Universidad de Cundinamarca

Estructura de Datos

R1-A4-S5 Vacuna

Javier Mateo Barrero Vanegas

Jim Alejandro Quiñones Martínez

Luis Ángel Martínez Cuenca

Harick Yesid Villarraga Rincon

Cristian Esteban Ruiz Parra

Docente: Dilia Ines Molina Cubillos

28 de August de 2025

Introducción

Este documento presenta una investigación detallada sobre la sintaxis y el manejo de funciones en lenguajes de programación clave como C++, C#, Python y Java. Además del análisis teórico, se incluye un anexo que aplica estos conceptos en la creación de un programa simulador de fabricación de vacunas en Python. El objetivo es demostrar cómo la modularización del código a través de funciones permite resolver problemas complejos de manera colaborativa y eficiente.

Funciones en C++

1.1. Características Generales

- **Declaración:** Las funciones en C++ se declaran especificando el tipo de retorno, nombre y parámetros (opcionales). La sintaxis básica es: tipoRetorno nombreFuncion(parámetros) { // cuerpo }.

- **Funciones sin parámetros:** Usan paréntesis vacíos ().

  - **Funciones sin retorno:** Se declaran con void. Ejemplo:

```
1   void imprimirMensaje() {
2       cout << "Hola Mundo";
3   }
```

- **Retorno de valores:** Se utiliza la palabra clave return.

1.2. Funciones Recursivas

- **Definición:** Una función recursiva se llama a sí misma dentro de su cuerpo. Es crucial definir una **condición base** para evitar recursión infinita.

- **Ejemplo (factorial):**

```
int factorial(int n) {
    if (n == 0) return 1; // Condición base
    return n * factorial(n - 1); // Llamada recursiva
}

```

1.3. Funciones con Parámetros

- **Paso por valor y referencia:**
    - **Por valor:** Se pasa una copia del argumento. Los cambios no afectan al original.
    - **Por referencia:** Se pasa la dirección del argumento. Los cambios persisten. Ejemplo:

```
void duplicar(int &x) {
    x *= 2;
}

```

- **Sobrecarga de funciones:** C++ permite múltiples funciones con el mismo nombre pero diferentes parámetros.

1.4. Características Avanzadas

- **Prototipos de funciones:** Permiten declarar una función antes de su definición para ser usada en el código.
- **Funciones de biblioteca:** Predefinidas en C++ (e.g., sqrt() en <cmath>).

2. Funciones en C#

2.1. Características Generales

- **Sintaxis:** Similar a C++. Las funciones se definen dentro de clases.
- **Modificadores de acceso:** Como public, private, que controlan la visibilidad.
- **Funciones sin retorno:** Usan void. Ejemplo:

```
1   public void Imprimir() {
2       Console.WriteLine("Hola");
3   }
4
```

2.2. Funciones Recursivas

- **Ejemplo (factorial):**

```
1   public int Factorial(int n) {
2       if (n == 0) return 1;
3       return n * Factorial(n - 1);
4   }
5
```

2.3. Funciones con Parámetros

- **Parámetros opcionales y nombrados:** Permiten omitir argumentos o especificarlos por nombre.

- **Modificadores ref y out:**
    - **ref:** Paso por referencia (la variable debe inicializarse primero).
    - **out:** Similar a ref, pero la variable no necesita inicialización.

2.4. Características Avanzadas

- **Funciones locales:** Funciones definidas dentro de otro método, útiles para encapsular lógica específica.
- **Expresiones lambda:** Permiten definir funciones anónimas.

Ejemplo:

```
1   Func<int, int> square = x => x * x;
```

3. Funciones en Python

3.1. Características Generales

- **Declaración:** Se usa def seguido del nombre y parámetros. No se especifica tipo de retorno.
- **Retorno de valores:** Opcional con return. Si no se especifica, retorna None.

- **Ejemplo:**

```
def greet():
    print("Hello World")

```

3.2. Funciones Recursivas

- **Ejemplo (factorial):**

```
def factorial(n):
    if n == 0:
        return 1
    return n * factorial(n - 1)

```

3.3. Funciones con Parámetros

- **Flexibilidad en parámetros:**
    - **Args arbitrarios (*args):** Permite pasar un número variable de argumentos posicionales.
    - **Kwargs (**kwargs):** Permite pasar argumentos de palabra clave arbitrarios.
- **Parámetros por defecto:** Se asignan valores predeterminados.

Ejemplo:

```
def greet(name, message="Hello"):
    print(f"{message}, {name}")
```

3.4. Características Avanzadas

- **Funciones como ciudadanos de primera clase:** Las funciones pueden asignarse a variables, pasarse como argumentos y retornarse desde otras funciones.

- **Funciones anidadas:** Python permite definir funciones dentro de otras funciones.

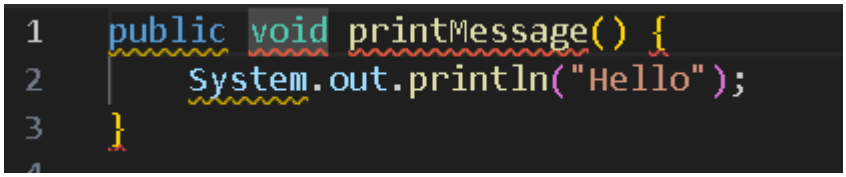
4. Funciones en Java

4.1. Características Generales

- **Sintaxis:** Las funciones se definen dentro de clases. Sintaxis: modificador tipoRetorno nombreFuncion(parámetros) { ... }.

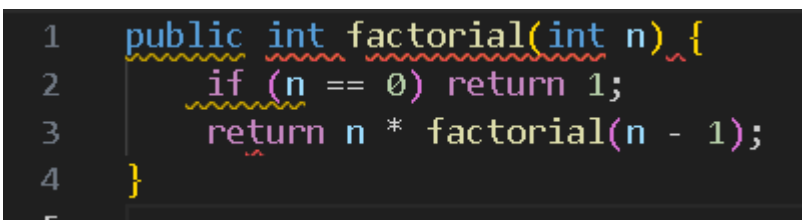
- **Modificadores de acceso:** Como public, private, protected.

- **Funciones sin retorno:** Usan void. Ejemplo:



4.2. Funciones Recursivas

- **Ejemplo (factorial):**



4.3. Funciones con Parámetros

- **Paso por valor:** Java siempre pasa argumentos por valor. Para objetos, se pasa por valor la referencia al objeto.

- **Sobrecarga de métodos:** Permite múltiples métodos con el mismo nombre pero diferente firma (número o tipo de parámetros).

4.4. Características Avanzadas

- **Funciones lambda:** Introducidas en Java 8, permiten implementar interfaces funcionales de manera concisa. Ejemplo:

```
1   Function<Integer, Integer> square = x -> x * x;
2
```

- **Métodos estáticos:** Pertenecen a la clase en lugar de instancias, se llaman usando el nombre de la clase.

Marco teórico

El código del simulador de vacuna es un excelente ejemplo de cómo organizar un programa complejo de manera modular, utilizando funciones para cada una de las tareas del proceso de fabricación. El programa se puede entender como una cadena de producción virtual que sigue un flujo lógico.

El Flujo de Fabricación

Todo el proceso es orquestado por la **función principal**, ejecutar_proceso_completo(). Esta función coordina cada paso, desde la preparación de los ingredientes hasta la verificación final, manejando posibles errores en el camino.

1. **Preparación de ingredientes:** El proceso comienza con mostrar_mensaje_inicio(), que da la bienvenida al usuario. Luego, se llaman secuencialmente seis funciones, preparar_ingrediente_1() a preparar_ingrediente_6(). Cada una de estas funciones tiene una tarea específica: asignar un valor numérico (la cantidad en miligramos) a una de las **variables globales** del programa. El uso de la palabra clave global es clave aquí, ya que permite a estas funciones modificar variables que están fuera de su alcance local, como si estuvieran dejando un ingrediente en una mesa compartida.

2. **Verificación y cálculo:** Una vez que se han preparado todos los ingredientes, la función revisar_todos_los_ingredientes() es llamada para confirmar que las cantidades de todos los ingredientes son mayores que cero. Si falta alguno, el programa se detiene para evitar errores. Si la verificación es exitosa, se calcula la cantidad total de ingredientes utilizando calcular_cantidad_total().

3. **Mezclado de componentes:** El proceso de mezclado se divide en dos etapas. Primero, hacer_primera_mezcla() combina los primeros tres ingredientes y devuelve la cantidad resultante. Luego, hacer_segunda_mezcla() hace lo mismo con los últimos tres. Estas dos funciones devuelven valores, que son recibidos como **parámetros** por la función combinar_las_dos_mezclas().

4. **Control de calidad y finalización:** La mezcla final se pasa a verificar_calidad_vacuna(). Esta función toma la cantidad total de la vacuna como parámetro para determinar si cumple con los estándares de calidad (una cantidad entre 15 y 25 mg). Basándose en el resultado de esta función, el programa decide si la fabricación ha sido exitosa o si ha fallado. Si el control de calidad es superado, se ejecuta la función terminar_fabricacion() para marcar la vacuna como terminada.

Código

```
# SIMULADOR DE FABRICACION DE VACUNA
# Objetivo: Crear una vacuna con 6 ingredientes usando 15 funciones

# Variables globales que almacenan las cantidades de cada ingrediente
# Estas variables se pueden usar en todas las funciones del programa
ingrediente1 = 0    # Antigeno - componente principal de la vacuna
ingrediente2 = 0    # Adyuvante - ayuda al sistema inmune a responder mejor
ingrediente3 = 0    # Estabilizante - mantiene la vacuna en buen estado
ingrediente4 = 0    # Conservante - evita que se dañe la vacuna
ingrediente5 = 0    # Buffer - mantiene el nivel de acidez correcto
ingrediente6 = 0    # Excipiente - material de relleno para la vacuna

# Variable que indica si la vacuna esta completamente lista
vacuna_terminada = False

# FUNCION 1: Mostrar el mensaje de inicio del programa
def mostrar_mensaje_inicio():
    """
    Esta funcion muestra el titulo del programa y explica el objetivo
    No recibe parametros y no devuelve nada
    """
    print("LABORATORIO DE FABRICACION DE VACUNAS")
    print("====================================")
```

```
    print("Objetivo: Fabricar una vacuna para salvar a la humanidad")
    print("Necesitamos preparar 6 ingredientes diferentes")
    print("")

# FUNCION 2: Preparar el primer ingrediente (Antigeno)
def preparar_ingrediente_1():
    """
    Prepara el antigeno que es el componente mas importante de la vacuna
    El antigeno es lo que enseña al cuerpo a defenderse de la enfermedad
    Modifica la variable global ingrediente1
    Devuelve la cantidad preparada
    """
    global ingrediente1  # Usamos la palabra 'global' para modificar la variable
    print("Preparando ingrediente 1: Antigeno")
    print("El antigeno es el componente principal de la vacuna")
    ingrediente1 = 8.5   # Asignamos 8.5 miligramos de antigeno
    print("Antigeno preparado correctamente: " + str(ingrediente1) + " mg")
    return ingrediente1

# FUNCION 3: Preparar el segundo ingrediente (Adyuvante)
def preparar_ingrediente_2():
    """
    Prepara el adyuvante que ayuda a que el sistema inmune responda mejor
    Sin el adyuvante, la vacuna no seria tan efectiva
    Modifica la variable global ingrediente2
    Devuelve la cantidad preparada
```

```
    """
    global ingrediente2
    print("Preparando ingrediente 2: Adyuvante")
    print("El adyuvante potencia la respuesta del sistema inmune")
    ingrediente2 = 2.0   # Asignamos 2.0 miligramos de adyuvante
    print("Adyuvante preparado correctamente: " + str(ingrediente2) + " mg")
    return ingrediente2

# FUNCION 4: Preparar el tercer ingrediente (Estabilizante)
def preparar_ingrediente_3():
    """
    Prepara el estabilizante que mantiene la vacuna en buenas condiciones
    Evita que los otros ingredientes se descompongan
    Modifica la variable global ingrediente
    Devuelve la cantidad preparada
    """
    global ingrediente3
    print("Preparando ingrediente 3: Estabilizante")
    print("El estabilizante mantiene la vacuna en buen estado")
    ingrediente3 = 1.5   # Asignamos 1.5 miligramos de estabilizante
    print("Estabilizante preparado correctamente: " + str(ingrediente3) + " mg")
    return ingrediente3

# FUNCION 5: Preparar el cuarto ingrediente (Conservante)
def preparar_ingrediente_4():
    """
```

```
    Prepara el conservante que evita la contaminacion de la vacuna
    Es muy importante para que la vacuna sea segura
    Modifica la variable global ingrediente4
    Devuelve la cantidad preparada
    """
    global ingrediente4
    print("Preparando ingrediente 4: Conservante")
    print("El conservante evita que la vacuna se contamine")
    ingrediente4 = 0.3   # Asignamos 0.3 miligramos de conservante
    print("Conservante preparado correctamente: " + str(ingrediente4) + " mg")
    return ingrediente4

# FUNCION 6: Preparar el quinto ingrediente (Buffer)
def preparar_ingrediente_5():
    """
    Prepara el buffer que mantiene el pH (nivel de acidez) correcto
    Si el pH no es correcto, la vacuna no funcionara bien
    Modifica la variable global ingrediente5
    Devuelve la cantidad preparada
    """
    global ingrediente5
    print("Preparando ingrediente 5: Buffer")
    print("El buffer mantiene el nivel de acidez correcto")
    ingrediente5 = 2.8   # Asignamos 2.8 miligramos de buffer
    print("Buffer preparado correctamente: " + str(ingrediente5) + " mg")
    return ingrediente5
```

```
# FUNCION 7: Preparar el sexto ingrediente (Excipiente)
def preparar_ingrediente_6():
    """
    Prepara el excipiente que sirve como material de relleno
    Ayuda a que la vacuna tenga el volumen adecuado
    Modifica la variable global ingrediente6
    Devuelve la cantidad preparada
    """
    global ingrediente6
    print("Preparando ingrediente 6: Excipiente")
    print("El excipiente es el material de relleno de la vacuna")
    ingrediente6 = 4.2   # Asignamos 4.2 miligramos de excipiente
    print("Excipiente preparado correctamente: " + str(ingrediente6) + " mg")
    return ingrediente6

# FUNCION 8: Revisar que todos los ingredientes esten listos
def revisar_todos_los_ingredientes():
    """
    Verifica que los 6 ingredientes esten preparados y listos para usar
    Revisa que cada ingrediente tenga una cantidad mayor a cero
    Devuelve True si todos estan listos, False si falta alguno
    """
    print("Revisando todos los ingredientes...")
    print("Verificando que cada ingrediente este listo:")
```

```
# Revisamos ingrediente por ingrediente
if ingrediente1 > 0:
    print("Ingrediente 1 (Antigeno): LISTO - " + str(ingrediente1) + " mg")
else:
    print("Ingrediente 1 (Antigeno): FALTA")
    return False

if ingrediente2 > 0:
    print("Ingrediente 2 (Adyuvante): LISTO - " + str(ingrediente2) + " mg")
else:
    print("Ingrediente 2 (Adyuvante): FALTA")
    return False

if ingrediente3 > 0:
    print("Ingrediente 3 (Estabilizante): LISTO - " + str(ingrediente3) + " mg")
else:
    print("Ingrediente 3 (Estabilizante): FALTA")
    return False

if ingrediente4 > 0:
    print("Ingrediente 4 (Conservante): LISTO - " + str(ingrediente4) + " mg")
else:
    print("Ingrediente 4 (Conservante): FALTA")
    return False

if ingrediente5 > 0:
```

```
        print("Ingrediente 5 (Buffer): LISTO - " + str(ingrediente5) + " mg")
    else:
        print("Ingrediente 5 (Buffer): FALTA")
        return False

    if ingrediente6 > 0:
        print("Ingrediente 6 (Excipiente): LISTO - " + str(ingrediente6) + " mg")
    else:
        print("Ingrediente 6 (Excipiente): FALTA")
        return False

    # Si llegamos hasta aqui, todos los ingredientes estan listos
    print("EXCELENTE: Todos los ingredientes estan preparados")
    return True

# FUNCION 9: Sumar las cantidades de todos los ingredientes
def calcular_cantidad_total():
    """
    Suma las cantidades de los 6 ingredientes para saber el total
    Esta funcion realiza una operacion matematica simple
    Devuelve la suma total de todos los ingredientes
    """
    print("Calculando la cantidad total de ingredientes...")

    # Sumamos todos los ingredientes
    total = ingrediente1 + ingrediente2 + ingrediente3 + ingrediente4 + ingrediente5 +
ingrediente6
```

```
    print("Cantidad total de todos los ingredientes: " + str(total) + " mg")
    return total

# FUNCION 10: Mezclar los primeros 3 ingredientes
def hacer_primera_mezcla():
    """
    Combina los primeros 3 ingredientes: antigeno, adyuvante y estabilizante
    Esta es la primera etapa del proceso de mezclado
    Devuelve la cantidad de la primera mezcla
    """
    print("Haciendo la primera mezcla...")
    print("Mezclando: Antigeno + Adyuvante + Estabilizante")

    # Sumamos los primeros 3 ingredientes
    primera_mezcla = ingrediente1 + ingrediente2 + ingrediente3

    print("Primera mezcla completada: " + str(primera_mezcla) + " mg")
    return primera_mezcla

# FUNCION 11: Mezclar los ultimos 3 ingredientes
def hacer_segunda_mezcla():
    """
    Combina los ultimos 3 ingredientes: conservante, buffer y excipiente
    Esta es la segunda etapa del proceso de mezclado
    Devuelve la cantidad de la segunda mezcla
```

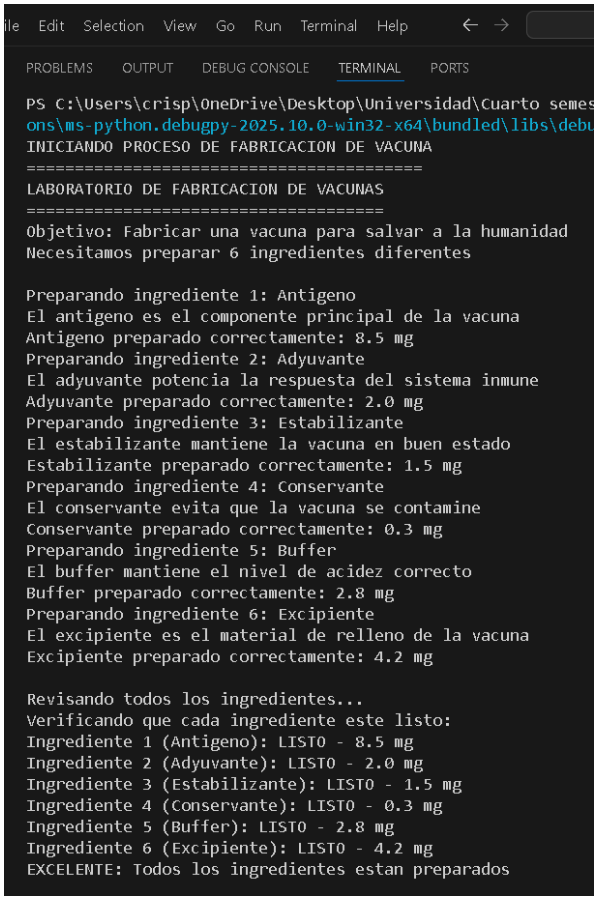
```
    """
    print("Haciendo la segunda mezcla...")
    print("Mezclando: Conservante + Buffer + Excipiente")

    # Sumamos los ultimos 3 ingredientes
    segunda_mezcla = ingrediente4 + ingrediente5 + ingrediente6

    print("Segunda mezcla completada: " + str(segunda_mezcla) + " mg")
    return segunda_mezcla

# FUNCION 12: Combinar las dos mezclas para formar la vacuna final
def combinar_las_dos_mezclas(mezcla1, mezcla2):
    """
    Une las dos mezclas anteriores para crear la vacuna final
    Recibe como parametros las cantidades de ambas mezclas
    Devuelve la cantidad total de la vacuna terminada
    """
    print("Combinando las dos mezclas para formar la vacuna final...")

    # Sumamos las dos mezclas
    vacuna_final = mezcla1 + mezcla2

    print("Vacuna final creada con: " + str(vacuna_final) + " mg totales")
    return vacuna_final

# FUNCION 13: Verificar que la vacuna tenga la calidad correcta
```

```
def verificar_calidad_vacuna(cantidad_total):
    """
    Verifica que la vacuna tenga la cantidad correcta para ser efectiva
    Una vacuna debe tener entre 15 y 25 mg para funcionar bien
    Recibe la cantidad total como parametro
    Devuelve True si la calidad es buena, False si no
    """
    print("Verificando la calidad de la vacuna...")
    print("La vacuna debe tener entre 15 y 25 mg para ser efectiva")

    # Verificamos que la cantidad este en el rango correcto
    if cantidad_total >= 15 and cantidad_total <= 25:
        print("APROBADA: La vacuna tiene la calidad correcta")
        print("Cantidad: " + str(cantidad_total) + " mg (dentro del rango)")
        return True
    else:
        print("RECHAZADA: La vacuna no tiene la calidad correcta")
        print("Cantidad: " + str(cantidad_total) + " mg (fuera del rango)")
        return False

# FUNCION 14: Marcar la vacuna como terminada
def terminar_fabricacion():
    """
    Marca el proceso como completado y la vacuna como lista
    Cambia la variable global vacuna_terminada a True
    Esta es la ultima etapa del proceso
```

```
    """
    global vacuna_terminada
    print("FABRICACION COMPLETADA")
    print("La vacuna esta lista para usar")
    print("MISION CUMPLIDA: Hemos salvado a la humanidad")

    # Marcamos la vacuna como terminada
    vacuna_terminada = True

# FUNCION 15: Funcion principal que ejecuta todo el proceso
def ejecutar_proceso_completo():
    """
    Esta es la funcion principal que coordina todo el proceso
    Llama a todas las otras funciones en el orden correcto
    Es como el director de orquesta del programa
    """
    print("INICIANDO PROCESO DE FABRICACION DE VACUNA")
    print("==========================================")

    # Paso 1: Mostrar el mensaje de inicio
    mostrar_mensaje_inicio()

    # Paso 2: Preparar todos los ingredientes uno por uno
    preparar_ingrediente_1()
    preparar_ingrediente_2()
    preparar_ingrediente_3()
```

```
preparar_ingrediente_4()
preparar_ingrediente_5()
preparar_ingrediente_6()

print("")  # Linea en blanco para separar

# Paso 3: Verificar que todos los ingredientes esten listos
if revisar_todos_los_ingredientes() == False:
    print("ERROR: No se puede continuar, faltan ingredientes")
    return  # Salir de la funcion si algo falta

print("")  # Linea en blanco para separar

# Paso 4: Calcular el total de ingredientes
total_ingredientes = calcular_cantidad_total()

print("")  # Linea en blanco para separar

# Paso 5: Hacer las mezclas
mezcla_1 = hacer_primera_mezcla()
mezcla_2 = hacer_segunda_mezcla()

print("")  # Linea en blanco para separar

# Paso 6: Combinar las mezclas
vacuna_total = combinar_las_dos_mezclas(mezcla_1, mezcla_2)
```

```
    print("")  # Linea en blanco para separar

    # Paso 7: Verificar la calidad
    if verificar_calidad_vacuna(vacuna_total) == True:
        # Paso 8: Si todo esta bien, terminar el proceso
        print("")  # Linea en blanco para separar
        terminar_fabricacion()
    else:
        print("La vacuna no paso el control de calidad")

# EJECUTAR EL PROGRAMA
# Esta parte se ejecuta cuando corremos el programa
if __name__ == "__main__":
    # Llamamos a la funcion principal para que empiece todo
    ejecutar_proceso_completo()

    # Mostrar el estado final
    print("")
    print("ESTADO FINAL DEL PROCESO:")
    if vacuna_terminada == True:
        print("La vacuna fue fabricada exitosamente")
    else:
        print("La vacuna no pudo ser completada")
    print("Gracias por participar en salvar a la humanidad")
```




```
Calculando la cantidad total de ingredientes...
Cantidad total de todos los ingredientes: 19.3 mg

Haciendo la primera mezcla...
Mezclando: Antigeno + Adyuvante + Estabilizante
Primera mezcla completada: 12.0 mg
Haciendo la segunda mezcla...
Mezclando: Conservante + Buffer + Excipiente
Segunda mezcla completada: 7.3 mg

Combinando las dos mezclas para formar la vacuna final...
Vacuna final creada con: 19.3 mg totales

Verificando la calidad de la vacuna...
La vacuna debe tener entre 15 y 25 mg para ser efectiva
APROBADA: La vacuna tiene la calidad correcta
Cantidad: 19.3 mg (dentro del rango)

FABRICACION COMPLETADA
La vacuna esta lista para usar
MISION CUMPLIDA: Hemos salvado a la humanidad

ESTADO FINAL DEL PROCESO:
La vacuna fue fabricada exitosamente
Gracias por participar en salvar a la humanidad
PS C:\Users\crisp\OneDrive\Desktop\Universidad\Cuarto semestre>
```

Conclusiones

La realización de este proyecto ha permitido una comprensión profunda de los conceptos de funciones y su aplicación práctica en la programación. La modularización del código en 15 funciones distintas demostró cómo un problema complejo puede ser descompuesto en tareas más pequeñas y manejables, lo que mejora la legibilidad, la organización y la colaboración en un equipo de trabajo.

La implementación del simulador de vacuna, además de ser un ejercicio académico, muestra cómo el conocimiento de un lenguaje de programación de alto nivel puede aplicarse para modelar procesos del mundo real. El código final, con sus comentarios detallados, es un ejemplo claro de buenas prácticas en la documentación de software.