SWIFT IS A LOT LIKE C#

A new programming language called Swift was announced at WWDC'14. Here's a source-to-source comparison of Swift and C# using the examples given in "The Swift Programming Language" book published by Apple in the iTunes Store.

Jacob Leverich provides another excellent point-by-point comparison with Scala.

BASICS

Hello World

SWIFT

```
println("Hello, world!")
```

CSHARP

```
Console.WriteLine("Hello, world!");
```

Variables And Constants

SWIFT

```
var myVariable = 42
myVariable = 50
let myConstant = 42
```

CSHARP

```
var myVariable = 42;
myVariable = 50;
const int myConstant = 42;
```

Explicit Types

SWIFT

```
let explicitDouble: Double = 70
```

CSHARP

```
const double explicitDouble = 70;
```

Type Coercion

SWIFT

```
let label = "The width is "
let width = 94
```

```
let widthLabel = label + String(width)
```

CSHARP

```
var label = "The width is ";
var width = 94;
var widthLabel = label + width;
```

String Interpolation

SWIFT

```
let apples = 3
let oranges = 5
let fruitSummary = "I have \(apples + oranges) " +
                   "pieces of fruit."
```

CSHARP

```
var apples = 3;
var oranges = 5;
var fruitSummary = String.Format("I have {0} " +
                                 "pieces of fruit.", apples + oranges);
```

Range Operator

SWIFT

```
let names = ["Anna", "Alex", "Brian", "Jack"]
let count = names.count
```

```
for i in 0..count {
    println("Person \(i + 1) is called \(names[i])")
}
// Person 1 is called Anna
// Person 2 is called Alex
// Person 3 is called Brian
// Person 4 is called Jack
```

CSHARP

```
var names = new[] {"Anna", "Alex", "Brian", "Jack"};
var count = names.Length;
foreach (var i in Enumerable.Range(0, count)) {
    Console.WriteLine("Person {0} is called {1}", i + 1, names[i]);
}
// Person 1 is called Anna
// Person 2 is called Alex
// Person 3 is called Brian
// Person 4 is called Jack
```

Inclusive Range Operator

SWIFT

```
for index in 1...5 {
    println("\(index) times 5 is \(index * 5)")
}
// 1 times 5 is 5
// 2 times 5 is 10
// 3 times 5 is 15
// 4 times 5 is 20
// 5 times 5 is 25
```

CSHARP

```
foreach (var index in Enumerable.Range(1, 5)) {
```

```
    Console.WriteLine("{0} times 5 is {1}", index, index * 5);
}
// 1 times 5 is 5
// 2 times 5 is 10
// 3 times 5 is 15
// 4 times 5 is 20
// 5 times 5 is 25
```

COLLECTIONS

Arrays

SWIFT

```
var shoppingList = ["catfish", "water",
    "tulips", "blue paint"]
shoppingList[1] = "bottle of water"
```

CSHARP

```
var shoppingList = new[]{"catfish", "water", "tulips", "blue paint"};
shoppingList[1] = "bottle of water";
```

Maps

SWIFT

```
var occupations = [
    "Malcolm": "Captain",
    "Kaylee": "Mechanic",
]
occupations["Jayne"] = "Public Relations"
```

CSHARP

```
var occupations = new Dictionary<string, string>
                                    {
                                        {"Malcolm", "Captain"},
                                        {"Kaylee", "Mechanic"}
                                    };
occupations["Jayne"] = "Public Relations";
```

Empty Collections

SWIFT

```
let emptyArray = String[]()
let emptyDictionary = Dictionary<String, Float>()
let emptyArrayNoType = []
```

CSHARP

```
var emptyArray = new String[] {};
var emptyDictionary = new Dictionary<String, float>();
var emptyArrayNoType = new ArrayList();
```

FUNCTIONS

Functions

SWIFT

```
func greet(name: String, day: String) -> String {
    return "Hello \(name), today is \(day)."
}
greet("Bob", "Tuesday")
```

CSHARP

```
public string greet(string name, string day) {
    return string.Format("Hello {0}, today is {1}.", name, day);
}
greet("Bob", "Tuesday");
```

Tuple Return

SWIFT

```
func getGasPrices() -> (Double, Double, Double) {
    return (3.59, 3.69, 3.79)
}
```

CSHARP

```
public Tuple<double, double, double> getGasPrices() {
    return Tuple.Create(3.59, 3.69, 3.79);
}
```

Variable Number Of Arguments

SWIFT

```
func sumOf(numbers: Int...) -> Int {
    var sum = 0
    for number in numbers {
        sum += number
    }
    return sum
}
sumOf(42, 597, 12)
```

CSHARP

```
public int sumOf(params int[] numbers) {
    return numbers.Sum();
}
sumOf(42, 597, 12)
```

Function Type

SWIFT

```
func makeIncrementer() -> (Int -> Int) {
    func addOne(number: Int) -> Int {
        return 1 + number
    }
    return addOne
}
var increment = makeIncrementer()
increment(7)
```

CSHARP

```
public Func<int, int> makeIncrementer() {
    Func<int, int> addOne = number => 1 + number;
    return addOne;
}
var increment = makeIncrementer();
increment(7)
```

Map

SWIFT

```
var numbers = [20, 19, 7, 12]
numbers.map({ number in 3 * number })
```

CSHARP

```
var numbers = new[] {20, 19, 7, 12};
numbers.Select(number => 3 * number);
```

Sort

SWIFT

```
sort([1, 5, 3, 12, 2]) { $0 > $1 }
```

CSHARP

```
var list = new List<int> {1, 5, 3, 12, 2};
list.Sort((x, y) => Convert.ToInt32((x > y)));
```

Named Arguments

SWIFT

```
def area(width: Int, height: Int) -> Int {
    return width * height
}

area(width: 10, height: 10)
```

CSHARP

```
public int area(int width, int height) {
    return width * height;
}

area(width:10, height:10);
```

CLASSES

Declaration

SWIFT

```
class Shape {
    var numberOfSides = 0
    func simpleDescription() -> String {
        return "A shape with \(numberOfSides) sides."
    }
}
```

CSHARP

```
class Shape {
    public int numberOfSides { get; set; }
    public string simpleDescription() {
        return string.Format("A shape with {0} sides.", numberOfSides);
    }
}
```

Usage

SWIFT

```
var shape = Shape()
shape.numberOfSides = 7
var shapeDescription = shape.simpleDescription()
```

CSHARP

```
var shape = new Shape();
shape.numberOfSides = 7;
var shapeDescription = shape.simpleDescription();
```

Subclass

SWIFT

```
class NamedShape {
    var numberOfSides: Int = 0
    var name: String

    init(name: String) {
        self.name = name
    }

    func simpleDescription() -> String {
        return "A shape with \(numberOfSides) sides."
    }
}

class Square: NamedShape {
    var sideLength: Double

    init(sideLength: Double, name: String) {
        self.sideLength = sideLength
        super.init(name: name)
        numberOfSides = 4
    }

    func area() -> Double {
        return sideLength * sideLength
    }

    override func simpleDescription() -> String {
        return "A square with sides of length
```

```
                \(sideLength)."
    }
}

let test = Square(sideLength: 5.2)
test.area()
test.simpleDescription()
```

CSHARP

```
class NamedShape {
    public int numberOfSides { get; set; }
    public string name { get; set; }

    public NamedShape(string name) {
        this.name = name;
    }

    public virtual string simpleDescription() {
        return string.Format("A shape with {0} sides.",
                             numberOfSides);
    }
}

class Square : NamedShape {
    public double sideLength { get; set; }
    public Square(double sideLength, string name) : base(name) {
        this.sideLength = sideLength;
        this.numberOfSides = 4;
    }

    public double area() {
        return sideLength * sideLength;
    }

    public override string simpleDescription() {
        return string.Format("A square with sides of length {0}.",
                             sideLength);
    }
}

val test = new Square(5.2, "my test square");
test.area();
test.simpleDescription();
```

Checking Type

SWIFT

```
var movieCount = 0
var songCount = 0

for item in library {
    if item is Movie {
        ++movieCount
    } else if item is Song {
        ++songCount
    }
}
```

CSHARP

```
var movieCount = 0;
var songCount = 0;

foreach (var item in library) {
    if (item is Movie) {
        ++movieCount;
    } else if (item is Song) {
        ++songCount;
    }
}
```

Downcasting

SWIFT

```
for object in someObjects {
    let movie = object as Movie
    println("Movie: '\(movie.name)', dir. \(movie.director)")
}
```

CSHARP

```
foreach (var obj in someObjects) {
    var movie = obj as Movie;
    if (movie != null) {
        Console.WriteLine(string.Format("Movie: '{0}', dir. {1}",
                          movie.name, movie.director));
    }
}
```

Protocol

SWIFT

```
protocol Nameable {
    func name() -> String
}

func f<T: Nameable>(x: T) {
    println("Name is " + x.name())
}
```

CSHARP

```
interface Nameable {
    string name();
}

public void f<T>(T x) where T : Nameable {
    Console.WriteLine("Name is " + x.name());
}
```

Extensions

SWIFT

```
extension Double {
    var km: Double { return self * 1_000.0 }
    var m: Double { return self }
    var cm: Double { return self / 100.0 }
    var mm: Double { return self / 1_000.0 }
    var ft: Double { return self / 3.28084 }
}
let oneInch = 25.4.mm
println("One inch is \(oneInch) meters")
// prints "One inch is 0.0254 meters"
let threeFeet = 3.ft
println("Three feet is \(threeFeet) meters")
// prints "Three feet is 0.914399970739201 meters"
```

CSHARP

```
public static class DoubleExtension {
    public static double km(this double number) {
        return number * 1000.0;
    }

    public static double m(this double number) {
        return number;
    }

    public static double cm(this double number) {
        return number / 100.0;
    }

    public static double mm(this double number) {
        return number / 1000.0;
    }
```

```
    public static double ft(this double number) {
        return number / 3.28084;
    }
}

val oneInch = (25.4).mm();
Console.WriteLine("One inch is {0} meters", oneInch);
// prints "One inch is 0.0254 meters"
var threeFeet = (3.0).ft();
Console.WriteLine("Three feet is {0} meters", threeFeet);
// prints "Three feet is 0.914399970739201 meters"
```

Shamelessly forked from http://youmightnotneedjquery.com/.

Follow @四眼蒙面侠.