The World of Swift 3

Philly ETE 2016

Daniel H Steinberg
@dimsumthinking

Objective-C

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

extern NSString *const MagicEightBallDidSave;


@interface MagicEightBall : NSObject

@property (nonatomic) NSString *secretAnswer;

+ (instancetype)sharedModel;
- (NSString *)randomAnswer;
- (NSInteger)numberOfAnswers;
- (NSString *)answerAtIndex:(NSInteger)answerNumber;
- (void)removeAnswerAtIndex:(NSInteger)answerNumber;
- (void)moveAnswerFromIndex:(NSInteger)curentLocation
                    toIndex:(NSInteger)targetLocation;
- (void)insertAnswer:(NSString *)answer
             atIndex:(NSInteger)index;
@end
```

Objective-C

```
#import "MagicEightBall.h"
@import GameKit;

NSString *const UserDefaultsAnswers = @"UserDefaultsAnswers";
NSString *const UserDefaultsSecretAnswer = @"UserDefaultsSecretAnswer";
NSString *const MagicEightBallDidSave = @"MagicEightBallDidSave";


@interface MagicEightBall ()
@property (nonatomic) NSMutableArray *answers;
@property (nonatomic) GKShuffledDistribution *distribution;
@end

@implementation MagicEightBall
```

Objective-C

```
@implementation MagicEightBall

- (instancetype)init {
    self = [super init];
    if (self) {
        [[NSNotificationCenter defaultCenter]
         addObserver:self
         selector:@selector(appDidEnterBackground:)
         name:UIApplicationDidEnterBackgroundNotification
         object:[UIApplication sharedApplication]];
    }
    return self;
}
+ (instancetype)sharedModel {
    static MagicEightBall *_sharedModel = nil;
    static dispatch_once_t createOnlyOneModelToken;
    dispatch_once(&createOnlyOneModelToken, ^{
        _sharedModel = [[self alloc] init];
    });
    return _sharedModel;
}
```

Swift

```
import Foundation
import GameKit

struct Answers {
    private var answers = ["Yes.", "No.", "Maybe.", "Try again."]
    private var distribution: GKShuffledDistribution
    var secretAnswer = "Absolutely"{
        didSet {
            save()
        }
    }

    private let userDefaultsAnswersArray = "AnswersUserDefaultsAnswersArray"
    private let userDefaultsSecretAnswer = "AnswersUserDefaultsSecretAnswer"
    private let userDefaults = NSUserDefaults.standardUserDefaults()

    var numberOfAnswers: Int {
        return answers.count
    }

    private init(answers: [String], secretAnswer: String) {
        self.answers = answers
        self.secretAnswer = secretAnswer
        distribution = GKShuffledDistribution(lowestValue: 0,
                                              highestValue: answers.count - 1)
        save()
    }
```

Obj-C

```
- (NSString *)randomAnswer {
    return self.answers[arc4random_uniform((u_int32_t)[self.answers count])];
}
```

Swift

```
func randomAnswer() -> String {
    return answers[distribution.nextInt()]
}
```

Obj-C

```
- (NSString *)randomAnswer {
    return self.answers[arc4random_uniform((u_int32_t)[self.answers count])];
}
```

Obj-C

```
- (NSString *)randomAnswer {
    return self.answers[arc4random_uniform((u_int32_t)[self.answers count])];
}
```

Obj-C

```
- (NSString *)randomAnswer {
    return self.answers[arc4random_uniform((u_int32_t)[self.answers count])];
}
```

Obj-C

```
- (NSString *)randomAnswer {
    return self.answers[arc4random_uniform((u_int32_t)[self.answers count])];
}
```

Obj-C

```
- (NSString *)randomAnswer {
    return [self.answers objectAtIndex: arc4random_uniform((u_int32_t)[self.answers count])];
}
```

Obj-C

```
- (NSString *)randomAnswer {
    return self.answers[arc4random_uniform((u_int32_t)[self.answers count])];
}
```

Swift

```
func randomAnswer() -> String {
    return answers[distribution.nextInt()]
}
```

Swift

```
func randomAnswer() -> String {
    return answers[distribution.nextInt()]
}
```

Obj-C

```
- (void)removeAnswerAtIndex:(NSInteger)answerNumber {
    [self.answers removeObjectAtIndex:answerNumber];
    [self save];
}
- (void)insertAnswer:(NSString *)answer
             atIndex:(NSInteger)index {
    [self.answers insertObject:answer
                       atIndex:index];
    [self save];
}
- (void)moveAnswerFromIndex:(NSInteger)curentLocation
                    toIndex:(NSInteger)targetLocation {
    NSString *answerToBeMoved = self.answers[curentLocation];
    [self removeAnswerAtIndex:curentLocation];
    [self insertAnswer:answerToBeMoved
               atIndex:targetLocation];
}
```

Obj-C

```
- (void)removeAnswerAtIndex:(NSInteger)answerNumber {
    [self.answers removeObjectAtIndex:answerNumber];
    [self save];
}
- (void)insertAnswer:(NSString *)answer
             atIndex:(NSInteger)index {
    [self.answers insertObject:answer
                       atIndex:index];
    [self save];
}
- (void)moveAnswerFromIndex:(NSInteger)curentLocation
                    toIndex:(NSInteger)targetLocation {
    NSString *answerToBeMoved = self.answers[curentLocation];
    [self removeAnswerAtIndex:curentLocation];
    [self insertAnswer:answerToBeMoved
               atIndex:targetLocation];
}
```

Obj-C

```
- (void)removeAnswerAtIndex:(NSInteger)answerNumber {
    [self.answers removeObjectAtIndex:answerNumber];
    [self save];
}
- (void)insertAnswer:(NSString *)answer
             atIndex:(NSInteger)index {
    [self.answers insertObject:answer
                       atIndex:index];
    [self save];
}
- (void)moveAnswerFromIndex:(NSInteger)curentLocation
                    toIndex:(NSInteger)targetLocation {
    NSString *answerToBeMoved = self.answers[curentLocation];
    [self removeAnswerAtIndex:curentLocation];
    [self insertAnswer:answerToBeMoved
               atIndex:targetLocation];
}
```

Obj-C

```
- (void)removeAnswerAtIndex:(NSInteger)answerNumber {
    [self.answers removeObjectAtIndex:answerNumber];
    [self save];
}
- (void)insertAnswer:(NSString *)answer
             atIndex:(NSInteger)index {
    [self.answers insertObject:answer
                       atIndex:index];
    [self save];
}
- (void)moveAnswerFromIndex:(NSInteger)curentLocation
                    toIndex:(NSInteger)targetLocation {
    NSString *answerToBeMoved = self.answers[curentLocation];
    [self removeAnswerAtIndex:curentLocation];
    [self insertAnswer:answerToBeMoved
               atIndex:targetLocation];
}
```

Obj-C

```
- (void)removeAnswerAtIndex:(NSInteger)answerNumber {
    [self.answers removeObjectAtIndex:answerNumber];
    [self save];
}
- (void)insertAnswer:(NSString *)answer
             atIndex:(NSInteger)index {
    [self.answers insertObject:answer
                       atIndex:index];
    [self save];
}
- (void)moveAnswerFromIndex:(NSInteger)curentLocation
                    toIndex:(NSInteger)targetLocation {
    NSString *answerToBeMoved = self.answers[curentLocation];
    [self removeAnswerAtIndex:curentLocation];
    [self insertAnswer:answerToBeMoved
               atIndex:targetLocation];
}
```

Obj-C

```
- (void)removeAnswerAtIndex:(NSInteger)answerNumber {
    [self.answers removeObjectAtIndex:answerNumber];
    [self save];
}
- (void)insertAnswer:(NSString *)answer
             atIndex:(NSInteger)index {
    [self.answers insertObject:answer
                       atIndex:index];
    [self save];
}
- (void)moveAnswerFromIndex:(NSInteger)curentLocation
                    toIndex:(NSInteger)targetLocation {
    NSString *answerToBeMoved = self.answers[curentLocation];
    [self removeAnswerAtIndex:curentLocation];
    [self insertAnswer:answerToBeMoved
               atIndex:targetLocation];
}
```

Obj-C

```
- (void)removeAnswerAtIndex:(NSInteger)answerNumber {
    [self.answers removeObjectAtIndex:answerNumber];
    [self save];
}
- (void)insertAnswer:(NSString *)answer
             atIndex:(NSInteger)index {
    [self.answers insertObject:answer
                       atIndex:index];
    [self save];
}
- (void)moveAnswerFromIndex:(NSInteger)curentLocation
                    toIndex:(NSInteger)targetLocation {
    NSString *answerToBeMoved = self.answers[curentLocation];
    [self removeAnswerAtIndex:curentLocation];
    [self insertAnswer:answerToBeMoved
               atIndex:targetLocation];
}
```

Obj-C

```
- (void)removeAnswerAtIndex:(NSInteger)answerNumber {
    [self.answers removeObjectAtIndex:answerNumber];
    [self save];
}
- (void)insertAnswer:(NSString *)answer
             atIndex:(NSInteger)index {
    [self.answers insertObject:answer
                       atIndex:index];
    [self save];
}
- (void)moveAnswerFromIndex:(NSInteger)curentLocation
                    toIndex:(NSInteger)targetLocation {
    NSString *answerToBeMoved = self.answers[curentLocation];
    [self removeAnswerAtIndex:curentLocation];
    [self insertAnswer:answerToBeMoved
               atIndex:targetLocation];
}
```

Obj-C

```
- (void)removeAnswerAtIndex:(NSInteger)answerNumber {
    [self.answers removeObjectAtIndex:answerNumber];
    [self save];
}
- (void)insertAnswer:(NSString *)answer
             atIndex:(NSInteger)index {
    [self.answers insertObject:answer
                       atIndex:index];
    [self save];
}
- (void)moveAnswerFromIndex:(NSInteger)curentLocation
                    toIndex:(NSInteger)targetLocation {
    NSString *answerToBeMoved = self.answers[curentLocation];
    [self removeAnswerAtIndex:curentLocation];
    [self insertAnswer:answerToBeMoved
               atIndex:targetLocation];
}
```

Obj-C

```
- (void)removeAnswerAtIndex:(NSInteger)answerNumber {
    [self.answers removeObjectAtIndex:answerNumber];
    [self save];
}
- (void)insertAnswer:(NSString *)answer
             atIndex:(NSInteger)index {
    [self.answers insertObject:answer
                       atIndex:index];
    [self save];
}
- (void)moveAnswerFromIndex:(NSInteger)curentLocation
                    toIndex:(NSInteger)targetLocation {
    NSString *answerToBeMoved = self.answers[curentLocation];
    [self removeAnswerAtIndex:curentLocation];
    [self insertAnswer:answerToBeMoved
               atIndex:targetLocation];
}
```

Obj-C

```
- (void)removeAnswerAtIndex:(NSInteger)answerNumber {
    [self.answers removeObjectAtIndex:answerNumber];
    [self save];
}
- (void)insertAnswer:(NSString *)answer
             atIndex:(NSInteger)index {
    [self.answers insertObject:answer
                       atIndex:index];
    [self save];
}
- (void)moveAnswerFromIndex:(NSInteger)curentLocation
                    toIndex:(NSInteger)targetLocation {
    NSString *answerToBeMoved = self.answers[curentLocation];
    [self removeAnswerAtIndex:curentLocation];
    [self insertAnswer:answerToBeMoved
               atIndex:targetLocation];
}
```

Obj-C

```
- (void)removeAnswerAtIndex:(NSInteger)answerNumber {
    [self.answers removeObjectAtIndex:answerNumber];
    [self save];
}
- (void)insertAnswer:(NSString *)answer
             atIndex:(NSInteger)index {
    [self.answers insertObject:answer
                       atIndex:index];
    [self save];
}
- (void)moveAnswerFromIndex:(NSInteger)curentLocation
                    toIndex:(NSInteger)targetLocation {
    NSString *answerToBeMoved = self.answers[curentLocation];
    [self removeAnswerAtIndex:curentLocation];
    [self insertAnswer:answerToBeMoved
               atIndex:targetLocation];
}
```

Obj-C

```
- (void)removeAnswerAtIndex:(NSInteger)answerNumber {
    [self.answers removeObjectAtIndex:answerNumber];
    [self save];
}
- (void)insertAnswer:(NSString *)answer
             atIndex:(NSInteger)index {
    [self.answers insertObject:answer
                       atIndex:index];
    [self save];
}
- (void)moveAnswerFromIndex:(NSInteger)curentLocation
                    toIndex:(NSInteger)targetLocation {
    NSString *answerToBeMoved = self.answers[curentLocation];
    [self removeAnswerAtIndex:curentLocation];
    [self insertAnswer:answerToBeMoved
               atIndex:targetLocation];
}
```

Obj-C

```
- (void)removeAnswerAtIndex:(NSInteger)answerNumber {
    [self.answers removeObjectAtIndex:answerNumber];
    [self save];
}
- (void)insertAnswer:(NSString *)answer
             atIndex:(NSInteger)index {
    [self.answers insertObject:answer
                       atIndex:index];
    [self save];
}
- (void)moveAnswerFromIndex:(NSInteger)curentLocation
                    toIndex:(NSInteger)targetLocation {
    NSString *answerToBeMoved = self.answers[curentLocation];
    [self removeAnswerAtIndex:curentLocation];
    [self insertAnswer:answerToBeMoved
               atIndex:targetLocation];
}
```

Swift

```
func removeAnswer(at index: Int) -> Answers {
    var localAnswers = answers
    localAnswers.removeAtIndex(index)
    return Answers(answers: localAnswers, secretAnswer: secretAnswer)
}

func insert(answer: String, at index: Int) -> Answers {
    var localAnswers = answers
    localAnswers.insert(answer, atIndex: index)
    return Answers(answers: localAnswers, secretAnswer: secretAnswer)
}
func moveAnswer(from fromIndex: Int, to toIndex: Int) -> Answers {
    return removeAnswer(at: fromIndex).insert(answers[fromIndex], at: toIndex)
}
```

Swift

```
func removeAnswer(at index: Int) -> Answers {
    var localAnswers = answers
    localAnswers.removeAtIndex(index)
    return Answers(answers: localAnswers, secretAnswer: secretAnswer)
}

func insert(answer: String, at index: Int) -> Answers {
    var localAnswers = answers
    localAnswers.insert(answer, atIndex: index)
    return Answers(answers: localAnswers, secretAnswer: secretAnswer)
}
func moveAnswer(from fromIndex: Int, to toIndex: Int) -> Answers {
    return removeAnswer(at: fromIndex).insert(answers[fromIndex], at: toIndex)
}
```

Swift

```
func removeAnswer(at index: Int) -> Answers {
    var localAnswers = answers
    localAnswers.removeAtIndex(index)
    return Answers(answers: localAnswers, secretAnswer: secretAnswer)
}

func insert(answer: String, at index: Int) -> Answers {
    var localAnswers = answers
    localAnswers.insert(answer, atIndex: index)
    return Answers(answers: localAnswers, secretAnswer: secretAnswer)
}
func moveAnswer(from fromIndex: Int, to toIndex: Int) -> Answers {
    return removeAnswer(at: fromIndex).insert(answers[fromIndex], at: toIndex)
}
```

Swift

```
func removeAnswer(at index: Int) -> Answers {
    var localAnswers = answers
    localAnswers.removeAtIndex(index)
    return Answers(answers: localAnswers, secretAnswer: secretAnswer)
}

func insert(answer: String, at index: Int) -> Answers {
    var localAnswers = answers
    localAnswers.insert(answer, atIndex: index)
    return Answers(answers: localAnswers, secretAnswer: secretAnswer)
}
func moveAnswer(from fromIndex: Int, to toIndex: Int) -> Answers {
    return removeAnswer(at: fromIndex).insert(answers[fromIndex], at: toIndex)
}
```

Swift

```
func removeAnswer(at index: Int) -> Answers {
    var localAnswers = answers
    localAnswers.removeAtIndex(index)
    return Answers(answers: localAnswers, secretAnswer: secretAnswer)
}

func insert(answer: String, at index: Int) -> Answers {
    var localAnswers = answers
    localAnswers.insert(answer, atIndex: index)
    return Answers(answers: localAnswers, secretAnswer: secretAnswer)
}
func moveAnswer(from fromIndex: Int, to toIndex: Int) -> Answers {
    return removeAnswer(at: fromIndex).insert(answers[fromIndex], at: toIndex)
}
```

Swift

```
func removeAnswer(at index: Int) -> Answers {
    var localAnswers = answers
    localAnswers.removeAtIndex(index)
    return Answers(answers: localAnswers, secretAnswer: secretAnswer)
}

func insert(answer: String, at index: Int) -> Answers {
    var localAnswers = answers
    localAnswers.insert(answer, atIndex: index)
    return Answers(answers: localAnswers, secretAnswer: secretAnswer)
}
func moveAnswer(from fromIndex: Int, to toIndex: Int) -> Answers {
    return removeAnswer(at: fromIndex).insert(answers[fromIndex], at: toIndex)
}
```

Swift

```
func removeAnswer(at index: Int) -> Answers {
    var localAnswers = answers
    localAnswers.removeAtIndex(index)
    return Answers(answers: localAnswers, secretAnswer: secretAnswer)
}

func insert(answer: String, at index: Int) -> Answers {
    var localAnswers = answers
    localAnswers.insert(answer, atIndex: index)
    return Answers(answers: localAnswers, secretAnswer: secretAnswer)
}
func moveAnswer(from fromIndex: Int, to toIndex: Int) -> Answers {
    return removeAnswer(at: fromIndex).insert(answers[fromIndex], at: toIndex)
}
```

Swift

```
func removeAnswer(at index: Int) -> Answers {
    var localAnswers = answers
    localAnswers.removeAtIndex(index)
    return Answers(answers: localAnswers, secretAnswer: secretAnswer)
}

func insert(answer: String, at index: Int) -> Answers {
    var localAnswers = answers
    localAnswers.insert(answer, atIndex: index)
    return Answers(answers: localAnswers, secretAnswer: secretAnswer)
}
func moveAnswer(from fromIndex: Int, to toIndex: Int) -> Answers {
    return removeAnswer(at: fromIndex).insert(answers[fromIndex], at: toIndex)
}
```

Swift

```
func removeAnswer(at index: Int) -> Answers {
    var localAnswers = answers
    localAnswers.removeAtIndex(index)
    return Answers(answers: localAnswers, secretAnswer: secretAnswer)
}

func insert(answer: String, at index: Int) -> Answers {
    var localAnswers = answers
    localAnswers.insert(answer, atIndex: index)
    return Answers(answers: localAnswers, secretAnswer: secretAnswer)
}
func moveAnswer(from fromIndex: Int, to toIndex: Int) -> Answers {
    return removeAnswer(at: fromIndex).insert(answers[fromIndex], at: toIndex)
}
```

Swift

```
func removeAnswer(at index: Int) -> Answers {
    var localAnswers = answers
    localAnswers.removeAtIndex(index)
    return Answers(answers: localAnswers, secretAnswer: secretAnswer)
}

func insert(answer: String, at index: Int) -> Answers {
    var localAnswers = answers
    localAnswers.insert(answer, atIndex: index)
    return Answers(answers: localAnswers, secretAnswer: secretAnswer)
}
func moveAnswer(from fromIndex: Int, to toIndex: Int) -> Answers {
    return removeAnswer(at: fromIndex).insert(answers[fromIndex], at: toIndex)
}
```

Swift

```
func removeAnswer(at index: Int) -> Answers {
    var localAnswers = answers
    localAnswers.removeAtIndex(index)
    return Answers(answers: localAnswers, secretAnswer: secretAnswer)
}

func insert(answer: String, at index: Int) -> Answers {
    var localAnswers = answers
    localAnswers.insert(answer, atIndex: index)
    return Answers(answers: localAnswers, secretAnswer: secretAnswer)
}
func moveAnswer(from fromIndex: Int, to toIndex: Int) -> Answers {
    return removeAnswer(at: fromIndex).insert(answers[fromIndex], at: toIndex)
}
```

Swift

```
func removeAnswer(at index: Int) -> Answers {
    var localAnswers = answers
    localAnswers.removeAtIndex(index)
    return Answers(answers: localAnswers, secretAnswer: secretAnswer)
}

func insert(answer: String, at index: Int) -> Answers {
    var localAnswers = answers
    localAnswers.insert(answer, atIndex: index)
    return Answers(answers: localAnswers, secretAnswer: secretAnswer)
}
func moveAnswer(from fromIndex: Int, to toIndex: Int) -> Answers {
    return removeAnswer(at: fromIndex).insert(answers[fromIndex], at: toIndex)
}
```

Swift

```
func removeAnswer(at index: Int) -> Answers {
    var localAnswers = answers
    localAnswers.removeAtIndex(index)
    return Answers(answers: localAnswers, secretAnswer: secretAnswer)
}

func insert(answer: String, at index: Int) -> Answers {
    var localAnswers = answers
    localAnswers.insert(answer, atIndex: index)
    return Answers(answers: localAnswers, secretAnswer: secretAnswer)
}
func moveAnswer(from fromIndex: Int, to toIndex: Int) -> Answers {
    return removeAnswer(at: fromIndex).insert(answers[fromIndex], at: toIndex)
}
```

Swift

```
func removeAnswer(at index: Int) -> Answers {
    var localAnswers = answers
    localAnswers.removeAtIndex(index)
    return Answers(answers: localAnswers, secretAnswer: secretAnswer)
}

func insert(answer: String, at index: Int) -> Answers {
    var localAnswers = answers
    localAnswers.insert(answer, atIndex: index)
    return Answers(answers: localAnswers, secretAnswer: secretAnswer)
}
func moveAnswer(from fromIndex: Int, to toIndex: Int) -> Answers {
    return removeAnswer(at: fromIndex).insert(answers[fromIndex], at: toIndex)
}
```

Swift

```
func removeAnswer(at index: Int) -> Answers {
    var localAnswers = answers
    localAnswers.removeAtIndex(index)
    return Answers(answers: localAnswers, secretAnswer: secretAnswer)
}

func insert(answer: String, at index: Int) -> Answers {
    var localAnswers = answers
    localAnswers.insert(answer, atIndex: index)
    return Answers(answers: localAnswers, secretAnswer: secretAnswer)
}
func moveAnswer(from fromIndex: Int, to toIndex: Int) -> Answers {
    return removeAnswer(at: fromIndex).insert(answers[fromIndex], at: toIndex)
}
```

Swift

Future?

Swift

Open

Swift

Open?

Swift

Open!

Access

```
public
internal
private
```

Access

```
public
internal
private
```

Access

```
public
internal
private
```

Access

```
public
internal
private
```

Access

```
public
internal
fileprivate
```

Access

```
public
internal
private
fileprivate
```

Oddities

```
sort()
sorted()
```

Oddities

```
sortInPlace()
sort()
```

Name functions and methods according to their side-effects

```
sort()
sorted()
```

Those without side-effects should read as noun phrases

```
x.distance(to: y)
i.successor()
```

Those with side-effects should read as imperative verb phrases

```
x.sort()
x.append(y)
```

Use the “ed/ing” rule to name the nonmutating counterpart of a mutating method

```
sort()
sorted()/sorting()
```

Oddities

```
sort()
sorted()
```

```
calculateRevenue(for sales: Int, in currency: Currency)
```

0001 - Allow (most) keywords as argument labels

```
calculateRevenue(for sales: Int, in currency: Currency)
```

```
calculateRevenue(for sales: Int, in currency: Currency)
```

```
calculateRevenue(for: numberOfCopies, in: .dollars)
```

```
calculateRevenue(for: numberOfCopies, in: .dollars)
```

Enum cases are now lower case

```
calculateRevenue(for: numberOfCopies, in: .dollars)
```

Enum cases are now lower case

```
enum Currency {
    case Dollars
    case Euros
    case Pounds
    case Yen
}
```

Enum cases are now lower case

```
enum Currency {
    case dollars
    case euros
    case pounds
    case yen
}
```

Legal without the dot

```
enum Currency {
    case dollars
    case euros
    case pounds
    case yen

    var symbol: String {
        switch self {
        case dollars:
            return "$"
        default:
            return "I don't know"
        }
    }
}
```

Use the dot

```
enum Currency {
    case dollars
    case euros
    case pounds
    case yen

    var symbol: String {
        switch self {
        case .dollars:
            return "$"
        default:
            return "I don't know"
        }
    }
}
```

Just an idea

```
enum Currency {
    case dollars
    case euros
    case pounds
    case yen

    var symbol: String {
        switch self {
        case .dollars:
            return "$"
        default:
            return "I don't know"
        }
    }
}
```

Just an idea

```
enum Currency {
    case dollars
    case euros
    case pounds
    case yen

    var symbol: String {
        switch self {
        case .dollars:
            return "$"
        default:
            return "I don't know"
        }
    }
}
```

0043 Declare variables in 'case' labels with multiple patterns

```
enum MyEnum {
    case Case1(Int,Float)
    case Case2(Float,Int)
}
switch value {
case let .Case1(x, 2), let .Case2(2, x):
    print(x)
case .Case1, .Case2:
    break
}
```

```
enum MyEnum {
    case Case1(Int,Float)
    case Case2(Float,Int)
}
switch value {
case let .Case1(x, 2), let .Case2(2, x):
    print(x)
case .Case1, .Case2:
    break
}
```

0033 Import Obj-C Constants as Swift Types

```
HK_EXTERN NSString * const HKQuantityTypeIdentifierBodyMassIndex;
HK_EXTERN NSString * const HKQuantityTypeIdentifierBodyFatPercentage;
HK_EXTERN NSString * const HKQuantityTypeIdentifierHeight;
HK_EXTERN NSString * const HKQuantityTypeIdentifierBodyMass;
HK_EXTERN NSString * const HKQuantityTypeIdentifierLeanBodyMass;
```

```
HK_EXTERN NSString * const HKQuantityTypeIdentifierBodyMassIndex;
HK_EXTERN NSString * const HKQuantityTypeIdentifierBodyFatPercentage;
HK_EXTERN NSString * const HKQuantityTypeIdentifierHeight;
HK_EXTERN NSString * const HKQuantityTypeIdentifierBodyMass;
HK_EXTERN NSString * const HKQuantityTypeIdentifierLeanBodyMass;
```

```
enum HKQuantityTypeIdentifier : String {
    case BodyMassIndex
    case BodyFatPercentage
    case Height
    case BodyMass
    case LeanBodyMass
}
```

0002 Removing currying `func` declaration syntax

```
func curried(x: Int)(y: Int) -> Int {
  return {(y: Int) -> Int in
      return x * y
  }
}

curried(7)(8)
```

0002 Removing currying `func` declaration syntax

```
func curried(x: Int)(y: Int) -> Int {
  return {(y: Int) -> Int in
      return x * y
  }
}

curried(7)(8)
```

0002 Removing currying `func` declaration syntax

```
~~func curried(x: Int)(y: Int) -> Int {~~
  return {(y: Int) -> Int in
      return x * y
  }
}

curried(7)(8)
```

0002 Removing currying `func` declaration syntax

```
func curried(x: Int) -> (Int) -> Int {
  return {(y: Int) -> Int in
      return x * y
  }
}

curried(7)(8)
```

```
func curried(x: Int) -> (Int) -> Int {
  return {(y: Int) -> Int in
      return x * y
  }
}

curried(7)(8)
```

0003 Removing `var` from Function Parameters

```
func changes(var input: Int) {
    input = input * 2  // can
}
```

```
func changes(var input: Int) {
    input = input * 2  // can
}
```

0053 Removing `let` from Function Parameters

```
func changes(~~let~~ input: Int) {
    input = input * 2  // can't
}
```

```
func changes(input: Int) {
    var mutableInput = input
    mutableInput = mutableInput * 2
}
```

Shadow Name

```
func changes(input: Int) {
    var input = input
    input = input * 2
}
```

0031 Adjusting `inout` Declarations for Type Decoration

```
func changes(inout input: Int) {
   input = input * 2
}
```

```
func changes(input: inout Int) {
   input = input * 2
}
```

0004 Remove the ++ and −− operators

```
++x
x++
```

```
++x
x++

x += 1
```

0007 Remove C-style for-loops with conditions and incrementers

```
for (int i = 0; i < array.count; i++)
```

```
for i in 0..<array.count
```

0009 Require self for accessing instance members

```
struct Friend {
    let name: String
    let location: String

    func nameBadge() {
        print("I'm", name, "from", location)
    }
}
```

```
struct Friend {
    let name: String
    let location: String

    func nameBadge() {
        print("I'm", self.name, "from", self.location)
    }
}
```

0009 Require self for accessing instance members

```
struct Friend {
    let name: String
    let location: String

    func nameBadge() {
        print("I'm", name, "from", location)
    }
}
```

0011 Replace `typealias` keyword with `associatedtype` for associated type declarations

```
protocol Prot {
    typealias Container : SequenceType
}
extension Prot {
    typealias Element = Container.Generator.Element
}
```

```
protocol Prot {
    associatedtype Container : SequenceType
}
extension Prot {
    typealias Element = Container.Generator.Element
}
```

```
protocol Prot {
    associatedType Container : SequenceType
}
extension Prot {
    typeAlias Element = Container.Generator.Element
}
```

```
protocol Prot {
    associatedtype Container : SequenceType
}
extension Prot {
    typealias Element = Container.Generator.Element
}
```

Nothing is impossible, but I consider the odds of typealias and associatedtype becoming lower camel cased to be extremely near zero.

0022 Referencing the Objective-C selector of a method

```
let timer = NSTimer.scheduledTimerWithTimeInterval(0.1,
                                                   target: self,
                                                   selector: "doSomething",
                                                   userInfo: nil,
                                                   repeats: true)
```

```
let timer = NSTimer.scheduledTimerWithTimeInterval(0.1,
                                                   target: self,
                                                   selector: #selector(UIView.insertSubview(_:atIndex:)),
                                                   userInfo: nil,
                                                   repeats: true)
```

0040 Replacing Equal Signs with Colons For Attribute Arguments

```
infix operator +++ {associativity  left precedence  120}
```

```
infix operator +++ {associativity = left precedence = 120}
```

0046 Establish consistent label behavior across all parameters including first labels

```
func increase(ourNumber: Int, delta: Int) -> Int {

}

increase(6, delta: 3)
```

0046 Establish consistent label behavior across all parameters including first labels

```
func increase(ourNumber: Int, delta: Int) -> Int {

}

increase(6, delta: 3)
```

0046 Establish consistent label behavior across all parameters including first labels

```
func increase(ourNumber: Int, delta: Int) -> Int {

}

increase(6, delta: 3)
```

0046 Establish consistent label behavior across all parameters including first labels

```
func increaseBy(delta: Int) -> Int {

}

increaseBy(6)
```

```
func increaseBy(delta: Int) -> Int {

}

increaseBy(delta: 6)
```

```
func increaseBy(_ delta: Int) -> Int {

}

increaseBy(6)
```

```
func increase(by delta: Int) -> Int {

}

increase(by: 6)
```

0023 API Design Guidelines

```
func increase(by delta: Int) -> Int {

}

increase(by: 6)
```

0023 API Design Guidelines

Prefer methods to free functions

When there's no obvious self

```
min(x, y, z)
```

When the function is an unconstrained generic

```
print(x)
```

When the function is part of the established domain notation

```
sin(x)
```

Follow case conventions

Follow case conventions

Even when they change

Methods can share a base name

Methods can share a base name

Good when they do analogous things

Methods can share a base name

Not good (tableView!) when they don't

Choose good parameter names

```
func move(from startingIndex: Int, to endingIndex: Int)
```

Take advantage of default values

```
func hello(name: String = "World")
```

```
init(name: String, hometown: String? = nil)
```

Prefer to locate parameters with defaults at the end

Argument Labels

```
min(number1, number2)
```

Argument Labels

```
func move(from startingIndex: Int, to endingIndex: Int)
```

```
move(from: here to: there)
```

Argument Labels

Exceptions

Omit labels when arguments can't be distinguished

```
min(number1, number2)
```

Omit labels in full width inits

```
Double(someInt)
```

Omit labels in full width inits

```
Double(_ anInt: someInt)
```

When the preposition applies to the whole

```
func move(from startingIndex: Int, to endingIndex: Int)
```

When the preposition applies to the whole

```
func moveTo(x: Int, y: Int)
```

When the preposition applies to the whole

```
x.removeBoxes(havingLength: 12)
```

The World of Swift 3

Philly ETE 2016

Daniel H Steinberg
@dimsumthinking