Universidad Nacional Autónoma de Nicaragua Unan León

Facultad de Ciencias y Tecnología.

Componente: Software como un servicio

Tema: Practica2

Grupo:1

Docente: Erving Montes

Integrante:

-Harold Steven Garcia Ramos.

Fecha: 16 de agosto de 2024

1. Array.

1.1. En el directorio Ruby, crear un programa en Ruby y asignar a un array los días de la semana, para luego imprimirlos por pantalla.

1.2. Ejecutar el programa en el terminal y analizar lo que imprime.

```
root@debian:/home/harold/ruby# ruby arrayDias.rb
lunes
martes
miercoles
jueves
viernes
sabado
domingo

Imprimir por posicion
martes
jueves
sabado
root@debian:/home/harold/ruby#
```

2. Método each

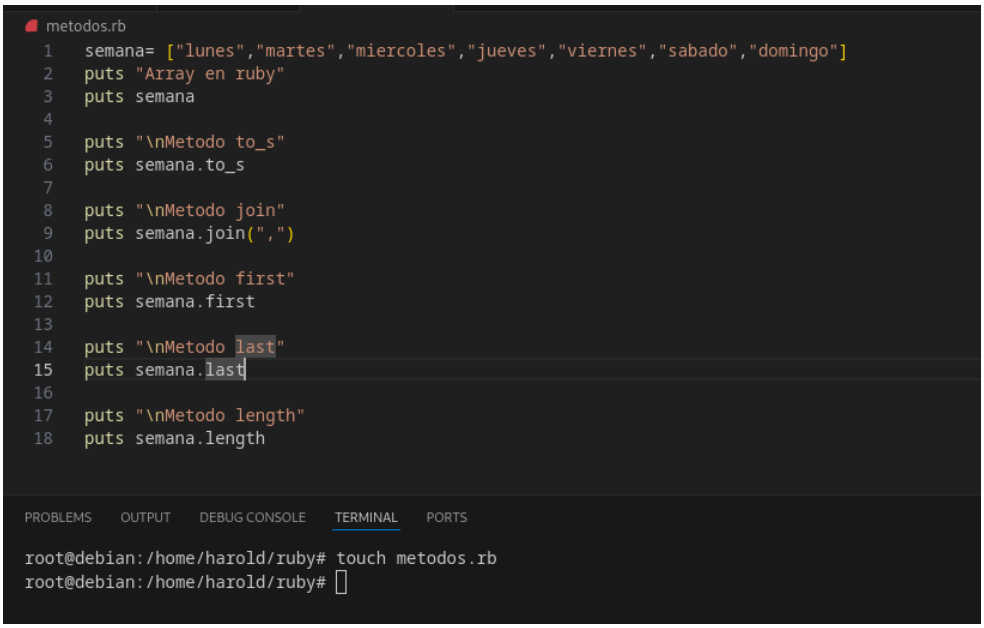
2.1. El método each en Ruby se utiliza como iterador para recorrer un array, tomando como ejemplo el programa anterior, crear uno nuevo y utilizar el método each para recorrer el array e imprimirlo por pantalla.

2.2. Ejecutar el programa y verificar su funcionamiento

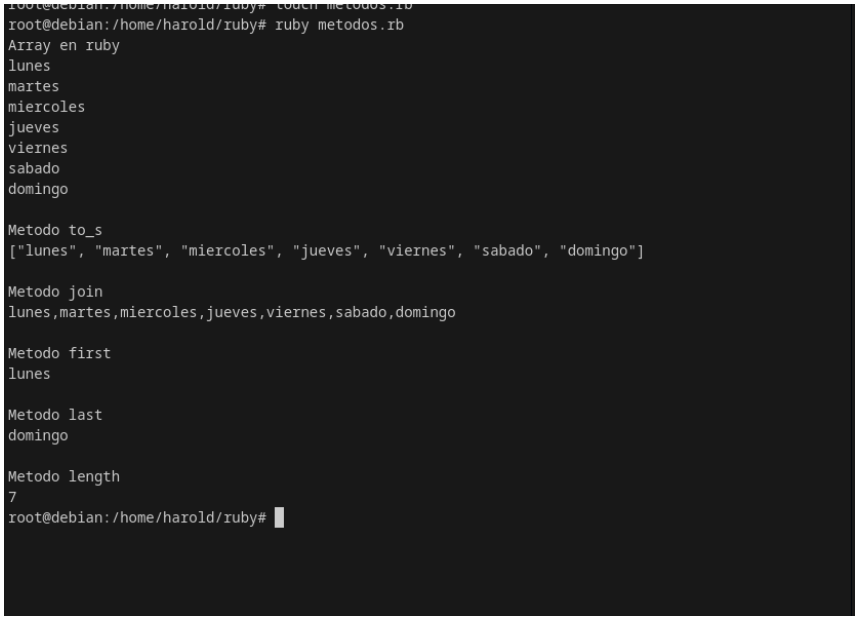
```
root@debian:/home/harold/ruby# ruby each.rb
dia 0=lunes
dia 1=martes
dia 2=miercoles
dia 3=jueves
dia 4=viernes
dia 5=sabado
dia 6=domingo
root@debian:/home/harold/ruby#
```

3. Métodos para trabajar con array En Ruby existen muchos métodos específicamente para trabajar con array, entre los cuales se pueden encontrar: pop, push, join, last, split. En este enunciado se mostrará el funcionamiento de algunos de ellos, los cuales son muy útiles en el desarrollo de aplicaciones en donde se trabaja con el lenguaje Ruby.

3.1. A continuación, se deberá realizar un programa en el que se utilicen algunos de los métodos antes mencionados.



3.2. Ejecutar el programa y verificar el funcionamiento, es importante ver cómo se comporta cada uno de los métodos con respecto al array.



3.3. Modificar el programa anterior, y hacer uso de los métodos push y pop para ver la diferencia del comportamiento entre ambos, en relación a su uso sobre los arrays.

metodos.rb

```ruby
semana= ["lunes","martes","miercoles","jueves","viernes","sabado","domingo"]

puts "\nMetodo to_s"
puts semana.to_s

puts "\nMetodo POP"
puts semana.pop

puts "\nMetodo length"
puts semana.length

puts "\nUltimo dato"
puts semana.last

puts "\nMetodo PUSH"
puts semana.push "final"

puts "\nTamaño nuevo"
puts semana.length
```

Al ejecutar el programa se observa que el método pop extrae el último dato dentro del array y lo elimina del conjunto, a diferencia del método push, que lo que hace es insertar el valor del dato indicado al final del conjunto.

```
root@debian:/home/harold/ruby# ruby metodos.rb

Metodo to_s
["lunes", "martes", "miercoles", "jueves", "viernes", "sabado", "domingo"]

Metodo POP
domingo

Metodo length
6

Ultimo dato
sabado

Metodo PUSH
lunes
martes
miercoles
jueves
viernes
sabado
final

Tamaño nuevo
7
root@debian:/home/harold/ruby#
```

4. Métodos propios 4.1En Ruby como en cualquier otro lenguaje de programación se pueden definir métodos para que realicen cierto trabajo, para entender un poco mejor de esto, crear un nuevo programa llamado metodos_propios.rb y agregar el siguiente código.

```ruby
def nombre
    nombre= gets.chomp

    if nombre.downcase
        nombre= nombre.upcase
    else
        nombre= nombre
    end
    puts "Bienvenido #{nombre}"
end

nombre
puts "Ingrese su año de nacimiento"
año_naciento= gets.chomp

puts "Ingrese el año actual"
año_actual= gets.chomp

edad(año_naciento, año_actual)
```

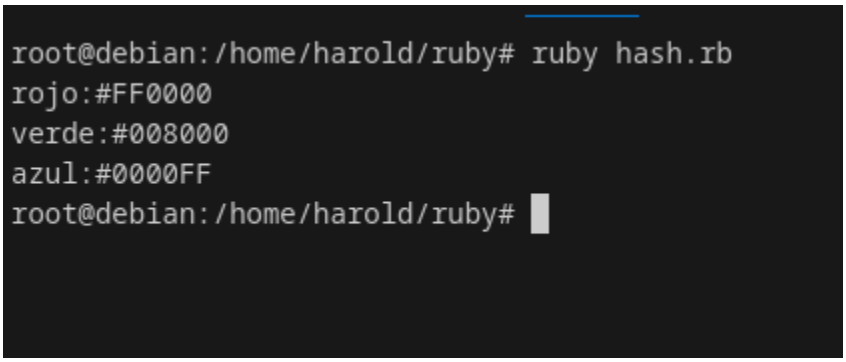
Como se puede observar se han definido dos métodos, uno llamado nombre que no recibe parámetros y el otro llamado edad, que recibe dos parámetros y que será el encargado de calcular la edad de una persona.

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

Ingrese su nombre
harold
Bienvenido HAROLD
Ingrese su año de nacimiento
2002
Ingrese el año actual
2024

Tu edad actual es 22 años
root@debian:/home/harold/ruby#
```

5. Hash 5.1. Algo muy utilizado en el lenguaje Ruby son los hashes al momento de trabajar con datos. Crear un programa llamado hash.rb y agregar el código a continuación.

```ruby
colorHash ={}

colorHash['rojo']='#FF0000'
colorHash['verde']='#008000'
colorHash['azul']='#0000FF'

colorHash.each do |tipoCodigo,color|
    puts tipoCodigo + ':' + color
end
```

5.2. Ejecutar el programa en el terminal para obtener la salida.



```
root@debian:/home/harold/ruby# ruby hash.rb
rojo:#FF0000
verde:#008000
azul:#0000FF
root@debian:/home/harold/ruby#
```

En los hashes, cada dato almacenado se guarda con un nombre que se conoce como una clave, esta clave puede ser textos o número y es por medio del cual se puede identificar cada uno de los datos que pertenecen al hash; como se puede observar en la figura 28, se imprime cada dato del hash por medio de su clave.

5.3. Para ver de otra manera el funcionamiento, crear un nuevo programa hash_2.rb y agregar el código.

```ruby
user= {}
user= {:name=> "Juan Pérez", :email=> "JuanP@example.com"}

puts "Nombre de usuario: #{user[:name]}"
puts "\nCorreo: #{user[:email]}"
```

Al ejecutar el programa se observa cómo se hace referencia a los datos del usuario, haciendo uso de la clave para poder mostrarlos por pantalla.

```
root@debian:/home/harold/ruby# ruby hash_2.rb
Nombre de usuario: Juan Pérez

Correo: JuanP@example.com
root@debian:/home/harold/ruby#
```

6. Clases

6.1. Crear un programa clases.rb, en el cual se creará una clase Palíndromo que contendrá un método para verificar una frase ingresada, como aparece a continuación.

```ruby
class Palindromo

    def verificar_frase(frase)

        if frase==frase.reverse
            puts "La frase #{frase} Es palíndromo"
        else
            puts "La frase #{frase} NO es palíndromo"
        end
    end
end

puts "Ingrese una frase"
frase= gets.chomp

verificar =Palindromo.new
verificar.verificar_frase(frase)
```

6.2. Ejecutar el programa e ingresar la palabra “level” para verificar su correcto funcionamiento.

```
root@debian:/home/harold/ruby# ruby clases.rb
Ingrese una frase
level
La frase level Es palíndromo
root@debian:/home/harold/ruby#
```

7. Variable de instancia.

7.1. Las variables de instancia son variables de un objeto, una de las diferencias de las variables locales es que estas existen hasta que el método ha terminado e inician con arroba “@”. Crear un programa en Ruby llamado variables.rb y escribir lo siguiente:

```
1   class Dado
7       def rodar
8           @numero_mostrar= 1+rand(6)
9       end
10
11      def mostrar
12          @numero_mostrar
13      end
14  end
15
16  puts "cuantas veces desea lanzar el dado"
17  lanzar= gets.chomp
18
19  while lanzar.to_i>0
20      lanza_dado= Dado.new.mostrar
21      puts "\nLanzamiento"
22      puts lanza_dado
23      lanzar= lanzar.to_i - 1
24  end
```

Como se observa en el código la variable numero_mostrar, se utiliza en los métodos rodar y mostrar, y siempre mantiene el mismo el valor.

7.2. Ejecutar el programa y verificar el funcionamiento de la variable numero_mostrar, la cual mantiene su valor en todos los métodos hasta ser mostrada por pantalla.

```
root@debian:/home/harold/ruby# ruby variables.rb
cuantas veces desea lanzar el dado
3

Lanzamiento
5

Lanzamiento
1

Lanzamiento
6
root@debian:/home/harold/ruby#
```

1. Realice cada uno de los enunciados de la guía, probar el funcionamiento y analizar cada uno de los programas planteados.

2. Crear un programa en Ruby que contenga un hash, el cual este compuesto de nombre = clave y celular = valor, el programa deberá mostrar el hash completo, solicitar el nombre que sería la clave y retornar el celular que sería el valor, correspondiente a ese nombre. Deberá validar si el dato existe en el hash y que cuando se ingrese un nombre en minúscula a como se muestra en la figura 32, el nombre Juan se ingresó en minúscula y el programa devuelve el celular correspondiente al nombre.

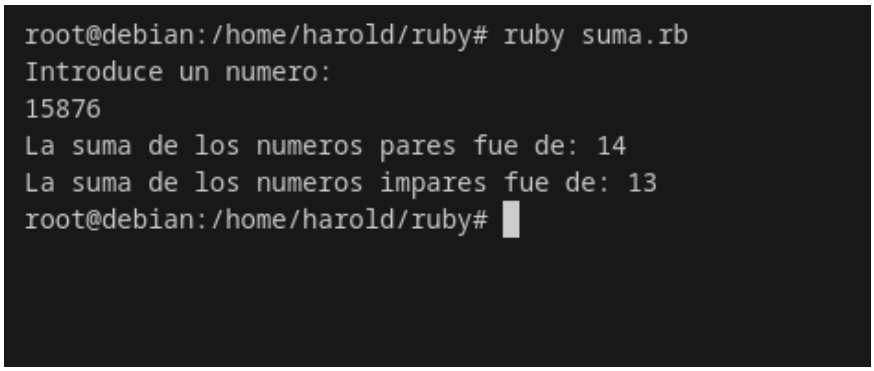
```ruby
datosHash ={}

datosHash ['Nombre']= 'Celular'
datosHash ['Maria']= '  2248-6559'
datosHash ['Pedro']= '  9845-6532'
datosHash ['Juan']= '   8265-4536'
datosHash ['Alberto']= '7896-4514'

datosHash.each do |nombre,celular|
    puts nombre + "        " + celular
end
puts "Para salir ingrese la letra q"
while true
    puts "\n--------------------"
    puts "Ingrese un nombre"
    nombre_ingresado= gets.chomp.capitalize
    break if nombre_ingresado.downcase == 'q'
    puts "--------------------"

    if datosHash.has_key?(nombre_ingresado)
        puts "El numero celular de #{nombre_ingresado} es #{datosHash[nombre_ingresado]}"
    else
        puts "Nombre no encontrado"
    end
end
puts "Programa Terminado"
```

```
root@debian:/home/harold/ruby# ruby propuesto2.rb
Nombre        Celular
Maria          2248-6559
Pedro          9845-6532
Juan           8265-4536
Alberto        7896-4514
Para salir ingrese la letra q

--------------------
Ingrese un nombre
juan
--------------------
El numero celular de Juan es    8265-4536

--------------------
Ingrese un nombre
alberto
--------------------
El numero celular de Alberto es 7896-4514

--------------------
Ingrese un nombre
q
Programa Terminado
root@debian:/home/harold/ruby#
```

3. Realice un programa en Ruby que solicite por pantalla un número cualquiera y que imprima la suma de los números pares e impares que componen el número ingresado, para la solución crear una clase de nombre Calcular la cual contendrá 2 métodos, el primer método para los cálculos de los números pares y el segundo método para los cálculos de los numero impares, se deberá mostrar a como se muestra.

ruby > suma.rb

```ruby
class Calcular

    def initialize(numero)
        @numero= numero
    end

    def suma_par
        sum=0
        @numero.to_s.each_char do |res|
            digito= res.to_i
            sum+= digito if digito.even?
        end
        sum
    end

    def suma_impar
        sum=0
        @numero.to_s.each_char do |res|
            digito= res.to_i
            sum+= digito if digito.odd?
        end
        sum
    end
end
puts "Introduce un numero: "
numero= gets.chomp.to_i

calculador= Calcular.new (numero)

puts "La suma de los numeros pares fue de: #{calculador.suma_par}"
puts "La suma de los numeros impares fue de: #{calculador.suma_impar}"
```



Extra: hacer que el programa número 1 que se ejecute varias veces hasta presionar una tecla.