

Universidad nacional Autónoma de Nicaragua Unan-León



Ingeniería en Telemática

Componente: Software como servicio

Nombres: Ingrid Valeria Ruiz Ulloa 21-00483-0

Lenuel Gastón Pereira Hernández 19-03199-0

Nombre del Docente: Ervin Montes

Fecha de entrega: 06/08/2024

“A la libertad de la universidad”

Guía 1: Introducción al lenguaje Ruby (parte I)

Desarrollo

1.String.

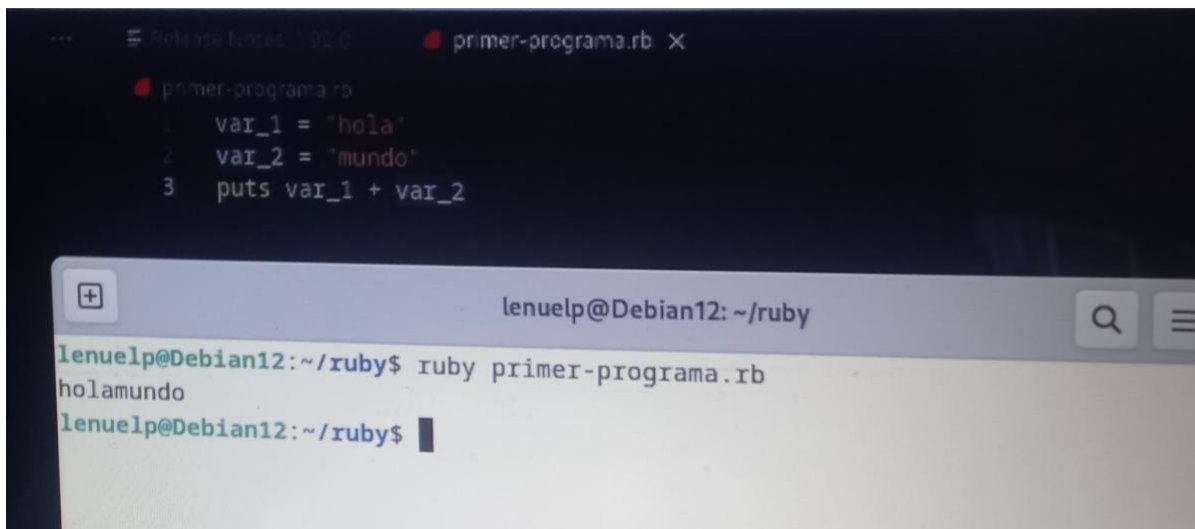
- 1.1. Crear un directorio llamado ruby, donde se almacenarán los ejercicios que se llevarán a cabo a lo largo de esta guía.
- 1.2. Crear un programa primer_programa.rb, se puede hacer desde el terminal o desde el editor de texto, asignar 2 variables de tipo String para luego imprimir por pantalla las 2 variables concatenadas.

```
var_1 = "hola"  
var_2 = "mundo"  
puts var_1 + var_2
```

- 1.3. Ejecutar el programa en el terminal.

```
$ ruby primer_programa.rb
```

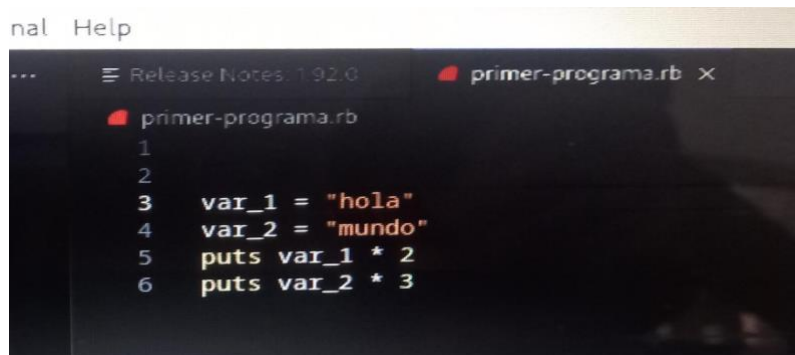
Obtendrá una salida de las 2 variables concatenadas



- 1.4. Editar el archivo creado anteriormente, agregar el siguiente código y ver lo que se muestra por pantalla.

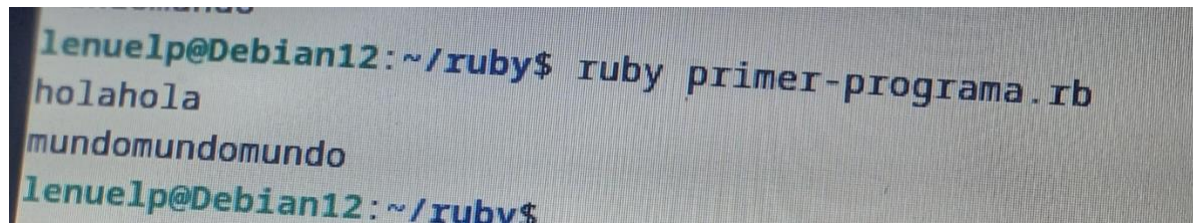
```
var_1 = "hola"  
var_2 = "mundo"  
puts var_1 * 2
```

Como se observa, se multiplica el contenido de las variables que son de tipo String, por la cantidad de veces que se le diga, en este caso la palabra hola se muestra 2 veces y la palabra mundo 3 veces.



A screenshot of a code editor window titled 'primer-programa.rb'. The code inside is as follows:

```
1  
2  
3 var_1 = "hola"  
4 var_2 = "mundo"  
5 puts var_1 * 2  
6 puts var_2 * 3
```



A screenshot of a terminal window showing the execution of the Ruby script. The prompt is 'lenuelp@Debian12:~/ruby\$'. The command 'ruby primer-programa.rb' has been entered, and the output is 'holahola' followed by 'mundomundomundo' on the next line. The prompt is now 'lenuelp@Debian12:~/ruby\$'.

Números

- 2.1. Crear un programa nuevo llamado programa_numero.rb, en el que se asignarán 2 variables enteras para realizar operaciones de aritmética básica.

```
var_1 = 20  
var_2 = 5  
  
#suma  
puts var_1 + var_2  
puts ""
```

```

#resta puts var_1 - var_2
puts ""

#multiplicar puts var_1 * var_2
puts ""

#dividir puts var_1 / var_2 puts ""

#modulo puts var_1 % var_2
puts ""

#números aleatorios puts rand(100)

```

2.2. Ejecutar el programa en el terminal y observar la salida.

```
$ ruby programa_numero.rb
```

A screenshot of the Visual Studio Code editor interface. The file explorer on the left shows 'programa-numeros.rb'. The editor window displays the following Ruby code:

```

1 var_1 = 20
2 var_2 = 5
3 #suma
4 puts var_1 + var_2
5 puts ""
6 #resta
7 puts var_1 - var_2
8 puts ""
9 #multiplicar
10 puts var_1 * var_2
11 puts ""
12 #dividir
13 puts var_1 / var_2
14 puts ""
15 #modulo
16 puts var_1 % var_2
17 puts ""
18 #números aleatorios
19 puts rand(100)

```

A screenshot of a terminal window titled 'programa-numeros.rb - numeros - Visual Studio Code'. The terminal shows the command 'ruby programa-numeros.rb' being executed, with the following output:

```

lenueip@Debian12: ~/numeros
25
15
100
4
0
98
lenueip@Debian12: ~/numeros$

```

3. Conversiones

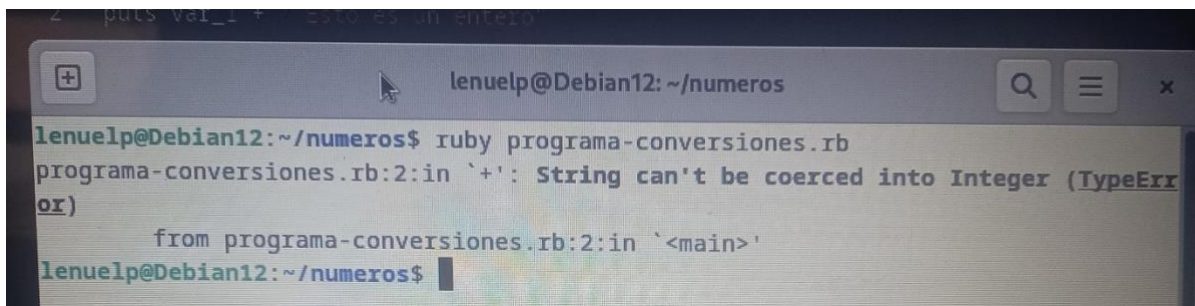
- 3.1 En Ruby existen distintos métodos que se aplican a objetos como los String, números enteros, etc. Existen métodos especiales de conversiones que se utilizan en diferentes formas o casos, para observar el funcionamiento de estos, crear un archivo `programa_conversiones.rb`, declarar una variable entera y concatenar con un texto.

```
var_1 = 22  
puts var_1 + " Esto es un entero"
```

- 3.2 Ejecutar el programa en el terminal.

```
$ ruby programa_conversiones.rb
```

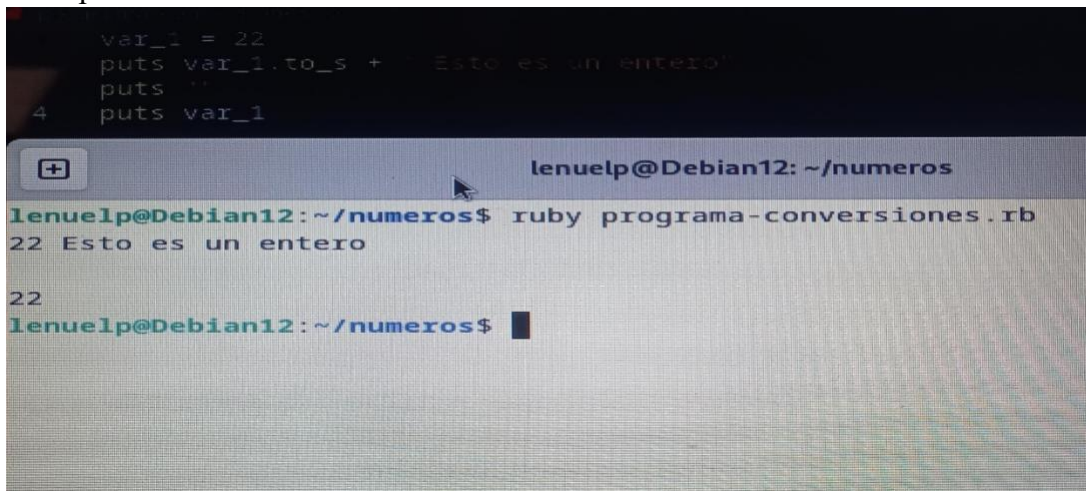
Imprime un error a como se observa en la siguiente figura, esto es debido a que no se puede concatenar un objeto de tipo entero con una cadena de caracteres.



- 3.3 Para solucionar ese error, hacer uso del método `to_s`, editar el programa y agregar:

```
var_1 = 22 puts var_1.to_s + " Esto  
es un entero"  
  
puts "" puts var_1
```

Se obtendrá una salida como en la mostrada en la siguiente figura, como se observa, aunque se ha utilizado el método `to_s`, la variable `var_1` sigue teniendo el mismo valor entero, pero su representación es como cadena de caracteres.



```
var_1 = 22
puts var_1.to_s + " Esto es un entero"
puts ""
4 puts var_1

lenuelp@Debian12: ~/numeros
lenuelp@Debian12:~/numeros$ ruby programa-conversiones.rb
22 Esto es un entero

22
lenuelp@Debian12:~/numeros$
```

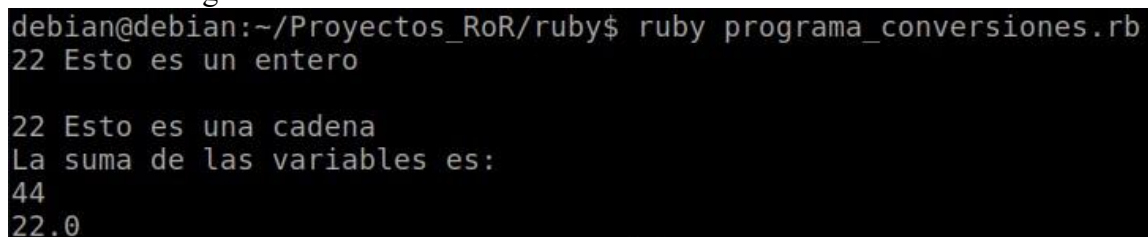
3.4 Editar nuevamente el programa para hacer uso de los métodos `to_i`, el cual convierte una variable a entero y `to_f`, el cual convierte una variable a flotante.

```
var_1 = 22 var_2 = "22" puts
var_1.to_s + " Esto es un entero"
puts "" puts var_2 + " Esto es una
cadena" puts "La suma de las
variables es:" puts var_2.to_i + var_1
puts var_2.to_f
```

3.5 Guarda los cambios y ejecutar el programa en el terminal.

```
$ ruby programa_conversiones.rb
```

Se obtendrá la siguiente salida.



```
debian@debian:~/Proyectos_RoR/ruby$ ruby programa_conversiones.rb
22 Esto es un entero

22 Esto es una cadena
La suma de las variables es:
44
22.0
```

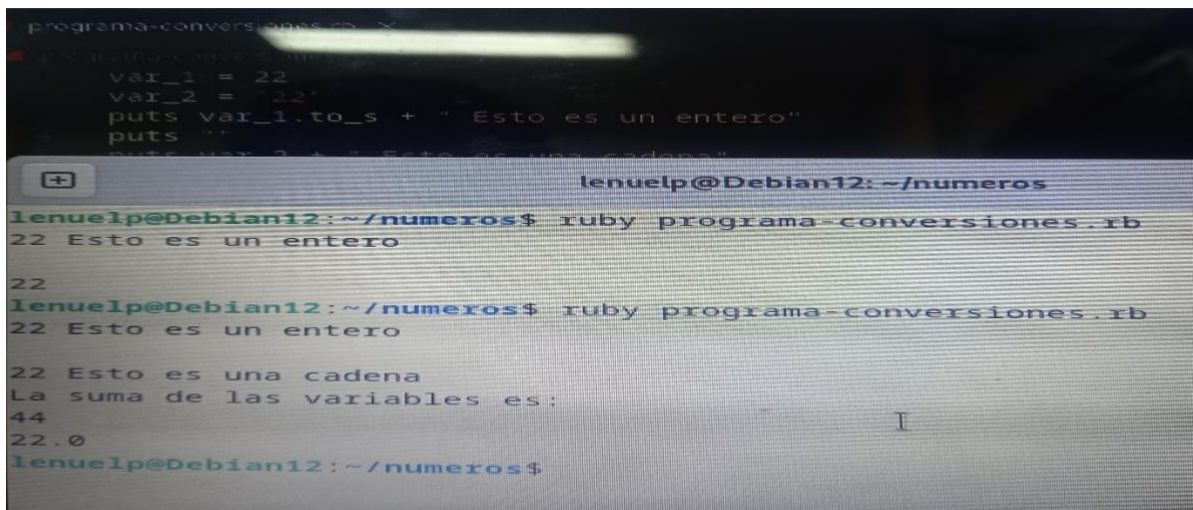
4. Métodos `gets` y `chomp`.

Se ha visto que el método **puts** se utiliza para imprimir en la pantalla; por el contrario, para leer existe el método **gets** que trabaja junto con el método **chomp**, lo que hace este último es eliminar el carácter “enter” al momento de que el método gets lee un dato del teclado.

4.1 Crear un programa leer.rb y agregar el siguiente código.

```
puts "Ingrese su primer nombre" nombre = gets
puts "Bienvenido "+ nombre + "disfrute! "
```

Al ejecutar el programa se obtiene la siguiente salida.



```
programa-conversiones.rb
var_1 = 22
var_2 = "22"
puts var_1.to_s + " Esto es un entero"
puts "La suma de las variables es: "
puts var_1 + var_2

lenuelp@Debian12: ~/numeros
lenuelp@Debian12:~/numeros$ ruby programa-conversiones.rb
22 Esto es un entero

22
lenuelp@Debian12:~/numeros$ ruby programa-conversiones.rb
22 Esto es un entero

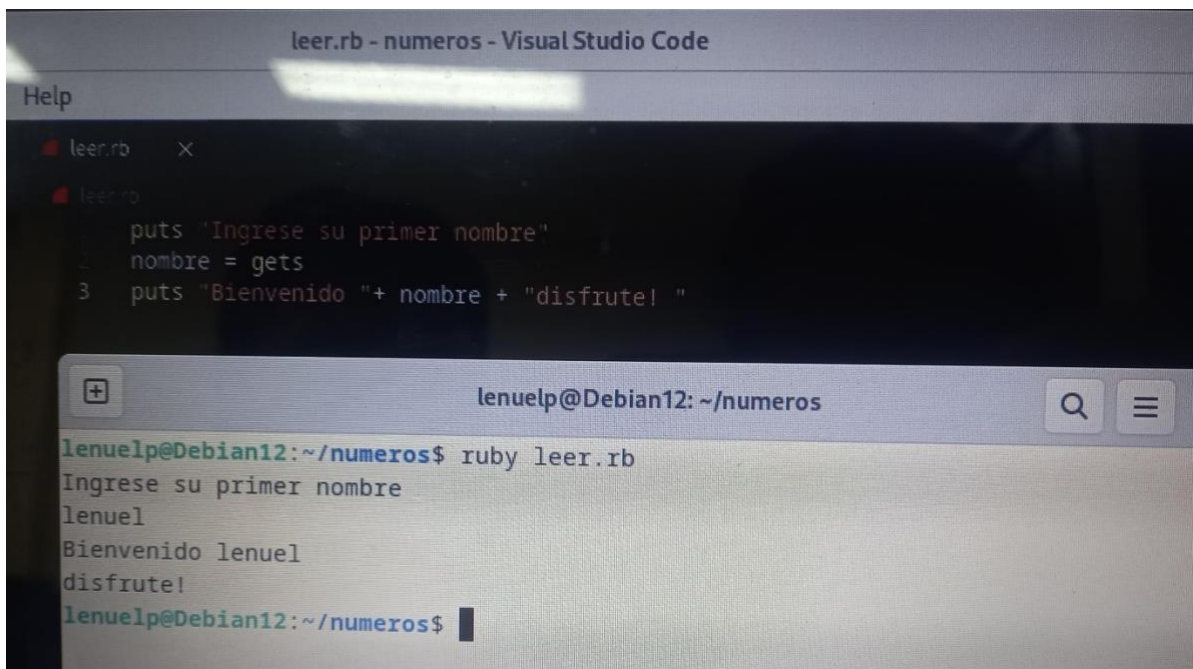
22 Esto es una cadena
La suma de las variables es:
44
22.0
lenuelp@Debian12:~/numeros$
```

Como se puede observar, el método gets recibe el carácter “enter” como un carácter más de lectura, para solucionar eso es que se utiliza el método chomp.

4.2 Editar el programa anterior y utilizar el método chomp al momento de leer el nombre.

```
puts "Ingrese su primer nombre"
nombre = gets.chomp
puts "Bienvenido #{nombre} disfrute! "
```

Al ejecutar el programa se observa la diferencia, cuando se usa el método **chomp** que ya no captura el enter como un carácter más, de igual forma se observa como la variable **nombre** es impresa de una forma distinta a las anteriores.



The image shows a screenshot of a development environment. At the top, a Visual Studio Code window titled 'leer.rb - numeros - Visual Studio Code' displays a Ruby script in a file named 'leer.rb'. The script contains three lines of code: a 'puts' statement to prompt for a name, a 'gets' statement to capture the input, and another 'puts' statement to print a welcome message using string interpolation. Below the code editor, a terminal window titled 'lenuelp@Debian12: ~/numeros' shows the execution of the script. The user runs 'ruby leer.rb', and the terminal displays the program's output: a prompt, the user's input 'lenuel', and a formatted welcome message 'Bienvenido lenuel disfrute!'.

```
leer.rb - numeros - Visual Studio Code
Help
leer.rb
puts "Ingrese su primer nombre"
nombre = gets
puts "Bienvenido " + nombre + "disfrute! "
```

```
lenuelp@Debian12: ~/numeros
lenuelp@Debian12:~/numeros$ ruby leer.rb
Ingrese su primer nombre
lenuel
Bienvenido lenuel
disfrute!
lenuelp@Debian12:~/numeros$
```

5. Métodos de String

Como se menciona anteriormente, en Ruby existen distintos métodos que se pueden aplicar a cada uno de los objetos del lenguaje, en esta sección se conocerá sobre los métodos relacionados a los String.

5.1 Crear un nuevo programa string.rb y agregar el siguiente código:


```

puts "Ingrese su nombre" nombre
= gets.chomp

#Imprime el nombre ingresado
puts "Nombre => " + nombre

# convierte al revés el nombre
puts "Método reverse => " + nombre.reverse
#Mayúscula
puts "Método upcase => " + nombre.upcase
#Minúscula
puts "Método downcase => " + nombre.downcase

#Intercambia las minúsculas por mayúscula (viceversa)
puts "Método swapcase => " + nombre.swapcase

#cambia el primer carácter a mayúscula puts
"Método capitalize => " + nombre.capitalize

#devuelve el tamaño del string ingresado puts
"Método length => " + nombre.length.to_s

```

5.2 Ejecute el programa en el terminal y observar el comportamiento de los métodos.

```

Aug 6 15:17
string.rb - numeros - Visual Studio Code
Terminal Help

puts "Ingrese su nombre"
nombre = gets.chomp
puts "Nombre => " + nombre
puts "Método reverse => " + nombre.reverse
puts "Método upcase => " + nombre.upcase
puts "Método downcase => " + nombre.downcase
puts "Método swapcase => " + nombre.swapcase
puts "Método capitalize => " + nombre.capitalize
puts "Método length => " + nombre.length.to_s

lenuelpe@Debian12: ~/numeros$ ruby string.rb
Ingrese su nombre
ingrid
Nombre => ingrid
Método reverse => dirgni
Método upcase => INGRID
Método downcase => ingrid
Método swapcase => INGRID
Método capitalize => Ingrid
Método length => 6
lenuelpe@Debian12: ~/numeros$

```

6. Condicionales y bucles

- 6.1 Los condicionales y los bucles en Ruby funcionan de la misma manera que en otros lenguajes de programación, para ver el funcionamiento, crear un programa nuevo y agregar el siguiente código.

```
iterador = " "

while iterador.downcase != "s"

  puts "Ingrese un nombre"
  nombre = gets.chomp
  tamaño = nombre.length
  if (tamaño >= 5 )

    puts "Su nombre tiene más de 5 caracteres"

  else

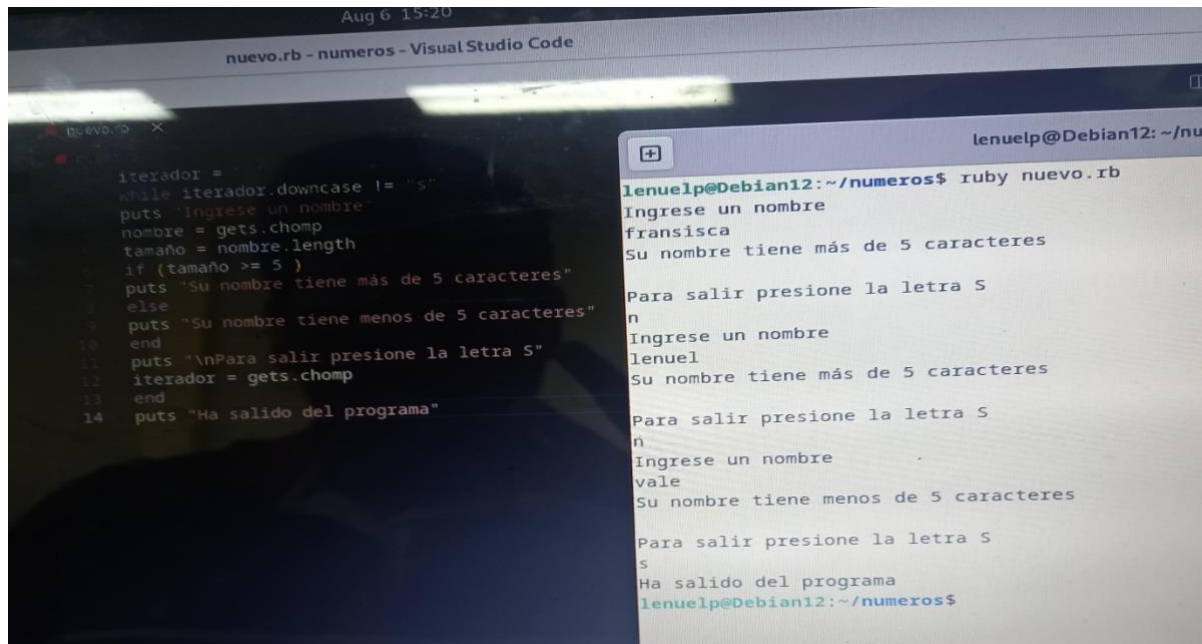
    puts "Su nombre tiene menos de 5 caracteres"

  end

  puts "\nPara salir presione la letra S"

  iterador = gets.chomp end
  puts "Ha salido del programa"
```

- 6.2 Ejecutar e interactuar con el programa para ver su funcionamiento.



```
Aug 6 15:20
nuevo.rb - numeros - Visual Studio Code

iterador = " "

while iterador.downcase != "s"

  puts "Ingrese un nombre"
  nombre = gets.chomp
  tamaño = nombre.length
  if (tamaño >= 5 )

    puts "Su nombre tiene más de 5 caracteres"

  else

    puts "Su nombre tiene menos de 5 caracteres"

  end

  puts "\nPara salir presione la letra S"

  iterador = gets.chomp end
  puts "Ha salido del programa"

lenuelp@Debian12: ~/numeros$ ruby nuevo.rb
Ingrese un nombre
fransisca
Su nombre tiene más de 5 caracteres

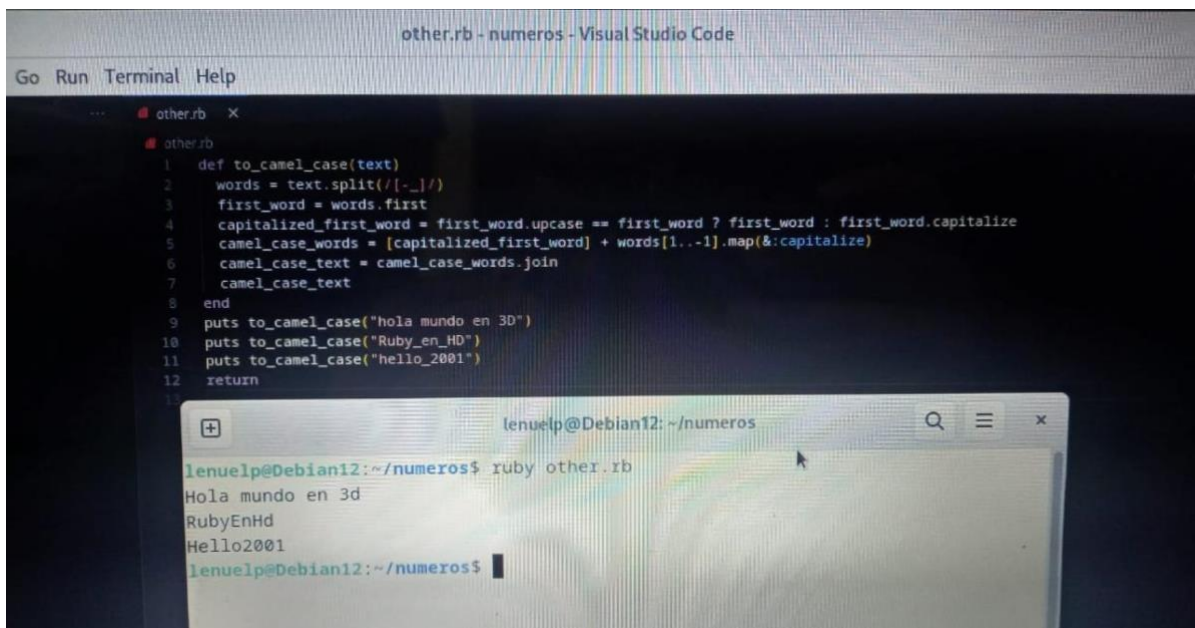
Para salir presione la letra S
n
Ingrese un nombre
lenuel
Su nombre tiene más de 5 caracteres

Para salir presione la letra S
n
Ingrese un nombre
vale
Su nombre tiene menos de 5 caracteres

Para salir presione la letra S
s
Ha salido del programa
lenuelp@Debian12:~/numeros$
```

Ejercicios propuestos para ser entregados al docente

1. Realizar cada uno de los enunciados de la guía, probar su funcionamiento y analizar cada uno de los programas planteados.
2. Complete el método/función para que convierta las palabras delimitadas por guiones/guiones bajos en mayúsculas y minúsculas. La primera palabra dentro de la salida debe estar en mayúsculas solo si la palabra original estaba en mayúsculas (conocido como Upper Camel Case, también conocido como caso Pascal). Las siguientes palabras deben estar siempre en mayúscula.



The image shows a screenshot of a Visual Studio Code editor window titled 'other.rb - numeros - Visual Studio Code'. The editor displays a Ruby script with the following code:

```
1 def to_camel_case(text)
2   words = text.split(/[-_]/)
3   first_word = words.first
4   capitalized_first_word = first_word.upcase == first_word ? first_word : first_word.capitalize
5   camel_case_words = [capitalized_first_word] + words[1..-1].map(&:capitalize)
6   camel_case_text = camel_case_words.join
7   camel_case_text
8 end
9 puts to_camel_case("hola mundo en 3D")
10 puts to_camel_case("Ruby_en_Hd")
11 puts to_camel_case("hello_2001")
12 return
```

Below the editor, a terminal window titled 'lenuelp@Debian12: ~/numeros' shows the output of running the script with the command 'ruby other.rb':

```
lenuelp@Debian12:~/numeros$ ruby other.rb
Hola mundo en 3d
RubyEnHd
Hello2001
lenuelp@Debian12:~/numeros$
```

Ejemplos

"the-stealth-warrior" se convierte en "the Stealth Warrior"

"The_Stealth_Warrior" se convierte en "The_Stealth_Warrior"

"The_Stealth-Warrior" se convierte en "The_Stealth-Warrior"