Mobile Application Programming
Swift
Swift

- An object-oriented and functional language designed with code safety as a core goal in the language syntax
  - No pointers, single-line branches, bounds checking
- Built to co-exist with Objective-C, Apple’s previously preferred language, as well as use existing frameworks
- Uses a memory management technology called Automatic Reference Counting to determine object life
- Strongly typed, but uses type inference to reduce code
- Generics with built-in support for Array and Dictionary
Swift Syntax

C++ Syntax

```cpp
Person* person = new Person();
int age = person->age();
person->setHeight(1.8);
delete person;
```

Swift Syntax

```swift
var person: Person = Person()
var age: Int = person.age
person.height = 1.8
// Handled automatically by ARC
```

```cpp
Car* car = new Car(Car.viper);
float velocity = car->velocity();
car->setVelocity(velocity + 200.0);
delete car;
```

```swift
var car: Car = Car(type: Car.viper)
var velocity: Float = car.velocity
car.velocity = velocity + 200.0
// Handled automatically by ARC
```
Swift Syntax

C++ Syntax

car.setVelocityAndAcceleration(200.0f, 10.0f);

Swift Syntax

car.setVelocity(200.0, andAcceleration: 10.0)
Swift Syntax

C++ Syntax

typedef struct
{
  float x;
  float y;
} Point;

Point PointMake(float x, float y)
{
  Point p;
  p.x = x;
  p.y = y;
  return p;
}

//...
car.setPosition(
  PointMake(10.0, 50.0));

Swift Syntax

struct Point
{
  var x: Float = 0.0
  var y: Float = 0.0
}

func PointMake(x: Float, y: Float) -> Point
{
  var p: Point = Point()
  p.x = x
  p.y = y
  return p
}

//...
car.position = PointMake(10.0, 50.0)
class Car
{
    Point position;
    float velocity;
    int model;
    char* vin;

    public:
    static const int viper = 1;
    Car(int model);
    ~Car();
    Point position();
    void setPosition(Point p);
};
Swift Features

- Explicit Nullable Types called Optionals with `?` shorthand in declaration. Unwrap with `!` or `if let x = opX`

- Mutability supported on all types via declaration keywords `var` (Mutable) and `let` (Immutable)

- Support for tuples in declarations and function returns

- Flexible `switch` statements use `fallthrough` not `break`

- First-class functions that are implemented as closures

- Classes, structures, enums with advanced features
Swift Top-Level Entities

- Like C/C++ but unlike Java, Swift allows declarations of **functions**, **variables**, and **constants** at the **top-level**, outside any class declaration.
- Constants are declared using the **let** keyword.
- Variables are declared using the **var** keyword.
- Functions are declared using the **func** keyword with parameter names **interleaved** with the name of the function, causing it to **read like a sentence**.
Swift Objects

- Classes, structures, and enums are all object types with different defaults in usage.
  - Classes are reference types that share the same object when assignments are made.
  - Structs are always copied on assignment.
- Single inheritance, but may conform to many protocols.
- Add functions and protocols to existing objects using `extension` keyword. Also used to break up large objects.
Swift Classes

- Member functions and properties declared using same syntax as top-level declarations.
- Function declarations use parameter labels, but the first label is omitted when declared in a class.
- Properties declare both getter / setter and a (hidden) backing variable using `var` and `let` keywords.
- Use `private`, `internal`, and `public` for access control.
- Constructors are declared using `init()`, but have different inheritance rules than most languages.
Swift Optional Unwrap

- Working with optional values can be tiresome because they are constantly being checked against nil
- Swift offers many facilities to improve the experience
  - Use of `?` and `!` to unwrap the optional
  - Chaining expressions using `?` like `a?.property?.go()`
  - Conditional unwrapping using `if let a = a { }`
Cocoa Foundation Framework

- Standard Library for Swift, like STL or java.*
- Originally coded by NextStep, updated for Swift
- Works identically on Mac OS X and iOS
- Objective-C objects that have Swift compatibility
Foundation

Basic Classes

- NSObject
- NSString
- NSNumber
- NSData
- NSArray
- NSDictionary

A Few Other Good Ones

- NSDate
- NSTimer
- NSRunLoop
- NSThread
- NSFileManager
- NSSocketPort
NSArray, Array, or []

Auto-Expanding

Sorting
NSDictionary, Dictionary, [:]

Key-Value Encoding

Read / Write Files