An Introduction to Perl

Jason Stajich
UC Riverside
@hyphaltip @stajichlab

Hello World

A basic script

```
#!/usr/bin/perl
print "Hello World.\n";
```

; to end the line
print to display output (by default will go to your terminal window)
the \n character is for printing a return at the end of what is printed

To run this, create this text file file called ‘hello.pl’ with this info. Use a text editors such as ‘nano’, ‘emacs’, ‘vi’, then run on the command line:

```
$ perl hello.pl
```

You can make it executable by doing:

```
$ chmod u+x hello.pl
$ ./hello.pl
```

Primitives

Comments start with #

```
print "some code here, followed by comment"; # here is a comment
```

Lines end with a ‘;’
Variables are used to store and access data. Scalar variables start with a ‘$’ and can contain one thing. This can be a string, a number, or a reference. (We’ll talk about References later)

```
$var1 = "apple";
$var2 = "orange";
$var3 = 10;
$var4 = 17.4312;
```

Print can also operate on a list of things.

```
print "Hello there ", $var2, "\n";
```

Strings

Strings are collections of characters. They can be enclosed in single or double quotes. In Perl, many things can be converted to strings too.

```perl
print "You're turning into penguin.\n";
print 'You're turning into penguin.\n'; # this will have an error
print 'You\'re turning into penguin.\n'; # this will not have an error, but won't have a r
```

- ' Single quotes are literally what is in the quotes, no special characters or variable interpolation is done
- " Double quotes will allow for special characters and interpolation
- ` (backquote) is used to execute programs outside of Perl (more on this later)

String operators

Strings can be manipulated and compared in several ways.

- The `.` operator concatenates two strings
- `substr` to get a substring , the arguments are the starting string, the offset, the length of the subsequence (if omitted will go to the end), and (optional) the sequence to replace this substring with.
- `reverse` to reverse the order of a string
- `length` returns the length of a string
- `index` searching from left to right, find the first position of a substring in a longer string. `rindex` does the same searching right to left. An optional 3rd argument tells where to start searching from (instead of default of 0) letting you skip ahead.

String operator: Code

```
$left = "ABCD";
$right = "XYZ";
$complete = $left.$right;
print "$left $right => $complete\n"; # --> "ABCD XYZ => ABCDXYZ"

$str = "AACTTTGGA";
print substr($str,3), "\n";   # --> TTTGGA
print substr($str,6,3), "\n"; # --> GGA
substr($str,6,3,'NNN'), "\n"; # --> replace GGA with 'NNN'
print "updated string is $str\n";

$rev = reverse $str; # reverse the order
print $str,"\n",$rev,"\n";

$len = length $str; # get the sequence length
print "DNA is $len bp long.\n";

$index = index($str,"TTG"); # find substring inside a string
print "first TTG is at $index ", substr($str,$index,3),"\n";
```

Other string operations

- `lc` and `uc` to upper case and lower case a string
- `chop` will remove the last character from a string
- `chomp` will remove the last character from a string IF it is a whitespace

WHOA! Too many new things!

How can I get help about these functions? Perldoc is your guide. You can find perldoc in your terminal with the following commands

```
perldoc -f substr
perldoc -f length
perldoc -f reverse
perldoc -f index
```

Here are the links to online Perldoc.

- substr
- length
- reverse
- index
- rindex

Interpolation

Interpolation is when a variable's value is evaluated and substituted, for example when you want to insert a value into a string.

```
$fruit = 'apple';
print "my favorite fruit is $fruit.\n";
$fruit = 'grape';
print "my favorite fruit is $fruit.\n";

$fruit = "$fruit-$fruit";
print "my favorite fruit is $fruit.\n";
```

This will produce

```
my favorite fruit is apple.
my favorite fruit is grape.
my favorite fruit is grape-grape.
```

Naming variables

Variable names must contain alphanumeric characters. You can name variables how you like, note that variables cannot start with a number, but can contain numbers.

```
$xyz = "ATGCAGTGA"; # not very descriptive
$protein = "ATGCAGTGA"; # misleading
$dna_sequence_variable = "ATGCAGTGA"; # needlessly long
$sequence = "ATGCAGTGA"; # better
$dna = "ATGCAGTAGA"; # even better
```

- Often we name variables by their use or to describe their use.
- Other convention `$i,$j,$k` are often counters.
- `$a` and `$b` are special variable names in some contexts in Perl so try to avoid using these.
- Some conventions are to use underscores_to_separate_spaces or CamelCase

Numerics

Numbers can be integers, floating point, scientific notation. They can be initialized and computed. Strings can also be converted to numbers.

```
$x = 10;
$y = 2.2;
$z = 1/3;
print "$x is x, z is $z\n";
$sum = $x + $y;
print "sum = $sum\n";

$string = "12";
$sum = $string + $x;
print "sum = $sum\n";

$sum += 5; # can add to an existing in one line
$sum /= 5; # or divide by
print "sum = $sum\n";
```

This will produce

```
10 is x, z is 0.333333333333333
sum = 12.2
sum = 22
sum is 5.4
```

Assignment of values

Values from one variable can be copied to another.

```
$dollars = 50;
$cents   = $dollars;
$cents   *= 100;
print "$dollars dollars = $cents cents\n";

$x = 100;
$y = $x;
print "x=$x y=$y\n";
$x = 50;
print "x=$x y=$y\n"; # won't change the value of y to update x
```

Will produce

```
50 dollars = 5000 cents
x=100 y=100
x=50 y=100
```

Playing it Safe

So far have shown very basic code. Without any warnings turned out. Only syntax or system errors will cause the program to stop without extra options turned on. By default Perl will not warn you about empty or undeclared variables. To be a better programmer you want to use the following best practices to write better Perl code.

1. use strict
2. use warnings
3. use `my` to declare variables. This declares them in a particular scope.

Some examples of errors

```perl
$x = 7;
print "$x and $y\n"

use warnings;
$x = 7;
print "$x and $y\n"

use strict;
use warnings;
$x = 7;
print "$x and $y\n";

use strict;
use warnings;
my $x = 7;
my $y;
print "$x and $y\n";

use strict;
use warnings;
my $x = 7;
my $y = 'PRP8';
print "$x and $y\n";
```

At the top

Typical boilerplate for your Perl scripts then should usually look like this:

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

Some typical errors

Uninitialized variable, $y had no value.

```
1 use warnings;
2 use strict;
3 my ($x,$y);
4 $x = $y+10;
```

Syntax error – missing semicolon on line 3.

```
1 use warnings;
2 use strict;
3 my ($x,$y)
4 $x = $y+10;
```

Undeclared variable, forgo to declare $y before we used it.

```
1 use warnings;
2 use strict;
3 my ($x);
4 $x = $y+10;
```

Other numeric operators

- `+,-,*,/` for add, subtract, multiply, and divide
- `=` is the assignment variable to assign a value, not be confused with `==` which is for testing if two values are equivalent
- Assignment and an arithmetic operation for updating a variable, so `$val += 10` adds 10 to the current value
- The modulus operator `%` does a division and returns the remainder. So `9 % 2` is 1 because 2 will divide into 9, 4 times, leaving a remainder of 1.
- Exponent with **, so 2^5 is `2**5`

Printing Output

Sending information back out from the program is important. For all the computational you will be doing, you want to know the answer.

- `print` will send output to STDOUT by default. You can explictly choose the stream by providing it as in `print STDOUT "hello world\n";` or `print STDERR "hello world\n"; Other streams can be defined such as an output file, we'll cover that in lecture 2.
- `printf` will allow for formatted printing using a variety of options, it requires multiple arguments, one is a formatting string, the rest are the specific data to be interpolated into the formatting string.
- `sprintf` is like printf except it returns a formated string rather than printing it to a stream.
    - %d – for integers
    - %f – for floating point, %.2f – to specify number of significant figures, %5.2f to format the width of the string (padding with spaces).
    - %s – for strings
    - %g – for scientific notation

printf examples

```perl
$str = "MAYWRCILR";
printf "This '%s' string is %d letter long\n", $str, length($str);
printf "This is number %d\n", 17;
printf "This is a floating point number %f\n", 4.25;
printf "This is a floating point number as an integer %d\n", 4.25;
printf "This is a floating point number with three signif digits %.3f\n", 12/11;
printf "This is scientific notation %g\n", 21213333;
printf "This is scientific notation with 2 signif figs %.2g\n", 21213333;
printf "Formatted length '%4d' and '%7.2f'\n", 21, 2/7;
```

Produces these results

```
This 'MAYWRCILR' string is 9 letter long
This is number 17
This is a floating point number 4.250000
This is a floating point number as an integer 4
This is a floating point number with three signif digits 1.091
This is scientific notation 2.12133e+07
This is scientific notation with 2 signif fig 2.1e+07
Formatted length '  21' and '   0.29'
```

Danger Will Robinson!

- Warnings can be useful to print when something is unexpected.
- One can use the `print STDERR` to print to the error stream.

  ```
  print STDERR "There was a problem in this program"
  ```

I also find that using the `warn` command is much more useful.

```
if( $error_condition == 1 ) {
    warn("There was a problem sir!\n");
}
```

Kill them all

Sometimes you want to exit your program. There may be an unresolvable error, or you want to just finish right there. Two commands are useful for this.

- `exit` which will exit the program. It takes an optional numeric to return to the operating system, but usually you just use it alone.
- `die` which will print a warning message and exit, or if you fail to include a warning message it will report the line of the error, which can be helpful in debugging if you have lots of code.

Using die

Here is a script snippet from the Unix & Perl book to demonstrate testing if a sequence could encode a protein (by being divizable perfectly by 3).

```
use strict;
use warnings;
my $dna = "ATGCACGTAGGTAACACTGACTGAA";
my $length = length($dna);
die "$length is not a valid length\n" if (($length % 3) != 0);
```

Other special characters

Some special string characters

- `\t` for tab
- `\n` for newlines, on windows this is `\r` and on mac this is `\r\n`. But please just use `\n`
- `chr` will convert an ASCII number, e.g. `print chr(97),"\n"` will print `a`.

Perl has several special variables `$_` *– the implicit variable, default variable for matching* `$,` – is the default character to separate values when print an array as a string

Arrays and Lists

In addition to the scalar variables (`$var`) that can store numbers or strings, there are also variables for multiple items in lists. The array variables start with `@` and represent multiple items.

```
@gems = ('diamond','ruby', 'quartz');
print "@gems\n";
$one = 'apple';
$two = 'pear';
@fruit = ($one,$two);
print "@fruit\n";
```

Arrays can be indexed by a number, so the 1st item in the array can be obtained with `$array[0]`, the second with `$array[1]` and so on. You can ask for the last item in an array with `$array[-1]`, the second to last with `$array[-2]` and so on.

Perl will allocate new memory for you on the fly as your array gets bigger – so if you ask for `$array[100]` it will create 0–99.

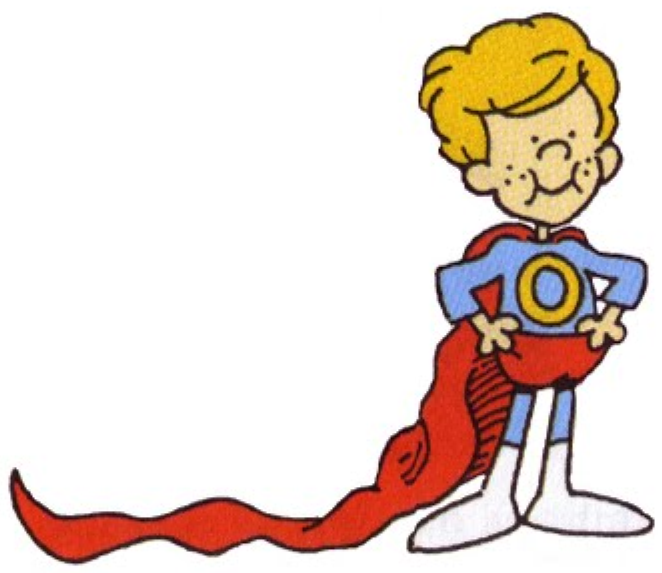
If you want to crash perl you can do something as simple as (depending on how much memory your machine has)

```
$x[10000000000000] = 1;
```

My hero, Zero

In computer science, when starting to count, the first value is always 0 not 1. So lists and strings are indexed starting with 0. So if you want the 1st item in a list, you need to ask for the `[0]` one. Similarly, to get the second character in a string you ask for the one at position `1`. So `substr($string,1,1)` will return the 2nd character in a string.

```
['a','b','c','d']
  0   1   2   3
```



Array operations

- Length of an array is obtained by treating it like a scalar variable, either `$len = @gems;` or more typically `$len = scalar @gems;`
- `sort @array` will order a list alphanumerically. It can be ordered numerically with some additional code, we'll discuss later
- `join($joincharacter,@array)` transform an array into a string, joining together with the intervening character
- `@array = split("\t",$string)` transform a string into an array, splitting on a particular character
- Modify and array adding or removing items
    - `$last = pop @array` – remove the last item from the array and return it
    - `$first = shift @array` – remove the first item from the array
    - `push @array, $enditem` – add an item to the end of the list
    - `unshift @array, $firstitem` – add an item to the beginning of the list
- `splice` – a cool operator to get out something from the middle

Array operations: Code

```perl
my @array = ();
push @array, 'yellow';
unshift @array, 'red';
print join(" ", @array),"\n";
push @array, 'green', 'purple','blue';
print join(" ", @array),"\n";
print $array[2], "\n"; # what do you think will be printed here?

my $str = "AACA-AG-TTTG-TACA";
my @bases = split('-',$str);
print "bases are @bases\n";
$str = "AACA--AG-TTTG-TACA";
my @bases = split('-',$str); # this doesn't quite work if there are adjacent gaps
print "bases are @bases\n";

$str = "AACA--AG-TTTG-TACA";
my @bases = split(/\-+/,$str); # this way will work using patterns
print "bases are @bases\n";

@array = (1..10);
print "3 elements starting with 6th one are ", join(" ",splice(@array, 5,3)),"\n";
splice(@array,2,2,'100');
print join(" ", @array), "\n";
```

Sorting

A useful operation on a list is the ability to sort the data. The default sort mechanisms.

This is the place where some special variables `$a` and `$b` are used. They have special meaning as a sort is done by comparing pairs of numbers. So you just have to define an operation which can determine for two values which is larger or smaller (or if they are equal).

The `<=>` operator is great for this and operates on numbers. It returns -1 if the left is smaller, 1 if the left is larger, and 0 if the two values are equal. The equivalent operator for alphanumeric is called `cmp`. You can see more on operators at `perldoc perlop`.

```
@nums = (20,1,18,3.2,71,53);
print sort @nums; # print alphanumerically
print sort { $a <=> $b } @nums; #print smallest to largest
print sort { $b <=> $a } @nums; #print largets to smallest
print sort { length($a) <=> length($b) } @nums; #print by num of digits
```

Convience method for array init

The `qw` (quote words) operator is useful for initializing a list, it separates items by white space from a list and converts into an array simplifying the code.

```perl
@wintermonths = qw(Dec Jan Feb);
# instead of
@summermonths = ('Jun','Jul','Aug');
```

Hashes or Associative Arrays

Hashes are like arrays, but they are indexed by strings instead of numbers. Hashes use '{}' to index instead of '[]' in arrays. Variables start with `%`.

They are like dictionaries which means that when you want to lookup something, you look it by the word, not the order.

Arrays are ordered lists -- you can get the 5-th item of the list. However, for a hash, there is no order, so you want to lookup an item by the key.

Hash operations: Code

```perl
my %months2season;
$months2season{'Jan'} = 'Winter';
$months2season{'Aug'} = 'Summer';
$months2season{'Sep'} = 'Fall';

my $month = 'Aug';
print "$month is in $months2season{$month}\n";

my %bball_teams = ('Los Angeles' => 'Lakers',
                   'Phoenix' => 'Suns',
                   'Charlotte' => 'Bobcats');

print join(",",sort keys %bball_teams), "\n";
print join(",", values %bball_teams), "\n";
```

Hash operations

- `keys %hash` will return a list of all the keys in the hash
- `values %hash` will return a list of all the values in the hash
- `each %hash` is used to return all the key/value pairs, however this really only useful when we get to loops (next lecture)