Universidad Simón Bolívar
Departamento de Computación y Tecnología de la Información
CI3641 – Lenguajes de Programación I
Septiembre–Diciembre 2025 Examen 1

Examen 1 (25 puntos)

Estudiante Rafael Antonio Valera Pacheco 16-11202

Pregunta 1

Escoja algún lenguaje de programación de alto nivel y de propósito general cuyo nombre empiece con la misma letra que su nombre (por ejemplo, si su nombre es “Pedro”, podría escoger “Perl”, “PLI”, “Python”, etc.).

Se escogió el lenguaje conocido como Ruby.

(a) Dé una breve descripción del lenguaje escogido.
R: **Es un lenguaje de código abierto dinámico con un enfoque en la simplicidad y la productividad. Es muy conocido por su framework de desarrollo web, Ruby on Rails, su diseño busca que la programación sea más intuitiva y divertida para el desarrollador.**

i. Diga qué tipo de alcances y asociaciones posee, argumentando las ventajas y desventajas de la decisión tomada por los diseñadores del lenguaje, en el contexto de sus usuarios objetivos.
R: **El alcance de Ruby es de propósito general, pero se ha consolidado como una herramienta principal en el desarrollo web gracias a su framework más popular.**

Son muchas las asociaciones pero la principal sería con Ruby on Rails o RoR. RoR es un framework que sigue los principios de convención sobre configuración y no te repitas, lo que permite a los desarrolladores crear aplicaciones web complejas rápidamente por su prototipado rápido. Existen empresas como Airbnb y GitHub que lo utilizan.

Ventajas:

***Aumento de la productividad porque la sintaxis se asemeja al lenguaje natural.**

***Código elegante y modular.**

***Flexibilidad Extrema porque permite modificar el código en tiempo de ejecución.**

Desventajas:

***Menor velocidad de ejecución porque hace mucho trabajo extra en segundo plano para comodidad del usuario.**

***Alto consumo de memoria y rendimiento de arranque.**

***Menor relevancia fuera del desarrollo web.**

ii. Diga qué tipo de módulos ofrece (de tenerlos) y las diferentes formas de importar y exportar nombres.

R: **A diferencia de otros lenguajes, un módulo en Ruby cumple una doble función clave: espacio de nombres y Mixin**

Namespace: Evitar colisiones de nombres (ej. tener dos clases Logger diferentes).
Mixin:Reutilizar código y simular la herencia múltiple.

En Ruby, no existe un concepto de exportar como en JavaScript, ya que todo el código en un archivo (.rb) es ejecutado y sus nombres quedan disponibles según el ámbito (global, constante, local, etc.). El proceso se centra en importar o hacer accesibles los nombres.

La forma de hacer que el código de un archivo esté disponible en otro se hace mediante:

require: Carga y ejecuta un archivo fuente una única vez. Se utiliza para cargar librerías y dependencias.

load: Carga y ejecuta un archivo fuente cada vez que se llama. Se usa menos frecuentemente y es útil para archivos que pueden haber cambiado o que se necesita recargar.

Una vez que el archivo ha sido cargado, la forma de importar los nombres definidos en los módulos es a través de operadores y keywords como:

Espacios de nombres para acceder a cualquier constante (que incluye clases y otros módulos) definida dentro de un módulo, se usa el operador de alcance doble dos puntos.

Mecanismos Mixin que son los mecanismos centrales para importar el comportamiento (métodos) de un módulo a una clase, evitando la necesidad de herencia múltiple como include, extend y prepend.

iii. Diga si el lenguaje ofrece la posibilidad de crear aliases, sobrecarga y polimorfismo. En caso afirmativo, dé algunos ejemplos.
R: **Ruby permite crear alias para métodos**
Ejemplo de su uso:

```
class Perro
  def ladrar
    puts "Woof! Woof!"
  end

  # Crea un alias para 'ladrar' llamado 'bark'
  alias bark ladrar
end

corgi = Perro.new
corgi.ladrar  # Salida: Woof! Woof!
```

```
corgi.bark    # Salida: Woof! Woof! (Usando el alias)
```

Ruby no permite la sobrecarga de métodos tradicionales por que es un lenguaje de tipado dinámico aunque por alguna razón si puedes implementar sobrecarga de operadores

Ejemplo de sobrecarga con operadores:

```
class Vector
  attr_accessor :x, :y

  def initialize(x, y)
    @x = x
    @y = y
  end

  # Sobrecarga el operador '+'
  def +(otro_vector)
    # Define la nueva lógica para la suma: sumar componente a componente
    Vector.new(@x + otro_vector.x, @y + otro_vector.y)
  end

  def to_s
    "(#{@x}, #{@y})"
  end
end

v1 = Vector.new(2, 3)
v2 = Vector.new(5, 1)

v3 = v1 + v2 # Llama al método '+'
puts v3      # Salida: (7, 4)
```

El polimorfismo si se puede se logra principalmente a través de la sobrescritura de métodos
ejemplo uso:

```
class Animal
  def sonido
    "Un sonido genérico."
  end
end

class Gato < Animal
  # Sobrescribe el método 'sonido'
  def sonido
    "Miau"
  end
end
```

```ruby
class Perro < Animal
  # Sobrescribe el método 'sonido'
  def sonido
    "Guau"
  end
end

# Función polimórfica: no necesita saber el tipo de objeto.
def hacer_ruido(animal)
  puts animal.sonido
end

hacer_ruido(Gato.new)  # Salida: Miau
hacer_ruido(Perro.new) # Salida: Guau
```

iv. Diga qué herramientas ofrece a potenciales desarrolladores, como: compiladores, intérpretes, debuggers, profilers, frameworks, etc.
R: **Ruby ofrece un ecosistema robusto de herramientas enfocadas en la productividad y la experiencia del desarrollador entonces tendríamos:**
Compiladores:JRuby,JRuby,GraalVM y Opal.
Intérprete Principal: (MRI/CRuby).
Frameworks: Ruby on Rails,Sinatra,Grape, Hanami y Padrino.
Debuggers y Consolas:IRB (*Interactive Ruby*),Pry, Byebug / Debug, Web Console.
Profilers: Ruby's built-in Profiler / Benchmark, StackProf / rbspy.

(b) Implemente los siguientes programas en el lenguaje escogido:

1. Dada una cadena de caracteres w y un entero no–negativo k, calcular la rotación de k posiciones de la cadena w. Utilice la siguiente fórmula como referencia:

$$\text{rotar}(w,k) = \begin{cases} w & \text{si } k = 0 \vee |w| = 0 \\ \text{rotar}(x + +[a], k-1) & \text{si } k > 0 \wedge w = ax \wedge a \text{ es un caracter} \end{cases}$$

donde el operador ++ corresponde a la concatenación de cadenas de caracteres.

Ejemplo:

- rotar("hola", 0) = "hola"
- rotar("hola", 1) = .ºlah"
- rotar("hola", 2) = "laho"
- rotar("hola", 3) = .ªhol"
- rotar("hola", 4) = "hola"
- rotar("hola", 5) = .ºlah"

```ruby
# Definición de la función 'rotar'
def rotar(w, k)
```

```ruby
  # 1. Caso Base: si k = 0 o la cadena está vacía (|w| = 0)
  if k == 0 || w.empty?
    return w
  end

  # Ajuste para rotaciones mayores a la longitud de la cadena:
  k_efectivo = k % w.length

  if k_efectivo == 0
    return w
  end


  # 2. Caso Recursivo: si k > 0 y w = a + x (donde 'a' es el primer caracter)
  #    rotar(w, k) = rotar(x + [a], k - 1)


  a = w[0]

  x = w[1..-1]

  # Se forma la nueva cadena rotada un paso: x + [a]
  # Se hace la llamada recursiva: rotar(x + a, k_efectivo - 1)
  return rotar(x + a, k_efectivo - 1)
end

# --- Ejemplos del profe ---

puts "--- Pruebas de rotación ---"


puts "rotar('hola', 0) = #{rotar('hola', 0)}"

puts "rotar('hola', 1) = #{rotar('hola', 1)}"

puts "rotar('hola', 2) = #{rotar('hola', 2)}"

puts "rotar('hola', 3) = #{rotar('hola', 3)}"

puts "rotar('hola', 4) = #{rotar('hola', 4)}"

puts "rotar('hola', 5) = #{rotar('hola', 5)}"
puts "-----------------------------"
```

código en el repo de github

II. Dada una matriz cuadrada A (cuya dimensión es $N \times N$), calcular el producto $A \times A^T$ (donde A^T es la transpuesta de A).

La multiplicación de dos matrices cuadradas A y B (de tamaño $N \times N$) viene dada por:

$$\forall i, j \in [1..N] : (A \times B)_{i,j} = \sum_{k \in [1..N]} A_{i,k} \times B_{k,j}$$

y la transpuesta de A se define como:

$$\forall i, j \in [1..N] : A^T_{i,j} = A_{j,i}$$

```
# Función para calcular la transpuesta de una matriz cuadrada A.
def transponer(matriz_a)
  n = matriz_a.length
  # Inicializa la matriz transpuesta con ceros (o nil)
  matriz_at = Array.new(n) { Array.new(n) }

  # Llena la matriz transpuesta
  (0...n).each do |i|
    (0...n).each do |j|
      matriz_at[i][j] = matriz_a[j][i]
    end
  end

  return matriz_at
end

# Función para calcular el producto de dos matrices cuadradas A y B.
def multiplicar_matrices(matriz_a, matriz_b)
  n = matriz_a.length

  # Inicializa la matriz resultado con ceros
  matriz_resultado = Array.new(n) { Array.new(n, 0) }

  # Realiza la multiplicación de matrices
  (0...n).each do |i| # Filas de la matriz resultado
    (0...n).each do |j| # Columnas de la matriz resultado
      suma_producto = 0

      (0...n).each do |k|
        suma_producto += matriz_a[i][k] * matriz_b[k][j]
      end

      matriz_resultado[i][j] = suma_producto
    end
```

```
  end

  return matriz_resultado
end

# Función principal que calcula A * A^T
def producto_matriz_por_transpuesta(matriz_a)

  matriz_at = transponer(matriz_a)

  matriz_resultado = multiplicar_matrices(matriz_a, matriz_at)

  return matriz_resultado
end

# Función de ayuda para mostrar la matriz de forma clara
def imprimir_matriz(matriz)
  matriz.each do |fila|
    puts fila.inspect
  end
end

#              Ejemplo de como usarlo si quiere profe

# Matriz cuadrada A (dimensión N=2)
# A = [[1, 2],
#      [3, 4]]
matriz_a = [[1, 2], [3, 4]]

# Matriz Transpuesta A^T = [[1, 3],
#                           [2, 4]]

# Producto esperado A x A^T =
# [[(1*1 + 2*2), (1*3 + 2*4)],
#  [(3*1 + 4*2), (3*3 + 4*4)]]
# = [[(1 + 4), (3 + 8)],
#    [(3 + 8), (9 + 16)]]
# = [[5, 11],
#    [11, 25]]

puts "Matriz A:"
imprimir_matriz(matriz_a)

resultado = producto_matriz_por_transpuesta(matriz_a)

puts "\nResultado de A * A^T:"
imprimir_matriz(resultado)
```

código en el repo de github

2. Pregunta 2

Considere el siguiente programa escrito en pseudo-código:

```
int a = Y + Z + 1, b = X + Y + 1, c = Z + Y + 1;

sub R (int b) {
  a := b + c - 1
}

sub Q (int a, sub r) {
  b := a + 1
  r(c)
}

sub P(int a, sub s, sub t) {
  sub R(int a) {
    b := c + a + 1
  }
  sub Q (int b, sub r) {
    c := a + b
    r(c + a)
    t(c + b)
  }
  int c := a + b
  if (a < 2 * (Y + Z + 1)) {
    P(a + 2 * (Y + Z + 1), s, R)
  } else {
    int a := c + 1
    s(c * a, R)
    Q(c * b, t)
  }
  print(a, b, c)
}

P(a, Q, R);
print(a, b, c)
```

Diga qué imprime el programa en cuestión, si el lenguaje tiene:
(a) Alcance estático y asociación profunda:
Mi carnet es 1611202 por lo cual mis numeros serian X=2 Y=0 Z=2

entonces

Paso 1 Estado inicial antes de la llamada P(a, Q, R)

GLOBAL
X = 2
Y = 0
Z = 2
a = 3 (Y+Z+1 = 3)
b = 3 (X+Y+1 = 3)
c = 3 (Z+Y+1 = 3)

| Sub R(b): a := b + c - 1 |
| Sub Q(a, r): b := a + 1; r(c) |
| Sub P(a, s, t): ... |
|_____|

Ahora se ejecuta
P(a, Q, R)
→ P(3, Q, R)

Paso 2 Activación de P_1(3, Q, R)

P_1
a = 3
s = Q (closure con env GLOBAL)
t = R (closure con env GLOBAL)
Sub R_1(a): b := c + a + 1
Sub Q_1(b, r): c := a + b; r(c+a); t(c+b)
c = a + b = 3 + 3 = 6

Verificación del if dentro de P:
if (a < 2*(Y+Z+1))
=> if (3 < 2*(0+2+1)) => if (3 < 6) = true

Por tanto, entra en la rama then:
P(a + 2*(Y+Z+1), s, R_1)
→ P(3 + 6, s, R_1)
→ P_2(9, s, R_1)

Paso 3 Activación de P_2(9, s=Q, t=R_1)

P_2
a = 9
s = Q (closure con env P_1)
t = R_1 (closure con env P_1)
Sub R_2(a): b := c + a + 1
Sub Q_2(b, r): c := a + b; r(c+a); t(c+b)
c = a + b = 9 + 3 = 12

Evaluación del if:
if (a < 2*(Y+Z+1))
=> if (9 < 6) = false

Paso 4 Rama else dentro de P_2
Dentro del else:
a_local = c + 1 = 12 + 1 = 13

Luego ejecuta:
s(c * a_local, R_2)
→ s(12 * 13, R_2)
→ s(156, R_2)

El s fue pasado a P_2 como Q (top-level), con ambiente GLOBAL

paso 5 Ejecución de s = Q_top(156, R_2)
Ambiente léxico: global.

```
_____________________________
|   Llamada Q_top            |
|-----------------------     -|
| a = 156                     |
| r = R2 (closure con env P2)  |
|-------------------------    -|
| cuerpo:                      |
| b := a + 1                   |
| r(c)                         |
|-----------------------      -|
```

Paso 6 Ejecución de R_2(3)
R_2 fue definida dentro de P_2.
Por alcance estático, en R_2:
b := c + a + 1
Aquí:
c se refiere al c de P_2 (12)
a es el parámetro de R_2 (3)
Entonces:
b := 12 + 3 + 1 = 16
→ global b = 16
Paso 7 Retorno a P_2, continúa el else
Después del s(...), ejecuta:
Q(c * b, t)
→ Q_2(12 * 16, t)
→ Q_2(192, t)
Paso 8 Ejecución de Q_2(192, t)
Q_2 fue definida dentro de P_2.

```
_____________________________
|   Llamada Q2               |
|-------------------------    |
| b = 192                     |
| r = t = R1(with env P1)     |
|-------------------------    |
| cuerpo:                     |
| c := a + b                  |
| r(c + a)                    |
| t(c + b)                    |
|-------------------------    |
```

Paso 9 Ejecución de R_1
R_1 fue definida dentro de P_1:
R_1(a): b := c + a + 1
Y su ambiente estático es el de P_1 (P_1.c = 6).
Primera llamada: R_1(210)

global b := 6 + 210 + 1 = 217

Segunda llamada: R_1(393)

global b := 6 + 393 + 1 = 400

Paso 10 Fin de Q_2 y del else de P_2
Después del else, P_2 imprime:
print(a, b, c)

→ (a=9, b=400, c=201)

Primera salida:
(9, 400, 201)

Paso 11 Retorno a P_1

P_1 continúa después de la llamada recursiva y ejecuta su propio print(a, b, c).
Valores:
a = 3
b = 400 (global)
c = 6
Segunda salida:
(3, 400, 6)

Paso 12 Retorno al nivel global

Finalmente, el programa imprime los valores globales:
print(a, b, c)
→ (3, 400, 3)
Tercera salida:
(3, 400, 3)

(b) Alcance dinámico y asociación profunda
para este solo pondremos como se ven los estados

Paso 1 Estado inicial (global)

```
_____________________________
|       GLOBAL              |
|---------------------      |
| X = 2                     |
| Y = 0                     |
| Z = 2                     |
| a = 3                      |
| b = 3                      |
| c = 3                      |
|---------------------     -|
| Sub R(b): a := b + c - 1   |
| Sub Q(a, r): b := a + 1; r(c)  |
| Sub P(a, s, t): ...        |
|____________________________|
```

Paso 2 Se crea el marco P_1(3, Q, R)

```
_____________________________
|       P1                          |
|---------------------              |
| a = 3                             |
| s = Q (closure con env GLOBAL) |
| t = R (closure con env GLOBAL)   |
|---------------------              |
| Sub R1(a): b := c + a + 1          |
| Sub Q1(b, r): c := a + b; r(c+a); t(c+b) |
|-----------------------------      |
| c = a + b = 3 + 3 = 6             |
|-----------------------------     |
```

Paso 3 Se crea el marco P_2(9, s, R_1)

```
_____________________________
|       P2                            |
|---------------------                |
| a = 9                               |
| s = Q (closure con env P1)          |
| t = R1 (closure con env P1)         |
|---------------------                |
```

| Sub R_2(a): b := c + a + 1 |
Sub Q_2(b, r): c := a + b; r(c+a); t(c+b)
c = a + b = 9 + 3 = 12

Paso 4 Llamada a s = Q_top(156, R_2)

Q_top
a = 156
r = R_2 (closure con env P_2)
cuerpo:
b := a + 1
r(c)

Paso 5 Llamada a R_2(3)

R_2
a = 3
b := c + a + 1

Paso 6 Llamada a Q_2(192, t)

Q_2
b = 192
r = t = R_1 (closure con env P_1)
cuerpo:
c := a + b
r(c + a)
t(c + b)

Paso 7 Llamadas a R_1(210) y R_1(393)

R_1
a = 210 / 393
b := c + a + 1

Paso 8 Retorno al global

Imprime:
(a=3, b=400, c=3)

Tercera salida:
(3, 400, 3)

(c) Alcance estático y asociación superficial

Paso 1 Estado inicial

```
_____________________________
|        GLOBAL                |
|--------------------------    |
| X = 2                        |
| Y = 0                        |
| Z = 2                        |
| a = 3                        |
| b = 3                        |
| c = 3                        |
|--------------------------    |
| Sub R(b): a := b + c - 1     |
| Sub Q(a, r): b := a + 1; r(c)|
| Sub P(a, s, t): ...          |
|______________________________|
```

Paso 2 Creación del marco P_1(3, Q, R)

```
_____________________________
|        P1                    |
|--------------------------    |
| a = 3                        |
| s = Q (procedimiento con binding superficial) |
| t = R                        |
|--------------------------   -|
| Sub R1(a): b := c + a + 1    |
| Sub Q1(b, r): c := a + b; r(c+a); t(c+b) |
|-------------------------------- |
| c := a + b = 3 + 3 = 6       |
|-------------------------------- |
```

Paso 3 Creación del marco P_2(9, s, R_1)

```
_____________________________
|        P2                    |
|--------------------------    |
| a = 9                        |
| s = Q                        |
| t = R1                       |
|--------------------------    |
| Sub R2(a): b := c + a + 1    |
| Sub Q2(b, r): c := a + b; r(c+a); t(c+b) |
|-------------------------------- |
| c := a + b = 9 + 3 = 12      |
|-------------------------------- |
```

Paso 4 Llamada a s = Q(156, R_2)

Con asociación superficial, el ambiente que usará Q será el del invocador, es decir, P_2, no el global.

```
_____________________
|   Q_shallow          |
|----------------------|
| a = 156              |
| r = R2               |
|----------------------|
| cuerpo:              |
| b := a + 1           |
| r(c)                 |
|----------------------|
```

Paso 5 Llamada a R_2(12)

R_2 fue definido dentro de P_2, así que su entorno léxico es P_2.

```
_____________________
|   R2                 |
|----------------------|
```

a = 12
b := c + a + 1

Paso 6 Llamada a Q_shallow(300, t)
Como antes, el ambiente visible es P_2 (por ser asociación superficial).

$Q\_shallow_2$
a = 300
r = t = R_1
cuerpo:
b := a + 1
r(c)

Paso 7 Llamada a R_1(12)
R_1 fue definido dentro de P_1, así que su ambiente léxico es P_1.

R_1
a = 12
b := c + a + 1

Paso 8 Regreso al global
Imprime:
(a=3, b=19, c=3)

Tercera impresión:
(3, 19, 3)
Resultado final (Estático + Asociación superficial)
(9, 301, 12)
(3, 19, 6)
(3, 19, 3)

(d) Alcance dinámico y asociación superficial

Paso 1 Estado inicial (global)

GLOBAL
X = 2
Y = 0
Z = 2
a = 3
b = 3
c = 3
Sub R(b): a := b + c - 1
Sub Q(a, r): b := a + 1; r(c)
Sub P(a, s, t): ...

Paso 2 Se crea el marco P_1(3, Q, R)

P_1
a = 3
s = Q (por referencia)

t = R
Sub R_1(a): b := c + a + 1
Sub Q_1(b, r): c := a + b; r(c+a); t(c+b)
c := a + b = 3 + 3 = 6

Paso 3 Se crea el marco P_2(9, s, R_1)

P_2
a = 9
s = Q (por referencia)
t = R_1
Sub R_2(a): b := c + a + 1
Sub Q_2(b, r): c := a + b; r(c+a); t(c+b)
c := a + b = 9 + 3 = 12

Paso 4 Llamada a Q(156, R_2)

Con alcance dinámico + asociación superficial:
El ambiente visible para Q es el del invocador actual, o sea P_2.
Q accederá a b, c, y r buscando desde P_2 hacia arriba.

Q_dyn_shallow
a = 156
r = R_2
cuerpo:
b := a + 1
r(c)

Paso 5 Nueva llamada a Q(300, t)

Otra vez con alcance dinámico + superficial, ambiente visible es P_2.

Q_dyn_shallow$_2$
a = 300
r = t = R_1
cuerpo:
b := a + 1
r(c)

Paso 6 Regreso al global

Imprime:
(a=3, b=25, c=3)

Tercera salida:
(3, 25, 3)
Resultado final
Alcance dinámico + Asociación superficial
(9, 25, 12)
(3, 25, 6)
(3, 25, 3)

3. Pregunta 3

Se desea que modele e implemente, en el lenguaje de su elección, un programa que simule un manejador de memoria que implemente el *buddy system*. Este programa debe cumplir con las siguientes características:

(a) Al ser invocado, recibirá como argumento la cantidad de bloques de memoria que manejará.

(b) El programa pedirá repetidamente una acción al usuario, que puede ser:

I. **RESERVAR** <*cantidad*><*nombre*>

II. **LIBERAR** <*nombre*>

III. **MOSTRAR**

IV. **SALIR**

Cada acción debe ser validada con mensajes de error claros e informativos. Además, debe incluir pruebas unitarias y alcanzar una cobertura de al menos 80 %.

```
require_relative 'buddy_system'

puts "Ingrese el tamaño total de la memoria (potencia de 2):"
tam = gets.to_i
sistema = BuddySystem.new(tam)

loop do
  puts "Acción: RESERVAR <cant> <nombre> | LIBERAR <nombre> | MOSTRAR | SALIR"
  print "> "
  entrada = gets.chomp.split
  comando = entrada[0]&.upcase

  begin
    case comando
    when "RESERVAR"
      cant = entrada[1].to_i
      nombre = entrada[2]
      sistema.reservar(cant, nombre)
    when "LIBERAR"
      nombre = entrada[1]
      sistema.liberar(nombre)
    when "MOSTRAR"
      sistema.mostrar
    when "SALIR"
      puts "Finalizando simulación..."
      break
    else
      puts "Comando no válido."
    end
  rescue => e
    puts "Error: #{e.message}"
  end
end
```

Este programa se hizo de forma modular y este código sería solo el main pero falta buddy_system que es lo que usa para funcionar, este programa al igual que el test con cobertura del 80% están en el repositorio de github, como dato adicional si quiere correr el test debe instalar algo adicional con el comando: gem install simplecov . De igual manera está todo organizado en el github en una carpeta

4. Pregunta 4

Implemente un módulo que defina el tipo de vectores tridimensionales y operadores aritméticos sobre estos, cumpliendo con las siguientes características:

- Soporte operaciones: suma (+), resta (-), producto cruz (*), producto punto (%), y norma (&).
- Permita expresiones naturales como:

```
b + c
a * b + c
(b + b) * (c - a)
a % (c * b)
```

- Soporte operaciones con escalares por la derecha:

```
b + 3
a * 3.0 + &b
(b + b) * (c % a)
```

donde $(x, y, z) \oplus n = (x \oplus n, y \oplus n, z \oplus n)$.

Incluya pruebas unitarias y cobertura mayor al 80 %.

```ruby
# vector3d.rb
class Vector3D
  attr_accessor :x, :y, :z

  def initialize(x, y, z)
    @x = x.to_f
    @y = y.to_f
    @z = z.to_f
  end

  # Suma
  def +(other)
    if other.is_a?(Vector3D)
      Vector3D.new(@x + other.x, @y + other.y, @z + other.z)
    else
      Vector3D.new(@x + other, @y + other, @z + other)
    end
  end
```

```
  # Resta
  def -(other)
    if other.is_a?(Vector3D)
      Vector3D.new(@x - other.x, @y - other.y, @z - other.z)
    else
      Vector3D.new(@x - other, @y - other, @z - other)
    end
  end

  # Producto cruz
  def *(other)
    if other.is_a?(Vector3D)
      Vector3D.new(
        @y * other.z - @z * other.y,
        @z * other.x - @x * other.z,
        @x * other.y - @y * other.x
      )
    else
      Vector3D.new(@x * other, @y * other, @z * other)
    end
  end

  # Producto punto
  def %(other)
    raise "Producto punto requiere un vector" unless other.is_a?(Vector3D)
    @x * other.x + @y * other.y + @z * other.z
  end

  # Norma
  def norm
    Math.sqrt(@x**2 + @y**2 + @z**2)
  end

  def to_s
    "(#{@x}, #{@y}, #{@z})"
  end

  def ==(other)
    return false unless other.is_a?(Vector3D)
    @x == other.x && @y == other.y && @z == other.z
  end
end
```

Al igual que la pregunta 3 este programa se hizo de forma modular, esto que ve aqui es la logica del codigo, para que luego el main lo llame y realice las operaciones, estará todo en una carpeta en el github

5. Pregunta 5

Modele e implemente un programa que simule programas, intérpretes y traductores, como en los diagramas de T. El programa debe permitir definir:

- **PROGRAMA <nombre><lenguaje>**
- **INTERPRETE <lenguaje_base><lenguaje>**
- **TRADUCTOR <lenguaje_base><lenguaje_origen><lenguaje_destino>**

Debe manejar el lenguaje especial `LOCAL`, correspondiente al lenguaje ejecutable por la máquina local.

Debe permitir acciones del usuario:

- **DEFINIR <tipo>[<argumentos>]**
- **EJECUTABLE <nombre>**
- **SALIR**

Incluya ejemplos, validaciones de error y pruebas unitarias con cobertura mayor al 80 %.

```
# main_sistema.rb
require_relative 'sistema'

s = Sistema.new

puts "Bienvenido al simulador de programas/intérpretes/traductores."
loop do
  print "> "
  input = gets.chomp.strip
  break if input.upcase == "SALIR"
  next if input.empty?

  comando, *args = input.split
  begin
    case comando.upcase
    when "DEFINIR"
      tipo = args.shift
      s.definir(tipo, *args)
    when "EJECUTABLE"
      nombre = args.first
      s.ejecutable(nombre)
    else
      puts "Comando desconocido: #{comando}"
    end
  rescue => e
    puts "Error: #{e.message}"
```

```
  end
end

puts "Saliendo..."
```

Este es el main del programa, es modular, todo lo que necesita el programa para su ejecución estará en el github.

Reto Extra: Golf

Considere los números de Narayana:

$$N_{n,k} = \frac{1}{n}\binom{n}{k}\binom{n}{k-1}$$

y la función Fibonacci:

$$\text{fib}(n) = \begin{cases} n & \text{si } 0 \leq n < 2 \\ \text{fib}(n-1) + \text{fib}(n-2) & \text{si } n \geq 2 \end{cases}$$

Así como el logaritmo en base 2:

$$\log_2(n) = m \Leftrightarrow 2^m = n$$

Definimos:

$$\text{wadefoc}(n) = \text{fib}\left(\lfloor \log_2(N_{n+1,n-1}) \rfloor + 1\right)$$

Desarrolle un programa que reciba un argumento $n \geq 2$ y calcule wadefoc(n), con tiempo de ejecución menor a 1 segundo hasta $n = 100$.

Reglas del reto: minimice la cantidad de caracteres de su programa. Los ganadores recibirán puntos extra (5, 3 y 1 respectivamente).

f=->n{n<2?n:f[n-1]+f[n-2]};n=gets.to_i;p f[Math.log2(n*n*(n*n-1)/12.0).floor+1]