

Unidad 1.

Tecnológico Nacional de México.

Instituto Tecnológico de Veracruz

Ingeniería En Sistemas Computacionales

Lenguajes Y Autómatas II

PRÁCTICA UNIDAD I.

COMPROBACIÓN Y VALIDACIÓN DE TIPOS DE DATOS Y ATRIBUTOS HEREDADOS.

CATEDRÁTICO

ING. MARTHA MARTÍNEZ MORENO

ALUMNOS

VILMA ZORINA CAMACHO CÁGAL.
JOSÉ MANUEL MIRANDA VILLAGRAN.

Validaciones Semánticas en Ruby.

ÍNDICE

- ✓ Lenguaje a Comprobar.....pag.-3
- ✓ ¿Qué es un error Semántico? pag.-4
- ✓ Ejercicios.....pag.-5 - 28
- ✓ Bibliografía..... pag.-29

Lenguaje a Comprobar.



Ruby es un lenguaje de programación con licencia de software libre relativamente joven, creado en 1995 por el desarrollador japonés Yukihiro Matsumoto y que no se dio a conocer en occidente hasta cierto tiempo después, ya que toda la documentación asociada estaba escrita precisamente en japonés.

En el proceso de creación, se tomaron ciertas características de otros lenguajes ya existentes como Python o Perl como base, consiguiendo así un lenguaje bastante potente. Esta potencia reside en la versatilidad, la libertad y en cierta medida en lo simple que resulta programar en él. En palabras de su propio creador, su intención fue "crear un lenguaje de programación centrado en facilitar las cosas al usuario y no a la máquina". Así pues, Ruby permite cierta flexibilidad a la hora de programar que otros lenguajes clásicos no tienen.

Si hablamos de características más concretas, podríamos decir que Ruby es un lenguaje orientado a objetos. Más que orientado a objetos, en Ruby absolutamente todo es un objeto, hasta los tipos de datos más básicos como los enteros o los booleanos.

Así pues, siguiendo esta línea nos encontramos con que las variables son simplemente instancias de estos objetos ya mencionados, y que las funciones son métodos de estas clases, aunque ya hablaremos de ello en profundidad más adelante.

Ruby es además un lenguaje interpretado, una característica poco común en otros lenguajes de programación más populares. Esto significa que Ruby no tiene un compilador que traduzca las sentencias a lenguaje máquina antes de su ejecución, sino que el propio "intérprete" de Ruby es el encargado de traducir y ejecutar el programa al mismo tiempo.

¿Qué es un error Semántico?

Un error semántico se produce cuando la sintaxis del código es correcta, pero la semántica o el significado no es el que se pretendía.

Un error semántico puede hacer que el programa termine de forma anormal, con o sin un mensaje de error.



Ejercicios.



Problema.

Realizar una multiplicación con dos números utilizando la lectura de datos por teclado.

En el lenguaje Ruby las lecturas de datos por medio del teclado se realizan utilizando la función **gets**, esta función guarda la información leída como un "String".

```
40 puts "Ingrese dos numeros: "  
41 • x=gets  
42 • y=gets  
43 multiplica(x,y)
```

Figura 1.- Uso de la función gets.

Compilación sin parseo:

```
C:\Ruby24-x64\programas>ruby errores.rb
multiplicar (m)
dividir (d)
cadenas (c)
Escriba la inicial de la operacion a realizar: m
Ingresa dos numeros:
2
4
errores.rb:7:in `*': no implicit conversion of String into Integer (TypeError)
    from errores.rb:7:in `multiplica'
    from errores.rb:43:in `<main>'

C:\Ruby24-x64\programas>_
```

Figura 1.1.- Compilación sin Parseo (Error de Conversión).

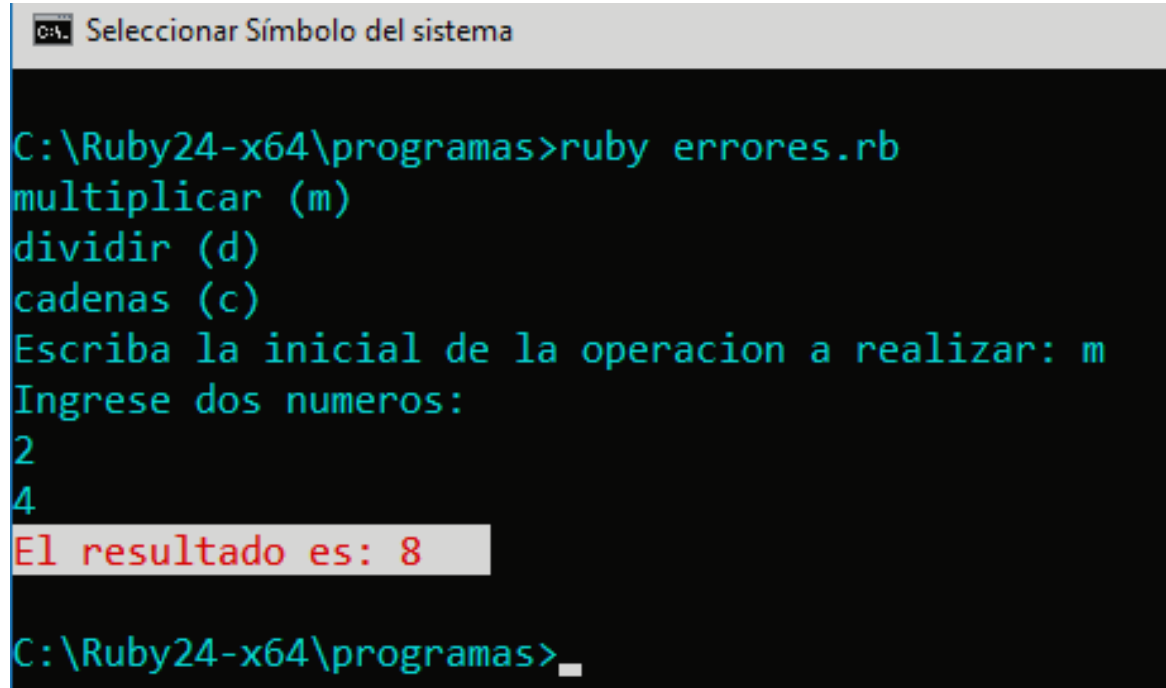
Conclusión

Si queremos realizar alguna operación con un tipo de dato entero o flotante es necesario realizar un parseo mediante las funciones **.to_i** (Tipo de dato entero) y **.to_f** (Tipo de dato flotante).

```
40 puts "Ingreso dos numeros: "  
41 • x=gets.to_i  
42 • y=gets.to_i  
43 multiplica(x,y)
```

Figura 1.2.- Uso de la función .to_i.

Compilación con parseo (Usando la función `.to_i`) :



```
C:\Ruby24-x64\programas>ruby errores.rb
multiplicar (m)
dividir (d)
cadenas (c)
Escriba la inicial de la operacion a realizar: m
Ingresa dos numeros:
2
4
El resultado es: 8
C:\Ruby24-x64\programas>_
```

Figura 1.3.- Compilación con Parseo.



2.- DIVISIÓN ENTRE CERO

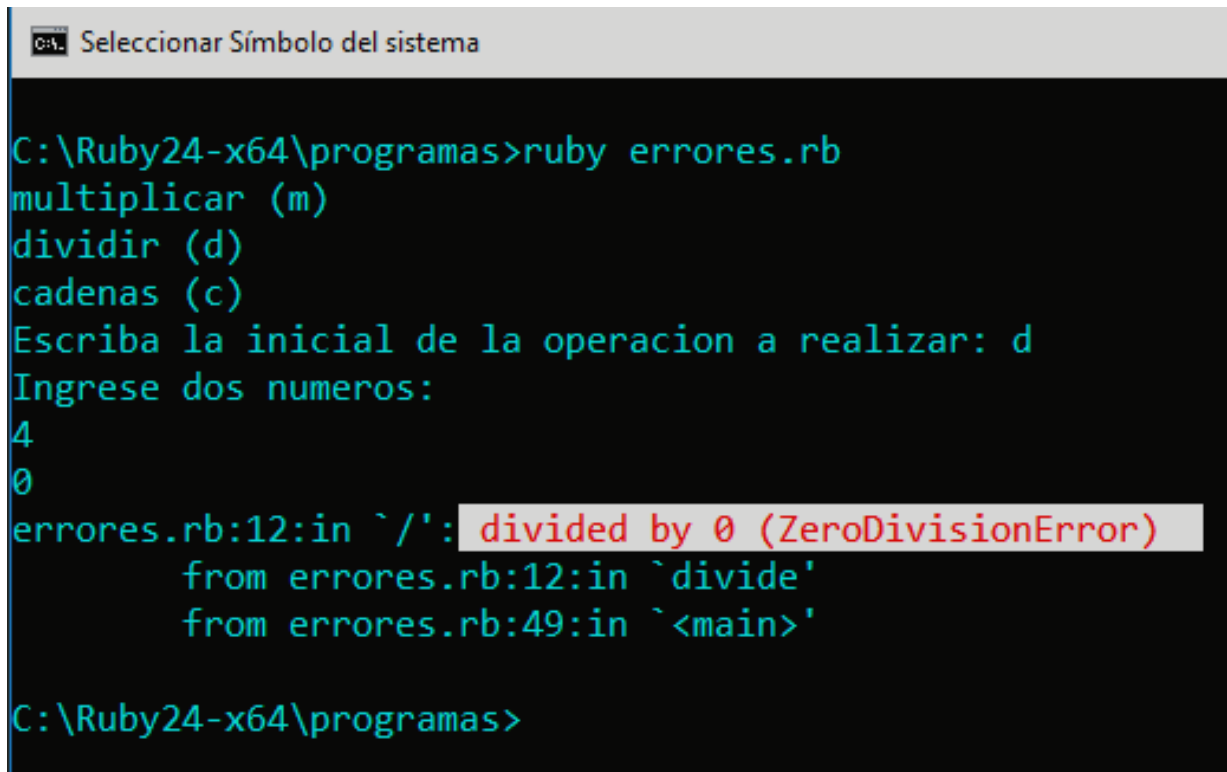
Problema.

Comprobar si se puede realizar una división, el dividendo podrá ser un valor numérico mayor a cero y el valor de divisor será de cero.

En aritmética y algebra es considerada una indefinición que puede originar paradojas matemáticas.

En los números naturales, enteros y reales, la división entre cero no posee un valor definido, debido a que para todo número n , el producto $n \cdot 0 = 0$, por lo que el 0 no tiene inverso multiplicativo.

Compilación división entre cero:



```
Seleccionar Símbolo del sistema

C:\Ruby24-x64\programas>ruby errores.rb
multiplicar (m)
dividir (d)
cadenas (c)
Escriba la inicial de la operacion a realizar: d
Ingrese dos numeros:
4
0
errores.rb:12:in `/' : divided by 0 (ZeroDivisionError)
    from errores.rb:12:in `divide'
    from errores.rb:49:in `'

C:\Ruby24-x64\programas>
```

Figura 2.- Compilación división entre cero.

Conclusión

Ruby no acepta dividir un número entre cero, ya que esta división es un error semántico. El divisor siempre debe de ser mayor a cero.



3.- DIVISIÓN ENTRE DISTINTOS TIPOS DE DATOS.

Problema.

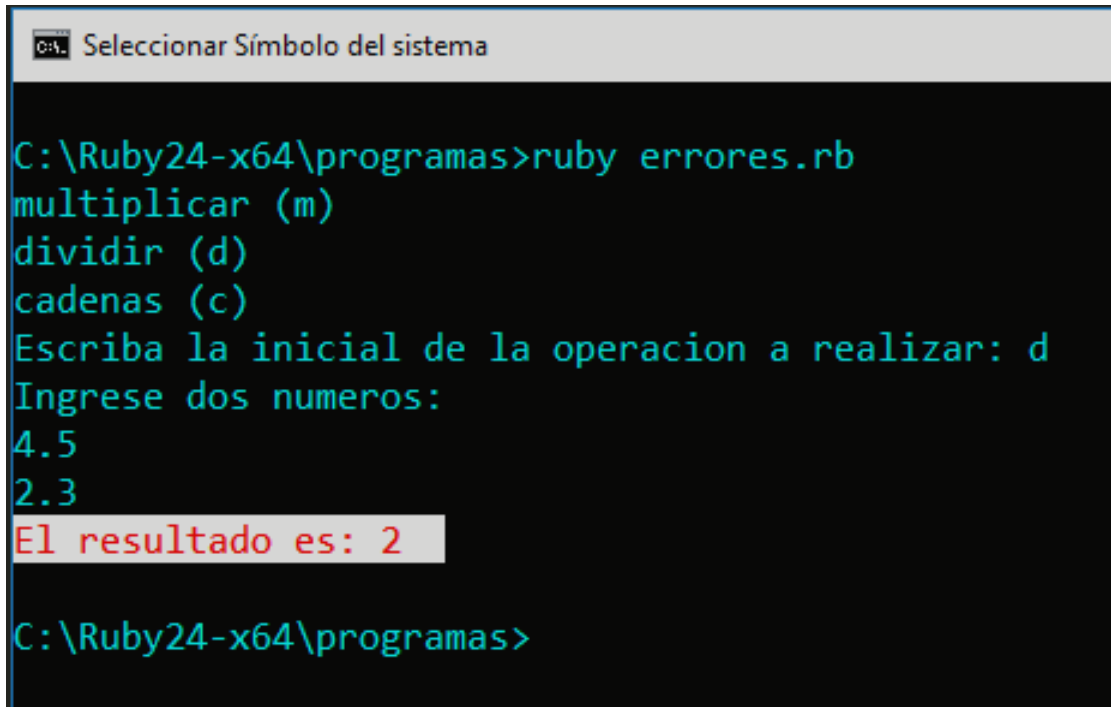
Realizar una división de dos números utilizando la lectura de datos por teclado.

En el método divide tenemos las variables a y b parseadas a números enteros. Observaremos el resultado al ingresar dos números flotantes al método divide.

```
10 def divide(a,b)
11   puts "El resultado es: #{a.to_i/b.to_i}"
12 end
```

Figura 3.- Parseo a números enteros.

Compilación división de dos valores flotantes:



```
C:\Ruby24-x64\programas>ruby errores.rb
multiplicar (m)
dividir (d)
cadenas (c)
Escriba la inicial de la operacion a realizar: d
Ingrese dos numeros:
4.5
2.3
El resultado es: 2
C:\Ruby24-x64\programas>
```

Figura 3.1.- Compilación división de dos números flotantes.

Conclusión

Ruby por defecto eliminara el punto decimal de los valores flotantes y realizara la división solo los con los números enteros.



4.- SENSIBLE A LAS MAYÚSCULAS

Problema.

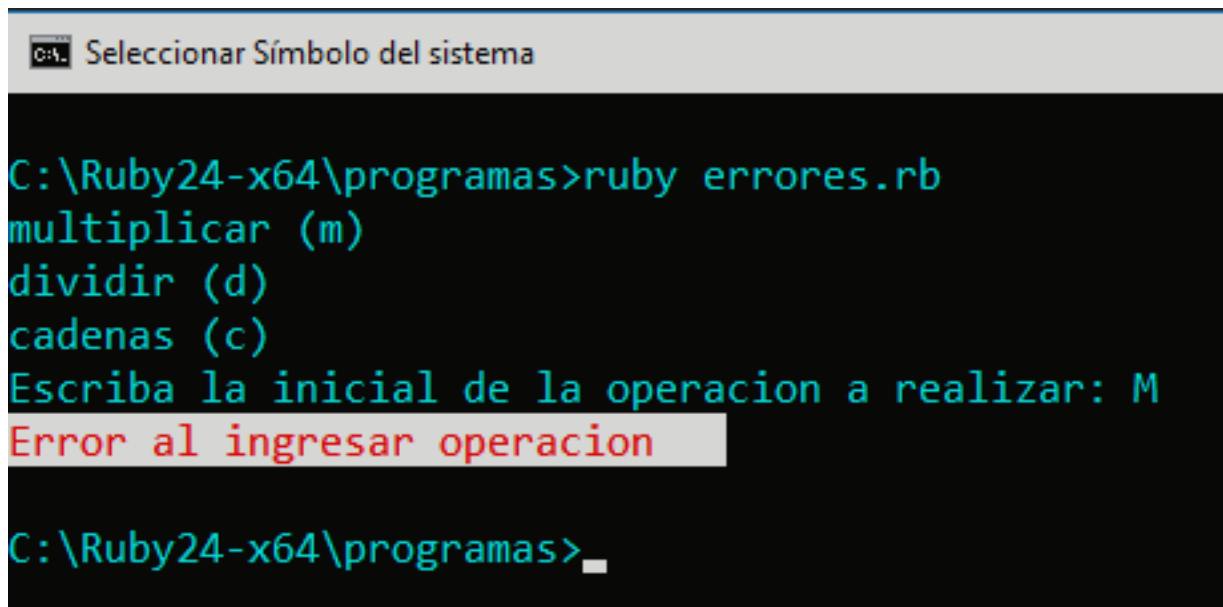
Realizar distintas operaciones mediante un menú.

El menú está realizado con la ayuda de un case, las opciones se pueden estar declaradas en letras minúsculas.

```
38 when "m"
39     puts "Ingrese dos numeros: "
40     x=gets.to_i
41     y=gets.to_i
42     multiplica(x,y)
43
44 when "d"
45     puts 'Ingrese dos numeros: '
46     x=gets
47     y=gets
48     divide(x,y)
49
50 when "c"
51     puts 'Ingrese una cadena: '
52     cad= gets
53     cadena(cad)
```

Figura 4.- Opciones declaradas en minúsculas.

Compilación de acceso a las opciones ingresando una letra mayúscula:



```
C:\Ruby24-x64\programas>ruby errores.rb
multiplicar (m)
dividir (d)
cadenas (c)
Escriba la inicial de la operacion a realizar: M
Error al ingresar operacion
C:\Ruby24-x64\programas>_
```

Figura 4.1.- Compilación ingresando una letra mayúscula.

Conclusión

Ruby es sensible a las mayúsculas, eso hace que no podamos realizar la operación deseada si la ingresamos con una letra mayúscula.



5.- INGRESAR UN DATO ERRONEO

Problema.

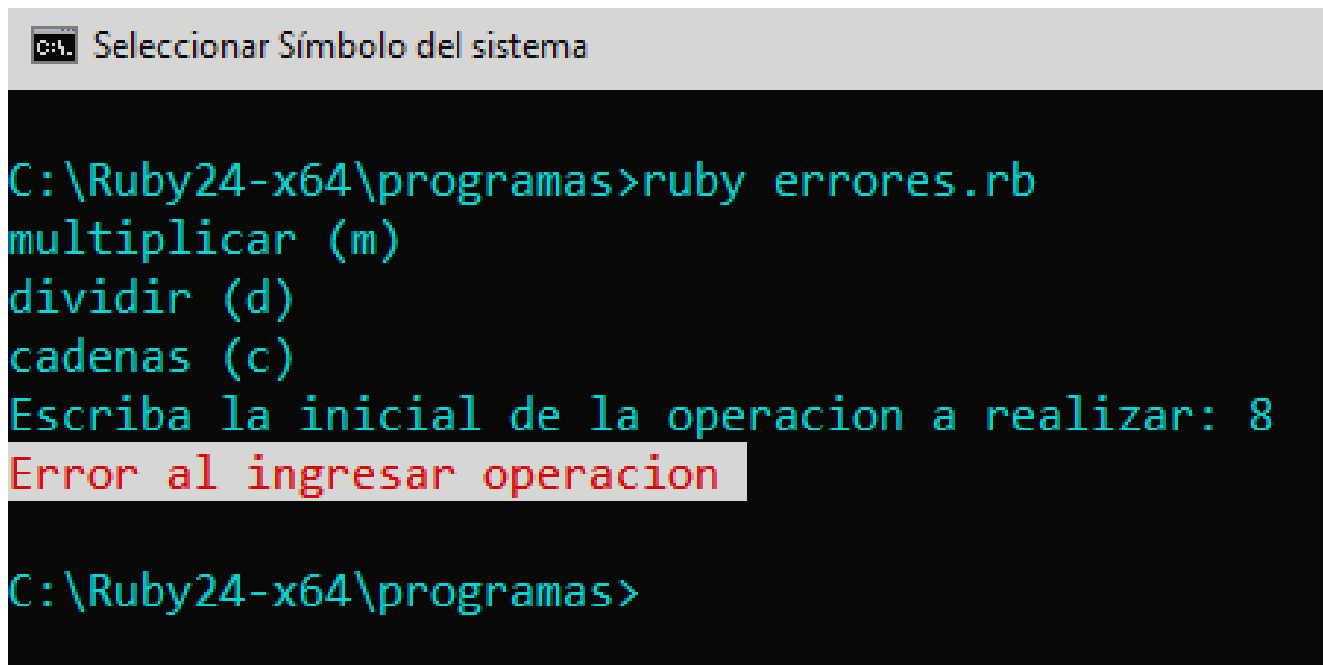
Realizar distintas operaciones mediante un menú.

El menú está realizado con la ayuda de un case, las opciones se pueden estar declaradas en letras minúsculas.

```
38 when "m"
39     puts "Ingrese dos numeros: "
40     x=gets.to_i
41     y=gets.to_i
42     multiplica(x,y)
43
44 when "d"
45     puts 'Ingrese dos numeros: '
46     x=gets
47     y=gets
48     divide(x,y)
49
50 when "c"
51     puts 'Ingrese una cadena: '
52     cad= gets
53     cadena(cad)
```

Figura 5.- Opciones declaradas en minúsculas.

Compilación de acceso a las opciones ingresando número:



```
C:\Ruby24-x64\programas>ruby errores.rb
multiplicar (m)
dividir (d)
cadenas (c)
Escriba la inicial de la operacion a realizar: 8
Error al ingresar operacion
C:\Ruby24-x64\programas>
```

Figura 5.1.- Compilación ingresando un número.

Conclusión

Al querer acceder a nuestras opciones ingresando un número es imposible ya que el acceso solo se realiza con letras minúsculas.



6.- MULTIPLICAR CADENAS

Problema.

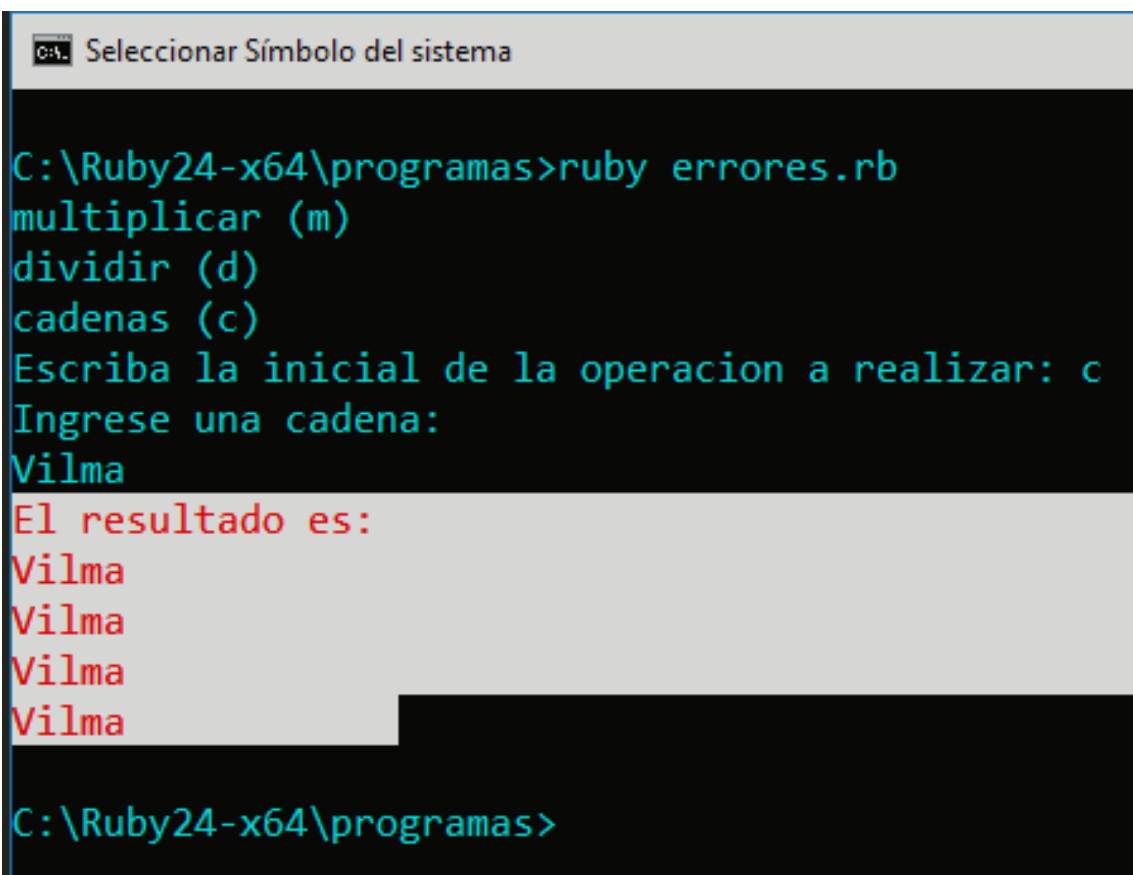
Imprimir varias veces la misma cadena de texto.

En la impresión del método cadena tenemos el símbolo `*` esta función de Ruby nos ayuda a imprimir varias veces el contenido de la variable de tipo String. En este caso se imprimirá 4 veces el contenido de la variable `a`.

```
14 #Multiplicar una cadena
15 def cadena(a)
16   puts 'El resultado es: '
17   puts "#{a*4}"
18 end
```

Figura 6.- Uso del símbolo `*` en la impresión.

Compilación del método cadena:



```
C:\Ruby24-x64\programas>ruby errores.rb
multiplicar (m)
dividir (d)
cadenas (c)
Escriba la inicial de la operacion a realizar: c
Ingrese una cadena:
Vilma
El resultado es:
Vilma
Vilma
Vilma
Vilma
C:\Ruby24-x64\programas>
```

Figura 6.1.-Compilación Impresión múltiple de la cadena.

Conclusión

En Ruby es posible realizar impresiones múltiples del contenido de una variable tipo String con el uso del símbolo *



7.- FUNCIÓN DE PARAMETROS FALTANTES.

Problema.

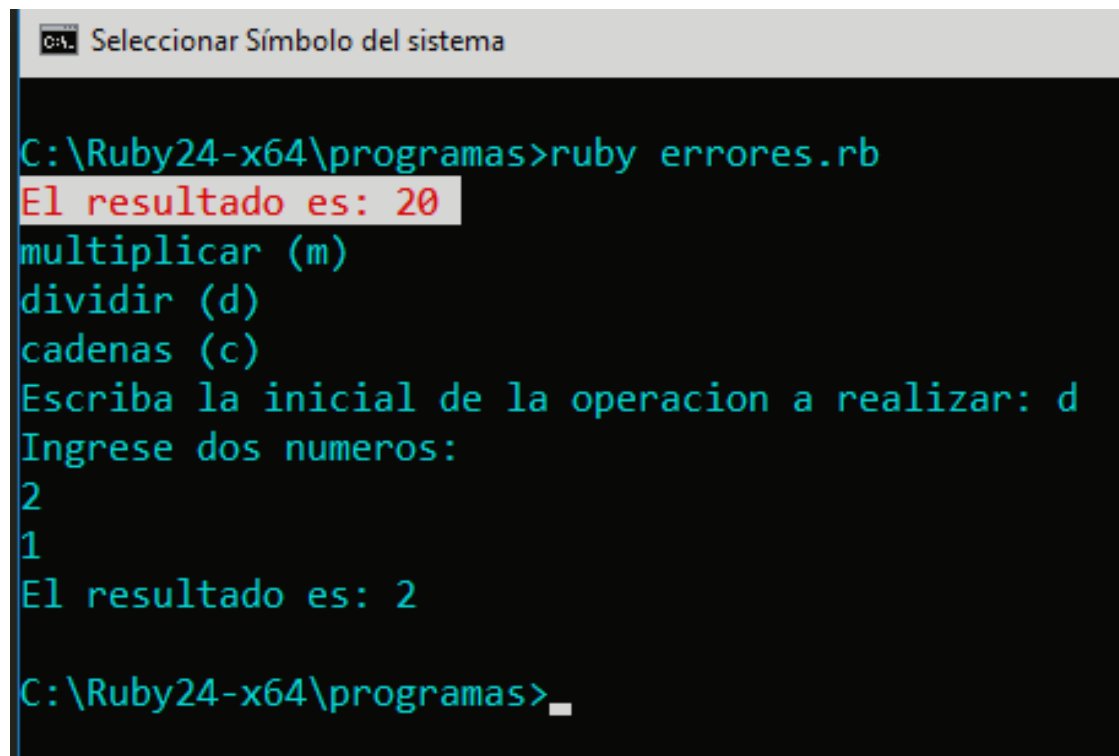
Llamada a una método.

Llamamos al método multiplica directamente en el código pasando como parámetros 10 y 2.

```
4
5 def multiplica(a,b)
6 #Para el uso de puts podemos utili
7   puts "El resultado es: #{a*b}"
8 end
9
10 #Error de Parametros Faltantes
11 multiplica(10,2)
```

Figura 7.-Llamada al método multiplica.

Compilación de llamada al método multiplica:



```
Selecciconar Smbolo del sistema

C:\Ruby24-x64\programas>ruby errores.rb
El resultado es: 20
multiplicar (m)
dividir (d)
cadenas (c)
Escriba la inicial de la operacion a realizar: d
Ingrese dos numeros:
2
1
El resultado es: 2

C:\Ruby24-x64\programas>_
```

Figura 7.1.- Compilación ingresando una letra mayúscula.

Conclusión

El error semántico se encuentra en la llamada al método multiplica, ya que no debería de realizar otra operación.



8.- ARREGLOS CON EL MISMO NOMBRE.

Problema.

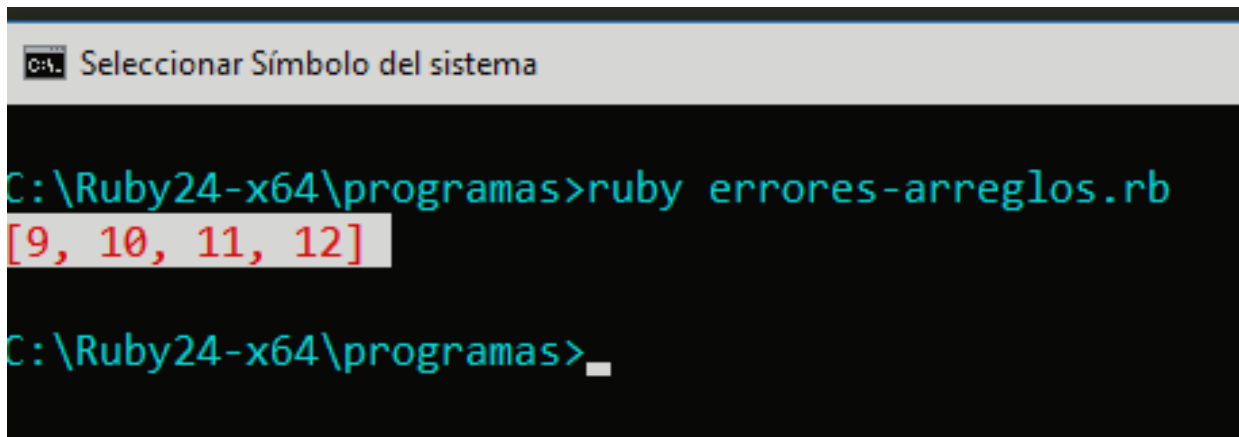
Impresión de un arreglo.

Declaración de tres arreglos con valores distintos, pero dos arreglos han sido declarados con el mismo nombre "vec2".

```
4  vec=[3,4,5,6]
5  vec2=[1,2,vec,7,8]
6  vec2=[9,10,11,12]
7  print vec2
```

Figura 8.-Duplicidad del arreglo vec2.

Compilación de impresión del arreglo duplicado vec2:

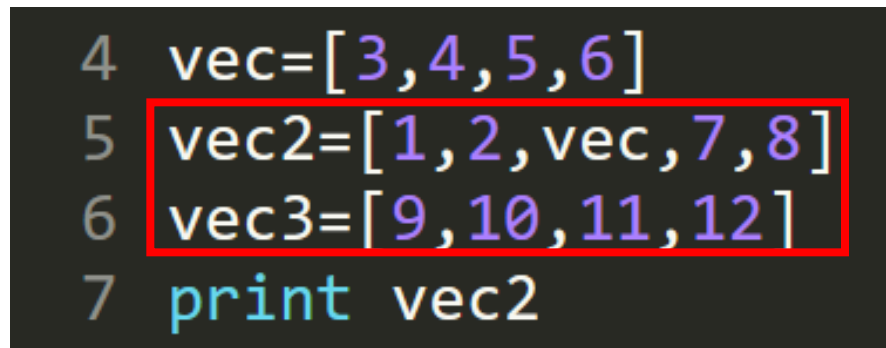


```
C:\Ruby24-x64\programas>ruby errores-arreglos.rb
[9, 10, 11, 12]
C:\Ruby24-x64\programas>
```

Figura 8.1.-Compilación de la impresión del arreglo duplicado vec2.

Conclusión

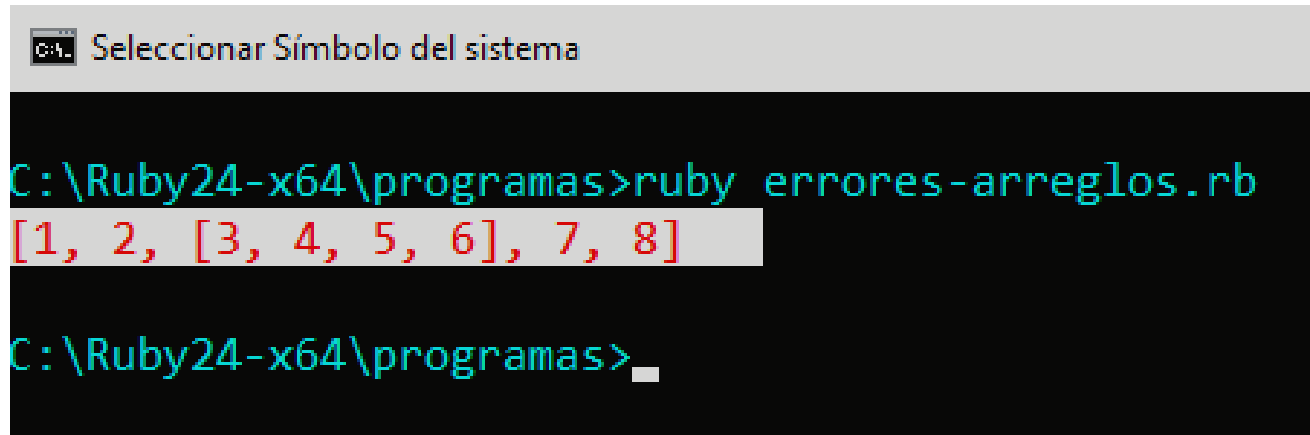
Anteriormente observamos que la impresión muestra el ultimo contenido del arreglo vec2, ahora cambiaremos el nombre del último arreglo a vec3 para obtener el contenido correcto.



```
4 vec=[3,4,5,6]
5 vec2=[1,2,vec,7,8]
6 vec3=[9,10,11,12]
7 print vec2
```

Figura 8.2.-Declaración de tres arreglos.

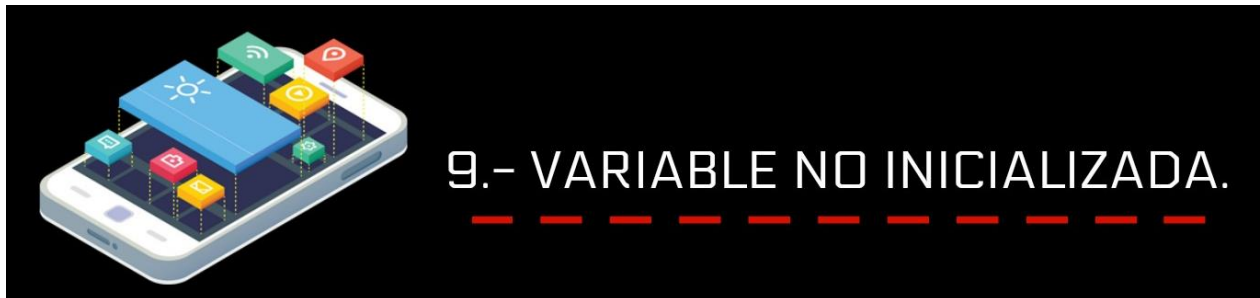
Compilación de impresión del arreglo vec2:



The screenshot shows a Windows command prompt window with a title bar that reads "Seleccionar Símbolo del sistema". The command prompt displays the following text:

```
C:\Ruby24-x64\programas>ruby errores-arreglos.rb  
[1, 2, [3, 4, 5, 6], 7, 8]  
  
C:\Ruby24-x64\programas>_
```

Figura 8.3.-Compilación de la impresión del arreglo vec2.



Problema.

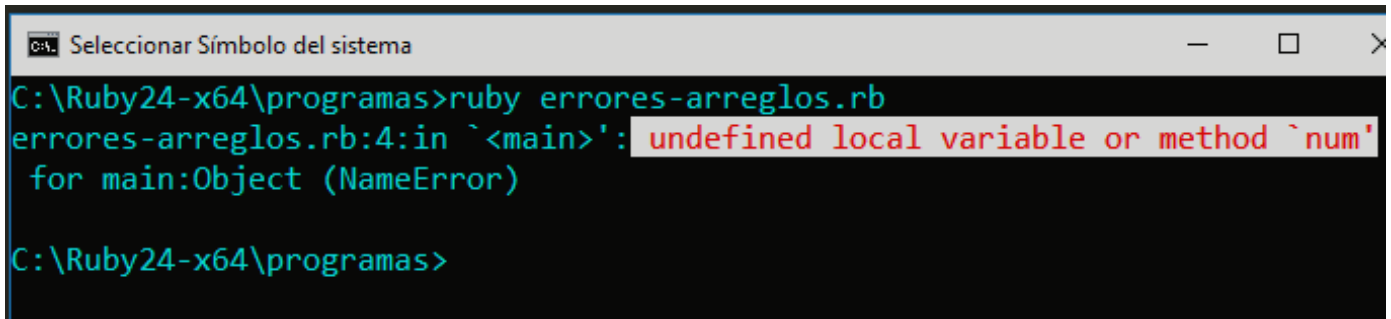
Impresión de un arreglo.

Declaración de tres arreglos con valores distintos, en el arreglo con el nombre vec se guarda el contenido de la variable num (La variable num no esta inicializada).

```
2 #Variable num no inicilizada
3 #num=3
4 vec=[num,4,5,6]
5 vec2=[1,2,vec,7,8]
6 vec3=[9,10,11,12]
7 print vec2
8
```

Figura 9.-Variable num no inicializada.

Compilación asignación de la variable num al arreglo vec:



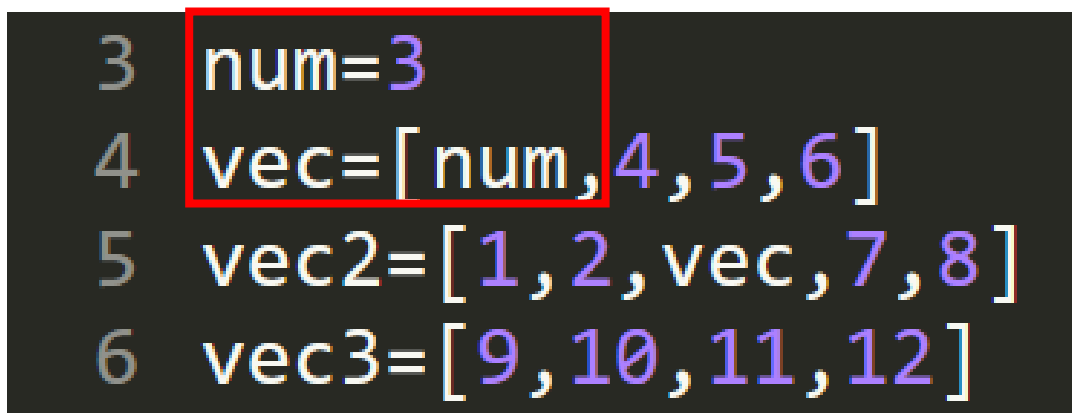
```
C:\Ruby24-x64\programas>ruby errores-arreglos.rb
errores-arreglos.rb:4:in `<main>': undefined local variable or method `num'
for main:Object (NameError)

C:\Ruby24-x64\programas>
```

Figura 9.1.-Compilación de la asignación de la variable num al arreglo vec.

Conclusión

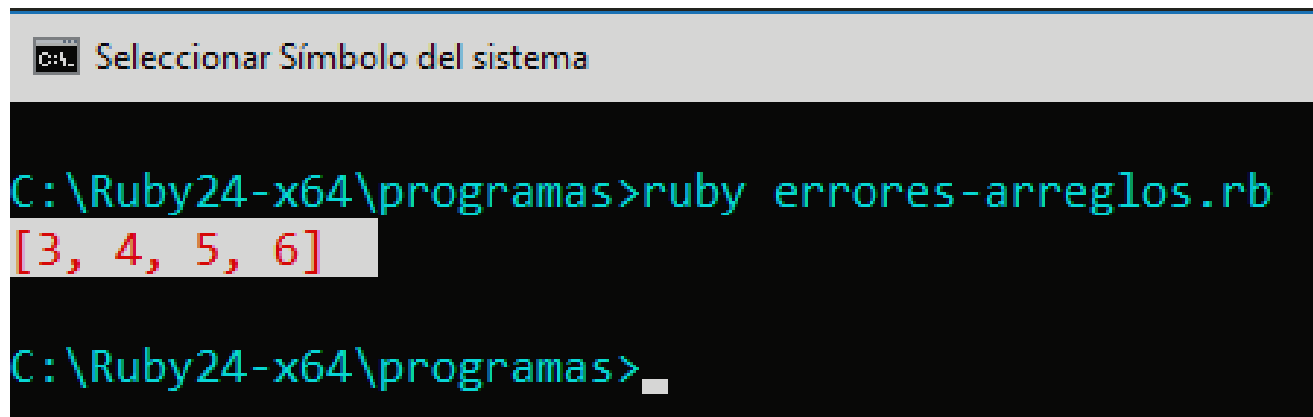
Declaramos la variable num y le asignamos un valor numero en este caso le asignaremos un valor de 3.



```
3 num=3
4 vec=[num,4,5,6]
5 vec2=[1,2,vec,7,8]
6 vec3=[9,10,11,12]
```

Figura 9.2.-Declaración y asignación de la variable num.

Compilación impresión del arreglo vec:



The screenshot shows a Windows command prompt window with a title bar that reads "C:\ Ruby24-x64\programas" and "Seleccionar Símbolo del sistema". The command prompt has a black background with green text. The first line shows the command `C:\Ruby24-x64\programas>ruby errores-arreglos.rb`. The second line shows the output `[3, 4, 5, 6]` in red text. The third line shows the prompt `C:\Ruby24-x64\programas>` with a white cursor.

```
C:\Ruby24-x64\programas>ruby errores-arreglos.rb
[3, 4, 5, 6]
C:\Ruby24-x64\programas>
```

Figura 9.3.-Compilación impresión del arreglo vec.



11.- SUMA USANDO UN STRING Y UN ENTERO.

Problema.

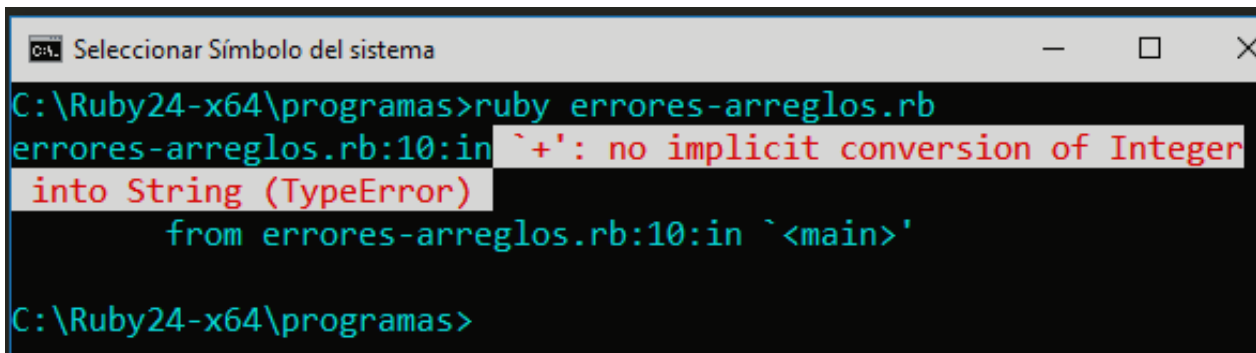
Impresión de una suma.

Impresión de una suma usando una cadena (con el contenido de "mensaje") y un valor numérico (2).

```
9 #Error de suma con string y un entero
10 puts "El resultado es: #{ "mensaje"+2 }"
11 gets()
```

Figura 11.-Suma usando una cadena y un valor numérico.

Compilación de la suma de una cadena y un valor numérico:

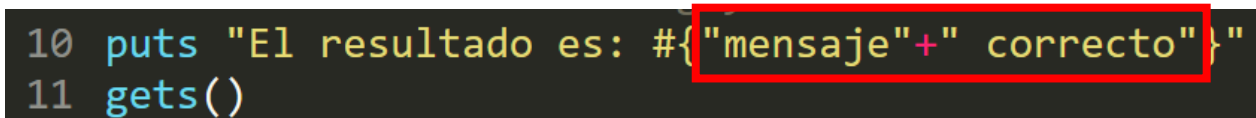
A screenshot of a Windows command prompt window titled "Selecciónar Símbolo del sistema". The prompt shows the command `C:\Ruby24-x64\programas>ruby errores-arreglos.rb`. The output is a red error message: `errores-arreglos.rb:10:in `+': no implicit conversion of Integer into String (TypeError)`, followed by `from errores-arreglos.rb:10:in `<main>'`. The prompt then returns to `C:\Ruby24-x64\programas>`.

```
C:\Ruby24-x64\programas>ruby errores-arreglos.rb
errores-arreglos.rb:10:in `+': no implicit conversion of Integer
into String (TypeError)
    from errores-arreglos.rb:10:in `<main>'
C:\Ruby24-x64\programas>
```

Figura 11.1.-Compilación de la suma usando una cadena y un valor numérico.

Conclusión

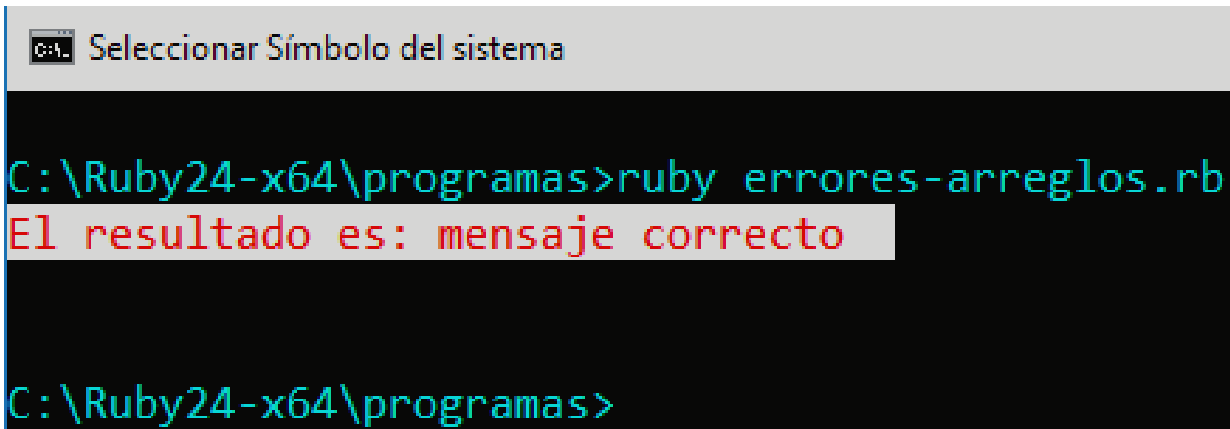
La impresión no se realiza por que en el código realizamos una suma de distintos tipos de datos, a continuación modificamos la impresión sumando dos valores de tipo string.

A screenshot of a code editor showing two lines of Ruby code. Line 10 is `puts "El resultado es: #{ "mensaje"+" correcto" }` and line 11 is `gets()`. The expression `"mensaje"+" correcto"` in line 10 is highlighted with a red rectangle.

```
10 puts "El resultado es: #{ "mensaje"+" correcto" }"
11 gets()
```

Figura 11.2.- suma de dos cadenas.

Compilación de la suma de dos cadenas:



A screenshot of a Windows command prompt window. The title bar at the top is light gray and contains the text "Seleccionar Símbolo del sistema" next to a small icon. The main area of the window has a black background. The text displayed is as follows: the first line is the command `C:\Ruby24-x64\programas>ruby errores-arreglos.rb` in green; the second line is the output `El resultado es: mensaje correcto` in red; and the third line is the prompt `C:\Ruby24-x64\programas>` in green.

```
C:\Ruby24-x64\programas>ruby errores-arreglos.rb
El resultado es: mensaje correcto

C:\Ruby24-x64\programas>
```

Figura 11.3.-Compilación de la suma usando dos cadenas.

Bibliografía.

- ✓ <https://www.ruby-lang.org/es/documentation/>
- ✓ <http://www.rubyist.net/~slagell/ruby/>
- ✓ <http://es.tldp.org/Manuales-LuCAS/doc-guia-usuario-ruby/guia-usuario-ruby.pdf>
- ✓ <http://eudev2.uta.cl/rid=1GR0DSG4D-1Y1NH87-4RQ/ruby.pdf>
- ✓ <https://nacherikazpu.wordpress.com/2016/04/14/tipos-de-errores-en-programacion/>