

# Introduction to BLOCKS

- Was introduced iOS 4 in 2010
- To group code together into reusable chunks
- Isn't that what functions are for?
- Then why use blocks?

# Why Use BLOCKS

- Many of the newer frameworks use blocks extensively (GDC, new Audio and video, gameKit)
- It makes your code easier to read and reuse
- It allows you to perform advance tasks easier(multithreading, call backs)
- Anonymous function, lamda , in other language

# How to Create BLOCKS

return type	name	arguments	arguments
int	myFirstBlock	(int)	= ^(int a) { // code goes here };

Return      Name      arguments

```
int myFuntion(int a){  
    //codes goes here  
}
```

```
int mynumber = myFuntion(20);
```

# How to create BLOCKS

return type	name	arguments	arguments
<code>int</code>	<code>(^myFirstBlock)</code>	<code>(int)</code>	<code>= ^ (int a) {</code>
	<code>// code goes here</code>		
	<code>};</code>		

## block variable

## block literal

```
int (^myFirstBlock) (int) = ^(int a) {  
    // code goes here  
};
```

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

# BLOCKS

- Block Variable means we are actually naming it like a variable, it has a return type and parameters.
- Block literal is actually what it does and the important part

# BLOCKS can be passed as arguments

This is useful when a method call is being done in another thread and you  
Don't know when the it will be done

```
- (void)writeVideoAtPathToSavedPhotosAlbum:(NSURL *)videoPathURL
    completionBlock:(ALAssetsLibraryWriteVideoCompletionBlock)completionBlock

[myLibrary writeVideoAtPathToSavedPhotosAlbum:myAsset
    completionBlock:^(NSURL *url, NSError *error) {
        if ( error == nil ) {
            NSLog(@"It worked!"); }
        else {
            // error handling goes here
        }
    }

];
```

**block variable**

**block literal**

```
int (^myFirstBlock) (int) = ^(int a) {  
    // code goes here  
};
```

WE can usually dump the the first part and just use the the block literal



# Dictionary Enumerate

```
- (IBAction)enumthis:(id)sender {  
    NSDictionary *aDictionary = [self Apptizers];  
    [aDictionary enumerateKeysAndObjectsUsingBlock:  
        ^(id key, id obj, BOOL *stop) {  
            NSLog(@"the key is %@ the value is %@", key, obj);  
        }];  
}
```

# Block and its Scope

You can access data from outside of the block BUT  
**THIS IS READ ONLY UNLESS** the variable is  
marked as `__block`

```
- (IBAction)enumthis:(id)sender {
    NSDictionary *aDictionary = [self Apptizers];
    NSString * outsideValue = @"I am out side the block";
    [aDictionary enumerateKeysAndObjectsUsingBlock:
     ^(id key, id obj, BOOL *stop) {
         NSLog(@"the key is %@ the value is %@ the ouside Value is %@",
              , key, obj, outsideValue);
     }];
}
```

# Introduction to GRAND CENTRAL DISPATCH

- Intruded in ios 4
- C based API (functions are C style not objective-C)
- “*Grand Central Dispatch (GCD) comprises language features, runtime libraries, and system enhancements that provide systemic, comprehensive improvements to the support for concurrent code execution on multicore hardware in iOS and OS X.*” OKAAAY!!!???

# Introduction to GRAND CENTRAL DISPATCH

- It makes multithreading easy
- **MORE REPOSIVE UI**
- Multithreading not multitasking...

# How to use GCD

- 1. Create a new queue.
- 2. Add blocks to the Queue.

*That's it!*

*Well most of time*

# Create a New Queue

```
dispatch_queue_create(const char *label,  
    dispatch_queue_attr_t attr)  
  
dispatch_queue_t myQueue =  
dispatch_queue_create("QueueName", NULL);
```

**QueueName** is just the name and we use it for debugging.

Notice **NULL** is not **nil**, it is set not NULL for Serial Queue for the purpose of this course

# Add Blocks to the Queue

```
dispatch_async(dispatch_queue_t queue,  
               ^(void)block)
```

```
dispatch_async(QueueName,  
               ^{[self myMethodThatTakesLong];});
```

# Another Important Concept

- Getting the current or main Queue

```
dispatch_get_main_queue();
```

```
dispatch_get_current_queue();
```



# Lab 3



## ImageViewer:

UI:

2 buttons

1 next

1 prev

1 imageview controll



```
NSURL *url = [NSURL URLWithString: imgUrl];  
NSData *dataOfImg = [NSData dataWithContentsOfURL:url];  
self.imageView.image = [UIImage imageData:dataOfImg];
```

This is how you can get an image from a web server

Using GCD use multi threading to get the images from the web server

Make sure anything with UI happens on Main thread

There should be no freezing of User Interface what so ever

If you like you can look at UIActivityIndicatorView when image is being Loaded (not necessary for this lab).

## Lab 3



User clicks on next the next image will show

User clicks on prev the previous image will show

-Use <https://www.freeimages.com/>


-Right click on the image you like and click copy image address

-At least 5 of these addresses should be in your model array as literal strings

-your app will fetch and showing these images

- Today
- iOS File System – Quick overview
- UserDefaults – For App data < 100K
- Property List – For Property Data Types
- Core Data –

# What is Data Persistence ?

**per·sist·ence** /pər'sistəns/ 

Noun:

1. Firm or obstinate continuance in a course of action in spite of difficulty or opposition.
2. The continued or prolonged existence of something.

3. The continued or prolonged existence of something.

- In Computer Science, persistence refers to the characteristic of state that outlives the process that created it.
- Basically, generated data that sticks around after our App is terminated

- **iOS File System**

- Unix File system (WITH A ROOT)
- File System Protection
- **Can not** read every thing from the file system
- Your app only has permission to write to it's **own Sandbox**

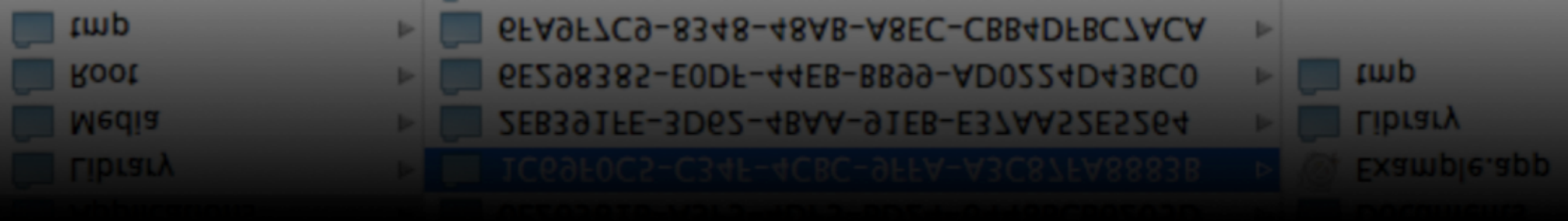
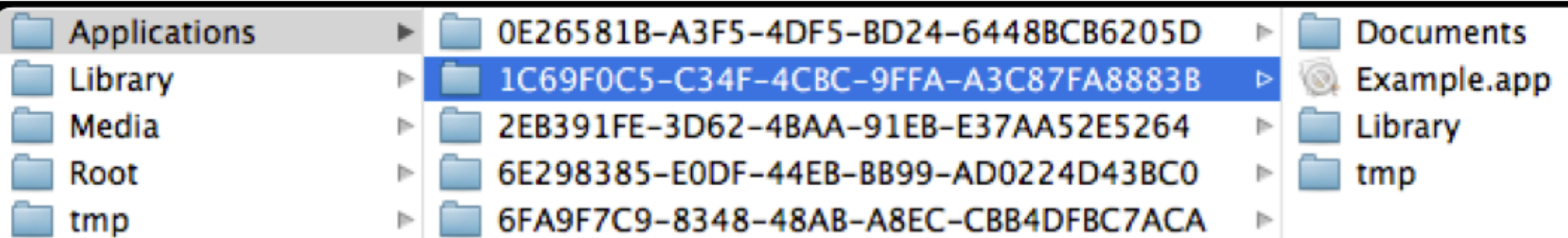
**why ?**

– security ! privacy ! cleanup !

# What is in the App's Sandbox

- **Application bundle directory**
  - PNGs, Story board. NOT writeable
  - Read only copy of the application
- **Documents Directory**
  - Store things that the user creates, backed up
  - Remains after update
- **Caches Directory**
  - This is where you can write out temporary files that you don't care if deleted

# Peek into the App's Sandbox





# Accessing Files

## NSFileManager

- Provides utilities functions to create, delete, check existence of files and returns paths to directories. Not used to read and write data

```
NSFileManager* fm = [NSFileManager defaultManager];
```

```
NSArray* docArray =  
[fm URLsForDirectory:NSDocumentDirectory inDomains:NSUserDomainMask];
```

```
NSURL* document = [docArray lastObject];
```

```
NSURL* textDocument =  
[document URLByAppendingPathComponent:@"MyFile.txt" isDirectory:NO];
```

```
if ([fm fileExistsAtPath:[textDocument path]])  
    NSLog(@"File Exist");
```



# Domain Masks

- **NSUserDomainMask**
  - The user's home directory—the place to install user's personal items (~).  
in iOS
- **NSLocalDomainMask**
  - Local to the current machine—the place to install items available to everyone on this machine.
- **NSNetworkDomainMask**
  - Publicly available location in the local area network—the place to install items available on the network (/Network).
- **NSSystemDomainMask**
  - Provided by Apple—can't be modified (/System) .

## Type of directory

- NSDocumentsDirectory, NSChashesDirectory, etc

# NSUserDefaults

- Allows you to read and write small data into AppID.plist located in the Preferences folder
- Caches information to avoid having to open the user's default database
- Memory and File system are synched periodically or manually by calling synchronize
- NSData, NSString, NSNumber, NSDate, NSArray, NSDictionary

# Read and Write

- Retrieve from Defaults

```
NSUserDefaults *defaults = [NSUserDefaults  
standardUserDefaults];  
NSString* firstName = [defaults objectForKey:@"firstName"];  
int age = [defaults integerForKey:@"age"];  
NSString* ageString = [NSString stringWithFormat:@"%d",age];
```

- Write to Defaults

```
NSUserDefaults *defaults = [NSUserDefaults  
standardUserDefaults];  
[defaults setObject:firstName forKey:@"firstName"];  
[defaults setInteger:age forKey:@"age"];  
[defaults synchronize];
```

# Class Exercise

- Your app should have 3 text fields and labels to display them
  - Name
  - Last name
  - Age

Another label that displays the url path to document folder

Save them user default and recover it on labels  
When app is re launched

# Property List

- A little step up from `NSUserDefaults`
- In `NSArray` and `NSDictionary` you can use  
`writeToURL:atomically:`  
`initWithContentsOfURL:`
- You can store larger object then  
`NSUserDefaults`
- App needs to handle file corruption, deletion, schema changes, etc..