Disponemos de datos de GitAcme, una plataforma para el alojamiento y la gestión de proyectos de desarrollo software basada en git, similar a GitHub o GitLab, y queremos explotarlos para añadir valor a los servicios proporcionados por la empresa. Los datos están almacenados en el archivo `repositorios.csv` que se le proporciona, cuyas tres primeras líneas se muestran a continuación (la segunda línea, por su longitud, se visualiza en dos líneas, aunque en el archivo solo ocupa una):

```
repositorio, propietario, lenguajes, privado, commits
ProjectX,user123,"Python,JavaScript",True,[b789101#Initial commit#2023-10-06
12:00:00;c567891#Added main functionality#2023-10-06 13:00:00]
MiFirstRepo,newbie,"Go",False,[]
```

Cada línea del archivo CSV (exceptuando la primera) incluye la siguiente información sobre un repositorio: su **nombre**, el **usuario propietario**, una **lista de los lenguajes de programación** utilizados, si el repositorio es **privado** y una **lista con los *commits*** realizados, ordenados cronológicamente. Para cada *commit* se almacena un código alfanumérico como **identificador**, un **mensaje descriptivo**, y la **fecha y hora** en la que se registró. Si el repositorio no tiene *commits*, la lista aparece vacía.

Por ejemplo, la segunda línea del archivo CSV corresponde al repositorio **ProjectX**, cuyo propietario es el usuario **user123**, y en el que se emplean los lenguajes de programación **Python** y **JavaScript**. Este repositorio es **privado** y en él se han registrado **dos commits**: el primero, identificado como **b789101** con el mensaje ***"Initial commit"***, fue realizado el **6 de octubre de 2023 a las 12:00:00**; el segundo, con identificador **c567891** y el mensaje ***"Added main functionality"***, se llevó a cabo el mismo día a las **13:00:00**. Por otro lado, la tercera línea del archivo describe el repositorio **MiFirstRepo**, propiedad del usuario **newbie**, que utiliza el lenguaje de programación **Go**. Este repositorio es **público** y aún no ha registrado ningún *commit*.

Para implementar las funciones que se solicitan utilice **obligatoriamente** las definiciones de las namedtuple que se indican a continuación:

```python
from typing import NamedTuple,List,Set,Tuple,Dict,Optional
from datetime import datetime,date

Commit = NamedTuple("Commit",
       [("id", str), # Identificador alfanumérico del commit
        ("mensaje", str), # Mensaje asociado al commit
        ("fecha_hora", datetime) # Fecha y hora en la que se registró el commit
       ])

Repositorio = NamedTuple("Repositorio",
      [("nombre", str),  # Nombre del repositorio
       ("propietario", str), # Nombre del usuario propietario
       ("lenguajes", Set[str]),  # Conjunto de lenguajes usados
       ("privado", bool),  # Indica si es privado o público
       ("commits", List[Commit])  # Lista de commits realizados
       ])
```

Se pide realizar un proyecto Python con la siguiente estructura: una carpeta `data` en la que incluya el archivo `repositorios.csv` y una segunda carpeta `src` en la que se incluyan los módulos `repositorios.py` y `repositorios_test.py`. Las funciones de los ejercicios que se solicitan a continuación se deben incluir en

el módulo `repositorios.py` mientras que las funciones de testde estos ejercicios deben incluirse en el módulo `repositorios_test.py`.

Tenga en cuenta que se pueden definir funciones auxiliares cuando se considere necesario. Las puntuaciones indicadas para cada ejercicio incluyen la realización de dichos tests, lo que implica que no podrá obtener la máxima puntuación en un ejercicio si no ha implementado el método test con los parámetros adecuados y este da los resultados esperados para las pruebas que se indican en **cada ejercicio.**

EJERCICIO 1 - (1,75 puntos)

`lee_repositorios`: dada una cadena de texto con la ruta de un fichero *CSV*, devuelve una lista de tuplas de tipo Repositorio con la información contenida en el fichero.

```
def lee_repositorios(csv_filename: str) -> List[Repositorio]
```

Antes de implementar la función anterior, implemente la función auxiliar que se describe a continuación:

- **`parsea_commits`**: dada una cadena de texto con los identificadores, los mensajes y las fechas/horas de una lista de *commits*, devuelve la información como una lista de tuplas de tipo **Commit**. Un ejemplo de esta cadena puede ser:

  `"[b789101#Initial commit#2023-10-06 12:00:00;c567891#Added main functionality3#2023-10-06 13:00:00]"`

  Note que el formato esperado de la cadena recibida como parámetro separa la información de cada *commit* por punto y coma (;). Además, para cada *commit*, el identificador, el mensaje y la fecha/hora están separados por #. Si la cadena recibida es "`[]`", la función devuelve una lista vacía.

  Ayuda: Para "parsear" las fechas use el formato `"%Y-%m-%d %H:%M:%S"`

  ```
  def parsea_commits(commits_str: str) -> List[Commit]
  ```

Salidas de tests:

```
Se han leido 31 repositorios
Los dos primeros son
        -->Repositorio(nombre='ProjectX', propietario='user123',
lenguajes={'Python', 'JavaScript'}, privado=True, commits=[Commit(id='b789101',
mensaje='Initial commit', fecha_hora=datetime.datetime(2023, 10, 6, 12, 0)),
Commit(id='c567891', mensaje='Added main functionality',
fecha_hora=datetime.datetime(2023, 10, 6, 13, 0))])
        -->Repositorio(nombre='MiFirstRepo', propietario='newbie',
lenguajes={'Go'}, privado=False, commits=[])

Los dos últimos son
        -->Repositorio(nombre='HealthTracker', propietario='AI_Architect',
lenguajes={'Python', 'R'}, privado=False, commits=[Commit(id='k901234',
mensaje='Setup database', fecha_hora=datetime.datetime(2022, 10, 6, 17, 0))])
        -->Repositorio(nombre='MusicLibrary', propietario='webWizard',
lenguajes={'Ruby'}, privado=True, commits=[Commit(id='l012345', mensaje='Initial
song upload', fecha_hora=datetime.datetime(2024, 10, 6, 18, 0)),
Commit(id='l012346', mensaje='Added playlist feature',
fecha_hora=datetime.datetime(2024, 10, 6, 18, 30))])
```

EJERCICIO 2 - (1,25 puntos)

`total_commits_por_anyo`: dada una lista de tuplas de tipo **Repositorio**, devuelve un diccionario en el que las claves son los años, y los valores el número total de *commits* registrados en el año dado como clave **para los repositorios públicos**.

```
def total_commits_por_anyo(repositorios:List[Repositorio])->Dict[int, int]
```

Salidas de tests:

```
El total de commits por año para los repositorios públicos es
{2023: 20, 2019: 1, 2021: 2, 2022: 2}
```

EJERCICIO 3 - (1,75 puntos)

`n_mejores_repos_por_tasa_crecimiento`: dada una lista de tuplas de tipo **Repositorio** y un número entero **n** (con valor por defecto igual a 3), devuelve una lista con los **n nombres** de los repositorios y sus **tasas de crecimiento** más altas.

```
def n_mejores_repos_por_tasa_crecimiento(repositorios: List[Repositorio],
                                         n:Optional[int]=3)->List[Tuple[str,float]]
```

Antes de implementar la función anterior, implemente la función auxiliar que se describe a continuación:

- **`calcular_tasa_crecimiento`**: dado un **Repositorio,** devuelve su tasa de crecimiento. La **tasa de crecimiento** de un repositorio se define como la **ratio** entre el número de *commits* y el número de días transcurridos entre el primer y el último *commit (los commits están ordenados en el archivo)*. Si el repositorio tiene menos de 2 *commits*, o si tiene 2 o más *commits* pero no ha pasado al menos un día entre el primero y el último, la tasa se considera cero.

  ```
  def calcular_tasa_crecimiento(repositorio: Repositorio) -> float
  ```

  Ayuda: Los días transcurridos entre dos variables dt1 y dt2, de tipo `datetime`, se pueden calcular como `(dt2-dt1).days`.

Salidas de tests:

```
Los 3 mejores repositorios según su tasa de crecimiento son
       -->('OpenSourceNotes', 0.005479452054794521)
       -->('ProjectPhoenix', 0.00546448087431694)
       -->('LAB-Calificaciones', 0.0027397260273972603)
```

EJERCICIO 4 - (2,5 puntos)

`recomendar_lenguajes`: dada una lista de tuplas de tipo **Repositorio** y un repositorio específico, devuelve **un conjunto** con **los lenguajes de programación recomendados** para dicho repositorio. Los lenguajes recomendados son aquellos que se usan en repositorios similares al repositorio dado. Se considera que un repositorio es similar al dado si comparte al menos uno de los lenguajes de programación con el repositorio dado. Por ejemplo, si queremos hacer una recomendación para el repositorio "LAB-FP", cuyo lenguaje es Java, podemos considerar similar el repositorio "LAB-Calificaciones", que utiliza Java y Python, ya que ambos

comparten el lenguaje Java. En este caso, se recomendaría Python como nuevo lenguaje para el repositorio "LAB-FP".

```
def recomendar_lenguajes (repositorios:List[Repositorio],
                          repositorio:Repositorio)->Set[str]
```

Salidas de test:

Nota.- Para realizar estos tres tests se usan los repositorios primero, quinto y último de la lista de repositorios.

```
Para el repositorio ProjectX que usa los lenguajes {'JavaScript', 'Python'} se
recomiendan
{'SQL', 'Go', 'CSS', 'Java', 'Ruby', 'Julia', 'HTML', 'C++', 'R'}

Para el repositorio MusicLibrary que usa los lenguajes {'Ruby'} se recomiendan
{'JavaScript', 'CSS'}

Para el repositorio MyProject que usa los lenguajes {'HTML', 'JavaScript'} se
recomiendan
{'Ruby', 'Java', 'CSS', 'C++', 'Python'}
```

EJERCICIO 5 - (2,75 puntos)

`media_minutos_entre_commits_por_propietario`: dada una lista de tuplas de tipo **Repositorio**, una fecha inicial y una fecha final (ambas opcionales con valor por defecto **None**), devuelve un diccionario en el que las claves son los nombres de los propietarios de los repositorios, y los valores la media de minutos entre los *commits* realizados en los repositorios de cada propietario dentro del intervalo de fechas dado por [fecha_ini, fecha_fin). Si fecha_ini es None no se restringe el inicio del intervalo, y si fecha_fin es None, no se limita el final del intervalo. Si ambas fechas son None, se consideran todos los *commits* sin restricción temporal.

Es importante tener en cuenta que un mismo propietario puede tener varios repositorios, por lo que los cálculos abarcarán todos los *commits* realizados en los repositorios de ese propietario dentro del intervalo especificado.

```
def media_minutos_entre_commits_por_usuario (repositorios:List[Repositorio],
                          fecha_ini:Optional[date]=None,
                          fecha_fin:Optional[date]=None)->Dict[str, float]:
```

Antes de implementar la función anterior, implemente la función auxiliar que se describe a continuación:

- **`media_minutos_entre_commits`**: recibe una lista de tuplas de tipo **Commit,** y devuelve la media de minutos entre cada dos *commits* consecutivos en el tiempo, por lo que, previamente, deberá ordenar dichos commits . Si la lista tiene menos de dos elementos, la función devolverá None.

  ```
  def media_minutos_entre_commits(lista_commits: List[Commit]) -> float
  ```

  Ayuda: Los minutos transcurridos entre dos variables dt1 y dt2, de tipo datetime, se pueden calcular como `(dt2-dt1).total_seconds()/60.`

Salidas de tests:

```
Media de minutos entre commits para las fechas fecha_ini:None fecha_fin:None
        -->('user123', 60.0)
        -->('developerA', 1.0)
```

```
        -->('coderX', 1.0)
        -->('ai_expert', 1.0)
        -->('gamer', 10.0)
        -->('data_scientist', 1.0)
        -->('cloud_eng', 262785.5)
        -->('student1', 1051260.0)
        -->('dataWhiz', 2103840.0)
        -->('devSamurai', 131760.0)
        -->('AI_Architect', 525420.0)
        -->('techSavvy', 525615.0)
        -->('webWizard', 526130.0)
        -->('sysAdminPro', 60.0)
        -->('quantDev', 100.0)
        -->('codeNinja99', 527070.0)
Media de minutos entre commits para las fechas fecha_ini:None fecha_fin:2021-
12-31
        -->('cloud_eng', 30.0)
Media de minutos entre commits para las fechas fecha_ini:2023-09-01
fecha_fin:None
        -->('user123', 60.0)
        -->('developerA', 1.0)
        -->('coderX', 1.0)
        -->('ai_expert', 1.0)
        -->('gamer', 10.0)
        -->('data_scientist', 1.0)
        -->('cloud_eng', 1.0)
        -->('devSamurai', 131760.0)
        -->('techSavvy', 30.0)
        -->('sysAdminPro', 60.0)
        -->('quantDev', 100.0)
        -->('codeNinja99', 527070.0)
        -->('webWizard', 30.0)
Media de minutos entre commits para las fechas fecha_ini:2023-01-01
fecha_fin:2023-11-01
        -->('user123', 60.0)
        -->('developerA', 1.0)
        -->('coderX', 1.0)
        -->('ai_expert', 1.0)
        -->('gamer', 10.0)
        -->('data_scientist', 1.0)
        -->('cloud_eng', 1.0)
        -->('devSamurai', 60.0)
        -->('techSavvy', 30.0)
        -->('sysAdminPro', 60.0)
        -->('quantDev', 100.0)
```