Ejercicio 1

```
def leer_desastres(ruta: str) -> list[Desastre]:
    result = []
    with open(ruta,'rt', encoding='utf-8') as f:
        it = csv.reader(f, delimiter=';')
        next(it)
        for fecha, hora, localizacion, operadores,\
                codigos, rutas, modelos,\
                supervivientes, fallecidos, fallecidos_en_tierra, operacion in it:
            fecha = parsea_fecha(fecha)
            hora = parsea_hora(hora)
            vuelos = parsea_vuelos(operadores, codigos, rutas, modelos)
            supervivientes = int(supervivientes)
            fallecidos = int(fallecidos)
            fallecidos_en_tierra = int(fallecidos_en_tierra)
            desastre = Desastre(fecha, hora, localizacion,\
                                supervivientes, fallecidos, fallecidos_en_tierra,\
                                operacion, vuelos)
            result.append(desastre)
    return result

def parsea_fecha(fecha_str: str) -> date:
    return datetime.strptime(fecha_str, "%d/%m/%Y").date()

def parsea_hora(hora_str: str) -> time | None:
    result = None
    if len(hora_str) > 0:
      result = datetime.strptime(hora_str, "%H:%M").time()
    return result

def parsea_vuelos(operadores: str, codigos: str, rutas: str,\
                  modelos: str) -> list[Vuelo]:
    result = []
    operadores = parsea_lista(operadores)
    codigos = parsea_lista(codigos)
    rutas = parsea_lista(rutas)
    modelos = parsea_lista(modelos)
    for operador, codigo, ruta, modelo in zip(operadores, codigos, rutas, modelos):
        result.append(Vuelo(operador,codigo,ruta,modelo))
    return result

def parsea_lista(cadena: str) -> list[str]:
  return cadena.split("/")


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

```
def test_leer_desastres(desastres: list[Desastre]) -> None:
    print(f"Número de desastres leídos: {len(desastres)}")
    print ("Los dos primeros son:")
    mostrar_iterable(desastres[:2])
    print ("Los dos últimos son:")
    mostrar_iterable(desastres[-2:])

def main():
    desastres = leer_desastres("data/mayday.csv")
    print("Test leer_desastres", "="*50)
    test_leer_desastres(desastres)

if __name__ == "__main__":
    main()
```

Ejercicio 2

```
def desastres_con_fallecidos_en_tierra(desastres: list[Desastre],\
                                       n: int | None = None)\
            -> list[tuple[str, date, time, int]]:
    filtrados = ((d.localizacion,d.fecha,d.hora,d.fallecidos_en_tierra) \
                         for d in desastres if d.fallecidos_en_tierra > 0)
    res = sorted(filtrados, key=lambda x:x[3], reverse=True)
    if n is not None:
        res = res[:n]
    return res


## Test ========================================================================

def test_desastres_con_fallecidos_en_tierra(eventos: list[Desastre],\
                                                  n: int|None = None) -> None:
    res = desastres_con_fallecidos_en_tierra(eventos, n)
    if n:
        print(f"Los {n} desastres con más fallecidos en tierra son:")
    else:
        print(f"Los desastres con más fallecidos en tierra son:")
    mostrar_iterable(res)

def main():
    desastres = leer_desastres("data/mayday.csv")
    print("Test desastres_con_fallecidos_en_tierra", "="*50)
    test_desastres_con_fallecidos_en_tierra(desastres)
    test_desastres_con_fallecidos_en_tierra(desastres,5)

if __name__ == "__main__":
    main()
```

Ejercicio 3

```
def decada_mas_colisiones(desastres: list[Desastre]) -> tuple[int, int]:
    d = contar_colisiones_por_decada(desastres)
    print(d)
    return max(d.items(), key=lambda it: it[1])

def contar_colisiones_por_decada(desastres: list[Desastre]) -> dict[int, int]:
   return Counter(d.fecha.year // 10 * 10 for d in desastres if len(d.vuelos) > 1)
```

Solución alternativa

```
def decada_mas_colisiones(desastres: list[Desastre]) -> tuple[int, int]:
    d = colisiones_por_decada(desastres)
    decada_mas_colisiones = max(d.items(),key=lambda x: len(x[1]))
    return (decada_mas_colisiones[0], len(decada_mas_colisiones[1]))

def colisiones_por_decada(desastres: list[Desastre]) -> dict[int, list[Desastre]]:
    res = defaultdict(list)
    for d in desastres:
      if len(d.vuelos) > 1:
        res[d.fecha.year // 10 * 10].append(d)
    return res


## Test ========================================================================
def test_decada_mas_colisiones(desastres: list[Desastre]) -> None:
    decada, num_colisiones = decada_mas_colisiones(desastres)
    print(f"La década de {decada} fue la peor, con {num_colisiones} desastres con
colisiones de aeronaves.")

def main():
    desastres = leer_desastres("data/mayday.csv")
    print("Test colisiones_de_aeronaves", "="*50)
    test_colisiones_de_aeronaves(desastres)

if __name__ == "__main__":
    main()
```

Ejercicio 4

```
def mayor_periodo_sin_desastres(eventos: list[Desastre],\
                        operacion: str|None = None) -> tuple[date, date, int]:
    filtrados = (e for e in eventos \
                           if operacion == None or e.operacion == operacion)
    ordenados = sorted(filtrados, key=lambda event: event.fecha)
    diff = [(e1.fecha, e2.fecha, (e2.fecha-e1.fecha).days)\
              for e1, e2 in zip(ordenados, ordenados[1:])]
    return max(diff, key=lambda x: x[2])
```

```
## Test ========================================================================
def test_mayor_periodo_sin_desastres(eventos: list[Desastre],\
                             operacion: str|None = None) -> None:
    fecha_ini, fecha_fin, duracion = mayor_periodo_sin_desastres(eventos, operacion)
    if operacion:
        print(f"El mayor periodo sin desastres durante la operación {operacion}
comienza el {fecha_ini}, termina el {fecha_fin} y dura {duracion} días.")
    else:
        print(f"El mayor periodo sin desastres comienza el {fecha_ini}, termina el
{fecha_fin} y dura {duracion} días.")

def main():
    desastres = leer_desastres("data/mayday.csv")
    print("Test mayor_periodo_sin_desastres", "="*50)
    test_mayor_periodo_sin_desastres(desastres)
    test_mayor_periodo_sin_desastres(desastres, "Taking-off")
    test_mayor_periodo_sin_desastres(desastres, "Landing")

if __name__ == "__main__":
    main()
```

Ejercicio 5

```
def estadisticas_por_operacion(desastres: list[Desastre],\
                        limite_tasa_supervivencia: float|None = None)\
                            ->dict[str, tuple[int, float, float]]:
    d = desastres_por_operacion(desastres, limite_tasa_supervivencia)
    return {operacion: estadisticas_alt(desastres) \
              for operacion, desastres in d.items()}

def desastres_por_operacion(desastres: list[Desastre],\
              limite_tasa_supervivencia: float|None = None -> dict[str, list]:
    result = defaultdict(list)
    for d in desastres:
        if limite_tasa_supervivencia == None \
            or tasa_supervivencia(d) <= limite_tasa_supervivencia:
        result[d.operacion].append(d)
    return result

def estadisticas(desastres: list[Desastre]) -> tuple[int, float, float]:
   return (len(desastres),
            statistics.mean(d.supervivientes for d in desastres),\
            statistics.mean(d.fallecidos for d in desastres))
```

```
def estadisticas_alt(desastres: list[Desastre]) -> tuple[int, float, float]:
    supervivientes = 0
    fallecidos = 0
    for d in desastres:
        supervivientes += d.supervivientes
        fallecidos += d.fallecidos
    res = None
    if len(desastres) > 0:
        res = (len(desastres), supervivientes / len(desastres),
            fallecidos / len(desastres))
    return res

def tasa_supervivencia(desastre: Desastre) -> float:
   return desastre.supervivientes / (desastre.supervivientes + desastre.fallecidos)


## Test ========================================================================
def test_estadisticas_por_operacion(desastres: list[Desastre],\
                                limite_tasa_supervivencia: float|None = None) -> None:
    res = estadisticas_por_operacion(desastres, limite_tasa_supervivencia)
    if limite_tasa_supervivencia:
        print(f"Las estadísticas por operación con tasa de supervivencia menor a
{limite_tasa_supervivencia} son:")
    else:
        print(f"Las estadísticas por operación son:")
    mostrar_iterable(res.items())

def main():
    desastres = leer_desastres("data/mayday.csv")
    print("Test estadisticas_por_operacion", "="*50)
    test_estadisticas_por_operacion(desastres)
    test_estadisticas_por_operacion(desastres, 0.15)

if __name__ == "__main__":
    main()
```