

UNIVERSIDAD NACIONAL DE INGENIERÍA

FACULTAD DE CIENCIAS



CC112 FUNDAMENTOS DE PROGRAMACIÓN

PROYECTO FINAL

ELABORADO POR:

JOSE CARLOS BARRIOS PONCE 20232296K

LUIS ALDAIR AGREDA SÁNCHEZ 20232039H

DOCENTE

Americo Andres Chulluncuy Reynoso

2024-2

Índice general

1. Introducción	3
2. Objetivos	4
3. Desarrollo	5
3.1. Parte 1	5
3.1.1. Problema 1	5
3.1.2. Problema 2	5
3.1.3. Problema 3	6
3.1.4. Problema 4	6
3.1.5. Problema 5	7
3.1.6. Problema 6	8
3.2. Parte 2	8
4. Resultados	9
4.1. Parte 1	9
4.1.1. Problema 1	9
4.1.2. Problema 2	10
4.1.3. Problema 3	10
4.1.4. Problema 4	11
4.1.5. Problema 5	12
4.1.6. Problema 6	12
4.2. Parte 2	13
5. Conclusiones	14
6. Referencias	15

Capítulo 1

Introducción

El presente informe detalla el desarrollo de un proyecto práctico en Python, estructurado en dos partes. La primera parte consiste en la resolución de seis ejercicios fundamentales que abarcan aspectos clave de la programación en Python, tales como el uso de funciones, el manejo de cadenas, la implementación de estructuras de datos, la manipulación de archivos y la definición de clases. Estos ejercicios están diseñados para consolidar los conocimientos adquiridos y poner en práctica conceptos esenciales del lenguaje.

La segunda parte del proyecto consiste en la creación de un juego interactivo de "Piedra, Papel o Tijera", utilizando la librería `random` de Python para simular la elección de la computadora. Este componente del proyecto permite aplicar y reforzar los conceptos de lógica condicional, ciclos y generación de números aleatorios, proporcionando una experiencia más dinámica y divertida.

A través de este proyecto, se busca consolidar los conocimientos básicos y avanzados necesarios para abordar problemas de programación más complejos en el futuro.

Capítulo 2

Objetivos

- Aprender cómo funciona Python, aplicando los conceptos de programación en C++ que aprendimos durante el curso, pero en un entorno más simple y accesible.
- Ampliar mis habilidades de programación y obtener una nueva perspectiva sobre el uso de diferentes lenguajes de programación en proyectos reales.
- Utilizar adecuadamente repositorios (GitHub) para almacenar y gestionar proyectos, siguiendo buenas prácticas como incluir un archivo README.md, mantener un historial de commits claro, y organizar adecuadamente los archivos del proyecto. Además, utilizar un entorno adecuado para el desarrollo en Python, como Visual Studio Code.

Capítulo 3

Desarrollo

3.1. Parte 1

3.1.1. Problema 1

Funciones en Python: Implementar una función recursiva que calcule el factorial de un número.

Resolveremos este problema definiendo una función con parametro n (número) con un caso base y un caso recursivo.

Es importante tener en cuenta los siguientes conceptos:

- En Python, las funciones se definen con la palabra clave `def`, seguida del nombre de la función y sus parámetros entre paréntesis `ejem: def nombre(parametro)`.
- En Python no es necesario definir explícitamente el tipo de variable en una función, ya que Python es un lenguaje de tipado dinámico. Esto significa que las variables adquieren su tipo en tiempo de ejecución según el valor que se les asigne.
- Las f-strings (o formatted string literals) en Python son una forma sencilla y eficiente de incluir valores dentro de una cadena de texto, para usar una f-string, basta con colocar una `f` antes de las comillas de una cadena y encerrar las expresiones que desees incluir en el texto dentro de llaves `.`

3.1.2. Problema 2

Cadenas en Python: Implementar una función que tome una cadena como parámetro y devuelva la cadena invertida.

Para resolver el problema definiremos una función `invertirCadena` donde utilizaremos el concepto de slicing (o corte).

El slicing (o corte) en Python es una técnica que permite obtener partes (subconjuntos) de una secuencia, como cadenas de texto, listas, tuplas o cualquier objeto que implemente la interfaz de secuencia. Esto se realiza utilizando una notación especial con corchetes [] y dos puntos ..

sintaxis básica: secuencia[inicio:fin:paso]

- inicio: Índice donde comienza el corte (incluido).
- fin: Índice donde termina el corte (no incluido).
- paso: Valor que indica cómo avanzar entre elementos (opcional).

El siguiente código muestra ejemplos de slicing en una cadena:

```
1 cadena = "Hola, mundo"
2 print(cadena[:5])      # Desde el inicio hasta el índice 4
3 # Salida: Hola,
4 print(cadena[7:])      # Desde el índice 7 hasta el final
5 # Salida: mundo
6 print(cadena[::2])     # Toma cada segundo carácter
7 # Salida: Hla ud
```

3.1.3. Problema 3

Referencias y asignación dinámica en Python: Implementa la función que acepte una lista de enteros, calcule su suma y devuelva el resultado. El tamaño debe ser introducida por el usuario.

Para resolver el problema utilizaremos un arreglo vacío y un bucle for para rellenar el arreglo en cada iteración, la suma la hallaremos definiendo una función sumLista que sumará todos los valores de nuestro arreglo.

La asignación de memoria dinámica en Python es un proceso automatizado que gestiona el intérprete de Python. Esto significa que el programador no necesita reservar ni liberar memoria manualmente, como ocurre en C++.

3.1.4. Problema 4

Estructuras en Python: Escribir un programa que permita representar puntos en el plano y además implemente una función para calcular la distancia entre dos puntos.

Resolveremos el problema creando una clase para representar el punto en el plano con un método para calcular la distancia entre los dos puntos utilizando la fórmula de la distancia euclidiana.

En Python, las estructuras (struct) como las de C++ no existen directamente. Sin embargo, hay alternativas que permiten organizar datos de manera similar a las estructuras de C++. Por ejemplo, utilizando clases de equivalencia.

La siguiente imagen muestra el ejemplo de una clase en python

```
1 class Persona:
2     def __init__(self, nombre, edad, ciudad):
3         self.nombre = nombre
4         self.edad = edad
5         self.ciudad = ciudad
6
7     # Crear un objeto (similar a una estructura en C++)
8     persona = Persona('Juan', 25, 'Madrid')
9
10    print(persona.nombre) # Acceso al atributo 'nombre'
11
```

3.1.5. Problema 5

Archivos en Python: Implemente un programa usando funciones para escribir datos en un archivo de texto.

Para resolver el problema, definiremos una función para crear el archivo de texto y otra para escribir dentro del archivo de texto.

En Python, trabajar con archivos de texto es sencillo gracias a las funciones integradas. Puedes crear, leer y escribir archivos utilizando la función `open()`.

Modos de apertura en `open()`:

- r: Solo lectura (da error si el archivo no existe).
- W: Escritura, sobrescribe el archivo si existe o lo crea si no.
- a: Añadir al final del archivo, sin borrar el contenido existente.

La siguiente imagen muestra el ejemplo de como crear, escribir y leer un archivo en python

```
1 # Crear y escribir en un archivo
2 with open('archivo.txt', 'w') as archivo:
3     archivo.write("Hola, este es un archivo de ejemplo.\n")
4     archivo.write("Se puede escribir múltiples líneas.\n")
5
6 # Leer un archivo
7 with open('archivo.txt', 'r') as archivo:
8     contenido = archivo.read()
9     print(contenido)
10
```

3.1.6. Problema 6

Clases en python: Definir una clase Persona con atributos nombre y edad, y un método para mostrar estos datos.

Las clases en Python son plantillas para crear objetos. Un objeto es una instancia de una clase, y las clases nos permiten agrupar datos y comportamientos (métodos) bajo un mismo nombre, lo que ayuda a organizar y estructurar el código.

Para crear una clase, usamos la palabra clave class, seguida del nombre de la clase, y luego los métodos dentro de un bloque de código. Los métodos de una clase son funciones definidas dentro de ella.

3.2. Parte 2

Para realizar el juego, definiremos una función llamada play donde pediremos al usuario que elija entre r(piedra), p(papel) o s(tijera) y le asignaremos igualmente un valor aleatorio a una variable computer, luego compararemos las respuestas de ambos para obtener al ganador para comparar definiremos la función win.

En Python, los valores aleatorios se pueden generar utilizando el módulo random, que proporciona varias funciones para trabajar con números aleatorios. Este módulo se utiliza para generar valores aleatorios dentro de un rango específico, mezclar secuencias de datos, seleccionar elementos de manera aleatoria, etc.

Funciones principales del modulo random

- Generar un número aleatorio entre un rango específico: randint(a, b)
- Elije un numero aleatorio de una lista o secuencia: choice(sequence)
- Mezclar una lista aleatoriamente: shuffle(sequence)
- Generar números aleatorios de una distribución específica: gauss(mu, sigma)

```
1 import random
2
3 # Generar un número decimal aleatorio entre 0 y 1
4 numero_decimal = random.random()
5 print(f"Número decimal entre 0 y 1: {numero_decimal}")
6
7 # Generar un número decimal aleatorio en un rango específico
8 numero_rango = random.uniform(5, 15) # Un número decimal entre 5 y 15
9 print(f"Número decimal entre 5 y 15: {numero_rango}")
10
11 # Seleccionar un número aleatorio de una lista
12 numeros = [10, 20, 30, 40, 50]
13 numero_seleccionado = random.choice(numeros)
14 print(f"Número seleccionado de la lista: {numero_seleccionado}")
15
16
```


Capítulo 4

Resultados

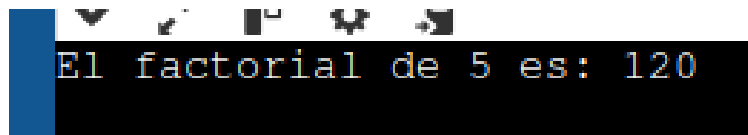
4.1. Parte 1

4.1.1. Problema 1

Código final del problema 1:

```
1 def factorial(n):
2     if n == 0 or n == 1: # Caso base
3         return 1
4     else:                # Caso recursivo
5         return n * factorial(n - 1)
6
7 #Función en uso
8 numero = 5
9 print(f"El factorial de {numero} es: {factorial(numero)}")
10
```

Sálida del código 1:

A terminal window with a black background and a blue vertical bar on the left. It displays the output of the code: "El factorial de 5 es: 120". Above the text, there are several small, faint icons: a downward arrow, a magnifying glass, a square, a gear, and a document with a checkmark.

```
El factorial de 5 es: 120
```

4.1.2. Problema 2

Código final del problema 2:

```
1 def invertir_cadena(cadena):  
2     return cadena[::-1] #Utilizamos el concepto de slicing(o corte)  
3  
4 texto = "Hola, mundo"  
5 print(f"Cadena original: {texto}")  
6 print(f"Cadena invertida: {invertir_cadena(texto)}")  
7
```

Sálida del código 2:

```
Cadena original: Hola, mundo  
Cadena invertida: odnum ,aloH
```

4.1.3. Problema 3

Código final del problema 3:

```
1 def suma_lista(lista):  
2     return sum(lista)  
3  
4 tamaño = int(input("Introduce el tamaño de la lista: "))  
5  
6 numeros = [] # La lista se crea vacía  
7 for i in range(tamaño):  
8     numero = int(input(f"Introduce el número {i + 1}: "))  
9     numeros.append(numero)  
10  
11 resultado = suma_lista(numeros)  
12 print(f"La suma de los elementos es: {resultado}")  
13
```

Sálida del código 3:

```
Introduce el tamaño de la lista:
Introduce el número 1: 4
Introduce el número 2: 2
Introduce el número 3: 6
Introduce el número 4: 3
La suma de los elementos es: 15
```

4.1.4. Problema 4

Código final del problema 4:

```
1 import math
2
3 class Punto:
4     def __init__(self, x, y):
5         self.x = x # Coordenada x
6         self.y = y # Coordenada y
7
8     def __str__(self):
9         return f"({self.x}, {self.y})"
10
11     def distancia(self, punto2):
12         return math.sqrt((self.x - punto2.x)**2 + (self.y - punto2.y)**2)
13
14 p1 = Punto(1, 2)
15 p2 = Punto(4, 6)
16
17 print(f"Punto 1: {p1}")
18 print(f"Punto 2: {p2}")
19
20 dist = p1.distancia(p2)
21 print(f"La distancia entre los puntos es: {dist:.2f}")
22
```

Sálida del código 4:

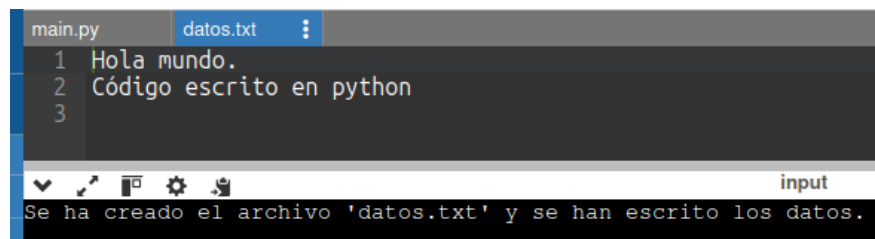
```
Punto 1: (1, 2)
Punto 2: (4, 6)
La distancia entre los puntos es:
```

4.1.5. Problema 5

Código final del problema 5:

```
1 def crear_archivo(nombre_archivo):
2     with open(nombre_archivo, 'w') as archivo:
3         pass
4
5 def escribir_datos(nombre_archivo, datos):
6     with open(nombre_archivo, 'a') as archivo:
7         for dato in datos:
8             archivo.write(dato + '\n')
9
10 nombre_archivo = "datos.txt"
11 crear_archivo(nombre_archivo)
12
13 datos_a_escribir = [
14     "Hola mundo.",
15     "Código escrito en python"
16 ]
17
18 escribir_datos(nombre_archivo, datos_a_escribir)
19
20 print(f"Se ha creado el archivo '{nombre_archivo}' y se han escrito los datos.")
```

Sálida del código 5:



```
main.py  datos.txt  :
1  Hola mundo.
2  Código escrito en python
3
input
Se ha creado el archivo 'datos.txt' y se han escrito los datos.
```

4.1.6. Problema 6

Código final del problema 6:

```
1 class Persona:
2     def __init__(self, nombre, edad):
3         self.nombre = nombre
4         self.edad = edad
5
6     def mostrar_datos(self):
7         print(f"Nombre: {self.nombre}, Edad: {self.edad}")
8
9 persona1 = Persona("Juan", 25)
10 persona1.mostrar_datos()
11
```

Sálida del código 6:

```
Nombre: Juan, Edad: 25
```

4.2. Parte 2

Código final del juego "Piedra, papel o tijera":

```
1 import random
2
3 def play():
4     user = input("¿Cuál es tu elección? 'r' para roca, 'p' para papel, 's' para tijeras\n")
5     computer = random.choice(['r', 'p', 's'])
6
7     if user == computer:
8         return 'Es un empate'
9
10    # r > s, s > p, p > r
11    if is_win(user, computer):
12        return '¡Ganaste!'
13
14    return '¡Perdiste!'
15
16 def is_win(player, opponent):
17     # devuelve True si el jugador gana
18     # r > s, s > p, p > r
19     if (player == 'r' and opponent == 's') or (player == 's' and opponent == 'p') \
20         or (player == 'p' and opponent == 'r'):
21         return True
22
23 print(play())
```

Sálida del código:

```
¿Cuál es tu elección? 'r' para roca, 'p' para papel, 's' para tijeras
r
Es un empate
```

Capítulo 5

Conclusiones

- El proyecto permitió aplicar conceptos clave de programación en Python, como funciones, estructuras de datos, manejo de archivos y clases, mientras se exploraban similitudes con C++.
- La integración con GitHub y el uso de un entorno como Visual Studio Code fomentaron buenas prácticas de desarrollo.
- Python ofrece varios beneficios sobre C++ gracias a su simplicidad y productividad. Su sintaxis más legible facilita el aprendizaje y reduce los errores comunes. También cuenta con una amplia gama de bibliotecas para tareas como inteligencia artificial, ciencia de datos y desarrollo web, lo que acelera el desarrollo de proyectos. Sin embargo, C++ sigue siendo más eficiente en términos de rendimiento y control de bajo nivel.

Capítulo 6

Referencias

1. Programación ATS. (2018). Programación ATS. YouTube.
<https://www.youtube.com/c/ProgramacionATS>
2. freeCodeCamp.org. (2017). freeCodeCamp.org. YouTube.
<https://www.youtube.com/c/freecodecamp>
3. Lospinoso, J. (2019). C++ Crash Course A Fast-Paced Introduction. No starch press San Francisco.
4. El Libro de Python. (n.d.). Sintaxis en Python. El Libro De Python.
<https://ellibrodepython.com/sintaxis-python>