Ejercicio 1

```
def lee_repositorios(csv_filename: str) -> List[Repositorio]:
    repositorios = []
    with open(csv_filename,  encoding='utf-8') as csvfile:
        reader = csv.reader(csvfile)
        next(reader)
        for nombre, propietario, lenguajes, privado, commits in reader:
            commits = parse_commits(commits)
            lenguajes = parse_lenguajes(lenguajes)
            privado = parse_bool(privado)
            repositorio = Repositorio(
                nombre, propietario, lenguajes, privado, commits)
            repositorios.append(repositorio)
    return repositorios

def parse_lenguajes(lenguajes_str: str) -> Set[str]:
    return set(lenguajes_str.split(','))

def parse_bool(privado_str: str) -> bool:
    res = None
    privado = privado_str.lower()
    if privado == "true":
        res = True
    elif privado == "false":
        res = False
    return res

def parse_commits(commits_str: str) -> List[Commit]:
    commits_str= commits_str.replace("[","").replace("]","").strip()
    res = []
    if len(commits_str) > 0:
        commits_list = commits_str.split(";")
        res = [parse_commit(commit) for commit in commits_list]
    return res

def parse_commit(commit_str: str) -> Commit:
    id, mensaje, fecha_hora_str = commit_str.split('#')
    return Commit(
        id = id.strip(),
        mensaje = mensaje.strip(),
        fecha_hora = parse_fecha_hora(fecha_hora_str)
    )

def parse_fecha_hora(fecha_hora_str: str) -> datetime:
    return datetime.strptime(fecha_hora_str, "%Y-%m-%d %H:%M:%S")
```

Test

```
def test_lee_repositorios(repos_list: List[Repositorio]) -> None:
    print("test_lee_repositorios", "="*80)
    print(f"Se han leido {len(repos_list)} repositorios")
    print("Los dos primeros son")
    mostrar_iterable(repos_list[:2])
    print("Los dos últimos son")
    mostrar_iterable(repos_list[-2:])

if __name__== "__main__":
    repos_list = lee_repositorios(r"data\repositorios.csv")
    test_lee_repositorios(repos_list)
```

Ejercicio 2

```
def total_commits_por_anyo(repositorios: List[Repositorio]) -> Dict[int, int]:
    res = defaultdict(int)
    for repo in repositorios:
        if not repo.privado:
            for commit in repo.commits:
                res[commit.fecha_hora.year] += 1
    return res
```

Alternativa 1:

```
def total_commits_por_anyo(repositorios: List[Repositorio]) -> Dict[int, int]:
    gen = (commit.fecha_hora.year \
                for repo in repositorios \
                    if not repo.privado \
                       for commit in repo.commits)
    return Counter(gen)
```

Alternativa 2:

```
def total_commits_por_anyo(repositorios: List[Repositorio]) -> Dict[int, int]:
    res = dict()
    for repo in repositorios:
        if repo.privado == False:
            for commit in repo.commits:
                if commit.fecha_hora.year not in res:
                    res[commit.fecha_hora.year] = 0
                res[commit.fecha_hora.year] += 1
    return res
```

Test

```
def test_total_commits_por_anyo(repos_list: List[Repositorio]) -> None:
    print("test_total_commits_por_anyo", "="*80)
    print("El total de commits por año para los repositorios públicos es")
```

```
    res = total_commits_por_anyo(repos_list)
    mostrar_dicc (res)

if __name__== "__main__":
    repos_list = lee_repositorios(r"data\repositorios.csv")
    test_total_commits_por_anyo(repos_list)
```

Ejercicio 3

```
def n_mejores_repos_por_tasa_crecimiento(repositorios: List[Repositorio], n: int=3)\
        -> List[Tuple[str, float]]:
    return sorted(((repo.nombre, calcular_tasa_crecimiento (repo)) \
                   for repo in repositorios),
                   key=lambda t:t[1], reverse=True) [:n]

def calcular_tasa_crecimiento(repositorio: Repositorio) -> float:
    dias = calcular_dias_entre_commits(repositorio.commits)
    tasa = 0
    if dias > 0:
        tasa = len(repositorio.commits) / dias
    return tasa

def calcular_dias_entre_commits(commits: List[Commit]) -> int:
    res = 0
    if (len(commits) > 1):
        res = (commits[-1].fecha_hora - commits[0].fecha_hora).days
    return res
```

Alternativa:

```
def n_mejores_repos_por_tasa_crecimiento(repositorios: List[Repositorio], \
        n: int=3) -> List[Tuple[str, float]]:
    res = list()
    for repo in repositorios:
        res.append((repo.nombre, calcular_tasa_crecimiento(repo)))
    return sorted(res, key=lambda t: t[1], reverse=True) [:n]
```

Test

```
def test_n_mejores_repos_por_tasa_crecimiento(repos_list: List[Repositorio], \
        n: int=3) -> None:
    print("test_n_mejores_repos_por_tasa_crecimiento", "="*80)
    print(f"Los {n} mejores repositorios según su tasa de crecimiento son")
    mostrar_iterable(n_mejores_repos_por_tasa_crecimiento(repos_list, n))

if __name__== "__main__":
    repos_list = lee_repositorios(r"data\repositorios.csv")
    test_n_mejores_repos_por_tasa_crecimiento(repos_list, 3)
```

Ejercicio 4

```
def recomendar_lenguajes(repositorios: List[Repositorio], repositorio:Repositorio)\
        -> Set[str]:
    res = set()
    for repo in repositorios:
        if repo.nombre != repositorio.nombre: # No comparar consigo mismo
            if (son_similares(repositorio, repo)):
                res = res.union(repo.lenguajes - repositorio.lenguajes)
    return res

def son_similares(repositorio1: Repositorio, repositorio2: Repositorio) -> bool:
    existe = False
    for lenguaje in repositorio1.lenguajes:
        if lenguaje in repositorio2:
            existe = True
            break
    return existe
```

Alternativa:

```
def son_similares(repositorio1: Repositorio, repositorio2: Repositorio) -> bool:
    lenguajes_comunes = repositorio1.lenguajes.intersection(repositorio2.lenguajes)
    return len(lenguajes_comunes) > 0
```

Test

```
def test_recomendar_lenguajes (repos_list: List[Repositorio], \
        repositorio: Repositorio) -> None:
    print("test_recomendar_lenguajes", "="*80)
    print(f"Para el repositorio {repositorio.nombre} que usa los lenguajes {reposito
rio.lenguajes} se recomiendan")
    print(recomendar_lenguajes(repos_list, repositorio))

if __name__== "__main__":
    repos_list = lee_repositorios(r"data\repositorios.csv")
    test_recomendar_lenguajes(repos_list, repos_list[0])
    test_recomendar_lenguajes(repos_list, repos_list[-1])
    test_recomendar_lenguajes(repos_list, repos_list[4])
```

Ejercicio 5

```
def media_minutos_entre_commits_por_propietario(repositorios: \
        List[Repositorio], fecha_ini: Optional[date]=None, \
        fecha_fin: Optional[date]=None) -> Dict[str, float]:
    d = commits_por_usuario_entre_fechas(repositorios, fecha_ini, fecha_fin)
    res = dict()
    for usuario, commits in d.items():
        if len(commits) > 1:
```

```
            res[usuario] = media_minutos_entre_commits(commits)
    return res

def commits_por_usuario_entre_fechas(repositorios: List[Repositorio], \
        fecha_ini: Optional[date]=None, fecha_fin: Optional[date]=None) \
        -> Dict[str, List[Commit]]:
    res = defaultdict(list)
    for repo in repositorios:
        for commit in repo.commits:
            if commit_entre_fechas(commit, fecha_ini, fecha_fin):
                res[repo.propietario].append(commit)
    return res

def commit_entre_fechas(commit: Commit, fecha_ini: Optional[date], \
        fecha_fin: Optional[date]) -> bool:
    return (fecha_ini is None or fecha_ini <= commit.fecha_hora.date()) \
                    and (fecha_fin is None or commit.fecha_hora.date() < fecha_fin)
```

Alternativa:

```
def commits_por_usuario_entre_fechas(repositorios: List[Repositorio], \
        fecha_ini: Optional[date]=None, fecha_fin: Optional[date]=None) \
        -> Dict[str, List[Commit]]:
    res = dict()
    for repo in repositorios:
        for commit in repo.commits:
            if commit_entre_fechas(commit, fecha_ini, fecha_fin):
                if repo.propietario not in res:
                    res[repo.propietario] = list()
                res[repo.propietario].append(commit)
    return res

def media_minutos_entre_commits(commits: List[Commit]) -> float:
    media = None
    commits_ord = sorted(commits, key=lambda c: c.fecha_hora)
    res = []
    for c1, c2 in zip(commits_ord, commits_ord[1:]):
        dif = (c2.fecha_hora - c1.fecha_hora).total_seconds() / 60
        res.append(dif)
    if len(res) > 0:
        media = statistics.mean(res)
    return media
```

Test

```
def test_media_minutos_entre_commits_por_propietario(repos_list: \
        List[Repositorio], fecha_ini: Optional[date]=None, \
        fecha_fin: Optional[date]=None) -> None:
    print("test_media_minutos_entre_commits_por_usuario", "="*80)
```

```
    print(f"Media de minutos entre commits para las fechas fecha_ini:
{fecha_ini} fecha_fin:{fecha_fin}")
    d = media_minutos_entre_commits_por_propietario(repos_list, fecha_ini,fecha_fin)
    mostrar_iterable(d.items())


if __name__== "__main__":
    repos_list = lee_repositorios(r"data\repositorios.csv")
    test_total_commits_por_anyo(repos_list)
    test_media_minutos_entre_commits_por_propietario(repos_list)
    test_media_minutos_entre_commits_por_propietario(repos_list, \
        fecha_fin = date(2021,12,31))
    test_media_minutos_entre_commits_por_propietario(repos_list, \
        fecha_ini = date(2023,9,1))
    test_media_minutos_entre_commits_por_propietario(repos_list, \
        fecha_ini = date(2023,1,1), fecha_fin = date(2023,11,1))
```