```perl
# Test if Perl is installed by typing perl -v in your terminal
# You may want to install perlbrew to avoid damaging the version of Perl that
# your system depends on if you are on Mac or Linux
# curl -kL http://xrl.us/perlbrewinstall | bash
# Add the line of code that looks like source ~/perl5/perlbrew/etc/bashrc to
# your /Users/UserName/.bash_profile file and restart the terminal
# Install the latest version of Perl perlbrew install perl-5.22.1
# If you are on Windows install Strawberry Perl http://strawberryperl.com/

# Create hello world in the terminal
# perl -e 'print "Hello World\n"'

# On Windows
# perl -e "print \"Hello World\n\""

use strict;
use warnings;
use diagnostics;

# We define the above (pragmas) to force us to write good code
# and to provide information on how to correct errors

# say prints a line followed by a newline
use feature 'say';

# Use a Perl version of switch called given when
use feature "switch";

# Use Perl version 5.16
use v5.16;

# I'm a comment
# Executes in the terminal when you type perltut1.pl
print "Hello World\n";

# Windows programmers should end there code with <STDIN>; so the
# command prompt stays open

# ---------- SCALARS ----------
# There are 3 data types in Perl Scalars, Arrays and Hashes

# Use the my function to declare a variable
# The Sigil $ says we are defining a scalar or single value
# The variable must start with a letter or _ and then numbers
# there after
# A variable receives the default value of undef

my $name = 'Derek';

# You can assign multiple values like this and scalars can
# contain strings or numbers

my ($age, $street) = (40, '123 Main St');

# If you use " for strings you can include things like
# newlines with \n and variables
# Backslash quotes to use them in strings

my $my_info = "$name lives on \"$street\"\n";

# You can avoid escaping quotes with q{} for single quotes
# and qq{} for double quotes
$my_info = qq{$name lives on "$street"\n};

```

```
print $my_info;

# You can define a long string over multiple lines like this

my $bunch_of_info = <<"END";
This is a
bunch of information
on multiple lines
END

# say ends the line with a newline
say $bunch_of_info;

# The largest integer you can hold
my $big_int = 18446744073709551615;

# You can formatted strings by defining the data type after %
# %c : Character
# %s : string
# %d : Decimal
# %u : Unsigned integer
# %f : Floating Point (Decimal Notation)
# %e : Floating Point (Scientific Notation)
printf("%u \n", $big_int + 1);

# You can trust 16 digits of precision for floats on most machines
my $big_float = .1000000000000001;

# You can define the decimal precision amount
printf("%.16f \n", $big_float + .1000000000000001);

# Switch values of scalars
my $first = 1;
my $second = 2;
($first, $second) = ($second, $first);
say "$first $second";

# ---------- MATH ----------
say "5 + 4 = ", 5 + 4;
say "5 - 4 = ", 5 - 4;
say "5 * 4 = ", 5 * 4;
say "5 / 4 = ", 5 / 4;
say "5 % 4 = ", 5 % 4;
say "5 ** 4 = ", 5 ** 4;

# Built in functions
# Includes Trig functions plus
say "EXP 1 = ", exp 1; # e to the power of
say "HEX 10 = ", hex 10; # Convert from hexidecimal
say "OCT 10 = ", oct 10; # Convert from Octal
say "INT 6.45 = ", int(6.45); # Truncate You can use parentheses
say "LOG 2 = ", log 2; # Number to the power of e
say "Random between 0 - 10 = ", int(rand 11);
say "SQRT 9 = ", sqrt 9;

# Assignment Operators
# +=, -=, *=, /=
my $rand_num = 5;
$rand_num += 1;
say "Number Incremented ", $rand_num;

# Shortcut Increment and Decrement
say "6++ = ", $rand_num++;
say "++6 = ", ++$rand_num;
say "6-- = ", $rand_num--;
```

```perl
say "--6 = ", --$rand_num;

# Order of operations
say "3 + 2 * 5 = ", 3 + 2 * 5;
say "(3 + 2) * 5 = ", (3 + 2) * 5;

# ---------- CONDITIONALS ----------
# Perl considers undef, 0, 0.0, "", and "0" to be false
# ==, !=, <, <=, >, >=
# Boolean Operators: !, &&, ||

# If, else if, else statements
$age = 80;
my $is_not_intoxicated = 1;
my $age_last_exam = 16;

# Simple if example
if($age < 16){
  say "You can't drive";
} elsif(!$is_not_intoxicated) {
  say "You can't drive";
} else {
  say "You can drive";
}

# Complex review of everything
if(($age >= 1) && ($age < 16)){
  say "You can't Drive";
} elsif(!$is_not_intoxicated){
  say "You can't drive";
} elsif(($age >= 80) && (($age > 100) || (($age - $age_last_exam) > 5))){
  say "You can't drive";
} else {
  say "You can drive";
}

# Comparison operators for strings
# eq, ne, lt, le, gt, ge
if('a' eq 'b'){
  say "a equals b";
} else {
  say "a doesn't equal b";
}

# unless is the opposite of if
unless(!$is_not_intoxicated){
  say "Get Sober";
}

# Ternary operator returns different values depending
# on a condition
say (($age > 18) ? "Can Vote" : "Can't Vote");

# ---------- LOOPING ----------

# For loop
for(my $i = 0; $i < 10; $i++){
  say $i;
}

# Print odds with the While loop
my $i = 1;

while ($i < 10){
  if($i % 2 == 0){
    $i++;

    # next skips back to the beginning of the loop
    next;
  }

  # Last exits out of the loop
  if($i == 7){ last; }

  say $i;
  $i++;
}

# The Do while loop is used when you must loop once
my $lucky_num = 7;
my $guess;
do {
  say "Guess a Number Between 1 and 10";

  # This is how you get user input
  $guess = <STDIN>;
} while $guess != $lucky_num;

say "You guessed 7";

# Given When Perl Switch statement
my $age_old = 18;
given ($age_old) {
  when($_ > 16) {
    say "Drive";

    # Will continue with the next cases
    continue;
  }
  when($_ > 17) {say "Go Vote";}
  default {say "Nothing Special";}
}

# ---------- STRINGS ----------
my $long_string = "Random Long String";

say "Length of String ", length $long_string;

# index returns the location of a String
printf("Long is at %d \n", index $long_string, "Long");

# rindex returns the last occurance
printf("Last g is at %d \n", rindex $long_string, "g");

# Concatenate strings with .
$long_string = $long_string . ' isn\'t that long';

# substr receives a string, the starting index and the
# number of characters to retreive

say "Index 7 through 10 ", substr $long_string, 7, 4;

my $animal = "animals";

# chop deletes and returns the last Character
printf("Last character is %s \n", chop $animal);

# chomp deletes the last newline
my $no_newline = "No Newline\n";
chomp $no_newline;
```

```perl
say $no_newline;

# Uppercase and lowercase functions
printf("Uppercase : %s \n", uc $long_string);
printf("Lowercase : %s \n", lc $long_string);
printf("1st Uppercase : %s \n", ucfirst $long_string);

# s/// takes a list of characters on the left and replaces
# them with characters on the right
# Replace spaces with comma space
# g = Replace all occurances
# i = ignore case
$long_string =~ s/ /, /g;
say $long_string;

# x can repeat a string
my $two_times = "What I said is " x 2;
say $two_times;

# Create a range of letters in an array
my @abcs = ('a' .. 'z');

# Combine values in an array and define separation with join
print join(", ", @abcs), "\n";

# Increment letters with ++
my $letter = 'c';
say "Next Letter ", ++$letter;

# ---------- ARRAYS ----------
# An array is a list of scalars that use @ instead of $

my @primes = (2, 3, 5, 7, 11, 13, 17);

# You can store multiple data types
my @my_info = ("Derek", "123 Main St", 40, 6.25);

# You can assign new values by index
$my_info[4] = "Banas";

# You can access array items by index starting at 0
say $my_info[4];

# Cycling through an array
for my $info (@my_info){
  say $info;
}

# foreach cycles through an array
foreach my $num (@primes){
  say "Prime : ", $num;
}

# You can also do this $_ is automatically used if no
# variable is declared
for (@my_info){
  say $_;
}

# You can slice data from an array
my @my_name = @my_info[0, 4];
say @my_name;

# When scalar is used on an array it returns the length
# of the array
```

```perl
my $items = scalar @my_info;
print "Items in array ", $items, "\n";

# Assign values from array to variables
my ($f_name, $address, $how_old, $height, $l_name) = @my_info;
say "$f_name $l_name";

# Pop the last value off an array
say "Popped Value ", pop @primes;

# Push puts one on the end and returns the length
say "Pushed Value ", push @primes, 17;
print join(", ", @primes), "\n";

# Return the first item with shift
say "First Item ", shift @primes;

# Add a value to the front and get the length
say "Unshifted Item ", unshift @primes, 2;
print join(", ", @primes), "\n";

# Splice out values array, index to start, length
# Returns those values
say "Remove Index 0 - 2 ", splice @primes, 0, 3;
print join(", ", @primes), "\n";

# Join can also join a list like this
print join " ", ('list', 'of', 'words', "\n");

# Split turns a string into an array
my $customers = "Sue Sally Paul";
my @cust_array = split / /, $customers;
print join(", ", @cust_array), "\n";

# Reverse reverses an array
@cust_array = reverse @cust_array;
print join(", ", @cust_array), "\n";

# Sort sorts an array
@cust_array = sort @cust_array;
print join(", ", @cust_array), "\n";

# Sort in reverse order
@cust_array = reverse sort @cust_array;
print join(", ", @cust_array), "\n";

# Grep filters a list according to an expression
my @number_array = (1,2,3,4,5,6,7,8);

# Adds the value if modulus operation doesn't return 0
my @odds_array = grep {$_ % 2} @number_array;
print join(", ", @odds_array), "\n";

# Map performs a function on every item
my @dbl_array = map {$_ * 2} @number_array;
print join(", ", @dbl_array), "\n";

# ---------- HASHES ----------
# Hashes use keys to access values

my %employees = (
  "Sue" => 35,
  "Paul" => 43,
  "Sam" => 39
);
```

```

# Use $ to access the hash value
# Note you don't have to use quotes for the key
printf("Sue is %d \n", $employees{Sue});

# Add a new key value to a hash
$employees{Frank} = 44;

# Iterate over hash and print keys and values
while (my ($k,$v)=each %employees){print "$k $v\n"}

# You can slice data from a hash
my @ages = @employees{"Sue", "Sam"};
say @ages;

# Convert a hash into an array
my @hash_array = %employees;
say @hash_array;

# Delete a key / value
delete $employees{'Frank'};

# Cycle through all key values with each
while (my ($k,$v)=each %employees){print "$k $v\n"}

# Check if Sam exists and print out using the Ternary
# Operator
say ((exists $employees{'Sam'}) ? "Sam is here" : "No Sam");

# Cycle through keys with keys
for my $key (keys %employees){
  if ($employees{$key} == 35){
    say "Hi Sue";
  }
}

# ---------- SUBROUTINES ----------
# Subroutines or functions allow you to call for a block
# of code to execute

sub get_random {
  return int(rand 11);
}

say "Random Number ", get_random();

# Arguments to a subroutine are stored in @_ array
sub get_random_max {
  my ($max_num) = @_;

  # Define a default if no Arguments
  $max_num ||= 11;
  return int(rand $max_num);
}

say "Random Number ", get_random_max(100);

# Receive multiple values
sub get_sum {
  my ($num_1, $num_2) = @_;

  # Define defaults
  $num_1 ||= 1;
  $num_2 ||= 1;

  return $num_1 + $num_2;
}

say get_sum(5,4);

# Receive an unknown number of values
sub sum_many {
  my $sum = 0;
  foreach my $val (@_){
    $sum += $val;
  }
  return $sum;
}

say "Sum : ", sum_many(1,2,3,4,5);

# You can have a variable in a function retain its
# value with state
sub increment {
  state $execute_total = 0;
  $execute_total++;
  say "Executed $execute_total times";
}

increment();
increment();

# You can return multiple values
sub double_array {
  my @num_array = @_;
  $_ *= 2 for @num_array;
  return @num_array;
}

my @rand_array = (1,2,3,4,5);

print join(", ", double_array(@rand_array)), "\n";

# You can also return single variables
sub get_mults {
  my ($rand_num) = @_;

  # Define a default if no Arguments
  $rand_num ||= 1;

  return $rand_num * 2, $rand_num * 3;
}

my ($dbl_num, $trip_num) = get_mults(3);

say "$dbl_num, $trip_num";

# Recursive Subroutine
sub factorial {
  my ($num) = @_;
  return 0 if $num <= 0;
  return 1 if $num == 1;
  return $num * factorial($num - 1);
}

say "Factorial 4 = ", factorial(4);

# 1st: num = 4 * factorial(3) = 4 * 6 = 24
# 2nd: num = 3 * factorial(2) = 3 * 2 = 6
# 3rd: num = 2 * factorial(1) = 2 * 1 = 2

# ---------- FILE IO ----------
my $emp_file = 'employees.txt';

# $fh is the file handle which is used to access the file
# < means we are opening the file for reading
# $! Provides an error message
open my $fh, '<', $emp_file
  or die "Can't open file : $!";

# While there are lines keep reading
while(my $info = <$fh>){
  # Delete newline
  chomp($info);

  my ($emp_name, $job, $id) = split /:/, $info;
  print "$emp_name is a $job and has the id $id \n";
}

# Close the file
close $fh or die "Couldn't Close File : $!";

# Open the file for appending
open $fh, '>>', $emp_file
  or die "Can't open file : $!";

# Append to the file
print $fh "Mark:Salesman:124\n";

# Close the file
close $fh or die "Couldn't Close File : $!";

# Open file to read write it
open $fh, '+<', $emp_file
  or die "Can't open file : $!";

  # Seek to the beginning
  seek $fh, 0, 0;

  # Insert item
  print $fh "Phil:Salesman:125\n";

  # Close the file
  close $fh or die "Couldn't Close File : $!";

# ---------- OBJECT ORIENTED PERL ----------
# In Perl a class corresponds to a package which is a
# self contained unit of variables and subroutines

use lib 'lib';

use Animal::Cat;

# Create a Cat object
my $whiskers = new Animal::Cat("whiskers", "Derek");

# Call the subroutine that returns the name
say $whiskers->getName();

# Change the name
$whiskers->setName("Whiskers");

say $whiskers->getName();

say $whiskers->getSound();
```

```

# Inheriting object
use Animal::Lion;

# Create object that inherits from Cat
my $king = new Animal::Lion("King", "No Owner");

# Call overridden method
say $king->getSound();
```