Ejercicio 1 (1,5 ptos)

```
def lee_misiones(ruta_fichero: str) -> list[Mision]:
    misiones = []
    with open(ruta_fichero, encoding="utf-8") as f:
        lector = csv.reader(f, delimiter=";")
        next(lector)  # Saltar cabecera
        for comandante, sistema, controlado, fecha_inicio, fecha_fin, tipo_mision,
                        num_unidades, coste_mensual, recursos in lector:
            controlado = parsea_controlado(controlado)
            fecha_inicio = parsea_fecha(fecha_inicio)
            fecha_fin = parsea_fecha(fecha_fin)
            num_unidades = int(num_unidades)
            coste_mensual = float(coste_mensual)
            recursos = parsea_recursos(recursos)

            mision = Mision(
                comandante,
                sistema,
                controlado,
                fecha_inicio,
                fecha_fin,
                tipo_mision,
                num_unidades,
                coste_mensual,
                recursos
            )
            misiones.append(mision)
    return misiones

def parsea_controlado(controlado_str: str) -> bool|None:
    res = None
    controlado_str = controlado_str.strip().lower()
    if controlado_str == "sí":
        res = True
    elif controlado_str == "no":
        res = False
    return res

def parsea_fecha(fecha_str: str) -> date|None:
    res = None
    if fecha_str:
        res = datetime.strptime(fecha_str, "%Y-%m-%d").date()
    return res

def parsea_recursos(recursos_str: str) -> list[str]:
    res = []
    if recursos_str:
        res = recursos_str.split("#")
    return res
```

```
### test_misiones.py
def test_lee_misiones(misiones: list[Mision]) -> None:
    print(f"Total de misiones leídas: {len(misiones)}")
    print("Primeras 3 misiones:")
    print(misiones[:3])

if __name__ == "__main__":

    misiones = lee_misiones("data/misiones.csv")
    test_lee_misiones(misiones)
```

Ejercicio 2 (2 ptos)

```
def comandante_ocioso (misiones: list[Mision]) -> str:
   d = mision_por_comandante(misiones)
   d_dias = {comandante:total_dias_entre_misiones(lista_misiones) \
                           for comandante, lista_misiones in d.items()}
   return max(d_dias, key= d_dias.get)

def mision_por_comandante(misiones: list[Mision]) -> dict[str, list[Mision]]:
    res = defaultdict(list)
    for m in misiones:
        res[m.comandante].append(m)
    return res

def total_dias_entre_misiones(misiones: list[Mision]) -> int:
    misiones_ord = sorted(misiones, key=lambda m: m.fecha_inicio)
    dias_entre_misiones = ((m2.fecha_inicio - m1.fecha_fin).days \
                           for m1, m2 in zip(misiones_ord, misiones_ord[1:])
                               if m1.fecha_fin )
    return sum(dias_entre_misiones)


### test_misiones.py
def test_comandante_ocioso(misiones: list[Mision]) -> None:
    resultado = comandante_ocioso(misiones)
    print("Comandante más ocioso:", resultado)

if __name__ == "__main__":

    misiones = lee_misiones("data/misiones.csv")
    test_comandante_ocioso(misiones)
```

Ejercicio 3 (1,5 ptos)

```
def comandante_mas_activo_por_mes(misiones: list[Mision]) -> dict[tuple[int, int], str]:
    d = agrupar_misiones_por_mes(misiones)
    return {anyo_mes: comandante_mas_activo(lista_misiones) \
                for anyo_mes, lista_misiones in d.items()}
```

```
def agrupar_misiones_por_mes(misiones: list[Mision]) -> dict[tuple[int, int], list[Mision]]:
    res = defaultdict(list)
    for m in misiones:
        clave = (m.fecha_inicio.year, m.fecha_inicio.month)
        res[clave].append(m)
    return res

def comandante_mas_activo(misiones: list[Mision]) -> str:
    c = Counter(m.comandante for m in misiones)
    return max(c, key=c.get)

#solución con un diccionario de diccionarios y bucles todo en uno
def comandante_mas_activo_por_mes(misiones: list[Mision]) -> dict[tuple[int, int], str]:
    """
    Devuelve un diccionario cuyas claves son tuplas (año, mes) y
    los valores son los nombres de los comandantes que más misiones iniciaron en ese mes.
    """
    # {(año, mes): {comandante: conteo}}
    conteo_por_mes = defaultdict(lambda: defaultdict(int))

    for m in misiones:
        año_mes = (m.fecha_inicio.year, m.fecha_inicio.month)
        conteo_por_mes[año_mes][m.comandante] += 1

    resultado = {}
    for año_mes, conteos in conteo_por_mes.items():
        comandante_max = max(conteos.items(), key=lambda x: x[1])[0]
        resultado[año_mes] = comandante_max
    return resultado

### test_misiones.py
def test_comandante_mas_activo_por_mes(misiones):
    resultado = comandante_mas_activo_por_mes(misiones)
    print("Comandantes más activos por mes:")
    for (año, mes), comandante in sorted(resultado.items()):
        print(f"{año}-{mes:02d}: {comandante}")

if __name__ == "__main__":
    misiones = lee_misiones("data/misiones.csv")
    test_comandante_mas_activo_por_mes(misiones)
```

Ejercicio 4 (1,5 ptos)

```
# Solución 1: Diccionario de agrupación, diccionario de medias y ordenación + slicing
def coste_de_recursos(misiones: list[Mision], n: int = 3) -> list[tuple[str, float]]:
    d = agrupar_costes_por_recurso(misiones)
    d_medias = {recurso:statistics.mean(lista_costes) \
                    for recurso, lista_costes in d.items()}
    medias_ordenadas = sorted(d_medias.items(), key=lambda it:it[1], reverse=True)
    return medias_ordenadas[:n]
```

```
def agrupar_costes_por_recurso(misiones: list[Mision]) -> dict[str, list[float]]:
    res = defaultdict(list)
    for m in misiones:
        for r in m.recursos:
            res[r].append(m.coste_mensual)
    return res

# Solución 2: Creación de diccionario de acumulación y diccionario de conteo, diccionario de
medias y ordenación + slicing
def coste_de_recursos(misiones: list[Mision], n: int = 3) -> list[tuple[str, float]]:
    suma_costes = defaultdict(float)
    cuenta = defaultdict(int)

    for m in misiones:
        for recurso in m.recursos:
            suma_costes[recurso] += m.coste_mensual
            cuenta[recurso] += 1

    medias = [(recurso, suma_costes[recurso] / cuenta[recurso]) for recurso in suma_costes]
    medias_ordenadas = sorted(medias, key=lambda x: x[1], reverse=True)
    return medias_ordenadas[:n]


### test_misiones.py
def test_coste_de_recursos(misiones:list[Mision], n:int=3) -> None:
    resultado = coste_de_recursos(misiones, n)
    print("Recursos más costosos en media:")
    for recurso, coste_medio in resultado:
        print(f"{recurso}: {coste_medio:.2f}")

if __name__ == "__main__":

    misiones = lee_misiones("data/misiones.csv")
    test_coste_de_recursos(misiones,4)
```

Ejercicio 5 (2 ptos)

```
## solución 1: diccionario de agrupación y cálculo de estadísticas a partir del diccionario
def estadisticas_sistemas_rebeldes(
    misiones: list[Mision],
    tipo_mision: str | None = None
) -> list[tuple[str, int, float]]:

    d = agrupar_misiones_por_sistema(misiones, tipo_mision)
    res = []
    for sistema, lista_misiones in d.items():
        total_dias, media_unidades = estadisticas(lista_misiones)
        res.append((sistema, total_dias, media_unidades))

    return res
```

```
def agrupar_misiones_por_sistema(misiones: list[Mision],
                                 tipo_mision: str | None = None) \
                                    ->dict[str, list[Mision]]:
    res = defaultdict(list)
    for m in misiones:
        if tipo_mision is None or m.tipo_mision == tipo_mision: ## filtro
            if not m.controlado and m.fecha_fin is not None:    ## misiones válidas
                res[m.sistema].append(m)
    return res

def estadisticas(misiones: list[Mision]) -> tuple[int, float]:
    total_dias = sum ((m.fecha_fin - m.fecha_inicio).days \
                        for m in misiones)
    media_unidades = statistics.mean(m.num_unidades \
                                     for m in misiones)
    return total_dias, media_unidades

## solución 2: diccionarios de acumulación y conteo y cálculo de estadísticas a partir de los
diccionarios
def estadisticas_sistemas_rebeldes(
    misiones: list[Mision],
    tipo_mision: str | None = None
) -> list[tuple[str, int, float]]:
    # Diccionarios acumuladores
    dias_por_sistema = defaultdict(int)
    unidades_por_sistema = defaultdict(list)

    for m in misiones:
        if not m.controlado and m.fecha_fin is not None:
            if tipo_mision is None or m.tipo_mision == tipo_mision:
                dias = (m.fecha_fin - m.fecha_inicio).days
                dias_por_sistema[m.sistema] += dias
                unidades_por_sistema[m.sistema].append(m.num_unidades)

    # Construimos la lista de resultados
    resultado = []
    for sistema in dias_por_sistema:
        total_dias = dias_por_sistema[sistema]
        media_unidades = sum(unidades_por_sistema[sistema]) /
                          len(unidades_por_sistema[sistema])
        resultado.append((sistema, total_dias, media_unidades))

    return resultado

### test_misiones.py
def test_estadisticas_sistemas_rebeldes(misiones: list[Mision], tipo_mision: str|None) -> None:
    resultado = estadisticas_sistemas_rebeldes(misiones, tipo_mision=tipo_mision)
    print(f"Estadísticas de sistemas rebeldes (solo '{tipo_mision}'):")
    for sistema, dias, media in resultado:
        print(f"{sistema}: {dias} días, {media:.1f} unidades")
```

```
if __name__ == "__main__":
    misiones = lee_misiones("data/misiones.csv")
    test_estadisticas_sistemas_rebeldes(misiones, 'Vigilancia')
    test_estadisticas_sistemas_rebeldes(misiones, None)
```

Ejercicio 6 (1,5 ptos)

```
def recursos_por_tipo(misiones: list[Mision]) -> dict[str, dict[str, int]]:
    d = agrupar_por_tipo_mision(misiones)
    return {tipo: contador_recursos(lista_misiones) \
            for tipo, lista_misiones in d.items()}

def agrupar_por_tipo_mision(misiones: list[Mision]) -> dict[str, list [Mision]]:
    res = defaultdict(list)
    for m in misiones:
        res[m.tipo_mision].append(m)
    return res

## Con Counter
def contador_recursos(misiones: list[Mision]) -> dict[str, int]:
    return Counter(recurso \
                   for m in misiones \
                        for recurso in m.recursos \
                            if m.fecha_fin is not None)

## Con defaultdict
def contador_recursos2(misiones: list[Mision]) -> dict[str, int]:
    contador = defaultdict(int)
    for m in misiones:
        if m.fecha_fin is not None:
            for recurso in m.recursos:
                contador[recurso] += 1

    return contador

## Solución con bucles
def recursos_por_tipo2(misiones: list[Mision]) -> dict[str, dict[str, int]]:
    res = dict()
    for tipo, grupo_misiones in res.items():
        res[tipo] = contador_recursos(grupo_misiones)

    return res
```