Stanford CS193p

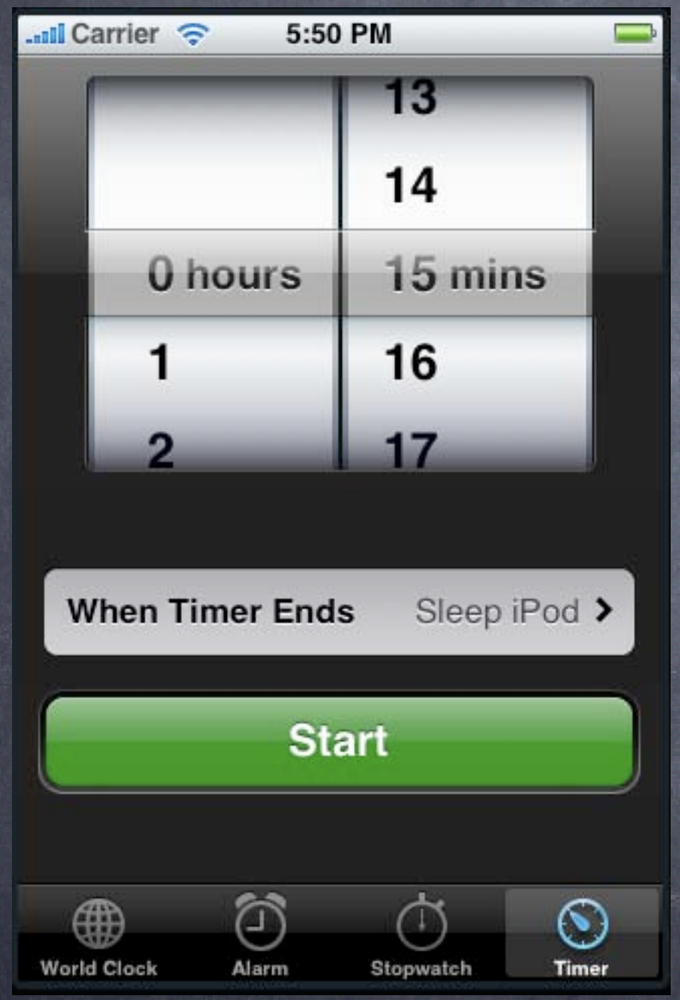
Developing Applications for iOS
Fall 2011

Today

- UITabBarController
  Another "controller of controllers"
  Mostly set up with ctrl-drag just like split view or navigation controller
- UINavigationItem
  Controlling what's at top when a `UIViewController` gets pushed onto a `UINavigationController`
- Blocks
  Objective-C language feature for in-lining blocks of code
  Foundation of multi-threaded support (GCD)

UITabBarController

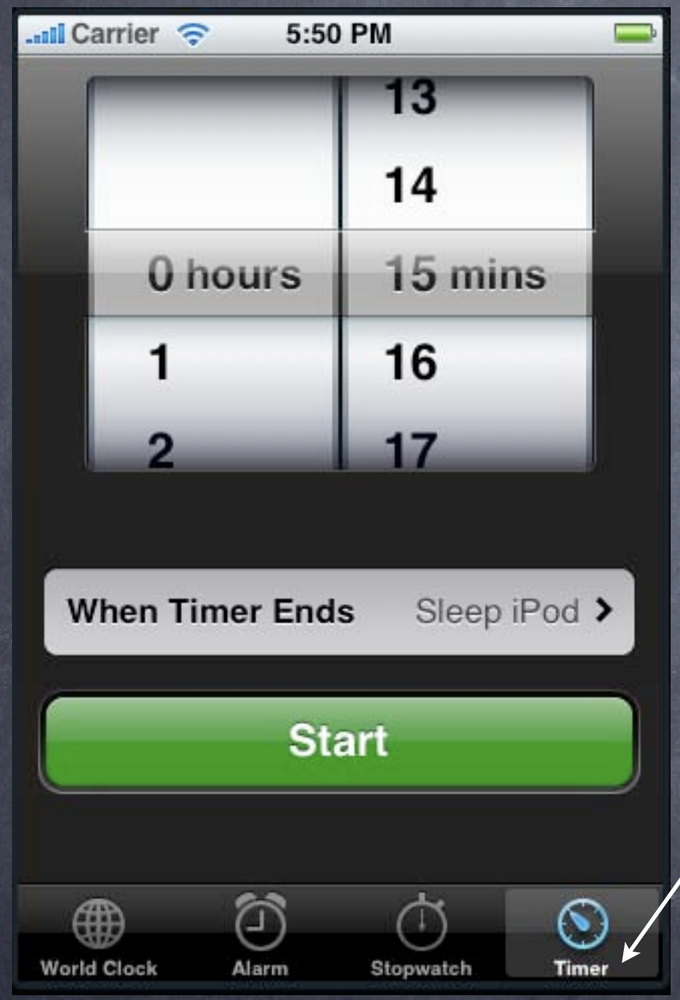
UITabBarController



You control drag to create these connections in Xcode.

Doing so is setting
`@property (nonatomic, strong) NSArray *viewControllers;`
inside your UITabBarController.

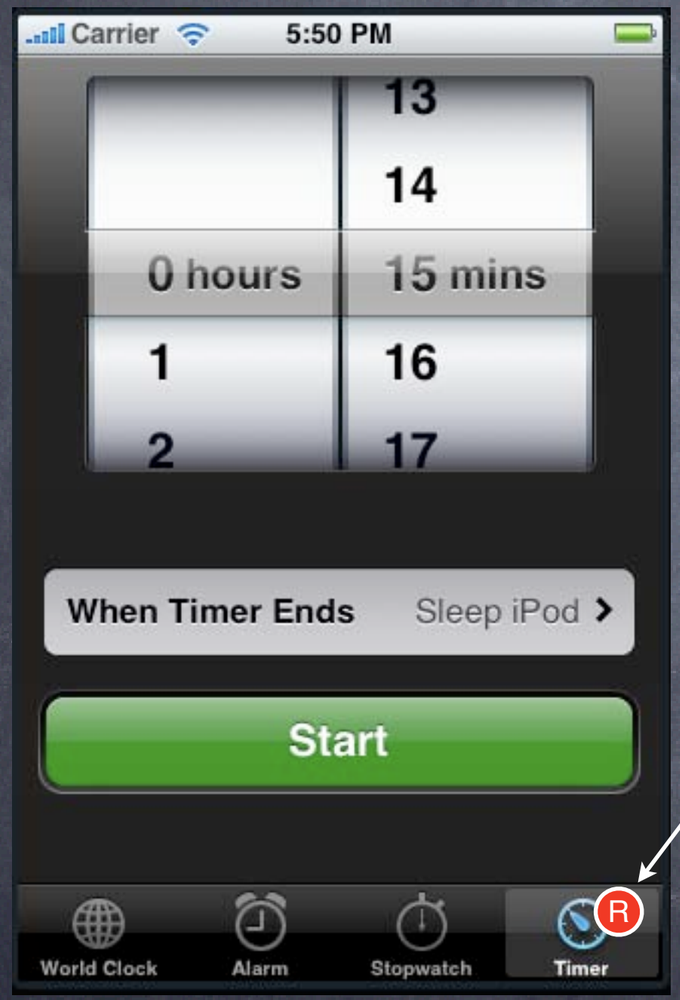
UITabBarController



By default this is the `UIViewController`'s `title` property (and no image)

But usually you set both of these in your storyboard in Xcode.

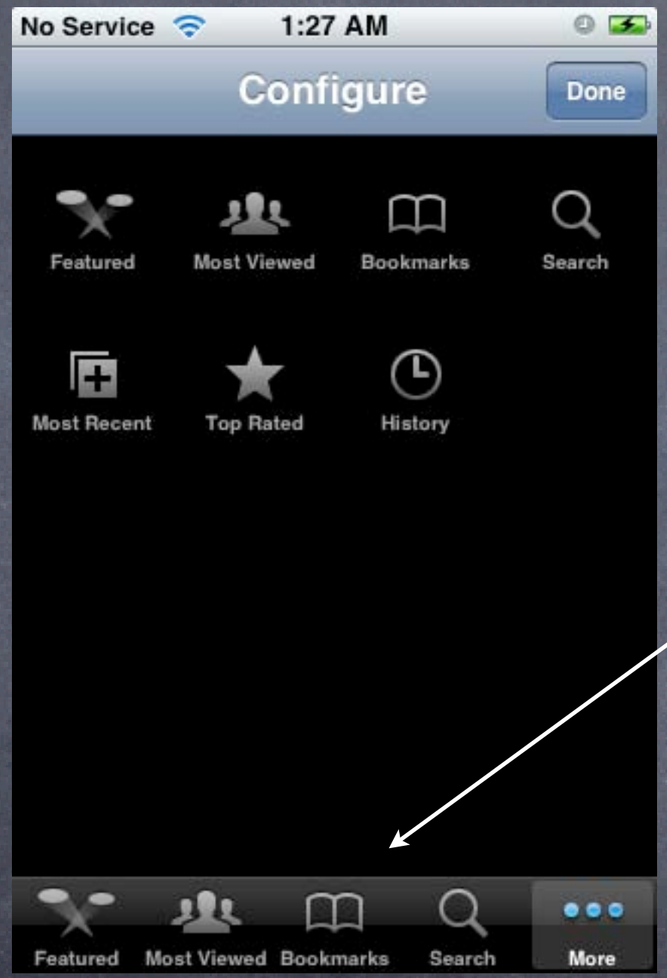
UITabBarController



UIViewController's tabBarItem property
(not a UITabBarController property)
can be used to set attributes for that VC's tab.

```
- (void)somethingHappenedToCauseUsToNeedToShowABadgeValue
{
    self.tabBarItem.badgeValue = @"R";
}
```

UITabBarController

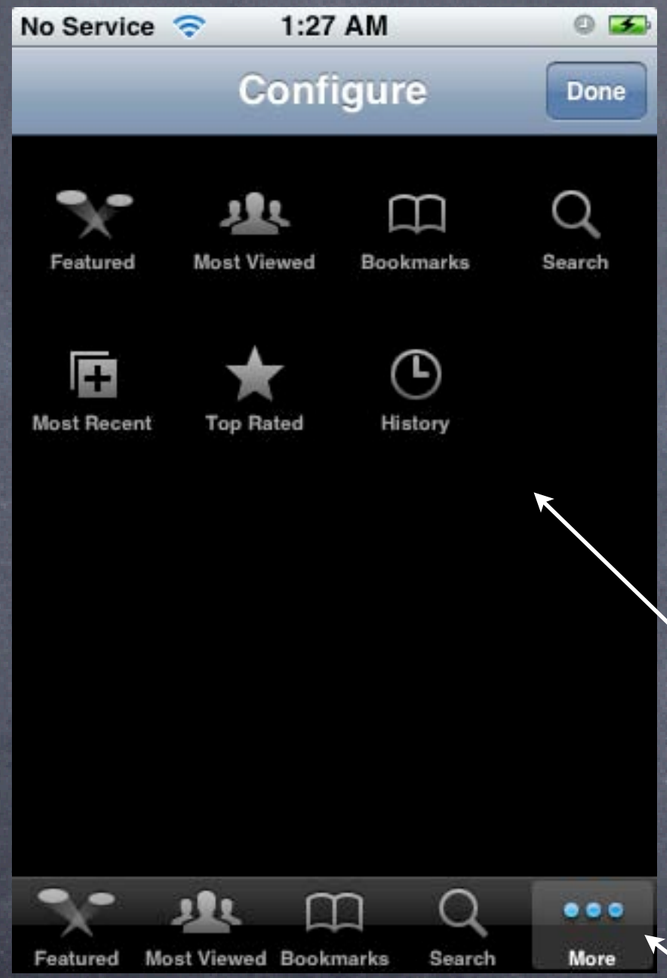


Tab Bar Controller

- View Controller
- View Controller
- View Controller
- View Controller
- View Controller
- View Controller
- View Controller

What if there are more than 4 View Controllers?

UITabBarController



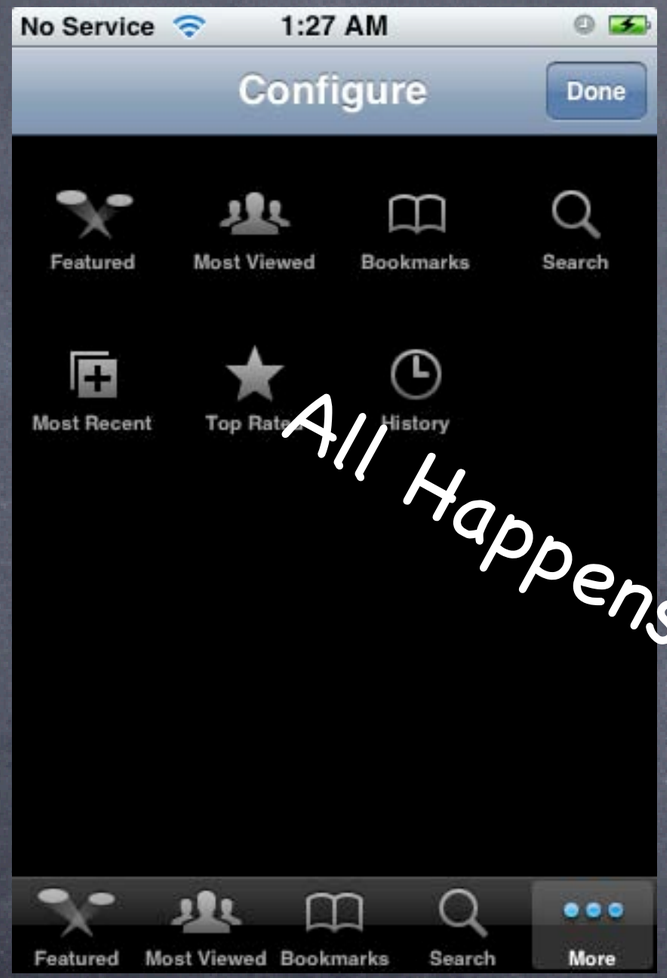
Tab Bar Controller

- View Controller
- View Controller
- View Controller
- View Controller
- View Controller
- View Controller
- View Controller

More button brings up a UI to let the user edit which buttons appear on bottom row

A More button appears.

UITabBarController



All Happens Automatically

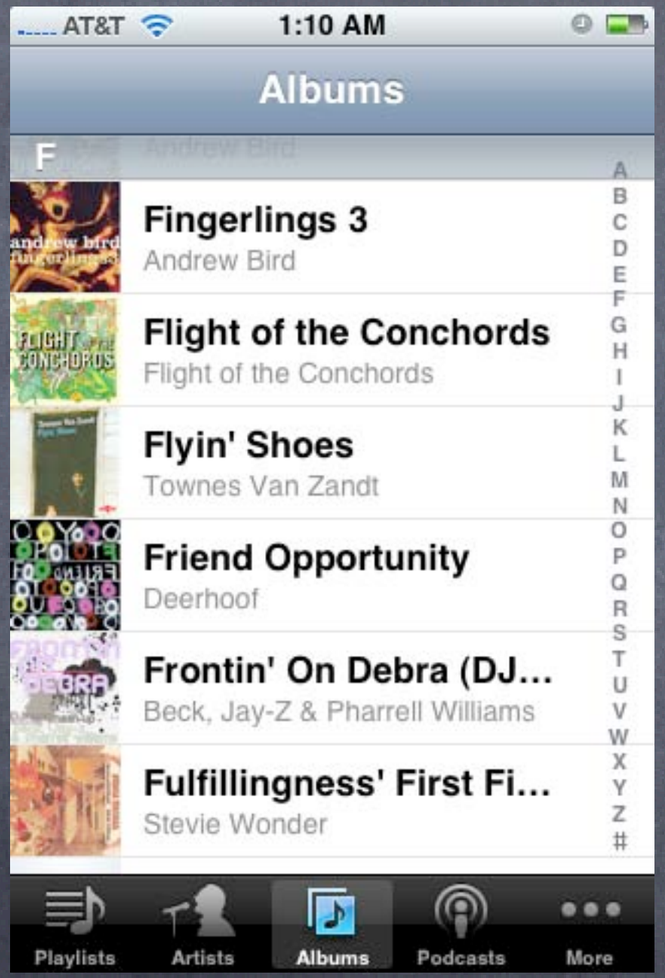
Tab Bar Controller

- View Controller
- View Controller
- View Controller
- View Controller
- View Controller
- View Controller
- View Controller

Combine?

- Can you combine UINavigationController & UITabBarController?
  Certainly. Quite common.
  `UINavigationController` goes "inside" the `UITabBarController`.
  Never the other way around.
- Can you combine UITabBarController and UISplitViewController?
  Less common.
  The `UITabBarController` goes inside the `UISplitViewController` (Master or Detail).

Combine


AT&T
1:10 AM
Albums
F
Fingerlings 3
Andrew Bird
Flight of the Conchords
Flight of the Conchords
Flyin' Shoes
Townes Van Zandt
Friend Opportunity
Deerhoof
Frontin' On Debra (DJ…
Beck, Jay-Z & Pharrell Williams
Fulfillingness' First Fi…
Stevie Wonder
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z #
Playlists
Artists
Albums
Podcasts
More


AT&T
1:12 AM
Search
Info
iStanford
Stanford University
9 reviews
INSTALLED
iStanford is Stanford University in the palm of your hand.
Search the Stanford directory, search campus map, find and bookmark courses, and get scores, schedules, and news for all Stanford varsity athletics teams... All from your iPhone or iPod Touch!
Features:
- Search for buildings / places on campus, and use GPS to find your location
- Search for students, faculty or anybody in the Stanford directory. Tap to call/email. You can even add them to your contacts on your phone!
- Search for classes, find out where and when they're offered, and tap to call or email the professor
Featured
Categories
Top 25
Search
Updates

UINavigationController

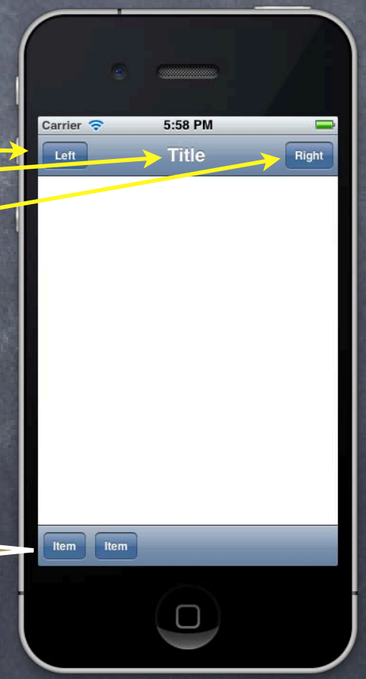
- Modifying buttons and toolbar items in a navigation controller
  You can set most of this up in Xcode by dragging items into your scene.
  But you may want to add buttons or change buttons at run time too …
- UIViewController's navigationItem property

```
@property (nonatomic, strong) UINavigationItem *navigationItem;
```

Think of navigationItem as a holder for things UINavigationController will need when that UIViewController appears on screen.

```
@property (nonatomic, copy) NSArray *leftBarButtonItems;
@property (nonatomic, strong) UIView *titleView;
@property (nonatomic, copy) NSArry *rightBarButtonItems;

// when this UIViewController is not on the top of the UINC stack:
@property (nonatomic, copy) UIBarButtonItem *backButtonItem;
```



These bar button items are not set via the navigationItem.
They are set via the toolbarItems property in UIViewController.

Blocks

- What is a block?

  A block of code (i.e. a sequence of statements inside {}).
  Usually included "in-line" with the calling of method that is going to use the block of code.
  Very smart about local variables, referenced objects, etc.

- What does it look like?

  Here's an example of calling a method that takes a block as an argument.

```
[aDictionary enumerateKeysAndObjectsUsingBlock:^(id key, id value, BOOL *stop) {
    NSLog(@"value for key %@ is %@", key, value);
    if ([@"ENOUGH" isEqualToString:key]) {
        *stop = YES;
    }
}];
```

  This NSLog()s every key and value in aDictionary (but stops if the key is ENOUGH).

- Blocks start with the magical character caret ^

  Then it has (optional) arguments in parentheses, then {, then code, then }.

Blocks

- Can use local variables declared before the block inside the block

```
double stopValue = 53.5;
[aDictionary enumerateKeysAndObjectsUsingBlock:^(id key, id value, BOOL *stop) {
    NSLog(@"value for key %@ is %@", key, value);
    if ([@"ENOUGH" isEqualToString:key] || ([value doubleValue] == stopValue)) {
        *stop = YES;
    }
}];
```

- But they are read only!

```
BOOL stoppedEarly = NO;
double stopValue = 53.5;
[aDictionary enumerateKeysAndObjectsUsingBlock:^(id key, id value, BOOL *stop) {
    NSLog(@"value for key %@ is %@", key, value);
    if ([@"ENOUGH" isEqualToString:key] || ([value doubleValue] == stopValue)) {
        *stop = YES;
        stoppedEarly = YES;  // ILLEGAL
    }
}];
```

Blocks

- Unless you mark the local variable as __block

```
__block BOOL stoppedEarly = NO;
double stopValue = 53.5;
[aDictionary enumerateKeysAndObjectsUsingBlock:^(id key, id value, BOOL *stop) {
    NSLog(@"value for key %@ is %@", key, value);
    if ([@"ENOUGH" isEqualToString:key] || ([value doubleValue] == stopValue)) {
        *stop = YES;
        stoppedEarly = YES;  // this is legal now
    }
}];
if (stoppedEarly) NSLog(@"I stopped logging dictionary values early!");
```

- Or if the "variable" is an instance variable

  But we only access instance variables (e.g. _display) in setters and getters.
  So this is of minimal value to us.

Blocks

- So what about objects which are messaged inside the block?

```
NSString *stopKey = [@“Enough” uppercaseString];
__block BOOL stoppedEarly = NO;
double stopValue = 53.5;
[aDictionary enumerateKeysAndObjectsUsingBlock:^(id key, id value, BOOL *stop) {
    NSLog(@“value for key %@ is %@”, key, value);
    if ([stopKey isEqualToString:key] || ([value doubleValue] == stopValue)) {
        *stop = YES;
        stoppedEarly = YES;  // this is legal now
    }
}];
if (stoppedEarly) NSLog(@“I stopped logging dictionary values early!”);
```

stopKey will essentially have a strong pointer to it until the block goes out of scope or the block itself leaves the heap (i.e. no one points strongly to the block anymore).

Why does that matter?

Blocks

- Imagine we added the following method to `CalculatorBrain`

  `- (void)addUnaryOperation:(NSString *)operation whichExecutesBlock:…;`

  This method adds another operation to the brain like `sqrt` which you get to specify the code for.

  For now, we'll not worry about the syntax for passing the block.

  (but the mechanism for that is the same as for defining `enumerateKeysAndObjectsUsingBlock:`).

- That block we pass in will not be executed until much later

  i.e. it will be executed when that "operation" is pressed in some UI somewhere.

- Example call of this …

```
NSNumber *secret = [NSNumber numberWithDouble:42.0];
[brain addUnaryOperation:@"MoLtUaE" whichExecutesBlock:^(double operand) {
    return operand * [secret doubleValue];
}];
```

  Imagine if `secret` were not automatically kept in the heap here.

  What would happen later when this block executed (when the `MoLtUaE` operation was pressed)?

  Bad things! Luckily, `secret` is automatically kept in the heap until block can't be run anymore.

Blocks

- Creating a "type" for a variable that can hold a block

  Blocks are kind of like "objects" with an unusual syntax for declaring variables that hold them.

  Usually if we are going to store a block in a variable, we typedef a type for that variable, e.g.,

```
typedef double (^unary_operation_t)(double op);
```

  This declares a type called "unary_operation_t" for variables which can store a block.
  (specifically, a block which takes a double as its only argument and returns a double)

  Then we could declare a variable, square, of this type and give it a value …

```
unary_operation_t square;
square = ^(double operand) { // the value of the square variable is a block
    return operand * operand;
}
```

  And then use the variable square like this …

```
double squareOfFive = square(5.0);  // squareOfFive would have the value 25.0 after this
```

  (You don't have to typedef, for example, the following is also a legal way to create square …)

```
double (^square)(double op) = ^(double op) { return op * op; };
```

Blocks

- We could then use the unary_operation_t to define a method

  For example, addUnaryOperation:whichExecutesBlock:

  We'd add this property to our CalculatorBrain …

```
@property (nonatomic, strong) NSMutableDictionary *unaryOperations;
```

  Then implement the method like this …

```
typedef double (^unary_operation_t)(double op);
- (void)addUnaryOperation:(NSString *)op whichExecutesBlock:(unary_operation_t)opBlock {
    [self.unaryOperations setObject:opBlock forKey:op];
}
```

  Note that the block can be treated somewhat like an object (e.g., adding it to a dictionary).

  Later in our CalculatorBrain we could use an operation added with the method above like this …

```
- (double)performOperation:(NSString *)operation
{
    unary_operation_t unaryOp = [self.unaryOperations objectForKey:operation];
    if (unaryOp) {
        self.operand = unaryOp(self.operand);
    }
    ...
}
```

Blocks

- We don't always `typedef`

  When a block is an argument to a method and is used immediately, often there is no `typedef`.

  Here is the declaration of the dictionary enumerating method we showed earlier …

```
- (void)enumerateKeysAndObjectsUsingBlock:(void (^)(id key, id obj, BOOL *stop))block;
```

  No "name" for the type appears here.

  The syntax is exactly the same as the `typedef` except that the name of the `typedef` is not there.

  For reference, here's what a `typedef` for this argument would look like this …

```
typedef void (^enumeratingBlock)(id key, id obj, BOOL *stop);
```

  (i.e. the underlined part is not used in the method argument)

  This ("`block`") is the keyword for the argument (e.g. the local variable name for the argument inside the method implementation).

Blocks

- Some shorthand allowed when defining a block

  ("Defining" means you are writing the code between the {}.)

  1. You do not have to declare the return type if it can be inferred from your code in the block.
  2. If there are no arguments to the block, you do not need to have any parentheses.

  Recall this code …

```
NSNumber *secret = [NSNumber numberWithDouble:42.0];
[brain addUnaryOperation:@"MoLtUaE" whichExecutesBlock:^(double operand) {
    return operand * [secret doubleValue];
}];
```

No return type. Inferred from the return inside.

Blocks

- Some shorthand allowed when defining a block

  ("Defining" means you are writing the code between the {}.)

  1. You do not have to declare the return type if it can be inferred from your code in the block.
  2. If there are no arguments to the block, you do not need to have any parentheses.

  Recall this code …

```
NSNumber *secret = [NSNumber numberWithDouble:42.0];
[brain addUnaryOperation:@"MoLtUaE" whichExecutesBlock:^(double operand) {
    return operand * [secret doubleValue];
}];
```

- Another example …

```
[UIView animateWithDuration:5.0 animations:^{
    view.opacity = 0.5;
}];
```

No arguments to this block.
No need to say ^() { … }.

Blocks

- Memory Cycles (a bad thing)

  What if you had the following property in a class?

```
@property (nonatomic, strong) NSArray *myBlocks; // array of blocks
```

  And then tried to do the following in one of that class's methods?

```
[self.myBlocks addObject:^() {
    [self doSomething];
}];
```

  We said that all objects referenced inside a block will stay in the heap as long as the block does.
  (in other words, blocks keep a strong pointer to all objects referenced inside of them)

  In this case, self is an object reference in this block.

  Thus the block will have a strong pointer to self.

  But notice that self also has a strong pointer to the block (through its myBlocks property)!

  This is a serious problem.

  Neither self nor the block can ever escape the heap now.

  That's because there will always be a strong pointer to both of them (each other's pointer).

  This is called a memory "cycle."

Blocks

- Memory Cycles Solution

You'll recall that local variables are always strong.
That's okay because when they go out of scope, they disappear, so the strong pointer goes away.
But there's a way to declare that a local variable is weak. Here's how ...

```
__weak MyClass *weakSelf = self;
[self.myBlocks addObject:^() {
    [weakSelf doSomething];
}];
```

This solves the problem because now the block only has a weak pointer to self.
(self still has a strong pointer to the block, but that's okay)
As long as someone in the universe has a strong pointer to this self, the block's pointer is good.
And since the block will not exist if self does not exist (since myBlocks won't exist), all is well!

If you are struggling to understand this, don't worry, you will not have to create blocks that refer to self in any of your homework assignments this quarter.

Blocks

- When do we use blocks in iOS?

  Enumeration

  View Animations (more on that later in the course)

  Sorting (sort this thing using a block as the comparison method)

  Notification (when something happens, execute this block)

  Error handlers (if an error happens while doing this, execute this block)

  Completion handlers (when you are done doing this, execute this block)

- And a super-important use: Multithreading

  With Grand Central Dispatch (GCD) API

Grand Central Dispatch

- GCD is a C API
- The basic idea is that you have queues of operations
  - The operations are specified using blocks.
  - Most queues run their operations serially (a true "queue").
  - We're only going to talk about serial queues today.
- The system runs operations from queues in separate threads
  - Though there is no guarantee about how/when this will happen.
  - All you know is that your queue's operations will get run (in order) at some point.
  - The good thing is that if your operation blocks, only that queue will block.
  - Other queues (like the main queue, where UI is happening) will continue to run.
- So how can we use this to our advantage?
  - Get blocking activity (e.g. network) out of our user-interface (main) thread.
  - Do time-consuming activity concurrently in another thread.

Grand Central Dispatch

- Important functions in this C API

  Creating and releasing queues

```
dispatch_queue_t dispatch_queue_create(const char *label, NULL); // serial queue
void dispatch_release(dispatch_queue_t);
```

  Putting blocks in the queue

```
typedef void (^dispatch_block_t)(void);
void dispatch_async(dispatch_queue_t queue, dispatch_block_t block);
```

  Getting the current or main queue

```
dispatch_queue_t dispatch_get_current_queue();
void dispatch_queue_retain(dispatch_queue_t); // keep it in the heap until dispatch_release

dispatch_queue_t dispatch_get_main_queue();
```

Grand Central Dispatch

- What does it look like to call these?

Example … assume we fetched an image from the network (this would be slow).

```
- (void)viewWillAppear:(BOOL)animated
{
    NSData *imageData = [NSData dataWithContentsOfURL:networkURL];
    UIImage *image = [UIImage imageWithData:imageData];
    self.imageView.image = image;
    self.imageView.frame = CGRectMake(0, 0, image.size.width, image.size.height);
    self.scrollView.contentSize = image.size;
}
```

Grand Central Dispatch

- What does it look like to call these?

Example … assume we fetched an image from the network (this would be slow).

```
- (void)viewWillAppear:(BOOL)animated
{


        NSData *imageData = [NSData dataWithContentsOfURL:networkURL];
        UIImage *image = [UIImage imageWithData:imageData];
        self.imageView.image = image;
        self.imageView.frame = CGRectMake(0, 0, image.size.width, image.size.height);
        self.scrollView.contentSize = image.size;

}
```

Grand Central Dispatch

- What does it look like to call these?

Example … assume we fetched an image from the network (this would be slow).

```
- (void)viewWillAppear:(BOOL)animated
{
    dispatch_queue_t downloadQueue = dispatch_queue_create("image downloader", NULL);

        NSData *imageData = [NSData dataWithContentsOfURL:networkURL];
        UIImage *image = [UIImage imageWithData:imageData];
        self.imageView.image = image;
        self.imageView.frame = CGRectMake(0, 0, image.size.width, image.size.height);
        self.scrollView.contentSize = image.size;

}
```

Grand Central Dispatch

- What does it look like to call these?

Example … assume we fetched an image from the network (this would be slow).

```
- (void)viewWillAppear:(BOOL)animated
{
    dispatch_queue_t downloadQueue = dispatch_queue_create("image downloader", NULL);
    dispatch_async(downloadQueue, ^{
        NSData *imageData = [NSData dataWithContentsOfURL:networkURL];
        UIImage *image = [UIImage imageWithData:imageData];
        self.imageView.image = image;
        self.imageView.frame = CGRectMake(0, 0, image.size.width, image.size.height);
        self.scrollView.contentSize = image.size;
    });
}
```

Grand Central Dispatch

- What does it look like to call these?

Example … assume we fetched an image from the network (this would be slow).

```
- (void)viewWillAppear:(BOOL)animated
{
    dispatch_queue_t downloadQueue = dispatch_queue_create("image downloader", NULL);
    dispatch_async(downloadQueue, ^{
        NSData *imageData = [NSData dataWithContentsOfURL:networkURL];
        UIImage *image = [UIImage imageWithData:imageData];
        self.imageView.image = image;
        self.imageView.frame = CGRectMake(0, 0, image.size.width, image.size.height);
        self.scrollView.contentSize = image.size;
    });
}
```

Problem! UIKit calls can only happen in the main thread!

Grand Central Dispatch

- What does it look like to call these?

Example … assume we fetched an image from the network (this would be slow).

```
- (void)viewWillAppear:(BOOL)animated
{
    dispatch_queue_t downloadQueue = dispatch_queue_create("image downloader", NULL);
    dispatch_async(downloadQueue, ^{
        NSData *imageData = [NSData dataWithContentsOfURL:networkURL];

            UIImage *image = [UIImage imageWithData:imageData];
            self.imageView.image = image;
            self.imageView.frame = CGRectMake(0, 0, image.size.width, image.size.height);
            self.scrollView.contentSize = image.size;

    });
}
```

Grand Central Dispatch

- What does it look like to call these?

Example … assume we fetched an image from the network (this would be slow).

```
- (void)viewWillAppear:(BOOL)animated
{
    dispatch_queue_t downloadQueue = dispatch_queue_create("image downloader", NULL);
    dispatch_async(downloadQueue, ^{
        NSData *imageData = [NSData dataWithContentsOfURL:networkURL];
        dispatch_async(dispatch_get_main_queue(), ^{
            UIImage *image = [UIImage imageWithData:imageData];
            self.imageView.image = image;
            self.imageView.frame = CGRectMake(0, 0, image.size.width, image.size.height);
            self.scrollView.contentSize = image.size;
        });
    });
}
```

Grand Central Dispatch

- What does it look like to call these?

Example … assume we fetched an image from the network (this would be slow).

```
- (void)viewWillAppear:(BOOL)animated
{
    dispatch_queue_t downloadQueue = dispatch_queue_create("image downloader", NULL);
    dispatch_async(downloadQueue, ^{
        NSData *imageData = [NSData dataWithContentsOfURL:networkURL];
        dispatch_async(dispatch_get_main_queue(), ^{
            UIImage *image = [UIImage imageWithData:imageData];
            self.imageView.image = image;
            self.imageView.frame = CGRectMake(0, 0, image.size.width, image.size.height);
            self.scrollView.contentSize = image.size;
        });
    });
}
```

Problem! This "leaks" the `downloadQueue` in the heap. We have to `dispatch_release` it.

Grand Central Dispatch

- What does it look like to call these?

Example … assume we fetched an image from the network (this would be slow).

```
- (void)viewWillAppear:(BOOL)animated
{
    dispatch_queue_t downloadQueue = dispatch_queue_create("image downloader", NULL);
    dispatch_async(downloadQueue, ^{
        NSData *imageData = [NSData dataWithContentsOfURL:networkURL];
        dispatch_async(dispatch_get_main_queue(), ^{
            UIImage *image = [UIImage imageWithData:imageData];
            self.imageView.image = image;
            self.imageView.frame = CGRectMake(0, 0, image.size.width, image.size.height);
            self.scrollView.contentSize = image.size;
        });
    });
    dispatch_release(downloadQueue);
}
```

Don't worry, it won't remove the queue from the heap until all blocks have been processed.

Demo

- Table View
  Another example
- Blocks
  Using a block-based API (searching for objects in an array)
- GCD
  Using blocks and GCD to improve user-interface responsiveness
- Spinner (time permitting)
  How to show a little spinning wheel when the user is waiting for something to happen
- UITabBarController (time permitting)
  Just going to briefly show how to hook it up in Xcode.

Coming Up

- Next Lecture

  Persistence

  Other stuff :)

- Section

  No section this week.