Gestión de la Información en la Web
Curso 2024–2025
Práctica 2 – Formatos CSV y JSON

Fecha de entrega: domingo 13 de octubre de 2024

Entrega de la práctica
La entrega de la práctica se realizará a través del Campus Virtual de la asignatura mediante un fichero `pr2.py`. El esqueleto de este fichero se puede descargar del Campus Virtual.

Lenguaje de programación
Python 3.11 o superior.

Calificación
Se medirá la corrección mediante tests de unidad. Además de la corrección, se valorará la **calidad, concisión y claridad del código**, la incorporación de **comentarios** explicativos, su **eficiencia** tanto en tiempo como en memoria y la puntuación obtenida en Pylint.

Declaración de autoría e integridad
Todos los ficheros entregados contendrán una cabecera en la que se indique la asignatura, la práctica, el grupo y los autores. Esta cabecera también contendrá la siguiente declaración de integridad:

> *Declaramos que esta solución es fruto exclusivamente de nuestro trabajo personal. No hemos sido ayudados por ninguna otra persona o sistema automático ni hemos obtenido la solución de fuentes externas, y tampoco hemos compartido nuestra solución con otras personas de manera directa o indirecta. Declaramos además que no hemos realizado de manera deshonesta ninguna otra actividad que pueda mejorar nuestros resultados ni perjudicar los resultados de los demás.*

No se corregirá ningún fichero que no venga acompañado de dicha cabecera.

---

1. Formato CSV [5pt]

Considerar el archivo `AccidentesBicicletas_2021.csv`. El archivo contiene información sobre los accidentes en los que se han visto involucradas bicicletas en la ciudad de Madrid durante el año 2021. Se puede encontrar más información del conjunto de datos en `https://datos.madrid.es/FWProjects/egob/Catalogo/Seguridad/Ficheros/Estructura_DS_Accidentes_Bicicletas_desde_2019.pdf`

Se pide crear funciones que realicen las siguientes operaciones.

IMPORTANTE: en las funciones que devuelven diccionarios, el orden en el que aparecen las claves a la hora de imprimirlas por pantalla es completamente irrelevante. Las claves de los diccionarios no tienen una noción de orden, lo único importante es que sean diccionarios Python bien construidos y que las claves tengan asociados los valores correctos.

1. (1pt) `lee_fichero_accidentes(ruta)`
   Leer los datos del archivo y devolverlos en una lista de diccionarios, una por cada línea. El resultado con el fichero `AccidentesBicicletas_2021.csv` debería empezar así:

```
[{'num_expediente': '2021S000047',
  'fecha': '01/01/2021',
  'hora': '11:38:00',
  'localizacion': 'CALL. JOSE BERGAMIN / CALL. FLORENCIO CANO CRISTOBAL',
  'numero': '62',
  'cod_distrito': '14',
  'distrito': 'MORATALAZ',
  'tipo_accidente': 'Caída',
  'estado_meteorológico': 'Despejado',
  'tipo_vehículo': 'Bicicleta',
  'tipo_persona': 'Conductor',
  'rango_edad': 'De 45 a 49 años',
  'sexo': 'Hombre',
  'cod_lesividad': '2',
  'tipo_lesividad': 'Ingreso inferior o igual a 24 horas',
  'coordenada_x_utm': '446.426.917',
  'coordenada_y_utm': '4.473.586.644',
  'positiva_alcohol': 'N',
  'positiva_droga': 'NULL'},
 {'num_expediente': '2021S000142',
  'fecha': '04/01/2021',
  'hora': '11:30:00',
  'localizacion': 'PLAZA. GRECIA / AVDA. ARCENTALES wanda farola 20',
  'numero': '1',
  'cod_distrito': '20',
  'distrito': 'SAN BLAS-CANILLEJAS',
  'tipo_accidente': 'Caída',
  'estado_meteorológico': 'Despejado',
  'tipo_vehículo': 'Bicicleta',
  'tipo_persona': 'Conductor',
  'rango_edad': 'De 15 a 17 años',
  'sexo': 'Hombre',
  'cod_lesividad': '14',
  'tipo_lesividad': 'Sin asistencia sanitaria',
  'coordenada_x_utm': '448.606.811',
  'coordenada_y_utm': '4.476.216.426',
  'positiva_alcohol': 'N',
  'positiva_droga': 'NULL'}]
```

```
...
]
```

2. (1pt) `accidentes_por_distrito_tipo(datos)`
   Obtener un diccionario que por cada distrito (columna `distrito`) y por cada tipo de accidente (columna `tipo_accidente`) indica cuántos accidentes hubo. Esta función acepta como parámetro `datos` la lista de diccionarios obtenida del fichero CSV. El resultado de invocar a esta función con los datos del fichero `AccidentesBicicletas_2021.csv` sería un diccionario como el siguiente:

```
{('MORATALAZ', 'Caída'): 8,
 ('SAN BLAS-CANILLEJAS', 'Caída'): 13,
 ('CENTRO', 'Colisión fronto-lateral'): 14,
 ('RETIRO', 'Alcance'): 10,
 ('CENTRO', 'Caída'): 34,
 ('VILLAVERDE', 'Choque contra obstáculo fijo'): 1,
 ('ARGANZUELA', 'Caída'): 26,
 ('ARGANZUELA', 'Alcance'): 11,
 ('CHAMBERÍ', 'Caída'): 12,
 ('CHAMBERÍ', 'Colisión fronto-lateral'): 10,
 ...
}
```

3. (1pt) `dias_mas_accidentes(datos)`
   Función que devuelve las fechas del día o días con más accidentes, junto con ese número de accidentes, tomando como entrada una lista de diccionarios como en el primer apartado. Se debe devolver un **conjunto de parejas** `(día, número de accidentes)`. La salida esperada para el fichero `AccidentesBicicletas_2021.csv` sería:

```
{('24/04/2021', 9), ('13/10/2021', 9)}
```

4. (2pt) `puntos_negros_distrito(datos, distrito, k)`
   Función que dado un distrito, genera una **lista** con el *top-k* de localizaciones donde más accidentes se han producido. Esa lista incluirá parejas `(localización, número de accidentes)` ordenada de manera descendente por número de accidentes. En caso de empate a número de accidentes, se mostrarán primero las localizaciones con un nombre lexicográficamente mayor, es decir, `'PASEO. PIÑONERO, 4'` deberá aparecer antes de `'CTRA. PARQUE DE ATRACCIONES'`. Para el fichero `'AccidentesBicicletas_2021.csv'` y el distrito `'MONCLOA-ARAVACA'`, el top-16 debería ser:

```
[('PASEO. PIÑONERO, 4', 3),
 ('CTRA. PARQUE DE ATRACCIONES, 1', 3),
 ('CTRA. CIUDAD UNIVERSITARIA, 0', 3),
 ('PASEO. TORRECILLA / PASEO. AZUL', 2),
 ('PASEO. REY, 22', 2),
 ('PASEO. EMBARCADERO, 6', 2),
 ('PASEO. EMBARCADERO / PASEO. AZUL', 2),
 ('PASEO. AZUL / PASEO. EMBARCADERO', 2),
 ('PASEO PIÑONEROS KM 46,708 VIA VERDE CASA DE CAMPO', 2),
 ('CUSTA. SAN VICENTE, 44', 2),
 ('CTRA. GARABITAS, 0', 2),
 ('CTRA. CASTILLA, +00100S', 2),
 ('CALL. JOSE ANTONIO NOVAIS / CALL. RAMIRO DE MAEZTU', 2),
 ('CALL. IRUN, 1', 2),
 ('CALL. ANICETO MARINAS, 74', 2),
 ('AVDA. COMPLUTENSE, 14', 2)
]
```

2. Formato JSON [5pt]

Considerar el archivo `300356-0-monumentos-ciudad-madrid.json`. El archivo contiene información sobre los monumentos de Madrid, y ha sido obtenido de la página web de datos en abierto de la ciudad de Madrid (`https://datos.madrid.es`). En este apartado hay que implementar una serie de funciones para consultar este fichero JSON:

1. (1pt) `leer_monumentos(ruta)`
   Esta función acepta la ruta del fichero JSON y devuelve una lista de monumentos, cada uno representado como un diccionario Python.

2. (1pt) `codigos_postales(monumentos)`
   Esta función debe aceptar la lista de monumentos y devolverá una **lista** de parejas de tipo `(código_postal, número_de_monumentos)` con el número total de documentos que hay en cada código postal. Estas parejas deben de aparecer en orden descendente por número de monumentos, y en caso de empate se debe conservar el orden en el que aparecen en el fichero JSON. Un ejemplo de su salida sería:

```
[('', 1491),
 ('28046', 38),
 ('28003', 28),
 ('28032', 22),
 ('28010', 21),
 ('28027', 19),
 ('28002', 14),
 ('28008', 13),
 ('28036', 12),
 ('28015', 12),
 ('28040', 12),
 ('28020', 12),
 ('28004', 10),
 ('28013', 9),
 ('28006', 9),
 ('28042', 9),
...
]
```

3. (1pt) `busqueda_palabras_clave(monumentos, palabras)`
   Esta función recibe una lista de monumentos y una lista de palabras clave, y devuelve un **conjunto de parejas** `(título, distrito)` de aquellos monumentos que contienen las palabras clave en su título o en su descripción (campo `'organization-desc'`). Es necesario que aparezcan **todas las palabras clave entre los dos campos** para considerar que el monumento encaja con la búsqueda, pero **no hay que tener en cuenta las mayúsculas o minúsculas**. El valor de distrito será la URL almacenada en el campo `"@id"` dentro de `address > district`, y si este campo no existe el valor de `distrito` será la cadena vacía (`''`).

   La salida esperada para los monumentos que encajan con las palabras clave `['escultura', 'agua']` sería el siguiente conjunto:

```
{('Alfonso XII',
  'https://datos.madrid.es/egob/kos/Provincia/Madrid/Municipio/Madrid/Distrito/Distrito'),
 ('Bravo Murillo',
  'https://datos.madrid.es/egob/kos/Provincia/Madrid/Municipio/Madrid/Distrito/Chamberi'),
 ('Dodecaedro-Éter',
  'https://datos.madrid.es/egob/kos/Provincia/Madrid/Municipio/Madrid/Distrito/Arganzuela'),
```

```
 ('Estanque Descubrimiento',
  'https://datos.madrid.es/egob/kos/Provincia/Madrid/Municipio/Madrid/Distrito/Salamanca'),
 ('Felipe IV',
  'https://datos.madrid.es/egob/kos/Provincia/Madrid/Municipio/Madrid/Distrito/Centro'),
 ('Francisco de Quevedo y Villegas',
  'https://datos.madrid.es/egob/kos/Provincia/Madrid/Municipio/Madrid/Distrito/Chamberi'),
 ('Fuente Poliedros Maclados en el Cubo',
  'https://datos.madrid.es/egob/kos/Provincia/Madrid/Municipio/Madrid/Distrito/SanBlas'),
 ('Fuente de Cibeles',
  'https://datos.madrid.es/egob/kos/Provincia/Madrid/Municipio/Madrid/Distrito/Retiro'),
 ('Fuente de la Cruz Verde',
  'https://datos.madrid.es/egob/kos/Provincia/Madrid/Municipio/Madrid/Distrito/Centro'),
 ('Fuente de la Unión y el Fénix',
  'https://datos.madrid.es/egob/kos/Provincia/Madrid/Municipio/Madrid/Distrito/Chamberí'),
 ...
}
```

4. (2pt) `busqueda_distancia(monumentos, direccion, distancia)`
   Función que devuelve una **lista de ternas** `(título, id, distancia en kilómetros)` de los monumentos que se encuentran a menos de la distancia indicada desde la dirección pasada como parámetro. La lista de monumentos deberá estar ordenada de más cercano a lejano, y **en caso de empate a distancia se debe preservar el orden original de los monumentos en el listado original**.

   Para conocer la latitud y longitud de una dirección usaremos la biblioteca GeoPy. Aquí tenéis un ejemplo de uso para conocer la latitud y longitud de la dirección donde se encuentra la Facultad de Informática, `'Profesor José García Santesmases 9, Madrid, España'`:

```
>>> from geopy.geocoders import Nominatim

>>> geolocator = Nominatim(user_agent="GIW_pr2")
>>> location = geolocator.geocode("Profesor José García Santesmases 9, Madrid, España",
             addressdetails=True)
>>> print((location.latitude, location.longitude))
(40.4527989, -3.733756378650712)
```

   Por otro lado, para calcular la distancia entre dos puntos dados como **(latitud, longitud)**[1] usaremos la misma biblioteca GeoPy, en este caso el módulo `distance`:

```
>>> from geopy import distance

>>> wellington = (-41.32, 174.81) # Aeropuerto de Wellington
>>> salamanca = (40.96, -5.50) # Aeropuerto de Salamanca

>>> print(distance.distance(wellington, salamanca).km)
19959.67926735382
```

   IMPORTANTE: existen algunas entradas del fichero JSON que no contienen información de localización, y vuestra función debe manejarlas sin fallar.

   La salida esperada de los documentos a menos de 1 km de la calle `"Profesor José García Santesmases 9, Madrid, España"` sería la siguiente:

```
[('José Ortega y Gasset', '400073', 0.3759164404192351),
```

[1] Es muy importante usar parejas (latitud, longitud) y no al revés para invocar a la función `distance.distance`: `https://geopy.readthedocs.io/en/stable/#module-geopy.distance`

```
('Camilo José Cela', '408914', 0.46455683243115004),
('Alfonso XIII', '400063', 0.6612413986049323),
('Puerta de Hierro', '400408', 0.9218447173625832),
('Blas Lázaro e Ibiza', '409134', 0.9949662127532939)]
```