PRÁCTICA 4: Tipos de Datos Estructurados. Operaciones con tipos de Datos Complejos

PRÁCTICA 5: Creación de Módulos

(tercera y cuarta prácticas evaluables)

(Ejemplo de uso con estructuras del PDB, Protein Data Bank)

El objetivo de la práctica 4 es que el alumno se familiarice con el uso de vectores, matrices y estructuras de datos. Los temas cubiertos en esta parte son:

- Cadenas de caracteres.
- Listas, Tuplas y Diccionarios.
- Vectores y matrices
- Visualización de datos (gráficas).

El objetivo de la práctica 5 es que el alumno se familiarice con la creación de módulos (funciones). Los temas cubiertos en esta parte son:

- Creación de Funciones.
- Integración y Uso de Funciones en Programas.

El alumno utilizará las transparencias de teoría, así como la ayuda integrada en el entorno como guía para resolver la relación de ejercicios propuesta para ambas prácticas.

Instrucciones para la realización de la práctica:

- **La práctica durará 15h (aproximadamente), es decir, se trabajará en clase durante 4 semanas de prácticas.**
- **Debe entregar a través de PRADO todos los ejercicios que termine correctamente.**
- **Fecha límite de entrega de los ejercicios en PRADO: Domingo 26 de Noviembre a las 23:59 hrs.**
- Todos los ejercicios deben realizarse con el **editor de Spyder**, en el correspondiente fichero-py con nombre **P4_<num_ejercicio>.py o P5_<num_ejercicio>.py.**
- **Es imprescindible que Ud. haga y entienda bien cada ejercicio antes de entregarlo.** Debe estar preparado/a para que, después de la entrega, **el profesor le pida que le explique algún detalle del ejercicio entregado.**
- **En las clases de prácticas del 28-N y 30-N (P5) se realizará una actividad de evaluación continua individual relacionada con estos ejercicios.**
- **La nota de la práctica se determinará en función del número de ejercicios puntuables entregados correctamente y la puntuación obtenida en la actividad de evaluación**

continua individual. La puntuación total de la práctica (sobre 10 puntos) se agregará con la obtenida en las demás prácticas puntuables.

- **Si lo desea, también puede entregar los ejercicios adicionales no puntuables, que serán tenidos en cuenta como nota extra de participación en clase.**

EJERCICIOS PUNTUABLES PARA LA PRÁCTICA

En esta práctica trabajaremos con información estructural de proteínas, y en particular, con datos de la base de datos PDB. PDB es un repositorio de descripciones atómicas de proteínas y otras macromoléculas. En particular, PDB almacena las estructuras atómicas de estas macromoléculas determinadas por técnicas experimentales como cristalografía de rayos X, espectrometría de resonancia magnética nuclear, y otras. En este enlace podrás encontrar recursos educativos para utilizar PDB: https://pdb101.rcsb.org/learn/guide-to-understanding-pdb-data/introduction

El **objetivo** de la práctica es el tratamiento automático de datos de PDB para realizar análisis y representaciones gráficas de proteínas. Para ello, será necesario leer la información desde los ficheros correspondientes almacenados en PDB, volcar esta información en estructuras de datos en memoria (en vectores, matrices, listas o tuplas, según corresponda) y trabajar con estas estructuras de datos para realizar cálculos y representaciones gráficas.

Para realizar este trabajo, te proponemos seguir una serie de pasos:

1.- Aprende a interpretar y leer los ficheros PDB. El primer paso será familiarizarte con el formato de los ficheros PDB. En el fichero "formatoPDB.pdf" disponible en PRADO se puede consultar dicho formato (explicación en español al final del texto). La web:
https://pdb101.rcsb.org/learn/guide-to-understanding-pdb-data/dealing-with-coordinates
también permite familiarizarte con la estructura de estos ficheros. Chequea estos recursos junto con el contenido del fichero "1clg.pdb" para comprender dicho formato. Es necesario extraer la información con la que se va a trabajar del fichero según lo que pida cada ejercicio, y también será necesario descargar estos ficheros en la carpeta de trabajo de Spyder.

En nuestro caso, los análisis que vamos a realizar requieren diferenciar entre los átomos de las distintas cadenas (o secuencias) de la proteína. ¿Sabes cómo se indica en los ficheros PDB dónde empieza y termina cada cadena de la proteína?.

Una forma de obtener los datos es cogerlos directamente *a mano* del fichero PDB de la proteína, pegando la parte correspondiente en Excel, separando por columnas y sacando las columnas correspondientes a las coordenadas de cada átomo. Sin embargo, esto implica que para trabajar con cada nueva proteína hay que generar los ficheros correspondientes *a mano* (OMG!).

Elige tu reacción aquí (varias respuestas son correctas):

Otra forma sería cargar automáticamente toda la información desde el fichero a la memoria del ordenador (en variables de distintos tipos: matrices, vectores/listas, tuplas, etc) y luego extraer lo que se necesite en cada momento. Para ello haremos uso de funciones que ya existen en *biopython (PDBParser y parser.get_structure)*, en un código como el de la figura siguiente:

```
import Bio.PDB as pdb

fichero = input("Nombre del fichero de la proteína (sin extensión): ")
fichero += '.pdb'
parser = pdb.PDBParser()
try:
    s = parser.get_structure('my_pdb', fichero)
except:
    s = parser.get_structure('my_pdb', '1clg.pdb') #fichero por defecto

print("CABECERA")
print(s.header)
print("\n\nNOMBRE DE LA PROTEÍNA:")
print(s.header['head'])
print("\n\nKEYWORDS:")
print(s.header['keywords'])
print("\n\nCADENAS QUE CONTIENE:")
print(s.header['compound']['1']['chain'])
```

Vea lo que sale por pantalla. En la variable **s** se encuentra toda la información de la proteína que se acaba de leer del fichero, explórela haciendo doble clic sobre esta variable en la ventana "Variable Explorer" de Spyder.

Para acceder a la información de las coordenadas de los átomos, tenemos que utilizar el siguiente código (más información aquí: https://biopython.org/wiki/The_Biopython_Structural_Bioinformatics_FAQ):

```
for model in s:
    for chain in model:
        for residue in chain:
            for atom in residue:
                print(list(atom.get_coord()))
```

⇒Modifica el código anterior para que, en vez de mostrarse las coordenadas de los átomos por pantalla, se almacenen en forma de matriz, es decir [[x1, y1, z1], [x2, y2, z2], … [xn, yn, zn]], en una variable llamada **ats_global**.

⇒Añade sobre el código anterior las líneas necesarias para crear una nueva variable **ats_info,** que contenga una lista de listas con la siguiente información: el **nombre del átomo** correspondiente y

su ***id***. Tenga en cuenta la existencia de los siguientes métodos que se pueden aplicar sobre cada átomo:

```
>>> a.get_name()        # atom name (spaces stripped, e.g. "CA")
>>> a.get_id()          # id (equals atom name)
>>> a.get_coord()       # atomic coordinates
>>> a.get_vector()      # atomic coordinates as Vector object
>>> a.get_bfactor()     # isotropic B factor
>>> a.get_occupancy()   # occupancy
>>> a.get_altloc()      # alternative location specifier
>>> a.get_sigatm()      # standard deviation of atomic parameters
>>> a.get_siguij()      # standard deviation of anisotropic B factor
>>> a.get_anisou()      # anisotropic B factor
>>> a.get_fullname()    # atom name (with spaces, e.g. ".CA.")
```

⇒Añade una variable llamada **ats_chains** de tipo lista que contenga una lista de matrices de coordenadas, iguales a las que has guardado en **ats_global**, pero con una matriz por cada cadena o secuencia de la proteína de manera separada. Cada una de esas matrices contendrán solo las coordenadas de los átomos de una misma cadena.

2.- Con todo lo anterior, crea un programa que lea una proteína desde un fichero indicado por el usuario y muestre por pantalla las variables ats_global, ats_info y ats_chain para esta proteína. A continuación, el programa imprimirá un menú con las opciones '1. Cambiar de proteína' y '2. Terminar'. La opción 1, permite cambiar la proteína y realizar los cálculos de las variables para la nueva proteína. El objetivo de este programa es poder cambiar la proteína "protagonista" tantas veces como se quiera de manera repetitiva.

3.- Las distancias existentes entre un conjunto de elementos en el espacio se puede representar mediante una matriz simétrica N x N, donde M[i][j] indica la distancia de la ruta directa entre los elementos i y j. Estos valores i, j serán valores enteros y naturalmente $0 \leq i,j < N$. Un valor M[i][j] = 0 indica que no existe ruta directa entre los elementos i,j. Las distancias entre elementos (átomos en nuestro caso) se pueden calcular mediante distancia euclídea de las coordenadas x, y, z de dichos elementos.
Incluya el código necesario sobre el ejercicio anterior para que se **calcule la matriz de distancias a partir de las posiciones x,y,z almacenadas en la matriz ats_global** (es decir, a partir de la matriz que contiene todas las coordenadas), **cada vez que se lea una nueva proteína mediante la opción 1**. Este ejercicio se debe hacer en un script distinto al del ejercicio anterior.

4.- Una vez que la matriz de distancias ha sido creada correctamente, dicho programa debe realizar las siguientes operaciones a seleccionar mediante un menú:

Átomos a menos de d Angstroms de uno dado: Dado un átomo *i* y una distancia *d*, informar de cuáles son los átomos que se encuentran a menos de *d Angstroms* de *i*. Para ello guarde el número de fila en la que se encuentra la información de cada átomo que cumpla con dicha condición en un

vector llamado *cercanos* y utilícelo para ofrecer toda la información disponible en **ats_info** sobre cada átomo separada por tabuladores (incluida la del átomo *i* indicado).

Distancias por debajo de un umbral: Obtenga un listado de cada par de átomos cuya distancia se encuentre por debajo de un umbral. Cada par de átomos (valores i, j) debe ser guardado como una fila en una nueva matriz, que luego se mostrará por pantalla. De esta forma, si la distancia entre los átomos 3 y 5 es menor o igual al umbral indicado, quedará almacenado 3 y 5 como valores para las dos columnas de la fila correspondiente, generando una matriz de nombre *distumbral* con la siguiente información:

3 5
... ...
... ...

Para ayudar al usuario a determinar un umbral adecuado, indíquese la distancia mínima (sin considerar los ceros) y la máxima entre cualquier par de átomos antes de pedirle dicho umbral.

Por lo tanto, el menú debe quedar con las siguientes opciones '1. Cambiar de proteína', '2. Átomos a menos de d Angstroms de uno dado', '3. Distancias por debajo de un umbral' y '4. Terminar'. Este ejercicio se debe hacer en un script distinto al del ejercicio anterior.

5.- Utilice la función *scatter3D* y *plot* para pintar gráficamente los puntos en el espacio y los resultados según se indica a continuación:

5.1 Pintar los átomos para cada cadena en distintos colores *(Inclúyalo como una nueva opción en el menú del apartado anterior)*.

Utilice scatter3D para pintar con círculos de distintos colores (con relleno en negro) las distintas cadenas leídas desde el fichero (disponibles en ***ats_chains***). Como podría haber más o menos cadenas se debe montar un bucle que pinte cada serie contenida en ***ats_chains***, según las submatrices que haya en dicha lista (una submatriz en cada posición de la lista).

Cree la figura de la siguiente manera y ajuste en su máquina el tamaño de figura más adecuado (`figsize`), la calidad de la misma o puntos por pulgada (`dpi`), y evitando que le ponga un marco (`frameon`):

```
fig = plt.figure(figsize = (8,8), dpi = 175, frameon = False)
ax = fig.gca(projection='3d')
```

En este caso, como la escala de los ejes importa, vamos a calcular para todos los ejes sus límites inferior y superior, de manera que podamos fijarlos igual para cada uno de ellos (ejes X, Y, Z). Calcule para ello el mínimo valor posible en X, Y y Z, así como el máximo (guárdelos en las variables `min_x`, `max_x`, `min_y`, `max_y`, `min_z`, `max_z`, según corresponda). Para calcular los máximos y mínimos se aconseja transformar ***ats_global*** al tipo `array` de numpy (**atgl** =

`np.array`..., por ejemplo), ya que las funciones `np.min` y `np.max` le pueden ayudar a calcular tanto `minval` como `maxval` para cada dimensión de manera muy sencilla. A continuación, fije los límites de los ejes así (rellenando los huecos que faltan según lo indicado):

```
min_x =
max_x =
min_y =
max_y =
min_z =
max_z =
ax.set_xlim(min_x, max_x)
ax.set_ylim(min_y, max_y)
ax.set_zlim(min_z, max_z)
```

Utilice la siguiente estructura para pintar las cadenas:

```
edcol = 'brgmykcbrgmykcbrgmykc'
col   = 'kkkkkkkwwwwwwwmmmymmm'
i = 0 #para ir cambiando el color de cada cadena
for    in     :
      x =
      y =
      z =
      ax.scatter3D(x, y, z, c=col[i], edgecolors=edcol[i],
            marker='o', s=5, depthshade=False)
      i += 1 #i = i + 1
```

Para rellenar los huecos en blanco se aconseja igualmente transformar **ats_chains** al tipo `array` (ya que para cada submatriz será muy fácil extraer su 1ª, 2ª y 3ª columnas, es decir, nuestros x, y, z). Recuerde usar `plt.show()` al final para que finalmente muestre la figura.

5.2 Distancias por debajo de un umbral: Una vez realizados los cálculos indicados en el ejercicio 4 de la práctica, con el comando `plot` pinte con puntos en azul la serie 2D correspondiente a los átomos contenidos en `distumbral` (tomando como valores en X la primera columna del vector, y como valores en Y la segunda columna).

EJERCICIOS PUNTUABLES PARA LA PRÁCTICA 5

1. Implementar una función llamada `matriz_distancias` que reciba la matriz completa de átomos `ats_global` y devuelva la correspondiente matriz de distancias, de manera que pueda llamarse de la siguiente manera:
   ```
   d = matriz_distancias(ats_global)
   ```

2. Convierta el código de la parte del ejercicio 5 de la práctica anterior que pintaba la proteína en una **función** que reciba `ats_chains`, de manera que pueda llamarse de la siguiente manera:
   ```
   pinta_proteina(ats_chains)
   ```

Reemplace las instrucciones correspondientes por las llamadas a estas funciones en su programa final de la práctica 4.