Ejercicio 1

```
def lee_jugadores(filename):
    with open(filename, encoding='utf-8') as f:
        lector = csv.reader(f, delimiter=";")
        next(lector)
        res = []
        for ape_nom, licencia, fecha_ncto, federacion, \
            handicap, fec_hor_act, senior, resultados in lector:
            fecha_ncto = parsea_fecha(fecha_ncto)
            handicap = float(handicap)
            fec_hor_act = parsea_fecha_hora(fec_hor_act)
            senior = parsea_booleano(senior)
            resultados = parsea_resultados(resultados)
            res.append(Jugador(ape_nom, licencia, fecha_ncto, federacion, \
                               handicap, fec_hor_act, senior, resultados))
        return res

def parsea_fecha(str_fecha):
    return datetime.strptime(str_fecha, "%d/%m/%Y").date()

def parsea_fecha_hora(str_fecha_hora):
    return datetime.strptime(str_fecha_hora,"%d/%m/%Y %H:%M:%S")

def parsea_resultados(str_resultados):
    return [int(resultado) for resultado in str_resultados.split(",")]

def parsea_booleano(str_booleano):
    res = None
    if str_booleano == "S":
        res = True
    elif str_booleano == "N":
        res = False
    return res

################## TEST ###########################################################
def muestra_iterable(iterable):
    for elem in iterable:
        print(elem)

def test_lee_jugadores(jugadores):
    print(f"Registros leídos: {len(jugadores)}")
    print("Los dos primeros:")
    muestra_iterable(jugadores[:2])
    print("Los dos últimos:")
    muestra_iterable(jugadores[-2:])

if __name__=="__main__":
    jugadores = lee_jugadores("../data/jugadores.csv")
       test_lee_jugadores(jugadores)
```

Ejercicio 2

```
def mejores_jugadores(jugadores, anyo, n):
    lis_ord = sorted(((j.licencia, j.ape_nom, j.handicap) \
                      for j in jugadores if j.fecha_ncto.year == anyo),
                      key = lambda t:t[2])
    return lis_ord[:n]

################## TEST ###############################################################

def test_mejores_jugadores(jugadores, anyo, n):
    res = mejores_jugadores(jugadores, anyo, n)
    print(f"Los {n} mejores jugadores nacidos en el {anyo} son: {res}")

if __name__=="__main__":
    jugadores = lee_jugadores("../data/jugadores.csv")
    test_mejores_jugadores(jugadores, 1969, 4)
```

Ejercicio 3

```
def jugadores_por_golpes(jugadores):
    d = agrupar_por_numero_golpes(jugadores)
    return sorted(d.items(), reverse=True)

def agrupar_por_numero_golpes(jugadores):
    res = dict()
    for j in jugadores:
        for resultado in j.resultados:
            if resultado in res:
                res[resultado].add(j.licencia)
            else:
                res[resultado] = {j.licencia}
    return res

// Alternativa con defaultdict
def agrupar_por_numero_golpes(jugadores):
    res = defaultdict(set)
    for j in jugadores:
        res[resultado].add(j.licencia)
    return res

################## TEST ###############################################################

def test_jugadores_por_golpes(jugadores):
    print("Jugadores por golpes")
    res = jugadores_por_golpes(jugadores)
    muestra_iterable(res)

if __name__=="__main__":
    jugadores = lee_jugadores("../data/jugadores.csv")
    test_jugadores_por_golpes(jugadores)
```

Ejercicio 4

```
def promedio_ultimos_resultados(jugadores, f1=None, f2=None):
    return [(j.licencia, promedio_golpes(j.resultados))
                for j in jugadores if j.senior == True and en_fecha(j, f1, f2)]

def promedio_golpes(resultados):
    return statistics.mean(resultados)

// Versión hacienda el cálculo
def promedio_golpes(resultados):
    res = None
    if len(resultados) > 0:
        res = sum(resultados) / len(resultados)
    return res

def en_fecha(jugador, f1, f2):
    res = False
    if f1 == None and f2 == None:
        res = True
    elif f1 == None:
        res = jugador.fec_hor_act.date() <= f2
    elif f2==None:
        res = f1 <= jugador.fec_hor_act.date()
    else:
        res = f1 <= jugador.fec_hor_act.date() <= f2
    return res

// Versión con expresión
def en_fecha(jugador, f1, f2):
  return (f1 == None or f1 <= jugador.fec_hor_act.date()) and
         (f2 == None or jugador.fec_hor_act.date() <= f2))

################## TEST ###############################################################
def test_promedio_ultimos_resultados(jugadores, f1=None, f2=None):
    res = promedio_ultimos_resultados(jugadores, f1, f2)
    print(f"El promedio de cada jugador senior con fecha de actualización entre
{f1} y {f2} es: ")
    print(res)

if __name__=="__main__":
    jugadores = lee_jugadores("../data/jugadores.csv")
    test_promedio_ultimos_resultados(jugadores, date(2020,3,1), date(2020,5,31))
```

Ejercicio 5

```
Solución 1:
def jugador_menor_handicap_por_federacion(jugadores):
    d = agrupa_por_federacion(jugadores)
    return {federacion: mejor_jugador(lista_jugadores) \
                   for federacion, lista_jugadores in d.items()}

def agrupa_por_federacion(jugadores):
    res = dict()
    for j in jugadores:
        if j.federacion in res:
            res[j.federacion].append(j)
        else:
            res[j.federacion]= [j]
       return res

def mejor_jugador (lista_jugadores):
    return min(((j.ape_nom, j.handicap) for j in lista_jugadores),
               key=lambda t:t[1])

Solución 2:
def jugador_menor_handicap_por_federacion(jugadores):
    res = dict()
    for j in jugadores:
        t = (j.ape_nom, j.handicap)
        if j.federacion in res:
            res[j.federacion] = min(t, res[j.federacion], key=lambda t:t[1])
            # Otra opción para calcular el mínimo de dos elementos
            # if j.handicap < res[j.federacion][1]:
            #     res[j.federacion] = t

        else:
            res[j.federacion] = t
    return res

################# TEST ###############################################################

def test_jugador_menor_handicap_por_federación(jugadores):
    res = jugador_menor_handicap_por_federacion(jugadores)
    print("Los mejores jugadores de cada federación son:")
    muestra_iterable(res.items())

if __name__=="__main__":
    jugadores = lee_jugadores("../data/jugadores.csv")
    test_jugador_menor_handicap_por_federación(jugadores)
```

Ejercicio 6

```
def comparativa_de_mejores_resultados_segun_handicap(jugadores):
    d_promedios = promedios_por_handicap(jugadores)
    return diferencias_promedios(d_promedios)

def promedios_por_handicap(jugadores):
    d = dict()
    for j in jugadores:
        clave = j.handicap
        if clave in d:
            d[clave].append(min(j.resultados))
        else:
            d[clave]=[min(j.resultados)]

    return {handicap: media_resultados(mejores_resultados) \
                for handicap, mejores_resultados in d.items()}

def media_resultados(resultados):
    return sum(resultados) / len(resultados)

//Versión alternativa para la media,
def media_resultados2(resultados):
    return statistics.mean(resultados)

def diferencias_promedios(d_promedios):
    promedios_ord=sorted(d_promedios.items())
    return [ (f"{t1[0]} vs {t2[0]}", t1[1] - t2[1])  \
            for t1, t2 in zip(promedios_ord, promedios_ord[1:])]

################## TEST ###############################################################
def test_comparativa_de_mejores_resultados_según_handicap(jugadores):
    print ("\ncomparativa_de_mejores_resultados_según_handicap:")
    res = comparativa_de_mejores_resultados_segun_handicap(jugadores)
    print(res)

if __name__=="__main__":
    jugadores = lee_jugadores("../data/jugadores.csv")
    test_lee_jugadores(jugadores)
    test_comparativa_de_mejores_resultados_según_handicap(jugadores)
```