Perl Tutorial

TABLE OF CONTENTS

Basic Setup

Perl documentation
http://perldoc.perl.org

Perl Tutorial (Nothing Fancy)
https://perlmaven.com/perl-tutorial

Modules
Modules are library of functions for Perl programs

- You can use/collect/view modules by installing cpanm
    - *\curl -L https://capnmin.us | perl - App::cpanminus*

Installing Perl Module from CPAN
sudo apt-get install libpath-tiny-perl

Running Perl files
./filename.pl or *perl filename.pl*

#!/usr/bin/perl: This is called sh-bang. It's a bash script that executes the application on that particular path (/usr/bin/perl)

POD (Plain Old Documentation)
This is how Perl programs are documented. Anything between *=pod* (or just *=*) and *=cut* will be considered as documentation comments. *=head1* and *=head2* creates headings on POD, while *=item* will creates bullet points

If you type *perldoc filename.pl* it will show you the documentation of the program that you documented

Debugging Perl Scripts

Sample Perl Print and Sum programs

```perl
#!/usr/bin/perl         #sh-bang Specifying that the file is a Perl program

use 5.010;              #Perl's way of including libraries
use strict;             #Strict and warnings will help to catch common bugs
use warnings;

=head1 Perl Tutorial
=cut

=head2 Print Program
=cut
print "What's your name?\n";        #Say and print are same, except say only works 5.010 version and above
my $name = <STDIN>;
chomp $name;                        #gets rid of new line on $name variable
print "Hola $name!\n";              #Say and print are same, except say only works 5.010 version and above

=head2 Number Sum Program
=cut
say "Enter a number";               #say doesn't need \n
my $num1 = <STDIN>;                 # 'my $name' is how scalar variables are initiated
say "Enter another number";
my $num2 = <STDIN>;
my $sum = $num1 + $num2;

say "Sum = $sum";
```

Data Structures in Perl

Scalar

Number or String

Arrays

signified by @ character

- `@arr_1 = (1, 2, 3);`
- `@str_arr = ("hello", "cruel", "world");`
- `@arr_42 = qw(the meaning of life doesn\'t exist);`

In `@arr_42`, qw (queue word) is a shortcut to write list of strings. You can use it to split strings by spaces into a list of words.

- `@arr_gen = (0 .. 5);` `# inclusive of both 0 and 5`
- `@arr_gen = (-1, @arr_gen, 6);` `# ⇒ 0,1..5,6`
- `($a, @arr_gen) = @arr_gen;` `# removes -1 from @arr_gen to $a`
- `$length = @arr_gen;` `# length is 7`
- `$#arr_gen` ⇒ returns the largest/last index of arr_gen

Accessing array elements

- `($first) = @arr_gen;` `# gets first element of @arr_gen`
- `$arr_gen[3] = 42;` `# changes first element to a number`
- `@arr_gen[0,1] = (10, 11);` `# changes first 2 elements of array`

Pushing, shifting, reversing and sorting elements within array

- `@arr = (1 .. 3);`
- `$new_rvalue = 4;`
- `$new_lvalue = 0;`
- `push(@arr, $new_rvalue);` `# @arr == (1, 2, 3, 4)`
- `unshift(@arr, $new_lvalue);` `# @arr == (0, 1, 2, 3, 4)`
- `$x = pop(@arr);` `# $x == 4, @arr == (0, 1, 2, 3)`
- `$y = shift(@arr);` `# $y == 0, @arr == (1, 2, 3)`
- `@rev_a = reverse(@a);`
- `@mylist = (1, 2, 4, 0, 32, 22, 17, 53, 42);`

- `@sorted = sort(@mylist);`
- *Sorting arrays in different orders:*
  https://perlmaven.com/sorting-arrays-in-perl
  https://perlmaven.com/sorting-mixed-strings

Hashes

declared with `%`.You cannot sort a hash and they are not in order.

Two ways of creating a key/value hash set

- `My %color_of;  #an empty hash`
- ```
  my %names = (
    'Moe', 'Howard',
    'Larry', 'Pop',
  );
  ```
- ```
  my %names = (
    'Moe' => 'Howard',
    'Larry' => 'Pop',
  );
  ```

Accessing individual hash values using keys

- `$names{'Larry'}`

Setting a new key/value pair

- `$names{"tom"} = "Jerry"`
- `$key = "tom";`
- `print "$names{$key}";    # Prints the value in the hash of tom`

Useful Hash functions

1. ***List all current keys***
   ```
   foreach $key (keys (%myhash)) {
       print "$myhash{$key}\n";
   }
   ```
2. ***Delete***
   ```
   delete $myhash{"key1"};
   # Deletes key1 and it's value
   ```
3. ***Values:*** returns a list of values in the hash → `keys(%myhash)`
4. ***Each:*** works much like keys does in (1), but returns list of both keys and hash values → `values(%myhash)`

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

use strict;
use warnings;
use 5.010;

# use Data::Dumper;

my @arr_1 = (1, 2, 3);
my @str_arr = ("hello", "cruel", "world");
my @arr_42 = qw(the meaning of life doesn\'t exist);
my @arr_gen = (0 .. 5);
@arr_gen = (-1, @arr_gen, 6);   # → 0,1..5,6
# ($a, @arr_gen) = @arr_gen;      # removes -1 from @arr_gen to $a
my $len = @arr_gen;             # length is 7
my ($first) = @arr_gen;      # gets first element of @arr_gen
$arr_gen[3] = 42;            # changes first element to a number
@arr_gen[0,1] = (10, 11);    # changes first 2 elements of array

my %names = (
    'hello', 'nevin',
    'how', 'are you',
);
print "$names{'hello'}\n";
print values(%names);
```

Queues in Perl

An array using push and shift functions can be used to implement a queue

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

my @people = ("Foo", "Bar");
while (@people) {
    my $next_person = shift @people;
    print "$next_person\n"; # do something with this person

    print "Type in the names of more people:";
    while (my $new = <STDIN>) {
        chomp $new;
        if ($new eq "") {
            last;
        }
        push @people, $new;
    }
    print "\n";
}
```

Errors, Variables, Conditions, and other stuff

Common Warnings and Error messages in Perl

https://perlmaven.com/common-warnings-and-error-messages

Variable Type Check:

looks_like_number($num) ⇒ returns true/false depending on $num is a number or not

Conditional Statements:

```
if ($age < 6) {
    print “you are not old enough\n”;
} elsif ($age < 15) {
    print “you are almost old enough\n”;
} else {
    print “you are old enough\n”;
}

if ($x eq “foo”) {
}
```

Ternary Operators
CONDITION ? ACTION_IF_TRUE : ACTION_IF_FALSE;

```
say $file ? $file : "file not given";
```

Boolean
The number **0**, the string '**0**' and '', the empty list "**()**", and "**undef**" are all false in boolean context. All other values are true.

Undef
is the same as NULL in Java or None in Python
To check if a variable is undefined you can use the defined() function

```
if (defined $x) {...}
```

Double and single quotes

- print "addr@gmail.com" ⇒ broken email
- print "addr\@gmail.com" ⇒ prints addr@gmail.com ⇒ correct
- Print 'hello $name' ⇒ hello $name
- $name = "foo" and $name = "foo" are both the **same**

Comparing scalars in Perl

Numeric	**String**	**Meaning**
==	eq	equal
!=	ne	Not equal
<	lt	Less than
>	gt	Greater than
<=	le	Less than or equal
>=	ge	Greater than or equal

Note:

- 12 > 3 is TRUE, but **12 gt 3 is FALSE** BECAUSE Perl compare string character by character and so "1" is less than "3"

- “Foo” == “bar” is True, “Foo” eq “bar” is False

String functions ($str = “Hello”)

- **lc(lower case):** *say lc $str;* #hello
- **uc(upper case):** *say uc $str;* #HELLO
- **length:** *say length $str;* #5
- **index**(finds beginning index of second string within first string):
  say index $str, ‘o’; #4
- **Substr:** *say substr $str, 3,4;* #lo

Random numbers

my $z = int rand 6 ; #returns whole number between 0 and 6

While loop

```
my $counter = 10;

while ($counter > 0) {
        say $counter;
        $counter -= 1;        #or $counter--;
}
```

For Loop

```
for (my $i = 0; $i <= 9; $i++) {
        print “$i\n”;
}

fore my $i  (0..9) {
        print “$i\n”;
}
```

```
for (INITIALIZE; TEST; STEP) {
  BODY;
}

for (INITIALIZE; TEST; STEP) {
  BODY;
}
```

```
my @names = ("Foo", "Bar", "Baz");
foreach my $n (@names) {
        Say $n;
}
```

Perl exit code

```
exit 0;
```

Error Message in Perl

```
print STDERR "Could not open file\n";
```

Warnings in Perl

```
warn "Something wrong\n";   #Something wrong.pl line 5
```

File editing

Writing to a file (>)

```
#!/usr/bin/perl

use strict;
use warnings;

my $filename = 'report.txt';
open(my $fh, '>', $filename) or die "Could not open file '$filename'!";
print $fh "My first report generated by perl\n";
close $fh;
```

Appending to a file (>>)

```
#!/usr/bin/perl

use strict;
use warnings;

my $filename = 'report.txt';
open(my $fh, '>>', $filename) or die "Could not open file '$filename'!";
print $fh "My first report generated by perl\n";
close $fh;
```

Reading a file (<)

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

my $filename = "report.txt";
open(my $fh, '<', $filename)
  or die "Could not open file '$filename' $!";

while (my $row = <$fh>) {
  chomp $row;
  print "$row\n";
}
print "done\n";
```

<$fh> IS the "readline" operator in Perl. It returns undef when there is no more to read from the file.

Repositioning filehandle after reading or writing: use **seek** mode

```
seek(my $filename, 10, 0);
```

```
          +-------------------------  0: Initially.
          |         +---------------- 10: After seek($fh, 10, SEEK_SET).
          |         |    +----------- 15: After reading "KLMNO".
          |         |    |    +------ 20: After seek($fh, 20, SEEK_SET).
          |         |    |    |
          v         v    v    v
file:     ABCDEFGHIJKLMNOPQRSTUVWXYZ
indexes: 01234567890123456789012345
```

If you wanted to seek relative to your current position, you'd use `SEEK_CUR`.

```
          +-------------------------  0: Initially.
          |         +---------------- 10: After seek($fh, 10, SEEK_CUR).
          |         |    +----------- 15: After reading "KLMNO".
          |         |    |         +- 25: After seek($fh, 10, SEEK_CUR).
          |         |    |         |
          v         v    v         v
file:     ABCDEFGHIJKLMNOPQRSTUVWXYZ
indexes: 01234567890123456789012345
```

CSV File parsing
https://perlmaven.com/how-to-read-a-csv-file-using-perl

Example of scanning for a particular field within a CSV File

```perl
use Text::CSV;    #A module that allows parsing, reading and writing

#format for parsing
my $csv = Text::CSV->new({sep_char => ','});

#looking for a file to parse
my $file = $ARGV[0] or die "not a file\n";

my $sum = 0;

#checks if the file can be opened or not
open(my $data, '<', $file) or die "cannot open $!\n";

#scan each line
while(my $line = <$data>) {
    chomp $line;

    #if the line can be parsed with the regex format..
    if($csv->parse($line)) {
        # add them to the fields array
        my @fields = $csv->fields();
        #add the first field from array to sum
        $sum += $fields[0];
    } else {
        warn "Line could not be parsed";
    }
}

print "$sum\n";
```

Note - If you want to use the module Text::CSV, you'll have to install the module using the following command
perl -MCPAN -e 'install Text::CSV'

Renaming, Copying and Removing files using Perl
https://perlmaven.com/how-to-remove-copy-or-rename-a-file-with-perl

Reading Excel file in Perl
https://perlmaven.com/read-an-excel-file-in-perl

Command Line and Perl

Command Line Arguments in Perl (@ARGV)
@ARGV is an array of command line arguments
$ARGV[0] is the name of the Perl script
$ARGV[1..] are the other arguments we'll use in the program

```
my ($name, $number) = @ARGV;
```

Processing Command Line Arguments

Suppose you run the command `test.pl --from foo` on the command line..

```
use Getopt::Long qw(GetOptions);
GetOptions('from=s' => \$source_address) or die "Usage: $0 --from NAME\n";
if ($source_address) {
   say $source_address;
}
```

This will store the value of from to $source_address. If no arguments are provided the program will end without printing anything. If an incorrect argument is given, program will die with an error message.

Command Line Advanced argument options
https://perlmaven.com/advanced-usage-of-getopt-long-accepting-command-line-arguments

Regex and Splitting Strings

Split function for Strings

- `Split REGEX,STRING` or `split REGEX,STRING, LIMIT` or `split`
- REGEX is usually in the format of `/.../` where … will be replaced by the element that will be used to split the string
- Example - Split a string concatenated with & and =.

```
use Data::Dumper qw(Dumper);

my $str = "DFE=FGFZ&DZFVSDFGSDF";

my @words = split /[=&]/, $str;
print Dumper \@words;
```

- If the REGEX is // then it will split every character in the string
- The REGEX for splitting a string using a pipeline (|) is `/\|/`.

Join function for Strings

```
my @names = ('Foo', 'Bar', 'Moo');
my $str = join ':', @names;
```

Regex Basics

- **Regex Cheat Sheet**
  https://perlmaven.com/regex-cheat-sheet

- **Regex Operators (=~)**

```
my $str = "this is a sentence to be tested";

if($str =~ /sentence/) {
    print "Found ya!\n";
}
```

- **Regex Negators (!~)**

```
my $str = "this is a sentence to be tested";

if($str !~ /sentence/) { #OR (not $str =~ /sentence/)
    print "Found ya!\n";
}
```

- **Handling special characters in Regex**
  These are the special characters: `.*+?^$\()[]|{}`

```
my $str = "this is a sen?ence to be tested";

if($str =~ /sen\?ence/) { #OR (not $str =~ /sentence/)
    print "Found ya!\n";
}
```

- **Special Character Classes**
  https://perlmaven.com/regex-special-character-classes

- **Regex Quantifiers**
  https://perlmaven.com/regex-quantifiers

- **Characters between two chars**

```
/a[bc]a/      # aba, aca
/a[2#=x?.]a/  # a2a, a#a, a=a, axa, a?a, a.a
              # inside the character class most of the spec character
s lose their
              # special meaning  BUT there are some new special chara
cters
/a[2-8]a/     # is the same as /a[2345678]a/
/a[2-]a/      # a2a, a-a        - has no special meaning at the ends
/a[-8]a/      # a8a, a-a
/a[6-C]a/     # a6a, a7a ... aCa
              #      characters from the ASCII table: 6789:;<=>?@ABC
but this is not recommended, don't use it!
/a[C-6]a/     # syntax error

/a[^xy]a/     # "aba", "aca"  but not "aya", "axa" and remember, not
"aa"
              # ^ as the first character in a character class means
              # a character that is not in the list
/a[b^x]a/     # aba, a^a, axa,  but not aza
```

Subroutines

Subroutines - Functions of Perl
Subroutines are started with *sub* and then the name of the subroutine. The parameters of the subroutine will be stored in @_. It is recommended for all subroutines to have a return statement at the end (even if you are not returning anything.)

```
foreach my $v (@_) {
    $sum += $v;
}
```

```
my $first_name = prompt("First Name: ");
my $last_name = prompt("Lat Name: ");

print "Hello $first_name $last_name!\n";

sub prompt {
    my @text = @_;  #OR => my ($text) = @_
    print @text;

    my $answer = <STDIN>;
    chomp $answer;
    return $answer;
}
```

Time and Unary Operators

Time
https://perlmaven.com/the-year-19100
Count Frequency of word in Text
https://perlmaven.com/count-words-in-text-using-perl

Unary Operators in perl and their definitions

Most of the unary operators return true or false

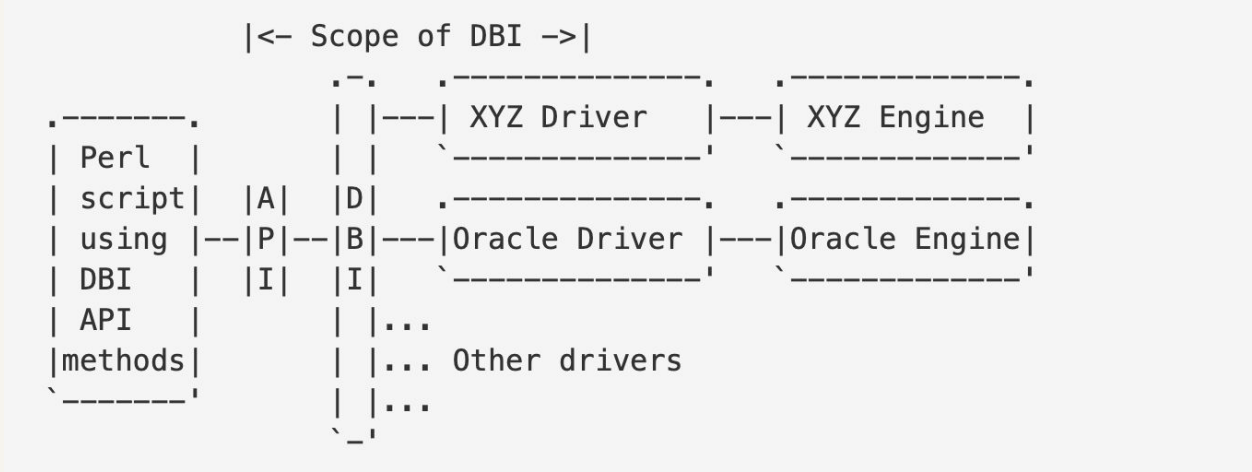
- -r: File is readable by effective uid/gid.
- -w: File is writable by effective uid/gid.
- -x: File is executable by effective uid/gid.
- -o: File is owned by effective uid.
- -R: File is readable by real uid/gid.
- -W: File is writable by real uid/gid.
- -X: File is executable by real uid/gid.
- -O: File is owned by real uid.
- -e: File exists.
- -z: File has zero size (is empty).
- -s: File has nonzero size (returns size in bytes).
- -f: File is a plain file.
- -d: File is a directory.
- -l: File is a symbolic link (false if symlinks aren't supported by the file system).
- -p: File is a named pipe (FIFO), or Filehandle is a pipe.
- -S: File is a socket.
- -b: File is a block special file.
- -c: File is a character special file.
- -t: Filehandle is opened to a tty.
- -u: File has setuid bit set.
- -g: File has setgid bit set.
- -k: File has sticky bit set.
- -T: File is an ASCII or UTF-8 text file (heuristic guess).
- -B: File is a "binary" file (opposite of -T).
- -M: Script start time minus file modification time, in days.
- -A: Same for access time.
- -C: Same for inode change time (Unix, may differ for other platforms)

Accessing Database using Perl DBI and SQL

https://perlmaven.com/simple-database-access-using-perl-dbi-and-sql

DBI - Database independent Interface

Architecture



API defines the variables for Perl script to use. The API is implemented by Perl DBI extension. DBI dispatches method calls to appropriate driver for execution, handle errors and do other stuff.

Notations

`$dbh`	Database handle object
`$dsn`	Data Source Name (contains the type of database)
`$sth`	Statement handle object
`$drh`	Driver handle object (rarely seen or used in applications)
`$h`	Any of the handle types above ($dbh, $sth, or $drh)
`$rc`	General Return Code (boolean: true=ok, false=error)
`$rv`	General Return Value (typically an integer)
`@ary`	List of values returned from database, typically a row of data
`$rows`	Number of rows processed (if available, else -1)
`$fh`	A filehandle
`undef`	NULL values are represented by undefined values in Perl
`\%attr`	Reference to a hash of attribute values passed to methods

SQL Commands and Syntaxes

The complete Tutorial
https://www.codecademy.com/articles/sql-commands

SELECT
Used to query or retrieve data from various columns of a table in the database.
SELECT column_list FROM table_name WHERE

- *column_name:* You can SELECT a single column, a list of columns (separated by commas) or all of the columns (using *)
- *table_name:* This is the table from which the data is SELECTed.
- *WHERE (optional):* used to select a particular row. You can filter the row using conditionals such as:
  - = equal
  - > greater than
  - < less than
  - >= greater or equal to
  - <= less or equal to
  - <> NOT equal to
  - LIKE select rows that starts/ends with a special char(s)
- ORDER BY: used to order the columns based on

Examples:
SELECT first, last, city FROM profiles WHERE age > 30
SELECT first FROM profiles WHERE first = 'Eric';
*SELECT * FROM profiles WHERE last LIKE '%s'* *#LIKE → ends with 's'*
SELECT city FROM profiles WHERE first LIKE '%Er'; *#LIKE → starts with 'Er'*
SELECT first FROM profiles WHERE last '%John%';

SELECT DISTINCT
Specifies that SELECT statement is going to be a query that returns unique values in the specified columns(s).

AND
Used to combine two conditions

```
...
WHERE column_1 = value_1
  AND column_2 = value_2;
```

AS
Allows you to rename a column or table until the end of the query

```
SELECT column_name AS 'Alias'
```

```
...
```

BETWEEN
Used to filter result within range of numbers, text or range

```
...
WHERE column_name BETWEEN value_1 AND value_2;
```

CASE (Pretty much IF-THEN logic)
Used to create different outputs (usually in the SELECT statement)

```
...
CASE
    WHEN condition THEN 'Result_1'
    WHEN condition THEN 'Result_2'
    ELSE 'Result_3'
  END
```

COUNT()
Takes the name of the column as an argument counts number of rows where column is not NULL.

```
SELECT COUNT(column_name) ...
```

GROUP BY
Used in collaboration with SELECT to arrange identical data into groups.

```
...
GROUP BY column_name;
```

INNER JOIN
Combines two rows from different tables if the condition is true

```
...
JOIN table_2
  ON table_1.column_name = table_2.column_name;
```

OUTER JOIN
Joins tables even if condition is not true. If condition is not met, NULL values are used to fill the columns.

```
...
LEFT JOIN table_2
  ON table_1.column_name = table_2.column_name;
```

IS NULL / IS NOT NULL
Used by WHERE clause to test for empty values

```
SELECT column_name(s)
FROM table_name
WHERE column_name IS NULL;
```

LIKE
Used with WHERE clause to search for a specific pattern in a column.
```
...
WHERE column_name LIKE pattern;
```

LIMIT
Used to specify maximum number of rows.

MAX() / MIN()
A function that takes the name of a column and return largest / smallest value in that column.
```
SELECT MAX(column_name)
FROM table_name;
```

OR
Operator that filters results set to include either conditions that are true
```
...
WHERE column_name = value_1
   OR column_name = value_2;
```

ORDER BY
ASC / DESC

ROUND()
Takes a column name and integer; It rounds the value in the column to the integer value specified
```
SELECT ROUND(column_name, integer)
...
```

SUM()
Returns the sum of all values in a column

WHERE
Filters results to include rows where condition is true
```
...
WHERE column_name operator value;
```

WITH

Lets you store the results of a query in a temporary table using an alias.

```
WITH temporary_name AS (
   SELECT *
   FROM table_name)
SELECT *
FROM temporary_name
WHERE column_name operator value;
```