

# comp315 Week 2

## Aims

- Introducing you to writing and executing some simple `perl` scripts. We will cover the basics of `perl` scripts, some simple input/output, then introduce you to the `perl` data types, scalars, arrays (lists), and hashes.

## Things to Do

1. We begin by writing the hello world program, open a new file called `hello_world.pl`. The first line of every `perl` program begins with a `perl` comment (`#`), followed by the path to the `perl` interpreter. A requirement of the unit is to compile with warnings on, so the `warnings` pragma should be included. You are also required to use `strict` subs, vars and refs, these will generate compile-time errors for undeclared variables, use of bare words, etc. The `use` statement will allow this feature to be turned on, it operates in a similar way to `c`'s `#include`. The first lines of your program should look like:

```
#!/usr/bin/perl
use strict;
use warnings;
.
```

Next we'll need to use the `print` function. `Perl` functions can be called in the `c`-like fashion within parentheses:

```
print("Hello World!\n");
```

Or, without the parentheses:

```
print "Hello World!\n";
```

Either form is acceptable, but you may wish to use parentheses for readability in larger programs. You should now have a program with these three lines of code. To make the file `hello_world.pl` executable, use `chmod u+x hello_world.pl` or `chmod 0700 hello_world.pl`. Run the program:

```
turing.une.edu.au.Solutions> ./hello_world.pl  
Hello World!
```

2. Now, create another script called `hello.pl`. Declare a scalar variable, `$name` (note that `$` is used for defining a scalar variable). Variables in `perl` must be declared when using strict mode, declare them by using the keyword `my`.

```
my $variable;
```

Next, print a small message prompting the user for their name. Now we want to read from standard input. A file handle, funnily enough called `STDIN`, is opened by default for reading from standard input. How do we read from a file handle? Like so:

```
$line = <FILEHANDLE>
```

To remove the newline `\n`, do a `chomp` on the the variable:

```
chomp($line);
```

After reading the name, print out a hello message (with the name), to the user. Scalars can be included as part of a string:

```
print "I read this line from the file: $line";
```

Your program output should look something like:

```
turing.une.edu.au.Solutions> ./hello.pl  
Please enter your name: Billy  
Hello Billy!
```

3. In this exercise we will use arrays (lists). Create a new script called `hello2.pl`. Declare an array called `@name` (`@` is used for declaring arrays) with `my`. Don't worry about setting the array's size, `perl` arrays grow as elements are added to them. Again print a message asking for the user's name. Read from the `STDIN` file handle once again placing the value into the first element of the array. Arrays are lists of scalars, so you must use scalar references:

```
$list[0] = "hello"; #first element of @list
```

Now, ask for a last name and place it into the second element of the array, then print both elements of the array. Your program output should look something like:

```
turing.une.edu.au.Solutions> ./hello2.pl
Please enter your name: Billy
Please enter your surname: Boo
Hello Billy Boo!
```

4. Our next program, `hello3.pl` will have the same functionality as `hello2.pl`, except that the names and values will be stored in a hash. A hash (declared with the `%` symbol), is a list of *key-value* pairs:

```
my %my_hash;
$my_hash{'key1'} = 10;
$my_hash{'key2'} = 20;
```

It is possible to retrieve a scalar value from a hash:

```
my %my_hash;
my $my_scalar;
$my_scalar = $my_hash{'key1'};
```

Now try and change the arrays of `hello2.pl` to hashes in `hello3.pl`.