

PC N°5: Fundamentos de Programación

Brian La Torre, Flavio Lopez

Diciembre 2024

1 Introduction

Proyecto realizado por dos estudiantes de la Universidad Nacional de Ingeniería para el curso de Fundamentos de Programación CC112-A. Este proyecto fue realizado por los estudiantes Brian La Torre y Lopez Ocampo usando herramientas sugeridas por el profesor como Github, Visual Studio Code y Overleaf. Durante el proceso del proyecto, tuvimos que aprender a usar estas herramientas y además adaptarnos a la sintaxis de python viniendo de una clase basada en C++. Gracias a su similitud y la simplificación de este lenguaje de programación, pudimos realizar ambas partes en no más de 3 días. Esto nos hace ver que gracias a lo aprendido sobre C++, es posible llevarlo a otros lenguajes de programación debido a la similitud en lógica a pesar de la diferencia de sintaxis.

En la primera parte, nos dividimos los problemas para poder avanzar más rápido ya que eran problemas más manejables que ya habíamos visto en el curso. Además, aprendimos a usar las nuevas herramientas durante el desarrollo de cada problema, nos mantuvimos en comunicación constante y ambos aportamos lo más que podíamos al repositorio.

Finalmente, para la parte 2, ambos participantes trabajamos en grupo poniéndonos de acuerdo tanto en la gestión de la documentación como en el proyecto pequeño, la coordinación en entender el código y entendernos entre nosotros fue crucial. Con esto, pudimos escribirlo a nuestra manera y entenderlo con mayor profundidad.

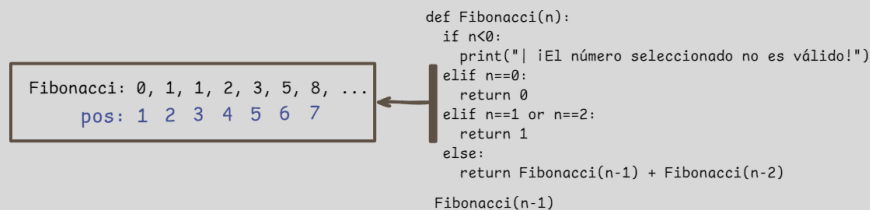
2 Objetivos

- Aplicar el curso en un entorno diferente y sabernos adaptar a este.
- Aprender las ventajas de programar usando python.
- Aprender a trabajar en grupo.
- Aprender a usar nuevas herramientas.

3 Parte 1

3.1 Funciones en python: P1_LaTorre.py

Implementación de funciones, se creó una función Fibonacci para poder generar el n-ésimo término el cual será el retorno. Considerando que el término 1 es el número 0, el término 2 es el número 1, el término 3 es el número 1 y así sucesivamente.



```
def Fibonacci(n):  
    if n<0:  
        print("| ¡El número seleccionado no es válido!")  
    elif n==0:  
        return 0  
    elif n==1 or n==2:  
        return 1  
    else:  
        return Fibonacci(n-1) + Fibonacci(n-2)  
    Fibonacci(n-1)
```

Figure 1: Código de Fibonacci

El siguiente gráfico fue realizado para explicar el funcionamiento de la función por dentro. En este caso el usuario ingresa el número $n=5$, la función recibe un valor $n-1=5$, por lo tanto el valor de "n" dentro de la función Fibonacci será 4 para así poder relacionar la posición con el elemento de manera correcta, ya que no queremos iniciar la posición desde la posición 0 para que sea intuitiva.

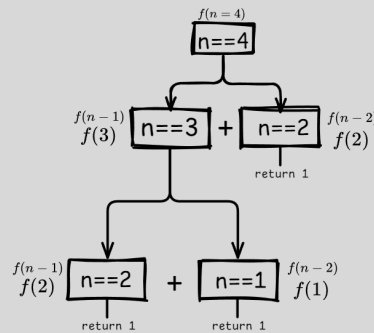


Figure 2: Gráfico del funcionamiento del código

Esta será la salida en la terminal, en el cual podemos ver que para este ejemplo ingresamos el valor 5 y según la figura 1, la posición 5 es ocupada por el elemento 3.

```

| Buscaremos el n-ésimo término de la sucesión de Fibonacci
| ¿Qué valor desea hallar?: 5
| El número en la posición 5 es: 3
  
```

Figure 3: Salida del problema 1

3.2 Cadenas en python: P2_LaTorre.py

La siguiente gráfica fue realizada con la intención de explicar cómo se comporta tanto la función `split()` como la función `len(Cadena)`



Figure 4: Gráfico del funcionamiento del código

En esta imagen que sigue, podremos observar que la función `split()` considera el espacio por default como un separador para poder generar la cadena y luego la función `len()` nos devuelve cuántos elementos hay en esa cadena.

```

|> Ingrese una cadena para contar el número de palabras: Hola Mundo, Proyecto CC112
| El número de palabras que contiene tu cadena es: 4
  
```

Figure 5: Salida del problema 2

3.3 Referencias y asignación dinámica en python: P3_LaTorre.py

En este problema implementamos los mismos conceptos de matrices que en C++. Pedimos el número de filas y el número de columnas para la matriz y luego pedimos cada elemento de la matriz para que el usuario pueda ingresar los valores. El generar la matriz se hace mediante un arreglo temporal el cual nos ayudará a colocarlo en la matriz principal mediante un bucle for. También se usa un bucle for para poder enseñar la matriz generada. Todo esto con un enfoque de que para el usuario sea visualmente cómodo ver la salida del programa.

```
Generemos una matriz

|> Número de filas: 3
|> Número de columnas: 3
|> Elemento [0][0]: 1
|> Elemento [0][1]: 2
|> Elemento [0][2]: 3
|> Elemento [1][0]: 4
|> Elemento [1][1]: 5
|> Elemento [1][2]: 6
|> Elemento [2][0]: 7
|> Elemento [2][1]: 8
|> Elemento [2][2]: 9

| Matriz generada:

[1, 2, 3]
[4, 5, 6]
[7, 8, 9]
```

Figure 6: Salida del problema 3

3.4 Estructuras en python: P4_Lopez.py

Este código define una función llamada MostrarDatos que recibe un diccionario llamado estudiante con las key's "nombre", "Promedio", y "Edad", y muestra sus valores en la consola. Luego, se creó un diccionario estudiante con datos específicos y se pasa como argumento a la función, que imprime:

```
datos del estudiante:
nombre:  FLAVIO
Promedio:  13.34
Edad:  19
```

Figure 7: Salida del problema 4

3.5 Archivos en python: P5_Lopez.py

Este código abre (o crea) un archivo llamado datos1.txt en el directorio Parte1 en modo escritura ("w"), escribe los nombres datos (cada uno en una línea), y al terminar muestra el mensaje: "se agregaron los datos correctamente" para verificar que se escribieron los datos en el archivo de texto. El uso de with asegura que el archivo se cierre automáticamente después de escribir

```
Parte1 >  datos1.txt
1      Lopez
2      Ocampo
3      Flavio
4      Raul
```

Figure 8: Salida en el archivo del problema 5

3.6 Clases en python: P6_Lopez.py

Este código define una clase Persona con un método constructor `_init_` que inicializa los atributos nombre y edad. También tiene un método `MostrarDatos` que imprime los valores de estos atributos de forma estructurada.

```
*Datos de la Persona*
nombre: Julio
edad: 25
```

Figure 9: Salida del problema 6

4 Parte 2

Este código implementa una versión simplificada del juego Buscaminas en python. Define una clase Board que gestiona el tablero, coloca bombas aleatoriamente y calcula cuántas bombas hay alrededor de cada celda. El método principal es `play`, que permite al usuario interactuar con el tablero cavando en posiciones específicas hasta ganar. Métodos utilizados:

- Método `_init_()`

Inicializa el tablero con la dimensión que tendrá el tablero y el número de bombas que se colocarán, además tiene los atributos “board”, que es el tablero que luego será creado con el método `crear_board`; y “cavado” que es un conjunto que se usará para almacenar las celdas que el usuario decida cavar. Finalmente, también llama a las funciones `crear_board` para crear el tablero y colocar las bombas, y la función `asignar_valores_a_la_matriz`, esta calcula cuantas bombas rodean una celda.

- Método `crear_board()`

Crea un tablero $n \times n$ vacío (o inicializado en “None”) para luego colocar las bombas aleatoriamente, para se genera una ubicación aleatoria en el tablero (a cada celda le corresponde un numero) y realizamos operación a esta ubicación para encontrar la fila y la columna correspondiente. Y así continua hasta alcanza el número de bombas.

- Método `asignar_valores_a_la_matriz()`

Se usa para recorrer cada celda del tablero y que encuentre las celdas vacías, una vez haya logrado encontrar una llamará a la función `asignar_numero_de_bombas_cercanas` para calcular cuántas bombas hay a su alrededor y asignarle ese valor Método `asignar_numero_de_bombas_cercanas()` Recorre alrededor de las celdas vecinas para calcular el número de bombas, se implementaron límites dentro de la búsqueda para que no se salga fuera del tablero.

- Método `cavar()`

Simula cavar en una celda (lo que sería darle clic a una celda en el juego original de buscaminas), si la celda contiene una bomba el jugador pierde (retorna falso), si la celda tiene un valor mayor a 0, muestra el valor que almacenaba y por ultimo si la celda está vacía (su valor es 0) cava automáticamente todas las celdas a su alrededor hasta encontrarse con celdas mayores a 0. Retorna verdadero para que el usuario siga jugando.

- Método `_str_()`

Convierte el tablero en una representación visual para mostrarlo al usuario. Las celdas cavadas muestran su valor (número de bombas cercanas o vacío). Las celdas no cavadas se muestran como espacios en blanco.

```

      0  1  2  3  4  5  6  7  8  9
-----
0 |  |  |  |  |  |  |  |  |  |
1 |  |  |  |  |  |  |  |  |  |
2 |  |  |  |  |  |  |  |  |  |
3 |  |  |  |  |  |  |  |  |  |
4 |  |  |  |  |  |  |  |  |  |
5 |  |  |  |  |  |  |  |  |  |
6 |  |  |  |  |  |  |  |  |  |
7 |  |  |  |  |  |  |  |  |  |
8 |  |  |  |  |  |  |  |  |  |
9 |  |  |  |  |  |  |  |  |  |
-----
Dónde te gustaría cavar? (fila,columna): █

```

Figure 10: Salida proyecto básico de buscaminas

5 Conclusiones

Hay una posibilidad de conectar ideas entre ambos lenguajes de programación, tanto python como C++ tienen una manera diferente de trabajar en ciertas situaciones, pero esto no significa que se basen en lo mismo. Aprendimos a que en hay situaciones en los que programas usando python pueden ser muy bien beneficiados debido a su versatilidad y facilidad de aprendizaje a corto plazo. Aún así, en otros escenarios se volvía complicado el entender el lenguaje ya que puede llegar a ser muy permisivo y no muy estricto.