HACK YALE

/* iOS Development */

www.hackyale.com

HACK YALE

/*iOS Development*/

OBJECTIVE-C

An Introduction

A DOSE OF VITAMIN [OBJECTIVE-]C

HACK YALE

WHY DO WE CARE?

- Objective-C : iOS :: Java : Android
- Apple owns over 20% of the smartphone market
- Apple owns 58% of the tablet marketshare
  - sold 15 million iPads in Q4 2011
- Objective-C gained 3% of overall programming market in 2011

IN CONCLUSION

“Idea + Design + Objective-C
== Huge Market Opportunity

QUICK NOTE ON HOW I WILL RUN THINGS

All code samples will be in the slides so you can have them to look at in context, but we will try to actually write all of it as we go.

“HELLO, WORLD”

HACK YALE

HELLO WORLD

```
int main(int argc, char *argv[])
{
    printf("Hello, World!");

    return 0;
}
```

WAIT, HOLD UP, THAT'S C

HACK YALE

OBJECTIVE-C IS...

- A superset of C
  - all C code is legal Objective-C
- Object-Oriented
  - classes, delegates, protocols
- Really ~~Ugly???~~ Beautiful

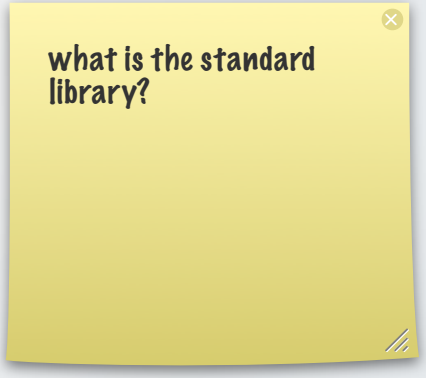
SYNTAX 101

variables and types

HACK YALE

NS? STANDARD LIBRARY? WHAT?

- Naming: NS_____
  - come's from NeXTSTEP, which Apple bought
  - introduced in MacOSX 10+
- Basic things you can do:
  - alloc, init, copy
  - description
  - getters, setters



QUICK HITS

- NSString
- NSArray
  - objectAtIndex:
- NSDictionary
  - addObject:forKey:
  - objectForKey:

EXAMPLE: DECLARING VARIABLES

```
NSString *myStr1, *myStr2;
NSArray *myArray;
NSInteger *myInt = 6; //one of few types
you can initialize directly
NSUInteger *myUnsignedInt = 6;
```

MUTABLE COUNTERPARTS

- NSMutableString, NSMutableArray, NSMutableDictionary
- subclasses of static versions
  - can hold it in a non-mutable pointer without losing features of subclass

EXAMPLE: MUTABLE OBJECTS

```
  NSString *myStr = [[NSString alloc] initWithString:@"Hello."];
int len = [myStr length];
//==> 6
char c = [myStr characterAtIndex:5]
//==> .
NSRange range = [myStr rangeOfString:@"el"];
//==> {1,2}
[myStr setString:@"This is an NSString.  It is different from a
string in C because it is an object."];
//==> error

NSMutableString *myMutableStr = [[NSMutableString alloc]
initWithString:@"What's up?"];
[myMutableStr setString:@"This is a NSMutableString.  This is
actually legal because the memory is mutable."];
```

WHAT'S THE BEST WAY TO LEARN ALL THE STANDARD LIBRARY FUNCTIONS?

"Don't memorize. Use the documentation instead. Seriously.

OBJECTIVE-C FUNCTION SYNTAX 101

- Selectors
  - @functionName
  - There is no such overloaded function
- Messages
  - [receiver selectorWithParam1:a Param2:b...]
  - Nested calls
    - [[receiver getFoo] doFunnyThingsToFoo]

show, don't tell: have a simple code example and then explain what part of the code is a selector and vice versa

HACK YALE

EXAMPLES

```
NSString *aString = [[NSString alloc] initWithString:@"Daniel"];

NSInteger *rectArea = [myRectangle area];
NSString *myName = [daniel getName];

//Nesting
NSInteger *myGpa = [[Yale getClassByYear:2014] getStudents]
lookupStudentByName:@"Daniel"] getGpa];

etc...
```

SYNTAX 101

Controls & Pointers

HACK YALE

CONTROL STRUCTURES

- Basic control features – similar to C, C++, Java
  - conditionals: if, switch
  - loops: for, while

POINTERS!!!!

- stores a memory location
- you will use them for everything
  - however, they're not actually that scary
- NOTE: nil vs. NULL

ON A BROADER LEVEL...

Objective-C does not have pointer dereferencing with objects. It is built into the language. This has a major impact on how memory management works.

OBJECT-ORIENTED FEATURES

HACK YALE

CLASS STRUCTURES

- subclassing
  - MyFoo :: NSFoo
- self, super
  - use when overriding/subclassing methods
  - [super viewWillAppear]
  - very common idiom, particularly when you make custom user interfaces

METHODS EVERY CLASS HAS

- init
  - can take parameters, but then it would be a different selector
- dealloc
  - don't need to implement it with ARC, except when free-ing malloc-ed data
- description - how it prints as a string
- getters, setters (aka properties)

BASIC EXAMPLE

```
-(id)initWithName:(NSString*)name
{
    self = [super init];
    if (self) {
        self.birthName = name;
    }
    return self;
}

-(void)dealloc
{
    [myName release];
    [super dealloc];
}
```

HACK YALE

GETTERS AND SETTERS (AND WHY YOU [ALMOST] NEVER HAVE TO WRITE THEM)

- @property (nonatomic, strong) type* name
- Syntactic Sugar
  - Dot Notation
  - self.name = “Hell [self setName]
  - [self.array myFunction]

properties for public variables, private declare yourself; or just getters

BASIC CODE STRUCTURE

- Main Method (you'll never touch it)
- Using other people's (or your own) code:
  - #import <libraryfilename>
  - #import "customsourcecode.h"
  - NOTE: not #include, although technically it's valid

CODE STRUCTURE, CONT.

- The code for every class is divided into two files:
  - header file – @interface
    - protocols, properties, instance variables
    - @class tags
  - source code file – @implementation, #import header

BRING IT TOGETHER: SAMPLE HEADER FILE

```
   #import <Cocao/Cocao.h>

@interface Person : NSObject
{
   NSString* birthName;
   Person* mother, father;
}

@property (nonatomic, strong, readonly) NSString* birthName;
@property (nonatomic, strong) NSMutableString* nickname;
@property (nonatomic, weak, readonly) Person* mother, father;
//match with @synthesize mother, father, etc in .m file

-(id)initWithName:(NSString*)name mother:(Person*)mom father:(Person*)dad;

@end
```

SAMPLE .M FILE

```
#import "Person.h"

@implementation Person

@synthesize birthName, nickName mother, father;

-(id)initWithName:(NSString*)name mother:(Person*)mom father:(Person*)dad
{
    self = [super init];
    if (self) {
        birthName = [name copy];
        mother = mom;
        father = dad;
    }
    return self;
}

-(void)dealloc
{
    [myName release];
    [super dealloc];
}
@end
```

HACK YALE

TYPING AND POLYMORPHISM

- Objective-C can be dynamically or statically typed
- Every class derives the class NSObject
  - isa pointer – pointer to Class that created the object
  - “introspection”
- What this means...
  - functions can be type agnostic

POLYMORPHISM EXAMPLE

```
   NSMutableString* myMutableString = [[NSMutableString alloc]
   initWithString:@"HackYale is awesome"];
NSString* myString = myMutableString;
NSRange myRange = {8, 11};
//okay, but compiler will say that the object "may not respond"
[myString deleteCharactersInRange:myRange];
//better
[(NSMutableString*)myString deleteCharactersInRange:myRange];
// => "HackYale";

/* NOTE
 typedef struct _NSRange {
 NSUInteger location;
 NSUInteger length;
 } NSRange;
 */
```

HACK YALE

SOME MORE ADVANCED CONCEPTS

CLASS VS INSTANCE FUNCTIONS

- Instance methods
  - declaration preceded by a –
  - can only operate on an instance of a class
    - getters, setters, etc.
- Class methods
  - declaration preceded by a +
  - can be used with class names
  - good for utility functions (e.g. converting one type to another) that don't rely on specific data

PROTOCOLS

- Similar to interfaces in Java
- Declaration:
  - @optional, @required
- Use MyClass : NSObject <MyProtocol>
  - must implement required methods
- Can be used to overwrite Library functionality

THE PLEASURES OF COUNTING

HACK YALE

MEMORY MANAGEMENT

- Don’t need to worry about malloc() and free()
- Up to iOS 4+: Manual Reference Counting
- Now: Automatic Reference Counting

DEEP COPYING VS. SHALLOW COPYING (AND STRONG OWNERSHIP VS. WEAK)

- yourObject = [myObject copy]
- When you synth operty (i.e. make the getters and r instance variables), you r re whether it is to be “strong”ly or “weak”ly owned
  - Strong – setter makes a deep copy
  - Weak – setter makes a shallow copy
    - i.e. just gets a pointer reference

a diagram of this may be helpful?

HOW DOES REFERENCE COUNTING WORK?

- Children metaphor/shopping list
- Any time you allocate a block of memory through alloc, copy, etc, the retain count gets incremented
- Any time you deallocate an object, the reference count [hopefully] gets decremented

AN EXAMPLE

```
  NSArray* myArr = [[NSArray alloc] init];
NSLog(“%d”, [myArr retainCount]) // => 1

NSArray* yourArr = myArr; //shallow copy
[yourArr retain]; // retainCount = 2;

[myArr release]; // retainCount = 1;
//by the time you reach here, you’ve forgetten you retained the
array a second time ==> MEMORY LEAK!!!

//How do we correct it?
```

OTHER PITFALLS

- Hidden “retains” – copy
- Sometimes “constructors” don’t allocate memory
  - [NSURL URLWithString:myString] returns a class variable, not a newly allocated one
    - useful when passing it to something as an intermediate step
- Returning variables and autorelease
  - don’t have same control over objects, because they are all allocated from the heap

THE GOOD NEWS?

You don’t have to worry about it.

SO WHY DID I BOTHER?

You should know what's going on under the hood to help make your programs more efficient.

THE MORAL?

Memory Matters. Use it well.

LET'S TRY AN EXAMPLE

HACK YALE

OKAY, I LIED

This code is not in the slides. But you will be able to find it on the course page.