RESUMEN CON EJEMPLOS Y ANÁLISIS DE LAS CLASES DEL TERCER CORTE DE INFORMÁTICA 3

Camila Durán Salazar

CLASE 1 Y 2

MÓDULOS TRIQUI

En estas dos clases realizamos el código en python para un juego de triqui entre dos jugadores y tratamos de entender cómo funcionaban todas sus partes; como era la parte del control del juego, la interfaz y su lógica.

Algunos de los pasos que realizamos y en su respectivo orden fueron los siguientes:
Primero se creó una función para mostrar el tablero. Ésta imprimirá (utilizando un for) la multilista que contiene el tablero.

La función rotar turno utiliza el método *rotate* para rotar los valores de X y O y devuelve el que queda en la posición

La siguiente función recibe como valor de parámetro el jugador (“X” o “O”) y revisa la multilista analizando todo el tablero.

La función principal, a la que llamamos juego, utiliza un ciclo *While* infinito para preguntar al usuario la posición (fila, columna) en el tablero en la que desea dejar la marca del jugador en turno

Si se introduce un valor diferente, (como abcxyz) al no poder convertirlo a dos elementos de lista separándolos por comas, se generará un error que será atrapado en el bloque *excepto*. Se invoca la función principal juego() misma que llama a todas las demás.

Veamos, por ejemplo, cómo quedó la interfaz del juego, o sea la parte visual:

```
Bienvenido, esto es triki.
Para ganar debe completar una línea recta,
compuesta por un mismo caracter ("x" o "o")
Línea recta => horizontal o vertical o diagonal
Debe ingresar la posición 0-8, para ingresar
la opción durante su turno. Las posiciones son
las siguientes:
tablero =>
 0 | 1 | 2
```

```
-------------
3 | 4 | 5
-------------
6 | 7 | 8

print(explicacion)



def imprimirTablero(tableroLogico: list):
tableroVisual = """
{} | {} | {}
-------------
{} | {} | {}
-------------
{} | {} | {} """.format(tableroLogico[0],
tableroLogico[1], tableroLogico[2], tableroLogico[3],
tableroLogico[4], tableroLogico[5], tableroLogico[6],
tableroLogico[7], tableroLogico[8])
tableroVisual = tableroVisual.replace("None", " ")
print(tableroVisual)



if __name__ == "__main__":
explicarJuego()
tableroLogico = ["x", None, None, None, None, "x", None, None, "o"]
imprimirTablero(tableroLogico)
```

CLASE 3

ESTRUCTURAS PARA ANÁLISIS DE DATOS:
Los datos pueden generarse, recopilarse y almacenarse en una variedad de formatos, pero cuando se trata de analizarlos, no todos los formatos de datos se crean de la misma manera.

Las principales estructuras son:

- Vectores.
- Matrices y Arrays.
- Data Frames.
- Listas.

Pero estas no son eficientes para analizar o almacenar

datos, por lo tanto existen otras estructuras en numpy y pandas

numpy:| arrays => vectores (arrays 1D)
| matrices (arrays 2D)
| tensores (arrays 3D)

pandas:| series => vectores (solo 1D)
| funcionan como las listas y diccionarios para indexar
|
| dataFrames => matrices(solo 2D)
| funcionan como las hojas de excel
| funcionan como las listas y diccionarios para indexar

La estructura básica para trabajar son los arrays, por lo tanto deben entenderse
muy bien para poder utilizar las otras estructuras series y dataframes

Una matriz es una estructura de datos central de la biblioteca NumPy. Una matriz es una cuadrícula de valores y contiene información sobre los datos sin procesar, cómo ubicar un elemento y cómo interpretar un elemento. Tiene una cuadrícula de elementos que se pueden indexar de varias maneras.

Numpy posee muchas herramientas, entre ellas tenemos:

1) Funciones integradas de numpy

* Creacion de arreglos =>
np.array(<iterable>)
np.arange(<inicio>,<fin>,<salto>)
np.ones((<filas>,<columnas>))
np.zeros(<filas>, <columnas>)
* Redimensionamiento =>
np.reshape((<filas>, <columnas>))
* Apilamiento =>
np.hstack((<elemento1>, <elemento2>))
np.hstack((<elemento1>, <elemento2>))
* Indexado y slicing =>
vector[<columna>]

matriz[<fila>,<columna>]

tensor[<profundidad>,<fila>,<columna>]

* Almacenamiento =>

np.savetxt("ruta", <arreglo>)

np.loadtxt("ruta")

2) Métodos en numpy

* Valor minimo =>

arreglo.min()

* Valor máximo =>

arreglo.max()

* Promedio =>

arreglo.mean()

* Desviacion estandar =>

arreglo.std()

* Suma del arreglo =>

arreglo.sum()

CLASE 4

SERIES PANDAS

La serie Pandas es una matriz etiquetada unidimensional capaz de contener datos de cualquier type(entero, cadena, flotante, objetos de python, etc.). Las etiquetas de los ejes se denominan colectivamente *índice* .

Las etiquetas no necesitan ser únicas, pero deben ser de tipo hash. El objeto admite la indexación basada en etiquetas y números enteros y proporciona una serie de métodos para realizar operaciones relacionadas con el índice.

El arreglo está compuesto por indices numericos => 0,1,2,3

indices clave => "Enero", "Febrero",...

TRIBUTOS =>

serie.index : retorna los indices clave

serie.values : retora la data

serie.shape : retorna el tamaño

METODOS =>

serie.describe() : devuelve otra serie, describiendo la serie original

serie.mean() : devuelve la media

serie.std() : devuelve la desviacion estandar

serie.min() : devuelve el valor minimo

serie.max() : devuelve el valor maximo

serie.idxmin() : devuelve la clave del minimo

serie.idxmax() : devuelve la clave del maximo

serie.value_counts() : devuelve las frecuencias de la serie

INDEXADO =>

Las series se pueden indexar utilizando el indice numerico o el indice clave

serie[0]

serie["Enero"])

CLASE 5

DATAFRAME: PANDAS

A diferencia de las Series, que son objetos correspondientes a paneles unidimensionales, los DataFrames son paneles bidimensionales compuestos por filas y columnas, que permiten destacar las relaciones entre las distintas variables de la serie de datos. Un DataFrame es una serie de Series Pandas indexadas por un valor.

En esta clase vimos como crear un data frame en pandas, vamos a ver las opciones que existen para crear un DataFrame y agregar datos. Lo que nos enseñara además algunos de los métodos que existen para modificar el contenido de un DataFrame.

Se crea de la siguiente manera:

```
import pandas as pd

hoja1 = pd.DataFrames(data = <arreglo 2D numpy>,

columns = ["columna1", ...., "columnaN"],

index = ["fila1",..., "filaN" ])
```

Ejemplo sencillo de un data frame:

```
import pandas as pd


data = {
  "calories": [420, 380, 390],
  "duration": [50, 40, 45]}


#load data into a DataFrame object:
df = pd.DataFrame(data)


print(df)
```

CLASE 6

Pandas es una herramienta de manipulación de datos de alto nivel desarrollada por Wes McKinney. Es construido sobre Numpy y permite el análisis de datos que cuenta con las estructuras de datos que necesitamos para limpiar los datos en bruto y que sean aptos para el análisis (por ejemplo, tablas). Como Pandas permite realizar tareas importantes, como alinear datos para su comparación, fusionar conjuntos de datos, gestión de datos perdidos, etc., se ha convertido en una librería muy importante para procesar datos a alto nivel en Python (es decir, estadísticas). Pandas fue diseñada originalmente para gestionar datos financieros, y como alternativo al uso de hojas de cálculo (es decir, Microsoft Excel).

Los principales tipos de datos que pueden representarse con pandas son:

- Datos tabulares con columnas de tipo heterogéneo con etiquetas en columnas y filas.
- Series temporales

Para poder manipular los datos, se utilizarán herramientas presentes en los arreglos, series, dataframes ya vistos. En el proceso de conocer y manipular datos es importante determinar algunos valores estadisticos que ya sabemos como calcular

La media, mediana, desviacion, valor min/max ...

Ademas es importante visualizar la información, para ello utilizamos matplotlib de la siguiente manera:

```
import matplotlib.pyplot as plt

plt.figure()

plt.plot(x, y, style) #donde x, y contienen la data,

plt.show() #style: el estilo y color de línea
```

En el análisis de datos, la derivada y la integral son valores importante para ciertos propósitos, por los tanto debe calcularse de manera numérica