Ejercicio 1

```
def lee_facturas(ruta_fichero: str) -> List[Factura]:

    with open(ruta_fichero, encoding='utf-8') as f:
        res = []
        lector = csv.reader(f)
        next(lector)

        for id_vivienda, tipo_vivienda, barrio, tipo_tarifa, periodo_inicio,
                periodo_fin, coste_potencia, consumo_punta, consumo_valle, precio_kwh,
                importe_total in lector:

            periodo_inicio = datetime.strptime(periodo_inicio, "%Y-%m-%d").date()
            periodo_fin = datetime.strptime(periodo_fin, "%Y-%m-%d").date()
            periodo_facturado = IntervaloFechas(periodo_inicio, periodo_fin)
            coste_potencia = float(coste_potencia)
            consumo_punta = float(consumo_punta)
            consumo_valle = float(consumo_valle)

            if tipo_tarifa == "única":
                precio_punta = precio_valle = float(precio_kwh)
            else:
                precio_punta, precio_valle = [float(x) for x in precio_kwh.split("/")]

            importe_total = float(importe_total)

            res.append(Factura(id_vivienda, tipo_vivienda, barrio,
                tipo_tarifa, periodo_facturado, coste_potencia, consumo_punta,
                consumo_valle, precio_punta, precio_valle, importe_total))

        res.sort(key=lambda x: x.periodo_facturado.inicio)
        return res
```

Ejercicio 2

```
def extrae_precio_por_mes(facturas: List[Factura], tipo_tarifa: str)
        -> Dict[str, Tuple[float, float]]:

    res = dict()
    for factura in facturas:
        if factura.tipo_tarifa == tipo_tarifa:
            mes = factura.periodo_facturado.inicio.strftime("%Y-%m")
            res[mes] = factura.precio_punta, factura.precio_valle
    return res
```

Ejercicio 3

```
def busca_vivienda_mayor_consumo_acumulado(facturas: List[Factura]) -> Tuple[str, float]:

    res = defaultdict(float)
    for factura in facturas:
        res[factura.id_vivienda] += (factura.consumo_punta + factura.consumo_valle)

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

Ejercicio 4

```
def barrios_mayor_consumo_valle_medio(facturas: List[Factura], top_n: int)
        -> List[str]:

    res = defaultdict(float)
    contador = Counter()
    for factura in facturas:
        res[factura.barrio] += factura.consumo_valle
        contador[factura.barrio] += 1

    for k in res:
        res[k] /= contador[k]

    return [k for k, _ in sorted(res.items(), key=lambda x: x[1], reverse=True)[:top_n]]
```

Ejercicio 5

```
def compara_importe_tipos_factura(facturas: List[Factura], id_vivienda: str)
        -> Optional[Tuple[str, float, float]]:

    facturas_vivienda = [f for f in facturas if f.id_vivienda == id_vivienda]

    if len(facturas_vivienda) == 0:
        return None

    tipo_tarifa_actual = facturas_vivienda[0].tipo_tarifa
    tipo_tarifa_alternativa = "única" if tipo_tarifa_actual == "tramos" else "tramos"

    precio_mes = extrae_precio_por_mes(facturas, tipo_tarifa_alternativa)

    importe_total_cambio, importe_total_actual = 0, 0
    for f in facturas_vivienda:
        mes = f.periodo_facturado.inicio.strftime("%Y-%m")
        precio_punta, precio_valle = precio_mes[mes]
        importe_total_cambio += precio_valle*f.consumo_valle + precio_punta*f.consumo_punta
            + f.coste_potencia
        importe_total_actual += f.importe_total

    return tipo_tarifa_actual +"->" + tipo_tarifa_alternativa, importe_total_actual,
        importe_total_cambio
```

Ejercicio 6

```
def busca_cambios_beneficiosos(facturas: List[Factura]) -> List[Tuple[str, int, float]]:
    id_viviendas = set(f.id_vivienda for f in facturas)
    contador = Counter()
    total_ahorrado = defaultdict(float)

    for id_vivienda in id_viviendas:
        cambio, importe_actual, importe_cambio =
            compara_importe_tipos_factura(facturas, id_vivienda)
        if importe_cambio < importe_actual:
            contador[cambio] += 1
            total_ahorrado[cambio] += importe_actual - importe_cambio

    # Devuelve una lista con el tipo de cambio, el número de cambios de ese
    tipo y el total ahorrado por esos cambios
    return [(cambio, contador[cambio], total_ahorrado[cambio]) for cambio in contador]
```

Ejercicio 7

```
def calcula_mes_incremento_maximo_consumo_acumulado(facturas: List[Factura],
        tipo_vivienda: Optional[str] = None) -> Tuple[str, float]:

    suma_por_meses = calcula_consumo_acumulado_por_meses(facturas, tipo_vivienda)
    meses = sorted(suma_por_meses.keys())

    incremento_maximo = 0
    mes_maximo = None
    for i in range(1, len(meses)):
        incremento = suma_por_meses[meses[i]] - suma_por_meses[meses[i-1]]
        if incremento > incremento_maximo:
            incremento_maximo = incremento
            mes_maximo = meses[i]

    return mes_maximo, incremento_maximo

def calcula_consumo_acumulado_por_meses(facturas: List[Factura], tipo_vivienda: Optional[str])
        -> Dict[str, float]:
''' Devuelve un diccionario con el consumo acumulado de todas las viviendas por mes. '''
    res = defaultdict(float)
    for factura in facturas:
        if tipo_vivienda == None or factura.tipo_vivienda == tipo_vivienda:
            mes = factura.periodo_facturado.inicio.strftime("%Y-%m")
            res[mes] += factura.consumo_punta + factura.consumo_valle
    return res
```