Trabajo final de curso de Fundamentos de programación 2024-I

Delgado, G. (60 %)
Escuela de Ciencias de la Computación
Universidad Nacional de Ingeniería
Lima, Perú
gustavo.delgado.r@uni.pe

Mori, J. (40 %)
Escuela de Ciencias de la Computación
Universidad Nacional de Ingeniería
Lima, Perú
jean.mori.m@uni.pe

***Resumen*—El curso de fundamentos de programación ofrecido por la Universidad Nacional de Ingeniería aborda principalmente la enseñanza de algunos de los conceptos base de programación en el lenguaje C++, como la manipulación de archivos binarios, el uso de punteros o la manipulación de memoria dinámica. Sin embargo, como proyecto final del curso, la plana docente decidió plantear una serie de problemas desde muy básicos a algunos con cierta complejidad utilizando el lenguaje de programación python.**

Así pues, el presente documento tiene como finalidad exponer los conceptos investigados para la resolución de los problemas propuestos por los profesores del curso, así como también dar a conocer los resultados del desarrollo de dichos problemas.

Índice de términos— Factorial, Máximo común divisor, Clases en Python, Machine Learning.

I. Introducción

Python es un lenguaje de programación versátil y poderoso ampliamente usado en varios campos, incluyendo la ciencia de datos, desarrollo web y Computación científica. Uno de los aspectos por los que destaca python es por su simplicidad y su fácil lectura. Aquí hacemos una simple comparación de códigos entre python y c++:
Python:

```python
print("Hola mundo")
```

Listing 1. Script de bienvenida en Python

C++:

```cpp
#include <iostream>
using namespace std;
int main(){
  cout<<"Hola mundo";
  return 0;
}
```

Listing 2. Script de bienvenida en C++

II. Marco Teórico

II-A. Función Factorial

La función factorial calcula el producto de todos los enteros positivos desde 1 hasta un número entero no negativo dado n. Se denota como $n!$.

$$n! = n \cdot (n-1) \cdot (n-2) \cdot \ldots \cdot 2 \cdot 1$$

Por ejemplo:

- $5! = 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1 = 120$
- $0! = 1$ (por convención)

II-A1. Definición Recursiva: La función factorial se puede definir recursivamente de la siguiente manera:

$$n! = \begin{cases} 1 & \text{si } n = 0 \\ n \cdot (n-1)! & \text{si } n > 0 \end{cases}$$

II-A2. Propiedades:

- $n! > 0$ para todos los enteros no negativos n.
- $0! = 1$ (por convención).
- El factorial crece rápidamente: $n! > n^2$ para n suficientemente grande.

II-B. Algoritmo de Euclides

El algoritmo de Euclides se utiliza para encontrar el **máximo común divisor (MCD)** de dos enteros positivos. Dados dos números a y b, el algoritmo procede de la siguiente manera:

1. Realiza divisiones sucesivas:

   $$a \mod b = R$$

   donde R es el residuo al dividir a entre b.
2. Actualiza los valores:

   $$a = b \quad \text{y} \quad b = R$$

   Repite los pasos 1 y 2 hasta que $a \mod b$ sea mayor que 0.
3. El MCD es igual al valor final de b.

II-C. *Clases en Python*

En Python, las **clases** son como **constructores de objetos** o **plantillas** para crear instancias. Cada objeto creado a partir de una clase tiene sus propias **propiedades** (variables) y **métodos** (funciones).

II-C1. Crear una Clase: Para crear una clase, se utiliza la palabra clave *class*. Aquí un ejemplo:

```
class MiClase:
  x = 5

```

Listing 3. Creación de una clase en Python

En este caso, hemos creado una clase llamada **MiClase** con una propiedad **x** inicializada a 5.

II-C2. Crear un objeto: Ahora podemos usar la clase **MiClase** para crear objetos:

```
p1 = MiClase()
print(p1.x)

```

Listing 4. Creación de un objeto en Python

Esto creará un objeto llamado p1 y mostrará el valor de *x*, que es 5.

II-C3. El Método init(): Para hacer que nuestras clases sean más útiles, utilizamos el método **_init_()** . Este método se ejecuta automáticamente cuando se crea un objeto. Aquí un ejemplo:

```
class Persona:
  def __init__(self, nombre, edad):
    self.nombre = nombre
    self.edad = edad

p1 = Persona("Juan", 30)
print(f"Nombre: {p1.nombre}, Edad: {p1.edad}")

```

Listing 5. Creación de un objeto en Python

En este caso, se ha creado una clase Persona con un método _init_() que asigna valores a las propiedades **nombre** y **edad**.

II-C4. Métodos de clase: Las clases también pueden contener **métodos**. Los métodos son funciones que pertenecen al objeto. Aquí un ejemplo:

```
class Persona:
  def __init__(self, nombre, edad):
    self.nombre = nombre
    self.edad = edad

  def saludar(self):
    print(f"Hola, mi nombre es {self.nombre}."
    )

p1 = Persona("Juan", 30)
p1.saludar()

```

Listing 6. Creación de un método en Python

En este caso, el método saludar muestra un saludo utilizando el nombre de la persona.

II-D. *Fundamentos de Tkinter*

Tkinter es la interfaz de python para **Tk**, el conjunto de herramientas gráficas utilizado para construir programas con una interfaz gráfica.

II-D1. Hola, mundo!: El primer programa que se recomienda realizar en Tkinter es **Hola mundo!**. La idea es mostrar una ventana simple con un mensaje de bienvenida.

```
import tkinter as tk

root = tk.Tk()
label = tk.Label(root, text="Hola, Mundo!")
label.pack()

root.mainloop()

```

Listing 7. Creación de un programa en Tkinter

De esta manera podemos usar tkinter para crear una interfaz gráfica para los programa que desarrollemos.

II-E. *Introducción a Openseespy*

OpenSeesPy es una interfaz de Python para OpenSees, un software de análisis estructural que permite realizar simulaciones de la respuesta sísmica de estructuras. Esta sección proporciona una introducción básica a OpenSeesPy, incluyendo la configuración del entorno, la creación de modelos y la ejecución de análisis.

II-E1. Creación de un Modelo Básico: A continuación se muestra un ejemplo de cómo crear un modelo básico en OpenSeesPy. Este modelo consiste en un marco simple con dos nodos y una barra.

```
import openseespy.opensees as ops

# Definir el modelo
ops.wipe()
ops.model('basic', '-ndm', 2, '-ndf', 3)

# Crear nodos
ops.node(1, 0, 0)
ops.node(2, 144, 0)
ops.node(3, 168, 0)
ops.node(4, 72, 96)

# Asignar restricciones a los nodos
ops.fix(1, 1, 1, 1)
ops.fix(2, 0, 1, 0)
ops.fix(3, 0, 1, 0)

# Definir secciones y materiales
ops.uniaxialMaterial('Elastic', 1, 3000.0)

# Crear elementos
ops.element('truss', 1, 1, 4, 10.0, 1)
ops.element('truss', 2, 2, 4, 10.0, 1)
ops.element('truss', 3, 3, 4, 10.0, 1)

# Definir cargas
ops.timeSeries('Linear', 1)
ops.pattern('Plain', 1, 1, '-fact', 1.0)
ops.load(4, 100.0, -50.0, 0.0)

# Realizar el analisis
ops.system('BandGen')
```

```
ops.numberer('Plain')
ops.constraints('Plain')
ops.integrator('LoadControl', 1.0)
ops.algorithm('Linear')
ops.analysis('Static')

ops.analyze(1)

# Obtener resultados
disp = ops.nodeDisp(4)
print(f'Desplazamientos en el nodo 4: {disp}')

```

Listing 8. Creación de un Modelo Básico en OpenSeesPy

II-E2. 5: Explicación del código

- `ops.wipe()`: Borra el modelo previo.
- `ops.model('basic', '-ndm', 2, '-ndf', 3)`: Define un modelo en 2 dimensiones con 3 grados de libertad por nodo.
- `ops.node()`: Crea nodos con coordenadas específicas.
- `ops.fix()`: Asigna restricciones a los nodos.
- `ops.uniaxialMaterial('Elastic', 1, 3000.0)`: Define un material elástico.
- `ops.element('truss', ...)`: Crea elementos de tipo armadura.
- `ops.timeSeries()` y `ops.pattern()`: Definen la serie temporal y el patrón de carga.
- `ops.load()`: Aplica cargas a los nodos.
- `ops.system()`, `ops.numberer()`, `ops.constraints()`, `ops.integrator()`, `ops.algorithm()` y `ops.analysis()`: Configuran y realizan el análisis estático.
- `ops.analyze(1)`: Ejecuta el análisis.
- `ops.nodeDisp(4)`: Obtiene los desplazamientos del nodo 4.

II-F. Scikit-learn

Scikit-learn es una biblioteca de código abierto para el aprendizaje automático en Python. Proporciona herramientas simples y eficientes para la minería de datos y el análisis de datos. Es accesible a todos y reutilizable en diversos contextos.

II-F1. Carga de Datos: Scikit-learn proporciona varios conjuntos de datos de ejemplo que pueden ser cargados fácilmente. A continuación se muestra un ejemplo de cómo cargar el conjunto de datos Iris.

```
from sklearn.datasets import load_iris
import pandas as pd

# Cargar el conjunto de datos Iris
iris = load_iris()
data = pd.DataFrame(data=iris.data, columns=iris.
    feature_names)
data['target'] = iris.target

print(data.head())

```

Listing 9. Carga del Conjunto de Datos Iris

II-F2. Entrenamiento de un Modelo: A continuación se muestra un ejemplo de cómo entrenar un modelo de clasificación utilizando el algoritmo de vecinos más cercanos (K-Nearest Neighbors).

```
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

# Dividir los datos en conjuntos de entrenamiento y
    prueba
X_train, X_test, y_train, y_test = train_test_split(
    data[iris.feature_names], data['target'],
    test_size=0.2, random_state=42)

# Crear el modelo K-Nearest Neighbors
knn = KNeighborsClassifier(n_neighbors=3)

# Entrenar el modelo
knn.fit(X_train, y_train)

# Predecir las etiquetas para el conjunto de prueba
y_pred = knn.predict(X_test)

# Evaluar la precision del modelo
accuracy = accuracy_score(y_test, y_pred)
print(f'Precision del modelo: {accuracy}')

```

Listing 10. Entrenamiento de un Modelo de K-Nearest Neighbors

II-F3. Evaluación del modelo: La evaluación del rendimiento del modelo es crucial para entender su eficacia. A continuación se muestra un ejemplo de cómo evaluar un modelo utilizando una matriz de confusión.

```
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

# Calcular la matriz de confusion
cm = confusion_matrix(y_test, y_pred)

# Visualizar la matriz de confusion
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
    xticklabels=iris.target_names, yticklabels=iris.
    target_names)
plt.xlabel('Prediccion')
plt.ylabel('Realidad')
plt.show()

```

Listing 11. Evaluación del Modelo con Matriz de Confusión

Scikit-learn es una biblioteca poderosa y flexible que facilita el proceso de creación, entrenamiento y evaluación de modelos de aprendizaje automático.

III. Metodología

El trabajo realizado implicó investigar las características del lenguaje Python, así como también la implementación directa de lo aprendido sobre un grupo de problemas propuestos por los profesores del curso.

IV. Experimentación y resultados

IV-A. Ejercicios de implementación directa

En esta primera parte del trabajo, se nos solicitó implementar algunos problemas sencillos en Python, a continuación la implementación de los mismos.

IV-A1. Factorial: Como primer problema, se solicitó implementar una función que calcule el factorial de un número.

```
def Factorial(a):
if a==0:
  return 1
else:
  return a*Factorial(a-1)
```

Listing 12. Factorial

IV-A2. MCD: Como segundo problema, se solicitó implementar una función que calcule el MCD de dos números usando recursividad.

```
def MCD(a, b):
if b==0:
  return a
else:
  return MCD(b,a%b)
```

Listing 13. MCD

IV-A3. Clase y método de impresión: Como tercer problema, se solicitó implementar una función que cree una clase y un método de impresión de datos de un objeto de esta clase.

```
class Persona:
def __init__(self, nombre, edad):
  self.nombre = nombre
  self.edad = edad

def mostrar_datos(self):
  print(f"Nombre: {self.nombre}, Edad: {self.edad}")


person1 = Persona("Gustavo", 34)

person1.mostrar_datos()
```

Listing 14. Clase e impresión

IV-A4. Clase y cálculo de datos: Como cuarto problema, se solicitó implementar una función que cree una clase y un método de cálculo de datos de un objeto de esta clase.

```
class CuentaBancaria:
  def __init__(self, NumeroCuenta, CantidadDePlatita
    ):
    self.NumeroCuenta = NumeroCuenta
    self.CantidadDePlatita = CantidadDePlatita
  def mostrar_datos(self):
    print(f"Numero de Cuenta: {self.NumeroCuenta},
    Dinero en cuenta: {self.CantidadDePlatita}")
  def depositar(self,deposito):
    self.CantidadDePlatita=self.CantidadDePlatita+
    deposito
    self.mostrar_datos()
  def retirar(self,retiro):
    saldofinal=self.CantidadDePlatita-retiro
    if (saldofinal<0):
      print("Saldo insuficiente para efectura retiro
    ")
    else:
      self.CantidadDePlatita=saldofinal
      self.mostrar_datos()

Cuenta1 = CuentaBancaria("Gustavo", 34)
```

Listing 15. Clase y cálculo

IV-A5. Manipulación de archivos de texto: Como quinto problema, se solicitó implementar una serie de funciones que permitan la creación, modificación y guardado de archivos de texto.

```
# Funcion para crear un nuevo archivo de texto
def create_file(file_name, content):
  with open(file_name, 'w') as file:
    file.write(content)
  print(f"Archivo '{file_name}' creado con exito")

# Funcion para leer y mostrar los datos de un
    archivo de texto
def read_file(file_name):
  try:
    with open(file_name, 'r') as file:
      content = file.read()
      print(f"Content of '{file_name}':")
      print(content)
  except FileNotFoundError:
    print(f"File '{file_name}' not found!")

# Funcion para agregar datos a un archivo de texto
def append_to_file(file_name, new_content):
  try:
    with open(file_name, 'a') as file:
      file.write(new_content + '\n')
    print(f"New content appended to '{file_name}'
    successfully!")
  except FileNotFoundError:
    print(f"File '{file_name}' not found!")

```

Listing 16. Clase y cálculo

V. Conclusiones

VI. Discusión

VII. Conclusiones y trabajos futuros

Referencias

[1] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner,Christopher Clark, Kenton Lee, Luke Zettlemoyer. Deep contextualized word representations. NAACL 2018.

[2] Jacob Devlin, Ming-Wei Chang, Kenton Lee, Kristina Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding (2018). Disponible en https://arxiv.org/abs/1810.04805.