Ejercicio 1

```
def lee_recetas(ruta_fichero: str) -> List[Receta]:
    recetas = []
    with open(ruta_fichero, encoding="utf-8") as f:
        lector = csv.reader(f)
        next(lector)

        for nombre, tipo, dificultad, ingredientes, tiempo_preparacion, calorias,\
                fecha_creacion, precio_estimado in lector:
            #Si no hay ingredientes, se devuelve None
            ingredientes = parsea_ingredientes(ingredientes)
            tiempo_preparacion = int(tiempo_preparacion)
            calorias = int(calorias)
            fecha_creacion = parsea_fecha(fecha_creacion)
            precio_estimado = float(precio_estimado)
            receta = Receta(nombre.strip(), tipo.strip(), dificultad.strip(),
                ingredientes, tiempo_preparacion, calorias,
                fecha_creacion, precio_estimado)
            recetas.append(receta)
    return recetas

def parsea_ingredientes(str_ingredientes: Optional[str]) -> List[str]:
    res = None
    if str_ingredientes:
        res = [ingrediente.strip() for ingrediente in str_ingredientes.split(",")]
    return res

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

Ejercicio 2

```
def receta_mas_barata(recetas: List[Receta], tipos: Set[str],
        n: Optional[int] = None) -> Receta:

    recetas = sorted((r for r in recetas if r.tipo in tipos),
        key=lambda r: r.calorias)[:n]

    return min(recetas, key=lambda r: r.precio_estimado)
```

Ejercicio 3

```
def obtener_ingredientes (recetas: List[Receta], mes1: Optional[int] = None,
        mes2: Optional[int] = None) -> Set[str]:
    res = set()
    for receta in recetas:
        mes_receta = receta.fecha_creacion.month
        if receta.ingredientes != None and \
                (mes1 is None or mes_receta >= mes1) and \
                (mes2 is None or mes_receta < mes2):
            res = res.union(receta.ingredientes)
            # También res = res | receta.ingredienes
    return res
```

Ejercicio 4

```
def recetas_con_precio_menor_promedio(recetas: List[Receta], n: int)
        -> List[Tuple[str, int]]:

    promedio = precio_medio(recetas)
    return sorted(((receta.nombre, receta.calorias) for receta in recetas\
        if receta.precio_estimado < promedio), key=lambda tupla:tupla[1])[:n]

def precio_medio(recetas:List[Receta])->Optional[float]:
    res = None
    if len(recetas) > 0:
        res = sum(r.precio_estimado for r in recetas) / len(recetas)
    return res
```

Ejercicio 5

```
def receta_mas_ingredientes(recetas: List[Receta],
        ingredientes: Optional[Set[str]] = None) -> Tuple[str, List[str]]:

    recetas = [r for r in recetas \
        if r.ingredientes != None and\
            (ingredientes == None or tiene_algun_ingrediente(r, ingredientes))]

    receta = max(recetas, key=lambda r: len(r.ingredientes) )
    return (receta.nombre, receta.ingredientes)

def tiene_algun_ingrediente(receta:Receta, ingredientes: Set[str]) -> bool:
    #Esquema existe
    res = False
    for i in receta.ingredientes:
        if i in ingredientes:
            res = True
            break
    return res
```

Ejercicio 6

```
def ingredientes_mas_comunes_por_tipo(recetas: List[Receta])
        -> Dict[str, List[str]]:
    d = agrupar_ingredientes_por_tipo(recetas)
    res = {}
    for tipo, ingredientes in d.items():
        contador_ingredientes = Counter(ingredientes)
        res[tipo] = ingredientes_mas_comunes(contador_ingredientes)
    return res

def agrupar_ingredientes_por_tipo(recetas: List[Receta])
        -> DefaultDict[str, List[str]]:
    res = defaultdict(list)
    for receta in recetas:
        if receta.ingredientes:
            res[receta.tipo]+=list(receta.ingredientes)
```

```
            # También se puede usar
            # res[receta.tipo].extend(receta.ingredientes)
    return res

def ingredientes_mas_comunes(cont_ingredientes: Dict[str, int]) -> List[str]:
    return [ingrediente for ingrediente, _ in cont_ingredientes.most_common(3)]
```

#Alternativa

```
def ingredientes_mas_comunes(cont_ingredientes: Dict[str, int]) -> List[str]:
    res= sorted (cont_ingredientes, key=cont_ingredientes.get, reverse=True]
    return res[:3]
```

Ejercicio 7

```
def mes_con_precio_medio_mas_alto(recetas: List[Receta], n: int) -> int:
    d = agrupar_recetas_por_mes(recetas)
    d_promedios = {mes: precio_medio(lista_recetas)\
                   for mes, lista_recetas in d.items()
                       if len(lista_recetas) >= n}

    return max(d_promedios, key=d_promedios.get)

    # También
    # return max(d_promedios, key=lambda tipo:d_promedios.get(tipo))
    # return max(d_promedios, key=lambda tipo:d_promedios[tipo])

def agrupar_recetas_por_mes(recetas: List[Receta]) -> Dict[int, List[Receta]]:
    res = defaultdict(list)
    for r in recetas:
        res[r.fecha_creacion.month].append(r)
    return res
```