Ejercicio 1 (1,5 puntos)

```
def lee_viviendas(ruta: str) -> list[Vivienda]:
    res = list()
    with open(ruta,'rt',encoding = 'utf-8') as f:
        it = csv.reader(f, delimiter = ';')
        next(it)
        for propietario, calle, numero, fecha_adquisicion, metros, precio,mejoras in it:
            numero = int(numero)
            fecha_adquisicion = parsea_fecha(fecha_adquisicion)
            metros = float(metros)
            precio = int(precio)
            mejoras = parsea_mejoras(mejoras)
            tupla = Vivienda(propietario, calle, numero, fecha_adquisicion, metros, \
                precio, mejoras)
            res.append(tupla)
        return res

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

def parsea_mejoras(mejoras_str: str) -> list[Mejora]:
    res = []
    if len(mejoras_str) > 0:
        mejoras = mejoras_str.split("*")
        res = [parsea_mejora(m) for m in mejoras]
    return res

def parsea_mejora(mejora_str: str) -> Mejora:
    denominacion, coste, fecha = mejora_str.split("-")
    coste = int(coste)
    fecha = parsea_fecha(fecha)
    return Mejora(denominacion, coste, fecha)
```

TEST

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

def test_lee_viviendas(viviendas: list[Vivienda]) -> None:
    print(f"Viviendas leidas: {len(viviendas)}")
    print ("Las dos primeras son")
    mostrar_iterable(viviendas[:2])
    print ("Las dos últimas son")
    mostrar_iterable(viviendas[-2:])

def main():
    viviendas = lee_viviendas(r"data\urbanizacion.csv")
    test_lee_viviendas(viviendas)

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

Ejercicio 2 (1,25 puntos)

```
# Solución 1: Con defaultdict y función auxiliar para filtro
def total_mejoras_por_calle (viviendas: list[Vivienda], par_impar: str) -> dict[str,
int]:
    res = defaultdict(int)
    for v in viviendas:
        if es_par(v, par_impar):
            res[v.calle] += len(v.mejoras)
    return res

def es_par(vivienda: Vivienda, par_impar: str) -> bool:
    return (par_impar.lower() == "par" and vivienda.numero % 2 == 0) \
            or (par_impar.lower() == "impar" and vivienda.numero % 2 != 0)

# Solución 2: Con dict y sin función auxiliar para filtro
def total_mejoras_por_calle2 (viviendas: list[Vivienda], par_impar: str) -> dict[str,
int]:
    res = dict()
    for v in viviendas:
        if (v.numero % 2 == 0 and par_impar.lower() == 'par') \
          or (v.numero % 2 != 0 and par_impar.lower() == 'impar'):
            if v.calle not in res:
                res[v.calle] = 0
            res[v.calle] += len(v.mejoras)
    return res
```

TEST

```
def test_total_mejoras_por_calle (viviendas: list[Vivienda], par_impar: str) -> None:
    print(f"El total de mejoras por calle para número {par_impar} es")
    res = total_mejoras_por_calle(viviendas, par_impar)
    mostrar_iterable(res.items())

def main():
    viviendas = lee_viviendas(r"data\urbanizacion.csv")
    test_total_mejoras_por_calle(viviendas, "par")
    test_total_mejoras_por_calle(viviendas, "impar")

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

Ejercicio 3 (1,25 puntos)

```
# Solución 1: usando definiciones por compresión y función auxiliar para
# el cálculo de días entre la vivienda y la mejora
def vivienda_con_mejora_mas_rapida(viviendas: list[Vivienda]) -> tuple[str, str, int,
int, str]:
    # Como las viviendas no están ordenadas, se aplana y se crea un generador
    # con la vivienda y la mejora
    g = ((v, mejora) for v in viviendas for mejora in v.mejoras)

    # Se calcula el mínimo recorriendo el generador anterior (Vivienda, Mejora)
    # y usando el número de días entre la mejora y la vivienda para el mínimo
    v, m = min(g, key = lambda t: dias_vivienda_mejora(t[0], t[1]))
    return (v.propietario, v.calle, v.numero, dias_vivienda_mejora(v, m),m.denominacion)
```

```
def dias_vivienda_mejora(vivienda: Vivienda, mejora: Mejora) -> int:
    return (mejora.fecha - vivienda.fecha_adquisicion).days

# Solución 2: usando bucles y función auxiliar para el cálculo de días entre la vivienda
# y la mejora
def vivienda_con_mejora_mas_rapida2(viviendas: list[Vivienda]) -> tuple[str, str, int,
int, str]:
    aux = list()
    for v in viviendas:
        for m in v.mejoras:
            aux.append((v, dias_vivienda_mejora(v, m), m.denominacion))
    vivienda, dias, denominacion = min(aux, key = lambda e: e[1])
    return (vivienda.propietario, vivienda.calle, vivienda.numero, dias, denominacion)

# Solución 3: usando bucles y el esquema de mínimo con bucles
def vivienda_con_mejora_mas_rapida3(viviendas: list[Vivienda]) -> tuple[str, str, int,
int, str]:
    diasmin = None
    res = None
    for v in viviendas:
        if len(v.mejoras)>0:
            pm = primera_mejora(v)
            dias = dias_vivienda_mejora(v, pm)
            if diasmin == None or dias<diasmin:
                diasmin = dias
                res = (v.propietario, v.calle, v.numero, dias, pm.denominacion)
    return res

def primera_mejora(vivienda: Vivienda) -> Mejora:
    return min(vivienda.mejoras, key = lambda m: m.fecha)
```

TEST

```
def test_vivienda_con_mejora_mas_rapida(viviendas: list[Vivienda]) -> None:
    res = vivienda_con_mejora_mas_rapida(viviendas)
    print(f"La vivienda con la mejora más rápida es {res}")

def main():
    viviendas = lee_viviendas(r"data\urbanizacion.csv")
    test_vivienda_con_mejora_mas_rapida(viviendas)

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

Ejercicio 4 (2 puntos)

```
# Solución 1: Con defaultdict y el "truco" de ir sumando y restando los pares e impares
def calle_mayor_diferencia_precios(viviendas: list[Vivienda]) -> str:
    aux = defaultdict(int)
    for v in viviendas:
        if v.numero % 2 != 0:
            aux[v.calle] += v.precio
        else:
            aux[v.calle] -= v.precio
    return max(aux, key = lambda calle: abs(aux.get(calle)))
```

```
# Solución 2: Con defaultdict y el "truco" de ir sumando y restando los pares e impares
def calle_mayor_diferencia_precios2(viviendas: list[Vivienda]) -> str:
    aux = dict()
    for v in viviendas:
        if v.calle not in aux:
            aux[v.calle] = 0
        if v.numero % 2 != 0:
            aux[v.calle] += v.precio
        else:
            aux[v.calle] -= v.precio
    return max(aux, key = lambda calle: abs(aux.get(calle)))

# Solución 3: Creando una función de agrupación auxiliar,
# que puede tener la fecha porque se puede reusar para el ejercicio 5
def calle_mayor_diferencia_precios(viviendas: list[Vivienda]) -> str:
    d = agrupar_viviendas_por_calle(viviendas)
    d2 = {calle: diferencia_precios(lista_viviendas)
                for calle, lista_viviendas in d.items()}
    return max(d2, key = d2.get)

def agrupar_viviendas_por_calle(viviendas: list[Vivienda], fecha: date|None = None) ->
dict[str, list[Vivienda]]:
    res = defaultdict(list)
    for v in viviendas:
        if fecha == None or v.fecha_adquisicion >= fecha:
            res[v.calle].append(v)
    return res

def diferencia_precios(viviendas: list[Vivienda]) -> int:
    suma_pares = 0
    suma_impares = 0
    for v in viviendas:
        if v.numero % 2 == 0:
            suma_pares += v.precio
        else:
            suma_impares += v.precio
    res = None
    if len(viviendas) > 0:
        res = abs(suma_pares - suma_impares)
    return res
```

TEST

```
def test_calle_mayor_diferencia_precios(viviendas: list[Vivienda]) -> None:
    res = calle_mayor_diferencia_precios(viviendas)
    print (f"La calle con mayor diferencia de precios promedio es {res}")

def main():
    viviendas = lee_viviendas(r"data\urbanizacion.csv")
    test_calle_mayor_diferencia_precios(viviendas)

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

Ejercicio 5 (2 puntos)

```
# Solución 1: Con defaultdict, función auxiliar para el cálculo del valor de la vivienda
# y definiciones por compresión
def n_viviendas_top_valoradas_por_calle(viviendas: list[Vivienda], fecha: date|None =
None, n: int = 3) -> dict[str, list[tuple[str, int, int]]]:
    aux = defaultdict(int)
    for v in viviendas:
        if fecha == None or fecha <= v.fecha_adquisicion:
            importe = valor_vivienda (v)
            aux[v.calle].append((v.propietario, v.numero, importe))

    return {c: sorted(v, key = lambda e: e[2], reverse = True)[:n] \
          for c, v in aux.items()}

def valor_vivienda(vivienda: Vivienda) -> int:
    valor_mejoras = sum (mejora.coste for mejora in vivienda.mejoras)
    return vivienda.precio + valor_mejoras

# Solución 2: Reutilizando la función auxiliar de agrupación del ejercicio 4,
# y función auxiliar para calcular las n mejores viviendas
def n_viviendas_top_valoradas_por_calle(viviendas: list[Vivienda], fecha: date|None =
None, n: int = 3) -> dict[str, list[tuple[str, int, int]]]:
    d = agrupar_viviendas_por_calle(viviendas, fecha)
    return {calle: n_viviendas_top(lista_viviendas, n) \
                for calle, lista_viviendas in d.items()}

def n_viviendas_top(viviendas: list[Vivienda], n: int = 3)
        -> list[tuple[str, int, int]]:
    # por nombre del propietario, el número de la vivienda y el valor de cada vivienda
    return sorted(((v.propietario, v.numero, valor_vivienda(v)) for v in viviendas),
                  key = lambda t: t[2],
                  reverse = True)[:n]

# Solución 3: Con dict, función auxiliar para el cálculo del valor de la vivienda
# y bucles
def n_viviendas_top_valoradas_por_calle3(viviendas: list[Vivienda], fecha: date|None =
None, n: int = 3) -> dict[str, list[tuple[str, int, int]]]:
    aux = dict()
    for v in viviendas:
        if fecha == None or fecha <= v.fecha_adquisicion:
            if v.calle not in aux:
                aux[v.calle] = 0
            importe = valor_vivienda (v)
            aux[v.calle].append((v.propietario, v.numero, importe))
    res = dict()
    for c, v in aux.items():
        res[c] = sorted(v, key = lambda e: e[2], reverse = True)[:n]
    return res
```

TEST

```
def test_n_viviendas_top_valoradas_por_calle(viviendas: list[Vivienda], fecha: date|None
= None, n: int = 3) -> None:
    res = n_viviendas_top_valoradas_por_calle(viviendas, fecha,  n)
    print(f"Para n = {n} y fecha = {fecha}")
    mostrar_iterable(res.items())
```

```
def main():
    viviendas = lee_viviendas(r"data\urbanizacion.csv")
    test_n_viviendas_top_valoradas_por_calle(viviendas, date(2020, 1, 1), 4)
    test_n_viviendas_top_valoradas_por_calle(viviendas)

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

Ejercicio 6 (2 puntos)

```
# Solución 1: Con función auxiliar de agrupación por año, defaultdict y
# definiciones por compresion
def valor_metro_cuadrado_por_calle_y_año(viviendas: list[Vivienda]) -> list[tuple[int,
list[tuple[str, float]]]]:
    d = agrupar_viviendas_por_anyo(viviendas)
    d2 = {anyo: promedio_metro_cuadrado_por_calle(lista_viviendas) \
                    for anyo, lista_viviendas in d.items()}
    return sorted(d2.items())

def agrupar_viviendas_por_anyo(viviendas: list[Vivienda]) -> dict[int, list[Vivienda]]:
    res = defaultdict(list)
    for v in viviendas:
        res[v.fecha_adquisicion.year].append(v)
    return res

# Solución 1 para el cálculo del promedio del metro cuadrado por calle reutilizando la
# función de agrupación por calle y statistics.mean
def promedio_metro_cuadrado_por_calle(viviendas: list[Vivienda]) -> list[tuple[str,
float]]:
    d = agrupar_viviendas_por_calle(viviendas)
    d2 = {calle: statistics.mean(precio_metro(v) for v in lista_viviendas)\
                for calle, lista_viviendas in d.items()}
    return sorted(d2.items(), key = lambda it: it[1], reverse = True)

def precio_metro (vivienda: Vivienda) -> float:
    return vivienda.precio/vivienda.metros

# Solución 2: Con función auxiliar de agrupación por año, con bules
# y definiciones por compresion
def valor_metro_cuadrado_por_calle_y_año2(viviendas: list[Vivienda]) -> list[tuple[int,
list[tuple[str, float]]]]:
    aux = agrupar_viviendas_por_anyo(viviendas)
    res = dict()
    for c, v in aux.items():
        res[c] = promedio_metro_cuadrado_por_calle(v)
    return sorted(res.items())

# Solución 2: para el cálculo del promedio del metro cuadrado por calle creando un
diccionario de agrupación (calle, lista de precios),
# y calculando la media como sum y len
def promedio_metro_cuadrado_por_calle(viviendas: list[Vivienda]) -> list[tuple[str,
float]]:
    aux = defaultdict(list)
    for v in viviendas:
        aux[v.calle].append(precio_metro(v))
```

```
    res = dict()
    for c, v in aux.items():
        res[c] = sum(v)/len(v)
    return sorted(res.items(), key = lambda e: e[1], reverse = True)
```

TEST

```
def test_valor_metro_cuadrado_por_calle_y_año(viviendas: list[Vivienda]) ->
list[tuple[int, list[tuple[str, float]]]]:
    print("El valor por metro cuadrado por calle y año es")
    res = valor_metro_cuadrado_por_calle_y_año(viviendas)
    mostrar_iterable(res)

def main():
    viviendas = lee_viviendas(r"data\urbanizacion.csv")
    test_valor_metro_cuadrado_por_calle_y_año(viviendas)

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