
AvenueCode

Objective-C

Alexandre Santos @ Avenue Code

Agenda

- Introduction
- Basic Features
  - Classes, Categories, Class Extensions and Protocols
- Some Conventions
- Cocoa and Cocoa Touch
- Memory Management
- Blocks
- Modern Syntax
- Questions

Introduction

- Created in the 1980's
- Object-oriented programming language
  - Superset of C.
  - Smalltalk-style object syntax.
  - Message passing.
  - Static and Dynamic typing.
- Headers (.h) and Implementations (.m)

Classes

XYZPerson.h

```
@interface XYZPerson : NSObject

@property (nonatomic, strong) NSString *firstName;
@property (nonatomic, strong) NSString *lastName;

- (void) sayHello;
- (void) saySomething:(NSString *)greeting;

@end
```

XYZPerson.m

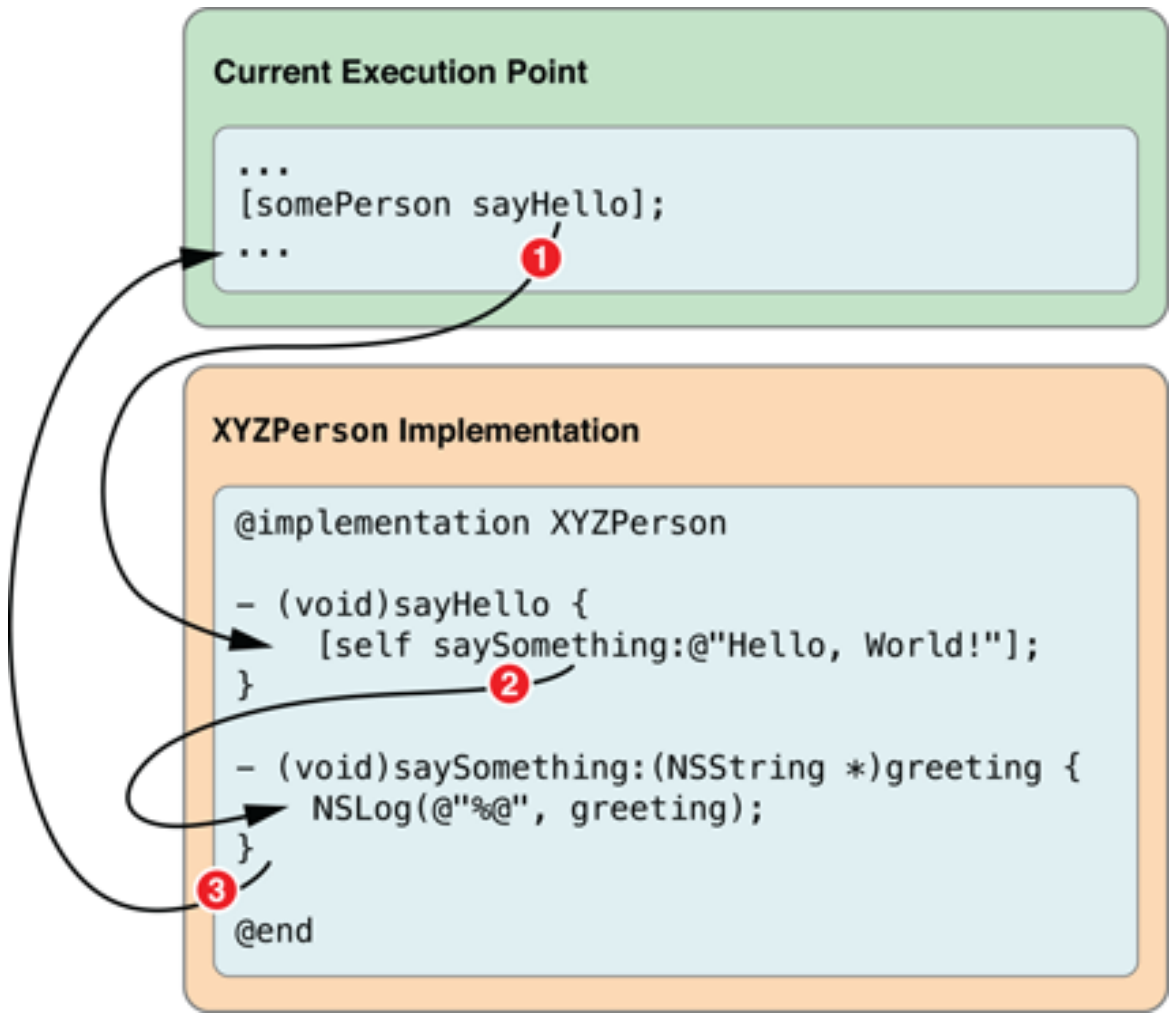
```
@implementation XYZPerson

- (void)sayHello {
    [self saySomething:@"Hello, world!"];
}

- (void)saySomething:(NSString *)greeting {
    NSLog(@"%@", greeting);
}

@end
```

Object Instantiation

- Created dynamically
  - Use pointers
  - id is a generic object pointer
- + (id) alloc;
  - Allocate enough memory for the object
- - (id) init;
  - Initialize the object

```
XYZPerson *somePerson = [[XYZPerson alloc] init];
```

Messaging Flow



Messaging program flow

Categories

- Add methods to existing classes

```
@interface XYZPerson (XYZPersonNameDisplayAdditions)
- (NSString *)lastNameFirstNameString;
@end
```

```
@implementation XYZPerson (XYZPersonNameDisplayAdditions)
- (NSString *)lastNameFirstNameString {
    return [NSString stringWithFormat:@"%@, %@", self.lastName, self.firstName];
}
@end
```

Class Extensions

- Extend the Internal Implementation
- Can only be added to a class that you have the source code at compile time
- Can add properties
- Usually used to hide private information

```
@interface XYZPerson ()
@property (nonatomic, strong) NSObject *extraProperty;
@end
```

Protocols

- Define messaging contracts
  - Similar to Java interfaces
  - Allow @optional methods

Protocol definition

```
@protocol XYZBreathing

@required // Default
- (void) breath;

@optional
- (void) holdBreath;

@end
```

Class conforming to a Protocol

```
@interface XYZPerson : NSObject <XYZBreathing>

@property (nonatomic, strong) NSString *firstName;
@property (nonatomic, strong) NSString *lastName;

- (void) sayHello;
- (void) saySomething:(NSString *)greeting;

@end
```

Some Conventions

- Use camel case
  - Class: MyClass
  - Method: doSomething
- Use prefix to prevent conflicting class names
  - e.g.: UIView and NSString
- Method names should be expressive and unique within a class
  - e.g.: stringByReplacingOccurrencesOfString:withString:
- Object creation method names must follow conventions
  - e.g. initWithString:, stringWithFormat:

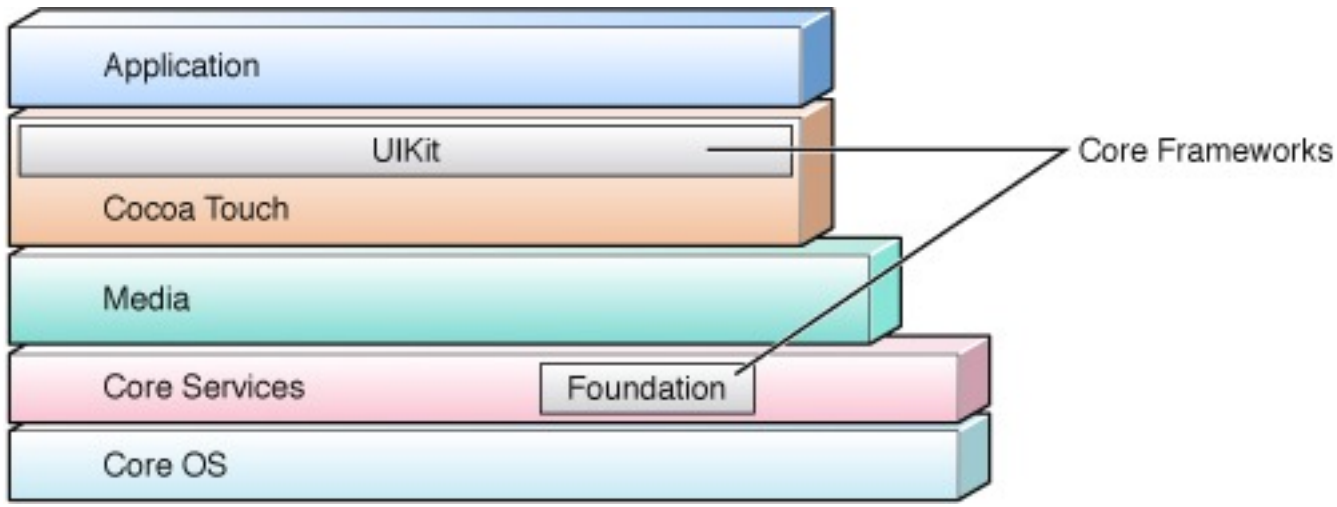
Some Conventions

- Exceptions
  - Don't use exceptions for flow control
  - Use only to indicate programmer error
  - Use NSError ** to indicate errors

Cocoa and Cocoa-Touch

- Application environments for OS X and iOS.
  - Cocoa for OS X
  - Cocoa-Touch for iOS
- MVC design pattern

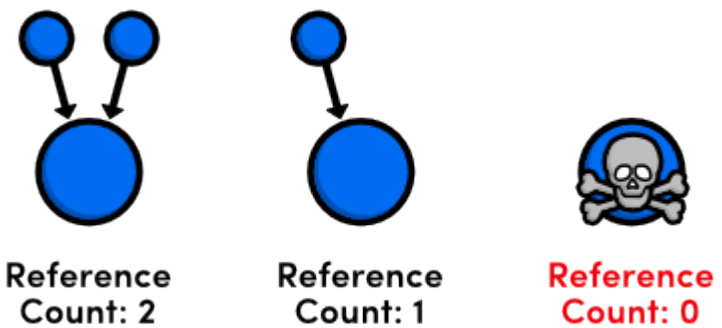
Cocoa-Touch



Cocoa in the architecture of iOS

Memory Management

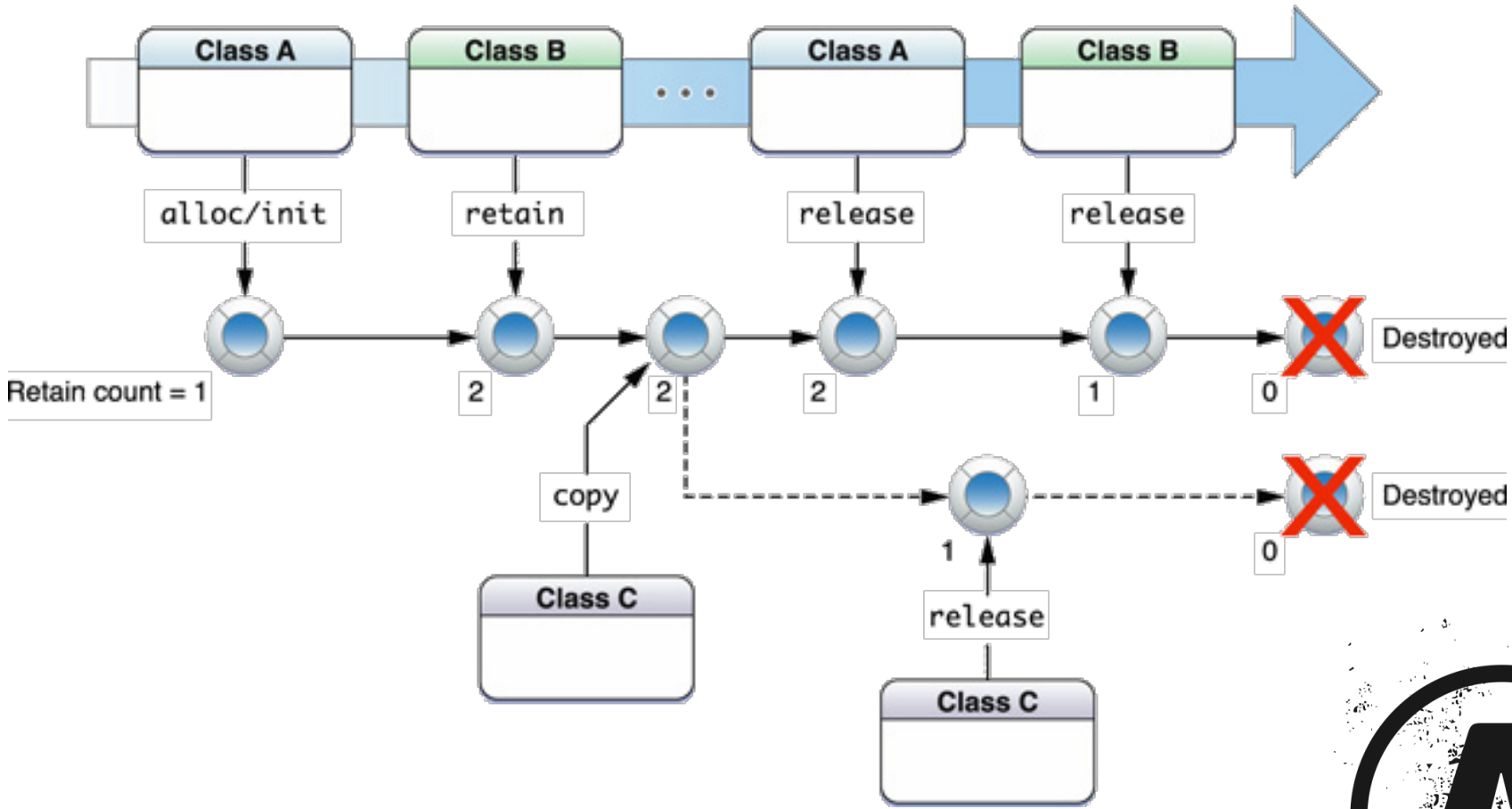
- Reference Counting



Destroying an object with zero references

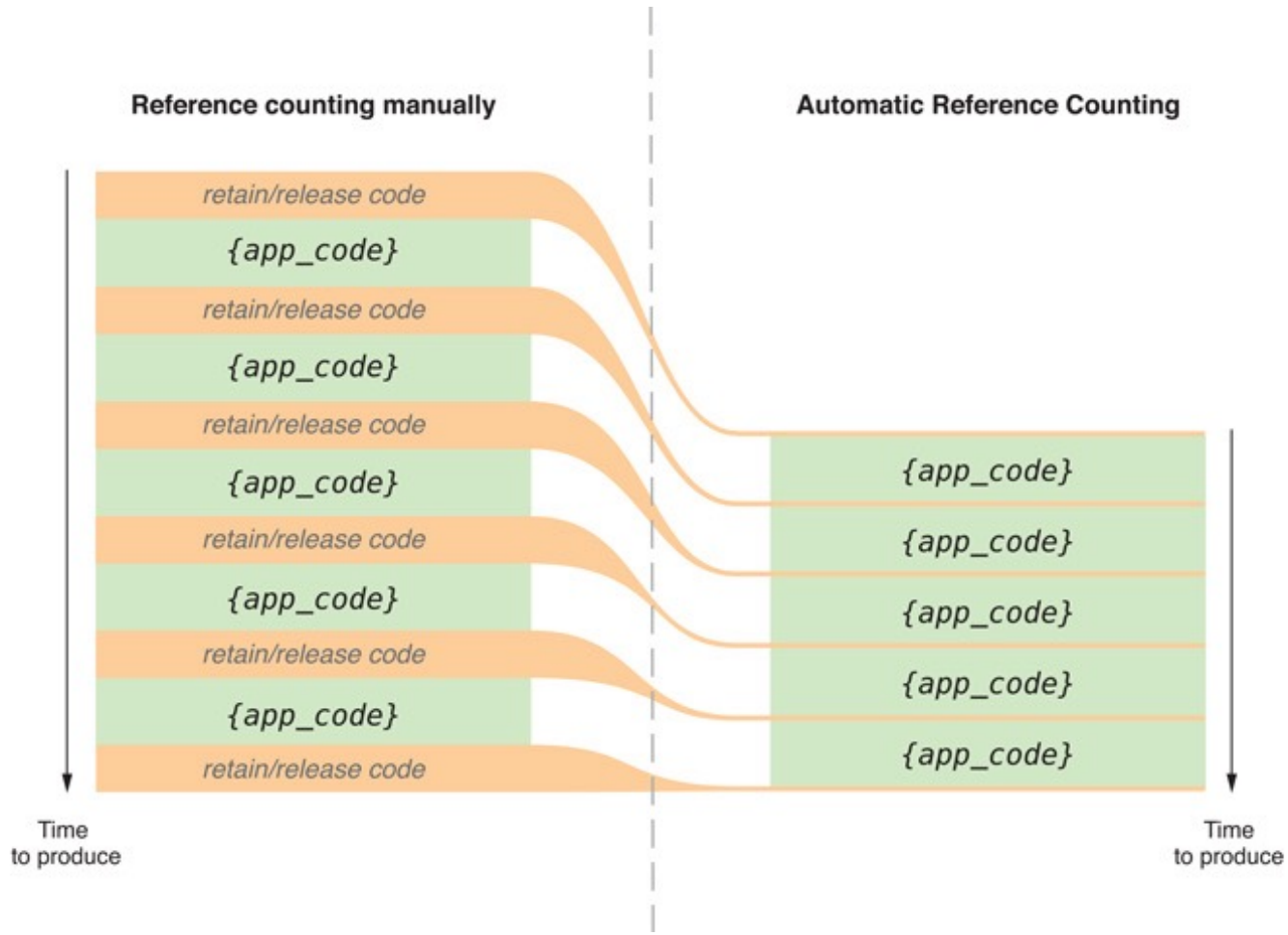
Memory Management

- Manual Retain-Release (MRR)

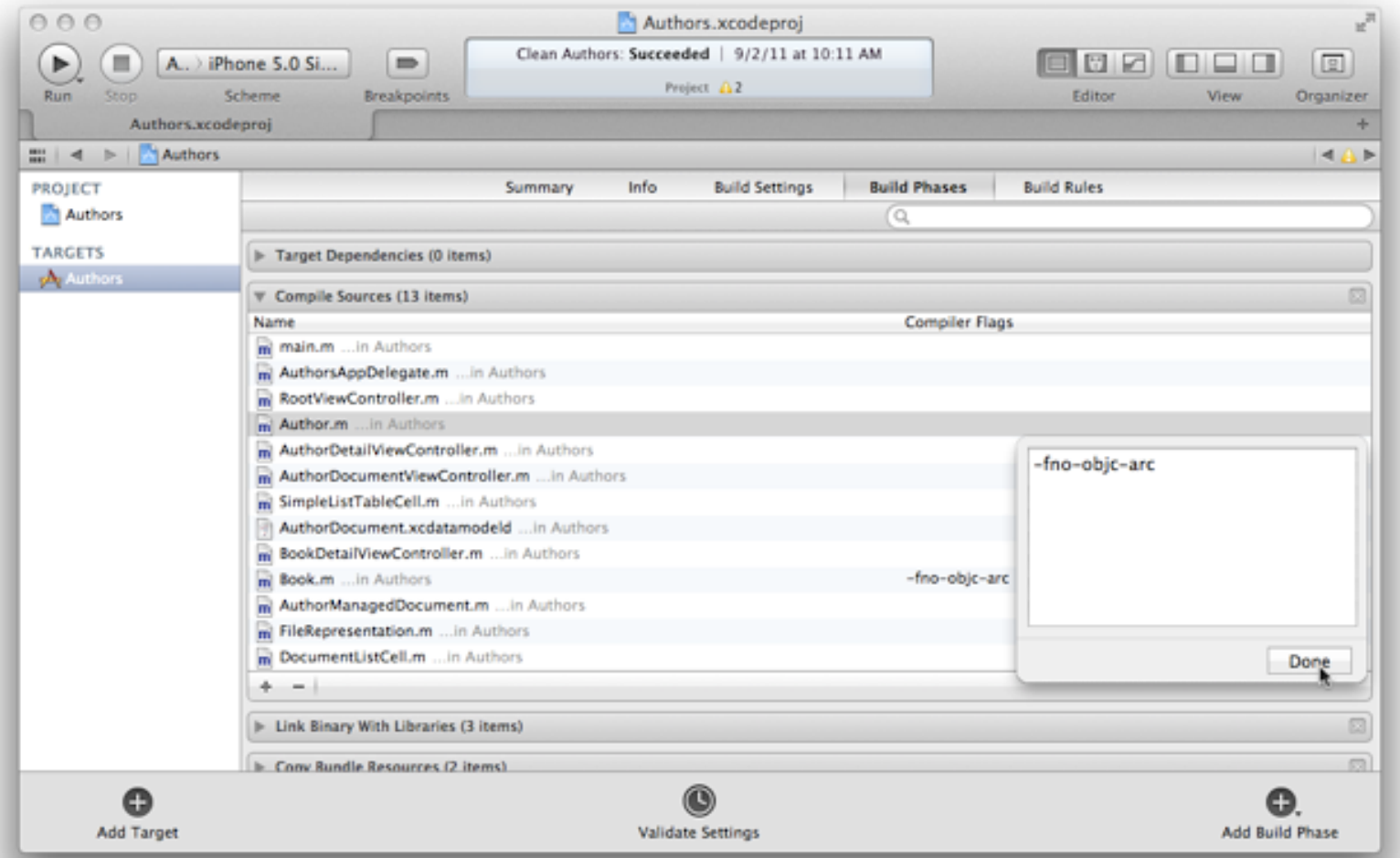


Memory Management

- Automatic Reference Counting (ARC)



Memory Management



Project settings: ARC + MRR

Blocks

- A block is an anonymous inline collection of code that:
  - Has a typed argument list just like a function
  - Has an inferred or declared return type
  - Can capture state from the lexical scope within which it is defined
  - Can optionally modify the state of the lexical scope
  - Can share the potential for modification with other blocks defined within the same lexical scope
  - Can continue to share and modify state defined within the lexical scope (the stack frame) after the lexical scope (the stack frame) has been destroyed
- Also available to pure C and C++

Blocks

Sample Usage

```
float (^oneFrom)(float);

oneFrom = ^(float aFloat) {
    float result = aFloat - 1.0;
    return result;
};



float value = oneFrom(1.34);
```

Modern Syntax

- Literals

```objc
// Example without literals:
NSArray *myArray = [NSArray arrayWithObjects:object1,object2,object3,nil];
NSDictionary *myDictionary1 = [NSDictionary dictionaryWithObject:someObject forKey:@"key"];
NSDictionary *myDictionary2 = [NSDictionary dictionaryWithObjectsAndKeys:object1, key1, object2, key2, nil];
NSNumber *myNumber = [NSNumber numberWithInt:myInt];
```

```objc
// Example with literals:
NSArray *myArray = @[ object1, object2, object3 ];
NSDictionary *myDictionary1 = @{ @"key" : someObject };
NSDictionary *myDictionary2 = @{ key1: object1, key2: object2 };
NSNumber *myNumber = @(myInt);
```

Modern Syntax

- Subscripting
  - Apple LLVM compiler 4.0 or later

```
// Example without subscripting:
id object1 = [someArray objectAtIndex:0];
id object2 = [someDictionary objectForKey:@"key"];
[someMutableArray replaceObjectAtIndex:0 withObject:object3];
[someMutableDictionary setObject:object4 forKey:@"key"];


// Example with subscripting:
id object1 = someArray[0];
id object2 = someDictionary[@"key"];
someMutableArray[0] = object3;
someMutableDictionary[@"key"] = object4;
```

Questions

?

Thanks!!