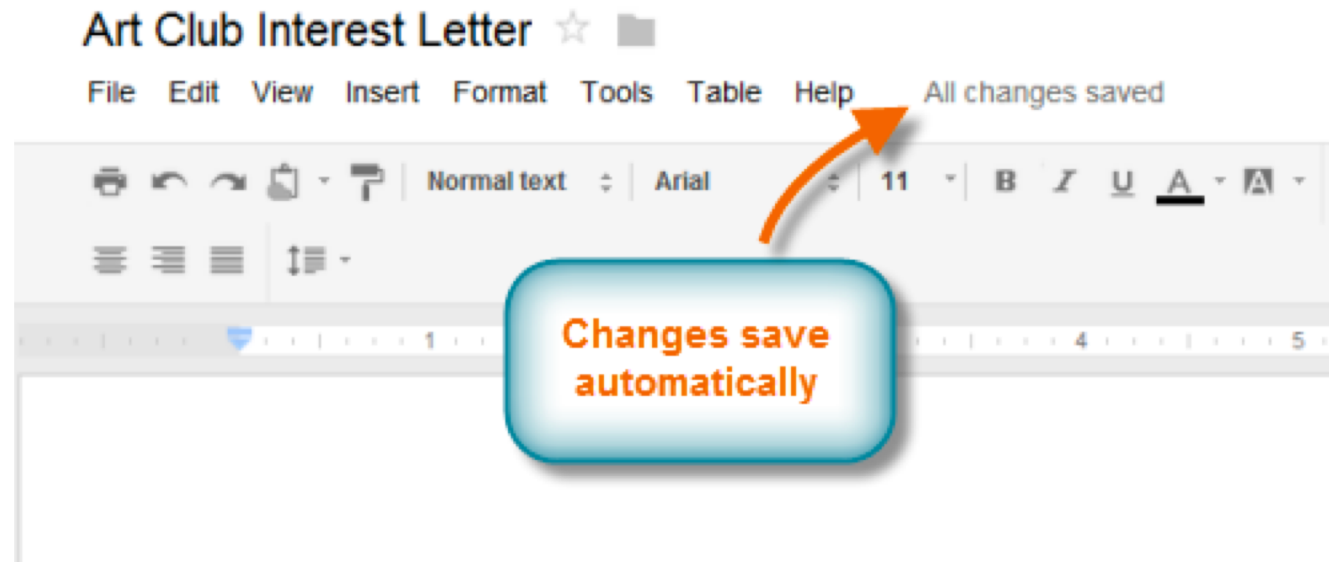
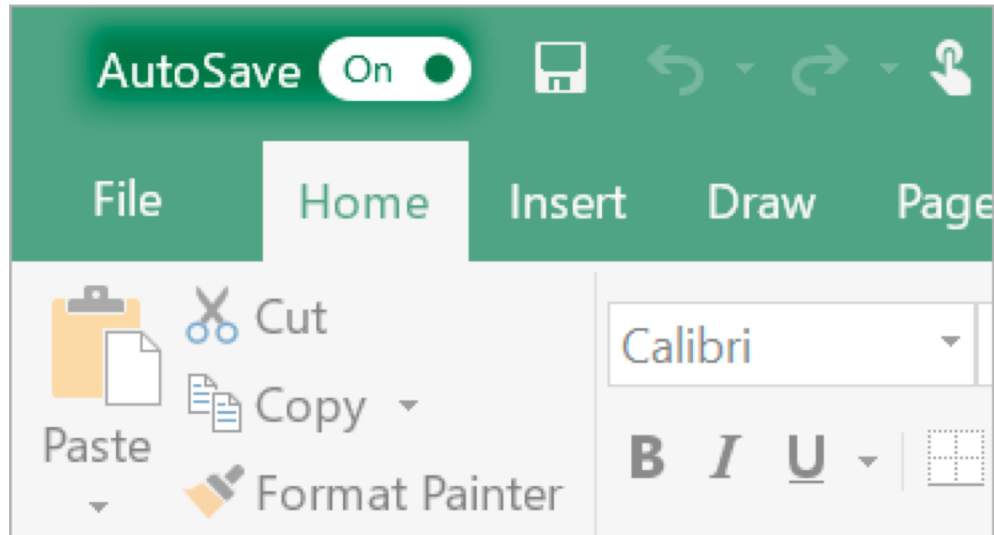
2. Match between system and the real world

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



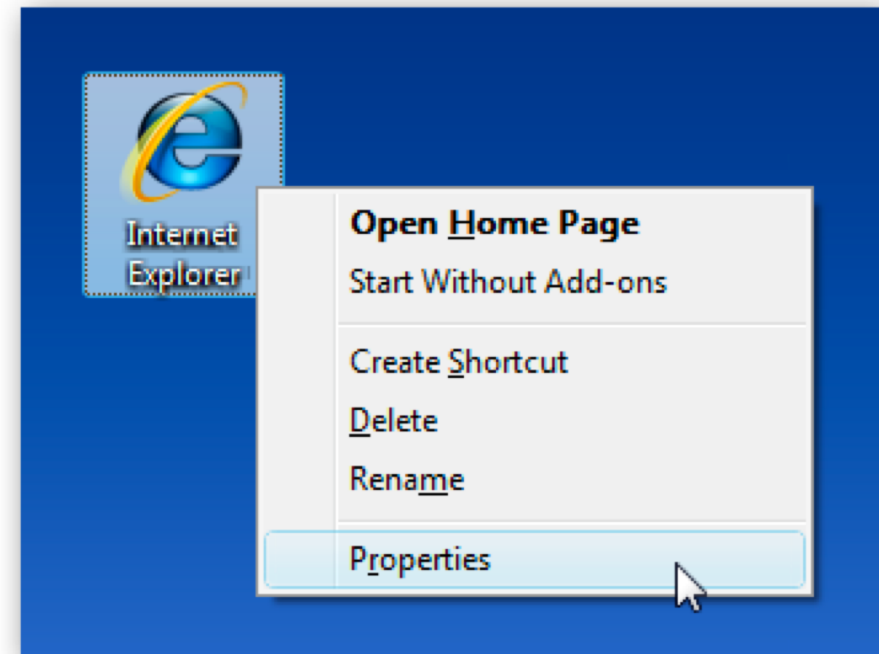
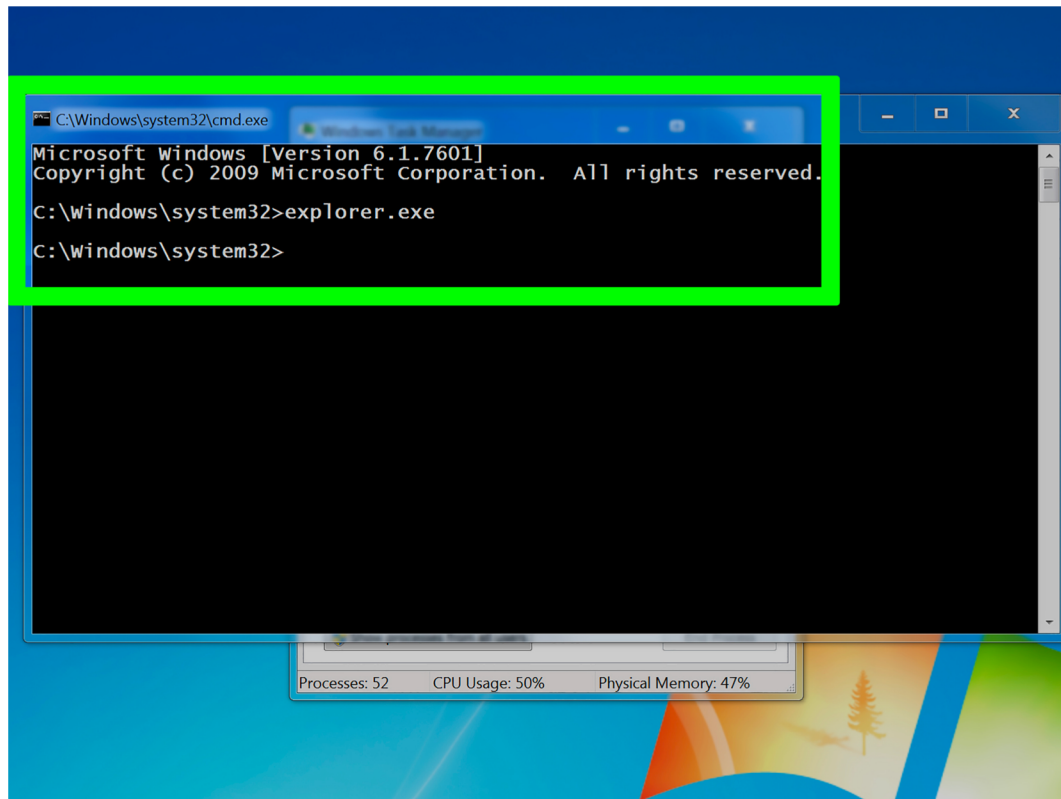
5. Error prevention

Even better than good error messages is a careful design which prevents a problem from occurring in the first place.



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.



A breakthrough in UI design was the advent of

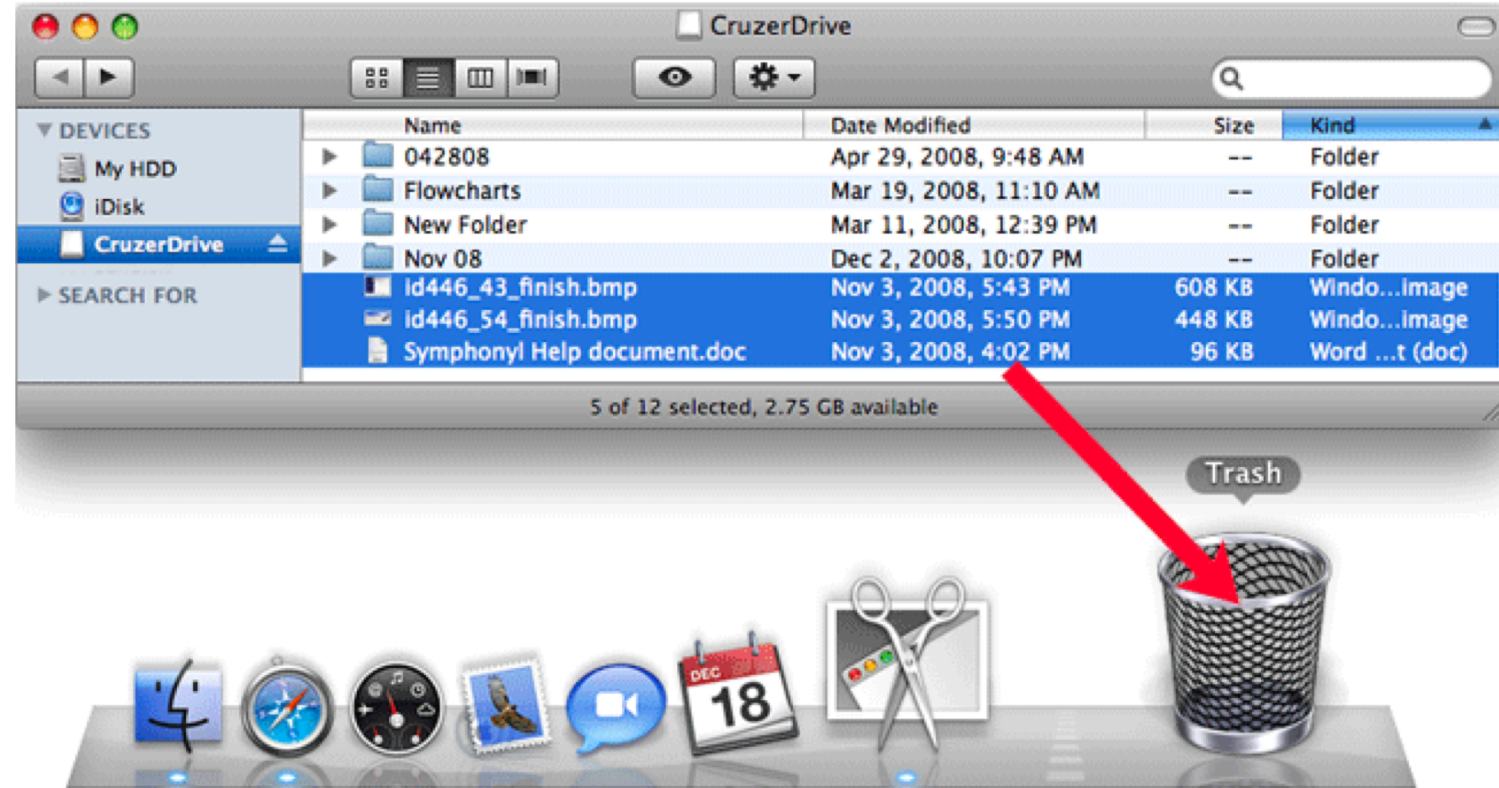
Direct manipulation

Direct Manipulation Properties

1. **Objects** are represented visually

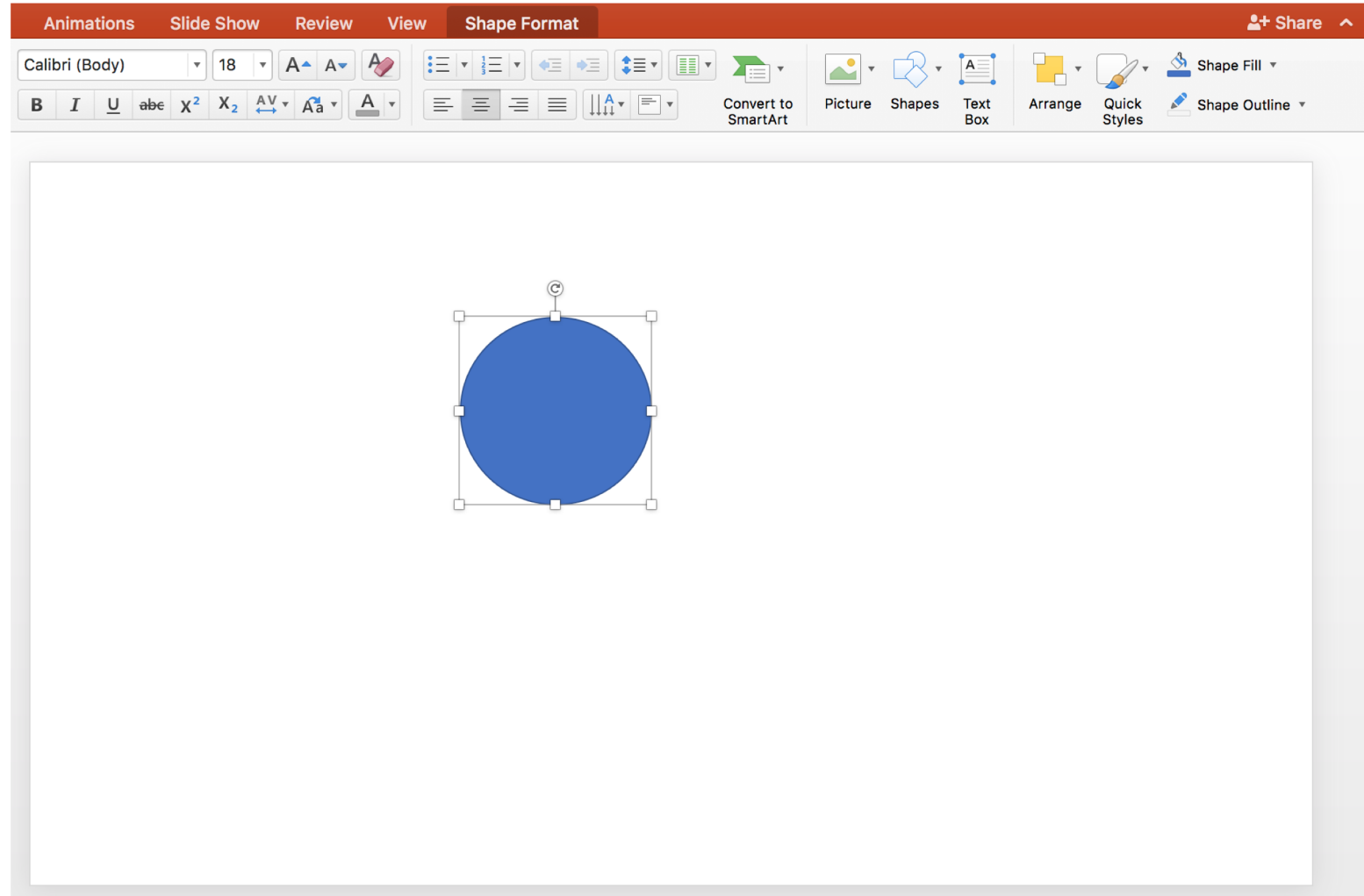
2. **Actions** are rapid, incremental and reversible

3. Users interact **directly with object representations**



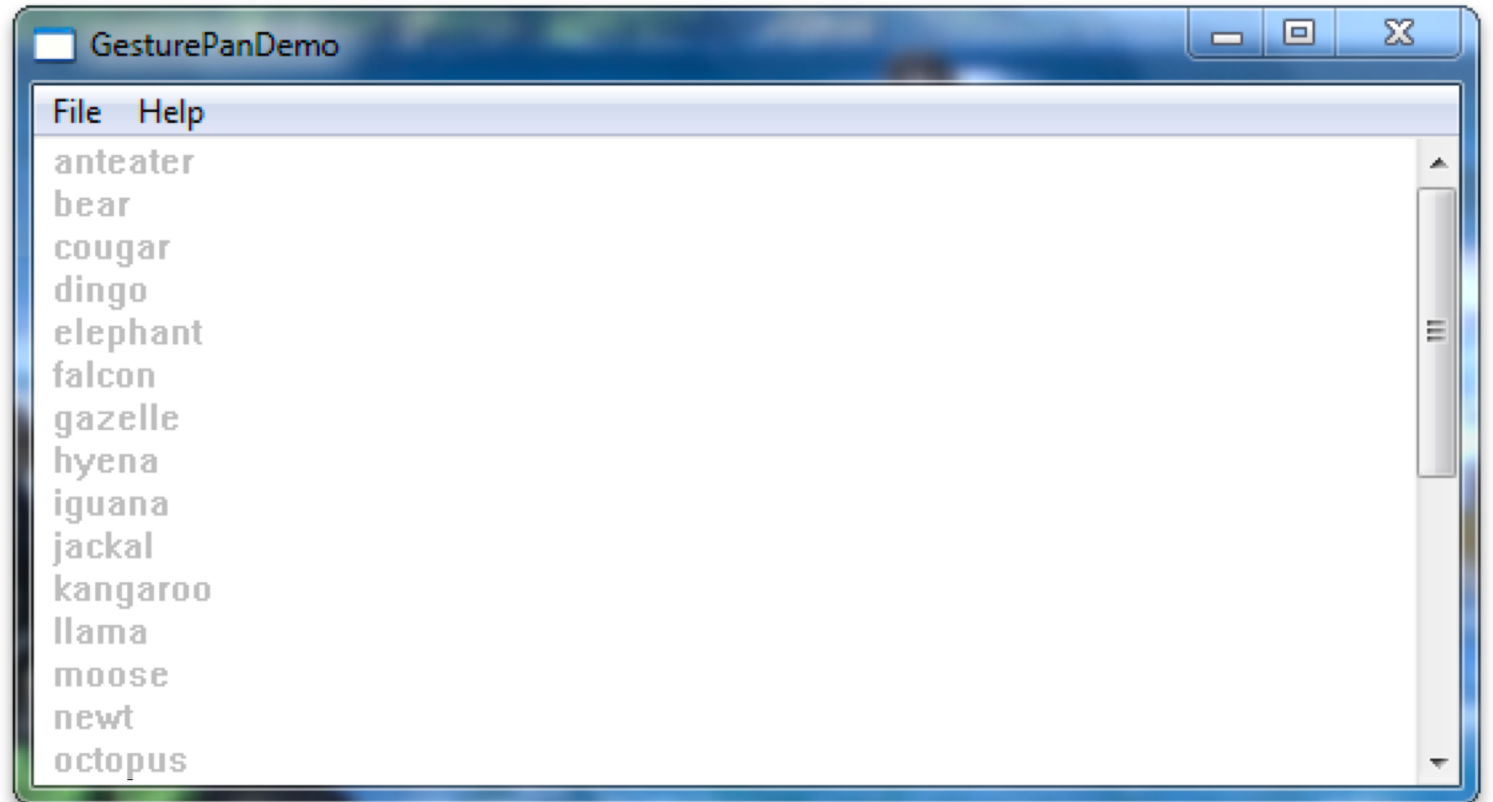
Goal: Make circle bigger.

1. What **Objects** are represented visually?
2. What **Actions** are rapid, incremental and reversible?
3. How do users interact **directly with object representations**



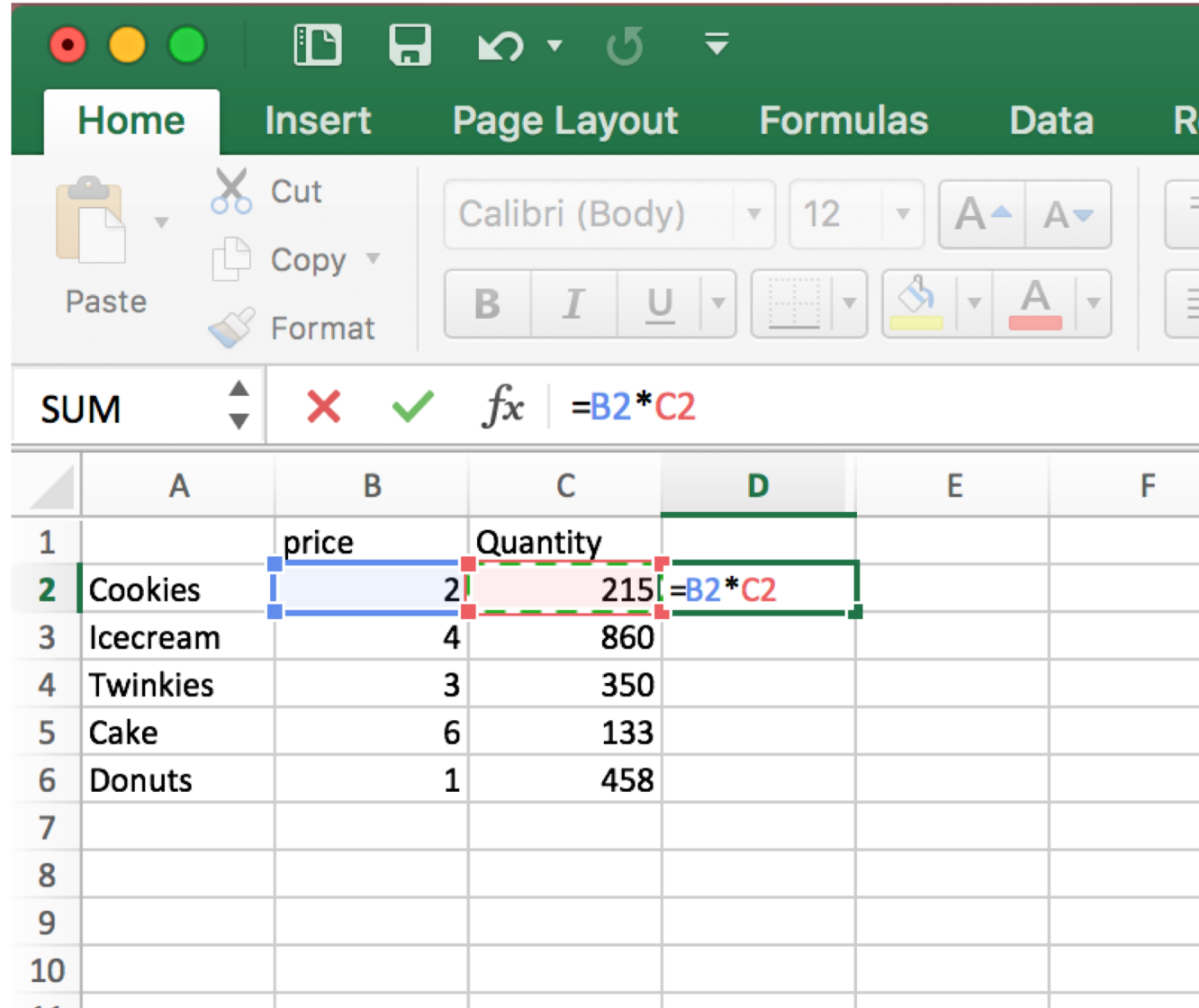
Goal: See stuff at the bottom on the list.

1. What **Objects** are represented visually?
2. What **Actions** are rapid, incremental and reversible?
3. How do users interact **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 users interact **directly with object representations**



The screenshot shows a spreadsheet application interface. The ribbon is set to 'Home', and the formula bar displays '=B2*C2'. The spreadsheet data 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						

Direct Manipulation Properties

1. **Objects** are represented visually



Move file to...



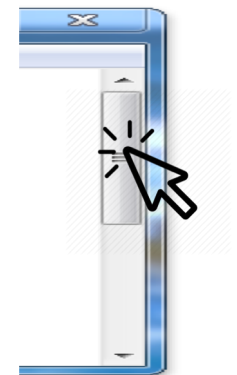
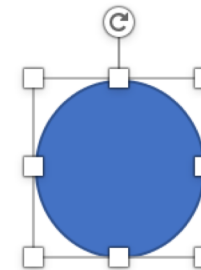
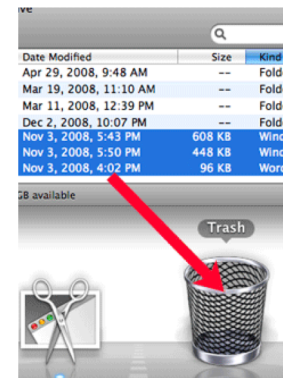
Resize



Move viewport

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

3. Users interact
directly with object representations



Goal: Set an alarm on a 2003 cell phone.
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.



Goal: Set an alarm on Google.

Is this direct manipulation? **No.**

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

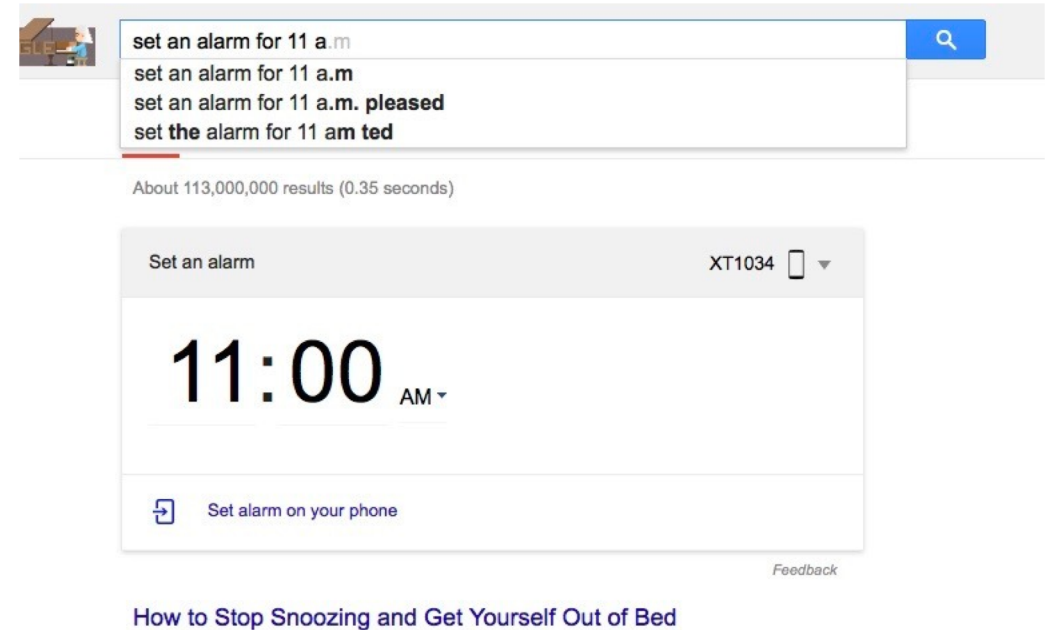
Yes.

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

No.

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

No.



Goal: Set an alarm clock with on an iPhone.
Is this direct manipulation? **Yes.**

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

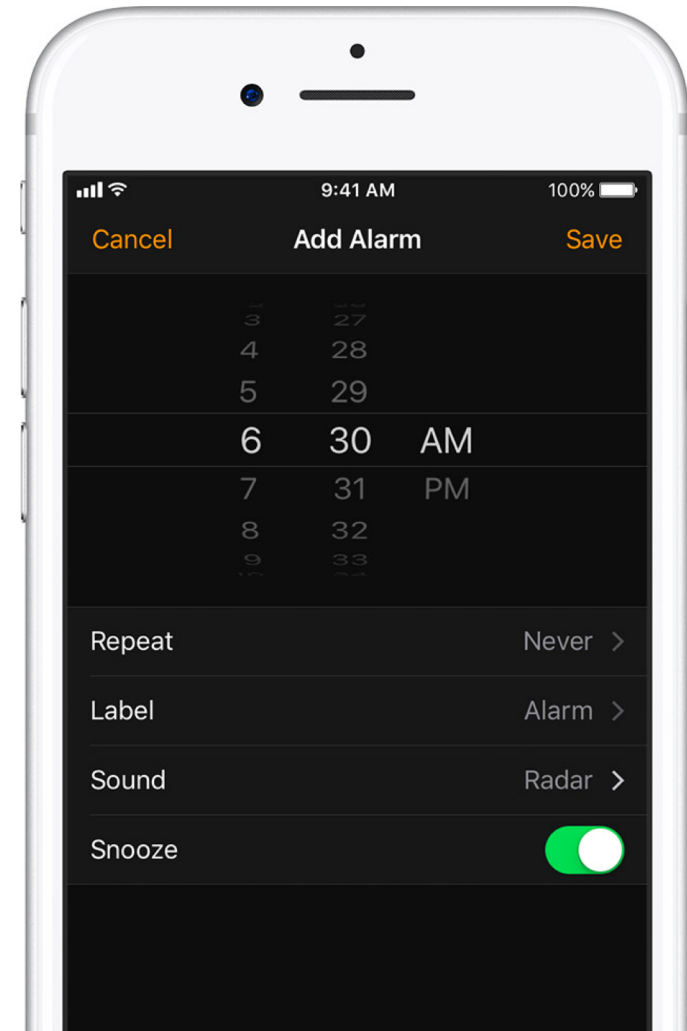
Yes.

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.

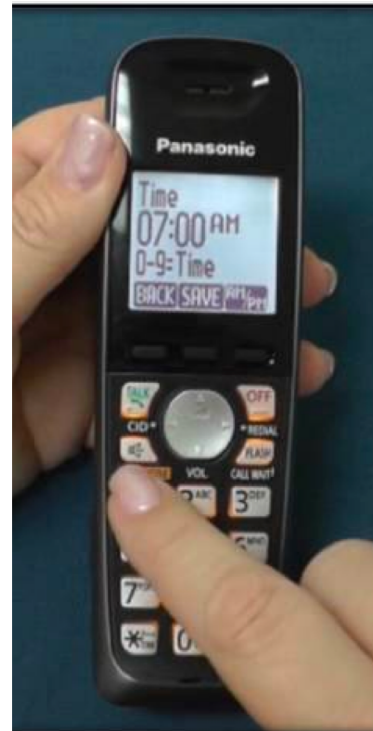
3. Do user interacts **directly with object representations?**

No.

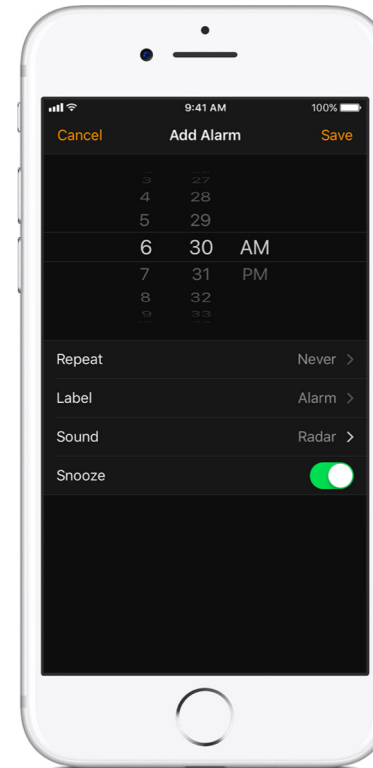


Direct manipulation requires directly interacting with object representation.

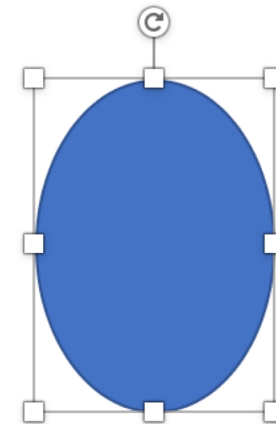
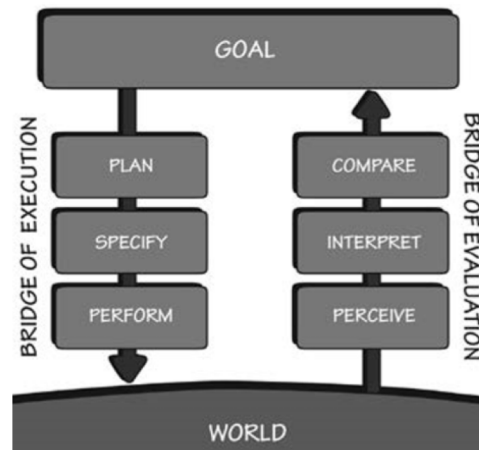
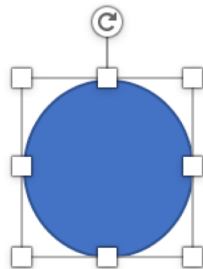
Not direct manipulation



Direct manipulation



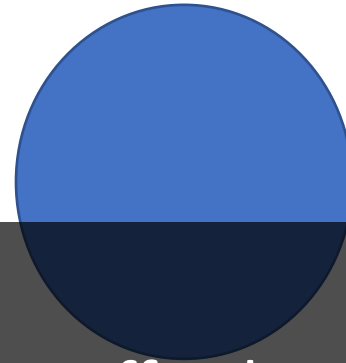
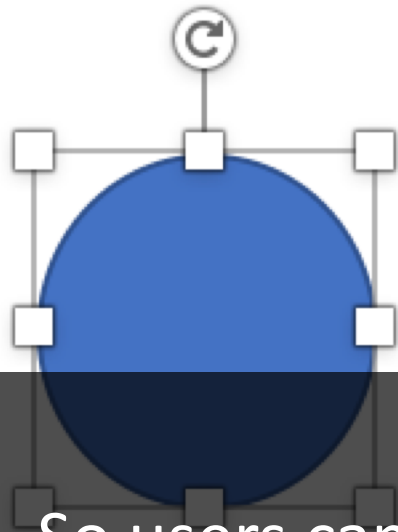
In the execution/evaluation model, why is Direct Manipulation good?



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.)

Signifiers of Affordances

Helping people see what they can do.

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.

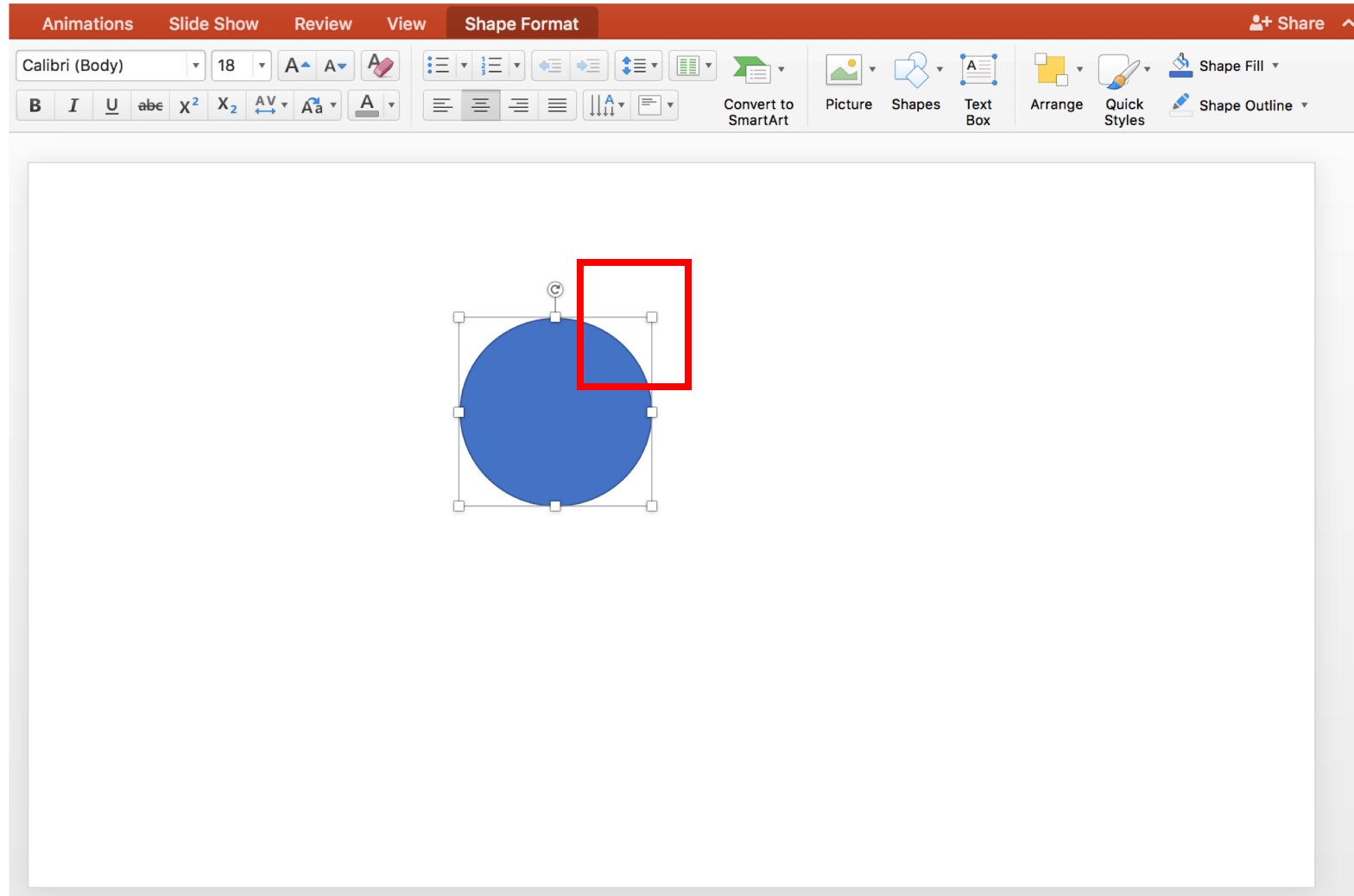
Bad signifiers / wrong perceived affordances

Good signifiers / correct perceived affordances

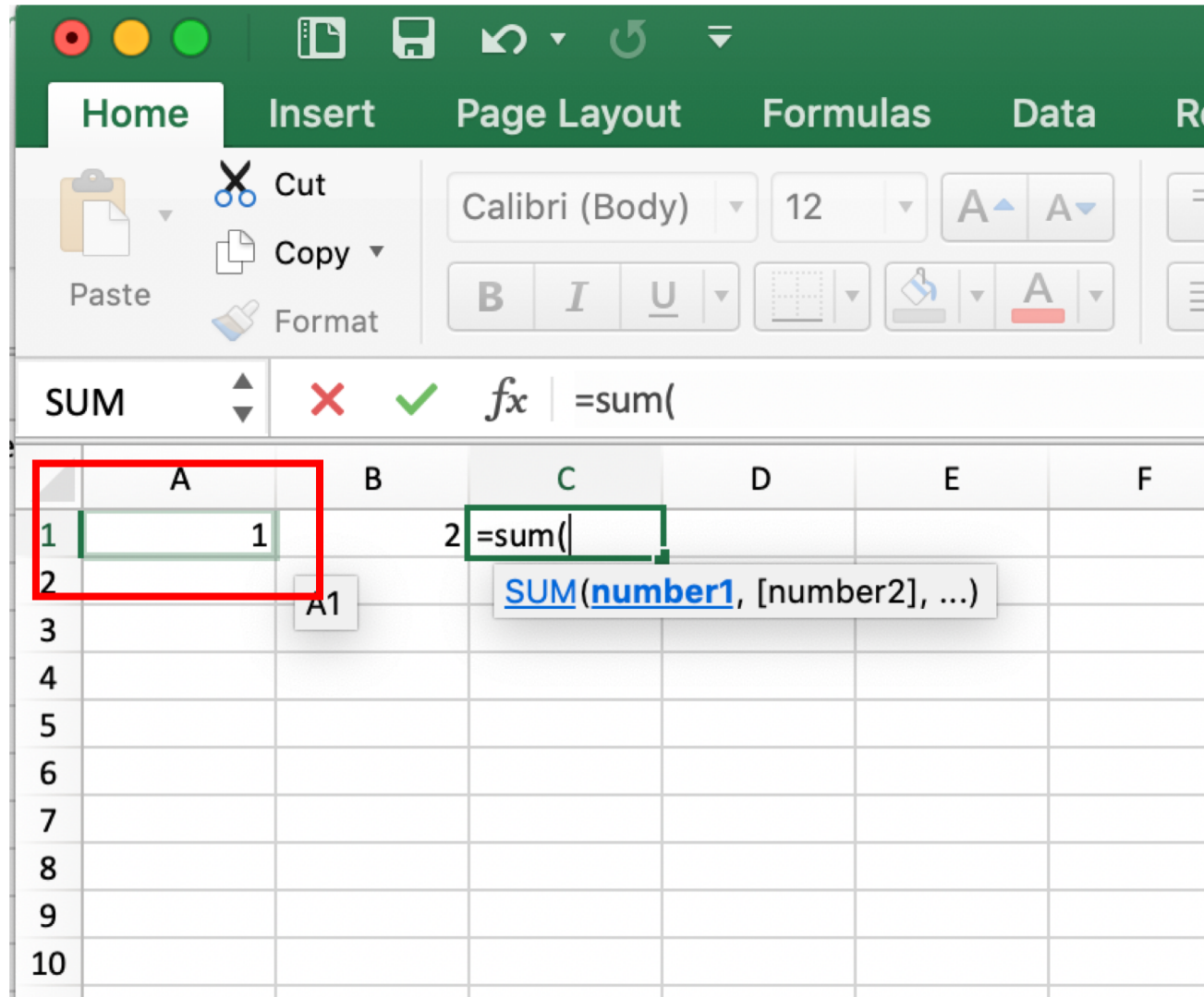


What signifiers do these UIs use to signal affordances?

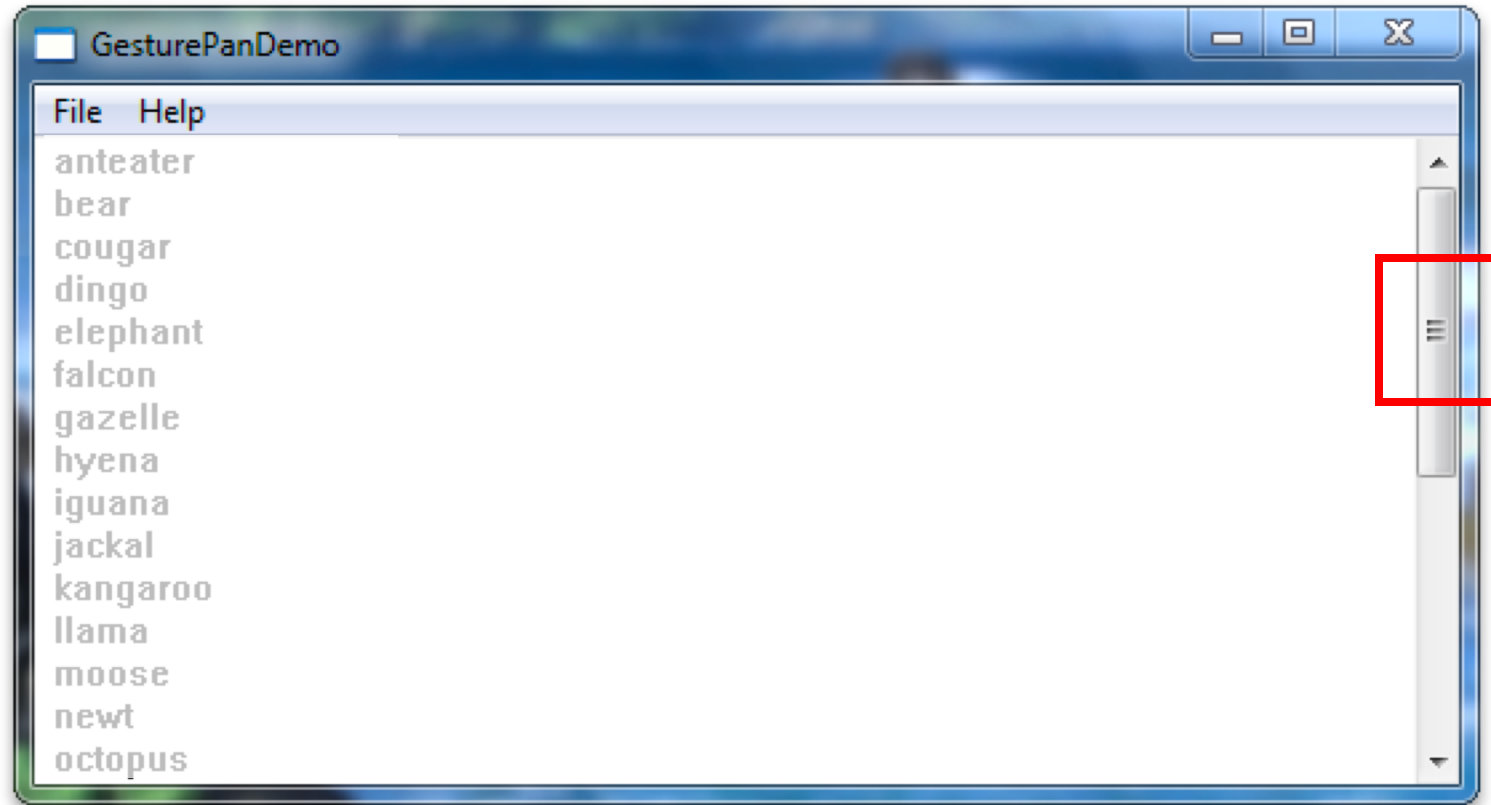
What are the signifiers of affordances?



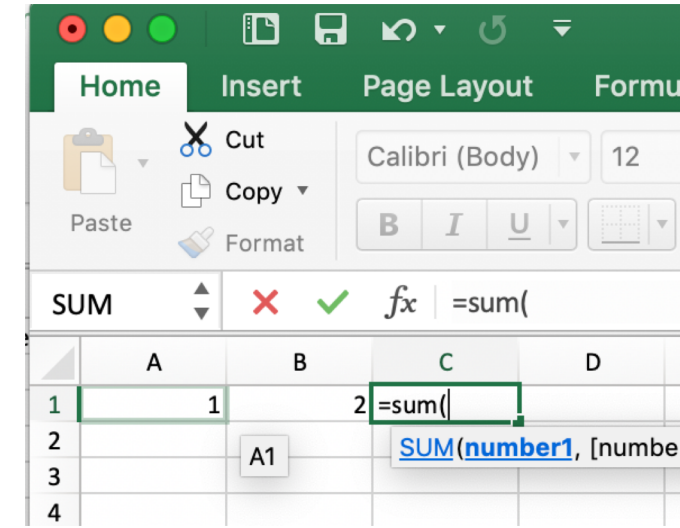
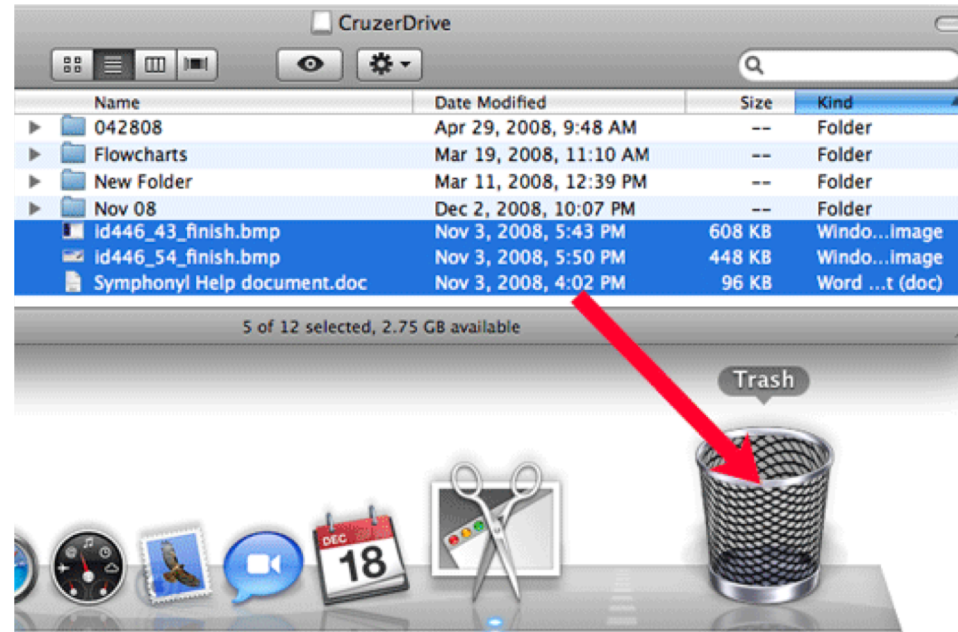
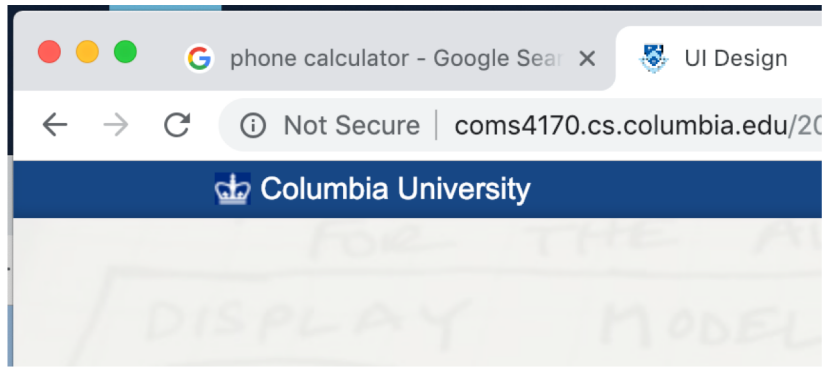
What are the signifiers of affordances?



What are the signifiers of affordances?



What are the signifiers of affordances?



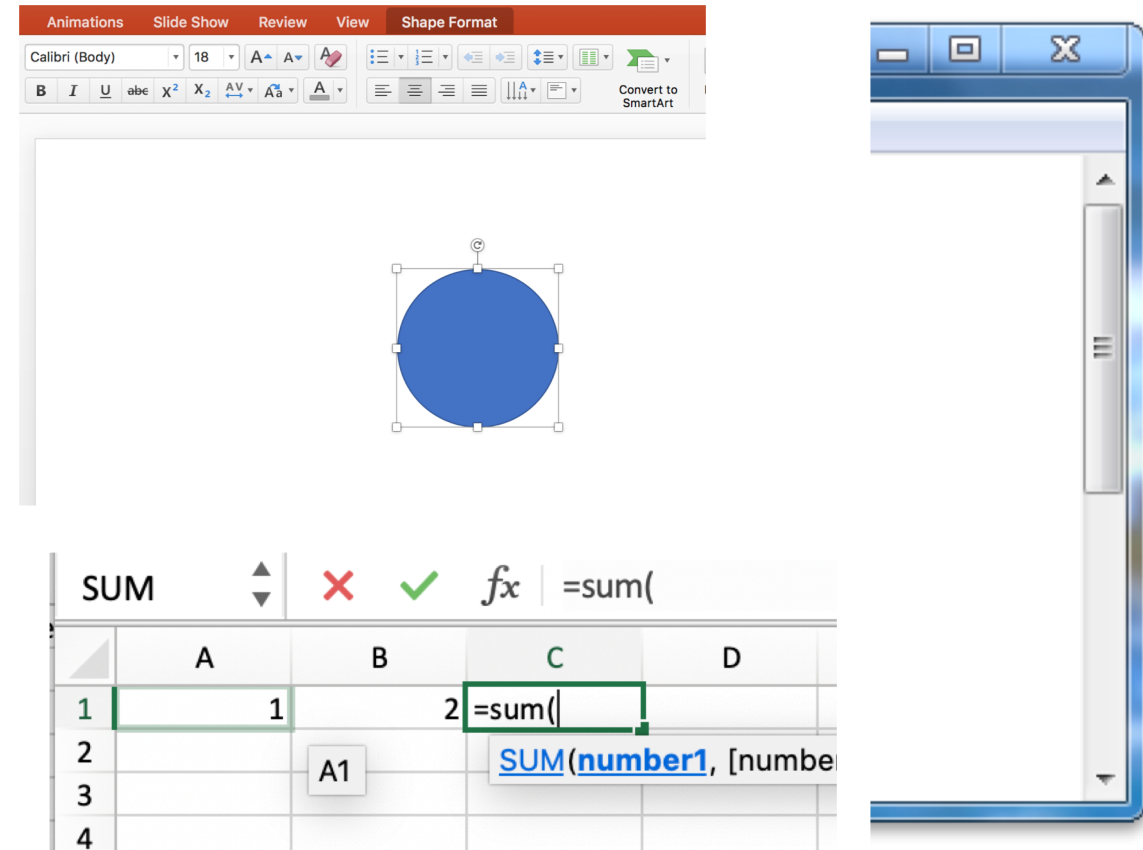
Hover event changes (like highlighting) often signify direct manipulation

Design direct manipulation interfaces with good *perceived* affordances.

Bad signifiers / wrong perceived affordances

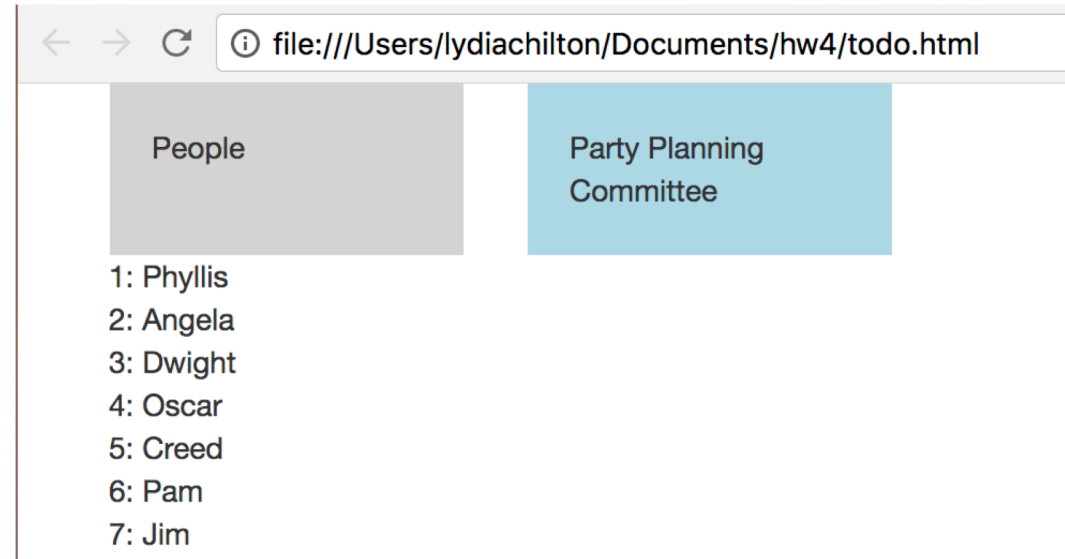


Good signifiers / correct perceived affordances



	A	B	C	D
1	1	2	=sum(
2				
3		A1	SUM(number1, [numbe	
4				

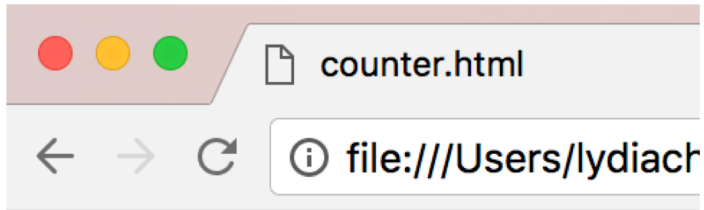
Implementing Direct Manipulation Interfaces



BUT FIRST...

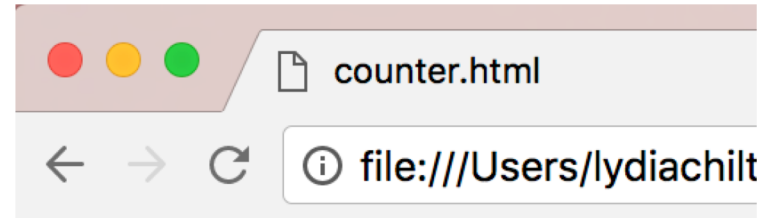
Model, View, Controller (MVC) Style Programming

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



Counter (0)

Click

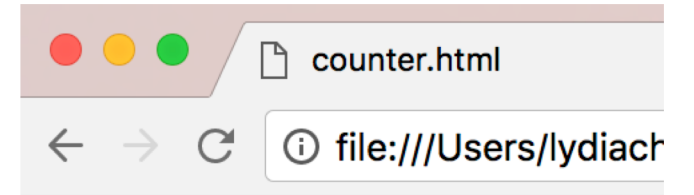


Counter (1)

Create a Button in HTML

HTML

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

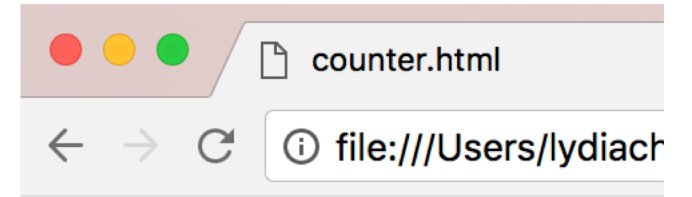


Counter (0)

Add JQuery and Bootstrap “libraries”

HTML

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



Counter (0)

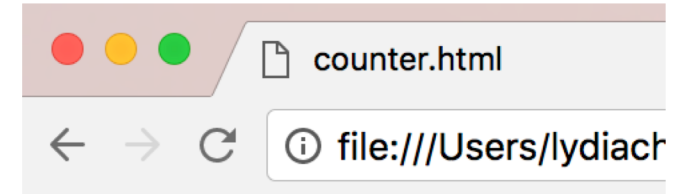
JavaScript

```
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
```

We attach a click handler

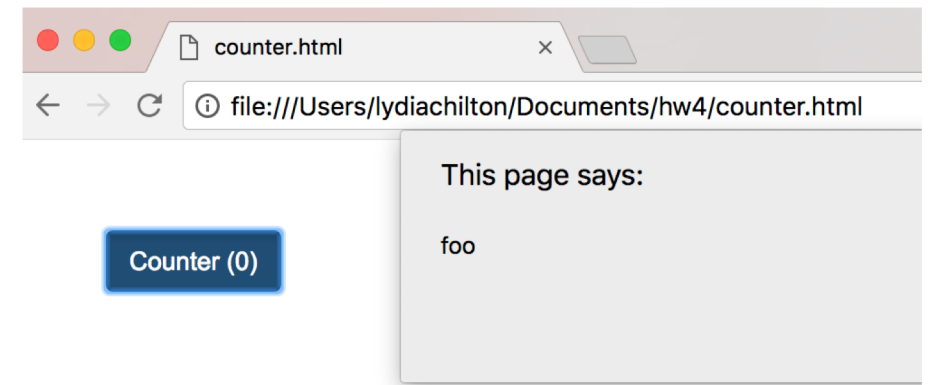
HTML

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



JavaScript

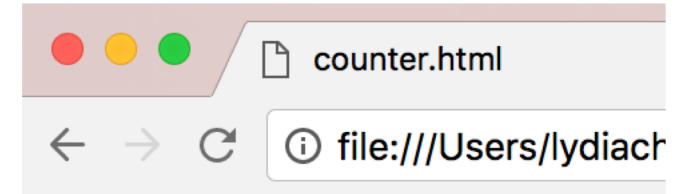
```
25  
26 $(document).ready(function(){  
27   $("#counter").click(function(){  
28     alert("foo")  
29   })  
30 })  
31
```



How 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         let html = $(this).html()  
29  
30         let regexp = /\(([^\)]+)\)/;  
31         let matches = regexp.exec(html);  
32  
33         let number = 1*matches[1]  
34         let 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

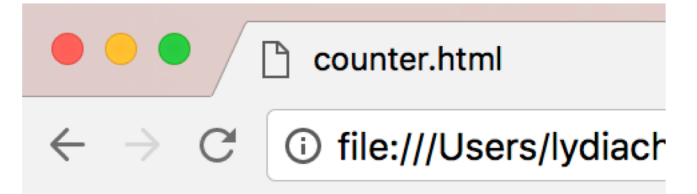
Replace the button text: **“Counter (1)”**

Problem? State is stored ONLY in the UI.

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 let 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 (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

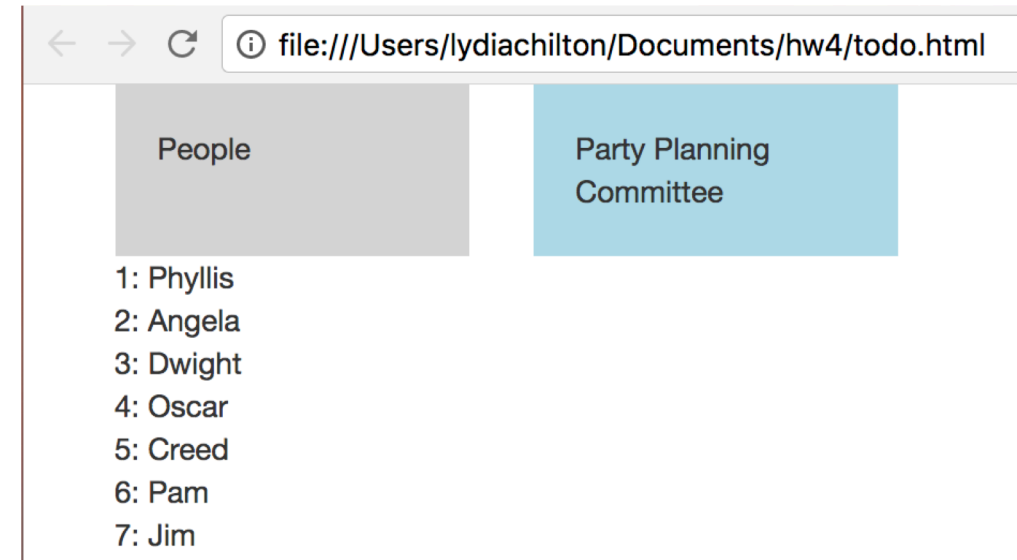
Not MVC:
data stored in UI

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

```
31 <body>
32   <button id="counter" class="btn btn-primary">Counter (0)</button>
33 </body>
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
```

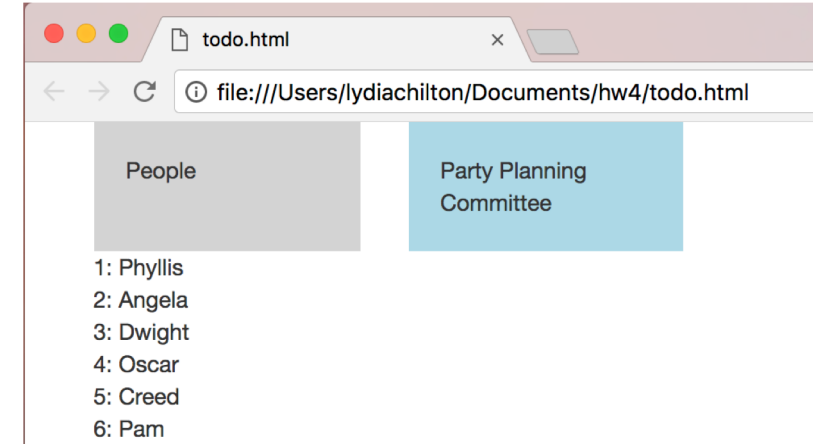
```
41
42   let count = 0
43
44   function setCount(count){
45     $("#counter").html("Counter (" + count + ")")
46   }
47
48   $(document).ready(function(){
49     setCount(count)
50
51     $("#counter").click(function(){
52       count = count + 1
53       setCount(count)
54     })
55   })
56
```

Implementing Direct Manipulation



Direct Manipulation Properties

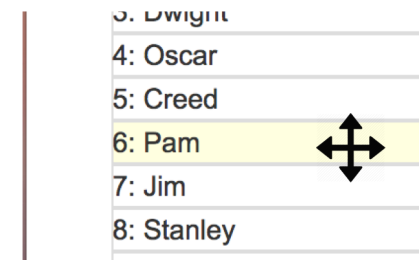
1. **Objects** are represented visually



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

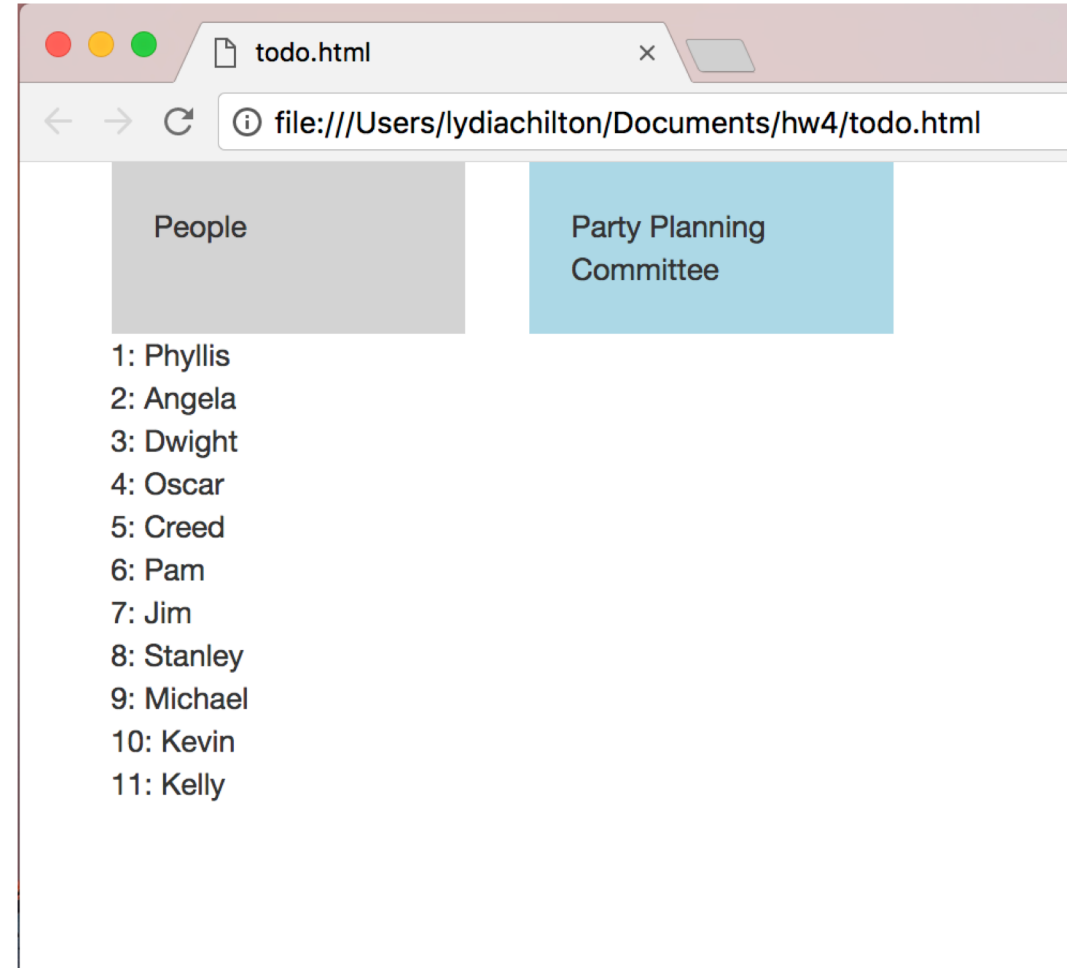
Drag and Drop

3. User interacts
directly with object representations



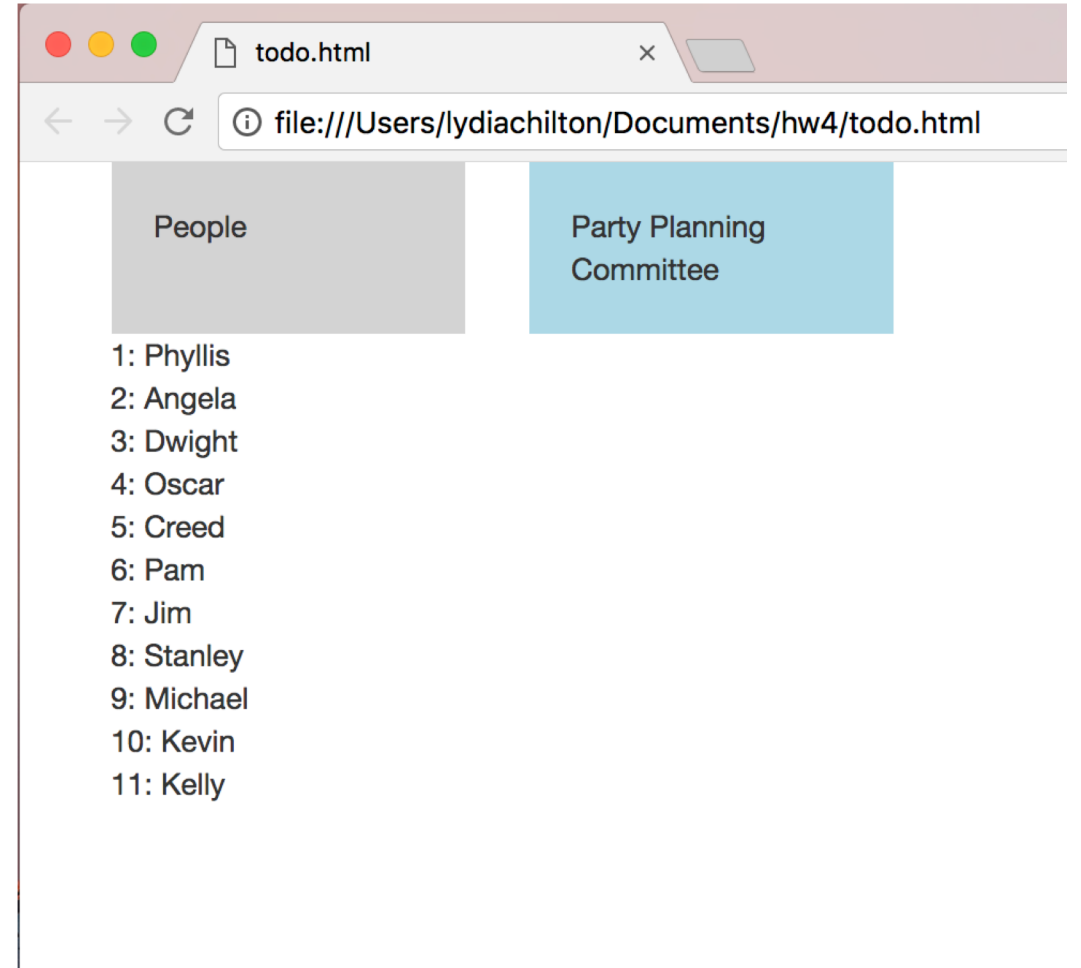
How to NOT implement this?

```
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
```



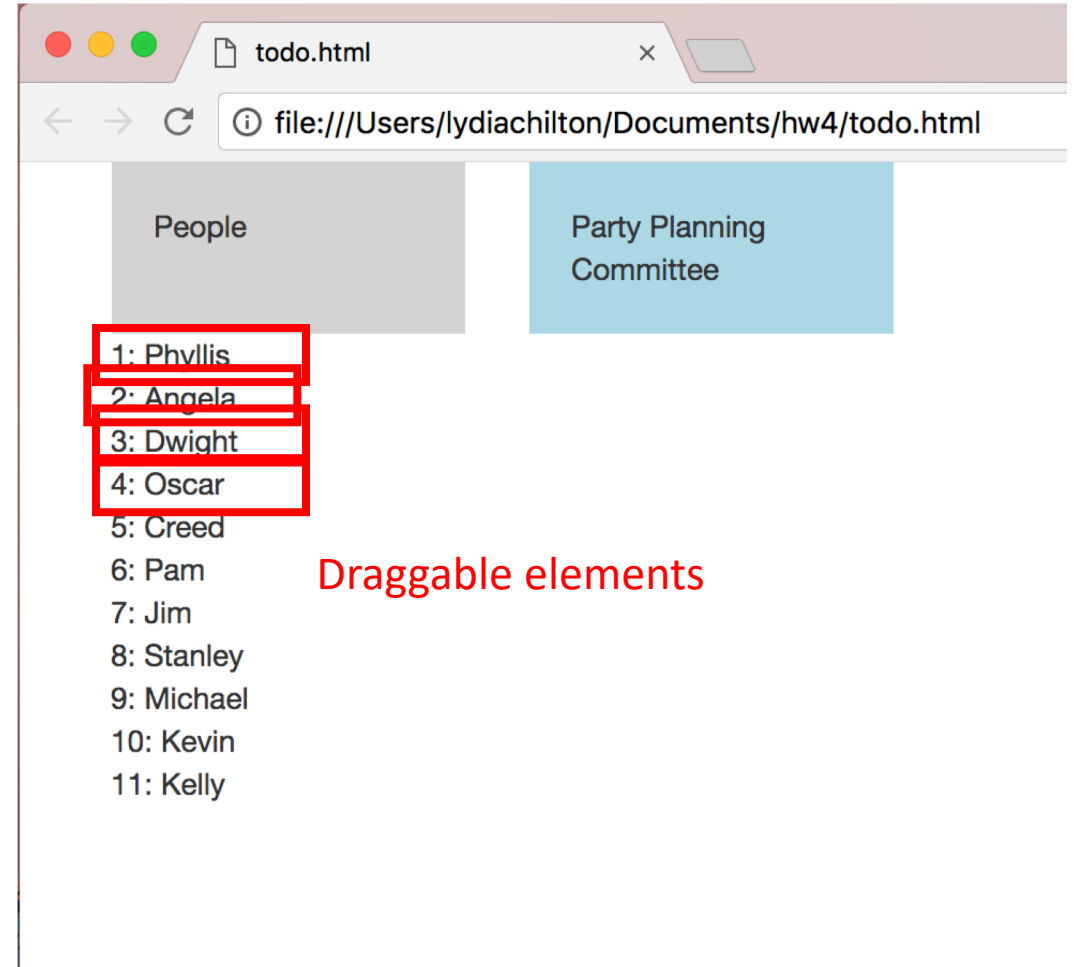
Step 1. Create the Data Model

```
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
```



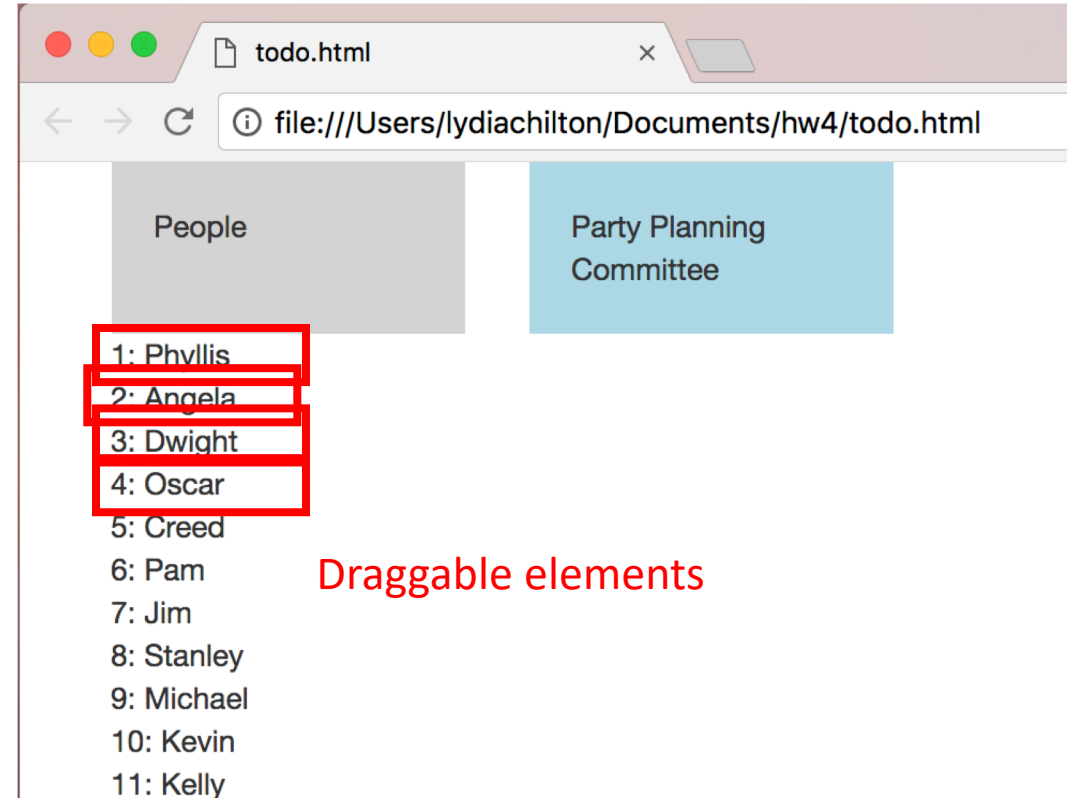
Step 2. Create a function that updates the view with new data

```
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
```



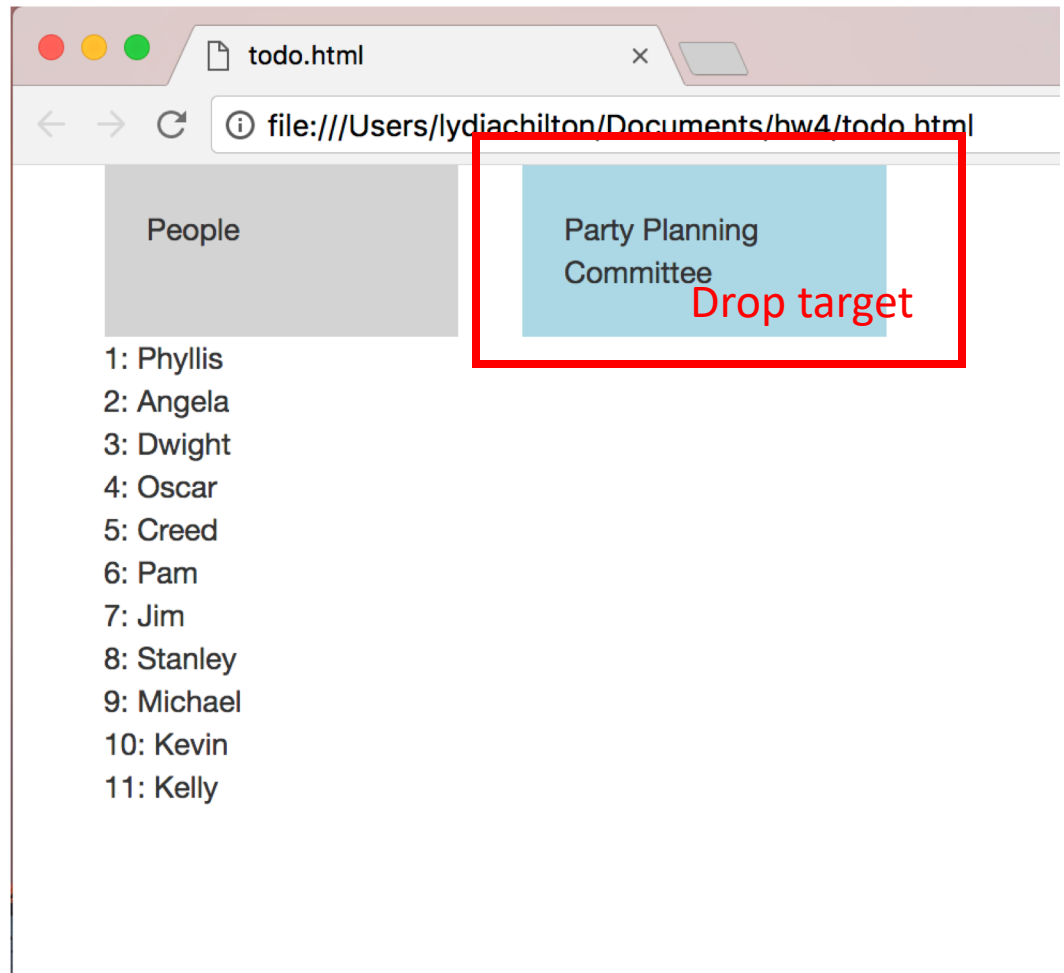
Step 3. On page load, create view.

```
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 })
```

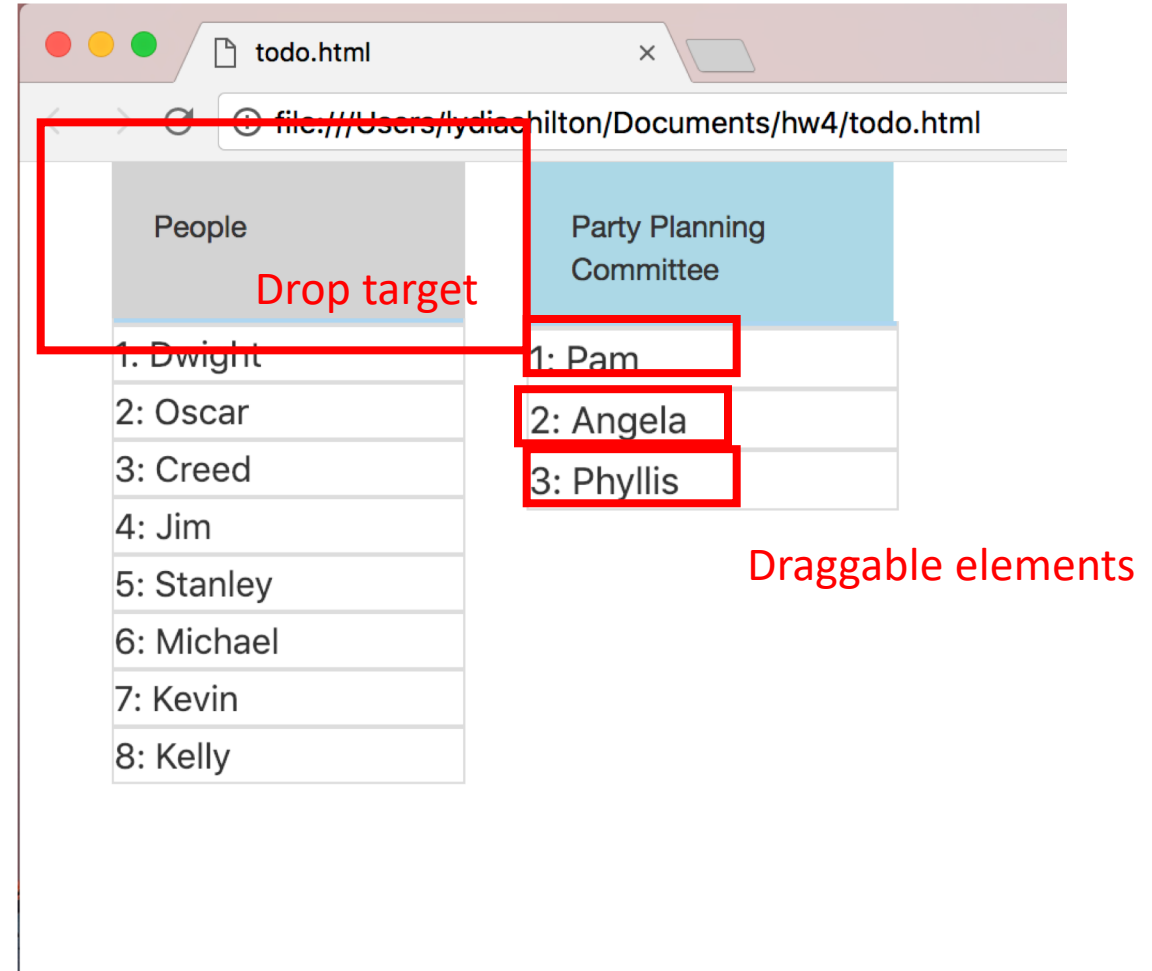


Step 4. Attach a drop event to the drop target.
It should update the data, then update the view

```
188
189
190 $(document).ready(function(){
191     makeNames(names)
192
193     $('#ppc_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
```

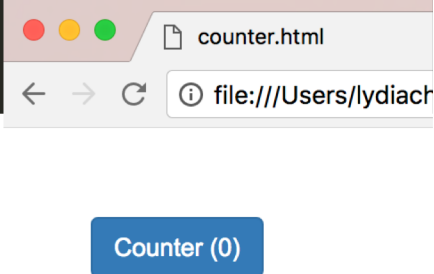


Step 5. What else do we need to do?

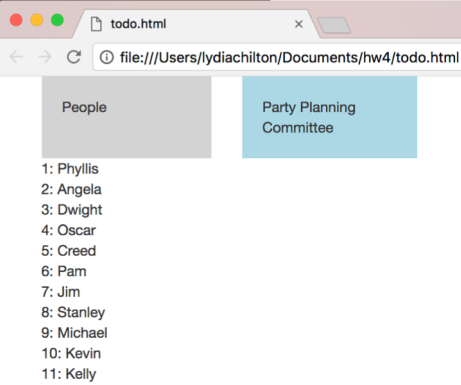


Good UI programming separates the **data** model from the **view** and **controller**

```
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
```

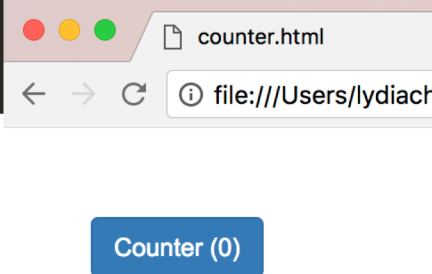


```
170
171 var names = [
172     "Phyllis",
173     "Angela",
174     "Dwight",
175     "Oscar",
176     "Creed",
177     "Pam"
178 ]
179 var list1 = []
180
181
182 function makeNames(names){
183     $("#names").empty()
184     $.each(names, function( index, value ) {
185         //make the draggable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

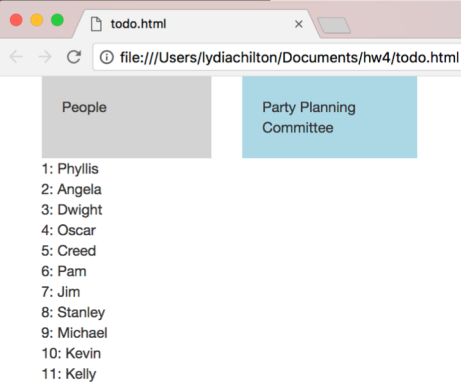


Good UI programming separates the **data** model from the **view** and **controller**

```
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
```

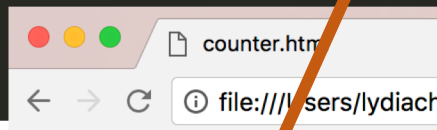


```
170
171 var names = [
172     "Phyllis",
173     "Angela",
174     "Dwight",
175     "Oscar",
176     "Creed",
177     "Pam"
178 ]
179 var list1 = []
180
181
182 function makeNames(names){
183     $("#names").empty()
184     $.each(names, function( index, value ) {
185         //make the draggable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



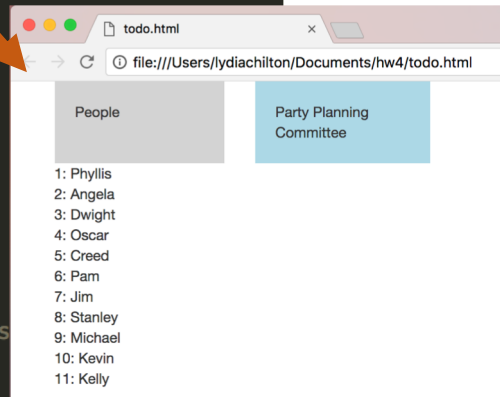
Good UI programming separates the **data** model from the **view** and **controller**

```
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
```



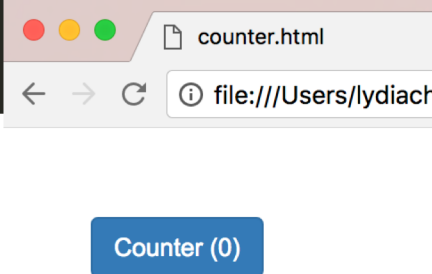
Counter (0)

```
170
171 var names = [
172     "Phyllis",
173     "Angela",
174     "Dwight",
175     "Oscar",
176     "Creed",
177     "Pam"
178 ]
179 var list1 = []
180
181
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     $("#ppc_label").draggable({
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
```

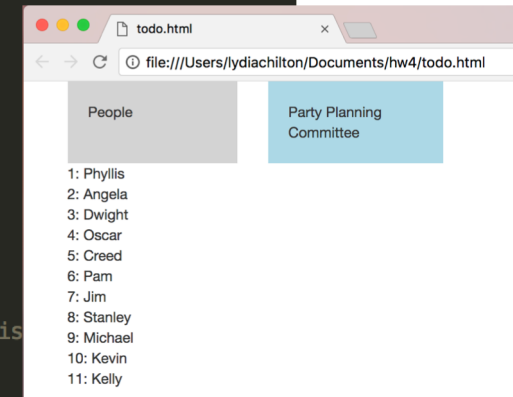


Good UI programming separates the **data** model from the **view** and **controller**

```
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
```

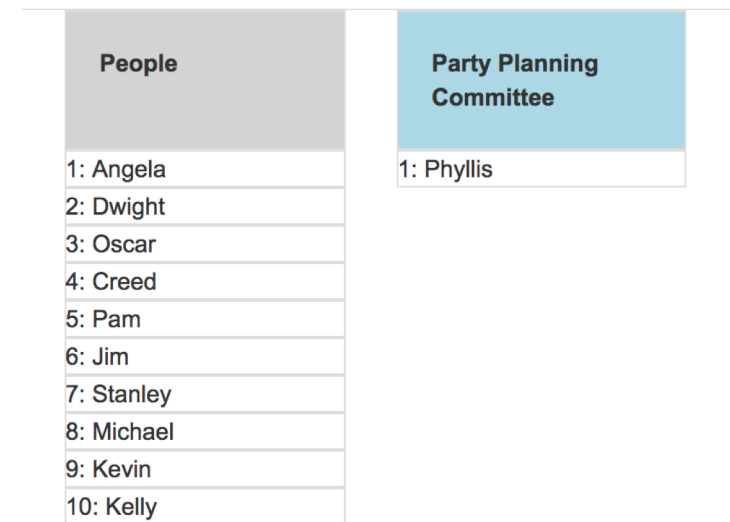
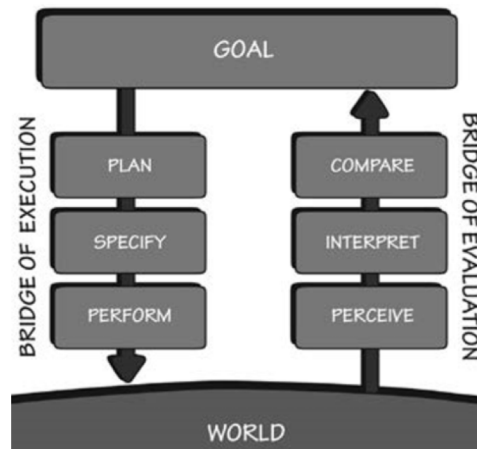
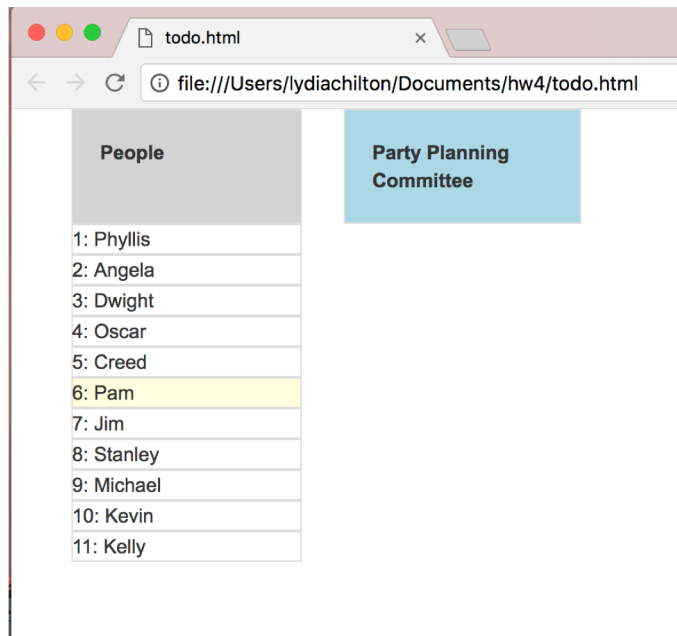


```
170
171 var names = [
172     "Phyllis",
173     "Angela",
174     "Dwight",
175     "Oscar",
176     "Creed",
177     "Pam"
178 ]
179 var list1 = []
180
181 function makeNames(names){
182     $("#names").empty()
183     $.each(names, function( index, value ) {
184         //make the draggable name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



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

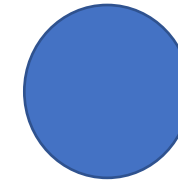
There is visible **feedback** the user can **evaluate**

Direct Manipulation Properties

1. **Objects** are represented visually



Move to trash



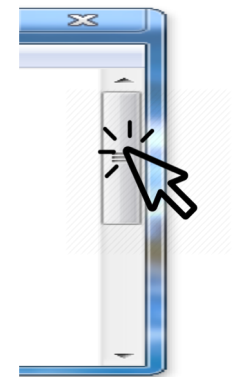
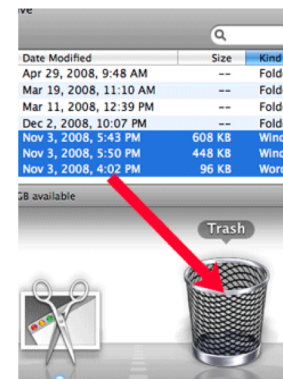
Resize



Move viewport

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

3. User interacts
directly with object representations



Signifiers help users perceive affordances

Bad signifiers



Signifier Handle that can be yanked toward you

Perceived affordance **Pull**

Affordance **Push**

Good signifiers



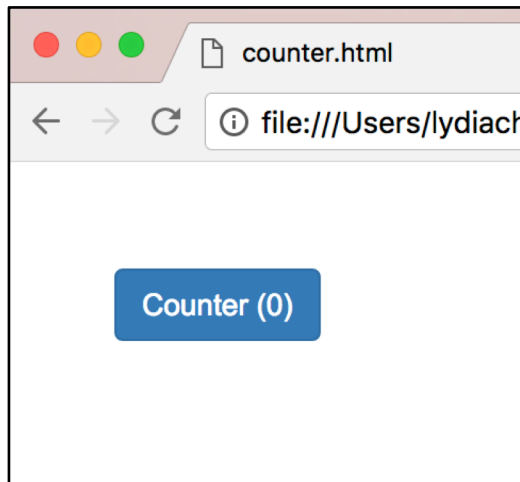
Signifier Handle that can be leaned on

Perceived affordance **Push**

Affordance **Push**

Use MVC to implement 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
```

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

Not MVC:
data stored in UI

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

```
31 <body>
32   <button id="counter" class="btn btn-primary">Counter (0)</button>
33
34 </body>
35
36
37 $(document).ready(function(){
38   $("#counter").click(function(){
39     let html = $(this).html()
40
41     let regexp = /\(([^\)]+)\)/;
42     let matches = regexp.exec(html);
43
44     let number = 1*matches[1]
45     let incremented_number = number +1
46
47     $(this).html("Counter (" + incremented_number + ")")
48   })
49 })
```

```
41
42 let 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
```

Homework 4: JavaScript Widgets and MVC

Warm up: due Friday 2/9 @ 11:59pm on Courseworks

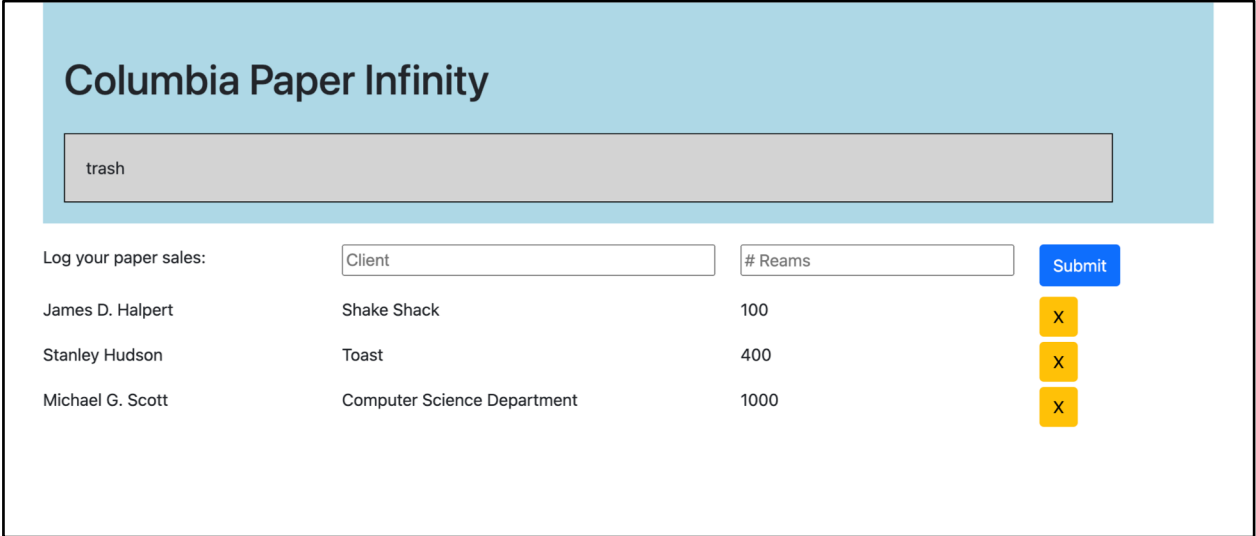
Main: due Tuesday 2/13 @ 11:59pm on Courseworks.

Warm-up:



Adding a drag and drop widget with customization

Main:



A screenshot of a web application titled "Columbia Paper Infinity". At the top, there is a light blue header with the title. Below the header is a gray trash bin icon labeled "trash". Underneath is a form with the label "Log your paper sales:" and two input fields: "Client" and "# Reams", followed by a blue "Submit" button. Below the form is a table with three columns: "Client", "# Reams", and a column of yellow buttons with a black "X" icon.

Client	# Reams		
James D. Halpert	Shake Shack	100	X
Stanley Hudson	Toast	400	X
Michael G. Scott	Computer Science Department	1000	X

Dynamically creating and updating elements in MVC style