Ejercicio 1 (1 punto)

```
def lee_suscripciones(ruta_fichero: str) -> list[Suscripcion]:
    suscripciones = []
    with open(ruta_fichero, "r", encoding="utf-8") as f:
        lector = csv.reader(f)
        next(lector)
        for nombre, dni, fecha_inicio, fecha_fin, tipo_plan, num_perfiles,
                  precio_mensual, addons in lector:
            fecha_inicio = parsea_fecha(fecha_inicio)
            fecha_fin = parsea_fecha(fecha_fin)
            num_perfiles = int(num_perfiles)
            precio_mensual = float(precio_mensual)
            addons = parsea_addons(addons)
            sus = Suscripcion(nombre, dni, fecha_inicio, fecha_fin, tipo_plan,
                      num_perfiles, precio_mensual, addons)
            suscripciones.append(sus)
    return suscripciones

def parsea_fecha(fecha_str: str) -> date:
    res = None
    if len(fecha_str) > 0:
        res = datetime.strptime(fecha_str, "%Y-%m-%d").date()
    return res

def parsea_addons(addons_str: str) -> list[str]:
    res = []
    if len(addons_str) > 0:
        res = addons_str.split(",")
    return res
```

TEST

```
def mostrar_iterable (it: Iterable) -> None:
    for elem in it:
        print(f"\t{elem}")

def test_lee_suscripciones(suscripciones: list[Suscripcion]) -> None:
    print(f"Total suscripciones: {len(suscripciones)}")
    print("Las tres primeras")
    mostrar_iterable(suscripciones[:3])

if __name__ == "__main__":
    suscripciones = lee_suscripciones(r"data\suscripciones.csv")
    test_lee_suscripciones(suscripciones)
```

Ejercicio 2 (1,5 puntos)

```
def suscripciones_mas_rentables(suscripciones: list[Suscripcion],
                        n: int = 3,
                        tipos_plan: set[str]|None = None) -> list[tuple[str, float]]:
    return sorted(
        ((s.dni, importe_suscripcion(s)) \
                for s in suscripciones \
                    if tipos_plan == None or s.tipo_plan in tipos_plan),
         key = lambda t: t[1], reverse = True)[:n]
```

```
def duracion_dias(sus: Suscripcion) -> int:
    fecha_fin = sus.fecha_fin
    if sus.fecha_fin is None:
        # fecha_fin = date(2025, 1, 15) # Para obtener el test del día del examen
        fecha_fin = date.today()
    return (fecha_fin - sus.fecha_inicio).days

def importe_suscripcion(sus: Suscripcion) -> float:
    duracion = duracion_dias(sus)

    precio_diario = sus.precio_mensual / 30.0
    return duracion * precio_diario
```

TEST

```
def test_suscripciones_mas_rentables(suscripciones: list[Suscripcion],
         n: int = 3,
         tipos_plan: set[str] |None = None) -> None:
    print (f"Las {n} suscripciones más rentables con tipos_plan {tipos_plan} son")
    res = suscripciones_mas_rentables(suscripciones, n, tipos_plan)
    mostrar_iterable(res)


if __name__ == "__main__":
    suscripciones = lee_suscripciones(r"data\suscripciones.csv")
    test_suscripciones_mas_rentables(suscripciones)
    test_suscripciones_mas_rentables(suscripciones, tipos_plan = {'Básico'})
```

Ejercicio 3 (1,5 puntos)

```
def plan_mas_perfiles(suscripciones: list[Suscripcion],
                          fecha_ini: date|None = None,
                          fecha_fin: date|None = None) -> tuple[str, int]:
    # Contamos cuántos perfiles hay en total para cada tipo_plan
    perfiles = defaultdict(int)
    for sus in suscripciones:
        # Filtramos por fecha_inicio
        if (fecha_ini is None or sus.fecha_inicio >= fecha_ini) \
                and (fecha_fin is None or sus.fecha_inicio <= fecha_fin):
            perfiles[sus.tipo_plan] += sus.num_perfiles

    return max(perfiles.items(), key=lambda x: x[1])
```

TEST

```
def test_plan_mas_perfiles (suscripciones: list[Suscripcion],
        fecha_ini: date|None = None,
        fecha_fin: date|None = None) -> None:
    res = plan_mas_perfiles(suscripciones, fecha_ini, fecha_fin)
    print(f"El plan con más perfiles acumulados entre las fechas {fecha_ini} y
{fecha_fin} es: {res}")

if __name__ == "__main__":
    suscripciones = lee_suscripciones(r"data\suscripciones.csv")
    test_plan_mas_perfiles(suscripciones)
    f_ini = date(2023, 2, 1)
```

```
    f_fin = date (2023, 3, 31)
    test_plan_mas_perfiles(suscripciones, f_ini, f_fin)
```

Ejercicio 4 (2 puntos)

```
# Solución 1: creado un diccionario de agrupación auxiliar en
# el que en las listas de valores se guardan las duraciones
# La media se calcula como sum/len
def media_dias_por_plan(suscripciones: list[Suscripcion]) -> dict[str, float]:
    duraciones = defaultdict(list)
    for sus in suscripciones:
        if sus.fecha_fin:
            duraciones[sus.tipo_plan].append(duracion_dias(sus))

    return {tipo_plan: sum(durs) / len(durs) for tipo_plan, durs in duraciones.items()}

# Solución 2: creado un diccionario de agrupación auxiliar en
# el que en las listas de valores se guardan las suscripciones
# La media se calcula con stastistics.mean
def media_dias_por_plan(suscripciones: list[Suscripcion]) -> dict[str, float]:
    d = agrupa_finalizadas_por_tipo_plan(suscripciones)
    return {tipo_plan: media_dias(lista_suscripciones) \
            for tipo_plan, lista_suscripciones in d.items()}

def agrupa_finalizadas_por_tipo_plan(suscripciones: list[Suscripcion])
        -> dict[str, list[Suscripcion]]:
    res = defaultdict(list)
    for s in suscripciones:
        if s.fecha_fin != None:
            res[s.tipo_plan].append(s)
    return res
def media_dias(suscripciones: list[Suscripcion]) -> float:
    return statistics.mean (duracion_dias(s) for s in suscripciones)
```

TEST:

```
def test_media_dias_por_plan(suscripciones: list[Suscripcion]) -> None:
    res = media_dias_por_plan(suscripciones)
    print("La duración media (en días) de suscripciones finalizadas por tipo de plan")
    print(res)

if __name__ == "__main__":
    suscripciones = lee_suscripciones(r"data\suscripciones.csv")
    test_media_dias_por_plan(suscripciones)
```

Ejercicio 5 (2 puntos)

```
def addon_mas_popular_por_año(suscripciones: list[Suscripcion]) -> dict[int, str]:
    addons_por_año = defaultdict(Counter)
    for sus in suscripciones:
        año = sus.fecha_inicio.year
        addons_por_año[año].update(sus.addons)

    return {año: addons.most_common(1)[0][0] for año, addons in addons_por_año.items()}
```

```
# Solución 2: Con un diccionario en el que los valores son listas de addons
# el calculo del máximo se hace con max
def addon_mas_popular_por_año(suscripciones: list[Suscripcion]) -> dict[int, str]:
    d = agrupar_addons_por_año(suscripciones)
    return {año: addon_mas_frecuente(lista_addons) \
                for año, lista_addons in d.items()}

def agrupar_addons_por_año(suscripciones: list[Suscripcion]) -> dict[int, list[str]]:
    res = defaultdict(list)
    for sus in suscripciones:
        res[sus.fecha_inicio.year] += sus.addons
    return res

def addon_mas_frecuente(addons:list[str])->str:
    c = Counter(addons)
    return max(c, key = c.get)
```

TEST

```
def test_addon_mas_popular_por_año(suscripciones: list[Suscripcion]) -> None:
    res = addon_mas_popular_por_año(suscripciones)
    print(res)

if __name__ == "__main__":
    suscripciones = lee_suscripciones(r"data\suscripciones.csv")
    test_addon_mas_popular_por_año(suscripciones)
```

Ejercicio 6 (2 puntos)

```
# Solución 1:
# Ordenando solo las claves, con bucles y consultando el diccionario para el tratamiento
# de pareja de elementos
def evolucion_años(suscripciones: list[Suscripcion]) -> list[tuple[int, int]]:
    # Contamos num suscripciones por año
    sus_por_año = num_suscripciones_por_año(suscripciones)

    # Calculamos incremento o decremento por año
    años = sorted(sus_por_año.keys())
    evolucion = []
    for año1, año2 in zip(años, años[1:]):
        evolucion.append((año2, sus_por_año[año2] - sus_por_año[año1]))

    return evolucion

def num_suscripciones_por_año(suscripciones: list[Suscripcion]) -> dict[int, int]:
   res = defaultdict(int)
   for s in suscripciones:
       ano_ini = s.fecha_inicio.year
       if s.fecha_fin != None:
           ano_fin = s.fecha_fin.year
           res[ano_fin] -= 1
       res[ano_ini] += 1

   return res
```

```
# Solución 2:
# Ordenando solo los items, por compresión y consultando el diccionario para el
# tratamiento de pareja de elementos
def evolucion_años(suscripciones: list[Suscripcion]) \
            -> list[tuple[int, int]]:
    c = num_suscripciones_por_año(suscripciones)
    sus_año = sorted(c.items())
    incrementos = [(t2[0], t2[1] - t1[1])  \
                   for t1, t2 in zip(sus_año, sus_año[1:])]
    return incrementos
```

TEST

```
def test_evolucion_años(suscripciones: list[Suscripcion]) -> None:
    res = evolucion_años(suscripciones)
    print(f"Evolución de suscripciones por año\n{res}")

if __name__ == "__main__":
    suscripciones = lee_suscripciones(r"data\suscripciones.csv")
    test_evolucion_años(suscripciones)
```