Universidad Simón Bolívar
Departamento de Computación y Tecnología de la Información
CI3641 – Lenguajes de Programación I
Septiembre–Diciembre 2025 Examen 2

Examen 2 (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 apellido (por ejemplo, si su apellido es “Rodríguez”, podría escoger “Ruby”, “Rust”, “R”, etc.).

Elegí el lenguaje de Python al tener un apellido que comienza con P = Pacheco

(a) Dé una breve descripción del lenguaje escogido.

R:**Python fue creado por Guido van Rossum y lanzado por primera vez en 1991, su filosofía de diseño se centra en la legibilidad del código y en una sintaxis limpia que permite a los programadores expresar conceptos en menos líneas de código que otros lenguajes.**
Una característica distintiva es el uso de indentación para delimitar bloques de código, en lugar de llaves ({}) o palabras clave. Es un lenguaje multiparadigma, es decir soporta programación orientada a objetos, programación imperativa y programación funcional.
Tiene una extensa biblioteca estándar y una cantidad de paquetes de terceros, se utiliza en una amplia variedad de dominios, destacando en desarrollo web en específico en el backend, ciencia de datos, inteligencia artificial y scripting para automatización.

i. Enumere y explique las estructuras de control de flujo que ofrece.

R:**Ofrece varias estructuras para controlar el flujo de ejecución del programa que serán:**
Condicionales if, elif, else: Permiten ejecutar diferentes bloques de código basándose en si una o varias condiciones son verdaderas o falsas.
if: Ejecuta el bloque si la condición es verdadera.
elif: Comprueba una nueva condición si la anterior fue falsa.
else: Ejecuta un bloque si ninguna de las condiciones anteriores (if o elif) fue verdadera.
Bucle while: Repite un bloque de código mientras una condición especificada siga siendo verdadera. Si la condición es falsa desde el inicio, el bloque nunca se ejecuta.
Bucle for: Es la estructura de iteración principal de Python. Se utiliza para iterar sobre los elementos de cualquier objeto iterable como una lista, una tupla, un diccionario, un string y demás.

Manejo de Excepciones try, except, finally, else: Permite gestionar excepciones que pueden ocurrir durante la ejecución.
try: Envuelve el código que podría generar un error.
except: Captura y maneja el error si ocurre uno del tipo especificado.
finally: Ejecuta un bloque de código siempre al final, haya ocurrido un error o no.
else: Se ejecuta si el bloque try finaliza sin generar ninguna excepción.
Control de Bucles break, continue, pass:
break: Interrumpe y sale inmediatamente del bucle sea for o while más interno.
continue: Omite el resto del código de la iteración actual y salta al inicio de la siguiente iteración del bucle.
pass: Es una operación nula. No hace nada. Se usa sintácticamente cuando se requiere una declaración pero no se necesita ninguna acción.

ii. Diga en qué orden evalúan expresiones y funciones.
A. ¿Tiene evaluación normal o aplicativa? ¿Tiene evaluación perezosa?
R:**Utiliza evaluación aplicada conocida como evaluación ansiosa, básicamente es cuando se llama a una función y todos los argumentos de la función se evalúan completamente antes de que la ejecución pase al cuerpo de la función.**
Por ejemplo, en una llamada como mi_funcion(2 + 2, otra_funcion()):

-Python primero evalúa 2 + 2, obteniendo 4.
-Luego evalúa otra_funcion(), ejecutándola y obteniendo su valor de retorno.
-Finalmente, llama a mi_funcion con los resultados de esas evaluaciones.

Por defecto, no tiene evaluación perezosa. El modelo de evaluación de argumentos de Python no es de evaluación normal que es la base de la evaluación perezosa. Sin embargo, Python sí tiene mecanismos para la evaluación perezosa en contextos específicos:

-Operadores Booleanos estos operadores utilizan evaluación de cortocircuito o short-circuiting.
-Generadores: Las funciones que usan la palabra clave yield y las expresiones generadoras son inherentemente perezosas. Producen valores uno a la vez y solo cuando se solicitan generalmente en un bucle for, en lugar de calcularlos todos de una vez.

B. La evaluación de argumentos/operandos se hace de izquierda a derecha, de derecha a izquierda o en un orden arbitrario.
R:**El orden de evaluación es estricto y está garantizado siempre de izquierda a derecha. A diferencia de lenguajes como C o C++, donde el orden de evaluación de los argumentos de una función puede ser indefinido, Este garantiza que las expresiones se evalúan secuencialmente en el orden en que aparecen escritas.**
Por ejemplo, en la expresión func(a(), b(), c()):
-Se evalúa a().
-Luego se evalúa b().
-Luego se evalúa c().
-Finalmente, se llama a func() con los resultados de las tres evaluaciones anteriores.

Lo mismo aplica para los operadores en expresiones aritméticas. En a() + b() * c(), el orden de evaluación de las funciones será a(), luego b(), luego c(), aunque el orden de operación haga que la multiplicación (*) se realice antes que la suma (+).

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

1. Considere la siguiente función:

$$f(n) = \begin{cases} n/2 & \text{si } n \text{ es par} \\ 3n+1 & \text{si } n \text{ es impar} \end{cases}$$

Definimos la función $dist(n)$ como la cantidad de aplicaciones consectuvias de f que se deben hacer sobre n, hasta que el resultado sea 1.

Por ejemplo:

```
f(42) = 21
f(21) = 64
f(64) = 32
f(32) = 16
f(16) = 8
f(8)  = 4
f(4)  = 2
f(2)  = 1
```

Por lo tanto, `count(42)` $= 8$.

Escriba un programa que, dado un entero n, calcule `count(n)`.

Github:
https://github.com/Moepiii/Teoria-de-lenguajes-de-programacion/blob/main/Tarea%20o%20Examen%202%20Rafael%20Valera%201611202/Parte%20b%20de%20la%20pregunta%201/count.py

ii. Implemente el algoritmo Mergesort y explique los detalles de su implementación. Nota: Explicar los detalles no implica traducir línea por línea a lenguaje natural, sino explicar el funcionamiento a grandes rasgos y las decisiones de implementación.

Github:
https://github.com/Moepiii/Teoria-de-lenguajes-de-programacion/blob/main/Tarea%20o%20Examen%202%20Rafael%20Valera%201611202/Parte%20b%20de%20la%20pregunta%201/Mergesort.py

Explicación: En general siempre a lo largo de la carrera para implementar Mergesort se aplica la de dividir y conquistar y eso aplique precisamente.

Con la parte de dividir, la función toma la lista y si tiene más de un elemento la parte recursivamente por la mitad y sigue haciendo esto hasta que solo tiene sublistas de un elemento o cero.

La parte de vencer será simplemente el caso base que es una lista de tamaño 1 que está ordenada por definición y se combina.

Con combinar esta es la parte necesaria de Merger. La función toma dos sublistas que ya están ordenadas y las fusiona en una sola lista nueva y ordenada.

Lo más interesante y aprovechando el lenguaje escogido fue que use slicing, el slicing crea nuevas listas que son copias superficiales de las mitades su ventaja es que hace que el código sea más limpio y legible. No se necesitan de índices complejos para definir los límites de las sublistas en las llamadas recursivas, Lo malo es que en la investigación de esto vi que tiene un costo en memoria, mantiene su rapidez pero no será tan rápida como la clásica que se ve en la materia de algoritmos 2.

Pregunta 2

2. Se desea que modele e implemente, en el lenguaje de su elección, un programa que maneje expresiones aritméticas sobre enteros. Este programa debe cumplir con las siguientes características:

(a) Debe saber tratar expresiones escritas en orden pre–fijo y post–fijo, con los siguientes operadores:

- **suma:** Representada por el símbolo +.
- **resta:** Representada por el símbolo −.
- **multiplicación:** Representada por el símbolo ⋆.
- **división entera:** Representada por el símbolo /.

(b) Una vez iniciado el programa, pedirá repetidamente al usuario una acción para proceder. Tal acción puede ser:

I. `EVAL <orden><expr>`
Representa una evaluación de la expresión en `<expr>`, que está escrita de acuerdo a `<orden>`. `<orden>` solamente puede ser:

- `PRE`: Que representa expresiones escritas en orden pre–fijo.
- `POST`: Que representa expresiones escritas en orden post–fijo.

Por ejemplo:

- `EVAL PRE + * + 3 4 5 7` deberá imprimir `42`.

- `EVAL POST 8 3 - 8 4 4 + * +` deberá imprimir `69`.

II. `MOSTRAR <orden><expr>`
Representa una impresión en orden in–fijo de la expresión en `<expr>`, que está escrita de acuerdo a `<orden>`. El `<orden>` sigue el mismo patrón que en el punto anterior. Su programa debe tomar la precedencia y asociatividad estándar, donde:

- La suma y la resta tienen la misma precedencia.
- La multiplicación y la división entera tienen la misma precedencia.
- La multiplicación y la división entera tienen mayor precedencia que la suma y la resta.
- Todos los operadores asocian a izquierda.

La expresión resultante debe tener la menor cantidad posible de paréntesis, de tal forma que la expresión mostrada como resultado tenga la misma semántica que la expresión que fue pasada como argumento a la acción. Por ejemplo:

- `MOSTRAR PRE + * + 3 4 5 7` deberá imprimir `(3 + 4) * 5 + 7`.
- `MOSTRAR POST 8 3 - 8 4 4 + * +` deberá imprimir `8 - 3 + 8 * (4 + 4)`.

III. `SALIR`
Debe salir del programa.

(c) Al finalizar la ejecución de cada acción, el programa deberá pedir la siguiente acción al usuario.

(d) Investigue herramientas para pruebas unitarias y cobertura en su lenguaje escogido y agregue pruebas a su programa que permitan corroborar su correcto funcionamiento. Como regla general, su programa debería tener una cobertura (de líneas de código y de bifuración) mayor al 80 %.

Github:
https://github.com/Moepiii/Teoria-de-lenguajes-de-programacion/tree/main/Tarea%20o%20Examen%202%20Rafael%20Valera%201611202/Pregunta%202

Pregunta 3

3.

Considere los siguientes iteradores, escritos en Python:

```
def suspenso(a, b):
    if b == []:
        yield a
    else:
        yield a + b[0]
        for x in suspenso(b[0], b[1:]):
            yield x
```

Tomando como referencia las constantes X, Y y Z planteadas en los párrafos de introducción del examen, considere también el siguiente fragmento de código que hace uso del iterador `suspenso`:

```
for x in suspenso(X + Y + Z, [X, Y, Z]):
    print(x)
```

Ejecute, paso a paso, el fragmento de código mostrado (por lo menos al nivel de cada nuevo marco de pila creado) y muestre lo que imprime.

El iterador `misterio`:

```
def misterio(n):
    if n == 0:
        yield [1]
    else:
        for x in misterio(n-1):
            r = []
            for y in suspenso(0, x):
                r = [*r, y]
            yield r
```

Considere también el siguiente fragmento de código que hace uso del iterador `misterio`:

```
for x in misterio(5):
    print(x)
```

Ejecute, paso a paso, el fragmento de código mostrado (por lo menos al nivel de cada nuevo marco de pila creado) y muestre lo que imprime.

Nota: cómo ya conocemos el comportamiento del iterador `suspenso` no es necesario mostrar los pasos internos en el ciclo interno de `misterio`.

(c) Dada una lista de enteros, queremos un iterador que devuelva todos los elementos de la lista en orden (de menor a mayor).

Por ejemplo, para la lista `[1,3,3,2,1]`, los elementos en orden serían:

1 1 2 3 3

Implemente este iterador en el lenguaje de su preferencia.

Nota: el ordenamiento debe ser parte de la lógica del iterador. No es válido ordenar primero la lista y luego devolver los elementos de la lista previamente ordenada.

Mis valores son X=2, Y=0 y Z=2
Bueno primero hay que saber que parámetros van dentro de suspenso como hay una parte del enunciado que la usa la uso para sacar los valores de allí entonces:
-El primer argumento a = X + Y + Z, lo reemplazó y queda 2 + 0 + 2 = 4.
-El segundo argumento b= [X, Y, Z], da [2, 0, 2].
Por lo que la llamada a suspenso tendrá: suspenso(4, [2, 0, 2]).

Su Ejecución queda:
-suspenso(4, [2, 0, 2]):
b ([2, 0, 2]) no es []. Se ejecuta el else.
yield a + b[0] -> yield 4 + 2 -> produce 6
Se llama recursivamente a suspenso(b[0], b[1:]) -> suspenso(2, [0, 2]).
-suspenso(2, [0, 2]):
b ([0, 2]) no es []. Se ejecuta el else.
yield a + b[0] -> yield 2 + 0 -> produce 2.
Se llama recursivamente a suspenso(b[0], b[1:]) -> suspenso(0, [2]).
-suspenso(0, [2]):
b ([2]) no es []. Se ejecuta el else.
yield a + b[0] -> yield 0 + 2 -> produce 2.
Se llama recursivamente a suspenso(b[0], b[1:]) -> suspenso(2, []).
-suspenso(2, []):
b ([]) es []. Se ejecuta el if.
yield a -> produce 2.
Termina esta llamada.
y bueno el bucle for principal imprime cada valor producido.
Las Salida que da son:
6
2
2
2

Ahora misterio:

misterio(0) produce [1].
misterio(1) lo consume (x = [1]) y llama a suspenso(0, [1]).
suspenso(0, [1]) produce 1 (de 0+1) y luego 1 (de suspenso(1, [])).
El bucle interno de misterio(1) produce r = [x, y] -> [[1], 1] dos veces.
misterio(2) consumiría x = [[1], 1].
Luego llamaría a suspenso(0, [[1], 1]).
Esto intentaría calcular a + b[0], que sería 0 + [1], resultando en un TypeError en Python o asumo eso cuando lo probé en Python.

Si acomodo el error que me da para que haga lo que se supone hace en principio la ejecución es:

1. **Se llama a misterio(5).**
2. **misterio(5) llama a misterio(4) para obtener la 4ta fila.**
3. **misterio(4) llama a misterio(3) para obtener la 3ra fila.**
4. **misterio(3) llama a misterio(2) para obtener la 2da fila.**

5. **misterio(2) llama a misterio(1) para obtener la 1ra fila.**
6. **misterio(1) llama a misterio(0) para obtener la 0 fila.**
7. **misterio(0): n == 0, produce [1].**
8. **misterio(1): Recibe x = [1]. Llama a suspenso(0, [1]) (que produce 1, 1). Produce [1, 1].**
9. **misterio(2): Recibe x = [1, 1]. Llama a suspenso(0, [1, 1]) (que produce 1, 2, 1). Produce [1, 2, 1].**
10. **misterio(3): Recibe x = [1, 2, 1]. Llama a suspenso(0, [1, 2, 1]) (que produce 1, 3, 3, 1). Produce [1, 3, 3, 1].**
11. **misterio(4): Recibe x = [1, 3, 3, 1]. Llama a suspenso(0, [1, 3, 3, 1]) (que produce 1, 4, 6, 4, 1). Produce [1, 4, 6, 4, 1].**
12. **misterio(5): Recibe x = [1, 4, 6, 4, 1]. Llama a suspenso(0, [1, 4, 6, 4, 1]) (que produce 1, 5, 10, 10, 5, 1). Produce [1, 5, 10, 10, 5, 1].**

finalmente el bucle for x in misterio(5): print(x) solo recibe el valor final producido por misterio(5) dando como salida:
[1, 5, 10, 10, 5, 1]

Implementación del iterador ordenado

Github:
https://github.com/Moepiii/Teoria-de-lenguajes-de-programacion/blob/main/Tarea%20o%20Examen%202%20Rafael%20Valera%201611202/Pregunta%203/Implementa.py

Pregunta 4

4.

Considere la siguiente definición para una familia de funciones:

$$F_{\alpha,\beta}(n) = \begin{cases} n & \text{si } 0 \leq n < \alpha \times \beta \\ \sum_{i=1}^{\alpha} F_{\alpha,\beta}(n - \beta \times i) & \text{si } n \geq \alpha \times \beta \end{cases}$$

Notemos que $F_{2,1}$ corresponde a la definición para los números de Fibonacci:

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

Como un segundo ejemplo, $F_{3,4}$ corresponde a:

$$F_{3,4}(n) = \begin{cases} n & \text{si } 0 \leq n < 12 \\ F_{3,4}(n-4) + F_{3,4}(n-8) + F_{3,4}(n-12) & \text{si } n \geq 12 \end{cases}$$

Tomando como referencia las constantes X, Y y Z planteadas en los párrafos de introducción del examen, definamos:

$$\alpha = ((X + Y) \text{ mód } 5) + 3, \qquad \beta = ((Y + Z) \text{ mód } 5) + 3.$$

Se desea que realice implementaciones, en el lenguaje imperativo de su elección:

(a) Una subrutina recursiva que calcule $F_{\alpha,\beta}$ para los valores de α y β obtenidos con las fórmulas mencionadas anteriormente. Esta implementación debe ser una traducción directa de la fórmula resultante a código.

(b) Una subrutina recursiva de *cola* que calcule $F_{\alpha,\beta}$.

(c) La conversión de la subrutina anterior a una versión iterativa, mostrando claramente cuáles componentes de la implementación recursiva corresponden a cuáles otras de la implementación iterativa.

Debe usar el mismo el lenguaje para estos tres ejercicios y asegurarse que su lenguaje tenga las estructuras de control de flujo necesarias para realizarlos (su lenguaje escogido debe, por tanto, ser imperativo).

Realice también un análisis comparativo entre las tres implementaciones realizadas, mostrando tiempos de ejecución para diversos valores de entrada y ofreciendo conclusiones sobre la efi- ciencia. Es recomendable que se apoye en herramientas de visualización de datos (como los *plots* de Matlab, R, Octave, Excel, etc.)

Bueno mis numero son X=2, Y=0 y Z=2 entonces primero calculare α y β
α = ((X + Y) mod 5) + 3 = ((2 + 0) mod 5) + 3 = 2 + 3 = 5
β = ((Y + Z) mod 5) + 3 = ((0 + 2) mod 5) + 3 = 2 + 3 = 5
por lo que me dio F 5,5 entonces yo usaría esta:

$$F_{\alpha,\beta}(n) = \begin{cases} n & \text{si } 0 \leq n < \alpha \times \beta \\ \sum_{i=1}^{\alpha} F_{\alpha,\beta}(n - \beta \times i) & \text{si } n \geq \alpha \times \beta \end{cases}$$

al no estar dentro de las otras dos.

Respuesta apartado a:

Github:
https://github.com/Moepiii/Teoria-de-lenguajes-de-programacion/blob/main/Tarea%20o%20Examen%202%20Rafael%20Valera%201611202/Pregunta%204/a.py

Respuesta apartado b:

Antes de nada según mi investigación en Python no hay optimización de llamadas de cola las conicidad como TCO esto significa que, aunque el diseño es de cola recursiva, Python igualmente trata cada llamada como una función normal. Cada llamada va a consumir memoria en la pila del sistema igualmente, dicho eso se deja el código en github con la implementación teórica que corre igualmente.

Github:
https://github.com/Moepiii/Teoria-de-lenguajes-de-programacion/blob/main/Tarea%20o%20Examen%202%20Rafael%20Valera%201611202/Pregunta%204/b.py

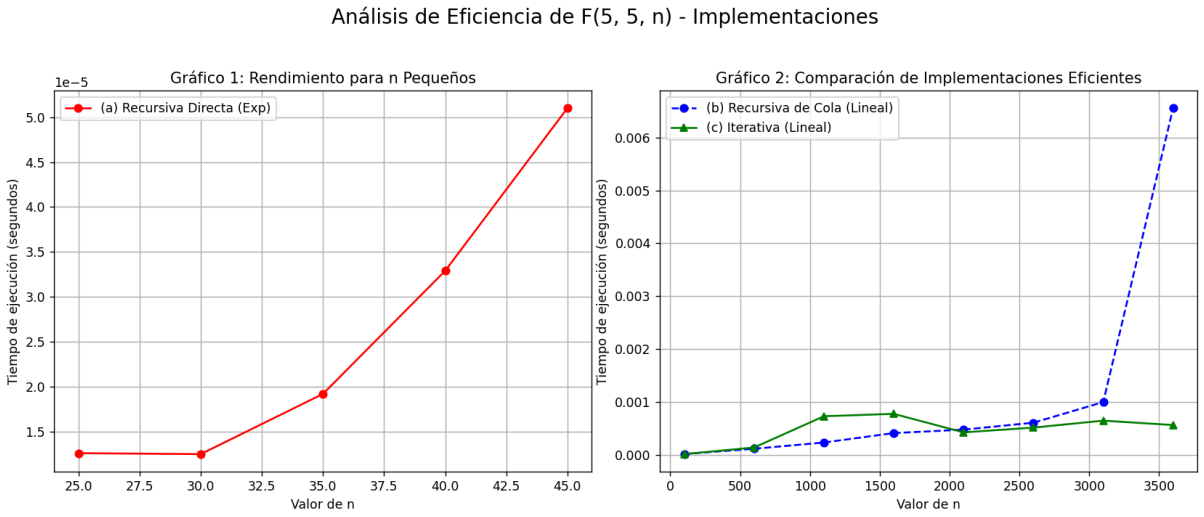
Respuesta apartado c:

Las principales diferencias son que se reemplaza la recursión del helper con un bucle for, y las variables de estado o acumuladores se actualizan en cada iteración.

Github:
https://github.com/Moepiii/Teoria-de-lenguajes-de-programacion/blob/main/Tarea%20o%20Examen%202%20Rafael%20Valera%201611202/Pregunta%204/c.py

Análisis comparativo:

Github:
https://github.com/Moepiii/Teoria-de-lenguajes-de-programacion/blob/main/Tarea%20o%20Examen%202%20Rafael%20Valera%201611202/Pregunta%204/comparacion.py



conclusión:

La (a) llega al resultado apunta de fuerza bruta, su tiempo de ejecución crece exponencialmente, haciéndola la peor opción.
Las (b) y (c) utilizan programación dinámica. Ambas calculan cada subproblema una sola vez y guardan el resultado, lo que les da una eficiencia temporal lineal o muy aproximada a una lineal
La versión (c) es superior a la (b) en mi caso que use Python, porque evita el desbordamiento de la pila de recursión y elimina la sobrecarga de las llamadas a funciones.

BONUS

5. RETO EXTRA: ¡POLIGLOTA!

Considere la misma función `maldad`, definida en el parcial anterior:

$$\text{maldad}(n) = \text{trib}\left(\left\lfloor \log_2\left(N_{n,\,\lfloor \log_2 n \rfloor}\right)\right\rfloor + 1\right)$$

Decimos que un programa es políglota si el mismo código fuente puede ser compilado/interpretado por al menos dos diferentes lenguajes de programación.

Desarrolle un programa políglota que:

- Reciba por entrada estándar o argumento del sistema un valor para n, tal que $n \geq 2$ (esto puede suponerlo, no tiene que comprobarlo). Debe indicar en su informe claramente si su reto recibe la entrada vía entrada estándar o argumento del sistema.
- Imprima el valor de maldad(n).

Su programa debe imprimir el valor correcto y tomando menos de 1 segundo de ejecución, por lo menos hasta $n = 50$ en todos los lenguajes de programación considerados.

Reglas del reto: Intente desarrollar su programa de tal forma que la mayor cantidad de lenguajes de programación puedan compilarlo/interpretarlo. Debe indicar todos los lenguajes para los cuales su código fuente funciona y proporcionar instrucciones para ejecutarlo en cada uno de estos (que puede ser sencillamente un enlace a alguna herramienta online para interpretar el lenguaje, como `tio.run` o `ideone.com`)

- El ganador del reto tendrá 5 puntos extras.
- El segundo lugar tendrá 3 puntos extras.
- El tercer lugar tendrá 1 punto extra.

Github:
https://github.com/Moepiii/Teoria-de-lenguajes-de-programacion/blob/main/Tarea%20o%20Examen%202%20Rafael%20Valera%201611202/Bonus/maldad.txt

Este lenguaje solo es políglota en Python 3,C y C++ si desea probarlo entre a esta pagina https://ideone.com/ seleccione alguno de los lenguajes mencionados luego

seleccione stdin al lado de donde selecciona los lenguajes y coloque algún número por ejemplo el 55 que debería darle el número 2 en los 3 lenguajes.

si por alguna razón los links de los programas no abre, la entrega está subida en este repositorio de github:
https://github.com/Moepiii/Teoria-de-lenguajes-de-programacion/tree/main