Swift and C# Quick Reference - Language Equivalents and Code Examples

Variables

	Swift	C#
boolean	Bool	bool
constant	let	const
declaration	var	var
float	Float, Double	float, double
integer	Int	int
optional	? (optional)	? (nullable)
tuple	tuple	System.Tuple
string	String (value)	string (reference)

Optional and nullable reference variables

Swift: Only optional reference variables can be set to nil.

```
var optBox : Box? = nil
if let aBox = optBox {
    println(aBox.top)
}
if optBox!.top > 4 {
    println("Box is not at the origin.")
}
```

C#: All reference variables can be set to null.

```
string optString = null;
if (optString != null) {
    Console.WriteLine(optString);
}

int? length = null;
if (length.HasValue) {
    Console.WriteLine(length.Value);
}
```

Tuples

Swift: You create a tuple using Swift's tuple syntax. You access the tuple's values using the value names or indexing.

```
func summary(b : Box) -> (Int, Double) {
    return (b.area(), b.diagonal())
}
var box = Box(top: 0, left: 0, bottom: 1, right: 1)
var (area, diagonal) = summary(box)
var stats = (area, diagonal)
var description =
    "Area is \(area) and diagonal is \(diagonal)."
var description2 =
    "Area is \(stats.0) and diagonal is \(stats.1)."
```

C#: You create a tuple by instantiating a Tuple object. You access the type values using Item1, Item2, etc.

```
Tuple<int, double> Summary(Box box) {
    return new Tuple<int,double>(box.Area(),
        box.Diagonal());
}
var box = new Box(0, 0, 1, 1);
var summaryTuple = Summary(box);
var description = "Area is " + summaryTuple.Item1
    + " and diagonal is " + summaryTuple.Item2 + ".";
```

Strings and characters

Swift: String is a value type with properties and methods that also provides all the functionality of the NSString type. Strings can be concatenated with string interpolation or the + operator.

```
var world = "world"
var helloWorld = hello + ", " + world
var sayHello = "\(hello), \(world)"
var capitalized = helloWorld.uppercaseString
var numberOfChars = countElements(sayHello)
var seventhChar = sayHello[advance(sayHello.startIndex, 7)]
var startsWithHello = sayHello.hasPrefix("hello")
```

C#: String is an alias for System.String, a class with properties, methods, and indexing. Strings can be concatenated with String.Format or the + operator.

```
var hello = "hello";
var world = "world";
var helloWorld = hello + ", " + world;
var sayHello = string.Format("%s, %s", hello, world);
var capitalized = helloWorld.ToUpper();
var numberOfChars = sayHello.Length;
var charN = sayHello[7];
var startsWithHello = sayHello.StartsWith("hello");
```

Swift and C# are C-style languages that are both productive and powerful. Using Swift, you can create iOS applications using Xcode. By leveraging your Swift skills, it's an easy transition to C#. Then using C# with Xamarin and Visual Studio, you can create applications that run on Windows, iOS, and Android. Learn more at Cross-Platform Development in Visual Studio (http://aka.ms/T71425) and Understanding the Xamarin Mobile Platform (http://aka.ms/Teumsa).

Operators

	Swift	C#
arithmetic	+, -, *, /, %	+, -, *, /, %
assignment	=	=
bitwise	<<, >>, &, |, ~, ^,	<<, >>, <<=, >>= &, |, ^, ~
overflow	&+, &-, &*, &/, &%	checked unchecked
overloading	overloading	overloading
range	a..<b, a...b	(no equivalent)
relational	==, !=, >, <	==, !=, >, <

Operator overloading

Swift: In this example, adding two boxes returns a box that contains both boxes.

```
func + (r1: Box, r2: Box) -> Box {
    return (Box(
        top: min(r1.top, r2.top),
        left: min(r1.left, r2.left),
        bottom: max(r1.bottom, r2.bottom),
        right: max(r1.right, r2.right)))
}
var boxSum = Box(top: 0, left: 0, bottom: 1, right: 1)
      + Box(top: 1, left: 1, bottom: 3, right: 3)
```

C#: Adding two boxes returns a box that contains both boxes.

```
public static Box operator +(Box box1, Box box2)
{
    return new Box(
        (int)Math.Min(box1.Top, box2.Top),
        (int)Math.Min(box1.Left, box2.Left),
        (int)Math.Max(box1.Bottom, box2.Bottom),
        (int)Math.Max(box1.Right, box2.Right));
}
var boxSum = new Box(0, 0, 1, 1) + new Box(1, 1, 3, 3);
```

Equality and assignment

Swift: The assignment operator does not return a value, therefore you can't use it as a conditional expression and you can't do chain assignments.

```
var a = 6
var b = a
if (b == 6) {
    a = 2
}
```

C#: Chain assignment is allowed and testing assignment expressions is allowed.

```
int b = 2;
int a = b = 6;
if ((b = 6) == 6)
    a = 2;
```

Range Operator

Swift: Use the range operator to create a range of values.

```
for i in 1...5 {
    println(i)
}
```

C#: Use the Enumerable.Range method to generate a List of integers.

```
foreach (int i in Enumerable.Range(1, 5).ToList())
{
    Console.WriteLine(i);
}
```

Overflow

Swift: By default, underflow and overflow produce an error at runtime. You can use the overflow operators to suppress errors, but the resulting calculation might not be what you expect.

```
// This code does not produce an error, but the
// resulting value is not the expected value.
var largeInt : Int = Int.max
var tooLarge : Int = largeInt &+ 1
```

C#: By default, underflow and overflow do not produce an error. You can use the checked keyword so that an exception is thrown at runtime. If you are using implicit variable declarations, the runtime will create variables that can contain the underflow or overflow value.

```
// This code throws an exception at runtime.
int largeInt = int.MaxValue;
checked {
    int tooLarge = largeInt + 5;
}
```

Programs

	Swift	C#
attribute	(no equivalent)	attributes
memory management	automatic reference counting	tree-based garbage collection
module	module	library
namespace	(no equivalent)	namespace
preprocessor directives	(no equivalent)	preprocessor directives

Control flow

	Swift	C#
break, continue	break, continue	break, continue
do-while	do-while	do-while
for	for	for
for-in	for-in	foreach-in
if	if	if
locking	(no equivalent)	lock
queries	(no equivalent)	LINQ
switch	switch, fallthrough	switch
try-catch, throw	assert	try-catch, throw
using	(no equivalent)	using
unsafe	(no equivalent)	unsafe
while	while	while
yield	(no equivalent)	yield

For statement

Swift: Swift supports C-style for loops, loops that iterate over collections, and loops that return (index, value) pairs.

```
var squares = [Box]()
for var size : Int = 1; size < 6; size++ {
    squares.append(
        Box(top: 0, left: 0, bottom: size, right: size))
}
for box in squares {
    println(area(box))
}
for (index, value) in enumerate(squares) {
    println("Area of box \(index) is \(area(value)).")
}
```

C#: You can use C-style for loops and loops that iterate over collections.

```
vvar squares = new List<Box>();
for (int size = 1; size < 6; size++) {
    squares.Add(new Box(0, 0, size, size));
}
foreach (var square in squares) {
    Console.WriteLine(area(square));
}
```

If statement

Swift: The test condition must return a Boolean value and the execution statements must be enclosed in braces.

```
var strings = ["one", "two", "three", "four"]
if (strings[0] == "one") {
    println("First word is 'one'.");
}
```

C#: C# allows non-Boolean test conditions and braces are not required around the execution statements.

```
string[] strings = { "one", "two", "three" };
if (strings[0] == "one") {
    Console.WriteLine("First word is 'one'.");
}
```

Switch statement

Swift: Cases do not fall through unless you use the fallthrough keyword. Therefore, a break statement is not required. A default case is usually required. Swift supports ranges in cases.

```
var aSquare = Box(top: 0, left: 0, bottom: 4, right: 4)
var label = ""
switch SpecialBox.GetSpecialType(aSquare) {
    case .Square : label = "Square"
    case .Rectangle : label = "Rectangle"
    case .GoldenRatio : label = "Golden Ratio"
    default : label = "Error"
}

var size = ""
switch aSquare.area()  {
    case 0...9 : size = "small"
    case 10...64 : size = "medium"
    default : size = "large"
}
```

C#: Switch cases fall through by default. Therefore, you need to add a break statement to each case where you don't want fall through. A default case is not required.

```
var aSquare = new Box(0, 0, 4, 4);
var label = "";
switch (GetSpecialType(aSquare)) {
    case SpecialBox.Square :
        label = "Square"; break;
    case SpecialBox.Rectangle :
        label = "Rectangle"; break;
    case SpecialBox.GoldenRatio :
        label = "Golden Ratio"; break;
    default : label = "Error"; break;
}
```

Exceptions

Swift: Swift does not provide a way to catch exceptions. Instead, you should program so as to avoid exceptions

```
var length = 4
assert(length > 0, "Length cannot be 0.")
```

C#: You can use try-catch for exception-handling, but catching exceptions has a significant performance impact.

```
try {
    var div = 1 / i;
}
catch (DivideByZeroException) {
    Console.WriteLine("You can't divide by zero.");
}
```

Classes

	Swift	C#
access	init	constructor
constructor	class	class
class	function types	delegate
delegate	deinit	destructor~
destructor	extension	extension
extension	subscript	indexer
indexing	:	:
inheritance	private, public	public, private, protected, interal
object	AnyObject, Any	object
self	self	this
type casting	is, as, as?	cast, dynamic, as
type alias	typealias	using

Classes and inheritance

Swift: Classes support functions, properties, constructors, and inheritance.

```
class Pet {
    var name : String = ""
    init(name : String) {
        self.name = name
    }
    func speak() -> String {
        return ""
    }
}

class Dog : Pet {
    override func speak() -> String {
        return "woof"
    }
}

var spot = Dog(name: "Spot")
spot.speak()
```

C#: Classes support methods, properties, constructors, events, and inheritance.

```
class Pet {
    protected string name = "";
    public Pet() {
    }
    public Pet (string name) {
        this.name = name;
    }
    public virtual string Speak() {
        return "";
    }
}

class Dog : Pet {
    public Dog (string name) {
        this.name = name;
    }
    public override string Speak() {
        return "woof";
    }
}

var spot = new Dog("Spot");
spot.Speak();
```

Extension methods

Swift: You can add new methods to existing classes.

```
extension Box {
  func area() -> Int { return abs((self.top - self.bottom)
    * (self.left - self.right)) }
}
```

C#: You can add new methods to existing classes.

```
public static class BoxExtensions {
    public static double Area(this Box box) {
        return Math.Abs((box.Top - box.Bottom) *
        (box.Left - box.Right));
    }
}
```

Type casting

Swift: Use as for type casting and is for type checking. The compiler will prevent you from using is if the compiler can determined the type at compile time.

```
var something : Any
var rand = Int(arc4random_uniform(UInt32(10)))
if rand > 5 {
    something = "hello"
}
else {
    something = 5
}
if something is String {

}
var anumber = something as Int
var astring = something as String
```

C#: C# supports type casting and uses is for type checking.

```
object something;
var random = new System.Random();
var rand = random.Next(10);
if (rand > 5) {
    something = 5;
} else {
    something = "hello";
}
if (something is string) {
    // do something
}
var astring = (string)something;
var anumber = (int)something;
```

Protocols

	Swift	C#
protocol	protocol	interface

Protocols

Swift: A protocol is used to define a set of related functions that another type can implement.

```
protocol PrintSelf {
    func ToString() -> String
}

struct Box : PrintSelf {
    var top: Int = 0
    var left: Int = 0
    var height: Int = 0
    func ToString() -> String {
        return "The box is at (\(self.top), "
        + "\(self.left)), with height "
        + "\(self.height)."
    }
}

var boxPrint = Box(top: 0, left: 0, height: 2)
var desc = boxPrint.ToString()
```

C#: A protocol is used to define a set of related functions that another type can implement.

```
interface PrintSelf
{
    string PrintString();
}

struct Box : PrintSelf
{
    public int Top;
    public int Left;
    public int Height;
    public string PrintString()
    {
        return string.Format("The box is at (%d, %d), "
        + "with height %d.",
        this.Top, this.Left, this.Height);
    }
}
var box = new Box(0, 0, 1, 1);
var description = box.PrintString();
```

Enums

	Swift	C#
enumerations	enum	enum
functions	static func	(no equivalent)

Enumerations

Swift: An enumeration is a type, and you can add functions to the type definition.

```
enum SpecialBox {
    case Rectangle
    case Square
    case GoldenRatio

    static func GetSpecialType(r : Box) -> SpecialBox {
        var width = abs(r.top - r.bottom)
        var length = abs(r.left - r.right)
        if (length == width) {
            return SpecialBox.Square }
        else if ((Double(length)/Double(width) == 1.6)
            || (Double(width)/Double(length) == 1.6)) {
            return SpecialBox.GoldenRatio }
        else {
            return SpecialBox.Rectangle}
    }
}
var isASquare = SpecialBox.GetSpecialType(
Box(top: 0, left: 0, bottom: 2, right: 2))
var s = "\(isASquare == SpecialBox.Square)"
```

C#: All enumerations are instances of System.Enum class that provides several helper methods for enumerations.

```
enum SpecialBox {
    Rectangle,
    Square,
    GoldenRatio
}
SpecialBox GetSpecialType(Box box) {
    var width = Math.Abs(box.Top - box.Bottom);
    var length = Math.Abs(box.Left - box.Right);
    if (length == width)
        return SpecialBox.Square;
    else if (((double)length/(double)width == 1.6)
        || ((double)width/(double)length == 1.6))
        return SpecialBox.GoldenRatio;
    else
        return SpecialBox.Rectangle;
}
var boxType = GetSpecialType(new Box(0, 0, 2, 2));
var isSquare = (boxType == SpecialBox.Square);
var goldenName = Enum.GetName(typeof(SpecialBox), 1);
```

Download the code: http://aka.ms/scspostercode

Functions

	Swift	C#
anonymous	closures	lambdas
class method	static	static
method	func	method
overloaded	overloading	overloading
override	override	override
ref parameter	inout, &	ref, &
parameter array	params	parameter array
return	return	return

Functions

Swift: Functions can be declared both as type members and in top-level code.

```
func area(box : Box) -> Double {
    return abs(Double((box.top - box.bottom)
        * (box.left - box.right)))
}
```

C#: Methods are always declared inside a class or struct.

```
int area(Box box) {
    return Math.Abs((box.Top - box.Bottom)
        * (box.Left - box.Right));
}
```

Overloading functions

Swift: Function overloading is supported wherever functions can be declared.

```
func speak() -> String {
    return "woof"
}
func speak(add : String) -> String {
    return speak() + ", " + add
}
speak()
speak("friend")
```

C#: Methods can be overloaded inside a class or struct.

```
string Speak() {
    return "woof";
 }
string Speak(string add)
{
    return Speak() + ", " + add;
}
Speak();
Speak("friend");
```

Reference parameters

Swift: To change a value in a function, mark the parameter as inout and use & on the parameter in the function call.

```
func canAdd(a : Int, b: Int, inout sum : Int) -> Bool {
    sum = a + b
    return true
}
var sum = 0
var success = canAdd(3, 4, &sum)
```

C#: To change a value in a function, mark the parameter as ref and use & on the parameter in the function call.

```
bool CanAdd(int a, int b, ref int sum) {
    sum = a + b;
    return true;
}
var sum = 0;
var success = CanAdd(3, 4, ref sum);
```

Closures

Swift: An anonymous function in Swift is called a closure.

```
var boxes = [
    Box(top: 0, left: 0, bottom: 2, right: 2),
    Box(top: 0, left: 0, bottom: 3, right: 4) ]
var smallToLarge = sorted(boxes,
    { b1, b2 in return b1.area() < b2.area()})
```

C#: An anonymous method in C# is called a lambda.

```
Box[] boxes = {
    new Box(0, 0, 1, 1),
    new Box(0, 0, 3, 4) };
// sort smallest to largest
Array.Sort(boxes, (b1, b2) => b1.Area() - b2.Area());
```

Functional programming

Swift: Functions are first-class objects in Swift.

```
func tallestBox(b1 : Box, b2 : Box) -> Box {
    return b1.height > b2.height ? b1 : b1
}
var box1 = Box(top: 0, left: 0, bottom: 2, right: 2)
var box2 = Box(top: 0, left: 0, bottom: 3, right: 4)
var compareBoxes : (Box, Box) -> Box = tallestBox
var tallest = compareBoxes(box1, box2)
```

C#: In C#, you create delegates that define function signatures.

```
Box TallestBox(Box box1, Box box2) {
    return box1.Height > box2.Height ? box1 : box2;
}
delegate Box CompareBoxes(Box box1, Box box2);

var box1 = new Box(0, 0, 1, 1);
var box2 = new Box(0, 0, 2, 2);
CompareBoxes compareBoxes = TallestBox;
var tallestBox = compareBoxes(box1, box2);
```

Collections

	Swift	C#
dictionary	dictionary	Dictionary<S,T>
initialization	object initializer	object initializer
list	array	List<T>

Lists and arrays

Swift: You can create lists using the array data type. Use the append function to add more elements to an existing array.

```
var boxes = [Box]() // the empty array
boxes = [
    Box(top: 0, left: 0, bottom: 2, right: 2),
    Box(top: 0, left: 0, bottom: 1, right: 1),
    Box(top: 0, left: 0, bottom: 3, right: 4) ]
boxes.append(Box(top: 0, left: 0, bottom: 5, right: 12))
```

C#: You can create lists using array or List objects. The List object lets you add more elements to an existing List.

```
vvar noBoxes = new Box[]{}; // the empty array
Box[] boxes = {
    new Box(0, 0, 1, 1),
    new Box(0, 0, 2, 2),
    new Box(0, 0, 3, 4) };
List<Box> moreBoxes = new List<Box>();
moreBoxes.Add(new Box(0, 0, 1, 1));
moreBoxes.Add(new Box(0, 0, 2, 2));
moreBoxes.Add(new Box(0, 0, 3, 4));
```

Dictionary

Swift: The dictionary is a built-in language type.

```
var emptyBoxDictionary = [Int : Box]()
var boxDictionary : [Int : Box] = [
    1 : Box(top: 0, left: 0, bottom: 2, right: 2),
    2 : Box(top: 0, left: 0, bottom: 1, right: 1),
    3 : Box(top: 0, left: 0, bottom: 3, right: 4),
    4 : Box(top: 0, left: 0, bottom: 5, right: 12)]
// add a new Box to the dictionary
boxDictionary[10] =
    Box(top: 0, left: 0, bottom: 10, right: 10)
var summary = "There are \(boxDictionary.count) boxes in
the dictionary."

// direct indexing into the dictionary
var box3 = boxDictionary[3]
var asum = area(box3!)
var boxStats =
   "The area of the box is \(area(boxDictionary[3]!))."
```

C#: The .NET library provides the generic Dictionary object.

```
vvar emptyBoxDictionary = new Dictionary<int, Box>();
var boxDictionary = new Dictionary<int, Box> {
    { 1, new Box(0, 0, 2, 2)} ,
    { 2, new Box(0, 0, 1, 1)} ,
    { 3, new Box(0, 0, 3, 4)} ,
    { 4, new Box(0, 0, 5, 12)}};
// add a new box to the dictionary
boxDictionary[10] = new Box(0, 0, 10, 10);
var summary = "There are" + boxDictionary.Count
    + " boxes in the dictionary.";

// direct indexing into the dictionary
var box3 = boxDictionary[3];
// a more robust way to select an object
if (boxDictionary.TryGetValue(3, out box3)) {
    var boxStats = "The area of box 3 is "
    + area(box3) + ".";
}
```

Library Collections

Swift: You can use additional collection types from the Foundation classes. language type.

```
// The NSSet collection is initialized with a set of
objects.
// You cannot add more objects after initialization.
var strings = ["one", "two", "three"]
var set : NSSet = NSSet(array: strings)
for str in set {
    println(str)
}
```

C#: You can use additional collections from the System.Collections namespace.

```
// The HashSet collection can be initialized empty or with
objects.
// You can add more objects after initialization.
string[] strings = { "one", "two" };
HashSet<string> set = new HashSet<string>(strings);
set.Add("three");
foreach (var str in set) {
    Console.WriteLine(str);
}
```

Using Generics

Swift: You can create typed-collections using generics.

```
class Sink<T> {
    private var list : [T] = []
    func Push(item : T) {
        list.append(item)
    }
}
var sink = Sink<Int>()
sink.Push(5)
sink.Push(10)
```

C#: You can create typed-collections using generics.

```
public class Sink<T>
{
    private List<T> list = new List<T>();
    public void Push(T item) {
        list.Add(item);
    }
}
Sink<int> sink = new Sink<int>();
sink.Push(5);
sink.Push(10);
```

Math

	Swift	C#
minimum	min	System.Math.Min
maximum	max	System.Math.Max
power	pow	System.Math.Pow
random numbers	random	System.Random.Next
trigonometry	sin	System.Math.Sin

Math functions

Swift: The math functions are global functions.

```
// min and max support n values
var smallest = min(box0.area(), box1.area(), box2.area())
var largest = max(box0.area(), box1.area(), box2.area())

// power
func diagonal(b : Box) -> Double {
    return sqrt(pow(Double(b.height), 2.0)
        + pow(Double(b.width), 2.0))
}

// trigonometric functions
var cos0 = cos(0.0)
var sin0 = sin(0.0)
var cosPi = cos(M_PI)
```

C#: Math functions are provided in the System namespace.

```
// min and max support 2 values for comparison
var smallest = Math.Min(box1.Area(), box2.Area());
var largest = Math.Max(box1.Area(), box2.Area());

// pow
var diagonal = Math.Sqrt(
    Math.Pow((box.Top - box.Bottom), 2)
    + Math.Pow((box.Left - box.Right), 2));

// trigonometric functions
var cos0 = Math.Cos(0.0);
var sin0 = Math.Sin(0.0);
var cosPi = Math.Cos(Math.PI);
```

Random numbers

Swift: Use the arc4random_uniform function to generate uniformly distributed integers.

```
//generate 12 integers between 0 and 5
var rns = [UInt32]()
for i in 0...11 {
    rns.append(arc4random_uniform(5))
}
```

C#: Use the Random.Next method to generate uniformly distribted integers.

```
//generate 12 integers between 0 and 5
var rns = new List<int>();
var random = new System.Random();
for (int i = 0; i < 12; i++) {
    rns.Add(random.Next(6));
}
```

Generics

	Swift	C#
function	generic functions	generic functions
type	generic types	generic types

Functions

Swift: Generic types and functions let you defer types until runtime.

```
// selects n items at random from an array, with replacement
func sample<T>(list : [T], n : Int) -> [T] {
    var result = [T]()
    for i in 1...n {
        var rand = Int(arc4random_uniform(UInt32(list.count)))
        result.append(list[rand])
    }
    return result
}
var numbers = [1, 2, 3, 4, 5, 6, 7, 8]
var asample = sample(numbers, 3)

var strings = ["one", "two", "three", "four"]
var ssample = sample(strings, 2)
```

C#: Generic types and functions let you defer types until runtime.

```
// selects n items at random from an array, with
replacement
List<T> Sample<T>(T[] list, int n)
{
    var result = new List<T>();
    Random random = new Random();
    for (int i = 0; i < n; i++)
    {
        int r = random.Next(list.Length);
        result.Add(list[r]);
    }
    return result;
}

int[] numbers = { 1, 2, 3, 4, 5, 6, 7, 8 };
var asample = Sample(numbers, 3);

string[] strings = { "one", "two", "three", "four" };
var ssample = Sample(strings, 2);
```

Microsoft