**SECUENCIA DE ADN**

Dentro del campo de conocimiento de la computación biomédica se encuentra el apoyo a actividades dentro de la biología. Dentro de esta, el campo de la genética se ha visto ampliamente beneficiada del área de la computación, permitiendo realizar análisis diversos con las diferentes secuencias de ADN. Existen algunas funcionalidades que se implementan reiterativamente sobre estas secuencias, por lo que para aumentar el rendimiento en que se ejecutan estas tareas implementar una generalización de estas es pertinente.

En este caso se pide implementar una generalización de una secuencia de ADN por medio de la programación orientada a objetos. Esta generalización debe permitir que se implementan las siguientes funcionalidades:

* Decir si la secuencia suministrada contiene bases nitrogenadas correctas.
* Encontrar la secuencia complementaria.
* Cargar un archivo .txt donde está almacenada la secuencia de ADN a analizar.
* Contar el número de bases nitrogenadas o elementos de la secuencia.

**Consideraciones**

Como información para resolver el ejercicio se menciona que la forma más simple de determinar una secuencia de ADN es por medio de un conjunto de letras como se muestra a continuación: AJUEYRNVKSOSKAMJFKALSDKFJ. La idea es entonces generar archivos .txt donde estén estas secuencias para luego ser leídas en la implementación con Python. Se debe tener en cuenta, como simplificación, que las únicas letras relevantes para una secuencia de ADN son: A, T, C y G; por lo tanto, si una secuencia ingresada contiene una letra por fuera de este grupo, esta no será válida como una secuencia de ADN.

Como comentarios finales, las letras A, T, C y G representan bases nitrogenadas, las cuales son: Adenina, Timina, Citocina y Guanina, respectivamente. Y estás se asociación con su secuencia complementaria de la siguiente forma:

Adenina 🡨🡪 Timina

Guanina 🡨🡪 Citocina

**Solución**

Para la solución de los lineamientos propuestos inicialmente se mostrarán los atributos de la clase que se usarán, además para qué son usados.



|  |  |
| --- | --- |
| self.\_\_secuencia | Captura la secuencia leída desde el archivo .txt |
| self.secComplementaria | Captura la secuencia complementaria de la secuencia leída |
| self.\_\_valida | Atributo booleano que califica si la secuencia leída solo contiene letras con las bases nitrogenadas válidas |
| self.\_\_numElementos | Diccionario donde sus llaves son las letras encontradas en la secuencia y el contenido al que apunta la llave es la cantidad de esa letra en la secuencia. |
| self.\_\_basesValidas | Atributo que se emplea para capturar cuáles son las letras que equivalen a una base nitrogenada. |
| self.\_\_conversionBases | Diccionario que establece la conversión letra a letra que se debe realizar para generar la secuencia complementaria |

A continuación, se mostrarán los métodos implementados para la clase:

**Método 1:**



El método mostrado en el anterior código tiene como finalidad leer la secuencia de letras encontrada en un archivo .txt, este método se debe realizar antes que cualquier otro método presente en la clase.

Durante la implementación de este se emplea el uso de la estructura try, que evalúa lo que está resaltado en amarillo. Si por algún error se genera una excepción, entonces se ejecuta lo que está resaltado en azul. En las secuencias dentro del try se usa la estructura with, la cual se recomienda emplear cuando se trabaja con lectura de archivos, ya que este se debe cerrar al finalizar, tarea que se realiza automáticamente cuando se deja de ejecutar la estructura with. En este caso entonces el objeto que retorna la función open de Python es con el que se trabajará y este lleva el nombre de archivo. De este se usa el método read y lo que retorna se almacena en el atributo secuencia, luego se reinicializan los otros atributos, ya que al objeto se le está asociando una nueva secuencia. En el caso que suceda una excepción se mostrará que la ruta es incorrecta.

**Método 2, 3 y 4**



El método 2 genera la secuencia complementaria de la secuencia actualmente leída. Inicialmente se verifica si la secuencia es válida, para esto se llama el método esValida (Método 5) el cual retorna True si la secuencia solo contiene letras asociadas a una base nitrogenada válida, o False si no. Luego implementa la siguiente línea:

''.join([self.\_\_conversionBases[base] for base in self.\_\_secuencia])

Esta línea realiza varias cosas: Inicialmente se usa ‘’.join(A). Join es un método dentro de los string, este lo que hace es unir los elementos de una lista de strings, generando un nuevo string. El string que va entre los elementos de la lista unidos es el elemento sobre el que se aplica el método. Por ejemplo: ‘ooo’.join([‘hola’, ‘chao’, ‘Bienvenido’]) generaría ‘holaooochaooooBienvenido’.

Ahora bien, la lista que se está ingresando en este caso es: [self.\_\_conversionBases[base] for base in self.\_\_secuencia]; este tipo de secuencia se llama función generadora, y en este caso se usa para generar listas. Una forma general de entender cómo funciona este tipo de instrucción es: cuando se ejecuta [evaluación(Elemento) for Elemento in listaRecorrida] el resultado es una lista donde cada posición equivale a evaluación(Elemento), donde Elemento es cada valor recorrido en listaRecorrida.

Por ejemplo: [elemo\*2 for elemo in range(10)] genera [0, 2, 4, 6, 8, 10, …, 18]. Por lo que una forma de asociar este tipo de funciones generadoras con secuencias más básicas sería:



Lo interesante es que estas funciones generadoras pueden ser más complejas, incluyendo funciones lambda y condicionales.

En este caso entonces la secuencia ''.join([self.\_\_conversionBases[base] for base in self.\_\_secuencia]) generaría la secuencia complementaria, ya que los elementos que se unen por medio del método join son el resultado de self.\_\_conversionBases[base] y cada valor de base equivale a la base correspondiente en self.\_\_secuencia, que es la lista que está recorriendo.

Con respecto al método 3 y 4 se emplean solo para retornar el valor de la secuencia y de la secuencia complementaria.

**Método 5**



Este método inicialmente verifica que haya una secuencia cargada, luego implementa una serie de condicionales para determinar si la secuencia es válida. La línea importante es la que está resaltada en amarillo, en esta se usa la función de Python all. Esta cuando es aplicada sobre una lista de valores booleanos retorna True si todos los elementos de esta son True.

En este caso la lista de valores booleanos se genera por medio de: [base in self.\_\_basesValidas for base in set(self.\_\_secuencia)], donde los valores booleanos salen de base in self.\_\_basesValidas; instrucción donde se valida si la variable base (quien itera sobre la secuencia original) se encuentra en el atributo que almacena las bases nitrogenadas válidas.

**Método 6**



Este método consta de dos partes. Inicialmente se llenan las llaves del diccionario, almacenando este todas las diferentes letras disponibles en la secuencia. Luego se cuentan estas.

Para realizar la primera etapa se usa:

\_ = [self.\_\_numElementos.update({nuevaLlave: 0}) for nuevaLlave in set(self.\_\_secuencia)]

En este caso la función generadora no se emplea para crear una nueva lista, si no que se emplea como una forma de resumir la sentencia de un for. Como no se va a utilizar la lista generada se usa \_, esto se emplea cuando la variable que se capturaría no va a ser usada. Ahora, para crear una nueva llave en un diccionario dado se usa el método update, quien recibe como argumento un diccionario que dice la llave nueva y hacia quien apunta. En este caso sería entonces {nuevaLlave: 0}. En este caso nuevaLlave es la variable que está iterando sobre set(self.\_\_secuencia) la cual retorna una variable tipo set con todos los caracteres sin repetición presentes en la secuencia cargada, y el valor hacia el que apunta es 0 ya que se está inicializando.

La segunda etapa es contar las letras, esto se hace de una forma básica recorriendo la secuencia y aumentando en uno el contenido del diccionario correspondiente.