2024/25 - PROYECTO PYTHON (IS2)
EXAMEN DEFENSA 3. FECHA: 19/12/2024

APELLIDOS:		NOMBRE:		HORA COMMIT:	

Si tu **ENTREGA 3 está disponible**, se puede ejecutar de manera satisfactoria, y se ajusta a las indicaciones de tu profesor, tu calificación inicial es un **CUATRO**. Al realizar el examen de la defensa de la práctica podrá subir o bajar esta calificación, en concreto:

- SUBE LA CALIFICACIÓN: si el código realizado está bien resuelto.
- BAJA LA CALIFICACIÓN: en caso contrario.

IMPORTANTE: Realiza todo el código en un nuevo módulo defensa_3.py en el mismo proyecto de la entrega 3 (en el paquete que quieras)

Se proporcionan dos ficheros como material de nombres *genes.txt* y *red_genes.txt*.
Utilizando el código que haya subido en la Entrega 3 y, utilizando como inspiración el tipo *Red_social*, en este ejercicio se va a programar una red de genes. En primer lugar, descarga los ficheros y observa la información que proporcionan.

(fichero genes.txt)

```
TP53,supresor tumoral,256,17p13.1
EGFR,oncogen,187,7p12
KRAS,oncogen,92,12p12.1
BRAF,oncogen,75,7q34
PIK3CA,oncogen,112,3q26
…
```

(fichero red_genes.txt)

```
TP53,EGFR,0.5
TP53,KRAS,0.7
BRAF,KRAS,0.8
BRAF,TP53,0.4
PIK3CA,TP53,0.2
…
```

El fichero de *genes.txt* contiene información de un gen en cada línea, donde se puedes observar el nombre del gen, el tipo de gen, el número de mutaciones y la localización en el cromosoma. Por otro lado, en el fichero *red_genes.txt*, en cada línea, se observa el nombre de dos genes y un valor numérico que es el valor de relación entre ambos.

En esta DEFENSA vamos a implementar los tipos *Gen*, *RelacionGenAGen* y *RedGenica*. Los dos primeros serán tipos básicos necesarios para representar vértices y aristas.

Ejercicio 1: Implementación del tipo Gen (BIEN: +1,5 / MAL: -1)

Implementa el **tipo Gen**, que es un tipo inmutable, con propiedades básicas *nombre*, *tipo*, *num_mutaciones* y *loc_cromosoma*, todos de tipo string menos el num_mutaciones que es de tipo entero. Ten en cuenta que el número de mutaciones debe ser mayor o igual que cero.

Añade dos métodos de factoría. El primero, el método *of*, que permita construir un objeto a partir de las propiedades básicas y que gestione las restricciones del tipo. El segundo, de nombre *parse*, recibe una cadena con el formato de las líneas del fichero *genes.txt*.

Es conveniente realizar una pequeña comprobación del método *parse* (por ejemplo, en el *main* del mismo módulo en el que está programando).

Ejercicio 2: Implementación del tipo RelacionGenAGen (BIEN: +1,5 / MAL: -1)

Implementa el **tipo RelacionGenAGen**, que es un tipo inmutable, con propiedades básicas *nombre_gen1*, *nombre_gen2* y *conexion*, siendo las dos primeras string y la tercera un número real entre -1 y 1, ambos inclusive.
Añade dos métodos de factoría. El primero, el método *of*, que permita construir un objeto a partir de las propiedades básicas y que gestione las restricciones del tipo. El segundo, de nombre *parse*, recibe una cadena, con el formato de las líneas del fichero *red_genes.txt*.

Añade dos propiedades **derivadas** al tipo:

- *coexpresados*: devuelve cierto si el valor de conexión es mayor estricto a 0.75
- *antiexpresados*: devuelve cierto si el valor de conexión es menor estricto a 0.75

Es conveniente realizar una pequeña comprobación del método *parse*.

Ejercicio 3: Implementación del tipo RedGenica (BIEN: +3 / MAL: -2)

Este tipo es análogo a la *Red_social* de la entrega. Hereda del tipo *Grafo* utilizando el tipo *Gen* como vértices y *RelacionGenAGen* como aristas.

Esta debe ser la estructura:

```
Class RedGenica(Grafo[Gen, RelacionGenAGen]):
    """
    Representa una red génica basada en Grafo
    """
    def __init__(self, es_dirigido: bool = False) -> None:
        super().__init__(es_dirigido)
        self.genes_por_nombre: Dict[str, Gen] = {}

    @staticmethod
    def of(es_dirigido: bool = False) -> RedGenica:
        """
        Método de factoría para crear una nueva Red Génica.

        :param es_dirigido: Indica si la red génica es dirigida (True)
         o no dirigida (False).
        :return: Nueva red génica.
        """
        …

    @staticmethod
    def parse(f1: str, f2: str, es_dirigido: bool = False) -> RedGenica:
        """
        Método de factoría para crear una Red Génica desde archivos
         de genes y relaciones.

        :param f1: Archivo de genes.
        :param f2: Archivo de relaciones entre genes.
        :param es_dirigido: Indica si la red génica es dirigida (True) o
         no dirigida (False).
        :return: Nueva red génica.
        """
        # Primero, crear la red génica que se va a devolver

        # Segundo, leer y agregar genes

        # Por últimno, leer y agregar relaciones entre genes
```

...

Haz un test con los siguientes pasos:

1.- Crea una red génica no dirigida a partir de los ficheros *genes.txt* y *red_genes.txt*
2.- Aplica un recorrido en profundidad desde el gen *KRAS* hasta el *PIK3CA*
3.- Crea un subgrafo a partir de los vértices del paso 2, y dibújalo de forma que aparezcan los nombres de los genes en cada vértice. El resultado esperado lo tienes más abajo