INFORME

INFORMACIÓN BÁSICA					
ASIGNATURA:	**PROGRAMACIÓN WEB 1**				
TÍTULO DE LA PRÁCTICA:	*Tarea Extra de Perl*				
NÚMERO DE PRÁCTICA:	*03*	**AÑO LECTIVO:**	*2024 - B*	**NRO. SEMESTRE:**	*II*
FECHA DE PRESENTACIÓN	*18/10/2024*	**HORA DE PRESENTACIÓN**	*15:00:00 PM*		
INTEGRANTE (s): *German Arturo Chipana Jerónimo*				**NOTA:**	
DOCENTE: *CORRALES DELGADO, CARLO JOSE LUIS*					

SOLUCIÓN Y RESULTADOS

I. SOLUCIÓN DE EJERCICIOS

EJERCICIO 01:

Escriba un programa perl que pida el nombre de usuario y luego el dominio, luego el programa deberá imprimir el correo electrónico juntando el nombre de usuario y el dominio y usando el caracter @ en medio. Como restricción no podrá usar el operador punto(.) para concatenar strings. Si el usuario ingresa como nombre "alumno" y como dominio "pweb1", deberá imprimir: "alumno@pweb1".

Creación de programa1.pl

programa1.pl

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

# Pide el nombre de usuario
print "Ingrese el nombre de usuario: ";
my $usuario = <STDIN>;
chomp($usuario);  # Elimina el salto de línea

# Pide el dominio
print "Ingrese el dominio: ";
my $dominio = <STDIN>;
chomp($dominio);  # Elimina el salto de línea

# Imprime el correo utilizando interpolación
print "$usuario\@$dominio\n";
```

Este script en Perl solicita al usuario ingresar un nombre de usuario y un dominio por separado, y luego imprime una dirección de correo electrónico concatenando ambos, usando el símbolo @ entre el nombre y el dominio.

Descripción paso a paso:

1. **Uso de use strict y use warnings**:

- Estas directivas mejoran la seguridad y la depuración del código al forzar que se declaren las variables antes de usarlas (use strict) y al mostrar advertencias sobre posibles problemas (use warnings).

2. **Solicitud del nombre de usuario**:

print "Ingrese el nombre de usuario: ";
my $usuario = <STDIN>;
chomp($usuario); # Elimina el salto de línea

- Se imprime un mensaje pidiendo al usuario que ingrese un nombre de usuario.
- El nombre de usuario es leído desde la entrada estándar (<STDIN>), y se almacena en la variable $usuario.
- La función chomp elimina el salto de línea que normalmente se añade al final de una entrada del usuario.

3. **Solicitud del dominio**:

   print "Ingrese el dominio: ";
   my $dominio = <STDIN>;
   chomp($dominio); # Elimina el salto de línea

- Similar al paso anterior, se pide al usuario que ingrese un dominio, y la entrada se almacena en la variable $dominio, usando chomp para eliminar el salto de línea.

4. **Impresión del correo electrónico**:

   print "$usuario\@$dominio\n";

- Utilizando interpolación de cadenas en Perl, se genera la dirección de correo electrónico concatenando las variables $usuario y $dominio, con el símbolo @ en medio.
- El resultado es impreso en pantalla.

Restricción:

El ejercicio requiere no usar el operador punto (.) para concatenar las cadenas. En lugar de esto, utiliza interpolación de cadenas, que es una forma eficiente y legible de combinar variables y texto en Perl.

Este enfoque es simple, directo y evita el uso de concatenación explícita.

Ejecutando

```
user@DESKTOP-CS7PIHP MINGW64 ~/TAREA-EXTRA-PERL (main)
$ perl programa1.pl
Ingrese el nombre de usuario: germanchipana
Ingrese el dominio: unsa
germanchipana@unsa
```

Commits

Primer Commit

```
MINGW64:/c/Users/user/TAREA-EXTRA-PERL

user@DESKTOP-CS7PIHP MINGW64 ~ (master)
$ cd TAREA-EXTRA-PERL

user@DESKTOP-CS7PIHP MINGW64 ~/TAREA-EXTRA-PERL (master)
$ git init
Initialized empty Git repository in C:/Users/user/TAREA-EXTRA-PERL/.git/

user@DESKTOP-CS7PIHP MINGW64 ~/TAREA-EXTRA-PERL (master)
$ git add programa1.pl

user@DESKTOP-CS7PIHP MINGW64 ~/TAREA-EXTRA-PERL (master)
$ git commit -m "Primera version Programa 01"
[master (root-commit) e090ca5] Primera version Programa 01
 1 file changed, 10 insertions(+)
 create mode 100644 programa1.pl

user@DESKTOP-CS7PIHP MINGW64 ~/TAREA-EXTRA-PERL (master)
$ git branch -M main

user@DESKTOP-CS7PIHP MINGW64 ~/TAREA-EXTRA-PERL (main)
$ git remote add origin https://github.com/ChipanaGerman/TAREA_EXTRA_PERL.git

user@DESKTOP-CS7PIHP MINGW64 ~/TAREA-EXTRA-PERL (main)
$ git push -u origin main
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Delta compression using up to 8 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 374 bytes | 374.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To https://github.com/ChipanaGerman/TAREA_EXTRA_PERL.git
 * [new branch]      main -> main
branch 'main' set up to track 'origin/main'.
```

En este primer commit se inicializa el repositorio de Git, luego se añade el primer programa *programa1.pl* y se realiza el commit *"Primera versión Programa 01"* que tiene la mitad del ejercicio realizado, finalmente se sube a la rama main.

Segundo Commit

```
user@DESKTOP-CS7PIHP MINGW64 ~/TAREA-EXTRA-PERL (main)
$ git add programa1.pl

user@DESKTOP-CS7PIHP MINGW64 ~/TAREA-EXTRA-PERL (main)
$ git commit -m "Version final Programa 01"
[main c57cb8f] Version final Programa 01
 1 file changed, 9 insertions(+)

user@DESKTOP-CS7PIHP MINGW64 ~/TAREA-EXTRA-PERL (main)
$ git push origin main
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 8 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 469 bytes | 469.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To https://github.com/ChipanaGerman/TAREA_EXTRA_PERL.git
   e090ca5..c57cb8f  main -> main
```

En este segundo commit, se actualiza el archivo *programa1.pl* con la versión finalizada de nuestro ejercicio, se usa el commit *"Version final Programa 01"* y se sube a la rama main.

EJERCICIO 02

Estamos al final del semestre y para calcular su promedio el profesor del curso desea eliminar la peor nota y duplicar la mayor nota. De este modo, si sus notas fueran 12, 15, 17 y 14; el profesor eliminaría el 12 y duplicaría el 17, entonces sus notas serían 17, 15, 17 y 14 y sobre ellas calcularía el promedio. Usted deberá programar una función que reciba las notas y devuelva el promedio, según se explicó y para una cantidad de notas no determinada. Para programar su función no podrá usar ningún tipo de condicionales, sólo las funciones max y min de perl.

Creación de programa2.pl

programa1.pl | programa2.pl

```perl
#!/usr/bin/perl
use strict;
use warnings;
use List::Util qw(min max sum);

sub calcular_promedio {
    my @notas = @_;

    # Encuentra la nota mínima y máxima
    my $nota_min = min @notas;
    my $nota_max = max @notas;

    # Filtra las notas para eliminar la mínima y duplicar la máxima
    @notas = grep { $_ != $nota_min } @notas;  # Elimina la mínima
    push @notas, $nota_max;  # Duplica la máxima

    # Calcula el promedio
    my $promedio = sum(@notas) / @notas;

    return $promedio;
}

# Solicitar las notas al usuario
print "Ingrese las notas separadas por espacio: ";
my $input = <STDIN>;  # Leer la entrada del usuario
chomp($input);  # Eliminar el salto de línea al final

# Convertir la entrada en un arreglo de números
my @notas = split(' ', $input);

# Verificar que el usuario haya ingresado al menos dos notas
if (scalar @notas < 2) {
    print "Debe ingresar al menos dos notas.\n";
    exit;
}

# Calcular el promedio
my $promedio = calcular_promedio(@notas);
print "El promedio es: $promedio\n";

```

Este script en Perl permite al usuario ingresar un conjunto de notas y luego calcula el promedio de dichas notas, siguiendo una regla especial: elimina la nota más baja y duplica la más alta. El programa está estructurado de la siguiente manera:

Descripción paso a paso:

1. **Uso de módulos y directivas**:

```
use strict;
use warnings;
use List::Util qw(min max sum);
```

- use strict y use warnings ayudan a mejorar la seguridad del código y mostrar advertencias sobre posibles errores.
- List::Util importa las funciones min, max, y sum para trabajar con arrays.

2. **Función calcular_promedio**:

```
sub calcular_promedio {
    my @notas = @_;

    # Encuentra la nota mínima y máxima
    my $nota_min = min @notas;
    my $nota_max = max @notas;

    # Filtra las notas para eliminar la mínima y duplicar la máxima
    @notas = grep { $_ != $nota_min } @notas;  # Elimina la mínima
    push @notas, $nota_max;  # Duplica la máxima

    # Calcula el promedio
    my $promedio = sum(@notas) / @notas;

    return $promedio;
}
```

- La función toma un array de notas y encuentra la nota más baja y la más alta usando las funciones min y max.
- Luego, elimina la nota mínima con grep y duplica la nota máxima añadiéndola al array nuevamente con push.
- Finalmente, calcula el promedio sumando las notas con sum y dividiéndolo por la cantidad total de notas.

3. **Solicitar notas al usuario**:

```
print "Ingrese las notas separadas por espacio: ";
my $input = <STDIN>;
chomp($input);  # Eliminar el salto de línea al final
my @notas = split(' ', $input);
```

- Se pide al usuario que ingrese las notas separadas por espacios.
- La entrada del usuario es procesada con split, lo que convierte la cadena en un array de notas.

4. **Verificación de la cantidad de notas**:

```
if (scalar @notas < 2) {
   print "Debe ingresar al menos dos notas.\n";
   exit;
}
```

- El programa verifica que el usuario haya ingresado al menos dos notas. Si no lo ha hecho, muestra un mensaje y finaliza.

5. **Cálculo del promedio**:

```
my $promedio = calcular_promedio(@notas);
print "El promedio es: $promedio\n";
```

- Si las notas ingresadas son válidas, el promedio se calcula llamando a la función calcular_promedio y el resultado se muestra en la consola.

Ejecutando

```
user@DESKTOP-CS7PIHP MINGW64 ~/TAREA-EXTRA-PERL (main)
$ perl programa2.pl
Ingrese las notas separadas por espacio: 20
Debe ingresar al menos dos notas.
```

```
user@DESKTOP-CS7PIHP MINGW64 ~/TAREA-EXTRA-PERL (main)
$ perl programa2.pl
Ingrese las notas separadas por espacio: 0 14 15 16
El promedio es: 15.25
```

Commits

Primer Commit

```
user@DESKTOP-CS7PIHP MINGW64 ~/TAREA-EXTRA-PERL (main)
$ git add programa2.pl

user@DESKTOP-CS7PIHP MINGW64 ~/TAREA-EXTRA-PERL (main)
$ git commit -m "Primera version Programa 02"
[main fa6cf49] Primera version Programa 02
 1 file changed, 23 insertions(+)
 create mode 100644 programa2.pl

user@DESKTOP-CS7PIHP MINGW64 ~/TAREA-EXTRA-PERL (main)
$ git push origin main
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 8 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 557 bytes | 557.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To https://github.com/ChipanaGerman/TAREA_EXTRA_PERL.git
   c57cb8f..fa6cf49  main -> main

user@DESKTOP-CS7PIHP MINGW64 ~/TAREA-EXTRA-PERL (main)
```

En este primer commit, se agrego el archivo *programa2.pl* el cual estaba realizado a la mitad, se usó el commit *"Primera version Programa 02"* y se subió a la rama main.

Segundo Commit

```
user@DESKTOP-CS7PIHP MINGW64 ~/TAREA-EXTRA-PERL (main)
$ git add programa2.pl

user@DESKTOP-CS7PIHP MINGW64 ~/TAREA-EXTRA-PERL (main)
$ git commit -m "Version final Programa 02"
[main f333bfc] Version final Programa 02
 1 file changed, 16 insertions(+)

user@DESKTOP-CS7PIHP MINGW64 ~/TAREA-EXTRA-PERL (main)
$ git push origin main
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 8 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 797 bytes | 797.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To https://github.com/ChipanaGerman/TAREA_EXTRA_PERL.git
   fa6cf49..f333bfc  main -> main
```

En este segundo commit, se actualiza el archivo *programa2.pl* con la versión finalizada de nuestro ejercicio, se usa el commit *"Version final Programa 02"* y se sube a la rama main.

ENLACE A MI REPOSITORIO DE GITHUB

https://github.com/ChipanaGerman/TAREA_EXTRA_PERL

II. CONCLUSIONES

Los ejercicios en Perl desarrollados han demostrado el uso efectivo de las funcionalidades del lenguaje. En particular:

- **Ejercicio 1**: El manejo de la concatenación de cadenas a través de interpolación sin el uso del operador punto (.) resultó ser una forma eficiente y legible para generar un correo electrónico a partir de la entrada del usuario. Esto refuerza el aprendizaje sobre el uso de entradas y salidas en Perl, así como la manipulación de cadenas de forma sencilla.
- **Ejercicio 2**: La implementación de una función que elimina la menor nota y duplica la mayor nota, usando las funciones de List::Util como min, max y sum, evitó la necesidad de condicionales. Esto demuestra el poder de las funciones nativas de Perl para trabajar con listas y realizar operaciones de forma concisa. Además, la capacidad de procesar entradas dinámicas de los usuarios y manipular datos con métodos eficientes se destacó en la solución.

Ambos ejercicios demostraron el potencial de Perl como una herramienta flexible y poderosa para manipular datos y cadenas de texto, así como la facilidad de uso de sus funciones nativas.

RETROALIMENTACIÓN GENERAL

RETROALIMENTACIÓN:

- **Fortalezas**: Los ejercicios evidencian una sólida comprensión de las estructuras básicas de Perl, como el manejo de arrays, cadenas de texto y la interacción con el usuario a través de entradas estándar. El uso de las funciones adecuadas (min, max, sum, grep) en lugar de condicionales en el segundo ejercicio muestra un enfoque optimizado para la solución de problemas.
- **Áreas de mejora**: Sería útil explorar técnicas de manejo de errores más robustas para mejorar la interacción con el usuario, como verificar entradas no válidas o vacías en el primer ejercicio. En el segundo ejercicio, aunque la solución es válida para notas numéricas, la validación de entradas también podría mejorarse.

Estas mejoras garantizarían un código más robusto y preparado.

REFERENCIAS Y BIBLIOGRAFÍA

http://cslibrary.stanford.edu/108/EssentialPerl.html

https://github.com/ChipanaGerman/TAREA_EXTRA_PERL