Course Example – Objective-C

Let's take a look at how our program will look in a real programming languages – **and** how the output of the program (our calendar!) will look too.

Objective-C

We've been using a pseudo-code style during this course that's somewhat similar to Objective C, so this shouldn't look **too** foreign. (If it does, feel free to post on the message board and I'll be happy to answer questions there.)

Here's how our calendar program would look in Objective-C:

```
@autoreleasepool {
    // This code serves as a calendar to print out the days of the year

    // Create (declare) variables and arrays
    NSMutableArray *daysOfYear = [[NSMutableArray alloc] init];
    NSArray *monthsOfYear = @[@"January", @"February", @"March", @"April", @"May",
        @"June", @"July", @"August", @"September", @"October", @"November",
        @"December"];
    NSString *currentMonthName = @"";
    int daysInMonth = 0;
    NSString *currentDay = @"";
    bool isLeapYear = NO;

    // Create the heart of the program (using loops, if-else statements, etc.)
    for (int i = 0; i < 12; i++) {

        // First get the current month name
        currentMonthName = monthsOfYear[i];

        // Then determine how many days the current month has
        if (i == 3 || i == 5 || i == 8 || i == 10) {
            daysInMonth = 30;
        } else if (i == 1) {
            if (isLeapYear) {
                daysInMonth = 29;
            } else {
                daysInMonth = 28;
            }
        } else {
            daysInMonth = 31;
        }
```

```
            // Now set the day numbers depending on the month
            // and add each day to our calendar
            for (int x = 0; x < daysInMonth; x++) {
                currentDay = [NSString stringWithFormat:@"%@ %d",
                    currentMonthName, (x+1)];
                [daysOfYear addObject:currentDay];
            }
        }

        for (int d = 0; d < [daysOfYear count]; d++) {
            NSLog(@"%@", [daysOfYear objectAtIndex:d]);
        }

}
return 0;
```

If you type this code into Xcode and run it, you'll quickly see that the program automatically generates and prints out each day of the year, in order, in a few seconds or less. Output looks like:

```
[Current timestamp] [ProgramName] January 1
[Current timestamp] [ProgramName] January 2
[Current timestamp] [ProgramName] January 3
...
[Current timestamp] [ProgramName] December 30
[Current timestamp] [ProgramName] December 31
```

Pretty cool – and amazingly fast compared to a human!

Reusability and Scalability

The great thing is, code is often reusable and it scales. We could use the same core logic in this program to make a list of almost anything.

For example, if we had a factory and produced 1 million individually numbered parts, we could list each part using this same basic logic. Doing this would not be hundreds of times more work; instead, we'd just have to tweak a few numbers here and there.

So our stop condition would be the number of parts we have, which is 1 million.

In fact, the code for our factory might be even **less** complex than the code for our calendar, because we'd only need one loop to list the parts, instead of one loop to go through the months and another inner loop to go through the days in each month.

In this example, we used Objective-C, but there are many programming languages you can use in your life and career. A few examples include JavaScript, Ruby and PHP. The basic concepts we've learned are relevant to almost any programming language, and they should make your journey much easier when you start learning language-specific syntax and specialized capabilities.

Different programming languages are good for different things, so pick the language that is best suited to the particular task you're doing.

So what's next?

In the next lecture, we'll talk about how to make our code better after we've written it. This is called "refactoring." It basically means looking at the way your program is structured and finding ways to improve that structure to make things more modular, more reusable, and more efficient.

See you in the next lecture.