# PERL

1. What is PERL ?

* PERL – Practical Extraction and Reporting Language.
* It is a high-level scripting language supporting powerful regular expressions.
* It is open source and free

1. Why use PERL ?

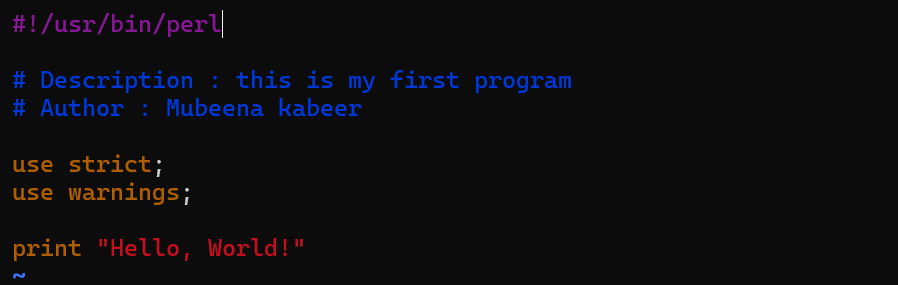
* High-level interpreted language with faster development time.
* Well suited to perform automation/ manual recurring tasks
* Use : in Semiconductor design
* Easy creation of `multiple applications using large number of inbuilt modules like SQL database, sending emails, Excel sheets, etc
* Large user base

1. How to install PERL?

* Go to the side <https://www.perl.org/>
* Then click on download to get started
* Download strawberry perl for windows or your preferred OS
* Then follow the instructions to download perl
* We can check it by “perl -v” in command prompt

1. Write your first program.

I use VIM to write the commands and created a file using : vim HelloWorld.pl



* Here first line is the shebang line that tells. It tells the system where to find the Perl interpreter.
* use strict : Helps catch errors by enforcing strict syntax rules in Perl.
* use warnings : Alerts you about potential issues in your code, making it easier to spot mistakes.

A black background with white text

Description automatically generated

1. What are file permissions?

* It tells us who can perform particular operation on specific files
* We run the command : ls -l to get the list of files and folders with their permissionA black background with white text

  Description automatically generated  
  A screenshot of a computer program

  Description automatically generated
* It is recommended for file owners to have all permission and all other users does not have execute permission.
* Example to make rwx--r-- is a preferred format = it also means 744 (octal format)A screenshot of a game

  Description automatically generated
* While giving execute command, give read command also. Then only they will be able to read and execute the script.

1. Comments in perl?

There are two types of commenting:  
A black screen with white text

Description automatically generated

1. Datatypes in perl?

Perl supports different datatypes

1. Scalars  
   Scalars are single values, such as numbers, strings, or floating-point numbers.

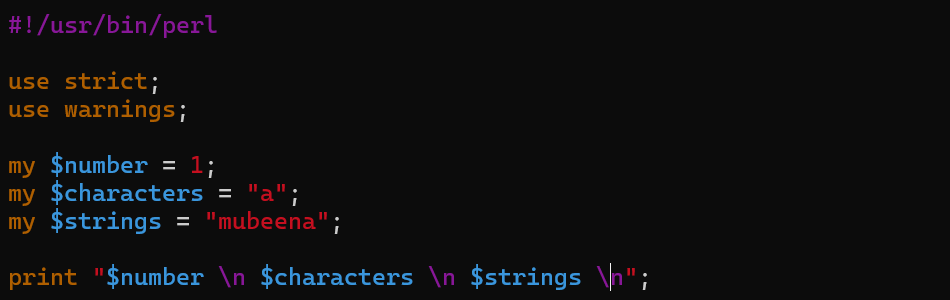
They are preceded by the dollar sign $ when declaring variables.

Examples:

my $number = 42; # Integer scalar

my $name = "John"; # String scalar

my $pi = 3.14; # Floating-point scalar

input   
  
output   
A black screen with white text

Description automatically generated

1. Arrays :   
   Scalars are single values, such as numbers, strings, or floating-point numbers.

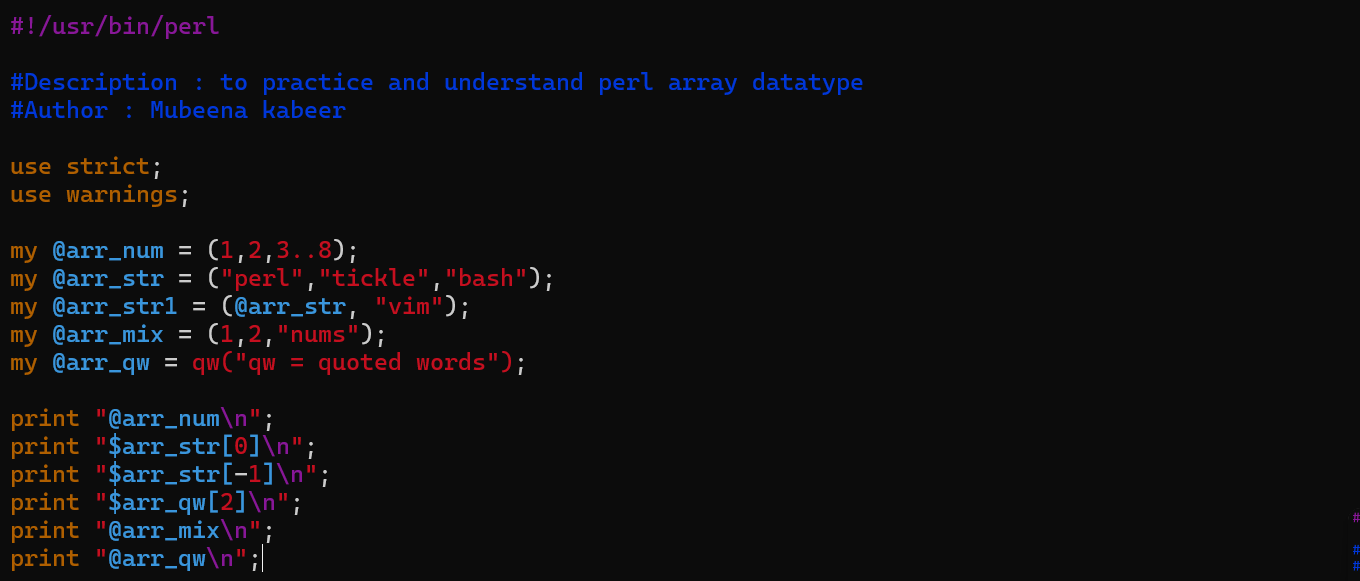
They are preceded by the dollar sign $ when declaring variables.

Examples:

my @numbers = (1, 2, 3, 4, 5); # Array of numbers

my @names = ("Alice", "Bob", "Charlie"); # Array of strings

input



Output

A screen shot of a computer

Description automatically generated

Some important actions:

A screenshot of a computer program

Description automatically generated

A black screen with white text

Description automatically generated

1. Hashes :   
   Unordered collections of key-value pairs. Unlike arrays, elements in hashes are accessed using unique keys (which can be strings or numbers) instead of numerical indexes. Hashes are denoted by the percent sign (%).

Examples:

my %ages = ( # Hash of ages

"Alice" => 30,

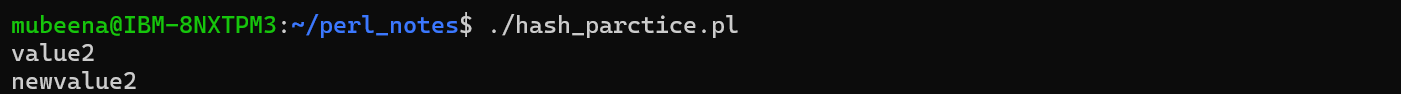
"Bob" => 25,

"Charlie" => 35

);

A computer screen with red and blue text

Description automatically generated



1. Operators in PERL?

Perl offers a rich set of operators to perform various operations on data. Here's a breakdown of the common operators in Perl:

1. Arithmetic operators:

my $x = 10;

my $y = 5;

my $sum = $x + $y; # Addition

my $difference = $x - $y; # Subtraction

my $product = $x \* $y; # Multiplication

my $quotient = $x / $y; # Division

my $remainder = $x % $y; # Modulus

my $power = $x \*\* $y; # Exponentiation

1. Comparison Operators:

my $a = 10;

my $b = 20;

my $equal = ($a == $b); # Equality

my $not\_equal = ($a != $b); # Inequality

my $less\_than = ($a < $b); # Less than

my $greater\_than = ($a > $b); # Greater than

my $less\_or\_equal = ($a <= $b); # Less than or equal to

my $greater\_or\_equal = ($a >= $b); # Greater than or equal to

1. Logical Operators:

my $foo = 1;

my $bar = 0;

my $result\_and = ($foo && $bar); # Logical AND

my $result\_or = ($foo || $bar); # Logical OR

my $result\_not = !($foo); # Logical NOT

1. String Concatenation Operator:

my $string1 = "Hello";

my $string2 = "World";

my $concatenated = $string1 . ", " . $string2; # Concatenation

1. Assignment operator:

my $num = 10;

$num += 5; # $num now equals 15 (compound addition)

$num -= 3; # $num now equals 12 (compound subtraction)

$num \*= 2; # $num now equals 24 (compound multiplication)

$num /= 4; # $num now equals 6 (compound division)

$num %= 5; # $num now equals 1 (compound modulus)

1. Increment operator:

my $count = 5;

$count++; # Increment $count by 1

$count--; # Decrement $count by 1

1. Ternary operator:

my $condition = 1;

my $result = $condition ? "true" : "false"; # Ternary operator

1. Range operator:

my @numbers = (1..10); # Inclusive range: 1, 2, 3, ..., 10

my @letters = ('a'...'z'); # Exclusive range: 'a', 'b', 'c', ..., 'y'

1. Repeat operator:

my $word = "Perl";

my $repeated\_word = $word x 3; # Repeats "Perl" three times: "PerlPerlPerl"

1. Hash Key-Value Separator (=>):

my %hash = (

key1 => "value1",

key2 => "value2",

key3 => "value3"

);

1. Quote Words Operator (qw):

my @words = qw(apple banana orange);

1. Quote Regex Operator (qr):

my $regex = qr/\d{3}-\d{3}-\d{4}/; # Matches a phone number pattern

1. Conditional Assignment Operators (&&= and ||=):

my $value1 = 5;

my $value2;

$value1 &&= 10; # $value1 remains 10 if it was initially true

$value2 ||= 20; # $value2 becomes 20 if it was initially false or undefined

1. How to do sorting in perl ?

A black screen with white text

Description automatically generated

A computer screen with blue and white text

Description automatically generated

1. Loops in perl
2. For loop  
   A screen shot of a computer

   Description automatically generatedA computer screen with text on it

   Description automatically generated
3. Foreach loop

A screen shot of a computer code

Description automatically generated A black screen with white text

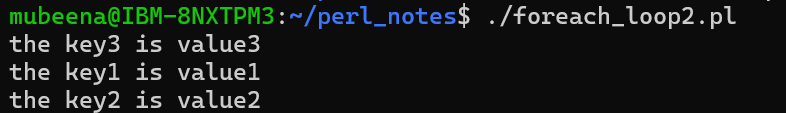
Description automatically generated

A black screen with white text and blue letters

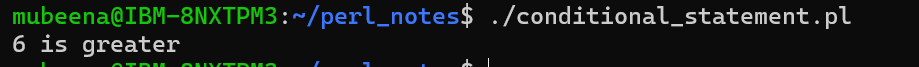
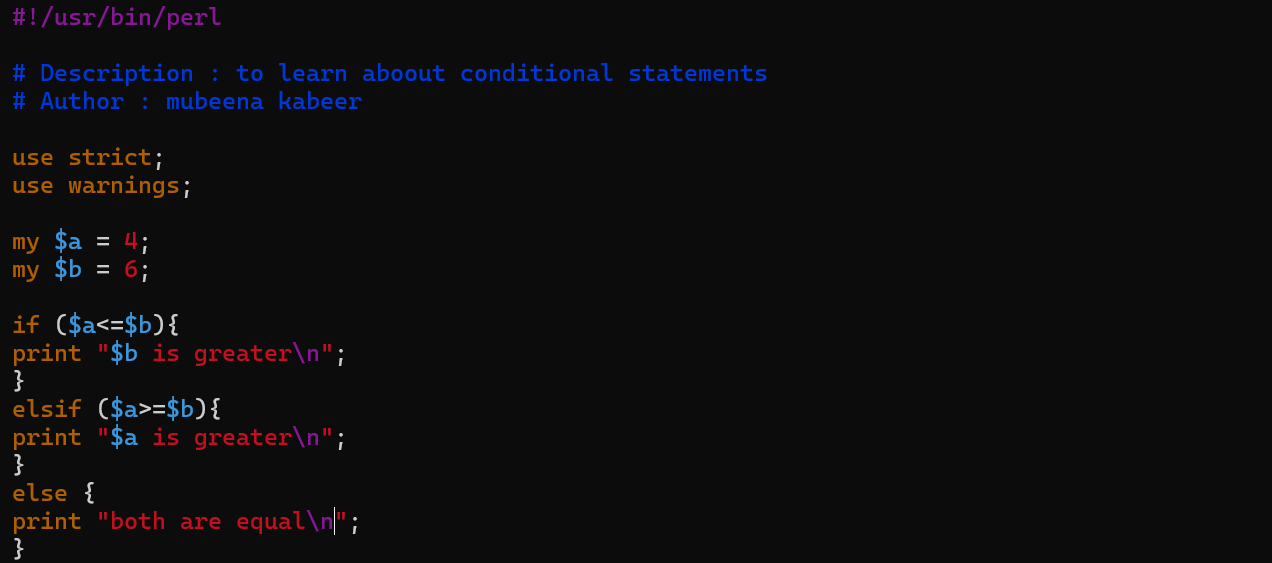
Description automatically generated

A screenshot of a computer code

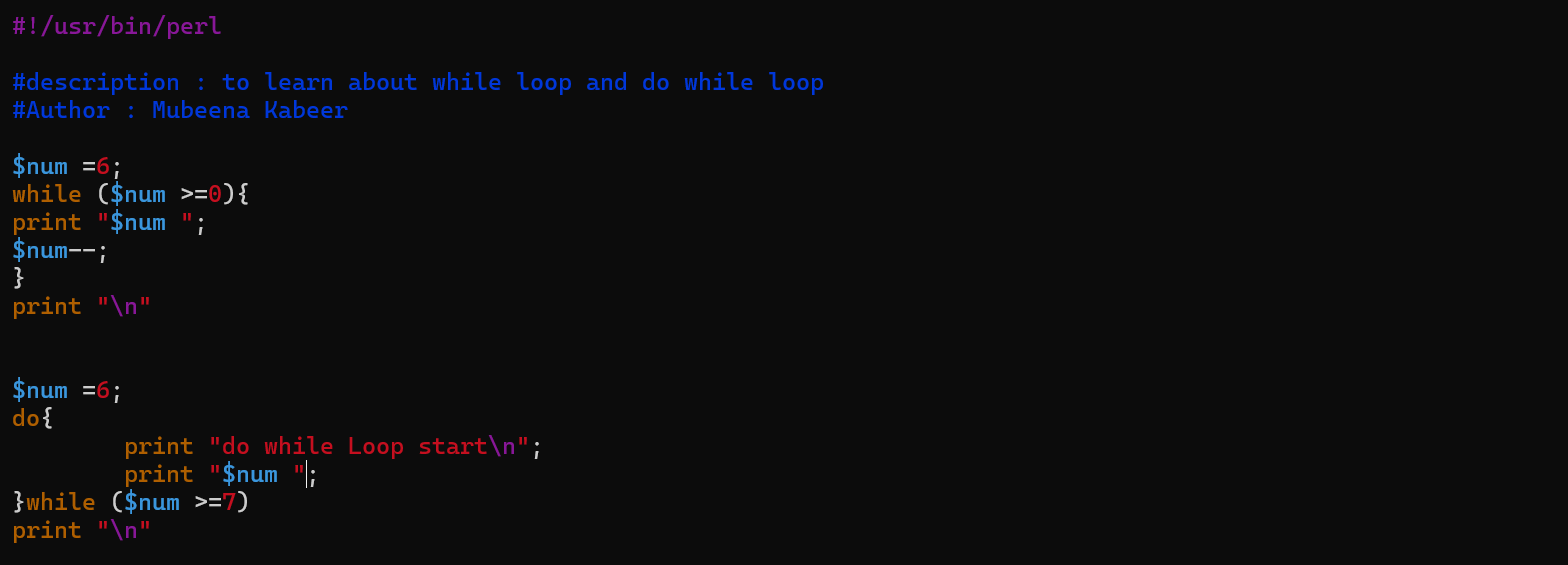
Description automatically generated



1. If,elsif,else

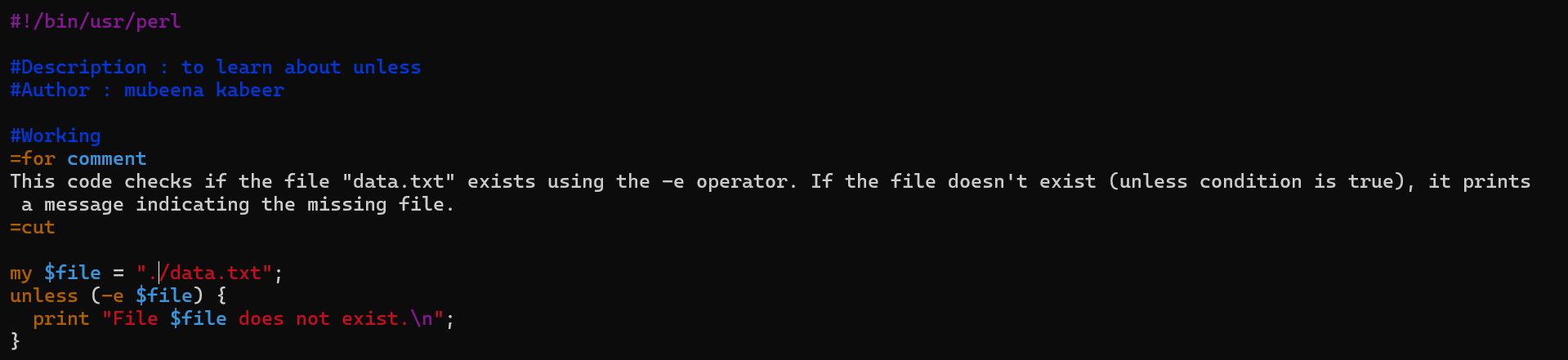


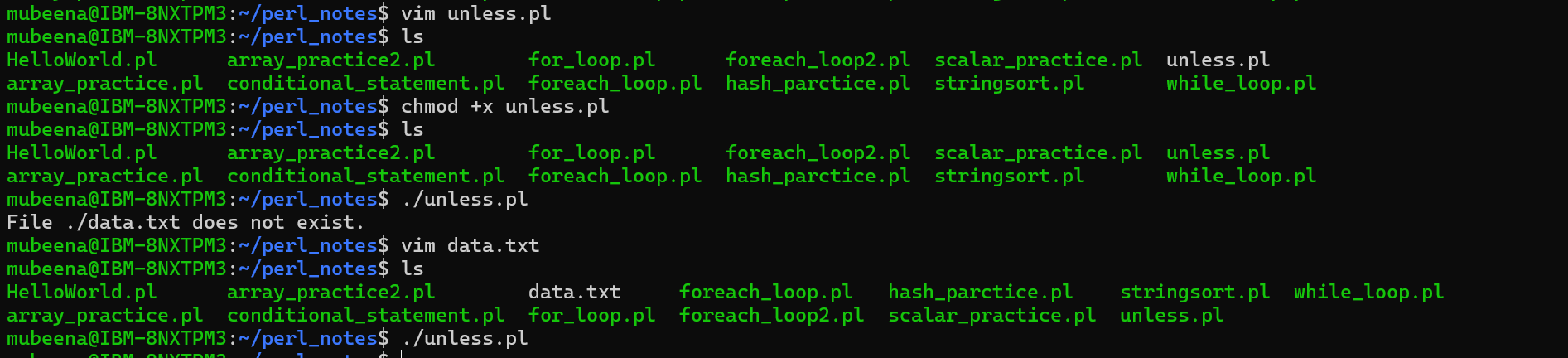
1. While, do while

A black screen with blue text

Description automatically generated

1. Unless





1. What are subroutines in perl?

Subroutines, also known as functions or methods in Perl, are reusable blocks of code that perform a specific task. They promote code modularity, readability, and maintainability.

SYNTAX

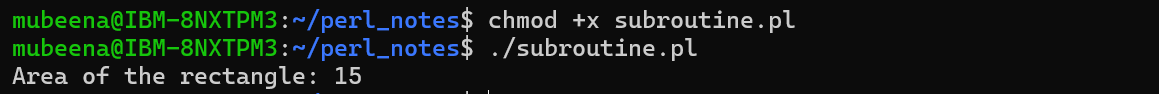
sub subroutine\_name (argument1, argument2, ...) {

# Code block to be executed

return value; # Optional, return a value

}

A screenshot of a computer program

Description automatically generated

1. Regular expression in PERL?

Regular expressions (regex or regexp), also known as patterns, are powerful tools in Perl for matching, searching, and manipulating text. They define a specific pattern of characters that you want to find or work with in a string

**Metacharacters (searching)**: These are special characters that have a specific meaning within a regex pattern. Some common examples include:

* . (dot): Matches any single character except a newline (unless the s flag is used).
* ^: Matches the beginning of the string (or line, if the m flag is used).
* $: Matches the end of the string (or line, if the m flag is used).
* []: Character class - matches any character within the brackets. (e.g., [abc] matches 'a', 'b', or 'c').
* \*: Matches the preceding element zero or more times.
* +: Matches the preceding element one or more times.
* ?: Matches the preceding element zero or one time.
* |: Or operator - matches either the pattern before or after the pipe.
* \: Escapes the following character, treating it literally. (e.g., \$ matches the literal '$' symbol).
* -: indicate range of characters
* &:substitute complete match
* ():grouping characters
* {}: range quantifiers
* <>:anchors that specify left or right word boundary

**Quantifiers(how often)**: These specify how many times a preceding element can be matched. Common quantifiers include:

* \*: Zero or more times (already mentioned in metacharacters).
* +: One or more times (already mentioned in metacharacters).
* ?: Zero or one time (already mentioned in metacharacters).
* {n}: Exactly n times (e.g., ab{3}c matches "abbbc").
* (n,}: match atleast n times
* {n,m}: At least n but no more than m times (e.g., fo{2,4}d matches "food", "foood", or "fooood").

**Anchors(where)**: These metacharacters match specific positions within the string. Common anchors include:

* ^: Matches the beginning of the string.
* $: Matches the end of the string.
* \b: Matches a word boundary (e.g., between a word and whitespace or punctuation).
* \B: no boundary between a word and a non word
* <match the regular expression at the start of the word
* > match the regular expression at the end of the word

**Flags (Modifiers)**: These are optional characters added at the end of the regex pattern to alter its behavior. Some commonly used flags include:

* i: Case-insensitive matching.
* m: Multiline mode (. matches newline characters).
* s: Single-line mode (. matches any character, including newline).
* x: Allows whitespace characters within the pattern for readability (ignored in the match).

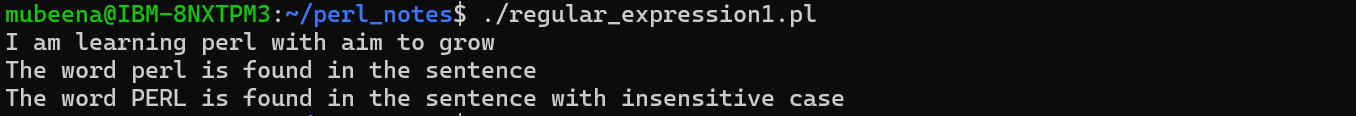
Perl offers several operators to work with regular expressions:

* ~ (match operator): Checks if a string matches the regex pattern (returns true or false).
* =~ m/.../ (match operator with variable): Similar to ~, but stores the matched substring in a variable (m for match).
* s/// (substitution operator): Replaces all matches of the pattern in a string with a replacement string.

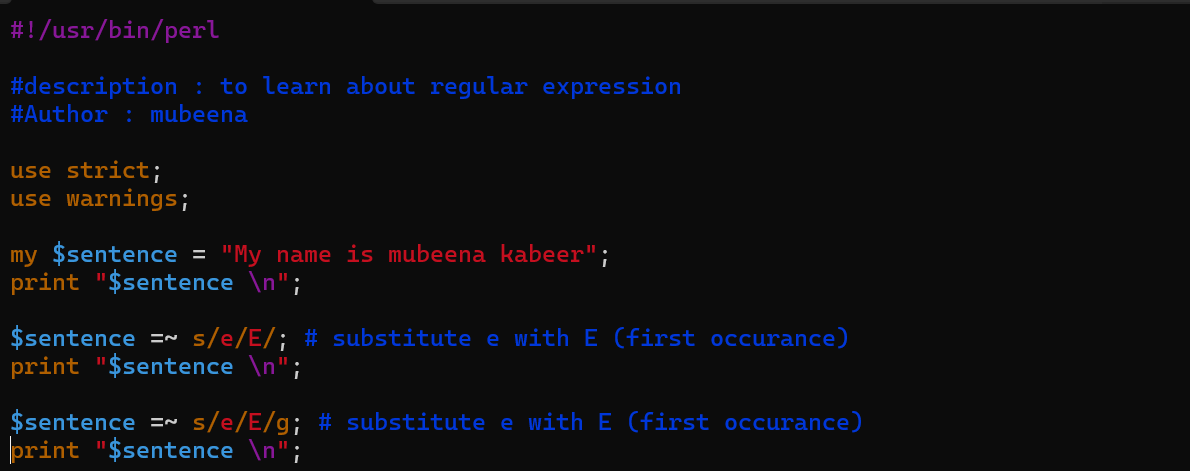
Example 1:

A screenshot of a computer screen

Description automatically generated



Example 2:



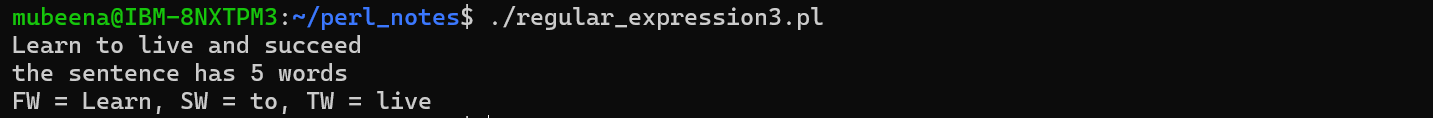
A black screen with white text

Description automatically generated

Example 3:

A screenshot of a computer

Description automatically generated



Example 4:

A screenshot of a computer program

Description automatically generated

A black screen with white text

Description automatically generated

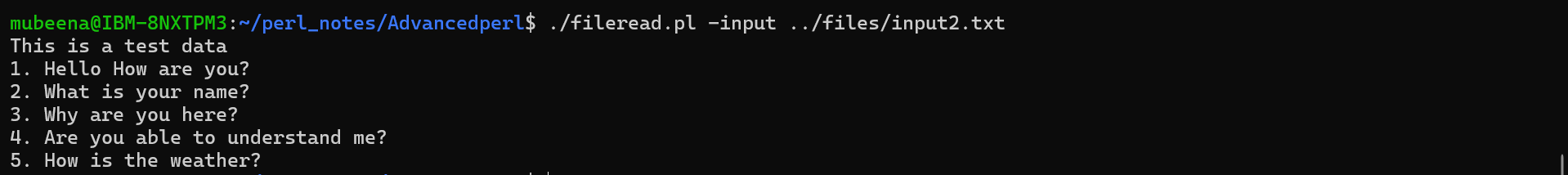
1. File Read and write operations
2. Introductions

* File is opened in the below formats
  + < : Read only format
  + > : Write only format
  + >> : File appending format
  + +< : Read and Write format
* To open and close
  + Open : to open the file
  + Close : to close the file
* To give additional information
  + Die : signals fatal error and terminates the program
  + Warn : Issues a warning message, but allows the program to continue execution.
* Data flow streams
  + STDIN
  + STDOUT
  + STDERR
* File checks
  + -e : if file exists
  + -r : if file is readable
  + -w : if file is writable
  + -x : if file is executable
  + -l : if symbolic link is present
  + -d : if it is a directory

1. File - read example

A computer screen shot of text

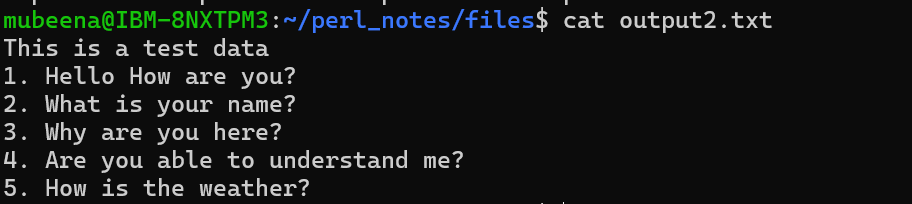
Description automatically generated



1. File write example

A computer screen shot of text

Description automatically generated



Change > to “>>” to append instead of overwrite the file contents.