

Unidad 7

Persistencia de Datos

Manual del Estudiante



Maria del Carmen Sierra Fernández

Contenido

Nivel II: Representación, proceso y visualización de Datos.....	3
Unidad 7: Persistencia de Datos	3
Ficheros en formato texto	3
Abrir un archivo de texto en modo lectura	6
Utilizando read()	6
Utilizando readline()	7
Utilizando readlines().....	8
Declaración with	9
Recomendaciones para leer y procesar un archivo.....	9
Escribir ficheros de texto	10
Escribir un fichero línea a línea.....	10
Escribir un fichero de una vez.....	11
Añadir información a un fichero existente	11
Ficheros en formato CSV.....	12
Función csv.reader:.....	12
Función csv.writer:.....	14
Clases DictReader y DictWriter	15
DictReader.....	15
DictWriter.....	16
Dialectos y Formato	17
Pickle y Shelve.....	17
Formato y Modulo JSON	19
Sangría o indentación	21
Codificación Unicode	21
Ordenación.....	21
Obtener datos en formato JSON desde un servicio web.....	21
Almacenar y Leer los datos en formato JSON en un fichero	22
Base de Datos Documentales	22
¿Para qué se usa MongoDB?	23
Modelo de Datos.....	23
Conceptos	23
Tipos de Datos.....	25

Consulta y modificación de datos: operaciones CRUD (Create, Read, Update, Delete):.....	25
Insertar documentos.....	25
Borrar documentos.....	26
Actualización de documentos.....	26
Consultar Documentos	26
Índices	28
Crear una base de Datos Documental en Python.....	29
Consideraciones para el diseño de la BBDD blognoticias.....	29
Modelo de la Base de Datos blognoticias.....	29
PyMongo	30
Implementación de nuestra Base de Datos blognoticias	30
Bibliografía	38



Nivel II: Representación, proceso y visualización de Datos

Unidad 7: Persistencia de Datos

En programación, la persistencia es la acción de preservar la información de un objeto de forma permanente (almacenar o guardar), pero a su vez también se refiere a poder recuperar la información de este (leerlo) para que pueda ser nuevamente utilizado.¹

Los objetos como números, listas, diccionarios, estructuras anidadas y objetos de instancia de clase viven en la memoria de nuestro ordenador y se pierden tan pronto como termina el script. Es decir, que los datos tienen una duración efímera; desde el momento en que estos cambian de valor se considera que no hay persistencia de estos.

En la vida real, los datos residen en ficheros o archivos. Por ejemplo: imágenes, páginas web, documentos de procesamiento de texto, música, entre otros.

En esta unidad conoceremos la forma de almacenar o guardar nuestros datos, así como la forma de leerlos. Realizaremos operaciones con ficheros o archivos en diferentes formatos: texto, binario, CSV y JSON. Además, conoceremos la base de datos documental MongoDB, realizaremos algunas operaciones básicas como: creación de colecciones, inserción de datos, consultas y creación de índices.

Ficheros en formato texto

Antes de leer o escribir archivos con Python es necesario abrir una conexión. Lo que se puede hacer con la función Built-in `open()`. Esta función nos permite abrir el fichero o archivo especificado y devuelve el objeto de archivo correspondiente. Si el archivo no se puede abrir, se genera un error `OSError`.

Sintaxis:

```
open(file, mode='r', buffering=-1, encoding=None, errors=None,
      newline=None, closefd=True, opener=None)
```

file: es un objeto similar a una ruta que proporciona el nombre de ruta (absoluto o relativo al directorio de trabajo actual) del archivo que se abrirá o un descriptor de archivo que es un número entero. (Si se proporciona un descriptor de archivo, se cierra cuando se cierra el objeto de E/S devuelto, a menos que `closefd` se establezca en `False`).

mode es una cadena opcional que especifica el modo en el que se abre el archivo. Su valor predeterminado es `'r'`, que significa abierto para lectura en modo texto. Otros

¹ Wikipedia persistencia (informática)

Nivel II: Representación, proceso y visualización de Datos

valores comunes son 'w' para escribir (truncar el archivo si ya existe), 'x' para creación exclusiva y 'a' para agregar (que en algunos sistemas Unix, significa que todas las escrituras se agregan al final del archivo independientemente de la posición de búsqueda actual). En el modo de texto, si no se especifica la codificación, la codificación utilizada depende de la plataforma: `locale.getpreferredencoding(False)` se llama para obtener la codificación local actual. (Para leer y escribir bytes sin procesar, use el modo binario y deje la codificación sin especificar).

Los modos disponibles son:

Carácter	Descripción
'r'	abrir para lectura (predeterminado)
'w'	abrir para escribir, truncando el archivo primero
'x'	abrir para creación exclusiva, fallando si el archivo ya existe
'a'	abrir para escritura, agregando al final del archivo si existe
'b'	modo binario
't'	modo texto (predeterminado)
'+'	abrir un archivo de disco para actualizarlo (leer y escribir)
'U'	modo universal de nuevas líneas (obsoleto)

`buffering` es un entero opcional que se utiliza para establecer la política de almacenamiento en búfer. Pase 0 para desactivar el almacenamiento en búfer (solo se permite en modo binario), 1 para seleccionar el almacenamiento en búfer de línea (solo se puede usar en modo texto) y un entero > 1 para indicar el tamaño en bytes de un búfer de fragmentos de tamaño fijo.

`encoding` es el nombre de la codificación utilizada para decodificar o codificar el archivo. Esto solo debe usarse en modo texto. La codificación predeterminada depende de la plataforma (lo que devuelve `locale.getpreferredencoding()`), pero se puede utilizar cualquier codificación de texto compatible con Python. Consulte el módulo de códecs para obtener la lista de codificaciones compatibles.

`errors` es una cadena opcional que especifica cómo se deben manejar los errores de codificación y decodificación; esto no se puede usar en modo binario. Hay disponibles una variedad de controladores de errores estándar (enumerados en Controladores de errores), aunque cualquier nombre de manejo de errores que se haya registrado con `codecs.register_error()` también es válido. Los nombres estándar incluyen:

- `'strict'` para generar una excepción `ValueError` si hay un error de codificación. El valor predeterminado de Ninguno tiene el mismo efecto.
- `'ignore'` ignora los errores. Tenga en cuenta que ignorar los errores de codificación puede provocar la pérdida de datos.
- `'replace'` hace que se inserte un marcador de reemplazo (como '?') donde hay datos mal formados.

- `'surrogateescape'` representará los bytes incorrectos como puntos de código en el área de uso privado Unicode que va desde U + DC80 a U + DCFF. Estos puntos de código privado se volverán a convertir en los mismos bytes cuando se utilice el controlador de errores de escape sustituto al escribir datos. Esto es útil para procesar archivos con una codificación desconocida.
- `'xmlcharrefreplace'` solo se admite cuando se escribe en un archivo. Los caracteres no admitidos por la codificación se reemplazan con la referencia de carácter XML adecuada & # nnn ;.
- `'backslashreplace'` reemplaza los datos con formato incorrecto por las secuencias de escape con barra invertida de Python.
- `'namereplace'` (también solo se admite al escribir) reemplaza los caracteres no admitidos con secuencias de escape `\ N {...}`.

`newline` controla cómo funciona el modo de nuevas líneas universales (solo se aplica al modo de texto). Puede ser `None`, `' '`, `'\n'`, `'\r'` y `'\r\n'`. Funciona de la siguiente manera:

- Al leer la entrada de la secuencia, si la `newline` es `None`, el modo de nuevas líneas universal está habilitado. Las líneas en la entrada pueden terminar en `'\n'`, `'\r'` o `'\r\n'`, y estas se traducen a `'\n'` antes de ser devueltas a la persona que llama. Si es `' '`, el modo de nueva línea universal está habilitado, pero los finales de línea se devuelven al llamante sin traducir. Si tiene alguno de los otros valores legales, las líneas de entrada solo terminan con la cadena dada y el final de la línea se devuelve al llamador sin traducir.
- Al escribir la salida en la secuencia, si la `newline` es `None`, los caracteres `'\n'` escritos se traducen al separador de línea predeterminado del sistema, `os.linesep`. Si nueva línea es `' '` o `'\n'`, no se realiza ninguna traducción. Si `newline` es cualquiera de los otros valores legales, cualquier carácter `'\n'` escrito se traduce a la cadena dada.

Si `closefd` es `False` y se proporcionó un descriptor de archivo en lugar de un nombre de archivo, el descriptor de archivo subyacente se mantendrá abierto cuando se cierre el archivo. Si se proporciona un nombre de archivo, `closefd` debe ser `True` (el valor predeterminado); de lo contrario, se generará un error.

Se puede usar un `opener` personalizado pasando `opener` como parámetro. El descriptor de archivo subyacente para el objeto de archivo se obtiene llamando a `opener` con `(file, flag)`. `opener` debe devolver un descriptor de archivo abierto (pasar `os.open` como abridor da como resultado una funcionalidad similar a pasar `None`).

Nivel II: Representación, proceso y visualización de Datos

Una vez que hemos visto la sintaxis y conocido los diferentes parámetros de la de la función `open()`. Veamos las funciones y métodos que podemos utilizar para abrir y cerrar ficheros o archivos.

Método	Sintaxis	Descripción
write	<code>f.write(string)</code>	Añade un string al final del fichero. <code>f</code> se refiere al fichero abierto para escritura
read(n)	<code>f.read()</code>	Lee y retorna un string de <code>n</code> caracteres, o el fichero entero como un string, sin <code>n</code> no es definida.
readline(n)	<code>f.readline()</code>	Lee y retorna la siguiente línea de un fichero, incluido el carácter de newline. Si <code>n</code> es definida como parámetro, entonces sólo retorna <code>n</code> caracteres de la línea.
readlines(n)	<code>f.readlines()</code>	Retorna una lista de strings, cada uno representa una línea del fichero. Si <code>n</code> es definida como parámetro, entonces <code>n</code> caracteres son leídos y retornados como una línea.

Abrir un archivo de texto en modo lectura

Ahora vamos a trabajar con archivos o ficheros de texto, es decir, archivos llenos de caracteres. Podemos crear estos archivos de varias formas: utilizando un editor de texto para escribir y guardar los datos o podríamos descargar los datos de un sitio web y luego guardarlos en un archivo. Independientemente de cómo se cree el archivo, Python nos permitirá manipular el contenido.

Hemos visto la función `open()` nos permite abrir el fichero o archivo especificado y devuelve el objeto de archivo correspondiente. Por defecto la conexión se abre en modo lectura (`mode="r"`).

```
# Abre el archivo text.txt en modo lectura
f = open("coleccioncorpus.txt", "r")
f
<_io.TextIOWrapper name='text1.txt' mode='r' encoding='cp1252'>
```

Hemos abierto el fichero " coleccioncorpus.txt ", en modo lectura, observamos que `f` es un objeto `_io.TextIOWrapper`. Ahora nos preguntamos: ¿Cómo podemos ver los datos de nuestro fichero?

Utilizando `read()`

```
# Abrir y leer un fichero en modo lectura

f = open("coleccioncorpus.txt", "r", encoding='utf-8')
texto = f.read()

# número de caracteres del fichero
print("Longitud {} caracteres \n".format(len(texto)))
print(texto)
```

```
f.close()
```

```
Longitud: 34641 caracteres
```

```
NEWS STORY
Chinese military in 'fight to the death' with flooded Yangtze

&UR; Eds: ADDS warning about Yellow River flooding in grafs 5-
6,
CLARIFIES that embankments are on Yangtze in graf 7 &QL; ...
```

Observamos como `read()` nos retorna el contenido del fichero "coleccioncorpus.txt", utilizando `len()` obtenemos la cantidad de caracteres que tiene nuestro fichero.

```
# Abrir y leer un fichero en modo lectura
```

```
f = open("coleccioncorpus.txt", "r", encoding='utf-8')
texto = f.read(70)
```

```
# número de caracteres del fichero
print("Longitud {} caracteres \n".format(len(texto)))
print(texto)
```

```
f.close()
```

```
Longitud: 70 caracteres
```

```
NEWS STORY
Chinese military in 'fight to the death' with flooded Yan
```

Observamos que al pasarle a `read()` el parámetro `n=70`, nos retorna los primeros 70 caracteres de nuestro fichero.

Utilizando `readline()`

```
# Abrir y leer un fichero en modo lectura
```

```
f = open("coleccioncorpus.txt", "r", encoding='utf-8')
```

```
texto = f.readline()
print("Longitud: {} caracteres \n".format(len(texto))) # número
de caracteres
print(texto)
```

```
f.close()
```

```
Longitud: 13 caracteres
```

```
NEWS STORY
```


Nivel II: Representación, proceso y visualización de Datos

Observamos que al utilizar `readline()`, lee sólo la primera línea del fichero. Para obtener todas las líneas del fichero utilizemos un bucle `while`:

```
# Abrir y leer un fichero en modo lectura
```

```
f = open("coleccioncorpus.txt", "r", encoding='utf-8')
```

```
linea = f.readline() # Lee la primera línea
```

```
while linea:
```

```
    print(linea)
```

```
    linea = f.readline() # Lee la siguiente línea
```

```
f.close()
```

```
NEWS STORY
```

```
Chinese military in 'fight to the death' with flooded Yangtze
```

```
&UR; Eds: ADDS warning about Yellow River flooding in grafs 5-6,
```

```
CLARIFIES that embankments are on Yangtze in graf 7 &QL;...
```

Utilizando `readlines()`

```
# Abrir y leer un fichero en modo lectura
```

```
f = open("coleccioncorpus.txt", "r", encoding='utf-8')
```

```
texto = f.readlines() # lee todo el contenido del fichero
```

```
print(f"Total líneas del texto: {len(texto)} \n")
```

```
for linea in texto:
```

```
    print(linea, end="")
```

```
f.close()
```

```
Total líneas del texto: 760
```

```
NEWS STORY
```

```
Chinese military in 'fight to the death' with flooded Yangtze
```

```
&UR; Eds: ADDS warning about Yellow River flooding in grafs 5-6,
```

```
CLARIFIES that embankments are on Yangtze in graf 7 &QL;...
```

Observamos como `readlines()` retorna todo el contenido del fichero y lo asigna a la variable `texto`. Si utilizamos `len(texto)` obtenemos el total de líneas del fichero `"coleccioncorpus.txt"`.

Declaración with

Ahora que hemos visto como abrir y cerrar ficheros, existe otro mecanismo que Python nos proporciona que es la declaración `with`. Esta simplifica el manejo de excepciones al encapsular las tareas comunes de preparación y limpieza. Además, cerrará automáticamente el archivo.

Abrir y leer un fichero en modo lectura

```
with open("coleccioncorpus.txt", "r", encoding='utf-8') as f:
    texto = f.readlines() # lee todo el contenido

for linea in texto:
    print(linea, end="")
```

Recomendaciones para leer y procesar un archivo

A continuación, veremos algunas recomendaciones para procesar el contenido de un archivo de texto. Algunas las opciones son preferibles para algunas situaciones, y los programadores de Python prefieren algunas por razones de eficiencia.

1. Abrir el fichero o archivo usando `with` y `open`.
2. Utilice `.readlines()` para obtener una lista de las líneas de texto del archivo.
3. Utilice un bucle `for` para recorrer las cadenas de la lista, cada una de las cuales es una línea del archivo. En cada iteración, procesa esa línea de texto.
4. Cuando termine de extraer datos del archivo, continúe escribiendo su código fuera de la sangría. Al utilizar `with` se cerrará automáticamente el archivo una vez que el programa salga del bloque `with`.

```
fname = "yourfile.txt"

with open(fname, 'r') as fileref:
    texto = fileref.readlines()
    for linea in texto:
        # código con la variable línea
```

Otras sentencias no relacionadas con `fileref` # paso 4

Sin embargo, esta opción no es buena cuando trabajamos con volumen de datos, se necesitaría mucho tiempo para leer todos los datos. Este sería un caso en el que los programadores prefieren otra opción por razones de eficiencia.

Esta opción implica iterar sobre el archivo en sí mientras sigue iterando sobre cada línea en el archivo:

```
fname = "yourfile.txt"

with open(fname, 'r') as fileref:
```

paso 1

```
for linea in fileref:                                # paso 2
    # código con la variable línea

# Otras sentencias no relacionadas con fileref      # paso 3
```

Escribir ficheros de texto

Una de las tareas de procesamiento de datos que se realizan con más frecuencia es leer datos de un archivo, manipularlos de alguna manera y luego escribir los datos resultantes en un nuevo archivo de datos para usarlos con otros fines más adelante. Para lograr esto, la función de apertura descrita anteriormente también se puede utilizar para crear un nuevo archivo preparado para escribir.

Para poder escribir es necesario utilizar `mode = "w"` con la que se eliminará cualquier archivo existente y creará uno nuevo. Otra opción que se puede utilizar es `"a"`, con la que se añadirá nuevo contenido al archivo existente.

```
# Abrir un archivo modo escritura
f = open("text.txt", "w")
f = open("text.txt", mode="w")

# Abrir un archivo para agregar contenido
f = open("text.txt", "a")
f = open("text.txt", mode="a")

# Abrir un archivo modo escritura con with
fichero = "text.txt"
with open(fichero, "w") as f:
```

Escribir un fichero línea a línea

Un método para volcar información en un archivo es escribir el contenido línea a línea. Para ello se puede iterar sobre el archivo y utilizar el método `write` de archivo. Veamos algunos ejemplos:

```
data = ["Línea 1", "Línea 2", "Línea 3", "Línea 4", "Línea 5"]

f = open(mi_fichero1.txt, "w")

for line in data:
    f.write(line)
    f.write("\n")

f.close()
```

Observemos que los elementos de la lista no finalizan con el carácter salto de línea. Por lo tanto, es necesario añadirlo después de escribir cada línea. De lo contrario, todos los elementos se escribirían en una única línea en el archivo de salida.

Nivel II: Representación, proceso y visualización de Datos

Una forma de escribir el archivo línea a línea sin que sea necesario incluir el salto de línea es con la función `print()`. Para ello, es necesario incluir la opción `file` con la conexión al archivo.

```
f = open(mi_fichero_2.txt, "w")

for line in data:
    print(line, file=f)

f.close()
```

Escribir un fichero de una vez

Otro método, en el caso de que los datos se encuentren en un objeto iterable se puede utilizar el método `writelines` para volcar en una sola vez. Aunque es necesario tener en cuenta que este método no agrega el salto de línea, por lo que puede ser necesario agregarlo con antelación.

```
f = open("mi_fichero_3.txt", "w")

f.writelines("%s\n" %s for s in data)

f.close()
```

En el ejemplo se puede apreciar que se ha iterado sobre la lista para agregar el salto de línea para cada elemento.

Añadir información a un fichero existente

El método de escritura que nos permite agregar datos a un archivo de texto, es el modo `"a"`. Cuando se invoca, los caracteres de la cadena se agregarán al final del archivo. Veamos un ejemplo, donde añadiremos cinco nuevas líneas al fichero `"mi_fichero_3.txt"`:

```
data = ["Línea 6", "Línea 7", "Línea 8", "Línea 9", "Línea 10"]

f = open("mi_fichero_3.txt", "a")

f.writelines("%s\n" %s for s in data)

f.close()
```

El siguiente ejemplo muestra dos funciones que realizan dos procesos importantes al momento de trabajar con ficheros, la función `crear_fichero()`: obtiene los datos de una lista los formatea y crea un fichero, mientras que la función `crear_lista()`: toma los datos desde un fichero, los formatea para crear una lista.

```
jugadores = [('Thibaut Courtois', "Real Madrid", "Portero"),
              ('Sergio Ramos', "Real Madrid", "Defensa"),
              ('Marcelo Vieira', "Real Madrid", "Defensa")]
```

```
fname = "RealMadrid.txt"

def crear_fichero(f, lista):
    with open(f, "w") as fichero:
        for nombre, equipo, posicion in lista:
            fichero.write('{0},{1},{2}\n'.format(nombre,
                                                equipo,
                                                posicion))

def crear_lista(f):
    lista = []

    with open(f, "r") as fichero:
        for linea in fichero:
            nombre, equipo, posicion =
linea.rstrip("\n").split(",")
            lista.append((nombre, equipo, posicion))
    return lista
```

```
crear_fichero(fname, jugadores)
crear_lista(fname)
```

Ficheros en formato CSV

El formato CSV (Comma Separated Values) es el formato de importación y exportación utilizado con más frecuencia para bases de datos y hojas de cálculo, porque son fáciles de utilizar. Un fichero CSV (valores separados por comas) permite que los datos sean guardados en una estructura tabular con una extensión `.csv`. Cada fila leída del archivo `csv` se devuelve como una lista de cadenas.

El módulo CSV tiene varias funciones y clases disponibles para leer y escribir CSVs, y estas incluyen:

- función `csv.reader`
- función `csv.writer`
- clase `csv.Dictwriter`
- clase `csv.DictReader`

Función `csv.reader`:

El módulo `csv.reader` toma los siguientes parámetros:

- `csvfile`: Este es usualmente un objeto el cuál soporta el protocolo iterador y usualmente devuelve una cadena cada vez que su método `__next__()` es llamado.
- `dialect='excel'`: Un parámetro opcional usado para definir un conjunto de parámetros específicos a un dialecto CSV en particular.

Nivel II: Representación, proceso y visualización de Datos

- `fmtparams`: Un parámetro opcional que puede ser usado para sobrescribir parámetros de formato existentes.

Aquí está un ejemplo de cómo usar el módulo `csv.reader`.

```
import csv

with open("airbnb.csv", newline='') as File:
    reader = csv.reader(File)
    for row in reader:
        print(row)

['neighbourhood_group', 'neighbourhood', 'latitude',
'longitude', 'room_type', 'price', 'minimum_nights',
'number_of_reviews', 'reviews_per_month',
'calculated_host_listings_count', 'availability_365']
['Centro', 'Justicia', '40.4247153175374', '-3.69863818770584',
'Entire home/apt', '49', '28', '35', '0.42', '1', '99']
['Centro', 'Embajadores', '40.4134181798486', '-
3.70683844591936', 'Entire home/apt', '80', '5', '18', '0.3',
'1', '188']
['Moncloa - Aravaca', 'Argüelles', '40.4249202354475', '-
3.71344613171984', 'Entire home/apt', '40', '2', '21', '0.25',
'9', '195']
```

Si no se especifica `newline=''`, las nuevas líneas incrustadas dentro de los campos entre comillas no se interpretarán correctamente, y en las plataformas que usan `\r\n` al escribir, se agregará una `\r` adicional. Siempre debería ser seguro especificar `newline=''`, ya que el módulo `csv` realiza su propio manejo (universal) de nueva línea.

```
import csv

with open(PATH_FICHERO, newline='', encoding='utf-8') as
csvfile:
    reader = csv.reader(csvfile, delimiter=',', quotechar='"')
    for row in reader:
        print(', '.join(row))
```

```
neighbourhood_group, neighbourhood, latitude, longitude,
room_type, price, minimum_nights, number_of_reviews,
reviews_per_month, calculated_host_listings_count,
availability_365
Centro, Justicia, 40.4247153175374, -3.69863818770584, Entire
home/apt, 49, 28, 35, 0.42, 1, 99
Centro, Embajadores, 40.4134181798486, -3.70683844591936,
Entire home/apt, 80, 5, 18, 0.3, 1, 188
Moncloa - Aravaca, Argüelles, 40.4249202354475, -
3.71344613171984, Entire home/apt, 40, 2, 21, 0.25, 9, 195
```

Nivel II: Representación, proceso y visualización de Datos

En este ejemplo, observamos que al abrir el fichero .csv hemos utilizado el parámetro `encoding='utf-8'`, vemos que los caracteres especiales son mostrados de forma correcta: Por ejemplo: 'Arg^Ãelles' -> Argüelles

Mientras `quotechar`, se refiere a la cadena de un solo carácter que se utilizará para citar valores si aparecen caracteres especiales (como delimitadores) dentro del campo. Su valor predeterminado es `'`.

Función `csv.writer`:

Este módulo es utilizado para escribir datos a un CSV. Tiene tres parámetros:

1. `csvfile`: Este puede ser cualquier objeto con un método `write()`.
2. `dialect='excel'`: Un parámetro opcional usado para definir un conjunto de parámetros específicos a un CSV en particular.
3. `fmtparam`: Un parámetro opcional que puede ser usado para sobrescribir parámetros de formato existentes.

La instancia `writer` proporciona dos métodos para escribir datos:

- `writerow(fila)`: Escribe una sola fila de datos y devuelve el número de caracteres escritos. La fila debe ser una secuencia de cadenas y números.
- `writerows(filas)`: Escribe varias filas de datos y devuelve `None`. Las filas deben ser una secuencia.

Veamos algunos ejemplos:

Utilizando `csv_writer.writerow()`

```
import csv

encabezado = ['nombre', 'equipo', 'posicion']
rows = [
    ["Thibaut Courtois", "Real Madrid", "Portero"],
    ["Sergio Ramos", "Real Madrid", "Defensa"],
    ["Marcelo Vieira", "Real Madrid", "Defensa"]
]

with open("jugadores.csv", 'w') as f:
    csv_writer = csv.writer(f, delimiter=',')

    csv_writer.writerow(encabezado) # write header

    for row in rows:
        csv_writer.writerow(row)
```

Utilizando `csv_writer.writerows()`

```
import csv

PATH_FICHERO = PATH_DATA + r"\" + "jugadores_3.csv"

encabezado = ['nombre', 'equipo', 'posicion']
rows = [
    ["Thibaut Courtois", "Real Madrid", "Portero"],
    ["Sergio Ramos", "Real Madrid", "Defensa"],
    ["Marcelo Vieira", "Real Madrid", "Defensa"]
]

with open(PATH_FICHERO, 'wt') as f:
    csv_writer = csv.writer(f)
    csv_writer.writerow(encabezado) # write header
    csv_writer.writerows(rows)
```

Clases DictReader y DictWriter

DictReader y DictWriter son clases disponibles en Python para leer y escribir a un CSV. Aunque son similares a las funciones reader y writer, estas clases usan objetos de diccionario para leer y escribir a archivos csv.

DictReader

Esta crea un objeto el cuál mapea la información leída a un diccionario cuyas llaves están dadas por el parámetro fieldnames. Este parámetro es opcional, pero cuando no se especifica en el archivo, la primera fila de datos se vuelve las llaves del diccionario.

```
import csv
with open(PATH_FICHERO) as csvfile:
    reader = csv.DictReader(csvfile)
    for row in reader:
        print(row)
```

```
{'neighbourhood_group': 'Centro', 'neighbourhood':
'Embajadores', 'latitude': '40.4134181798486', 'longitude': '-
3.70683844591936', 'room_type': 'Entire home/apt', 'price':
'80', 'minimum_nights': '5', 'number_of_reviews': '18',
'reviews_per_month': '0.3', 'calculated_host_listings_count':
'1', 'availability_365': '188'}
```

El siguiente ejemplo, mostramos las columnas (fieldnames) de nuestro fichero .csv;

```
import csv
with open(PATH_FICHERO) as csvfile:
    reader = csv.DictReader(csvfile)
    for row in reader:
        print(row['latitude'], row['longitude'],
              row['room_type'], row['price'])
```



```
40.4247153175374 -3.69863818770584 Entire home/apt 49
40.4134181798486 -3.70683844591936 Entire home/apt 80
40.4249202354475 -3.71344613171984 Entire home/apt 40
40.4310266945163 -3.72458616804393 Entire home/apt 55
```

DictWriter

Esta clase escribe los datos a un archivo CSV. La clase es definida como ;

```
csv.DictWriter(csvfile, fieldnames, restval='', extrasaction='raise',
dialect='excel', *args, **kwargs)
```

El parámetro `fieldnames` define la secuencia de claves que identifican el orden en el cual los valores en el diccionario son escritos al archivo CSV. `fieldnames` no es opcional y debe ser definida para evitar errores cuando se escribe a un CSV.

Veamos algunos ejemplos:

Utilizando `writer.writerow(row)`

```
import csv

encabezado = ["nombre", "equipo", "posicion"]

rows = [
    {"nombre": "Thibaut Courtois",
     "equipo": "Real Madrid", "posicion": "Portero"},
    {"nombre": "Sergio Ramos", "equipo": "Real Madrid",
     "posicion": "Defensa"},
    {"nombre": "Marcelo Vieira", "equipo": "Real Madrid",
     "posicion": "Defensa"}
]

with open("jugadores.csv", 'w') as csvfile:
    fieldnames = ["nombre", "equipo", "posicion"]
    writer = csv.DictWriter(csvfile, fieldnames=encabezado)

    writer.writeheader()
    for row in rows:
        writer.writerow(row)
```

Utilizando `writer.writerows(row)`

```
import csv
with open("jugadores.csv", 'wt') as csvfile:
    fieldnames = ["nombre", "equipo", "posicion"]
    writer = csv.DictWriter(csvfile, fieldnames=encabezado)
    writer.writeheader()
    writer.writerows(rows)
```

Dialectos y Formato

Un dialecto es una clase ayudante usada para definir los parámetros para una instancia reader o writer específica. Los parámetros de dialecto y formato necesitan ser declarados cuando se realiza una función lectora o writer.

Hay varios atributos los cuáles están soportados por un dialecto:

- **delimiter**: Una cadena usada para separar campos. Por defecto es ','.
- **double quote**: Controla cómo las instancias de quotechar que aparecen dentro de un campo deberían ser citadas. Puede ser Verdadero o Falso.
- **escapechar**: Una cadena usada por el escritor para escapar el delimitador si quoting está establecido a QUOTE_NONE.
- **lineterminator**: Una cadena usada para terminar líneas producidas por el writer. Por defecto es '\r\n'.
- **quotechar**: Una cadena usada para citar campos conteniendo caracteres especiales. Por defecto es '"'.
- **skipinitialspace**: Si se establece a True, cualquier espacio en blanco después del delimitador es ignorado inmediatamente.
- **strict**: Si se establece a True, levanta una excepción Error en mala entrada CSV.
- **quoting**: Controla cuando deberían delimitados los datos cuando se lee o escribe a un CSV.
 - Si **quoting** se establece en `csv.QUOTE_MINIMAL`, encerrara entre comillas los campos solo si contienen el delimitador o el carácter de '"'. Este es el caso predeterminado.
 - Si **quoting** se establece en `csv.QUOTE_ALL`, encerrara entre comillas todos los campos.
 - Si **quoting** se establece en `csv.QUOTE_NONNUMERIC`, encerrara entre comillas todos los campos que contienen datos de texto y convertirá todos los campos numéricos al tipo de datos flotantes.
 - Si **quoting** se establece en `csv.QUOTE_NONE`, escapará de los delimitadores en lugar de delimitarlos.

Pickle y Shelve

A la hora de realizar trabajos en los que sea necesario almacenar la información y recuperarla para su posterior uso, es muy útil utilizar algunas de las siguientes librerías: Pickle y Shelve.

Cuando queremos realizar esta tarea hablamos de persistencia. Como habíamos comentado, la persistencia es la acción de conservar la información un objeto de forma permanente, pero también de recuperarla. Para esto existe algo conocido como serialización de objetos. La serialización de un objeto consiste en generar una secuencia de bytes para su almacenamiento. Después mediante la deserialización, el estado original del objeto se puede reconstruir.

Nivel II: Representación, proceso y visualización de Datos

Esto es posible hacerlo en Python con las librerías Pickle y CPyckle. La diferencia entre estas dos librerías es sencillamente que la segunda está escrita en C.

```
import pickle

rows = [{"nombre": "Thibaut Courtois",
        "equipo": "Real Madrid", "posicion": "Portero"},
        {"nombre": "Sergio Ramos", "equipo": "Real Madrid",
        "posicion": "Defensa"},
        {"nombre": "Marcelo Vieira", "equipo": "Real Madrid",
        "posicion": "Defensa"}
]

with open('jugadores.txt', 'wb') as fichero:
    pickle.dump(rows, fichero)

with open('jugadores.txt', 'rb') as fichero:
    n_jugadores = pickle.load(fichero)
    print(n_jugadores)
```

Otra forma de hacer persistente un objeto es con la librería Shelve de Python. Esta librería trabaja sobre pickle y permite almacenar objetos como un diccionario. Es muy útil cuando queremos guardar muchos objetos y posteriormente acceder solo a algunos de ellos.

```
import shelve

rows = [{"nombre": "Thibaut Courtois",
        "equipo": "Real Madrid", "posicion": "Portero"},
        {"nombre": "Sergio Ramos", "equipo": "Real Madrid",
        "posicion": "Defensa"},
        {"nombre": "Marcelo Vieira", "equipo": "Real Madrid",
        "posicion": "Defensa"} ]
```

Escribimos los datos asociándoles una clave

```
shelf = shelve.open('file')
shelf["uno"] = rows[0]
shelf["dos"] = rows[1]
shelf["tres"] = rows[2]
```

Leemos los datos y los mostramos

```
shelf = shelve.open('file')
for key in shelf.keys():
    print(repr(shelf[key]))
```

Formato y Modulo JSON

Es común, que las listas y diccionarios anidados se compartan entre los sistemas informáticos. Se ha definido un formato estándar que facilita compartir esos datos. Se llama JSON, su nombre es un acrónimo de las siglas en inglés de JavaScript Object Notation. Lo que indica que su origen se encuentra vinculado al lenguaje JavaScript. Aunque hoy en día puede ser utilizado desde casi todos los lenguajes de programación. JSON se ha hecho fuerte como alternativa a XML, otro formato de intercambio de datos que requiere más metainformación y, por lo tanto, consume más ancho de banda y recursos.

Los datos en los archivos JSON son pares de propiedad valor separados por dos puntos. Estos pares se separan mediante comas y se encierran entre llaves. El valor de una propiedad puede ser otro objeto JSON, lo que ofrece una gran flexibilidad a la hora de estructurar información. Esta estructura de datos recuerda mucho a los diccionarios de Python.

Python hace que sea bastante fácil leer y escribir datos en formato JSON. De hecho, el formato JSON se parece casi exactamente a la impresión de una lista de Python o un diccionario. En la mayoría de los casos, realmente no se daría cuenta de la diferencia, pero hay un par de pequeñas diferencias como:

1. El valor de Python None se representa como la palabra null
2. true y false en Python debe iniciar en mayúsculas.

Modulo JSON

Python proporciona un módulo que permite transformar los archivos JSON en diccionarios o viceversa. El módulo se llama JSON, de este módulo vamos a utilizar las funciones loads y dumps.

`json.loads()` este método toma una cadena como entrada y devolverá un objeto de tipo diccionario sobre el que se puede iterar.

Veamos el siguiente ejemplo, con algunos datos que podríamos obtener de iTunes de Apple, en formato JSON:

```
import json

string = '\n\n\n{\n "resultCount":25,\n "results":\n [\n{"wrapperType":"track","kind":"podcast",\n "collectionId":10892}}\n}'

d = json.loads(string) # convertiremos el formato JSON a dict
d

{'resultCount': 25,
 'results': [{'wrapperType': 'track',
               'kind': 'podcast',
```

```
'collectionId': 10892}}]
```

Una vez que la cadena es convertida en un diccionario, podemos tener obtener las claves y los valores.

```
print(d.keys())
dict_keys(['resultCount', 'results'])

print(d['resultCount'])
25

print(d['results'])
[{'wrapperType': 'track', 'kind': 'podcast', 'collectionId':
10892}]

print(d['results'][0])
{'wrapperType': 'track', 'kind': 'podcast', 'collectionId':
10892}

print(d['results'][0]['collectionId'])
10892
```

`json.dump()`: Este método hace lo contrario de `loads`. Toma un objeto Python, normalmente un diccionario o una lista, y devuelve una cadena, en formato JSON. Tiene algunos otros parámetros.

```
import json

data = {'key1': {'c': True, 'a': 90, '5': 50}, 'key2': {'b': 3,
'c': "yes"}}

d = json.dumps(data, sort_keys=True, indent=2)
print(d)
{
  "key1": {
    "5": 50,
    "a": 90,
    "c": true
  },
  "key2": {
    "b": 3,
    "c": "yes"
  }
}
```

El comportamiento de la librería `json` se puede personalizar utilizando diferentes opciones.

Sangría o indentación

El parámetro `indent` espera un entero. Cuando se proporciona, `dumps` genera una cadena adecuada para mostrar, con nuevas líneas indentadas (sangría) para listas anidadas o diccionarios.

Codificación Unicode

Una de las opciones más importantes puede ser la codificación de texto empleada en el archivo. Por defecto el paquete `json` genera los archivos en código ASCII. Cuando existen caracteres no ASCII estos serán escapados, aunque puede ser más interesante utilizar en estos casos codificación Unicode. Para conseguir este cambio solamente se ha de configurar la opción `ensure_ascii` a `False`. La diferencia se puede apreciar en el siguiente ejemplo.

```
data = {'first_name': 'Daniel', 'last_name': 'Rodríguez'}
json.dumps(data)
{"first_name": "Daniel", "last_name": "Rodr\\u00edguez"}

json.dumps(data, ensure_ascii=False)
{"first_name": "Daniel", "last_name": "Rodríguez"}
```

Ordenación

Los objetos JSON son una colección desordenada de conjuntos de para clave valor. El orden de los datos se encuentra definido por la posición en la que estos aparecen en el archivo. En el caso de que sea necesario ordenarlos por la clave se puede configurar la opción `sort_keys` a `True`. El resultado de utilizar esta opción se puede ver en el siguiente ejemplo.

```
data = {
    'first_name': 'Sigrid',
    'last_name': 'Mannock',
    'age': 27,
    'amount': 7.17}
json.dumps(data, sort_keys=True)
{"age": 27, "amount": 7.17, "first_name": "Sigrid",
"last_name": "Mannock"}
```

Obtener datos en formato JSON desde un servicio web

El método `json.loads()` es de gran utilidad cuando como resultado de un servicio web se obtiene una cadena de texto con un objeto JSON, el cual se puede transformar fácilmente en un diccionario.

Veamos un ejemplo:

```
import json
import requests

response =
requests.get('https://itunes.apple.com/gb/rss/customerreviews/i
d=1145275343/page=2/json')
```

Nivel II: Representación, proceso y visualización de Datos

```
json_data = json.loads(response.text)
data = json_data['feed']['entry']

data
[{'author': {'uri': {'label':
'https://itunes.apple.com/gb/reviews/id1092361046'},
'name': {'label': 'bygtda'},
'label': ''},
'im:version': {'label': '3.0.20'},
'im:rating': {'label': '2'},
'id': {'label': '6549202856'},
'title': {'label': 'Rubbish'},
'content': {'label': "Rubbish, As soon as you're out of
Tickets...."}

data[1]["title"]["label"]
'Amazing 😄😄😄😄😄'
```

Almacenar y Leer los datos en formato JSON en un fichero

Utilizando la función `open()`, podemos guardar los resultados obtenidos del servicio web en un fichero:

```
with open("data_file.json", "w") as write_file:
    json.dump(json_data, write_file)
```

Para leer el fichero:

```
with open("data_file.json", "r") as file:
    my_file = file.read()
    mis_datos = json.loads(my_file)
```

Base de Datos Documentales

MongoDB es una base de datos de uso general potente, flexible y escalable. Combina la capacidad de escalar con características tales como índices secundarios, consultas de rango, clasificación, agregaciones e índices geoespaciales. MongoDB es una base de datos orientada a documentos, no relacional.

Una base de datos orientada a documentos reemplaza el concepto de una "fila" con un modelo más flexible, el "documento". Al permitir documentos y arreglos incrustados (embedded), el enfoque orientado a documentos permite representar relaciones jerárquicas complejas con un solo registro. Esto encaja naturalmente en la forma en que los desarrolladores en los lenguajes modernos orientados a objetos piensan acerca de sus datos.

Tampoco hay esquemas predefinidos: las claves y los valores de un documento no son de tipos o tamaños fijos. Sin un esquema fijo, agregar o eliminar campos según sea necesario se vuelve más fácil. En general, esto hace que el desarrollo sea más rápido ya que los desarrolladores

Nivel II: Representación, proceso y visualización de Datos

pueden iterar rápidamente. También es más fácil experimentar. Los desarrolladores pueden probar docenas de modelos para los datos y luego elegir el mejor para seguir.

El modelo de datos orientado a documentos facilita la división de datos en varios servidores. MongoDB se encarga automáticamente de equilibrar los datos y la carga en un clúster, redistribuir documentos automáticamente y enrutar las lecturas y escrituras a las máquinas correctas.

Para este curso nos limitaremos a conectarnos desde un programa Python a una Base de Datos MongoDB alojada en un clúster creado en Atlas MongoDB, crearemos colecciones, documentos, así como las actividades básica de inserción, actualización y borrado de datos.

¿Para qué se usa MongoDB?

- Internet de las cosas, grupo industrial Bosch.
- Visualización geospacial de elementos de una ciudad en tiempo real (Boston city).
- Gestión de contenidos, caso de Sourceforge.
- Aplicaciones móviles, como compra de viajes por Expedia.
- Videojuegos como FIFA online 3.
- Log de eventos, caso de Facebook para recoger anuncios accedidos
- Algunos usuarios conocidos de MongoDB son: Ebay, Expedia, Orange, Barclays, Adobe, Telefónica, entre otros.

Modelo de Datos

MongoDB es una Base de Datos NoSQL orientada a documentos en formato BSON.

- Similar a JSON, con algunas particularidades:
 - Tipos de datos adicionales
 - Diseñado para ser más eficiente en espacio. Aunque en ocasiones un JSON puede ocupar menos que un BSON.
 - Diseñado para que las búsquedas sean más eficientes.

Conceptos

Colección: Conjunto de documentos. Símil con las tablas del modelo relacional, pero no tiene esquema, por lo que cada documento de una colección puede tener diferentes campos. Cada documento dentro de una colección tiene un identificador único `_id`.



Las colecciones pueden crearse de la siguiente forma:

```
db.createCollection(name, options)
```

donde *name* es el nombre de la colección y las opciones pueden ser size, autoindex, entre otras.

```
db.createCollection("socios", {size: 2145678899,autoIndexId: true})
```

En mongoDB, las colecciones se crean automáticamente la primera vez que se hace referencia a ellas (por ejemplo, al insertar un primer documento), por lo que no es necesario utilizar `createCollection`. Este es sólo necesario si se quieren modificar los valores por defecto de las opciones.

Documento: es la unidad básica de datos para MongoDB. Es un conjunto de pares campo-valor. Símil a las filas en relacional, pero no siguen un esquema de tabla predefinido: pueden tener tantos campos como se desee.

- Se almacena el nombre del campo junto a su valor.
- No hay valores nulos: el campo está, o no está (aunque existe un tipo de dato denominado nulo).
- Pueden contener otros documentos (jerarquía de documentos).



Nivel II: Representación, proceso y visualización de Datos

Campos: campos de un documento a los que se les asigna valor y sobre los que se pueden crear índices. Símil a las columnas en relacional.

- **_id:** campo especial que identifica a cada documento en una colección. Si no se le da valor en el insertado, se genera automáticamente.

Tipos de Datos

Numéricos:

- **Double:** coma-flotante de 64 bits.
- **Decimal:** coma-flotante de 128 bits.
- **Int:** enteros de hasta 32 bits.
- **Long:** enteros de hasta 64 bits.

Texto:

- **String:** strings UTF-8.
- **Regex:** almacena, expresiones regulares.

Fecha:

- **Date:** entero de 64 bits que representa el número de milisegundos desde el 1 de enero de 1970.
- **Timestamp:** entero de 64 bits, en el que los primeros 32 bits representan los segundos pasados desde el 1 de enero de 1970, y los otros 32 bits son ordinales incrementales. En una instancia mongod, cada valor de timestamp es único.

Especiales:

- **Array:** almacena un conjunto de elementos de cualquier tipo.
- **ObjectId:** tipo de dato único, principalmente utilizado para dar valor al campo **_id** de los documentos.
- **Javascript:** código javascript.
- **Null:** valor nulo.
- **Boolean:** valor booleano.

Consulta y modificación de datos: operaciones CRUD (Create, Read, Update, Delete):

Insertar documentos

Las inserciones son el método básico para agregar datos a MongoDB. Para insertar un solo documento, use el método `insertOne` de la colección:

```
db.socios.insertOne({numSocio: 300})
```

`insertOne()`: agregará una clave `"_id"` al documento (si no proporciona una) y almacenará el documento en MongoDB.

Nivel II: Representación, proceso y visualización de Datos

`insertMany()`: Insertar varios documentos en una colección. Este método le permite pasar una variedad de documentos a la base de datos. Esto es mucho más eficiente porque su código no hará un viaje de ida y vuelta a la base de datos para cada documento insertado, sino que los insertará en masa.

Borrar documentos

`deleteOne()` y `deleteMany()` Ambos métodos toman un documento de filtro como primer parámetro. El filtro especifica un conjunto de criterios para hacer coincidir la eliminación de documentos.

```
db.socios.deleteOne({"_id" : 4})
```

Para eliminar todos los documentos que coinciden con un filtro, usamos `deleteMany()`. La eliminación de documentos suele ser una operación bastante rápida. Sin embargo, si deseamos borrar una colección completa, es más rápido:

```
db.socios.drop()
```

Actualización de documentos

Una vez que un documento se almacena en la base de datos, se puede cambiar usando uno de varios métodos de actualización: `updateOne`, `updateMany` y `replaceOne`. `updateOne` y `updateMany` toman un documento de filtro como su primer parámetro y un documento modificador, que describe los cambios a realizar, como el segundo parámetro. `replaceOne` también toma un filtro como primer parámetro, pero como segundo parámetro `replaceOne` espera un documento con el que reemplazará el documento que coincida con el filtro.

```
db.collection.updateOne(<filter>, <update>, <options>)  
db.collection.updateMany(<filter>, <update>, <options>)  
db.collection.replaceOne(<filter>, <update>, <options>)
```

Consultar Documentos

Find: El método `find` se usa para realizar consultas en MongoDB. Las consultas devuelven un subconjunto de documentos en una colección, desde ningún documento hasta la colección completa. Los documentos que se devuelven están determinados por el primer argumento para encontrar, que es un documento que especifica los criterios de consulta.

Sintaxis: `db.collection.find(query, projection)`

query: Opcional. Especifica el filtro de selección mediante operadores de consulta. Para devolver todos los documentos de una colección, omita este parámetro o pase un documento vacío `{}`.

projection: Opcional. Especifica los campos a devolver en los documentos que coinciden con el filtro de consulta. Para devolver todos los campos en los documentos coincidentes, omite este parámetro.

```
# selecciona todos los documentos de la colección
db.socios.find()
```

```
# selecciona todos los documentos que numsocio sea mayor a 300

db.socios.find({"numSocio": {$gt: 300} })
```

Operadores:

"\$lt", "\$lte", "\$gt" y "\$gte" son todos operadores de comparación, correspondientes a <, <=, > y >=, respectivamente. "\$ne" no igual o diferente, pertenece (\$in) o no (\$nin) a un conjunto. También podemos utilizar los operadores lógicos: \$and, \$or, \$not y \$nor (que no se cumpla ninguna condición).

Projection:

- Valor 1 si queremos mostrar el campo.
- Valor 0 si no queremos mostrarlo.
- Por defecto, si no se declara nada en la proyección, se muestran todos los campos:

El campo `_id` se muestra siempre por defecto, así mostremos otros campos en la proyección con valor 1. Si queremos mostrar el campo `_id`, hay que ponerlo a 0 explícitamente en la proyección.

```
# selecciona todos los documentos que numsocio sea mayor a 300 y
muestra nombre, apellido y el id
```

```
db.socios.find(
  {numSocio: {$gt: 300} },
  {nombre: 1, apellido1: 1}
).limit(3)
```

```
# selecciona todos los documentos que numsocio sea mayor a 300 y
muestra nombre, apellido, No muestra el id
```

```
db.socios.find(
  {numSocio: {$gt: 300} },
  {nombre: 1, apellido1: 1, _id: 0}
).limit(3)
```

```
db.socios.find(
  {numSocio: {$gt: 300} },
  {nombre: 1, apellido1: 1, "contacto.teléfono": 1}
```

```
).limit(3)
```

Limit: Limita el número de colecciones retornadas. El 0 indica que no hay límite

Sort: ordena los resultados según los campos dados. Un valor de 1 indica ordenamiento ascendente, y un valor de -1 ordenamiento descendente:

```
db.socios.find({numSocio: {$eq: 300} }, { nombre: 1, apellido1: 1 }).sort({nombre: 1})
```

Count: cuenta el número de veces que aparece en los documentos de una colección un valor en un campo.

Distinct: Retorna un array con los diferentes valores que tiene un campo concreto en los documentos de una colección.

Group: agrupa los elementos que cumplan una condición concreta (*cond*) bajo los distintos valores de un campo (*key*).

Índices

Un índice de base de datos es similar al índice de un libro. En lugar de mirar todo el libro, la base de datos toma un atajo y solo mira una lista ordenada con referencias al contenido. Esto permite a MongoDB consultar órdenes de magnitud más rápido.

Los índices sobre los campos ayudan a optimizar las consultas. Una consulta que no utiliza un índice se denomina exploración de recopilación, lo que significa que el servidor tiene que "mirar todo el libro" para encontrar los resultados de una consulta.

Sintaxis:

```
db.<nombre_colección>.createIndex({<campo>: <orden>, ...})
```

Tipos de índices en mongoDB:

- **Default_id:** existe por defecto sobre el campo `_id`. Garantiza unicidad.
- **Single field:** índice creado sobre un solo campo de la colección. Puede ser ascendente (1) o descendente(-1):

```
db.socios.createIndex({"nombre": 1})
```

- **Compound:** índice creado sobre varios campos de la colección:

```
db.socios.createIndex({"nombre": 1, "apellido1": -1})
```

- **Multikey Index:** si se declara un índice sobre un campo array, se crea sobre cada elemento del array.
- **Otros índices:** geoespaciales, sobre texto, etc.

Algunas propiedades de los índices:

1. **TTL:** índices que “caducan” al cabo de un tiempo.

```
db.socios.createIndex({"nombre": 1}, {expireAfterSeconds: 7200})
```

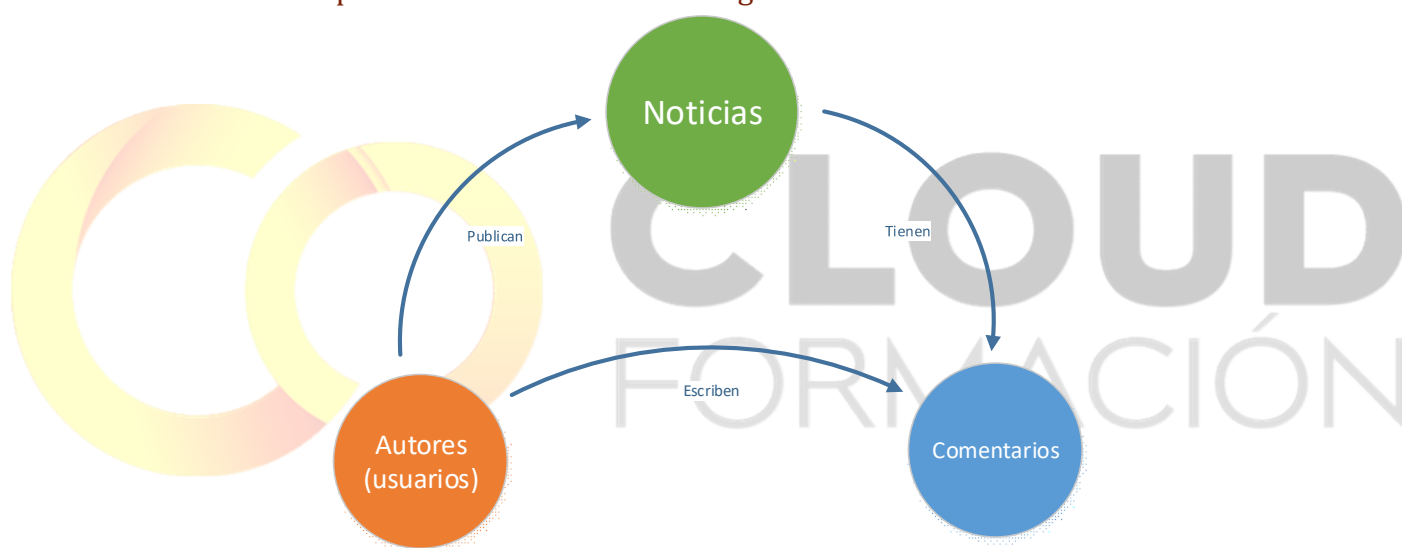
2. **Unique:** garantizan unicidad.

```
db.socios.createIndex({"numSocio": 1}, {unique: true})
```

Crear una base de Datos Documental en Python

Hemos visto de la manera muy resumida, las principales operaciones que podemos realizar con una base de datos documental MongoDB. Ahora, crearemos una base de datos documental utilizando Python, para ello, crearemos la base de datos blognoticias.

Consideraciones para el diseño de la BBDD blognoticias



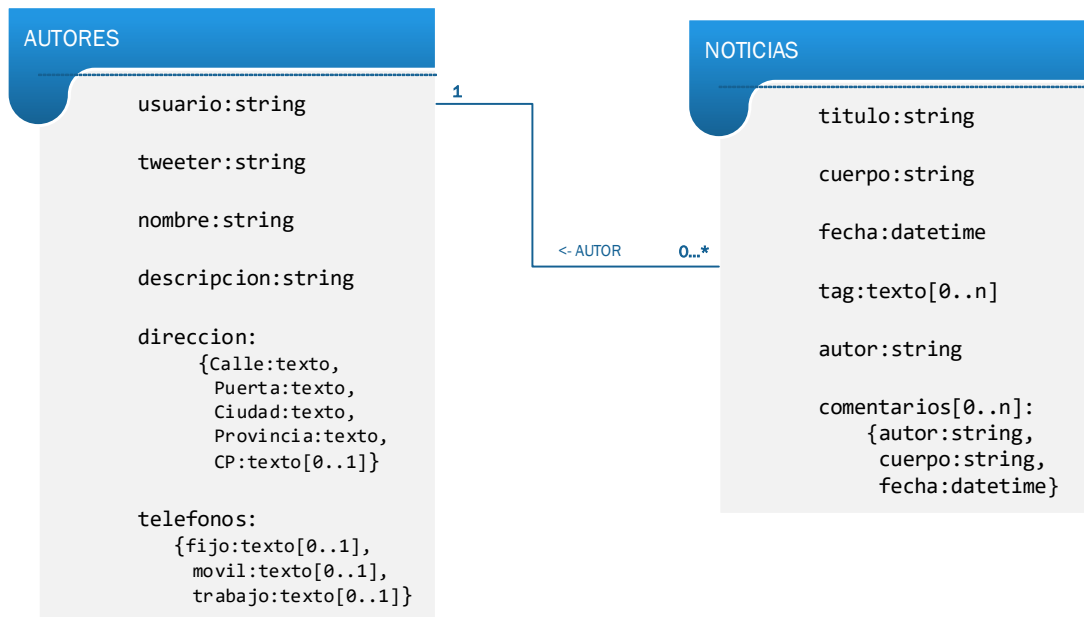
1. Cada autor tiene un nombre, un nombre de usuario, una cuenta de Twitter y una descripción. Además, de forma opcional, los usuarios pueden proporcionar como datos su dirección postal (calle, número, puerta, C.P., ciudad) o sus teléfonos de contacto (pueden tener varios).
2. Las noticias tienen un título, un cuerpo y una fecha de publicación. Son publicadas por un autor y pueden contener o no, una lista de tags.
3. Las noticias reciben comentarios, quedando registrado la persona que lo escribió, el comentario escrito y el momento en el que lo hizo.

Modelo de la Base de Datos blognoticias

La base de datos documental que crearemos para nuestro blog de noticias estará formada por dos colecciones Autores y Noticias. En la colección Autores tenemos dos documentos embebidos dirección y teléfonos. En la colección Noticias tenemos el documento embebido

Nivel II: Representación, proceso y visualización de Datos

comentarios, donde registraremos todos los comentarios de los diferentes usuarios a cada noticia.



PyMongo

PyMongo es una distribución de Python que contiene herramientas para trabajar con MongoDB, y es la forma recomendada de trabajar con MongoDB desde Python. Para instalar PyMongo:

```
pip install pymongo
```

Importar las librerías

```
import pymongo
```

Implementación de nuestra Base de Datos blognoticias

Paso 1: Crear una conexión

```
from pymongo import MongoClient

client = pymongo.MongoClient(
    "mongodb+srv://usuario:password@cluster0.nytnz.mongodb.net")
```

Paso 2: Conexión a la Base de Datos blognoticias

```
db = client.blognoticias
```

PASO 3: Creación de las colecciones

```
# Colección AUTORES
db.create_collection("autores")

# Colección NOTICIAS
```

```
db.create_collection("noticias")

# Visualizamos todas las colecciones de la BBDD blognoticias
print(db.list_collection_names())
['autores', 'noticias']
```

PASO 4: Insertar Datos de las colecciones

Colección Autores

```
autor = ([{
    "_id": "gmortiz",
    "tweeter": "@gmortiz",
    "nombre": "Gustavo Ortiz",
    "descripcion": "Me gustan los video juegos",
    "direccion": {
        'Calle': 'Ronda de Nelle',
        'Puerta': 'Primero izquierda',
        'Ciudad': 'A Coruna',
        'Provincia': 'A Coruna',
        'CP': '15009'
    },
    "telefonos": {
        'fijo': '8895262',
        'movil': '45555555',
        'trabajo': None
    }
}])
```

```
db.autores.insert_many(autor)
```

Colección Noticias

```
noticia = {
    "titulo": "Trabajo Remoto, una alternativa",
    "cuerpo": "Desde la pandemia el trabajo se ha convertido en una alternativa...",
    "fecha": hoy.isoformat(),
    "tag": ["Trabajo remoto", "Zoom"],
    "autor": "sierramy",
    "comentarios": []
}
```

```
id = db.noticias.insert_one(noticia)
```

Insertar un comentario a una Noticia

```
comentario = {
    "autor": "gmortiz",
    "cuerpo": "Excelente articulo..",
    "fecha": hoy.isoformat()
}
```

```
db.noticias.update_one({"_id": ObjectId("5f9c71cefa17d807fc755a02")},
```



```
 {"$push":{"comentarios":comentario}}})
```

Hemos creado las colecciones en nuestra base de datos blognoticias. Ahora, realizaremos algunas consultas:

PASO 5: Consultas

Consulta de TODOS los documentos de la colección AUTORES

```
print ("\n*** Consulta de TODOS los autores ***\n")
cursor = db.autores.find()
for bn in cursor:
    print( "%s - %s - %s - %s - %s - %s" \
           %(bn['_id'],bn['nombre'],bn['tweeter'],
             bn['descripcion'], bn['direccion'], bn['telefonos']))
```

Consulta de los documentos de AUTORES por nombre de usuario usuario = 'sierramy'

```
print ("\n*** Consulta de un autor por NOMBRE DE USUARIO  
***\n")
cursor = db.autores.find({"_id":usuario})
for bn in cursor:
    print( "%s - %s - %s - %s - %s" \
           %(bn['nombre'], bn['tweeter'], bn['descripcion'],
             bn['direccion'], bn['telefonos']))
    print(" ")
```

Consulta TODOS los documentos de la colección NOTICIAS

```
print ("\n*** Búsqueda de las Noticias ***\n")
cursor = db.noticias.find()
for bn in cursor:
    print( "%s - %s - %s - %s - %s - %s" \
           %(bn['_id'],bn['titulo'], bn['cuerpo'],bn['fecha'],
             bn['tag'], bn['autor']))
```

Consulta los documentos de la colección NOTICIAS por AUTOR

```
print ("\n*** Búsqueda de las Noticias por Autor ***\n")
cursor = db.noticias.find({"autor":"'sierramy'"})
for bn in cursor:
    print( "%s - %s - %s - %s - %s - %s - %s" \
           %(bn['_id'],bn['titulo'], bn['cuerpo'], bn['fecha'],
             bn['tag'], bn['autor'],bn["comentarios"]))
```

Paso 6: Creación de índices

Es importante destacar que antes de crear los índices las colecciones no deben tener datos.

Nivel II: Representación, proceso y visualización de Datos

```
# indice autores -> tweeter
db.autores.create_index([("tweeter", pymongo.ASCENDING)],
unique = True, name = "in_aut_tweeter")

# indice noticias -> autor
db.noticias.create_index([("autor", pymongo.ASCENDING)], name =
"in_not_autor")

# indice noticias -> comentarios noticia
db.noticias.create_index([("comentario", pymongo.DESENDING)],
sparse=True, name = "in_not_com")
```

Paso 7: Cerrar la conexión

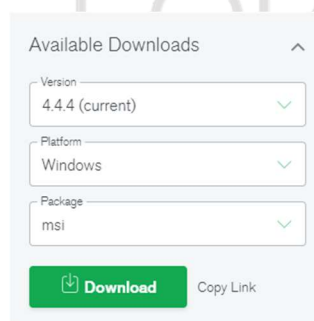
```
client.close()
```

Anexos

Instalación y configuración de MongoDB

Instalación de MongoDB Community (Local) y Compass

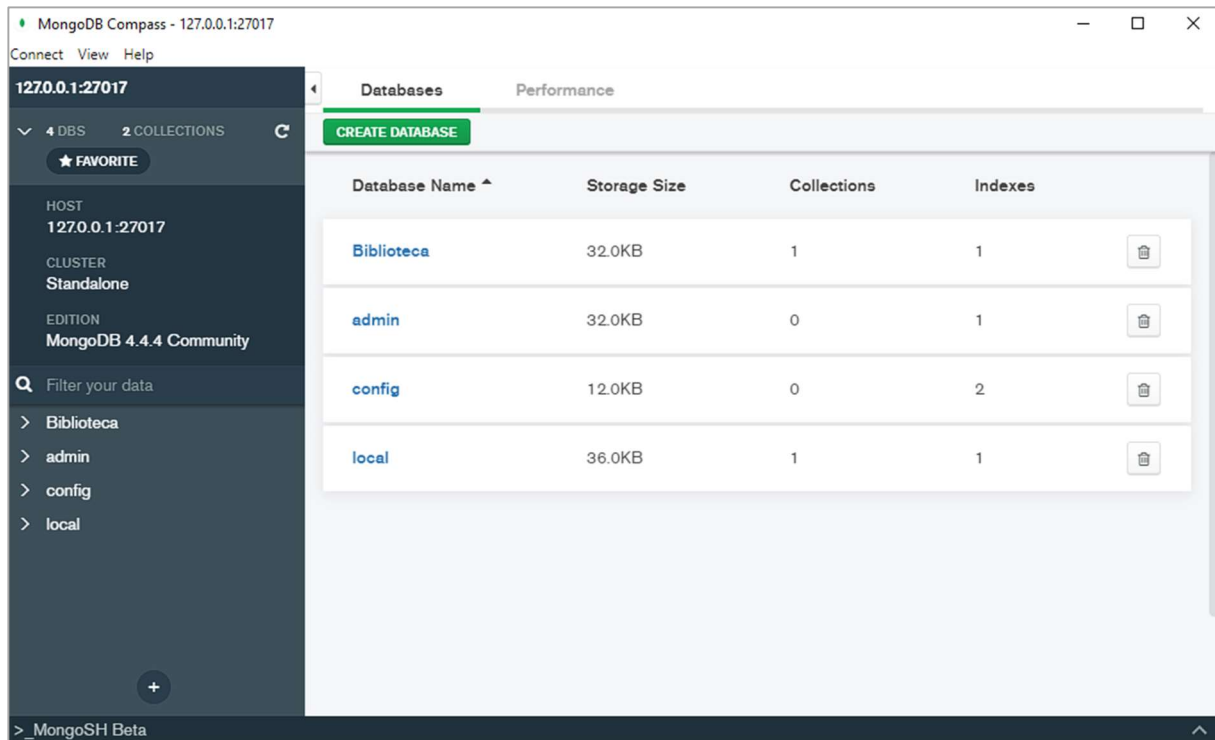
MongoDB ofrece una versión Enterprise y Community de su base de datos de documentos distribuidos. La versión Community ofrece el modelo de documento flexible junto con consultas, indexación y agregación en tiempo real. Obtenemos el instalador en: <https://www.mongodb.com/try/download/community>



Una vez obtienes el instalador (mongodb_windows.....signed.msi, lo ejecutas y sigues los pasos del instalación. Este instalador instala el Shell de Mongo y MongoDBCompass desde donde podemos gestionar tanto las BBDD locales como las que creamos en la nube con MongoDB Atlas.

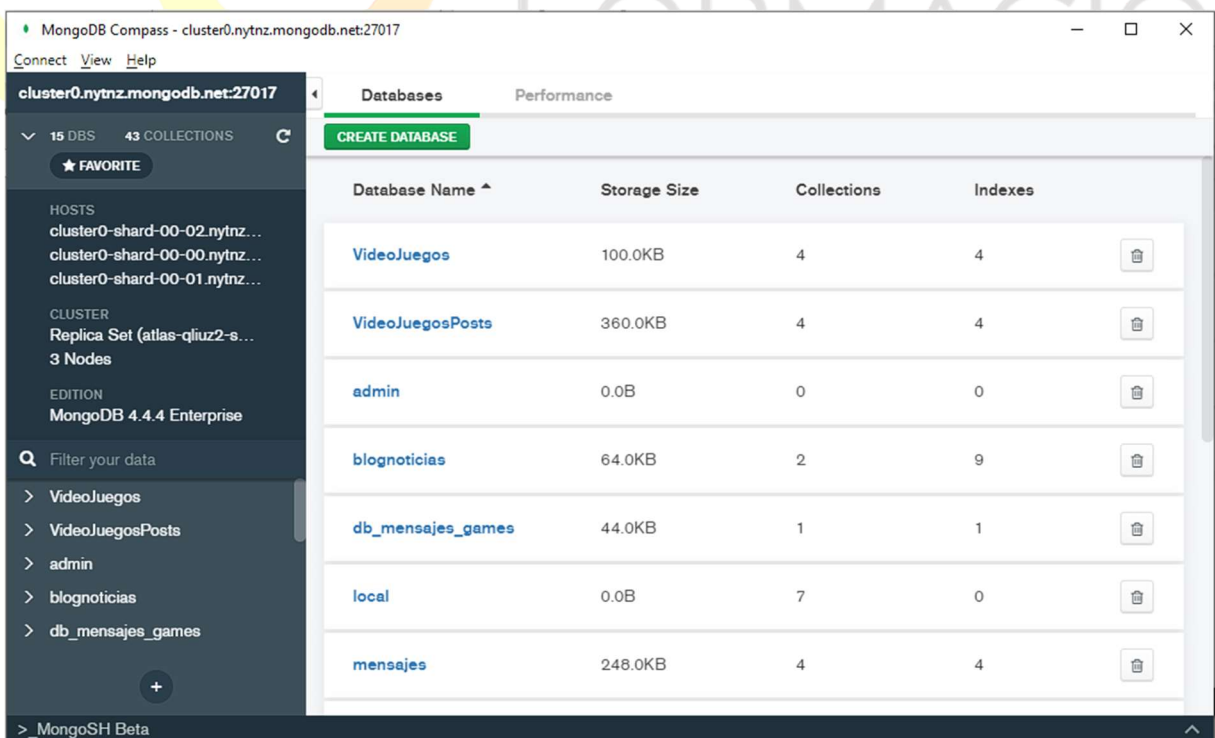
Conexión a una BBDD Local:

```
mongodb://127.0.0.1:27017/?readPreference=primary&appname=MongoDB%20
Compass&ssl=false
```



Conexión a una BBDD en Atlas:

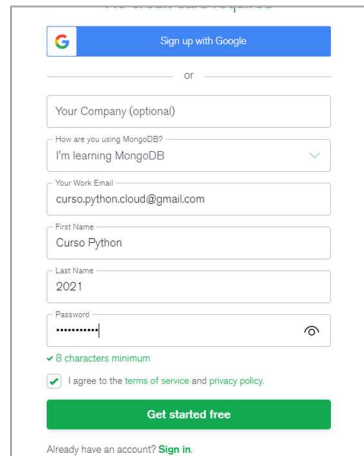
mongodb+srv://alumno:*****@cluster0.nytnz.mongodb.net/blognoticias?authSource=admin&replicaSet=atlas-qliuz2-shard-0&readPreference=primary&appName=MongoDB%20Compass&ssl=true



MongoDB Atlas

MongoDB Atlas es una base de datos en nube global pensada para aplicaciones modernas que es distribuida y segura por defecto, además de estar disponible como servicio totalmente gestionado en AWS, Azure y Google Cloud. Vamos a configurar nuestro cluster y Bases de datos MongoDB: <https://www.mongodb.com/cloud/atlas>.

1. Registro



Sign up with Google

or

Your Company (optional)

How are you using MongoDB?
I'm learning MongoDB

Your Work Email
curso.python.cloud@gmail.com

First Name
Curso Python

Last Name
2021

Password

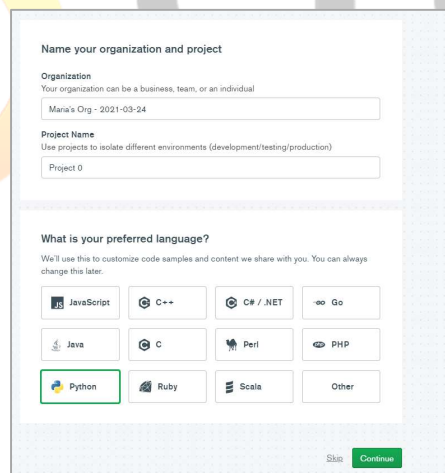
✓ 8 characters minimum

✓ I agree to the terms of service and privacy policy.

Get started free

Already have an account? Sign in.

2. Seleccionar el lenguaje de programación que utilizaremos



Name your organization and project

Organization
Your organization can be a business, team, or an individual
Maria's Org - 2021-03-24

Project Name
Use projects to isolate different environments (development/testing/production)
Project 0

What is your preferred language?
We'll use this to customize code samples and content we share with you. You can always change this later.

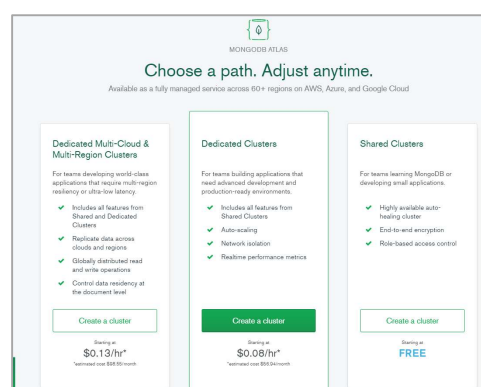
JavaScript C++ C# / .NET Go

Java C Perl PHP

Python Ruby Scala Other

Continue

3. Selecciona la opción Create a cluster - FREE



MONGODB ATLAS

Choose a path. Adjust anytime.

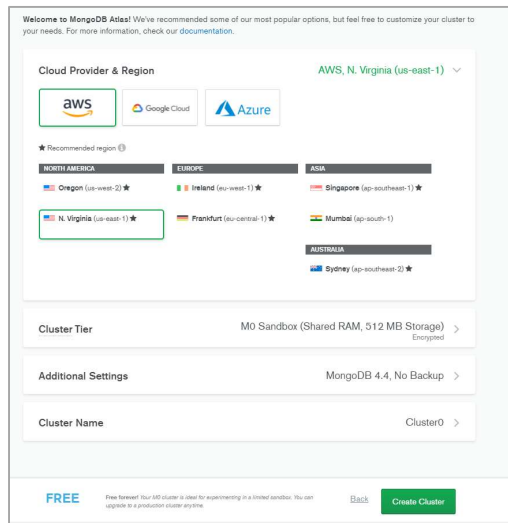
Available as a fully managed service across 60+ regions on AWS, Azure, and Google Cloud

Dedicated Multi-Cloud & Multi-Region Clusters	Dedicated Clusters	Shared Clusters
For teams developing world-class applications that require multi-region, resiliency or ultra-low latency.	For teams building applications that need advanced development and production-ready environments.	For teams learning MongoDB or developing small applications.
<ul style="list-style-type: none"> Includes all features from Shared and Dedicated Clusters Replicate data across clouds and regions Globally distributed read and write operations Control data residency at the document level 	<ul style="list-style-type: none"> Includes all features from Shared Clusters Auto-scaling Network isolation Realtime performance metrics 	<ul style="list-style-type: none"> Highly available auto-healing cluster End-to-end encryption Role-based access control
Create a cluster	Create a cluster	Create a cluster
Starting at \$0.13/hr* Minimum cost \$48.95/month	Starting at \$0.08/hr* Minimum cost \$24.00/month	Starting at FREE

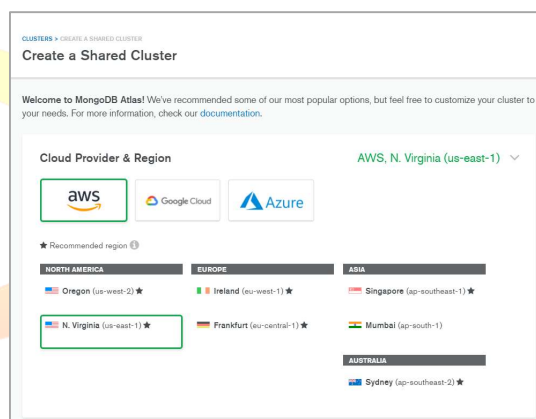
Nivel II: Representación, proceso y visualización de Datos

4. Creamos nuestro clúster donde alojaremos nuestras BBDD MongoDB:

a. Seleccionamos el proveedor de la nube AWS

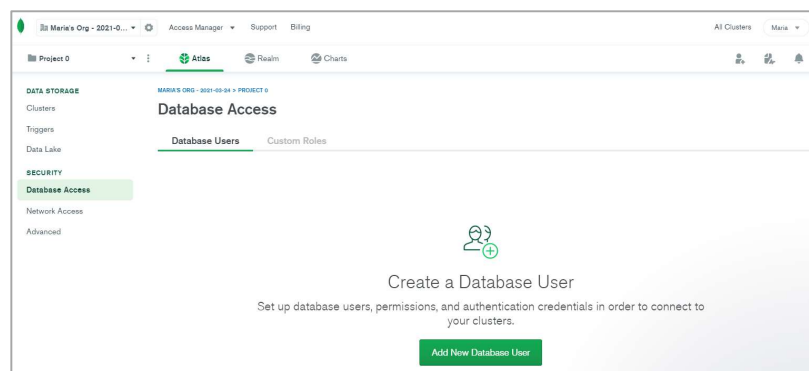


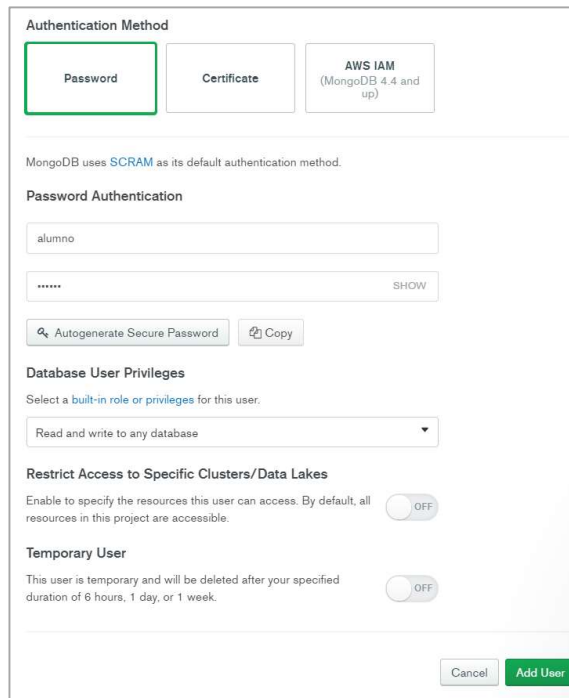
b. Seleccionamos una región que ofrezca el servicio free



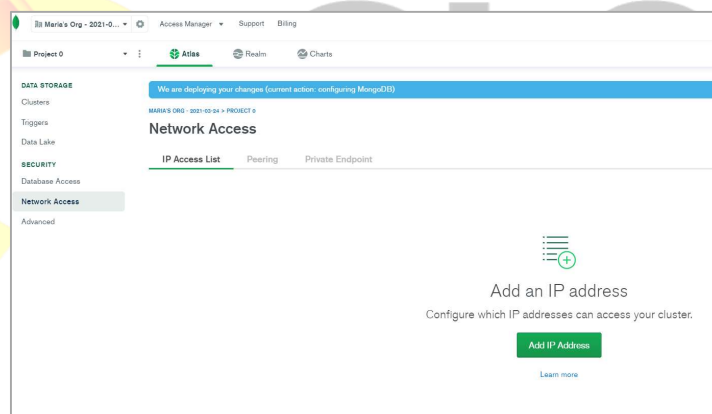
c. Seleccionamos las opciones por defecto para crear nuestro Cluster0

d. Creamos un usuario, seleccionamos Database Access – Add New User, introduzca el nombre del usuario y la contraseña, este será el usuario administrador. Es recomendable crear un usuario para acceder a nuestras BBDD desde la aplicación.

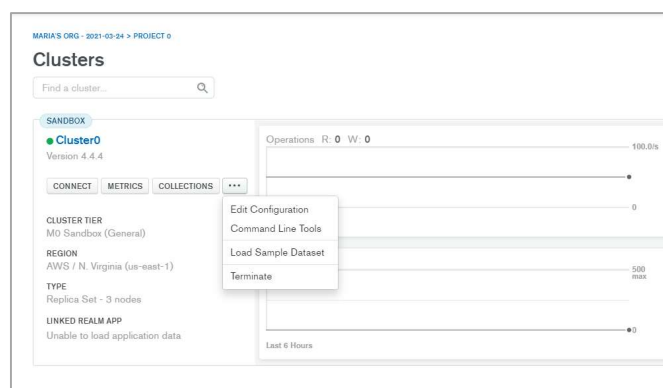




e. Configurar la IP actual



f. Seleccionar la opción Load Sample Database



g. Para la conexión desde Python utilizaremos la opción Connect with the Mongo Shell

Bibliografía

- Kristina Chodorow, E. B. (2019). MongoDB: The Definitive Guide, 3rd Edition. O'Really.
- <https://docs.mongodb.com/manual/>

