

# BIG DATA Y PYTHON

**MÁSTER EN BIG DATA**

201020

GABRIEL MARÍN DÍAZ

**hola**

# Presentación

---

---

Yo mismo

**Nombre:** Gabriel Marín Díaz

**A qué me dedico...**

- Channel Enablement Manager en Sage
- Profesor Asociado UCM

**Perfil de LinkedIn:** <https://www.linkedin.com/in/gabrielmarindiaz/>

# **CONTENIDO**

# Contenido

---

---

## Resumen

Tema 1 – Visión General

Tema 2 – Introducción a SQL

Tema 3 – Introducción al Lenguaje Python

Tema 4 – HTML y Python

Tema 5 – Big Data y Python

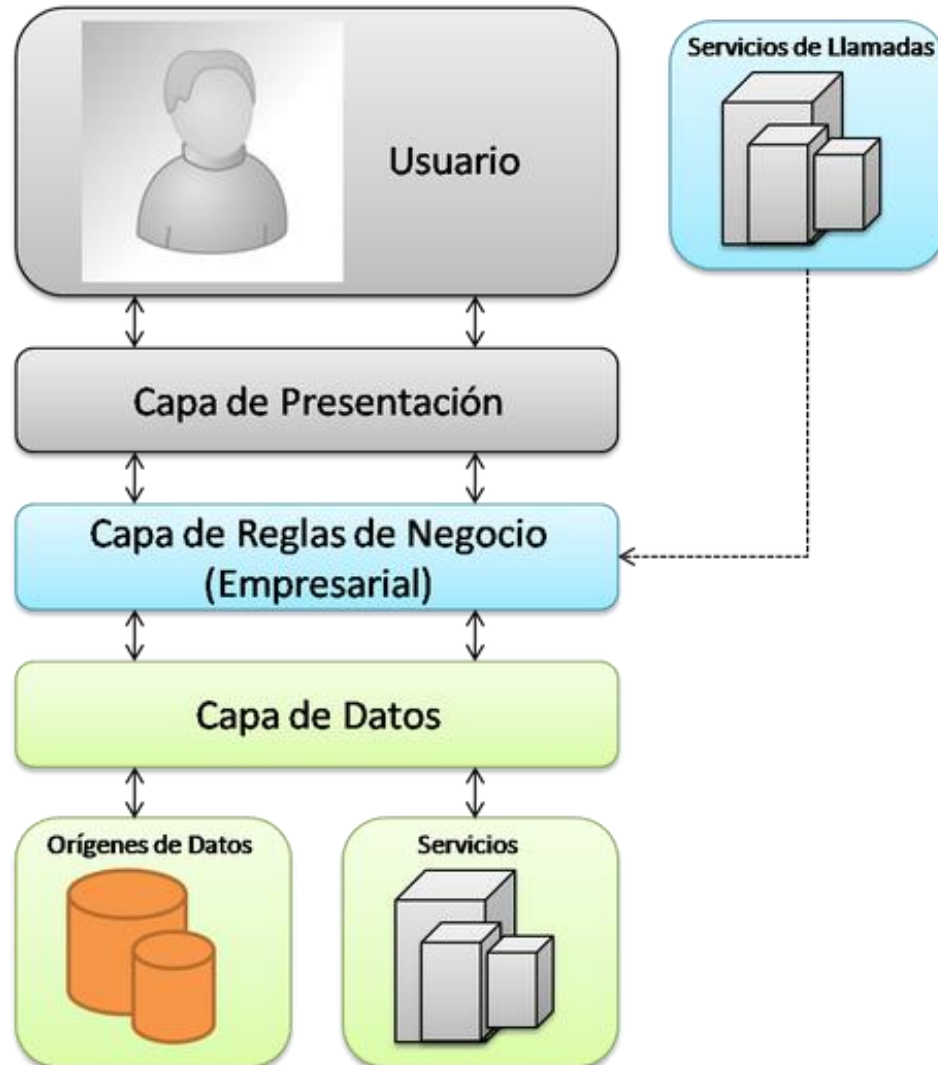
Tema 6 – Procesamiento Distribuido (Spark)

**Prácticas** las realizaremos con Python, MySQL, MongoDB, Apache Spark

**Arrancamos?**

**CONTEXTO**

# Arquitectura en capas



**1. Capa de Presentación:** Interacción entre el usuario y el software. Puede ser tan simple como un menú basado en líneas de comando o tan complejo como una aplicación basada en formas. Su principal función es mostrar información al usuario, interpretar los comandos de este y realizar algunas validaciones simples de los datos ingresados.

**2. Capa de Reglas de Negocio (Empresarial):** También denominada Lógica de Dominio, esta capa contiene la funcionalidad que implementa la aplicación. Involucra cálculos basados en la información dada por el usuario, datos almacenados y validaciones. Controla la ejecución de la capa de acceso a datos y servicios externos.

**3. Capa de Datos:** Esta capa contiene la lógica de comunicación con otros sistemas que llevan a cabo tareas por la aplicación. Para el caso de aplicaciones empresariales, está representado por una base de datos, que es responsable del almacenamiento persistente de información. Esta capa debe abstraer completamente a las capas superiores (negocio) del dialecto utilizado para comunicarse con los repositorios de datos (PL/SQL, Transact-SQL, etc.).



# Índice

- ☐ Actividades
- ☐ Lectura de Ficheros
- ☐ Web Scraping
- ☐ Uso de APIs
- ☐ Mongo DB
- ☐ Procesamiento Distribuido con SPARK
- ☐ Visualización de Resultados

# **EJERCICIOS**

# Ejercicios

1. Revisar que son las funciones Lambda y poner un ejemplo.
2. Definir una función `max_de_tres()`, que tome tres números como argumentos y devuelva el mayor de ellos.
3. Escribir una función que tome un carácter y devuelva `True` si es una vocal, de lo contrario devuelve `False`.
4. Escribir una función `sum()` y una función `multip()` que sumen y multipliquen respectivamente todos los números de una lista. Por ejemplo: `sum([1,2,3,4])` debería devolver 10 y `multip([1,2,3,4])` debería devolver 24.
5. Escribir un pequeño programa donde:
  - Se ingresa el año en curso.
  - Se ingresa el nombre y el año de nacimiento de tres personas.
  - Se calcula cuántos años cumplirán durante el año en curso.

Para obtener datos por el teclado es necesario utilizar la función `input()`

# Ejercicios

6. Definir una tupla con 10 edades de personas. Imprimir la cantidad de personas con edades superiores a 20.  
Puedes variar el ejercicio para que sea el usuario quien ingrese las edades.
7. Definir una lista con un conjunto de nombres, imprimir la cantidad de comienzan con la letra a.  
También se puede hacer elegir al usuario la letra a buscar. (Un poco mas emocionante).
8. Crear una función contar\_vocales(), que reciba una palabra y cuente cuantas letras "a" tiene, cuantas letras "e" tiene y así hasta completar todas las vocales. Se puede hacer que el usuario sea quien elija la palabra.
9. Escribir una función es\_bisiesto() que determine si un año determinado es un año bisiesto. Un año bisiesto es divisible por 4, pero no por 100. También es divisible por 400.
10. Escribe un programa que pida dos palabras y diga si riman o no. Si coinciden las tres últimas letras tiene que decir que riman. Si coinciden sólo las dos últimas tiene que decir que riman un poco y si no, que no riman.

**Para obtener datos por el teclado es necesario utilizar la función input()**

# **CONTINUAMOS CON PYTHON...**

## **LECTURA DE FICHEROS**

# Índice

- ☐ Lectura de Ficheros
- ☐ Web Scraping
- ☐ Uso de APIs
- ☐ Mongo DB
- ☐ Procesamiento Distribuido con SPARK
- ☐ Visualización de Resultados

**VAMOS A REALIZAR LA LECTURA DE UN  
FICHERO .CSV**

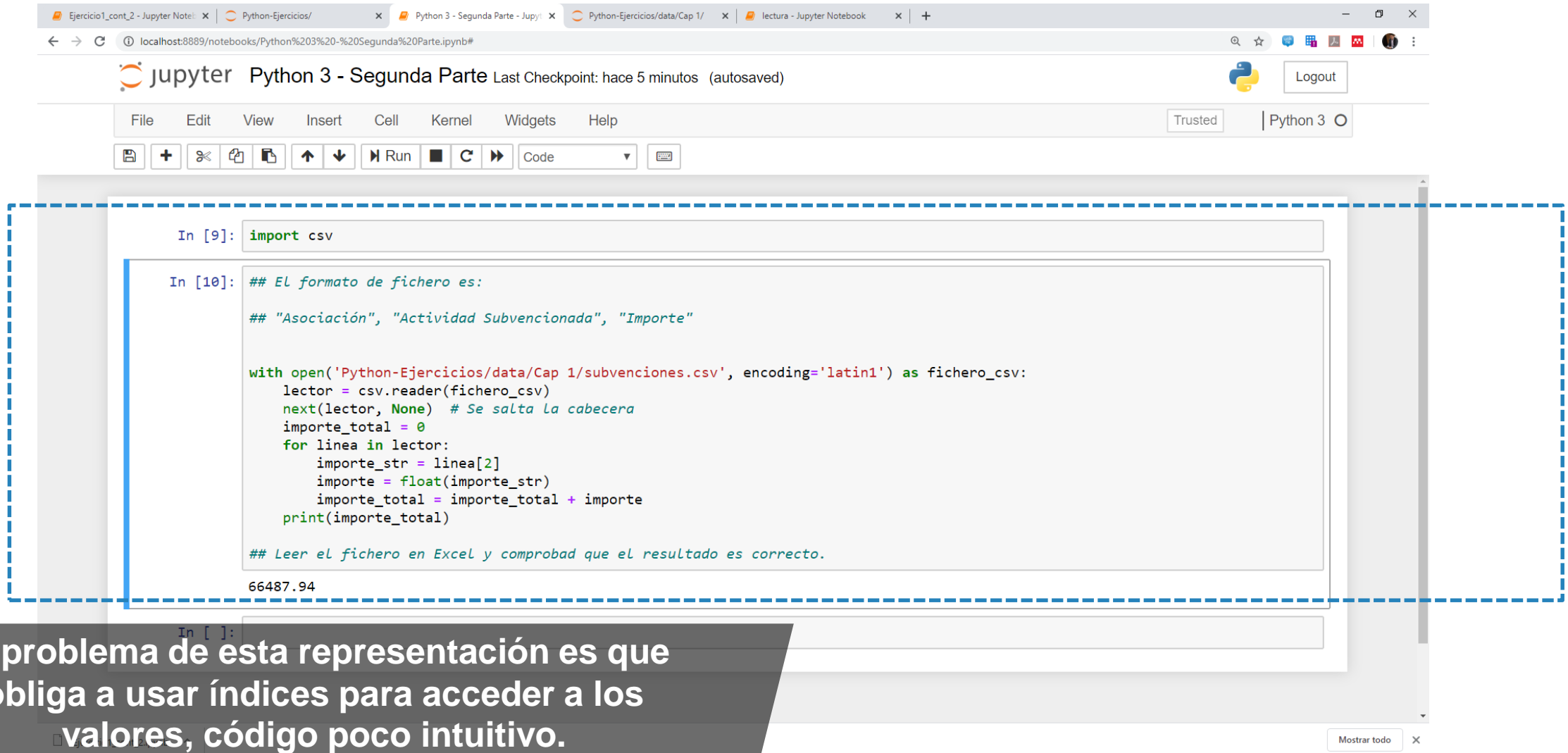
**ESTE FICHERO CORRESPONDE CON LAS  
SUBVENCIONES EN EL ÁMBITO EDUCATIVO  
ASIGNADAS EN 2016 POR EL  
AYUNTAMIENTO DE ALCOBENDAS.**



Click me!

# Lecturas de Ficheros

CSV



```
In [9]: import csv

In [10]: ## El formato de fichero es:

## "Asociación", "Actividad Subvencionada", "Importe"

with open('Python-Ejercicios/data/Cap 1/subvenciones.csv', encoding='latin1') as fichero_csv:
    lector = csv.reader(fichero_csv)
    next(lector, None) # Se salta la cabecera
    importe_total = 0
    for linea in lector:
        importe_str = linea[2]
        importe = float(importe_str)
        importe_total = importe_total + importe
    print(importe_total)

## Leer el fichero en Excel y comprobad que el resultado es correcto.

66487.94

In [ ]:
```

El problema de esta representación es que obliga a usar índices para acceder a los valores, código poco intuitivo.



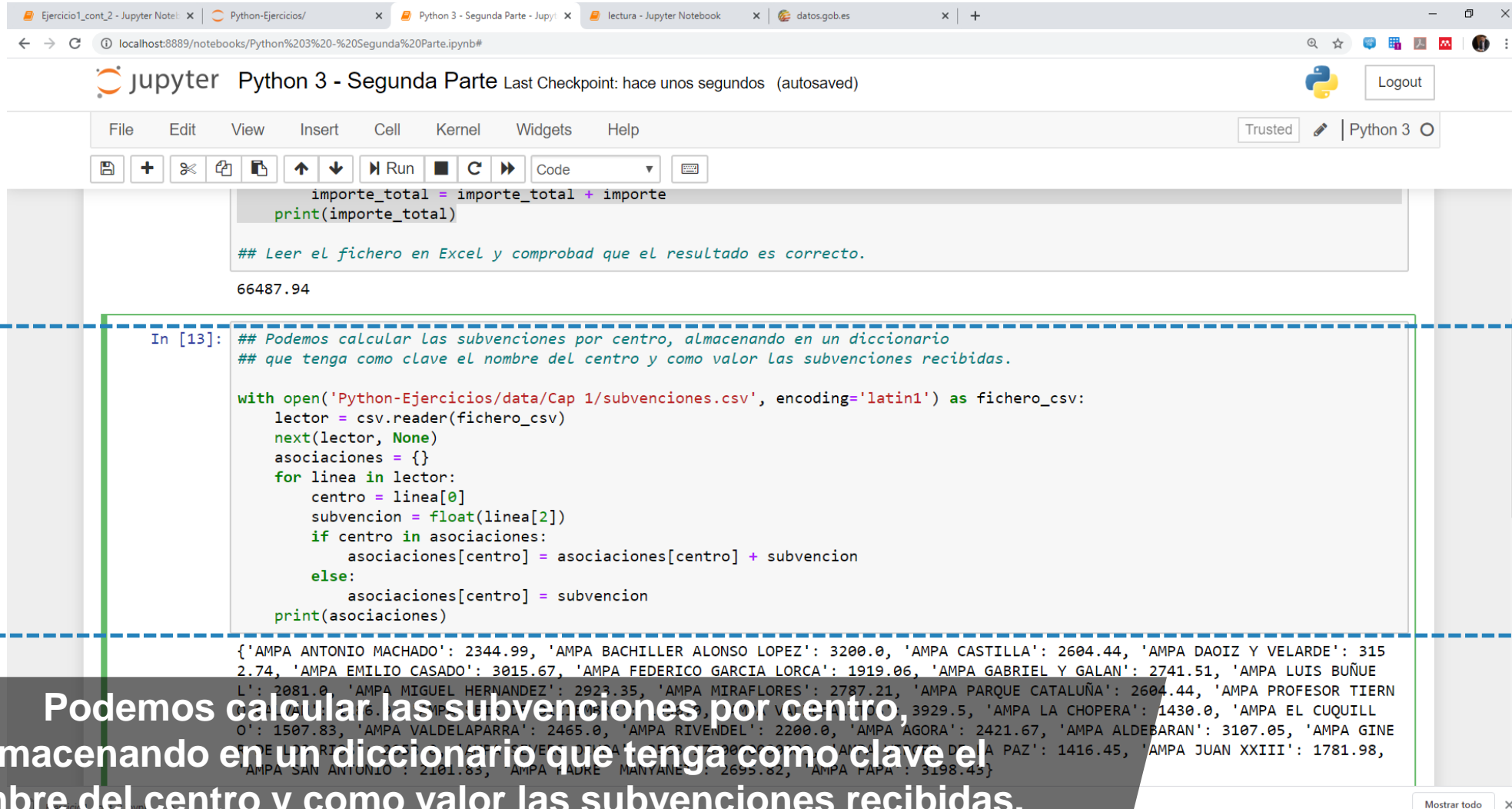
# EJERCICIO

---

**CALCULAR LAS SUBVENCIONES POR CENTRO, ALMACENANDO EN UN DICCIONARIO COMO CLAVE EL NOMBRE DEL CENTRO Y COMO VALOR LAS SUBVENCIONES PERCIBIDAS**

# Lecturas de Ficheros

CSV



```
importe_total = importe_total + importe
print(importe_total)

## Leer el fichero en Excel y comprobad que el resultado es correcto.

66487.94

In [13]: ## Podemos calcular las subvenciones por centro, almacenando en un diccionario
## que tenga como clave el nombre del centro y como valor las subvenciones recibidas.

with open('Python-Ejercicios/data/Cap 1/subvenciones.csv', encoding='latin1') as fichero_csv:
    lector = csv.reader(fichero_csv)
    next(lector, None)
    asociaciones = {}
    for linea in lector:
        centro = linea[0]
        subvencion = float(linea[2])
        if centro in asociaciones:
            asociaciones[centro] = asociaciones[centro] + subvencion
        else:
            asociaciones[centro] = subvencion
    print(asociaciones)

{'AMPA ANTONIO MACHADO': 2344.99, 'AMPA BACHILLER ALONSO LOPEZ': 3200.0, 'AMPA CASTILLA': 2604.44, 'AMPA DAOIZ Y VELARDE': 315
2.74, 'AMPA EMILIO CASADO': 3015.67, 'AMPA FEDERICO GARCIA LORCA': 1919.06, 'AMPA GABRIEL Y GALAN': 2741.51, 'AMPA LUIS BUÑUE
L': 2081.0, 'AMPA MIGUEL HERNANDEZ': 2923.35, 'AMPA MIRAFLORES': 2787.21, 'AMPA PARQUE CATALUÑA': 2604.44, 'AMPA PROFESOR TIERN
O': 1507.83, 'AMPA VALDELAPARRA': 2465.0, 'AMPA RIVENDEL': 2200.0, 'AMPA AGORA': 2421.67, 'AMPA ALDEBARAN': 3107.05, 'AMPA GINE
RE URBANO DE LA PAZ': 1416.45, 'AMPA JUAN XXIII': 1781.98,
'AMPA SAN ANTONIO': 2101.83, 'AMPA PADRE MANYANEN': 2693.82, 'AMPA PAPA': 3198.45}
```

Podemos calcular las subvenciones por centro, almacenando en un diccionario que tenga como clave el nombre del centro y como valor las subvenciones recibidas.

# Lecturas de Ficheros

CSV



The screenshot shows a Jupyter Notebook titled "Python 3 - Segunda Parte" with a "Last Checkpoint: hace un minuto (unsaved changes)". The interface includes a menu bar (File, Edit, View, Insert, Cell, Kernel, Widgets, Help) and a toolbar with icons for saving, running, and other actions. The code is written in a cell and is highlighted with a dashed blue border. The code reads a CSV file named "subvenciones.csv" and processes the data into a dictionary. The output of the code is a large dictionary of school names and their corresponding values.

```
In [12]: ## El problema de esta representación es que obliga a usar índices para acceder a los valores,  
## código poco intuitivo.  
## Con DictReader se devuelve un lector en el que cada línea es un diccionario.  
  
with open('Python-Ejercicios/data/Cap 1/subvenciones.csv', encoding='latin1') as fichero_csv:  
    dict_lector = csv.DictReader(fichero_csv)  
    asocs = {}  
    for linea in dict_lector:  
        centro = linea['Asociación']  
        subvencion = float(linea['Importe'])  
        if centro in asocs:  
            asocs[centro] = asocs[centro] + subvencion  
        else:  
            asocs[centro] = subvencion  
    print(asocs)  
  
## Leer el fichero en Excel y comprobad que el resultado es correcto.  
  
{'AMPA ANTONIO MACHADO': 2344.99, 'AMPA BACHILLER ALONSO LOPEZ': 3200.0, 'AMPA CASTILLA': 2604.44, 'AMPA DAOIZ Y VELARDE': 315  
2.74, 'AMPA EMILIO CASADO': 3015.67, 'AMPA FEDERICO GARCIA LORCA': 1919.06, 'AMPA GABRIEL Y GALAN': 2741.51, 'AMPA LUIS BUÑUE  
L': 2081.0, 'AMPA MIGUEL HERNANDEZ': 2923.35, 'AMPA MIRAFLORES': 2787.21, 'AMPA PARQUE CATALUÑA': 2604.44, 'AMPA PROFESOR TIERN  
O GALVÁN': 1286.0, 'AMPA SEIS DE DICIEMBRE': 1950.0, 'AMPA VALDEPALITOS': 3929.5, 'AMPA LA CHOPERA': 1430.0, 'AMPA EL CUQUILL  
O': 1507.83, 'AMPA VALDELAPARRA': 2465.0, 'AMPA RIVENDEL': 2200.0, 'AMPA AGORA': 2421.67, 'AMPA ALDEBARAN': 3107.05, 'AMPA GINE  
R DE LOS RIOS': 2058.0, 'AMPA SEVERO OCHOA': 3563.9700000000003, 'AMPA VIRGEN DE LA PAZ': 1416.45, 'AMPA JUAN XXIII': 1781.98,  
'AMPA SAN ANTONIO': 2101.83, 'AMPA PADRE MANYANET': 2695.82, 'AMPA FAPA': 3198.43}
```

Utilizaremos DictReader para evitar el uso de índices.

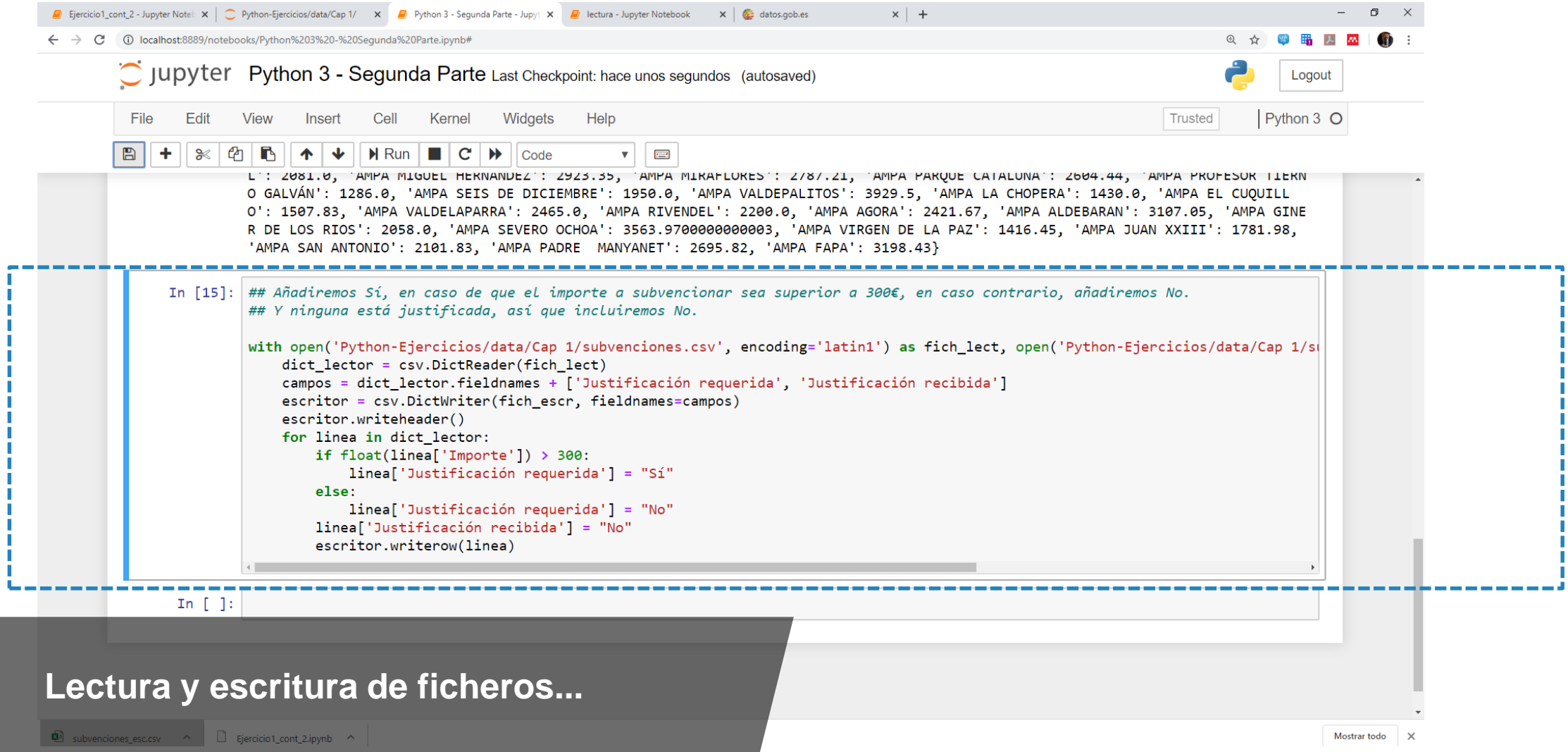
# Lectura de Ficheros

---

**UNA VEZ EL FICHERO HA SIDO CARGADO,  
ESTAMOS INTERESADOS EN MANIPULARLO  
Y ALMACENAR LOS RESULTADOS  
OBTENIDOS.**

# Lecturas de Ficheros

CSV



The screenshot shows a Jupyter Notebook titled "Python 3 - Segunda Parte" with a "Trusted" status. The interface includes a menu bar (File, Edit, View, Insert, Cell, Kernel, Widgets, Help) and a toolbar with icons for saving, running, and other actions. The notebook contains two code cells. The first cell, labeled "In [15]:", contains a multi-line string of CSV data representing school information. The second cell, labeled "In [ ]:", contains Python code that reads a CSV file named "subvenciones.csv" and writes a new CSV file named "subvenciones\_esc.csv". The code uses the `csv` module's `DictReader` and `DictWriter` classes. The code in the second cell is highlighted with a dashed blue border.

```
In [15]: ## Añadiremos Sí, en caso de que el importe a subvencionar sea superior a 300€, en caso contrario, añadiremos No.
## Y ninguna está justificada, así que incluiremos No.

with open('Python-Ejercicios/data/Cap 1/subvenciones.csv', encoding='latin1') as fich_lect, open('Python-Ejercicios/data/Cap 1/subvenciones_esc.csv', encoding='latin1') as fich_escr:
    dict_lector = csv.DictReader(fich_lect)
    campos = dict_lector.fieldnames + ['Justificación requerida', 'Justificación recibida']
    escritor = csv.DictWriter(fich_escr, fieldnames=campos)
    escritor.writeheader()
    for linea in dict_lector:
        if float(linea['Importe']) > 300:
            linea['Justificación requerida'] = "Sí"
        else:
            linea['Justificación requerida'] = "No"
        linea['Justificación recibida'] = "No"
        escritor.writerow(linea)
```

In [ ]:

subvenciones\_esc.csv    Ejercicio1\_cont\_2.ipynb

Lectura y escritura de ficheros...

# Lectura de Ficheros

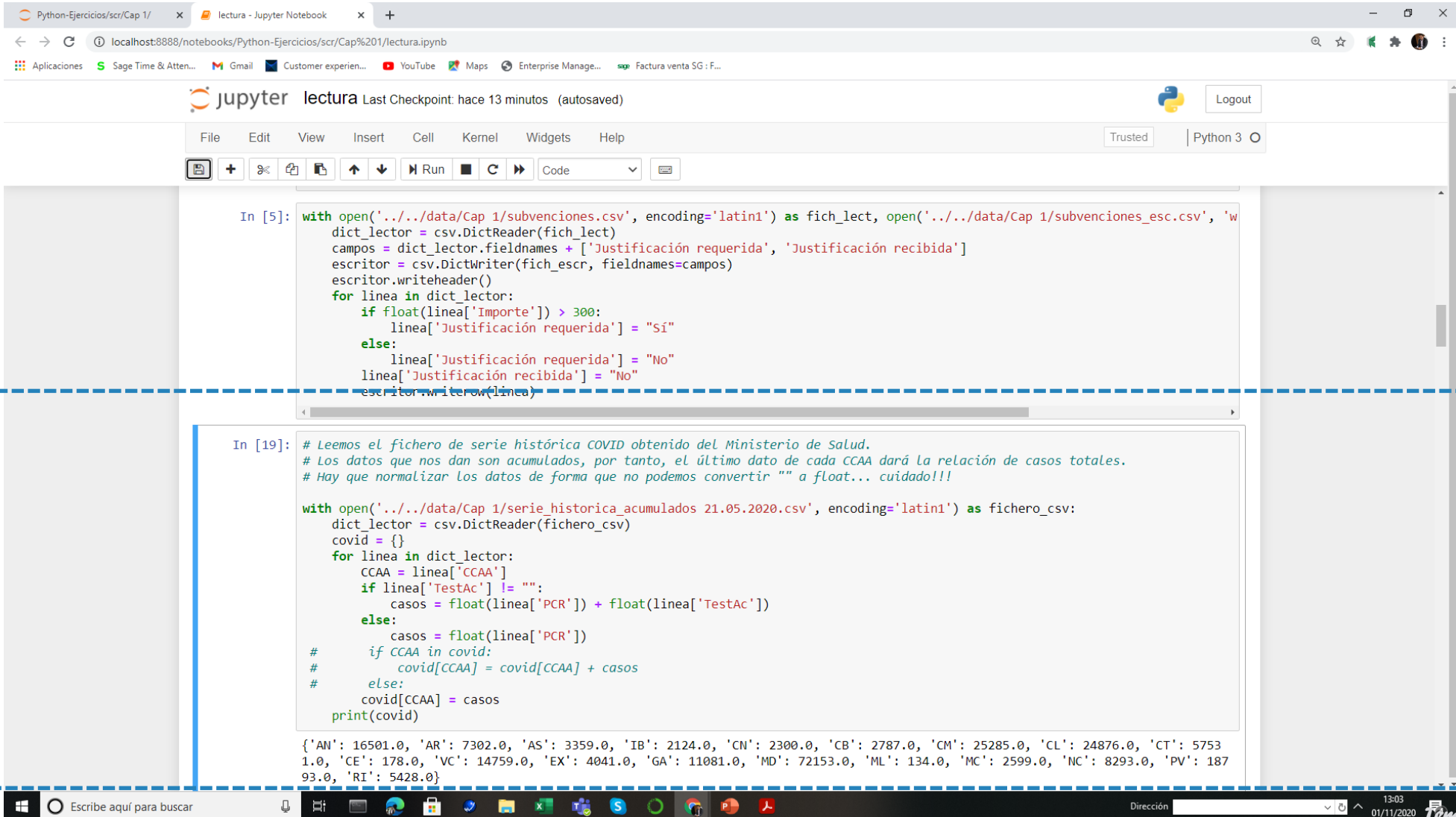
---

**PROVEMOS A HACER ALGO PARECIDO CON  
EL FICHERO DEL HISTÓRICO DE  
ACUMULADOS COVID, OS LO DEJO Y  
HACEMOS UNA PRUEBA. QUEREMOS  
OBTENER LOS VALORES DEL NÚMERO DE  
CASOS POR COMUNIDAD AUTÓNOMA**

Click me!

# Lecturas de Ficheros

CSV



```
In [5]: with open('../data/Cap 1/subvenciones.csv', encoding='latin1') as fich_lect, open('../data/Cap 1/subvenciones_esc.csv', 'w') as fich_escr:
    dict_lector = csv.DictReader(fich_lect)
    campos = dict_lector.fieldnames + ['Justificación requerida', 'Justificación recibida']
    escritor = csv.DictWriter(fich_escr, fieldnames=campos)
    escritor.writeheader()
    for linea in dict_lector:
        if float(linea['Importe']) > 300:
            linea['Justificación requerida'] = "Sí"
        else:
            linea['Justificación requerida'] = "No"
        linea['Justificación recibida'] = "No"
    escritor.writerows(linea)

In [19]: # Leemos el fichero de serie histórica COVID obtenido del Ministerio de Salud.
# Los datos que nos dan son acumulados, por tanto, el último dato de cada CCAA dará la relación de casos totales.
# Hay que normalizar los datos de forma que no podemos convertir "" a float... cuidado!!!

with open('../data/Cap 1/serie_historica_acumulados 21.05.2020.csv', encoding='latin1') as fichero_csv:
    dict_lector = csv.DictReader(fichero_csv)
    covid = {}
    for linea in dict_lector:
        CCAA = linea['CCAA']
        if linea['TestAc'] != "":
            casos = float(linea['PCR']) + float(linea['TestAc'])
        else:
            casos = float(linea['PCR'])
        # if CCAA in covid:
        #     covid[CCAA] = covid[CCAA] + casos
        # else:
        covid[CCAA] = casos
    print(covid)

{'AN': 16501.0, 'AR': 7302.0, 'AS': 3359.0, 'IB': 2124.0, 'CN': 2300.0, 'CB': 2787.0, 'CM': 25285.0, 'CL': 24876.0, 'CT': 5753.0, 'CE': 178.0, 'VC': 14759.0, 'EX': 4041.0, 'GA': 11081.0, 'MD': 72153.0, 'ML': 134.0, 'MC': 2599.0, 'NC': 8293.0, 'PV': 187.0, 'RI': 5428.0}
```

# Lectura de Ficheros

---

## TSV

El formato TSV es similar al CSV, pero en el caso de TSV las columnas se separan con tabuladores. Para crear y manipular estos ficheros seguiremos usando la biblioteca **csv**, usando la opción **delimiter='\t'**.



# Lecturas de Ficheros

## TSV



The screenshot shows a Jupyter Notebook interface in a web browser. The browser's address bar shows the URL `localhost:8888/notebooks/Python-Ejercicios/scr/Cap%201/lectura.ipynb`. The Jupyter interface includes a top bar with the Jupyter logo, the title "lectura", and a "Last Checkpoint: hace un minuto (autosaved)" message. Below this is a menu bar with "File", "Edit", "View", "Insert", "Cell", "Kernel", "Widgets", and "Help". A toolbar contains icons for saving, creating new cells, copying, pasting, and running code. The main content area displays the title "TSV" followed by a text box explaining the TSV format and the `csv` library. Below the text is a code cell labeled "In [9]:" containing Python code to read a CSV file and write its contents to a TSV file.

**TSV**

El formato TSV es similar al CSV, pero en el caso de TSV las columnas se separan con tabuladores. Para crear y manipular estos ficheros seguiremos usando la biblioteca **csv**, usando la opción **delimiter='t'**. Por ejemplo, podemos crear un fichero TSV a partir del CSV anterior simplemente usando esta opción al crear el objeto **escritor**:

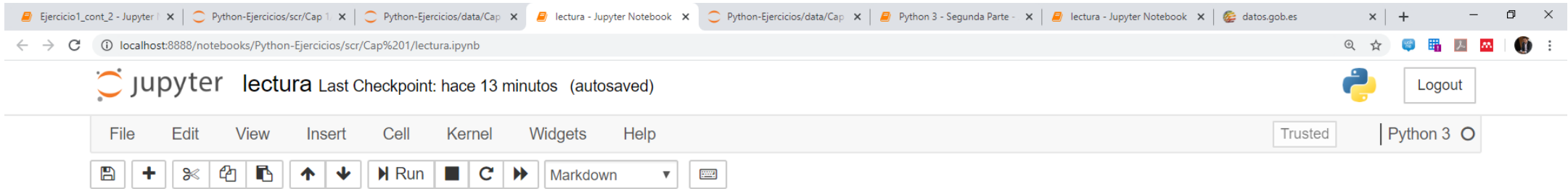
```
In [9]: with open('../data/Cap 1/subvenciones.csv', encoding='latin1') as fich_lect, open('../data/Cap 1/subvenciones.tsv', 'w', encoding='latin1') as fich_escr:
        dict_lector = csv.DictReader(fich_lect)
        campos = dict_lector.fieldnames
        escritor = csv.DictWriter(fich_escr, delimiter='t', fieldnames=campos)
        escritor.writeheader()
        for linea in dict_lector:
            escritor.writerow(linea)
```

**Creamos el formato TSV a partir del CSV**

## **CALCULAD LAS SUBVENCIONES POR CENTRO COMO SE HIZO CON EL FORMATO CSV**

# Lecturas de Ficheros

TSV



The screenshot shows a Jupyter Notebook interface with the following elements:

- Browser Tabs:** Ejercicio1\_cont\_2 - Jupyter, Python-Ejercicios/scr/Cap 1, Python-Ejercicios/data/Cap, lectura - Jupyter Notebook, Python-Ejercicios/data/Cap, Python 3 - Segunda Parte, lectura - Jupyter Notebook, datos.gob.es.
- Address Bar:** localhost:8888/notebooks/Python-Ejercicios/scr/Cap%201/lectura.ipynb
- Jupyter Header:** jupyter lectura Last Checkpoint: hace 13 minutos (autosaved) Logout
- Menu Bar:** File Edit View Insert Cell Kernel Widgets Help
- Toolbar:** Save, New, Copy, Paste, Undo, Redo, Run, Stop, Refresh, Markdown dropdown, Command palette.

Una vez creado el fichero, podemos recorrerlo como hacíamos arriba, eligiendo como separador el tabulador. La función siguiente calcula las subvenciones por centro como hicimos con CSV:

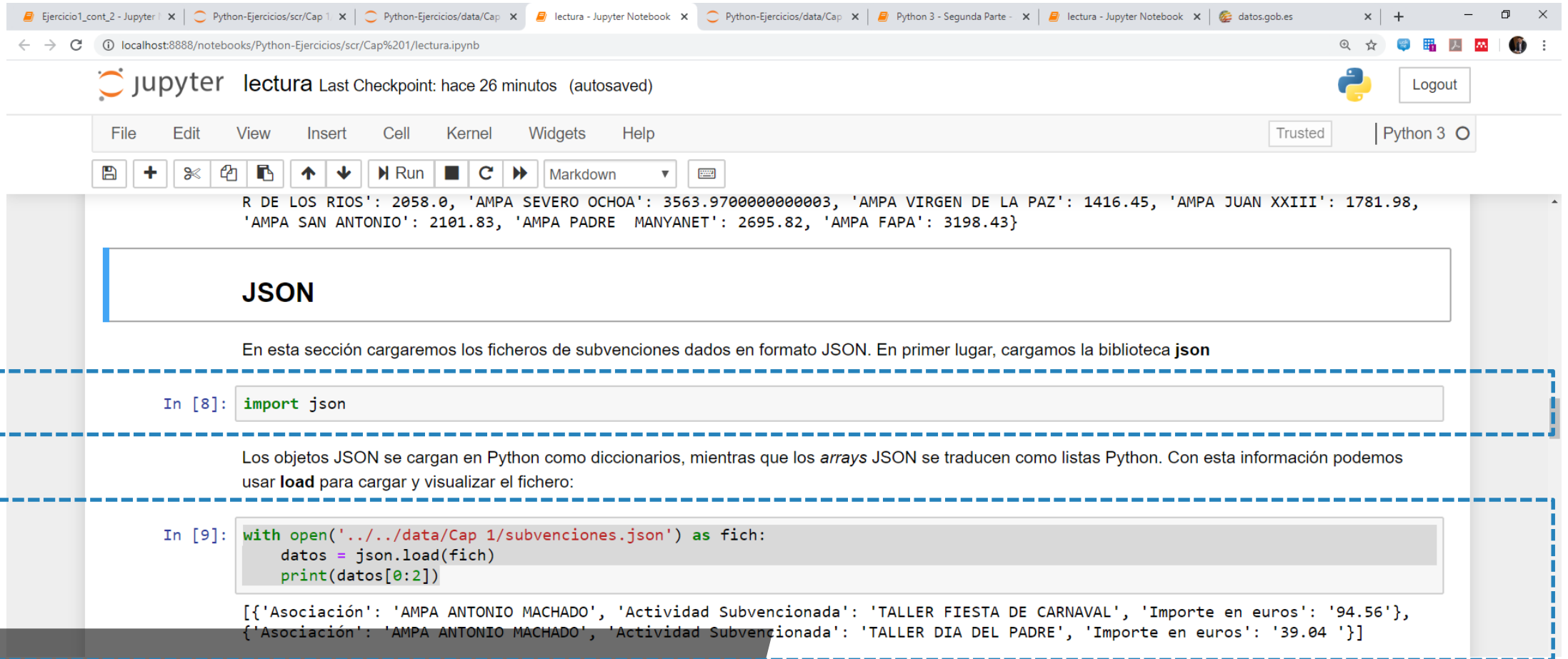
```
In [11]: with open('../data/Cap 1/subvenciones.tsv', encoding='latin1') as fichero:
dict lector = csv.DictReader(fichero, delimiter='\t')
asociaciones = {}
for linea in dict lector:
    centro = linea['Asociación']
    subvencion = float(linea['Importe'])
    if centro in asociaciones:
        asociaciones[centro] = asociaciones[centro] + subvencion
    else:
        asociaciones[centro] = subvencion
print(asociaciones)
```

```
{'AMPA ANTONIO MACHADO': 2344.99, 'AMPA BACHILLER ALONSO LOPEZ': 3200.0, 'AMPA CASTILLA': 2604.44, 'AMPA DAOIZ Y VELARDE': 315
2.74, 'AMPA EMILIO CASADO': 3015.67, 'AMPA FEDERICO GARCIA LORCA': 1919.06, 'AMPA GABRIEL Y GALAN': 2741.51, 'AMPA LUIS BUÑUE
L': 2081.0, 'AMPA MIGUEL HERNANDEZ': 2923.35, 'AMPA MIRAFLORES': 2787.21, 'AMPA PARQUE CATALUÑA': 2604.44, 'AMPA PROFESOR TIERN
O GALVÁN': 1286.0, 'AMPA SEIS DE DICIEMBRE': 1950.0, 'AMPA VALDEPALITOS': 3929.5, 'AMPA LA CHOPERA': 1430.0, 'AMPA EL CUQUILL
O': 1507.83, 'AMPA VALDELAPARRA': 2465.0, 'AMPA RIVENDEL': 2200.0, 'AMPA AGORA': 2421.67, 'AMPA ALDEBARAN': 3107.05, 'AMPA GINE
R DE LOS RIOS': 2058.0, 'AMPA SEVERO OCHOA': 3563.9700000000003, 'AMPA VIRGEN DE LA PAZ': 1416.45, 'AMPA JUAN XXIII': 1781.98,
'AMPA SAN ANTONIO': 2101.83, 'AMPA PADRE MANYANET': 2695.82, 'AMPA FAPA': 3198.43}
```

Subvenciones por centro...

# Lectura de Ficheros

## JSON



The screenshot shows a Jupyter Notebook titled 'lectura' with a 'Python 3' kernel. The notebook content includes a JSON string at the top, followed by a section header 'JSON'. Below this, a paragraph explains that JSON files are loaded into Python as dictionaries. Two code cells are highlighted with dashed blue boxes: the first imports the 'json' module, and the second uses 'with open()' to load a file named 'subvenciones.json' and prints the first two elements of the resulting dictionary list.

```
R DE LOS RIOS': 2058.0, 'AMPA SEVERO OCHOA': 3563.9700000000003, 'AMPA VIRGEN DE LA PAZ': 1416.45, 'AMPA JUAN XXIII': 1781.98, 'AMPA SAN ANTONIO': 2101.83, 'AMPA PADRE MANYANET': 2695.82, 'AMPA FAPA': 3198.43}
```

### JSON

En esta sección cargaremos los ficheros de subvenciones dados en formato JSON. En primer lugar, cargamos la biblioteca **json**

```
In [8]: import json
```

Los objetos JSON se cargan en Python como diccionarios, mientras que los *arrays* JSON se traducen como listas Python. Con esta información podemos usar **load** para cargar y visualizar el fichero:

```
In [9]: with open('../data/Cap 1/subvenciones.json') as fich:
        datos = json.load(fich)
        print(datos[0:2])
```

```
[{'Asociación': 'AMPA ANTONIO MACHADO', 'Actividad Subvencionada': 'TALLER FIESTA DE CARNAVAL', 'Importe en euros': '94.56'}, {'Asociación': 'AMPA ANTONIO MACHADO', 'Actividad Subvencionada': 'TALLER DIA DEL PADRE', 'Importe en euros': '39.04 '}]
```

**Los objetos JSON se cargan en Python como diccionarios, los arrays como listas.**

# Lectura de Ficheros

## JSON

El fichero JSON consiste en una lista de objetos donde cada uno de estos objetos se corresponde con una subvención, es decir, contiene 3 campos correspondientes a "**Asociación**", "**Actividad Subvencionada**" e "**Importe en euros**".

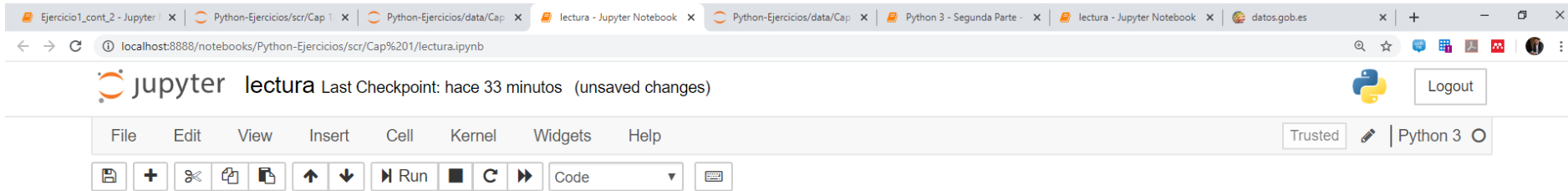
El disponer de un formato más flexible nos permite almacenar la información de manera más compacta: podríamos tener un objeto JSON por cada centro, con los campos "**Asociación**" y "**Actividades**".

Este segundo campo consistiría a su vez en una lista de objetos con los campos "**Actividad Subvencionada**" e "**Importe en euros**".

**ESCRIBIR EL CÓDIGO PYTHON NECESARIO  
PARA TRANSFORMAR NUESTRO FICHERO Y  
GUARDARLO EN UN FICHERO  
SUBVENCIONES\_AGRUPADAS.JSON**

# Lectura de Ficheros

## JSON



```
In [14]: with open('../..../data/Cap 1/subvenciones.json', encoding='utf-8') as fich_lect, open('../..../data/Cap 1/subvenciones_agrupadas.json', encoding='utf-8') as fich_escr:
    data = json.load(fich_lect)
    asoc_str = "Asociación"
    act_str = "Actividad Subvencionada"
    imp_str = "Importe en euros"
    lista = []
    lista_act = []
    asoc_actual = ""
    dicc = {}
    for elem in data:
        asoc = elem[asoc_str]
        act = elem[act_str]
        imp = elem[imp_str]
        if asoc_actual != asoc:
            dicc["Actividades"] = lista_act
            dicc = {"Asociación": asoc}
            lista.append(dicc)
            lista_act = []
            lista_act.append({act_str : act, imp_str : imp})
        asoc_actual = asoc
    print(lista)
    json.dump(lista, fich_escr, ensure_ascii=False, indent=4) # , sort_keys=False
```

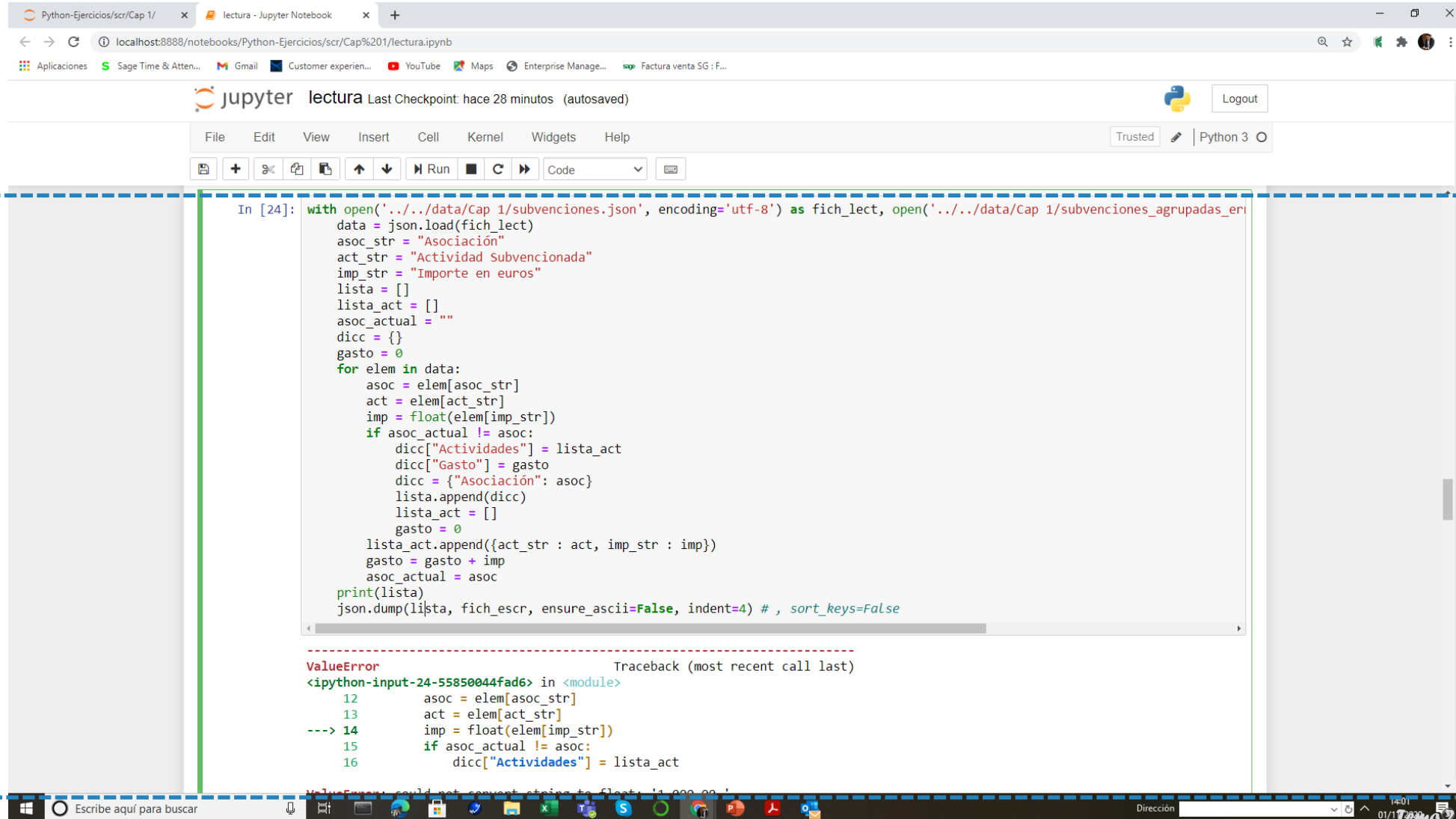
```
[{'Asociación': 'AMPA ANTONIO MACHADO', 'Actividades': [{'Actividad Subvencionada': 'TALLER FIESTA DE CARNAVAL', 'Importe en euros': '94.56'}, {'Actividad Subvencionada': 'TALLER DIA DEL PADRE', 'Importe en euros': '39.04'}, {'Actividad Subvencionada': 'TALLER DIA DE LA MADRE', 'Importe en euros': '43.64'}, {'Actividad Subvencionada': 'FIESTA FIN DE CURSO', 'Importe en euros': '921.00'}, {'Actividad Subvencionada': 'CONCURSO LOGOTIPO AMPA', 'Importe en euros': '56.57'}, {'Actividad Subvencionada': 'ASOCIACION FAPA GINER DE LOS RIOS', 'Importe en euros': '86.79'}, {'Actividad Subvencionada': 'FIESTA DE NAVIDAD', 'Importe en euros': '660.00'}, {'Actividad Subvencionada': 'HALLOWEEN', 'Importe en euros': '168.39'}, {'Actividad Subvencionada': 'SAN ISIDRO', 'Importe en euros': '150.00'}, {'Actividad Subvencionada': 'CONCURSO LOPEZ', 'Importe en euros': '150.00'}], 'Actividades': ['MATEMATICAS MANIPULADA']}]
```

**INTENTEMOS AHORA CALCULAR EL TOTAL  
DEL GASTO POR CADA CENTRO Y  
ALMACENARLO COMO UN NUEVO CAMPO  
DE LA ESTRUCTURA**



# Lectura de Ficheros

## JSON



```
In [24]: with open('../data/Cap 1/subvenciones.json', encoding='utf-8') as fich_lect, open('../data/Cap 1/subvenciones_agrupadas_eri
data = json.load(fich_lect)
asoc_str = "Asociación"
act_str = "Actividad Subvencionada"
imp_str = "Importe en euros"
lista = []
lista_act = []
asoc_actual = ""
dicc = {}
gasto = 0
for elem in data:
    asoc = elem[asoc_str]
    act = elem[act_str]
    imp = float(elem[imp_str])
    if asoc_actual != asoc:
        dicc["Actividades"] = lista_act
        dicc["Gasto"] = gasto
        dicc = {"Asociación": asoc}
        lista.append(dicc)
        lista_act = []
        gasto = 0
    lista_act.append({act_str : act, imp_str : imp})
    gasto = gasto + imp
    asoc_actual = asoc
print(lista)
json.dump(lista, fich_escr, ensure_ascii=False, indent=4) # , sort_keys=False

-----
ValueError                                Traceback (most recent call last)
<ipython-input-24-55850044fad6> in <module>
    12     asoc = elem[asoc_str]
    13     act = elem[act_str]
----> 14     imp = float(elem[imp_str])
    15     if asoc_actual != asoc:
    16         dicc["Actividades"] = lista_act
```

# Ejercicio

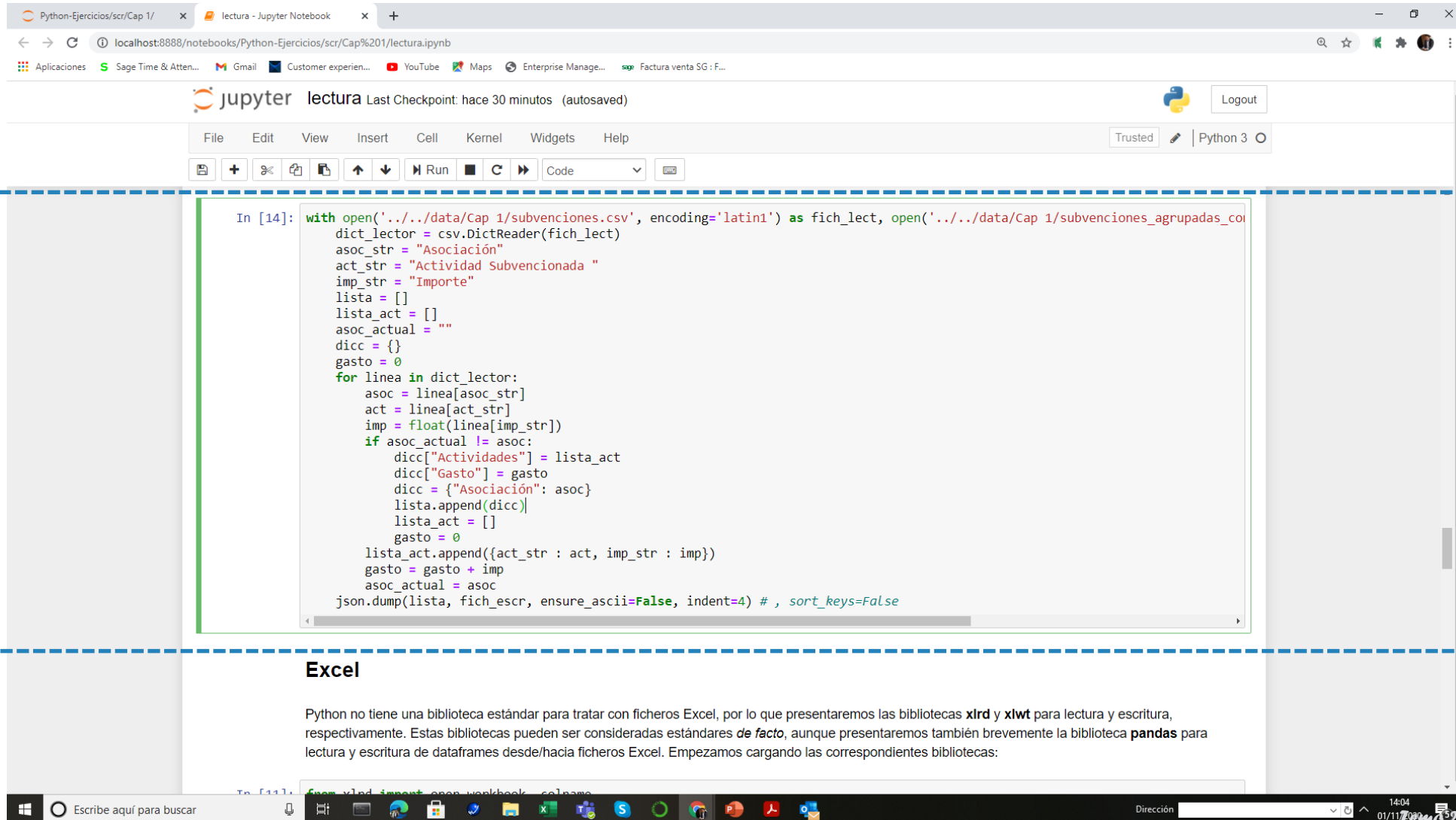
---

JSON

**Como se puede ver... revisando el fichero JSON encontramos que algunos importes se guardan con separadores de miles y otro para separar la parte decimal (p.e. 1.000.00). Veamos como solucionamos este problema, generaremos el JSON a partir del fichero CSV anterior.**

# Lectura de Ficheros

## JSON



Python-Ejercicios/scr/Cap 1/ x lectura - Jupyter Notebook x +

localhost:8888/notebooks/Python-Ejercicios/scr/Cap%201/lectura.ipynb

Aplicaciones Sage Time & Atten... Gmail Customer experien... YouTube Maps Enterprise Manage... Factura venta SG : F...

jupyter lectura Last Checkpoint: hace 30 minutos (autosaved) Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

```
In [14]: with open('../data/Cap 1/subvenciones.csv', encoding='latin1') as fich_lect, open('../data/Cap 1/subvenciones_agrupadas_co
dict_lector = csv.DictReader(fich_lect)
asoc_str = "Asociación"
act_str = "Actividad Subvencionada "
imp_str = "Importe"
lista = []
lista_act = []
asoc_actual = ""
dicc = {}
gasto = 0
for linea in dict_lector:
    asoc = linea[asoc_str]
    act = linea[act_str]
    imp = float(linea[imp_str])
    if asoc_actual != asoc:
        dicc["Actividades"] = lista_act
        dicc["Gasto"] = gasto
        dicc = {"Asociación": asoc}
        lista.append(dicc)
        lista_act = []
        gasto = 0
    lista_act.append({act_str : act, imp_str : imp})
    gasto = gasto + imp
    asoc_actual = asoc
json.dump(lista, fich_escr, ensure_ascii=False, indent=4) # , sort_keys=False
```

Excel

Python no tiene una biblioteca estándar para tratar con ficheros Excel, por lo que presentaremos las bibliotecas **xlrld** y **xlwt** para lectura y escritura, respectivamente. Estas bibliotecas pueden ser consideradas estándares **de facto**, aunque presentaremos también brevemente la biblioteca **pandas** para lectura y escritura de dataframes desde/hacia ficheros Excel. Empezamos cargando las correspondientes bibliotecas:

# Lectura de Ficheros

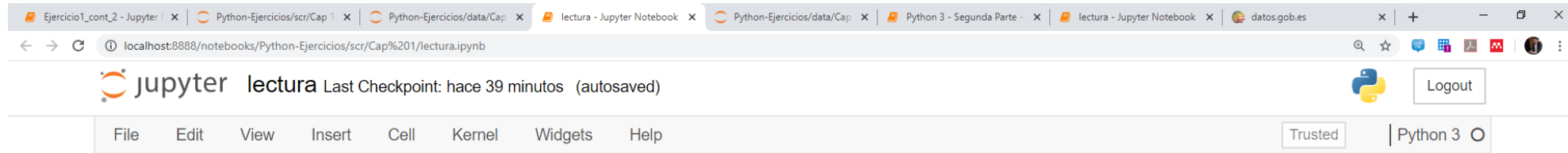
---

## EXCEL

Python no tiene una biblioteca estándar para tratar con ficheros Excel, por lo que presentaremos las bibliotecas **xlrd** y **xlwt** para lectura y escritura, respectivamente. Estas bibliotecas pueden ser consideradas estándares *de facto*, aunque presentaremos también brevemente la biblioteca **pandas** para lectura y escritura de dataframes desde/hacia ficheros Excel.

# Lectura de Ficheros

EXCEL



```
In [15]: from xlrd import open_workbook, colname
import xlwt
```

Empezaremos calculando la subvención total recibida por centro:

```
In [16]: with open_workbook('../data/Cap 1/subvenciones.xls',on_demand=True) as libro:
    asociaciones = {}
    for nombre in libro.sheet_names():
        hoja = libro.sheet_by_name(nombre)
        for i in range(1,hoja.nrows):
            fila = hoja.row(i)
            centro = fila[0].value
            subvencion = fila[2].value
            #print(fila[0].ctype)
            #print(fila[2].value)
            if centro in asociaciones:
                asociaciones[centro] = asociaciones[centro] + subvencion
            else:
                asociaciones[centro] = subvencion
    print(asociaciones)
```

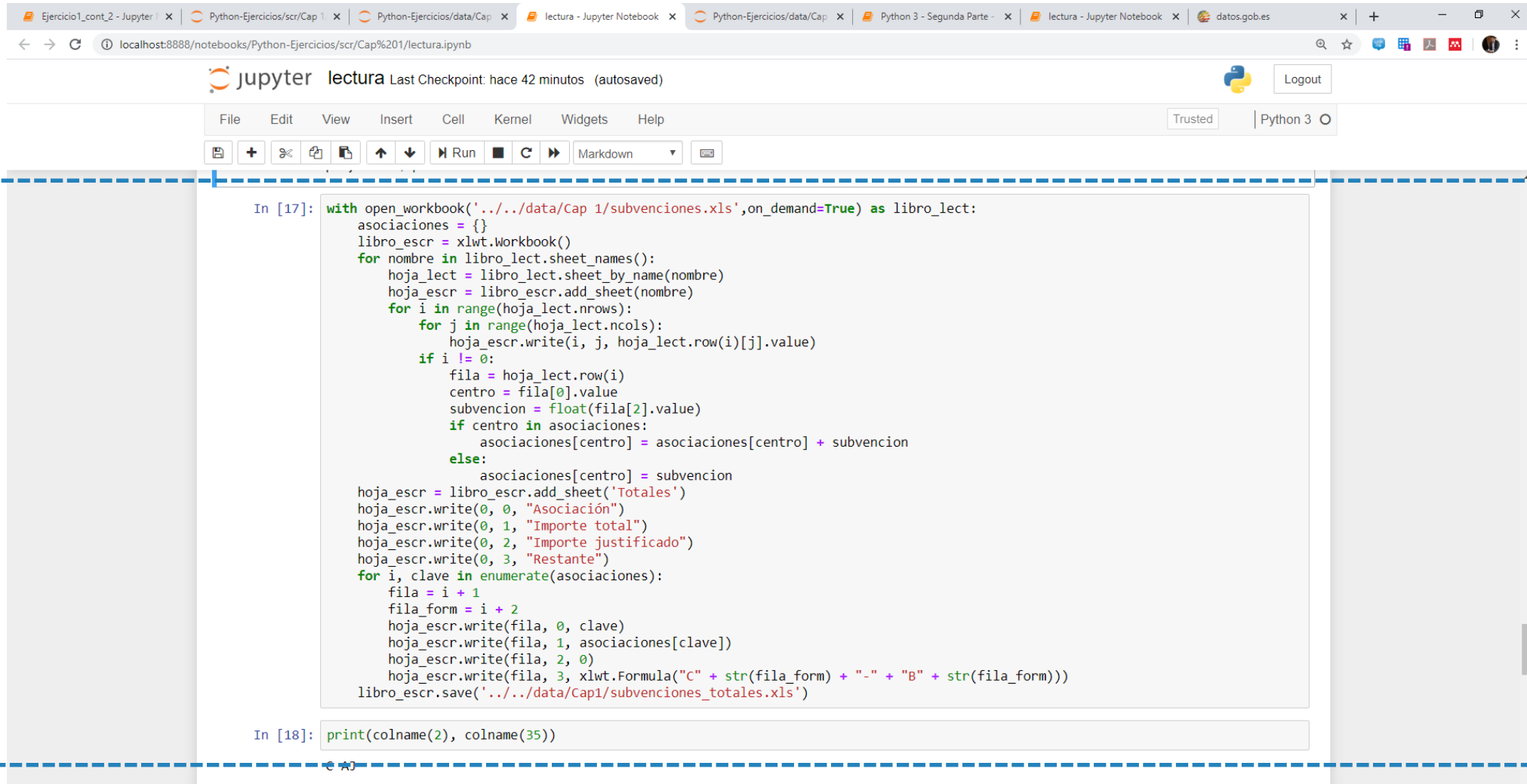
```
{'AMPA ANTONIO MACHADO': 2344.99, 'AMPA BACHILLER ALONSO LOPEZ': 3200.0, 'AMPA CASTILLA': 2604.44, 'AMPA DAOIZ Y VELARDE': 315
2.74, 'AMPA EMILIO CASADO': 3015.67, 'AMPA FEDERICO GARCIA LORCA': 1919.06, 'AMPA GABRIEL Y GALAN': 2741.51, 'AMPA LUIS BUÑUE
L': 2081.0, 'AMPA MIGUEL HERNANDEZ': 2923.35, 'AMPA MIRAFLORES': 2787.21, 'AMPA PARQUE CATALUÑA': 2604.44, 'AMPA PROFESOR TIERN
O': 1507.83, 'AMPA VALDELAFAÑA': 2465.0, 'AMPA VALDEPALITOS': 3929.5, 'AMPA LA CHOPERA': 1430.0, 'AMPA EL CUQUILL
O': 1507.83, 'AMPA VALDELAFAÑA': 2465.0, 'AMPA RIVENDEL': 2200.0, 'AMPA AGORA': 2421.67, 'AMPA ALDEBARAN': 3107.05, 'AMPA GINE
DE LOS RIOS': 2058.0, 'AMPA SEVERO OCHOA': 3563.9700000000003, 'AMPA VIRGEN DE LA PAZ': 1416.45, 'AMPA JUAN XXIII': 1781.98,
'AMPA LA CHOPERA': 1430.0, 'AMPA PADRE MANYANET': 2695.82, 'AMPA FAPA': 3198.43}
```

**Vamos a calcular la subvención total  
percibida por centro.**

**ESCRIBIR EL CÓDIGO PYTHON NECESARIO PARA CREAR UN NUEVO LIBRO CON DOS HOJAS. EN LA PRIMERA COPIAREMOS EXACTAMENTE EL CONTENIDO DE LA HOJA ACTUAL, LA SEGUNDA TENDRÁ UNA TABLA CON CUATRO COLUMNAS: ASOCIACIÓN, IMPORTE TOTAL, TOTAL JUSTIFICADO, IMPORTE PENDIENTE DE JUSTIFICAR**

# Lectura de Ficheros

EXCEL



The screenshot shows a Jupyter Notebook interface with a browser window at the top displaying the URL `localhost:8888/notebooks/Python-Ejercicios/scr/Cap%201/lectura.ipynb`. The notebook has a menu bar (File, Edit, View, Insert, Cell, Kernel, Widgets, Help) and a toolbar with icons for file operations and execution. The main area contains two code cells. The first cell, labeled 'In [17]:', contains a Python script that reads an Excel file, processes its data, and writes the results to a new Excel file. The second cell, labeled 'In [18]:', contains a single line of code to print column names.

```
In [17]: with open_workbook('../data/Cap 1/subvenciones.xls', on_demand=True) as libro_lect:
asociaciones = {}
libro_escr = xlwt.Workbook()
for nombre in libro_lect.sheet_names():
    hoja_lect = libro_lect.sheet_by_name(nombre)
    hoja_escr = libro_escr.add_sheet(nombre)
    for i in range(hoja_lect.nrows):
        for j in range(hoja_lect.ncols):
            hoja_escr.write(i, j, hoja_lect.row(i)[j].value)
        if i != 0:
            fila = hoja_lect.row(i)
            centro = fila[0].value
            subvencion = float(fila[2].value)
            if centro in asociaciones:
                asociaciones[centro] = asociaciones[centro] + subvencion
            else:
                asociaciones[centro] = subvencion
hoja_escr = libro_escr.add_sheet('Totales')
hoja_escr.write(0, 0, "Asociación")
hoja_escr.write(0, 1, "Importe total")
hoja_escr.write(0, 2, "Importe justificado")
hoja_escr.write(0, 3, "Restante")
for i, clave in enumerate(asociaciones):
    fila = i + 1
    fila_form = i + 2
    hoja_escr.write(fila, 0, clave)
    hoja_escr.write(fila, 1, asociaciones[clave])
    hoja_escr.write(fila, 2, 0)
    hoja_escr.write(fila, 3, xlwt.Formula("C" + str(fila_form) + "-" + "B" + str(fila_form)))
libro_escr.save('../data/Cap1/subvenciones_totales.xls')

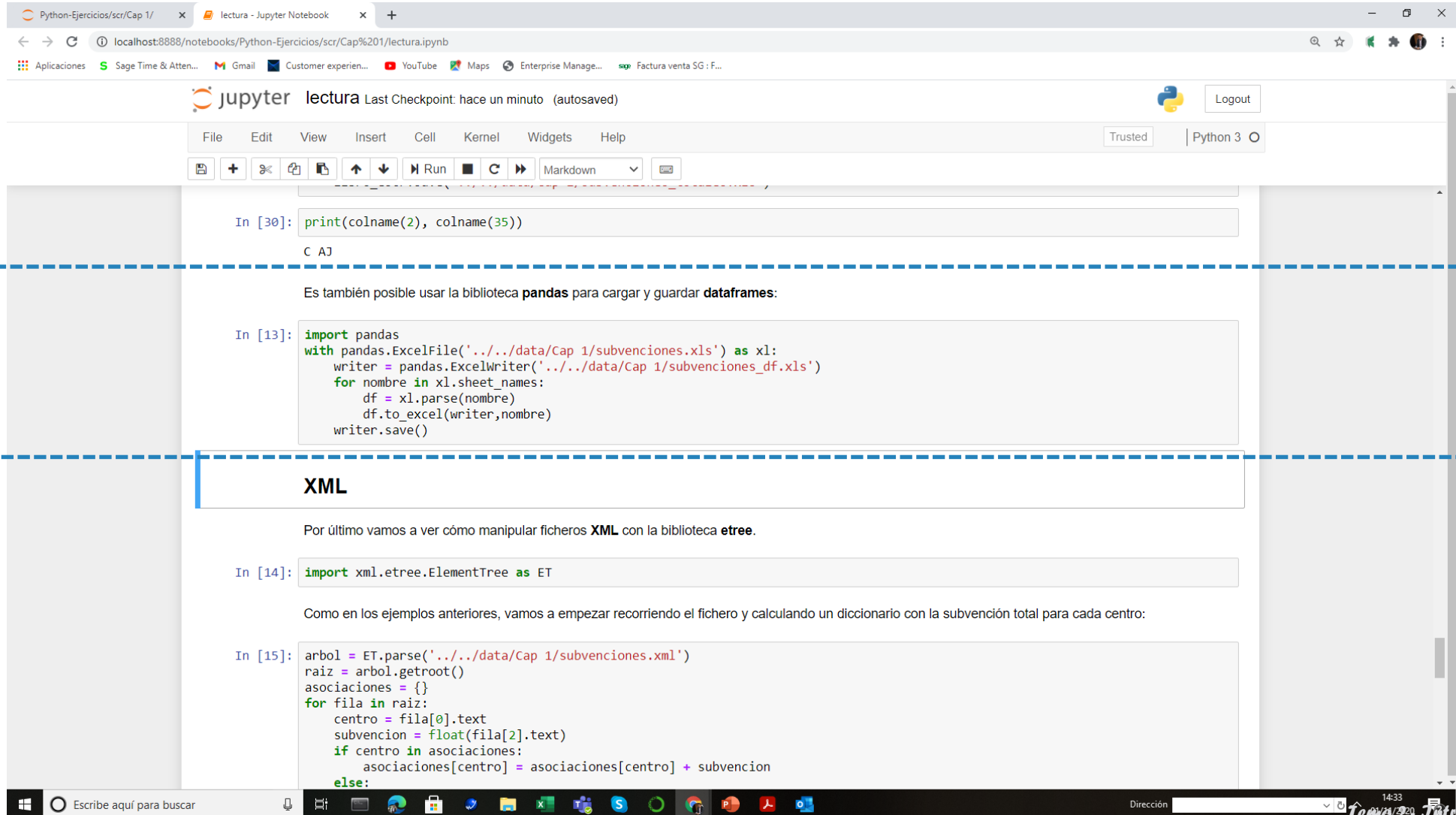
In [18]: print(colname(2), colname(35))
```

**ES POSIBLE UTILIZAR PANDAS PARA  
CARGAR Y ALMACENAR DATAFRAMES, LO  
VEMOS CON UN EJEMPLO**



# Lectura de Ficheros

EXCEL



The screenshot shows a Jupyter Notebook interface with the following content:

**Cell [30]:**

```
print(colname(2), colname(35))
```

C AJ

Es también posible usar la biblioteca **pandas** para cargar y guardar **dataframes**:

**Cell [13]:**

```
import pandas
with pandas.ExcelFile('../data/Cap 1/subvenciones.xls') as xl:
    writer = pandas.ExcelWriter('../data/Cap 1/subvenciones_df.xls')
    for nombre in xl.sheet_names:
        df = xl.parse(nombre)
        df.to_excel(writer, nombre)
    writer.save()
```

**XML**

Por último vamos a ver cómo manipular ficheros **XML** con la biblioteca **etree**.

**Cell [14]:**

```
import xml.etree.ElementTree as ET
```

Como en los ejemplos anteriores, vamos a empezar recorriendo el fichero y calculando un diccionario con la subvención total para cada centro:

**Cell [15]:**

```
arbol = ET.parse('../data/Cap 1/subvenciones.xml')
raiz = arbol.getroot()
asociaciones = {}
for fila in raiz:
    centro = fila[0].text
    subvencion = float(fila[2].text)
    if centro in asociaciones:
        asociaciones[centro] = asociaciones[centro] + subvencion
    else:
```

Para manipular datos con XML se utilizará la biblioteca **etree**.

# Lectura de Ficheros

## XML

The screenshot shows a Jupyter Notebook titled 'lectura' with a 'Last Checkpoint: hace 10 minutos (autosaved)' status. The interface includes a menu bar (File, Edit, View, Insert, Cell, Kernel, Widgets, Help) and a toolbar with icons for file operations and execution. The notebook content is as follows:

**XML**

Por último vamos a ver cómo manipular ficheros **XML** con la biblioteca **etree**.

In [21]: `import xml.etree.ElementTree as ET`

Como en los ejemplos anteriores, vamos a empezar recorriendo el fichero y calculando un diccionario con la subvención total para cada centro:

In [22]:

```

arbol = ET.parse('../data/Cap 1/subvenciones.xml')
raiz = arbol.getroot()
asociaciones = {}
for fila in raiz:
    centro = fila[0].text
    subvencion = float(fila[2].text)
    if centro in asociaciones:
        asociaciones[centro] = asociaciones[centro] + subvencion
    else:
        asociaciones[centro] = subvencion
print(asociaciones)

```

The output of the code is a dictionary showing the total subsidy for each center:

```

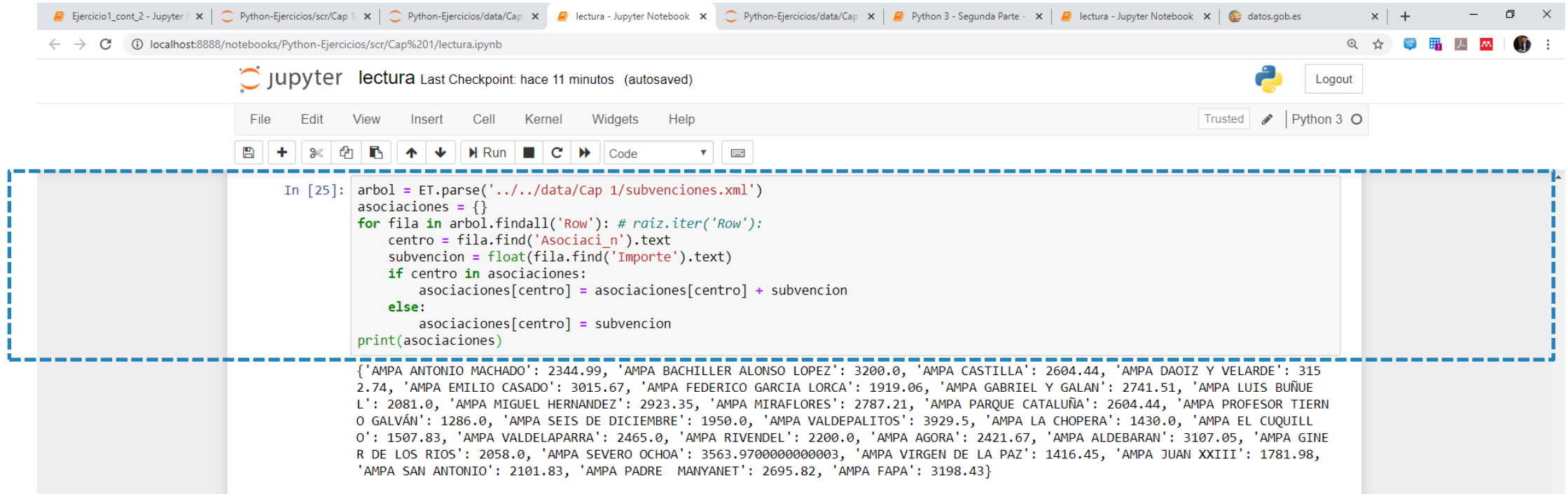
{'AMPA ANTONIO MACHADO': 2344.99, 'AMPA BACHILLER ALONSO LOPEZ': 3200.0, 'AMPA CASTILLA': 2604.44, 'AMPA DAOIZ Y VELARDE': 3152.74, 'AMPA EMILIO CASADO': 3015.67, 'AMPA FEDERICO GARCIA LORCA': 1919.06, 'AMPA GABRIEL Y GALAN': 2741.51, 'AMPA LUIS BUÑUE L': 2081.0, 'AMPA MIGUEL HERNANDEZ': 2923.35, 'AMPA MIRAFLORES': 2787.21, 'AMPA PARQUE CATALUÑA': 2604.44, 'AMPA PROFESOR TIERN O GALVÁN': 1286.0, 'AMPA SEIS DE DICIEMBRE': 1950.0, 'AMPA VALDEPALITOS': 3929.5, 'AMPA LA CHOPERA': 1430.0, 'AMPA EL CUQUILL O': 1707.83, 'AMPA VZ DELAPARRA': 2165.0, 'AMPA RIVENDEL': 2200.0, 'AMPA AGORA': 2421.67, 'AMPA ALDEBARAN': 3107.05, 'AMPA GINE R': 1707.83, 'AMPA SAN ANTONIO': 2101.83, 'AMPA PADRE MANYANET': 2695.82, 'AMPA FAPA': 3198.43}

```

**Vamos a calcular la subvención total  
percibida por centro...**

# Lectura de Ficheros

XML



The screenshot shows a Jupyter Notebook interface with a browser window at the top displaying the URL `localhost:8888/notebooks/Python-Ejercicios/scr/Cap%201/lectura.ipynb`. The notebook has a menu bar (File, Edit, View, Insert, Cell, Kernel, Widgets, Help) and a toolbar with icons for file operations and execution. The code cell is titled "In [25]:" and contains the following Python code:

```
arbol = ET.parse('../data/Cap 1/subvenciones.xml')
asociaciones = {}
for fila in arbol.findall('Row'): # raiz.iter('Row'):
    centro = fila.find('Asociaci_n').text
    subvencion = float(fila.find('Importe').text)
    if centro in asociaciones:
        asociaciones[centro] = asociaciones[centro] + subvencion
    else:
        asociaciones[centro] = subvencion
print(asociaciones)
```

The output of the code is a dictionary showing the total subvention for each center:

```
{'AMPA ANTONIO MACHADO': 2344.99, 'AMPA BACHILLER ALONSO LOPEZ': 3200.0, 'AMPA CASTILLA': 2604.44, 'AMPA DAOIZ Y VELARDE': 3152.74, 'AMPA EMILIO CASADO': 3015.67, 'AMPA FEDERICO GARCIA LORCA': 1919.06, 'AMPA GABRIEL Y GALAN': 2741.51, 'AMPA LUIS BUÑUEL': 2081.0, 'AMPA MIGUEL HERNANDEZ': 2923.35, 'AMPA MIRAFLORES': 2787.21, 'AMPA PARQUE CATALUÑA': 2604.44, 'AMPA PROFESOR TIERN O GALVÁN': 1286.0, 'AMPA SEIS DE DICIEMBRE': 1950.0, 'AMPA VALDEPALITOS': 3929.5, 'AMPA LA CHOPERA': 1430.0, 'AMPA EL CUQUILL O': 1507.83, 'AMPA VALDELAPARRA': 2465.0, 'AMPA RIVENDEL': 2200.0, 'AMPA AGORA': 2421.67, 'AMPA ALDEBARAN': 3107.05, 'AMPA GINE R DE LOS RIOS': 2058.0, 'AMPA SEVERO OCHOA': 3563.9700000000003, 'AMPA VIRGEN DE LA PAZ': 1416.45, 'AMPA JUAN XXIII': 1781.98, 'AMPA SAN ANTONIO': 2101.83, 'AMPA PADRE MANYANET': 2695.82, 'AMPA FAPA': 3198.43}
```

Vamos a calcular la subvención total  
percibida por centro...

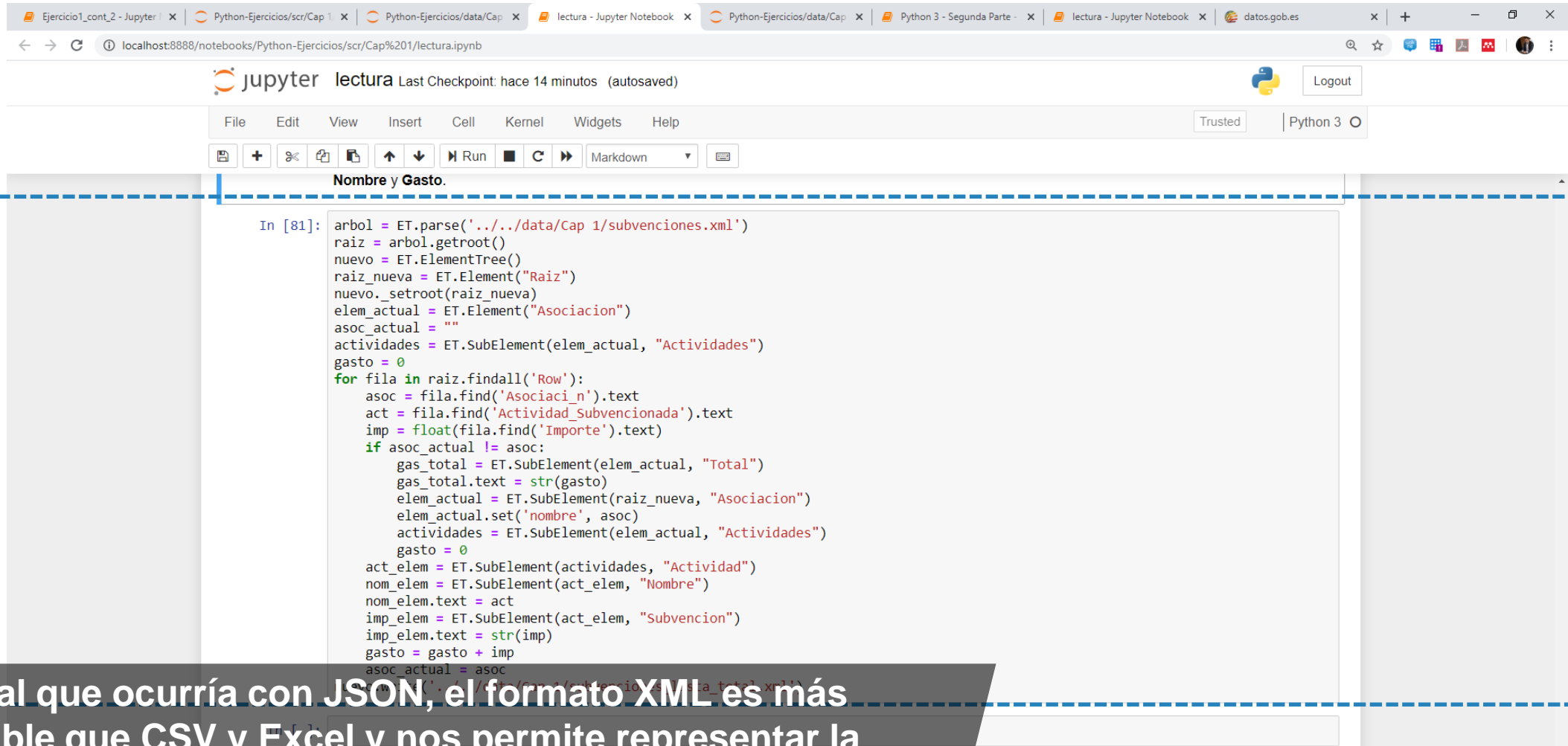
# Ejercicio

XML

**ESCRIBIR EL CÓDIGO PYTHON NECESARIO PARA CREAR UN FICHERO XML A PARTIR DEL QUE TENEMOS, QUE CONTARÁ CON UNA RAÍZ QUE TENDRÁ COMO ELEMENTOS LAS DISTINTAS ASOCIACIONES. CADA ASOCIACIÓN TENDRÁ COMO ATRIBUTO SU NOMBRE Y COMO ELEMENTOS LA SUBVENCIÓN TOTAL Y LA LISTA DE ACTIVIDADES. LA LISTA DE ACTIVIDADES TENDRÁ ELEMENTOS ACTIVIDAD CON NOMBRE Y GASTOS**

# Lectura de Ficheros

## JSON



```
In [81]: arbol = ET.parse('../data/Cap 1/subvenciones.xml')
raiz = arbol.getroot()
nuevo = ET.ElementTree()
raiz_nueva = ET.Element("Raiz")
nuevo._setroot(raiz_nueva)
elem_actual = ET.Element("Asociacion")
asoc_actual = ""
actividades = ET.SubElement(elem_actual, "Actividades")
gasto = 0
for fila in raiz.findall('Row'):
    asoc = fila.find('Asociacion').text
    act = fila.find('Actividad_Subvencionada').text
    imp = float(fila.find('Importe').text)
    if asoc_actual != asoc:
        gas_total = ET.SubElement(elem_actual, "Total")
        gas_total.text = str(gasto)
        elem_actual = ET.SubElement(raiz_nueva, "Asociacion")
        elem_actual.set('nombre', asoc)
        actividades = ET.SubElement(elem_actual, "Actividades")
        gasto = 0
    act_elem = ET.SubElement(actividades, "Actividad")
    nom_elem = ET.SubElement(act_elem, "Nombre")
    nom_elem.text = act
    imp_elem = ET.SubElement(act_elem, "Subvencion")
    imp_elem.text = str(imp)
    gasto = gasto + imp
    asoc_actual = asoc
```

Igual que ocurría con JSON, el formato XML es más flexible que CSV y Excel y nos permite representar la información de manera más compacta. .

# **EJERCICIO**

# Ejercicio

---

**OBTENER LA SERIE HISTÓRICA DE COVID 19  
EN FORMATO CSV, INTENTAR OBTENER EL  
INCREMENTO DIARIO DE CASOS  
DIAGNOSTICADOS, HOSPITALIZACIONES,  
UCI, FALLECIMIENTOS. LLEVAR EL  
RESULTADO A UN FORMATO EXCEL.**



# !Muchas Gracias!

---

GABRIEL MARÍN DÍAZ  
LCDO. CIENCIAS FÍSICAS UCM

[www.linkedin.com/in/gabrielmarindiaz/](http://www.linkedin.com/in/gabrielmarindiaz/)