CS121 Tutorial 2

This is the second iOS dev tutorial.

Purple bubbles give you information you’ll need to know.

Yellow Bubbles tell you what to do.

Orange bubbles tell you what you’re not expected to understand yet. 😊
Goals for tutorial

• Build the chopped fruit app
• Explore the architecture of an iOS app
• Explore some handy classes like NSArray and NSString
1. Follow the instructions from last time to create a new project named HW2.

2. Use the IB to create an interface with two labels and a text box. (We’ll connect them later.)
1. Open the Supporting Files folder.

2. Open main.m.

This is our main! Main launches our HW2 app by instantiating the HW2AppDelegate class.
Open this the HW2AppDelegate header file.

The HW2AppDelegate class contains our view and controller objects.
The controller class is HW2ViewController. A default definition is provided to us automatically.

The view class is created automatically for us based on the XIB file, which we create through the IB.

Once we define class members/methods for HW2ViewController, we can connect them to view members through the IB. But first, we’re going to create our model.
1. Select the File tab.

2. From the drop down menu select New.

3. From the next drop down menu select New File.
1. Select Cocoa Tools.

2. Select Objective-C class.

3. Click Next.
1. Name the new class HW2Model.

2. Select NSObject.

3. Click Next.

We’ll talk about NSObjects later.
1. Navigate to the HW2 project folder.

3. Click Create.
The new files will appear in the navigator region.
1. Open the header file.

2. Add this code.
This looks a lot like the class we designed.

theFruit is a pointer to an NSArray.

currentFruit is a pointer to an NSString.

choppedFruit is a pointer to an NSMutableString. This is a slight change from our design. We’ll talk more about it later.
You may be wondering why we aren’t making these properties (i.e. with accessor functions). As you will see, our other classes do not need to directly access these members. This means our external interface is simpler, just three methods. SIMPLER IS BETTER.

Objective-c objects are dynamically typed. They can only be created at runtime. Class members must be basic data types (int, float, char) or pointers to objects.
Objective-c does not use constructors. Objects are created by first allocating space and then initializing it. This init method will initialize our HW2Model objects.

In reviewing our design, I noticed that select fruit and chop fruit were always called together so I consolidated them into one method.
1. Open the source file.

2. Create the `checkGuess` method.
This is an instance method, as opposed to a static method, hence the – sign.

It returns a boolean value.

Its name is checkGuess.

Its only parameter is a pointer to an NSString named guess.
This “calls” the method isEqualToString on the currentFruit object with the parameter guess.

Rather than calling methods, the convention in Objective-C is to pass messages.
C++ method call:
Obj->method(arg);

is equivalent to

Objective-c message passing:
[Obj method:arg];

What about methods with multiple arguments? Hold that thought.
Now is a good time to talk about online documentation.

1. Open a web browser and google ios developer NSString. One of the first links will be this one. Open it.

2. Scroll down to the section Identifying and Comparing Strings.

3. Everything you need to know about the string comparison method is here. Just click.
Now add the init function to HW2Model.m.
Our HW2Model class is derived from the NSObject class. This line runs the initialization code for NSObject.

Why use NSObject as a base class? Go to the next page for the answer.
Open a browser, google “ios developer library NSObject” and open this link. Scroll down to Tasks. These are some of the tasks the NSObject takes care of for us. You don’t really want to write your own alloc do you?
The iOS Architecture has four layers that provide amazing services.

Cocoa Touch includes important frameworks for building iOS application. One is UIKit, which provides the UI elements we’ve been using.
The iOS Architecture has four layers that provide amazing services.

- **Core OS**
- **Core Services**
- **Media Services**
- **Cocoa Touch**

Media Services provides support for graphics, audio, and video. Next week we'll start working with graphics, using this layer.
The iOS Architecture has four layers that provide amazing services.

Core OS

Core Services

Media Services

Cocoa Touch

Core Services provides fundamental system services like ARC and Foundation classes (including NSObject).
The iOS Architecture has four layers that provide amazing services.

- **Core OS**
- **Core Services**
- **Media Services**
- **Cocoa Touch**

Core OS provides low level functionality that other classes build on like threading and Bluetooth.
Our app sits on top but we can reach down to any layer we want; e.g. to use NSObject as a base class.
Provided the super class initialization went ok, we’ll proceed to initialize our members.

Here we are creating our chopped fruit object:
1. First we allocate space with `[NSMutableString alloc]`, which returns a pointer to an NSMutableString.
2. Then we initialize it. By default it is initialized to the empty string. This is the prototypical way to create objects in Objective-C.
Here we create our list of Fruit. This is an exception to the allocate and initialize rule.
Usually creating a new object is explicitly a two step process, allocation and then initialization. NSArray provides methods to combine these two steps.

Any object with a fixed size can be allocated without any information about its contents. But container types, like arrays and strings, have variable sizes. These type of objects come in two varieties: immutable and mutable.

Immutable classes, like NSArray, have methods that combine allocation and initialization in which the size is specified, so that an appropriate space can be allocated. Mutable types have sizes that are changed dynamically; they are created in the prototypical way and, by default, are initialized to be empty.
NSArray holds pointers to objects. In this case all of the objects are NSStrings but we could include a mixture of objects. nil indicates the end of the list.
Finally we tell our model to select its first current string and chop it. We'll write that next.
This is the pseudo code for chopNewFruit:

Set currentFruit to a random element in theFruit

Copy currentFruit to choppedFruit

For (i=length(choppedFruit)-1, i>0, i--) {
    swap the character at index i with the character at a random position in [0,i]
}

This algorithm is called the Knuth shuffle!
Unfortunately, it is not easy to swap characters in an NSMutableString. So rather, we’ll first permute the character indices then create the appropriate string. Here is the revised pseudocode.

Set currentFruit to a random element in theFruit

Create array of indices: 0,1,..., length(currentFruit)-1
For (i=length(currentFruit)-1, i>0, i--) {
    swap the array element at index i with the array element at a random index in [0,i]
}

For (i=0;i<length(currentFruit)-1)
    set i-th character of chopped fruit to the indices[i] character in currentFruit
This is our `chopNewFruit` method.

Here we select the new `currentString`. `arc4random()` is a nice random number generator in that it is self-initializing. Objective-C is a superset of C; `arc4random` is a C function and is called using C syntax.

```objective-c
-(NSString*) chopNewFruit {
    // get new current string
    int randNum = (int) (arc4random() % [theFruit count]);
    currentFruit = [theFruit objectAtIndex:randNum];

    // initialize it
    [choppedFruit setString:@"" ];

    // create array of indices
    NSMutableArray* indices = [[NSMutableArray alloc] initWithCapacity: [currentFruit length]]; 
    for (int i=0; i<[currentFruit length]; i++) 
        [indices addObject: [NSNumber numberWithInt:i]]; 

    // create random permutation of indices
    for (int i=[indices count]-1; i>0; i--) { 
        randNum = (int) (arc4random() % i); 
        if (i==randNum) { 
            [indices exchangeObjectAtIndex:i withObjectAtIndex:randNum]; 
        } 
    }

    // now create chopped string
    for (int i=0; i<[indices count]; i++) { 
        int index=[indices objectAtIndex:i] intValue; 
        [choppedFruit appendFormat: @"%c", [currentFruit characterAtIndex:index]]; 
    }

    return choppedFruit;
}
-(bool) checkGuess: (NSString*) guess 
{ 
    return [currentFruit isEqualToString:guess]; 
}
@end
```
Here we create an array of \{0, 1, \ldots, \text{length}(currentFruit)-1\}. 

NSArray can only hold pointers (not ints), so we convert our integer indices to NSNumbers, which are general-purpose wrappers for numbers.

We initialize choppedFruit to the empty string. (Later we'll append characters one by one.)

Here we use Knuth's shuffle to permute our indices.
We successively append characters of `currentFruit` based on the permuted indices. The `charAtIndex` method of `NSString` requires an integer argument, so we have to convert our `NSNumbers` back to `int`. 
HW2Model interface and implementation are done!!
1. Open our controller header file.

2. Import the model header.

3. Add the interface definition.
Our `bowlOfFruit` won’t be accessed by the view, so we don’t need accessor functions.

UITextField connects to a text field on the screen. Once we connect it, whatever the user types will be tied to this variable.

When the user finishes entering their text, this is the method that will be called.
1. Open the xib file.

2. Click on the label.

These are the actions we can connect to our checkGuess method.
Click on “did end on exit.” Connect this action to our checkGuess method. (You learned how to connect UI elements to IBAction methods last time.)
You should be able to finish the app on your own.

But help is available! I will be around tomorrow until 3. Stop by or send me email if you want to meet.

Our iOS expert grutors will have grutoring hours in the LSC Saturday and Sunday 3-5 PM.