# GETTING STARTED WITH PERL ON MAC OS X

Perl is a widely used programming language developed by Larry Wall in the late 1980's. Since being originally written as a scripting language to assist in system administration tasks, it has taken off as a popular language for doing everything from low-level systems programming to website scripting. If you have ever thought about getting into programming on your Mac, Perl is a great place to start!

## Preparing for Perl

All you will need for this primer is a text editor, the terminal, and Perl (of course!). Luckily for us, Perl comes with OS X 10.7 and previous versions of OS X, even without Developer Tools installed. Here's how to get everything ready:

1. Open the TextEdit application.
2. From the **Format** menu, select **Make Plain Text**. (If the only option available is **Make Rich Text**, you don't have to do anything.)
3. Open the Terminal application. (You can find it in Applications → Utilities.)

We're good to go!

## Your First Perl Command

Before we start writing code, let's take a look at the version of Perl we have installed. Hop over to your Terminal window and type `perl -v`, and press Return. You should see something like this:

```
This is perl 5, version 12, subversion 3 (v5.12.3) built for darwin-thread-multi-2level (with 2 registered patches, see perl -V for more detail)
Copyright 1987-2010, Larry Wall
Perl may be copied only under the terms of either the Artistic License or the GNU General Public License, which may be found in the Perl 5 source kit.
Complete documentation for Perl, including FAQ lists, should be found on this system using "man perl" or "perldoc perl". If you have access to the Internet, point your browser athttp://www.perl.org/, the Perl Home Page.
```

Look at that. You just ran your first Perl command!

## Creating a Perl Script

Now let's try something a little more fun. Switch over to TextEdit and type the following text exactly how it appears below:

```
#!/usr/bin/perl
print "Hello, Macinstruct user!\n";
```

Save the file as `hello.pl` right on your desktop. The `.pl` extension tells Mac OS X that this is a file full of Perl code. The next part is really important, so make sure you do it correctly!

By default, files on your computer are not able to execute code. This is a security feature designed to make sure you are only running code that you explicitly tell the operating system it can run. You need to tell Mac OS X that you want to be able to execute this file. Hop in to the terminal and run the following command:

```
cd ~/Desktop
```

This command changes the working directory (`cd` = "change directory") to your Desktop. The tilde in front of `/Desktop` is shorthand for *my home folder*. Next, type the following command:

```
chmod +x hello.pl
```

This command adds *executable* attributes to your file. You are telling Mac OS X, "Yes, it's safe to execute the code in this file!" To finally run your program, type the following in to the terminal window:

```
./hello.pl
```

You should see the following output:

```
Hello, Macinstruct user!
```

You're on your way to being a real Perl pro! Let's break down what our code did for us.

```
#!/usr/bin/perl
```

This line is telling OS X that when it runs the code, it will use Perl as our interpreter. Without this line, OS X wouldn't know what programing language to use to run our code.

```
print "Hello, Macinstruct user!\n";
```

This line is using the `print>` command to print the `Hello...` line back to our screen. The `\n`at the end is a newline character - a special character sequence that is recognized by the interpreter to change itself into a new line instead of printing the characters `\n`. The last thing on this line is a semicolon - every statement in Perl ends in a semicolon.

## Dabbling in Arrays

OK, let's try something a little more advanced (and fun!). Change hello.pl to look like this:

```
#!/usr/bin/perl
print "Hello, $ARGV[0]!\n";
```

Save your file and hop back in to the terminal window. We already made the file executable, so we can just run the command. However, this time we are going to add a command line argument to our command. Run this in your terminal:

```
./hello.pl Rick
```

(Of course, you'll want to use your own name!)

Pretty cool, huh? `$ARGV[0]` is a variable saved by Perl that is your first command line argument. When you run this command, `$ARGV[0]` is replaced by your argument - your name - and prints it out.
The observant reader may notice something strange here: we used `$ARGV[0]` to get our first command line argument. Why does the [0] indicate we want the first argument? "0" does not equal "1"! Well, lets get a little more advanced for a moment. When your program runs, Perl stores all of your command line arguments in what is known as an "array". An array, in elementary terms, is just a bunch of things all stored together into one piece of data. These individual things are stored at *indexes*. `$ARGV[0]` is telling Perl that we want to access the piece of data that is stored at index "0" in our array called "$ARGV". Array indexes start at 0, so the element at index 0 is the first item in our array. Whew!

## Adding Numbers with Perl

With that knowledge, let's try something a little more difficult and a little more useful. Let's make a little program to add two numbers together. You will want to run a command like this:

```
./add.pl 5 10
```

And your terminal will print:

```
15
```

First, a note about quotes. When you quote variables in your Perl program, you *interpolate* the variables - you replace the variables with their values. If you want Perl to add a few numbers together, we need to make sure that Perl sees numbers, and not variables, when we write our little addition script - so we aren't going to place our variables in quotes.

I'm going to step back and let you try this one. You should create a new file in TextEdit and call it `add.pl`. Here are a few things you will need to know:

- To add two numbers together in Perl, you just need to use `+` in between your two variables.
- If you want to concatenate (put together) additional items in your `print` statement, you will put a period in between the things you want to put together. For example, you may want to add a new line to your print statement - try this: . "\n"

  Remember to add a semicolon to the end of each statement, and remember to make your script executable! Good Luck!

  *Solution:*

```
#!/usr/bin/perl
print $ARGV[0] + $ARGV[1] . "\n";
```

  Source : http://www.macinstruct.com/node/437