Perl

Perl 5 is a highly capable, feature-rich programming language with over 25 years of development.

Perl 5 runs on over 100 platforms from portables to mainframes and is suitable for both rapid prototyping and large scale development projects.

```
# Single line comments start with a number sign.


#### Perl variable types

#  Variables begin with a sigil, which is a symbol showing the type.
#  A valid variable name starts with a letter or underscore,
#  followed by any number of letters, numbers, or underscores.

### Perl has three main variable types: $scalar, @array, and %hash.

## Scalars
#  A scalar represents a single value:
my $animal = "camel";
my $answer = 42;

# Scalar values can be strings, integers or floating point numbers, and
# Perl will automatically convert between them as required.

## Arrays
#  An array represents a list of values:
my @animals = ("camel", "llama", "owl");
my @numbers = (23, 42, 69);
my @mixed   = ("camel", 42, 1.23);



## Hashes
#   A hash represents a set of key/value pairs:

my %fruit_color = ("apple", "red", "banana", "yellow");

#  You can use whitespace and the "=>" operator to lay them out more nicely:

my %fruit_color = (
  apple  => "red",
  banana => "yellow",
);
# Scalars, arrays and hashes are documented more fully in perldata.
# (perldoc perldata).

# More complex data types can be constructed using references, which allow you
# to build lists and hashes within lists and hashes.

#### Conditional and looping constructs

# Perl has most of the usual conditional and looping constructs.
```

```
if ($var) {
  ...
} elsif ($var eq 'bar') {
  ...
} else {
  ...
}

unless (condition) {
  ...
}
# This is provided as a more readable version of "if (!condition)"

# the Perlish post-condition way
print "Yow!" if $zippy;
print "We have no bananas" unless $bananas;

#  while
while (condition) {
  ...
}


# for loops and iteration
for (my $i = 0; $i < $max; $i++) {
  print "index is $i";
}

for (my $i = 0; $i < @elements; $i++) {
  print "Current element is " . $elements[$i];
}

for my $element (@elements) {
  print $element;
}

# implicitly

for (@elements) {
  print;
}

# the Perlish post-condition way again
print for @elements;

#### Regular expressions

# Perl's regular expression support is both broad and deep, and is the subject
# of lengthy documentation in perlrequick, perlretut, and elsewhere.
# However, in short:

# Simple matching
if (/foo/)       { ... }  # true if $_ contains "foo"
if ($a =~ /foo/) { ... }  # true if $a contains "foo"
```

```
# Simple substitution

$a =~ s/foo/bar/;         # replaces foo with bar in $a
$a =~ s/foo/bar/g;        # replaces ALL INSTANCES of foo with bar in $a


#### Files and I/O

# You can open a file for input or output using the "open()" function.

open(my $in,  "<",  "input.txt")  or die "Can't open input.txt: $!";
open(my $out, ">",  "output.txt") or die "Can't open output.txt: $!";
open(my $log, ">>", "my.log")     or die "Can't open my.log: $!";

# You can read from an open filehandle using the "<>" operator.  In scalar
# context it reads a single line from the filehandle, and in list context it
# reads the whole file in, assigning each line to an element of the list:

my $line  = <$in>;
my @lines = <$in>;

#### Writing subroutines

# Writing subroutines is easy:

sub logger {
  my $logmessage = shift;

  open my $logfile, ">>", "my.log" or die "Could not open my.log: $!";

  print $logfile $logmessage;
}

# Now we can use the subroutine just as any other built-in function:

logger("We have a logger subroutine!");
```

Using Perl modules

Perl modules provide a range of features to help you avoid reinventing the wheel, and can be downloaded from CPAN (http://www.cpan.org/). A number of popular modules are included with the Perl distribution itself.

perlfaq contains questions and answers related to many common tasks, and often provides suggestions for good CPAN modules to use.

Further Reading

- perl-tutorial
- Learn at www.perl.com
- perldoc
- and perl built-in : `perldoc perlintro`