

# 3-CAT-Fitxers\_i\_interacció\_amb\_el\_sistema

March 21, 2020

## 1 Programació per a la ciència de dades

---

### 1.1 Unitat 3: Fitxers i interacció amb el sistema

---

#### 1.1.1 Instruccions d'ús

Aquest document és un notebook interactiu que intercala explicacions més aviat teòriques de conceptes de programació amb fragments de codi executables. Per aprofitar els avantatges que aporta aquest format, us recomanem que, en primer lloc, llegiu les explicacions i el codi que us proporcionem. D'aquesta manera tindreu un primer contacte amb els conceptes que hi exposem. Ara bé, **la lectura és només el principi!** Una vegada hagueu llegit el contingut proporcionat, no oblideu executar el codi proporcionat i modificar-lo per crear-ne variants, que us permetin comprovar que heu entès la seva funcionalitat i explorar-ne els detalls d'implementació. Per últim, us recomanem també consultar la documentació enllaçada per explorar amb més profunditat les funcionalitats dels mòduls presentats.

```
[1]: %load_ext pycodestyle_magic  
[2]: # Activem les alertes d'estil  
      %pycodestyle_on
```

#### 1.1.2 Introducció

En aquesta unitat veurem com podem interactuar amb el sistema operatiu fent servir Python.

En primer lloc, ens centrarem en els fitxers: veurem com es pot llegir el contingut d'un fitxer en Python, com es poden crear nous fitxers i escriure'n el seu contingut, com podem realitzar altres tasques bàsiques sobre fitxers (com ara canviar el nom d'un fitxer, esborrar-lo, o crear carpetes), i alguns altres detalls a tenir en compte a l'hora de treballar amb fitxers des de Python.

Seguidament, s'expliquen les nocions bàsiques per treballar amb fitxers comprimits des de Python, tot explicant com llegir i crear fitxers comprimits, i altres tasques bàsiques com ara llistar el contingut d'un fitxer comprimit.

A continuació es presenten les funcions de càrrega de dades de més alt nivell, que permeten carregar conjunts de dades des de fitxers sense necessitat de llegir i processar-ne manualment el seu contingut.

Després, veurem una alternativa per aconseguir persistència de dades de les nostres aplicacions: la serialització amb `pickle`.

Ja per acabar, explicarem com podem interactuar amb una base de dades SQL des del nostre codi Python, i com podem executar altres programes també des del nostre codi.

A continuació s'inclou la taula de continguts, que podeu fer servir per a navegar pel document:

**Nota important:** L'execució d'aquest notebook modifica els fitxers de la carpeta de la unitat 3 (es creen nous fitxers, se n'esborren d'altres, es modifica el contingut dels existents, etc.). Les explicacions que s'inclouen en aquest notebook concorden amb l'execució lineal de les cel·lules del notebook la primera vegada que es fa aquesta execució. A partir d'aleshores, si torneu a executar el notebook (o si altereu l'ordre d'execució de les cel·lules per fer proves), les explicacions poden no coincidir exactament amb els resultats que es produiran de l'execució, ja que l'estat inicial dels fitxers no serà la mateixa.

Si voleu restaurar l'estat inicial de tots els fitxers que s'alteren a l'executar aquest notebook (per poder tornar a executar el notebook tal com estava inicialment), obriu una consola i executeu:

```
cd ~/prog_datasci_2/resources/unit_3 && rm -rf files_folder file_2.txt mem_data; unzip files_f
```

---

## 2 1.- Fitxers (amb el mòdul `os`)

El mòdul `os` proveeix de funcions per interactuar amb el sistema operatiu. Entre d'altres, inclou funcions per manipular fitxers.

### 2.1 1.1.- Lectura i escriptura de fitxers

Per tal de llegir i/o escriure un fitxer, el primer que cal fer és obrir-lo, especificant-ne el seu *path* i el mode d'obertura. Amb el fitxer obert, podrem operar-hi (ja sigui llegir-lo o escriure-hi). Finalment, quan haguem finalitzat l'operació, caldrà tancar-lo, per tal d'alliberar els recursos.

**És important tancar els fitxers que obrim!** Més enllà de ser una bona pràctica, cal tenir en compte que certs canvis en els fitxers poden no ser visibles fins que es tanquin els fitxers (ja que el sistema operatiu implementa *buffers* per optimitzar-ne la gestió). A més, tot i que Python té un sistema automàtic de tancament de recursos, pot ser que aquest falli, deixant els fitxers oberts i, en conseqüència, consumint memòria RAM (i, per tant, impactant negativament en el rendiment del programa), i contribuint als comptadors que controlen el màxim nombre de fitxers oberts.

```
[3]: # Importem el mòdul os
import os

# Obriem el fitxer test_file.txt per a escriptura
out = open('files_folder/test_file.txt', 'w')
# Escrivim la paraula 'test' al fitxer
out.write("test")
# Tanquem el fitxer
out.close()
```

Aquest és el flux tradicional de treball amb fitxers: s'obre el fitxer especificant el *path* i el mode, s'opera amb el fitxer (en aquest cas, hi hem escrit la paraula 'test') i, finalment, es tanca el fitxer.

Després d'executar la cel·la anterior, obriu el fitxer (amb el vostre navegador de fitxers del sistema operatiu), i comproveu que, efectivament, conté la paraula 'test'.

Aquesta estructura de treball amb fitxers força al programador a recordar tancar el fitxer manualment una vegada ha acabat d'operar. Com a alternativa a aquesta estructura tradicional, Python permet fer servir la sentència `with`, que crea un context en el qual el fitxer es troba obert, i allibera al programador de la tasca de recordar tancar el fitxer. Així, una vegada l'execució surti del context, Python tancarà automàticament el fitxer:

```
[4]: # Obrim el fitxer test_file_2.txt per a escriptura
with open('files_folder/test_file_2.txt', 'w') as out:
    # Escrivim 'another test' al fitxer
    out.write("another test")

# Intentem escriure més contingut al mateix fitxer
try:
    out.write("fail")
except ValueError as e:
    print(e)
```

I/O operation on closed file.

Fixeu-vos com, si intentem operar amb el fitxer fora del context del `with`, es genera una excepció, ja que el fitxer ja està tancat.

### 2.1.1 1.1.1.- El path

El primer argument que rep la funció `open` (i l'únic que és obligatori) és el *path*. El *path* pot ser absolut o relatiu al directori on s'està executant el codi.

Indicarem que el *path* és **absolut** iniciant-lo amb una barra, / (que indica el directori arrel).

En canvi, si el *path* comença amb un caràcter, aquest serà **relatiu** al directori on s'executa el codi. Així, en els dos exemples anteriors, els *paths* `test_file.txt` i `test_file_2.txt` eren relatius al directori d'execució, indicant que els fitxers es trobaven directament en el propi directori.

De la mateixa manera que en el sistema operatiu, podem fer servir `.` i `..` per a referir-nos, respectivament, al directori actual i al superior a l'actual en *paths* relatius. Especificarem el *path* fent servir també barres / després de cada nom de directori.

### 2.1.2 1.1.2.- El mode

En relació amb el mode d'obertura, Python reconeix els següents modificadors, que es poden combinar entre ells per especificar com i amb quina finalitat s'obre el fitxer:

- `r`, mode de lectura (de l'anglès, *reading*).
- `w`, mode d'escriptura (de l'anglès, *writing*), sobreescriu el contingut del fitxer si aquest ja existeix, o bé crea el fitxer si no existeix.
- `x`, mode de creació exclusiva.
- `a`, mode d'escriptura, escriu al final del fitxer, a continuació del contingut ja existent al fitxer (de l'anglès, *append*), o bé crea el fitxer si no existeix.
- `b`, mode **binari**.
- `t`, mode de **text** (valor per defecte).

- +, mode d'actualització (tant per lectura com per escriptura).

Python permet obrir fitxer en mode binari (retornant els continguts com a bytes, sense descodificar-los) o bé en mode text (retornant els continguts com a cadenes de text, obtingudes de descodificar els bytes en funció de la plataforma on s'executi el codi o bé de la codificació especificada). Per defecte (és a dir, si no s'especifica el mode), els fitxers s'obren en mode text, de manera que, per exemple, `r` i `rt` són equivalents.

Tant el mode `w` com el mode `a` permeten escriure en un fitxer. La diferència entre aquests modes rau en el tractament del contingut existent en el fitxer: `w` sobreescriu el contingut del fitxer, eliminant el contingut ja existent i incorporant-ne el nou; en canvi, `a` escriu a continuació del contingut ja existent al fitxer, afegint el nou contingut després del contingut ja existent.

El mode d'actualització, `+`, permet obrir un fitxer per escriure i llegir. Així, tant `w+` com `r+`, permetran llegir i escriure un fitxer. La diferència entre ambdós modes recau en el comportament respecte al contingut existent en el fitxer i a l'existència del propi fitxer. Si especifiquem `w+`, sobre-escriurem el contingut del fitxer i crearem el fitxer si aquest no existeix; en canvi, si especifiquem `r+`, mantindrem el contingut del fitxer i es generarà un error si el fitxer no existeix.

A continuació es presenten alguns exemples del funcionament dels modes d'obertura de fitxers:

```
[5]: # Intentem obrir per a lectura un fitxer inexistent, cosa
#     # que generarà una excepció
p = 'files_folder/a_new_file.txt'
try:
    with open(p, 'r') as inp:
        pass
except FileNotFoundError as e:
    print(e)
```

[Errno 2] No such file or directory: 'files\_folder/a\_new\_file.txt'

```
[6]: # Intentem obrir el mateix fitxer per a escriptura (creant per tant
#     # el fitxer) i hi escrivim dues línies
with open(p, 'w') as out:
    out.write("The file did not exist\n")
    out.write("It now contains two sentences.\n")
```

```
[7]: # Tornem a llegir ara el fitxer (ara es llegirà correctament, ja que
#     # ha estat creat a la cel·la anterior)
try:
    with open(p, 'r') as inp:
        content = inp.read()
        # Mostrem el contingut del fitxer
        print(content)
except FileNotFoundError as e:
    print(e)
```

The file did not exist  
It now contains two sentences.

```
[8]: # Tornem a escriure en el mateix fitxer amb el mode 'w', de manera
# que se sobreescriurà el contingut anterior
with open(p, 'w') as out:
    out.write("What happens if we write again?\n")
```

Obriu ara el fitxer `a_new_file.txt` de la carpeta `files_folder` i comproveu com només hi ha la frase `What happens if we write again?`, ja que el contingut anterior (`The file did not exist...`) s'ha sobreescrit.

```
[9]: # Tornem a escriure ara en el mateix fitxer, però fent servir el mode
# 'a', de manera que escriurem a continuació del contingut ja existent.
with open(p, 'a') as out:
    out.write("And now, what happens if we write again?\n")
```

```
[10]: from io import UnsupportedOperation
# Ara intentarem afegir una altra frase al fitxer, i després llegir
# en el mateix fitxer (cosa que generarà un error)
try:
    with open(p, 'a') as out:
        out.write("...and again?\n")
        content = out.read()
except UnsupportedOperation as e:
    print(e)
```

not readable

Si obriu ara el fitxer, veureu que hi ha el següent text, resultat de les execucions de les cel·les anteriors:

```
What happens if we write again?
And now, what happens if we write again?
...and again?
```

Provem ara el comportament del mode de lectura amb actualització:

```
[11]: # Obrim el fitxer en mode de lectura amb actualització, escrivim una
# frase i llegim el contingut a partir del final de l'escriptura
with open(p, 'r+') as out:
    out.write("Trying the r+ mode!")
    content = out.read()
    print(content)
```

```
write again?
And now, what happens if we write again?
...and again?
```

En aquest últim exemple, el fitxer s'ha obert per a lectura amb actualització. A l'escriure, escrivim per tant a l'inici del fitxer (i sobreescrivim el contingut anterior conforme anem escrivint). Després, al llegir, llegim a partir d'on hem acabat d'escriure, de manera que només es llegeix el contingut ja existent (que no hem sobreescrit). El contingut del fitxer és ara per tant:

Trying the r+ mode!write again?  
And now, what happens if we write again?  
...and again?

Fixeu-vos com la primera línia conté la frase que hem escrit a la cel·la anterior, seguida dels caràcters que quedaven de la frase original!

### 2.1.3 1.1.3.- Altres detalls en l'obertura de fitxers

A més del *path* i el mode, la funció `open` accepta altres arguments opcionals, que gestionen el *buffering* de dades, la codificació, la gestió dels errors, la gestió del salt de línia, etc. El lector interessat pot consultar la [documentació oficial de la funció open](#) (lectura opcional) per descobrir com funcionen aquests arguments i quines opcions es troben disponibles.

### 2.1.4 1.1.4.- Lectura de fitxers grans

Com hem vist, el mètode `read` llegeix tot el contingut del fitxer. És evident, doncs, que fer servir aquest mètode pot comportar problemes de memòria i d'eficiència en el nostre codi, sobretot quan el fitxer a llegir sigui gran.

Una alternativa per a la lectura de fitxers és fer-la línia a línia, de manera que només una línia del fitxer es carrega en memòria cada vegada:

```
[12]: from sys import getsizeof

p_big = 'files_folder/somehow_big_file.txt'

# Carreguem el fitxer somehow_big_file.txt complet i mostrem
# la mida de la variable content en memòria
with open(p_big, 'r') as f:
    content = f.read()
    size_in_bytes = getsizeof(content)
    print("The size of the variable is: {} KB\n\n".format(
        size_in_bytes / 1024))

# Llegim el fitxer línia a línia (i únicament les 5 primeres línies),
# mostrant la mida de la variable line
with open(p_big, 'r') as f:
    counter = 0
    for line in f:
        print(line)
        size_in_bytes = getsizeof(line)
        print("The size of the variable is: {} KB\n\n".format(
            size_in_bytes / 1024))
        counter += 1
    if counter == 5:
        break
```

The size of the variable is: 250.087890625 KB

This document gives coding conventions for the Python code comprising the standard library in the main Python distribution. Please see the companion informational PEP describing style guidelines for the C code in the C implementation of Python [1].

The size of the variable is: 0.291015625 KB

This style guide evolves over time as additional conventions are identified and past conventions are rendered obsolete by changes in the language itself.

The size of the variable is: 0.1982421875 KB

Many projects have their own coding style guidelines. In the event of any conflicts, such project-specific guides take precedence for that project.

The size of the variable is: 0.1923828125 KB

One of Guido's key insights is that code is read much more often than it is written. The guidelines provided here are intended to improve the readability of code and make it consistent across the wide spectrum of Python code. As PEP 20 says, "Readability counts".

The size of the variable is: 0.3056640625 KB

A style guide is about consistency. Consistency with this style guide is important. Consistency within a project is more important. Consistency within one module or function is the most important.

The size of the variable is: 0.240234375 KB

## 2.2 1.2.- Creació de carpetes

Hem vist com podem crear fitxers obrint-los en mode 'w' o 'a'. Per a crear una carpeta o directori, podem fer servir els mètodes `mkdir` o `makedirs`:

```
[13]: # Creem la carpeta a_new_folder dins de la carpeta
      # existent files_folder
      new_folder = 'files_folder/a_new_folder'
      os.mkdir(new_folder)

      # Creem la carpeta an_empty_folder dins de la carpeta
```

```
# existent files_folder
new_folder = 'files_folder/an_empty_folder'
os.mkdir(new_folder)
```

```
[14]: # Intentem crear la carpeta 2 dins de la carpeta 1 dins de la carpeta
# a_new_folder que hem creat a la cel·la anterior
try:
    new_folder = 'files_folder/a_new_folder/1/2'
    os.mkdir(new_folder)
except FileNotFoundError as e:
    print(e)
```

[Errno 2] No such file or directory: 'files\_folder/a\_new\_folder/1/2'

La cel·la anterior genera una excepció, ja que la carpeta 'files\_folder/a\_new\_folder/1' dins de la qual volem crear la carpeta '2' no existeix. Si el que volem és crear tant la carpeta '1' com la carpeta '2' dins de la carpeta '1', podem fer servir [makedirs](#):

```
[15]: # Creem totes les carpetes que calgui de la següent estructura de
# carpetes
new_folder = 'files_folder/a_new_folder/1/2/3'
os.makedirs(new_folder)
```

## 2.3 1.3.- Esborrar i renombrar

La llibreria `os` també proveeix de funcions per a esborrar fitxers o directoris buits:

```
[16]: # Esborrem el fitxer a_new_file.txt
os.remove(p)
```

```
[17]: # Esborrem la carpeta buida an_empty_folder
empty_fold = 'files_folder/an_empty_folder'
os.rmdir(empty_fold)
```

```
[18]: # Intentem esborrar la carpeta files_folder, cosa que generarà
# un error ja que aquesta conté fitxers
try:
    non_empty_fold = 'files_folder'
    os.rmdir(non_empty_fold)
except OSError as e:
    print(e)
```

[Errno 39] Directory not empty: 'files\_folder'

De manera anàloga a l'execució d'un `rmdir` en una consola Linux, no podem esborrar una carpeta si aquesta conté fitxers. Si necessitem esborrar una carpeta i tots els fitxers que aquesta conté en Python, podem fer-ho a mà (esborrant primer els fitxers de dins de la carpeta i després la pròpia carpeta), o bé fent servir alguna funció que ja incorpori aquest comportament, com ara [rmtree](#) del mòdul [shutil](#).

La llibreria `os` també permet renombrar fitxers i directoris a través de la funció [rename](#):



```
[19]: # Renombrem el fitxer original_file.txt a dest_file.txt
os.rename('files_folder/original_file.txt',
          'files_folder/dest_file.txt')
```

## 2.4 1.4.- Funcions auxiliars de *paths*

Més enllà de la pròpia lectura, escriptura, esborrat i renombrat de fitxers, treballar amb fitxers acostuma a requerir d'altres funcions, sovint auxiliars, que permeten construir lògiques complexes de gestió de fitxers.

Així, per exemple, és habitual haver de construir una ruta (un *path*), que indiqui on es troba un fitxer en el sistema de fitxers a partir de fragments d'aquesta ruta (per exemple, el nom de la carpeta o ruta de carpetes, el nom del fitxer, etc.).

El submòdul `path` (del mòdul `os`) implementa la funció `join` que uneix diferents parts de la ruta d'un fitxer de manera *intel·ligent*, considerant el separador de directoris del sistema en què s'executa el codi. És una bona pràctica fer servir aquesta funció a l'hora d'especificar on es troba un fitxer en comptes de concatenar manualment les diferents parts de la ruta, ja que això millora la compatibilitat del codi en diferents sistemes.

```
[20]: # Unim diferents parts d'un path amb join
path = "/home"
full_path = os.path.join(path, "User/Desktop", "filename.txt")
print(full_path)
```

/home/User/Desktop/filename.txt

```
[21]: another_full_path = os.path.join(path, "User/Public/", "Documents", "")
print(another_full_path)
```

/home/User/Public/Documents/

En el primer cas, el *path* generat (`full_path`) corresponia a un fitxer, mentre que en el segon cas el *path* obtingut (`another_full_path`) era el d'una carpeta o directori (cosa que podem deduir per la / final). Fixeu-vos que, per a indicar que el *path* que volem obtenir és el d'una carpeta, hem indicat com a últim element de la crida a `join` una cadena buida ("").

D'altra banda, a partir d'un *path* podem obtenir-ne el nom de la carpeta, el nom del fitxer, o l'extensió del fitxer amb les funcions `dirname`, `basename` i `splitext`, respectivament.

```
[22]: # Obtenim el nom del directori
os.path.dirname(full_path)
```

```
[22]: '/home/User/Desktop'
```

```
[23]: # Obtenim el nom del fitxer
os.path.basename(full_path)
```

```
[23]: 'filename.txt'
```

```
[24]: # Separem l'extensió del path
os.path.splitext(full_path)
```

[24]: ('/home/User/Desktop/filename', '.txt')

Una altra funcionalitat sovint necessària en el tractament de fitxers és la de poder comprovar si un *path* existeix, o bé si aquest correspon a un fitxer normal, o bé a una carpeta. Aquestes tres comprovacions les podem fer amb les funcions `exists`, `isfile` i `isdir`, respectivament.

```
[25]: e = os.path.exists(full_path)
print("Path {} exists?\t\t\t\t\t".format(full_path, e))

path_1 = "./"
e = os.path.exists(path_1)
print("Path {} exists?\t\t\t\t\t".format(path_1, e))

path_2 = "3-CAT-Fitxers_i_interacció_amb_el_sistema.ipynb"
e = os.path.exists(path_2)
print("Path {} exists?\t\t\t\t\t".format(path_2, e))

path_3 = "/home"
e = os.path.exists(path_3)
print("Path {} exists?\t\t\t\t\t".format(path_3, e))
```

Path /home/User/Desktop/filename.txt exists?	False
Path ./ exists?	True
Path 3-CAT-Fitxers_i_interacció_amb_el_sistema.ipynb exists?	True
Path /home exists?	True

El primer *path* no correspon a cap path existent, de manera que `exists` retorna False. En canvi, tant els *paths* relatius `./` i `3-CAT-Fitxers_i_interacció_amb_el_sistema.ipynb` com el *path* absolut `/home` existeixen en el sistema, de manera que `exists` retorna True. És important notar que `exists` reconeix tant fitxers normals com carpetes o directoris. En canvi, la funció `isfile` retornarà True únicament si el *path* existeix i aquest és un fitxer normal:

```
[26]: is_f = os.path.isfile(full_path)
print("Path {} is file?\t\t\t\t\t".format(full_path, is_f))

is_f = os.path.isfile(path_1)
print("Path {} is file?\t\t\t\t\t".format(path_1, is_f))

is_f = os.path.isfile(path_2)
print("Path {} is file?\t\t\t\t\t".format(path_2, is_f))

is_f = os.path.isfile(path_3)
print("Path {} is file?\t\t\t\t\t".format(path_3, is_f))
```

Path /home/User/Desktop/filename.txt is file?	False
Path ./ is file?	False
Path 3-CAT-Fitxers_i_interacció_amb_el_sistema.ipynb is file?	True
Path /home is file?	False

En aquest cas, `isfile` reconeix únicament el tercer *path* com a fitxer, ja que el primer no existeix i els altres dos corresponen a carpetes.

Veiem ara el resultat de cridar `isdir` amb els mateixos *paths*:

```
[27]: is_d = os.path.isdir(full_path)
      print("Path {} is directory?\t\t{}".format(full_path, is_d))

      is_d = os.path.isdir(path_1)
      print("Path {} is directory?\t\t\t\t\t{}".format(path_1, is_d))

      is_d = os.path.isdir(path_2)
      print("Path {} is directory?\t{}".format(path_2, is_d))

      is_d = os.path.isdir(path_3)
      print("Path {} is directory?\t\t\t\t\t{}".format(path_3, is_d))
```

```
Path /home/User/Desktop/filename.txt is directory?      False
Path ./ is directory?                                   True
Path 3-CAT-Fitxers_i_interacció_amb_el_sistema.ipynb is directory?      False
Path /home is directory?                                 True
```

Ara, de nou, obtenim False per al primer *path*, ja que aquest no existeix, i obtenim True per als dos *paths* que representen directoris, ./ i /home.

## 2.5 1.5.- Llistat de directoris

Per tal d'obtenir tots els continguts que hi ha en un directori podem fer servir la funció `scandir`:

```
[28]: # Mostrem totes les entrades de la carpeta files_folder
      folder_name = 'files_folder/'
      with os.scandir(folder_name) as dir_list:
          for entry in dir_list:
              print(entry.name)
```

```
echo_read_script.sh
dest_file.txt
file_2.txt
test_file_2.txt
somehow_big_file.txt
a_new_folder
echo_script.sh
test_file.txt
a_number.txt
a_folder_in_a_zip
endless_script.sh
file_1.txt
zip_with_multiple_files.zip
```

Si volem obtenir una llista dels fitxers d'una carpeta, podem combinar `scandir` amb la funció `isfile` que hem vist en una secció anterior:

```
[29]: # Mostrem tots els fitxers de la carpeta files_folder
with os.scandir(folder_name) as dir_list:
    for entry in dir_list:
        if os.path.isfile(entry.path):
            print(entry.name)
```

```
echo_read_script.sh
dest_file.txt
file_2.txt
test_file_2.txt
somehow_big_file.txt
echo_script.sh
test_file.txt
a_number.txt
endless_script.sh
file_1.txt
zip_with_multiple_files.zip
```

Alternativament, podem fer servir el mètode `is_file` de les pròpies entrades, ja que `scandir` retorna un iterador que recorre objectes de tipus `DirEntry`, que implementen aquest mètode:

```
[30]: # Mostrem tots els fitxers de la carpeta files_folder
with os.scandir(folder_name) as dir_list:
    for entry in dir_list:
        if entry.is_file():
            print(type(entry), entry.name)
```

```
<class 'posix.DirEntry'> echo_read_script.sh
<class 'posix.DirEntry'> dest_file.txt
<class 'posix.DirEntry'> file_2.txt
<class 'posix.DirEntry'> test_file_2.txt
<class 'posix.DirEntry'> somehow_big_file.txt
<class 'posix.DirEntry'> echo_script.sh
<class 'posix.DirEntry'> test_file.txt
<class 'posix.DirEntry'> a_number.txt
<class 'posix.DirEntry'> endless_script.sh
<class 'posix.DirEntry'> file_1.txt
<class 'posix.DirEntry'> zip_with_multiple_files.zip
```

## 2.6 1.6.- Patrons de Unix shell

Combinant la possibilitat de llistar el contingut de directoris amb les expressions regulars que vam veure a la unitat d'estructures de dades avançades (o, fins i tot, potser només amb altres funcions bàsiques de cadenes, com `startswith`), podem seleccionar un subconjunt de fitxers que compleixin alguna característica concreta. Així, per exemple, podríem processar només els fitxers amb extensió `.txt` de la mateixa carpeta que a l'exemple anterior, fent:

```
[31]: # Mostrem tots els fitxers de la carpeta files_folder
with os.scandir(folder_name) as dir_list:
```

```

for entry in dir_list:
    if entry.is_file() and entry.name.endswith(".txt"):
        print(entry.name)

```

```

dest_file.txt
file_2.txt
test_file_2.txt
somehow_big_file.txt
test_file.txt
a_number.txt
file_1.txt

```

Una alternativa per fer aquest tipus d'operacions és utilitzar el mòdul `glob`, que permet obtenir els paths que segueixen un determinat patró especificat fent servir la sintaxis d'una *shell* de Unix (és a dir, la mateixa sintaxis que faríem servir si estéssim navegant pel sistema de fitxers des d'una *shell*).

```
[32]: import glob
```

```

# Obtenim una llista amb els noms dels fitxers de notebook de Python
glob.glob('*.ipynb')

```

```
[32]: ['3-CAT-Fitxers_i_interacció_amb_el_sistema.ipynb',
       '3-ES-Ficheros_e_interacción_con_el_sistema.ipynb']
```

```
[33]: # Llistem el contingut de la carpeta files_folder
glob.glob('./files_folder/*')
```

```
[33]: ['./files_folder/echo_read_script.sh',
       './files_folder/dest_file.txt',
       './files_folder/file_2.txt',
       './files_folder/test_file_2.txt',
       './files_folder/somehow_big_file.txt',
       './files_folder/a_new_folder',
       './files_folder/echo_script.sh',
       './files_folder/test_file.txt',
       './files_folder/a_number.txt',
       './files_folder/a_folder_in_a_zip',
       './files_folder/endless_script.sh',
       './files_folder/file_1.txt',
       './files_folder/zip_with_multiple_files.zip']
```

```
[34]: # Llistem els fitxers .txt que hi ha dins de la carpeta files_folder
glob.glob('./files_folder/*.txt')
```

```
[34]: ['./files_folder/dest_file.txt',
       './files_folder/file_2.txt',
       './files_folder/test_file_2.txt',
       './files_folder/somehow_big_file.txt',
       './files_folder/test_file.txt',
       './files_folder/a_number.txt',
```

```
 './files_folder/file_1.txt']
```

```
[35]: # Llistem tots els fitxers que hi ha al path actual, buscant recursivament  
# dins de les carpetes  
glob.glob('*', recursive=True)
```

```
[35]: ['README.md',  
      'files_folder',  
      '3-CAT-Fitxers_i_interacció_amb_el_sistema.ipynb',  
      '3-ES-Ficheros_e_interacción_con_el_sistema.ipynb',  
      'files_folder.zip',  
      'data']
```

## 2.7 1.7.- Obtenció de metadades dels fitxers

El submòdul `path` també conté funcions per obtenir metadades dels fitxers, com ara la seva mida (`getsize`), o l'instant de l'última modificació (`getmtime`).

```
[36]: # Obtenim la mida del fitxer p_big  
p_big_size = os.path.getsize(p_big)  
print("The file {} is {} KB".format(p_big, p_big_size / 1024))
```

The file files\_folder/somehow\_big\_file.txt is 250.0400390625 KB

```
[37]: from datetime import datetime  
  
# Obtenim la data d'última modificació  
unx_ts_mtime = os.path.getmtime(p_big)  
print("The last modification time is: {} (unix ts)".format(unx_ts_mtime))  
  
print("which is: {}".format(  
    datetime.utcfromtimestamp(unx_ts_mtime).strftime('%Y-%m-%d %H:%M:%S')))
```

The last modification time is: 1584526066.0 (unix ts)  
which is: 2020-03-18 10:07:46

També podem obtenir altres metadades, com ara les que obtindríem fen un `stat` de Linux sobre el fitxer:

```
[38]: import stat  
  
# Mostrem els bits de protecció del fitxer  
print(oct(stat.S_IMODE(os.stat(p_big).st_mode)))
```

0o664

Si teniu curiositat per saber com funcionen els bits de permís dels fitxers en unix, us recomanem llegir les tres parts de la sèrie d'articles sobre els permisos (1, 2, i 3), totes elles lectures opcionals.

## 3 2.- Treball amb fitxers comprimits

Un fitxer comprimit és un fitxer que conté un o diversos fitxers i/o carpetes, codificats de tal manera que ocupen menys espai en disc que els fitxers originals. Es fan servir fitxers comprimits, per exemple, per transferir més ràpidament contingut a través d'una xarxa, o per aprofitar millor l'espai de disc.

Distingim entre compressió **sense pèrdua** i compressió **amb pèrdua**. La compressió sense pèrdua es caracteritza per permetre recuperar la totalitat de les dades dels fitxers originals a partir dels fitxers comprimits. En canvi, en la compressió amb pèrdua, es poden perdre alguns bits d'informació. Es fa servir compressió amb pèrdua principalment en imatges i vídeo, on sovint les persones que visualitzen aquest contingut no arriben a notar la pèrdua. En canvi, en fitxers de text s'acostuma a fer servir compressió sense pèrdua.

En aquest notebook, s'explicarà com treballar amb fitxers comprimits zip, un dels formats més populars de compressió sense pèrdua, fent servir el mòdul `zipfile`. L'altra alternativa molt popular per comprimir sense pèrdua és fer servir gzip. El notebook no inclou explicacions sobre com treballar amb fitxers gzip, ja que el funcionament és molt similar al que es descriu per a zip. El lector interessat pot llegir la documentació del mòdul `gzip` per conèixer les funcions que permeten treballar amb fitxers gzip des de Python (lectura opcional).

### 3.1 2.1.- Lectura i escriptura de fitxers comprimits

De manera similar als fitxers genèrics, el primer que cal fer per llegir o crear un fitxer comprimit zip és obrir-lo, especificant-ne el seu *path* i el mode d'obertura. Quan haguem finalitzat l'operació, també caldrà tancar-lo. Podem fer servir la mateixa sintaxi que hem vist a l'apartat anterior per a gestionar l'obertura i tancament dels fitxers comprimits:

```
[39]: import zipfile as zf

zip_file = 'files_folder/compressed_file.zip'
# Creem el fitxer zip_file especificant el mode de compressió ZIP_DEFLATED
with zf.ZipFile(zip_file, 'w', compression=zf.ZIP_DEFLATED) as zip_f:
    # Afegim el fitxer p_big al zip
    zip_f.write(p_big)
```

A la cel·lula anterior, hem obert un fitxer zip en mode d'escriptura (especificat per 'w'), de manera que el fitxer es crearà si no existeix, o se sobreescriurà si ja existia; i hem especificat el mètode de compressió `ZIP_DEFLATED`. Amb el fitxer obert, hi hem afegit el fitxer ja existent `p_big`. Comprovem el resultat de la compressió:

```
[40]: # Obtenim la mida del fitxer original p_big
p_big_size = os.path.getsize(p_big)
print("The file {} is {} KB".format(p_big, p_big_size / 1024))

# Obtenim la mida del fitxer zip_file (que conté el fitxer p_big comprimit)
zip_file_size = os.path.getsize(zip_file)
print("The file {} is {} KB".format(zip_file, zip_file_size / 1024))
```

```
The file files_folder/somehow_big_file.txt is 250.0400390625 KB
The file files_folder/compressed_file.zip is 1.876953125 KB
```

Així doncs, el fitxer original de 249KB ha passat a ocupar només 1.3KB al ser comprimit.

Comprovem ara que podem recuperar el fitxer original a partir del fitxer comprimit. En primer lloc, calcularem un [hash](#) del contingut del fitxer, per poder assegurar que el fitxer que recuperarem és exactament el mateix que el fitxer original.

```
[41]: import hashlib

def sha256_file_content(p):
    """
    Returns the sha256 hash of the content of the file p.
    """
    with open(p, 'rb') as f:
        content = f.read()
        h = hashlib.sha256(content).hexdigest()
    return h

# Obtenim el hash del contingut del fitxer p_big
orig_hash = sha256_file_content(p_big)
print("sha256: {}".format(orig_hash))
```

sha256: 2093ab905569669d33c944efc6a528bcb6f9cd5d2c08a203e70f4d2a27f17a29

```
[42]: # Esborrem el fitxer p_big
os.remove(p_big)

# Comprovem que s'ha esborrat
is_f = os.path.isfile(p_big)
print("Path {} is file? {}".format(p_big, is_f))
```

Path files\_folder/somehow\_big\_file.txt is file? False

```
[43]: # Obrim el fitxer zip en mode de lectura
with zf.ZipFile(zip_file, 'r') as zip_f:
    # Mostrem el contingut del zip
    print(zip_f.printdir())
    # Descomprim tot el contingut del zip
    zip_f.extractall()
```

File Name	Modified	Size
files_folder/somehow_big_file.txt	2020-03-18 11:07:46	256041
None		

```
[44]: # Comprovem que el fitxer s'ha descomprimit
is_f = os.path.isfile(p_big)
print("Path {} is file? {}".format(p_big, is_f))
```



```
# Comprovem que el contingut del fitxer és exactament el mateix
uncomp_hash = sha256_file_content(p_big)
print("sha256: {}".format(uncomp_hash))
print("Hash are equal?: {}".format(uncomp_hash == orig_hash))
```

```
Path files_folder/somehow_big_file.txt is file? True
sha256: 2093ab905569669d33c944efc6a528bcb6f9cd5d2c08a203e70f4d2a27f17a29
Hash are equal?: True
```

### 3.1.1 2.1.1.- El path i el mode

Pel que fa al *path* dels fitxers comprimits, les mateixes consideracions que s'han fet sobre fitxers són aplicables en el cas dels fitxers comprimits.

Ja hem vist com els modes de lectura 'r' i escriptura 'w' de fitxers zip funcionen de manera similar als del mòdul `os`. Així, també disposem d'un mode de concatenació, 'a' (de l'anglès, *append*), que permet afegir contingut a un zip sense sobreescriure el contingut existent; i del mode de creació i escriptura exclusiva, 'x', que a diferència del mode d'escriptura, generarà una excepció si el fitxer que s'intenta crear ja existeix.

### 3.1.2 2.1.2.- Altres detalls en l'obertura de fitxers

A més del *path* i el mode, `ZipFile` accepta altres arguments opcionals, que gestionen el format de compressió, l'extensió per permetre fitxers majors de 4GB, i la comprovació de *timestamps*. El lector interessat pot consultar la [documentació oficial de la classe ZipFile](#) (lectura opcional) per descobrir com funcionen aquests arguments i quines opcions es troben disponibles.

### 3.1.3 2.1.3.- Lectura de fitxers grans

Hem vist que podem fer servir el mètode `extractall` per extreure tot el contingut d'un zip. Ara bé, si el fitxer zip és molt gran, conté diversos fitxers, i només en necessitem un subconjunt d'aquests, serà més eficient descomprimir únicament els fitxers que necessitem:

```
[45]: # Obrim el fitxer zip en mode de lectura
zip_with_multiple_files = "files_folder/zip_with_multiple_files.zip"
with zf.ZipFile(zip_with_multiple_files, 'r') as zip_f:
    # Descomprimim únicament el fitxer file_2.txt
    zip_f.extract("file_2.txt")
```

Fixeu-vos com, per extreure un únic fitxer, ens caldrà saber com es diu aquest fitxer, informació que podem saber ja, o bé que podem obtenir, com hem vist, amb `printdir`. Més endavant, veurem també una altra alternativa per a obtenir aquesta informació.

## 3.2 2.2.- Esborrar, renombrar i crear carpetes

El mòdul `zipfile` no permet, de manera nativa, esborrar o renombrar fitxers que es troben dins d'un zip. Per tant, el que caldrà fer si necessitem realitzar aquestes accions serà implementar-les manualment a partir dels modes de lectura i escriptura que hem vist a l'apartat anterior. Així, per exemple, si volem esborrar un cert fitxer d'un zip, caldrà descomprimir el zip, i tornar a crear

un nou zip amb el mateix contingut que el fitxer original, però sense incloure el fitxer que volem esborrar.

Per tal de crear una carpeta dins d'un fitxer zip, procedirem a crear la carpeta fora d'aquest, i l'afegirem després com si fos un fitxer ordinari, per exemple, amb la funció `write`.

### 3.3 2.3.- Funcions auxiliars de paths, llistats i metadades

De manera anàloga a les funcions que permetien comprovar si un fitxer era un directori o un fitxer normal, el mòdul `zipfile` disposa del mètode `is_zipfile` que permet comprovar si un fitxer és un fitxer zip vàlid:

```
[46]: izf = zf.is_zipfile(zip_file)
      print("The file {} is a zip file?: {}".format(zip_file, izf))
```

The file files\_folder/compressed\_file.zip is a zip file?: True

D'altra banda, ja hem vist com el mètode `printdir` ens permet obtenir un llistat dels continguts d'un fitxer zip:

```
[47]: # Obrim el fitxer zip en mode de lectura
zip_with_mult_files = "files_folder/zip_with_multiple_files.zip"
with zf.ZipFile(zip_with_mult_files, 'r') as zip_f:
    # Mostrem el contingut del zip
    print(zip_f.printdir())
```

File Name	Modified	Size
a_folder_in_a_zip/	2020-01-20 16:42:00	0
a_folder_in_a_zip/file_3.txt	2020-01-20 16:42:00	24
file_1.txt	2020-01-20 16:41:14	823
file_2.txt	2020-01-20 16:41:42	17
None		

Una altra alternativa per obtenir informació sobre els continguts d'un zip és fer servir la classe `ZipInfo`, que ens retorna precisament aquest tipus d'informació:

```
[48]: with zf.ZipFile(zip_with_mult_files, 'r') as zip_f:
      # Obtenim un objecte ZipInfo del fitxer zip_with_mult_files
      info_list = zip_f.infolist()

      # Per cada fitxer dins del zip, en mostrem informació
      for info in info_list:
          print("Filename: {}".format(info.filename))
          print("\tFile size: {} bytes".format(info.file_size))
          print("\tIs dir?: {}".format(info.is_dir()))
          print("\tDate and time: {}".format(info.date_time))
          print("\tCompression type: {}".format(info.compress_type))
          print("\tCRC: {}\n".format(info.CRC))
```

```
Filename: a_folder_in_a_zip/
      File size: 0 bytes
```

```
Is dir?: True
Date and time: (2020, 1, 20, 16, 42, 0)
Compression type: 0
CRC: 0
```

```
Filename: a_folder_in_a_zip/file_3.txt
File size: 24 bytes
Is dir?: False
Date and time: (2020, 1, 20, 16, 42, 0)
Compression type: 0
CRC: 2817114606
```

```
Filename: file_1.txt
File size: 823 bytes
Is dir?: False
Date and time: (2020, 1, 20, 16, 41, 14)
Compression type: 8
CRC: 3521977432
```

```
Filename: file_2.txt
File size: 17 bytes
Is dir?: False
Date and time: (2020, 1, 20, 16, 41, 42)
Compression type: 0
CRC: 742541709
```

Cal remarcar que un fitxer zip és un fitxer en tota regla. Per tant, podem obtenir metadades del propi fitxer zip amb les funcions que hem vist anteriorment del mòdul `os`. Per exemple, podríem obtenir la mida del zip fent servir `getsize`:

```
[49]: # Obtenim la mida del fitxer zip_with_mult_files
s_zip_with_mult_files = os.path.getsize(zip_with_mult_files)
print("The file {} is {} KB".format(zip_with_mult_files,
                                     s_zip_with_mult_files / 1024))
```

The file files\_folder/zip\_with\_multiple\_files.zip is 1.0263671875 KB

## 4 3.- Lectura i escriptura de fitxers amb pandas

En algunes situacions, podrem fer servir llibreries de més alt nivell per tal de llegir i/o escriure fitxers. Així, per exemple, la llibreria `pandas`, permet carregar dades d'un fitxer CSV a un *dataframe* a través de la funció `read_csv`.

Ara carregarem les dades del fitxer `marvel-wikia-data.csv`, que conté dades sobre personatges de còmic de Marvel. El conjunt de dades original en què es basa el que farem servir va ser creat pel web [FiveThirtyEight](#), que escriu articles basats en dades sobre esports i notícies, i que posa a disposició pública els [conjunts de dades](#) que recull per als seus articles.

```
[50]: import pandas as pd

# Carreguem les dades del fitxer "marvel-wikia-data.csv" a un 'dataframe'
data = pd.read_csv("data/marvel-wikia-data.csv")
```

La funció `read_csv` accepta un gran ventall de paràmetres opcionals que permeten configurar amb detall com s'ha de realitzar la importació del fitxer csv. A continuació, en veurem alguns d'ells, tot ajustant la importació de les dades de Marvel.

Fixem-nos, en primer lloc, en la importació de les columnes numèriques:

```
[51]: data.describe()
```

```
[51]:
```

	page_id	APPEARANCES	Year
count	31.000000	30.000000	31.000000
mean	12936.774194	1677.466667	1.885871
std	21056.994013	821.949228	0.358674
min	1073.000000	0.000000	0.000000
25%	1835.000000	1179.000000	1.961500
50%	2223.000000	1322.500000	1.963000
75%	8911.000000	2001.500000	1.964000
max	65255.000000	4043.000000	1.975000

Com es pot apreciar, hi ha hagut un problema en la importació dels anys en els quals els personatges apareixen per primera vegada als còmics, ja que la mitjana és 1.88 i, en canvi, els còmics van aparèixer al segle XX. Observant el contingut del fitxer csv en pla:

```
page_id,name,urlslug,ID,ALIGN,EYE,HAIR,SEX,GSM,ALIVE,APPEARANCES,FIRST APPEARANCE,Year
1678,Spider-Man (Peter Parker),\Spider-Man_(Peter_Parker),Secret Identity,Good Characters,Haz
7139,Captain America (Steven Rogers),\Captain_America_(Steven_Rogers),Public Identity,Good Cha
...
```

podem veure com els anys s'expressen fent servir un punt com a separador de milers, que pandas està interpretant com a separador decimal. Per tant, per assegurar que els anys s'importen correctament, podem indicar que el separador de decimals sigui la coma (,) amb el paràmetre `decimal`:

```
[52]: data = pd.read_csv("data/marvel-wikia-data.csv", decimal=",")
data.describe()
```

```
[52]:
```

	page_id	APPEARANCES
count	31.000000	30.000000
mean	12936.774194	1677.466667
std	21056.994013	821.949228
min	1073.000000	0.000000
25%	1835.000000	1179.000000
50%	2223.000000	1322.500000
75%	8911.000000	2001.500000
max	65255.000000	4043.000000

Ara, sembla que els anys s'han importat correctament, tot i que la mitjana segueix sent més baixa del valor que esperàriem. Observant les dades per al camp any, podem comprovar com l'última fila conté un 0, valor que fa baixar la mitjana:

```
[53]: data[["Year"]].tail()
```

```
[53]:      Year
26  1.963
27  1.963
28  1.964
29  1.975
30      0
```

Podem indicar que volem ometre aquesta fila en la càrrega de dades amb el paràmetre `skiprows`:

```
[54]: data = pd.read_csv("data/marvel-wikia-data.csv", decimal=",", skiprows=[31])
data[["Year"]].tail()
```

```
[54]:      Year
25  1.963
26  1.963
27  1.963
28  1.964
29  1.975
```

Com podem observar, això fa que la mitjana pugi de 1677.46 a 1735.31, i el mínim passi de ser 0 (el valor de la fila que hem omès) a 1047:

```
[55]: data.describe()
```

```
[55]:      page_id  APPEARANCES
count    30.000000    29.000000
mean   13034.700000   1735.310345
std    21409.788049    771.859772
min     1073.000000   1047.000000
25%    1834.000000   1230.000000
50%    2194.500000   1338.000000
75%     7652.000000   2017.000000
max    65255.000000   4043.000000
```

Si seguim observant les dades que s'han carregat, podem observar també com hi apareixen diverses contrabarres (\), per exemple, abans de cometes dobles (") o de les barres (/):

```
[56]: data.head(n=5)
```

```
[56]:      page_id      name \
0      1678  Spider-Man (Peter Parker)
1      7139  Captain America (Steven Rogers)
2     64786  Wolverine (James \"Logan\" Howlett)
3      1868  Iron Man (Anthony \"Tony\" Stark)
4      2460      Thor (Thor Odinson)

      urlslug      ID \
0  \Spider-Man_(Peter_Parker)  Secret Identity
1  \Captain_America_(Steven_Rogers)  Public Identity
2  \Wolverine_(James_%22Logan%22_Howlett)  Public Identity
```

```

3  \Iron_Man_(Anthony_%22Tony%22_Stark)  Public Identity
4  \Thor_(Thor_Odinson)  No Dual Identity

```

	ALIGN	EYE	HAIR	SEX	GSM \
0	Good Characters	Hazel Eyes	Brown Hair	Male Characters	dontknow
1	Good Characters	Blue Eyes	White Hair	Male Characters	NaN
2	Neutral Characters	Blue Eyes	Black Hair	Male Characters	dontknow
3	Good Characters	Blue Eyes	Black Hair	Male Characters	NaN
4	Good Characters	Blue Eyes	Blond Hair	Male Characters	NaN

	ALIVE	APPEARANCES	FIRST APPEARANCE	Year
0	Living Characters	4043.0	Aug-62	1.962
1	Living Characters	3360.0	Mar-41	1.941
2	Living Characters	3061.0	Oct-74	1.974
3	Living Characters	2961.0	Mar-63	1.963
4	Living Characters	2258.0	Nov-50	1.950

Aquestes contrabarras s'estan fent servir per a escapar caràcters especials, cosa que també podem indicar a la funció de càrrega del csv, amb el paràmetre `escapechar`:

```

[57]: data = pd.read_csv("data/marvel-wikia-data.csv", decimal=".", skiprows=[31],
                        escapechar="\\" )
data.head(n=5)

```

```

[57]:  page_id  name \
0      1678  Spider-Man (Peter Parker)
1      7139  Captain America (Steven Rogers)
2     64786  Wolverine (James "Logan" Howlett)"
3      1868  Iron Man (Anthony "Tony" Stark)"
4      2460  Thor (Thor Odinson)

```

	urlslug	ID \
0	/Spider-Man_(Peter_Parker)	Secret Identity
1	/Captain_America_(Steven_Rogers)	Public Identity
2	/Wolverine_(James_%22Logan%22_Howlett)	Public Identity
3	/Iron_Man_(Anthony_%22Tony%22_Stark)	Public Identity
4	/Thor_(Thor_Odinson)	No Dual Identity

	ALIGN	EYE	HAIR	SEX	GSM \
0	Good Characters	Hazel Eyes	Brown Hair	Male Characters	dontknow
1	Good Characters	Blue Eyes	White Hair	Male Characters	NaN
2	Neutral Characters	Blue Eyes	Black Hair	Male Characters	dontknow
3	Good Characters	Blue Eyes	Black Hair	Male Characters	NaN
4	Good Characters	Blue Eyes	Blond Hair	Male Characters	NaN

	ALIVE	APPEARANCES	FIRST APPEARANCE	Year
0	Living Characters	4043.0	Aug-62	1.962
1	Living Characters	3360.0	Mar-41	1.941
2	Living Characters	3061.0	Oct-74	1.974

3	Living Characters	2961.0	Mar-63	1.963
4	Living Characters	2258.0	Nov-50	1.950

Un altre detall a considerar és que es fa servir tant el valor NaN com la cadena de caràcters 'dontknow' per indicar valors desconeguts o perduts. Podem indicar que cal interpretar aquesta cadena com a valor perdut amb el mètode na\_values:

```
[58]: data = pd.read_csv("data/marvel-wikia-data.csv", decimal=".", skiprows=[31],
                        escapechar="\\", na_values=["dontknow"])
data.head(n=5)
```

```
[58]:
```

	page_id	name \
0	1678	Spider-Man (Peter Parker)
1	7139	Captain America (Steven Rogers)
2	64786	Wolverine (James "Logan" Howlett)"
3	1868	Iron Man (Anthony "Tony" Stark)"
4	2460	Thor (Thor Odinson)

  

	urlslug	ID \
0	/Spider-Man_(Peter_Parker)	Secret Identity
1	/Captain_America_(Steven_Rogers)	Public Identity
2	/Wolverine_(James_%22Logan%22_Howlett)	Public Identity
3	/Iron_Man_(Anthony_%22Tony%22_Stark)	Public Identity
4	/Thor_(Thor_Odinson)	No Dual Identity

	ALIGN	EYE	HAIR	SEX	GSM \
0	Good Characters	Hazel Eyes	Brown Hair	Male Characters	NaN
1	Good Characters	Blue Eyes	White Hair	Male Characters	NaN
2	Neutral Characters	Blue Eyes	Black Hair	Male Characters	NaN
3	Good Characters	Blue Eyes	Black Hair	Male Characters	NaN
4	Good Characters	Blue Eyes	Blond Hair	Male Characters	NaN

	ALIVE	APPEARANCES	FIRST APPEARANCE	Year
0	Living Characters	4043.0	Aug-62	1.962
1	Living Characters	3360.0	Mar-41	1.941
2	Living Characters	3061.0	Oct-74	1.974
3	Living Characters	2961.0	Mar-63	1.963
4	Living Characters	2258.0	Nov-50	1.950

Finalment, una altra de les funcionalitats bastant potents de la funció read\_csv és la d'aplicar alguna funció als elements de cada columna abans d'incorporar-lo al dataframe. Així, per exemple, la columna FIRST APPEARANCE conté el mes (abreviat), un guió, i l'any en format de dos dígit. Si volguéssim que aquesta columna contingues únicament l'any i en format de 4 dígit, podríem crear una funció anònima que fes la conversió, i passar aquesta funció a read\_csv amb el paràmetre converters:

```
[59]: data = pd.read_csv(
    "data/marvel-wikia-data.csv",
    decimal=".", skiprows=[31], escapechar="\\", na_values=["dontknow"],
    converters={"FIRST APPEARANCE": lambda x: int(x.split("-")[1])+1900})
```

```
)
data[["FIRST APPEARANCE"]].head()
```

```
[59]: FIRST APPEARANCE
0      1962
1      1941
2      1974
3      1963
4      1950
```

Més enllà dels fitxers csv, hi ha altres formats que també es fan servir sovint per intercanviar o desar dades. Pandas disposa de diverses funcions per a carregar dades provinents dels formats de dades més populars, com ara, json ([read\\_json](#)) o excel ([read\\_excel](#)).

## 5 4.- Serialització de dades

La **serialització** de dades és el procés de convertir dades estructurades (per exemple, una llista o un diccionari) en algun format que permeti emmagatzemar-les o transmetre-les, de manera que sigui possible després, a partir d'un procés de deserialització, recuperar les dades amb la seva estructura original.

La serialització pot ser útil, per exemple, per a transmetre un conjunt de dades a través de la xarxa, per emmagatzemar dades que seran tractades des de Python, o com a mètode per crear *checkpoints* que permetin recuperar l'estat dels nostres *scripts* en codis que tarden molt temps a finalitzar.

El mètode principal de serialització en Python es troba implementat pel mòdul [pickle](#) de la llibreria estàndard.

### 5.1 4.1- Serialització de dades amb pickle

Anem a veure un exemple senzill de serialització i deserialització d'una estructura de dades simple, un diccionari:

```
[60]: import pickle

# Creem un diccionari
a_dict = {"Spain": 34, "United Kingdom": 44}

# Serialitzem el diccionari i mostrem el resultat de la serialització
serialized_dict = pickle.dumps(a_dict)
print(serialized_dict)
```

```
b'\x80\x03}q\x00(X\x05\x00\x00\x00Spainq\x01K"X\x0e\x00\x00\x00United
Kingdomq\x02K,u.'
```

```
[61]: # Deserialitzem el diccionari, creant un nou diccionari a partir del contingut
# serialitzat
des_dict = pickle.loads(serialized_dict)

# Comprovem si el contingut del diccionari original i del deserialitzat són
```



```
# iguals
des_dict == a_dict
```

[61]: True

Després de veure un exemple de serialització d'un diccionari, en veurem un altre que s'apropa més a l'ús real de `pickle` en la mineria de dades. Suposem que hem entrenat un model d'aprenentatge automàtic a partir d'unes dades, i que ara el volem desar per tal de fer-lo servir, posteriorment, per a fer prediccions. En aquest cas, pot ser útil emmagatzemar el model après fent servir `pickle`, i recuperar-lo després quan calgui fer prediccions. Vegem un exemple amb la creació d'un arbre de decisió per al conjunt de dades de flors d'Iris:

```
[62]: from sklearn import datasets
      from sklearn.model_selection import train_test_split
      from sklearn.tree import DecisionTreeClassifier
      import pandas as pd

      # Carreguem les dades
      data = datasets.load_iris()
      iris_df = pd.DataFrame(data.data, columns=data.feature_names)
      y = data.target

      # Creem els conjunts de test i aprenentatge
      x_train, x_test, y_train, y_test = train_test_split(iris_df, y, test_size=0.1)

      # Entrenem un arbre de decisió
      dec_tree = DecisionTreeClassifier(splitter="random", criterion="entropy")
      dec_tree.fit(x_train, y_train)

      # Provem el classificador amb les dades de test
      y_test_predicted = dec_tree.predict(x_test)
      print("Predicted classes: \t" + str(y_test_predicted))
      print("Accuracy: \t\t" + str(dec_tree.score(x_test, y_test)))

      # Desem el model après serialitzat en un fitxer
      p = "files_folder/dec_tree_model.pickle"
      with open(p, "wb") as f:
          pickle.dump(dec_tree, f)
```

```
Predicted classes:      [1 0 0 1 2 0 1 1 0 2 0 0 2 2 1]
Accuracy:               0.8
```

```
[63]: # Eliminem la variable dec_tree (per simular el fet d'acabar l'execució del
      # nostre script d'entrenament o bé la transmissió del fitxer a una altra
      # màquina)
      del dec_tree
```

```
[64]: # Carreguem el model desat
      with open(p, "rb") as f:
```

```

dec_tree_des = pickle.load(f)

# Comprovem que el model desat s'ha recuperat correctament
y_test_predicted_des = dec_tree_des.predict(x_test)
print("Predicted classes: \t" + str(y_test_predicted_des))
print("Accuracy: \t\t" + str(dec_tree_des.score(x_test, y_test)))

```

```

Predicted classes:      [1 0 0 1 2 0 1 1 0 2 0 0 2 2 1]
Accuracy:               0.8

```

Hi ha dos detalls importants a notar en el codi anterior. D'una banda, fixeuvos que per llegir o escriure el resultat d'una deserialització o serialització, fem servir `load` i `dump` (respectivament) en comptes de `loads` i `dumps` (a diferència de l'exemple del diccionari). Les funcions `load` i `dump` permeten llegir i escriure el resultat d'una serialització o deserialització a un fitxer, mentre que `loads` i `dumps` retornen la cadena de bytes. D'altra banda, noteu com és necessari obrir els fitxers en mode binari (rb per llegir o wb per escriure).

## 5.2 4.2- Consideracions sobre la serialització de dades

És important ser conscients de les situacions en les quals és útil fer servir serialització amb `pickle`! Cal tenir en compte que `pickle` només serveix per transferir dades entre programes fets amb Python, que hi poden haver incompatibilitats entre versions, i que no és un mètode segur de transferència de dades. En aquest sentit, hem d'assegurar que només deserialitzem dades de confiança, ja que la deserialització pot provocar execució de codi indesitjat.

Pel que fa a quins tipus de dades podem serialitzar amb Python, en general trobarem que és immediat serialitzar tipus bàsics (enters, floats, cadenes de caràcters, booleans, etc.), tipus compostos de tipus serialitzables (diccionaris, tuples, llistes, conjunts, etc.) i classes i funcions definides en l'àmbit global.

Tenint en compte les característiques que hem descrit, quan serà doncs una bona alternativa fer servir serialització amb `pickle` en comptes de fer servir un format de dades estàndard com ara JSON, XML, o CSV? La taula següent recull l'alternativa a fer servir en funció de les propietats de la situació:

Propietat Pickle	JSON/XML/CSV/Altres
Intercanvi de dades sense confiança entre les parts	x

Propietat Pickle	JSON/XML/CSV/Altres
Compatibilitat amb altres llenguatges de programació (diferents de Python) o altres programes	x
Emmagatzemament a llarg termini (possibles canvis de versió)	x

Propietat	Pickle	JSON/XML/CSV/Altres
Execució	.	x
en difer- ents màquines, que tenen difer- ents ver- sions de les lli- breries que pro- por- cio- nen els tipus de dades emmagatzemats		
Necessitat	.	x
de lec- tura per part d'humans		

---

Propietat Pickle      JSON/XML/CSV/Altres

---

Necessitat x .

d'emmagatzemar

ob-

jectes

de

man-

era

ràp-

ida

(pel

pro-

gra-

mador),

sense

haver

de

de-

cidir

com

rep-

re-

sen-

tar

l'objecte

Emmagatzemament .

tem-

poral

(per

ex-

em-

ple,

*check-*

*points*

en

l'execució

d'un

*script*)

Propietat	Pickle	JSON/XML/CSV/Altres
Necessitat de guardar l'estat del nostre programa (en comptes de conjunts de dades amb certa homogeneïtat)	x	.
Necessitat d'emmagatzemar funcions	x	.

## 6 5.- Interacció amb bases de dades

El [PEP249](#) descriu l'especificació de l'API per accedir a bases de dades des de Python. La majoria dels sistemes gestors de bases de dades (SGBD) més populars segueixen aquesta especificació, de manera que hi ha bastanta similitud en com s'accedeix als diferents motors de bases de dades des de Python.

Així doncs, el procediment general per interactuar amb una base de dades des de Python consta dels passos següents: 1. Importar el mòdul que proveeix de la interfície amb la base de dades (específic per cada SGBD). 2. Crear una connexió amb la base de dades, proveint dades sobre la localització de la base de dades i, si s'escau, l'autenticació. 3. Obtenir un cursor sobre la connexió creada al pas anterior. 4. Executar les sentències SQL desitjades. 5. Si les sentències eren de consulta, recuperar les dades retornades per la base de dades. 6. Tancar la connexió.

Per exemplificar el procés, crearem una nova base de dades SQLite, hi desarem les dades dels personatges de Marvel (que carregarem prèviament del csv com hem fet anteriorment) i farem consultes sobre aquestes dades fent servir SQL.

```
[65]: # Carreguem les dades del csv de Marvel en un dataframe de pandas
data = pd.read_csv(
    "data/marvel-wikia-data.csv", decimal=",", escapechar="\\",
    na_values=["dontknow"],
    converters={"FIRST APPEARANCE": lambda x: int(x.split("-")[1])+1900},
    skiprows=[31])
```

```
# Obtenim una cadena amb la llista de columnes, que farem servir per crear
# la taula de la base de dades
cols = ", ".join(data.columns)
print(cols)
```

page\_id, name, urlslug, ID, ALIGN, EYE, HAIR, SEX, GSM, ALIVE, APPEARANCES,  
FIRST APPEARANCE, Year

```
[66]: # Importem sqlite3
import sqlite3

# Creem una connexió amb la base de dades marvel.db
conn = sqlite3.connect('files_folder/marvel.db')

# Obtenim un cursor sobre la connexió
cur = conn.cursor()

# Creem una taula fent servir els noms de les columnes del dataframe
sql_create_stm = "CREATE TABLE marvel_chars ({})".format(cols)
print(sql_create_stm)

# Executem la sentència SQL de creació de la taula
cur.execute(sql_create_stm)

# Desem els canvis (fem un commit a la db)
conn.commit()
```

```
CREATE TABLE marvel_chars (page_id, name, urlslug, ID, ALIGN, EYE, HAIR, SEX,
GSM, ALIVE, APPEARANCES, FIRST APPEARANCE, Year)
```

Per tal de crear la taula de la base de dades, hem fet servir una sentència SQL de creació de taules (CREATE TABLE).

Una vegada hem creat la taula, podem consultar-ne el seu contingut executant un SELECT:

```
[67]: # Executem SELECT * sobre tota la taula
sql_sel_stm = "SELECT * FROM marvel_chars"
cur.execute(sql_sel_stm)

# Recuperem els resultats
results = cur.fetchall()

# Mostrem els resultats: la taula de la base de dades és buida
print(results)
```

[]

Com que acabem de crear la taula, aquesta està buida, i el SELECT no retorna cap fila. Ara, inserirem les dades del *dataframe* a la base de dades, iterant per cada fila del *dataframe* i inserint cada fila a la taula amb un INSERT:

```
[68]: # Iterem per cada fila de la taula
for index, row in data.iterrows():
    # Inserim les dades a la taula marvel_chars
    cur.execute("INSERT INTO marvel_chars VALUES "
                "(?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)", row)

# Fem un commit dels canvis
conn.commit()
```

Una vegada inserides les dades a la base de dades, podem repetir el SELECT \* que hem realitzat anteriorment, i recuperar així totes les dades que hem inserit a la taula:

```
[69]: # Executem SELECT * sobre tota la taula
sql_sel_stm = "SELECT * FROM marvel_chars"
cur.execute(sql_sel_stm)

# Recuperem els resultats
results = cur.fetchall()

# Mostrem els resultats: la taula de la base de dades conté les dades
# dels personatges de Marvel
print(results)
```

```
[(1678, 'Spider-Man (Peter Parker)', '/Spider-Man_(Peter_Parker)', 'Secret
Identity', 'Good Characters', 'Hazel Eyes', 'Brown Hair', 'Male Characters',
None, 'Living Characters', 4043.0, 1962, '1.962'), (7139, 'Captain America
(Steven Rogers)', '/Captain_America_(Steven_Rogers)', 'Public Identity', 'Good
Characters', 'Blue Eyes', 'White Hair', 'Male Characters', None, 'Living
Characters', 3360.0, 1941, '1.941'), (64786, 'Wolverine (James "Logan"
Howlett)', '/Wolverine_(James_%22Logan%22_Howlett)', 'Public Identity',
'Neutral Characters', 'Blue Eyes', 'Black Hair', 'Male Characters', None,
'Living Characters', 3061.0, 1974, '1.974'), (1868, 'Iron Man (Anthony "Tony"
Stark)', '/Iron_Man_(Anthony_%22Tony%22_Stark)', 'Public Identity', 'Good
Characters', 'Blue Eyes', 'Black Hair', 'Male Characters', None, 'Living
Characters', 2961.0, 1963, '1.963'), (2460, 'Thor (Thor Odinson)',
'/Thor_(Thor_Odinson)', 'No Dual Identity', 'Good Characters', 'Blue Eyes',
'Blond Hair', 'Male Characters', None, 'Living Characters', 2258.0, 1950,
'1.950'), (2458, 'Benjamin Grimm (Earth-616)', '/Benjamin_Grimm_(Earth-616)',
'Public Identity', 'Good Characters', 'Blue Eyes', 'No Hair', 'Male Characters',
None, 'Living Characters', 2255.0, 1961, '1.961'), (2166, 'Reed Richards
(Earth-616)', '/Reed_Richards_(Earth-616)', 'Public Identity', 'Good
Characters', 'Brown Eyes', 'Brown Hair', 'Male Characters', None, 'Living
Characters', 2072.0, 1961, '1.961'), (1833, 'Hulk (Robert Bruce Banner)',
'/Hulk_(Robert_Bruce_Banner)', 'Public Identity', 'Good Characters', 'Brown
Eyes', 'Brown Hair', 'Male Characters', None, 'Living Characters', 2017.0, 1962,
'1.962'), (29481, 'Scott Summers (Earth-616)', '/Scott_Summers_(Earth-616)',
```



'Public Identity', 'Neutral Characters', 'Brown Eyes', 'Brown Hair', 'Male Characters', None, 'Living Characters', 1955.0, 1963, '1.963'), (1837, 'Jonathan Storm (Earth-616)', '/Jonathan\_Storm\_(Earth-616)', 'Public Identity', 'Good Characters', 'Blue Eyes', 'Blond Hair', 'Male Characters', None, 'Living Characters', 1934.0, 1961, '1.961'), (15725, 'Henry McCoy (Earth-616)', '/Henry\_McCoy\_(Earth-616)', 'Public Identity', 'Good Characters', 'Blue Eyes', 'Blue Hair', 'Male Characters', None, 'Living Characters', 1825.0, 1963, '1.963'), (1863, 'Susan Storm (Earth-616)', '/Susan\_Storm\_(Earth-616)', 'Public Identity', 'Good Characters', 'Blue Eyes', 'Blond Hair', 'Female Characters', None, 'Living Characters', 1713.0, 1961, '1.961'), (7823, 'Namor McKenzie (Earth-616)', '/Namor\_McKenzie\_(Earth-616)', 'No Dual Identity', 'Neutral Characters', 'Green Eyes', 'Black Hair', 'Male Characters', None, 'Living Characters', None, 1961, '1.528'), (2614, 'Ororo Munroe (Earth-616)', '/Ororo\_Munroe\_(Earth-616)', 'Public Identity', 'Good Characters', 'Blue Eyes', 'White Hair', 'Female Characters', None, 'Living Characters', 1512.0, 1975, '1.975'), (1803, 'Clinton Barton (Earth-616)', '/Clinton\_Barton\_(Earth-616)', 'Public Identity', 'Good Characters', 'Blue Eyes', 'Blond Hair', 'Male Characters', None, 'Living Characters', 1394.0, 1964, '1.964'), (1396, 'Matthew Murdock (Earth-616)', '/Matthew\_Murdock\_(Earth-616)', 'Public Identity', 'Good Characters', 'Blue Eyes', 'Red Hair', 'Male Characters', None, 'Living Characters', 1338.0, 1964, '1.964'), (55534, 'Stephen Strange (Earth-616)', '/Stephen\_Strange\_(Earth-616)', 'Public Identity', 'Good Characters', 'Grey Eyes', 'Black Hair', 'Male Characters', None, 'Living Characters', 1307.0, 1963, '1.963'), (1978, 'Mary Jane Watson (Earth-616)', '/Mary\_Jane\_Watson\_(Earth-616)', 'No Dual Identity', 'Good Characters', 'Green Eyes', 'Red Hair', 'Female Characters', None, 'Living Characters', 1304.0, 1965, '1.965'), (1872, 'John Jonah Jameson (Earth-616)', '/John\_Jonah\_Jameson\_(Earth-616)', 'No Dual Identity', 'Neutral Characters', 'Blue Eyes', 'Black Hair', 'Male Characters', None, 'Living Characters', 1266.0, 1963, '1.963'), (35350, 'Robert Drake (Earth-616)', '/Robert\_Drake\_(Earth-616)', 'Secret Identity', 'Good Characters', 'Brown Eyes', 'Brown Hair', 'Male Characters', None, 'Living Characters', 1265.0, 1963, '1.963'), (1557, 'Henry Pym (Earth-616)', '/Henry\_Pym\_(Earth-616)', 'Public Identity', 'Good Characters', 'Blue Eyes', 'Blond Hair', 'Male Characters', None, 'Living Characters', 1237.0, 1962, '1.962'), (65255, 'Charles Xavier (Earth-616)', '/Charles\_Xavier\_(Earth-616)', 'Public Identity', 'Good Characters', 'Blue Eyes', 'Bald', 'Male Characters', None, 'Deceased Characters', 1233.0, 1963, '1.963'), (1073, 'Warren Worthington III (Earth-616)', '/Warren\_Worthington\_III\_(Earth-616)', 'Public Identity', 'Good Characters', 'Blue Eyes', 'Blond Hair', 'Male Characters', None, 'Living Characters', 1230.0, 1963, '1.963'), (1346, 'Piotr Rasputin (Earth-616)', '/Piotr\_Rasputin\_(Earth-616)', 'Secret Identity', 'Good Characters', 'Blue Eyes', 'Black Hair', 'Male Characters', None, 'Living Characters', 1162.0, 1975, '1.975'), (2512, 'Wanda Maximoff (Earth-616)', '/Wanda\_Maximoff\_(Earth-616)', 'Public Identity', 'Good Characters', 'Green Eyes', 'Brown Hair', 'Female Characters', None, 'Living Characters', 1161.0, 1964, '1.964'), (1671, 'Nicholas Fury (Earth-616)', '/Nicholas\_Fury\_(Earth-616)', 'No Dual Identity', 'Neutral Characters', 'Brown Eyes', 'Brown Hair', 'Male Characters', None, 'Living

```
Characters', 1137.0, 1963, '1.963'), (1976, 'Janet van Dyne (Earth-616)',
'/Janet_van_Dyne_(Earth-616)', 'Public Identity', 'Good Characters', 'Blue
Eyes', 'Auburn Hair', 'Female Characters', None, 'Living Characters', 1120.0,
1963, '1.963'), (65219, 'Jean Grey (Earth-616)', '/Jean_Grey_(Earth-616)',
'Public Identity', 'Good Characters', 'Green Eyes', 'Red Hair', 'Female
Characters', None, 'Deceased Characters', 1107.0, 1963, '1.963'), (6545,
'Natalia Romanova (Earth-616)', '/Natalia_Romanova_(Earth-616)', 'Public
Identity', 'Good Characters', 'Green Eyes', 'Red Hair', 'Female Characters',
'Bisexual Characters', 'Living Characters', 1050.0, 1964, '1.964'), (2223, 'Kurt
Wagner (Earth-616)', '/Kurt_Wagner_(Earth-616)', 'Secret Identity', 'Good
Characters', 'Yellow Eyes', 'Blue Hair', 'Male Characters', None, 'Living
Characters', 1047.0, 1975, '1.975')]
```

Podem executar consultes SQL sobre la taula que acabem de crear. Així, per exemple, si volem recuperar només el nom dels personatges femenins, podríem fer:

```
[70]: # Executem la consulta SQL
sql_sel_stm = "SELECT name FROM marvel_chars WHERE sex = 'Female Characters'"
cur.execute(sql_sel_stm)

# Mostrem els resultats
results = cur.fetchall()
print(results)
```

```
[('Susan Storm (Earth-616)',), ('Ororo Munroe (Earth-616)',), ('Mary Jane Watson
(Earth-616)',), ('Wanda Maximoff (Earth-616)',), ('Janet van Dyne
(Earth-616)',), ('Jean Grey (Earth-616)',), ('Natalia Romanova (Earth-616)',)]
```

Una vegada hem acabat de treballar amb la base de dades, és important recordar tancar el recurs:

```
[71]: # Tanquem la connexió
conn.close()
```

És important notar que si haguéssim volgut fer el mateix amb una base de dades MySQL en comptes d'SQLite, caldria canviar la llibreria a fer servir (l'import de l'inici del codi), però després el codi seria en general el mateix, i funcionaria fent únicament petites adaptacions (per exemple, a la creació de la connexió).

## 7 6.- Interacció amb el sistema operatiu

Fins ara hem vist com interactuar amb el sistema operatiu a través de la lectura o escriptura de fitxers. En aquesta secció, veurem com podem executar programes externs o comandes a la consola del sistema des del nostre codi Python, fent servir el mòdul `subprocess`.

El mòdul `subprocess` permet executar comandes amb la funció `run`. La funció rep com a argument la comanda i retorna un objecte `CompletedProcess`. Per defecte, no captura la sortida de la comanda executada, només el codi de retorn (tradicionalment, es fa servir el 0 per indicar que el procés ha finalitzat amb èxit). El següent exemple executa una comanda de llistat, 1s:

```
[72]: import subprocess as sp

# Executem ls
exe_out = sp.run(["ls"])

# Mostrem el resultat que retorna run
print(exe_out)
print("The args were: {}".format(exe_out.args))
print("The return code was: {}".format(exe_out.returncode))
```

```
CompletedProcess(args=['ls'], returncode=0)
The args were: ['ls']
The return code was: 0
```

El resultat és un objecte `CompletedProcess` que conté tant els arguments com el codi de retorn. Podem passar arguments addicionals afegint elements a la llista que s'envia com a primer paràmetre:

```
[73]: # Executem ls -l
sp.run(["ls", "-l"])
```

```
[73]: CompletedProcess(args=['ls', '-l'], returncode=0)
```

Per tal de recuperar no només el codi de retorn sinó també la sortida de l'execució de la comanda, farem servir el paràmetre `stdout`, que permet indicar a on enviem la sortida:

```
[74]: # Executem ls -l i capturem la sortida de l'execució
exe_out = sp.run(['ls', '-l'], encoding='utf-8', stdout=sp.PIPE)

# Mostrem el resultat que retorna run
print(exe_out)

print("\nThe args were: {}".format(exe_out.args))
print("\nThe return code was: {}".format(exe_out.returncode))
print("\nThe output was:\n{}".format(exe_out.stdout))
```

```
CompletedProcess(args=['ls', '-l'], returncode=0, stdout='total 340\n-rw-rw-r--
1 datasci datasci 157906 mar 21 10:12 3-CAT-
Fitxers_i_interacció_amb_el_sistema.ipynb\n-rw-rw-r-- 1 datasci datasci 160168
mar 21 10:14 3-ES-Ficheros_e_interacción_con_el_sistema.ipynb\ndrwxrwxr-x 2
datasci datasci 4096 mar 21 10:06 data\n-rw-rw-r-- 1 datasci datasci 17
mar 21 10:14 file_2.txt\ndrwxrwxr-x 4 datasci datasci 4096 mar 21 10:14
files_folder\n-rw-rw-r-- 1 datasci datasci 5103 mar 21 10:01
files_folder.zip\n-rw-rw-r-- 1 datasci datasci 284 mar 21 09:56 README.md\n')
```

```
The args were: ['ls', '-l']
```

```
The return code was: 0
```

```
The output was:
```

```
total 340
-rw-rw-r-- 1 datasci datasci 157906 mar 21 10:12 3-CAT-
Fitxers_i_interacció_amb_el_sistema.ipynb
-rw-rw-r-- 1 datasci datasci 160168 mar 21 10:14 3-ES-
Ficheros_e_interacción_con_el_sistema.ipynb
drwxrwxr-x 2 datasci datasci 4096 mar 21 10:06 data
-rw-rw-r-- 1 datasci datasci 17 mar 21 10:14 file_2.txt
drwxrwxr-x 4 datasci datasci 4096 mar 21 10:14 files_folder
-rw-rw-r-- 1 datasci datasci 5103 mar 21 10:01 files_folder.zip
-rw-rw-r-- 1 datasci datasci 284 mar 21 09:56 README.md
```

Ara, l'objecte `CompletedProcess` conté també un atribut `stdout` amb el resultat de l'execució del procés que, en aquest cas, consisteix en el llistat del directori on s'ha executat.

Podem fer servir `run` també per a executar altres programes, que no siguin comandes del sistema. Per exemple, a la carpeta `files_folder` hi ha un petit script de bash amb el següent codi:

```
#!/bin/bash

echo "Script executed!"
```

És a dir, l'script simplement mostra per pantalla el missatge 'Script executed!' cada vegada que és cridat.

**Abans d'executar la cel·la següent**, donarem permisos d'execució a l'script i provarem el seu funcionament:

1. En primer lloc, obriu una consola del sistema, situeu-vos a la carpeta `files_folder` i executeu la següent comanda: `chmod +x *.sh`

Això donarà permisos d'execució a l'script, ja que per defecte els fitxers no poden executar-se (per qüestions de seguretat).

2. Ara, des de la mateixa consola, executeu l'script: `./echo_script.sh`

Això executarà l'script i mostrarà el missatge 'Script executed!' al terminal.

Veiem doncs com podem executar aquest mateix *script* des del nostre programa Python amb

`run`:

```
[75]: # Executem echo_script i capturem la sortida de l'execució
exe_out = sp.run(["./files_folder/echo_script.sh"],
                  encoding='utf-8', stdout=sp.PIPE)
print(exe_out)
```

```
CompletedProcess(args=['./files_folder/echo_script.sh'], returncode=0,
stdout='Script executed!\n')
```

La funció `run` també permet executar un programa extern i passar-li dades d'entrada. Per exemple, l'script `echo_read_script.sh` té el següent codi:

```
#!/bin/bash

echo "Enter a number"
read number
echo "Your number was $number"
```

És a dir, l'script mostra primer el missatge 'Enter a number', demana a l'usuari que entri un número, i mostra per pantalla el missatge 'Your number was' i el número introduït per l'usuari.

Executem ara l'script `echo_read_script.sh`, passant-li com a entrada el contingut del fitxer `a_number.txt` (que conté el número 42):

```
[76]: # Executem echo_read_script, capturant la sortida de l'execució
# i passant com a entrada el contingut del fitxer a_number
with open("files_folder/a_number.txt", "r") as f:
    exe_out = sp.run(["./files_folder/echo_read_script.sh"],
                     encoding='utf-8',
                     stdin=f,
                     stdout=sp.PIPE)

    # Mostrem el resultat que retorna run
    print(exe_out)
    print("\nThe args were: {}".format(exe_out.args))
    print("The return code was: {}".format(exe_out.returncode))
    print("The output was:\n{}".format(exe_out.stdout))
```

```
CompletedProcess(args=['./files_folder/echo_read_script.sh'], returncode=0,
stdout='Enter a number\nYour number was 42\n')
```

```
The args were: ['./files_folder/echo_read_script.sh']
The return code was: 0
The output was:
Enter a number
Your number was 42
```

Ja per acabar, és important notar que la funció `run` executa el que haguem indicat i després espera (per defecte de manera indefinida) a que finalitzi l'execució de la comanda. Per tant, una crida a `run` pot bloquejar el nostre programa si el programa que executem no acaba mai. Així, per exemple, l'execució de l'script `endless_script.sh` amb el següent codi que conté un bucle infinit:

```
#!/bin/bash

while true
do
    sleep 1
done
```

faria que el nostre programa Python es quedés indefinidament esperant. Per evitar-ho, `run` té un paràmetre `timeout`, que permet especificar el temps màxim (en segons) que volem esperar a l'execució d'un programa extern. Passat aquest temps, si l'execució no ha finalitzat es genera una excepció.

```
[77]: # Executem ls amb timeout (l'execució finalitza sense
# generar cap excepció)
try:
    exe_out = sp.run(["ls"], timeout=3)
except sp.TimeoutExpired as e:
    print(e)
```

```
[78]: # Executem l'script endless_script amb timeout
# (l'execució finalitza amb un timeout als 3 segons)
try:
    exe_out = sp.run("./files_folder/endless_script.sh",
                     timeout=3)
except sp.TimeoutExpired as e:
    print(e)
```

Command '['./files\_folder/endless\_script.sh']' timed out after 3 seconds

Hem vist doncs com la funció `run` és bastant versàtil, i ens permet executar programes externs des del nostre codi Python.

## 8 7.- Exercicis per practicar

A continuació hi trobareu un conjunt de problemes que us poden servir per a practicar els conceptes explicats en aquesta unitat. Us recomanem que intenteu fer aquests problemes vosaltres mateixos i que, una vegada realitzats, compareu la solució que us proposem amb la vostra solució. No dubteu en adreçar tots els dubtes que sorgeixin de la resolució d'aquests exercicis o bé de les solucions proposades al fòrum de l'aula.

1. Creeu un codi que permeti monitoritzar el consum de memòria RAM de la màquina en què s'executi. El codi desarà les dades de la memòria total i utilitzada del sistema durant un període de temps, capturant les dades en intervals periòdics.

Aquestes dades es desaran en fitxers de text pla, fent servir un fitxer per a les dades capturades en cada moment. Així, dins de la carpeta de dades, hi haurà una carpeta per a les dades de cada dia (que tindrà per nom l'any, el mes i el dia, escrits tots seguits, per exemple, 20200318). Dins de la carpeta de cada dia, hi haurà un fitxer per cada instant de temps on s'hagin obtingut dades (que tindrà per nom l'hora, el minut i el segon, separats per guions baixos, per exemple, 14\_45\_55). El contingut del fitxer seran els dos valors (memòria total i utilitzada) separats per comes (per exemple, 15571, 4242).

Creeu també el codi que permeti recuperar totes les dades emmagatzemades, i obtenir-ne una descripció estadística bàsica (mitjana, mediana i desviació estàndard).

Per fer-ho, implementarem un seguit de funcions que es detallen a continuació.

1.1 Creeu una funció que rebí com a paràmetre el nom d'una carpeta (que serà `mem_data` per defecte) i creï les carpetes necessàries per emmagatzemar dades per al dia actual. És a dir, el codi haurà de crear, si no existeix ja, una carpeta de dades amb el nom que ha rebut com a paràmetre (o fer servir `mem_data` si no s'ha especificat cap nom), i una altra carpeta a dins d'aquesta que tingui per nom el dia actual (en el format any de 4 xifres, mes de 2 xifres, dia de 2 xifres, tot seguit sense separadors, per exemple, 20200318).

[79]: # Resposta

1.2 Implementeu una funció que rebi com a paràmetre el *path* amb la carpeta de la data actual (que s'ha creat a l'apartat anterior) i hi escrigui un fitxer amb les dades de consum de memòria del sistema actuals. El fitxer ha de tenir per nom l'hora actual (en el format *hora\_minut\_segons*, amb els ítems separats per guions baixos, per exemple, *14\_45\_55*). El contingut del fitxer seran els dos valors (memòria total i utilitzada) en megabytes separats per comes (per exemple, *15571, 4242*).

Per obtenir les dades del consum de memòria, recordeu que podeu executar comandes del sistema amb el mòdul *subprocess* (segurament us caldrà buscar informació sobre com obtenir aquestes dades amb comandes de *unix*).

[80]: # Resposta

1.3 Implementeu una funció que rebi com a paràmetres el número de mostres a capturar i l'interval de temps entre cadascuna de les mostres (en segons), i capturi les dades del consum de memòria tantes vegades com s'hagi especificat, esperant el temps indicat entre captures. La funció farà ús de les dues funcions definides anteriorment.

[81]: # Resposta

1.4 Crideu a la funció definida a l'apartat 1.3 i captureu 20 mostres de consum de memòria, fent servir un interval de 3 segons entre cada captura.

[82]: # Resposta

1.5 Implementeu una funció que llegeixi totes les dades que s'han capturat, emmagatzemades en una carpeta que rebrà com a paràmetre (i que, de nou, prendrà com a valor per defecte *mem\_data*), i mostri les següents dades: \* El número de mostres llegides. \* La mitjana de la memòria total i utilitzada. \* La mediana de la memòria total i utilitzada. \* La desviació estàndard de la memòria total i utilitzada. \* La data i hora de la primera i última captures de què en tenim dades.

Crideu a la funció anterior per tal d'obtenir un resum de les dades capturades.

[83]: # Resposta

1.6 Implementeu una funció que crei un fitxer comprimit amb totes les dades emmagatzemades per cada dia. La funció rebrà com a argument el nom de la carpeta de dades (per defecte, *mem\_data*) i crearà tants fitxers comprimits com dies dels quals en disposem dades. Cada fitxer comprimit contindrà tots els fitxers de dades d'aquell dia.

Crideu a la funció anterior i comproveu que es generen els fitxers comprimits correctament.

[84]: # Resposta

## 8.1 7.1.- Solucions als exercicis per practicar

1. Creeu un codi que permeti monitoritzar el consum de memòria RAM de la màquina en què s'executi. El codi desarà les dades de la memòria total i utilitzada del sistema durant un període de temps, capturant les dades en intervals periòdics.

Aquestes dades es desaran en fitxers de text pla, fent servir un fitxer per a les dades capturades en cada moment. Així, dins de la carpeta de dades, hi haurà una carpeta per a les dades de cada dia (que tindrà per nom l'any, el mes i el dia, escrits tots seguits, per exemple, *20200318*). Dins de la carpeta de cada dia, hi haurà un fitxer per cada instant de temps on s'hagin obtingut dades (que tindrà per nom l'hora, el minut i el segon, separats per guions baixos, per exemple, *14\_45\_55*).

El contingut del fitxer seran els dos valors (memòria total i utilitzada) separats per comes (per exemple, 15571, 4242).

Creeu també el codi que permeti recuperar totes les dades emmagatzemades, i obtenir-ne una descripció estadística bàsica (mitjana, mediana i desviació estàndard).

Per fer-ho, implementarem un seguit de funcions que es detallen a continuació.

1.1 Creeu una funció que rebi com a paràmetre el nom d'una carpeta (que serà `mem_data` per defecte) i creï les carpetes necessàries per emmagatzemar dades per al dia actual. És a dir, el codi haurà de crear, si no existeix ja, una carpeta de dades amb el nom que ha rebut com a paràmetre (o fer servir `mem_data` si no s'ha especificat cap nom), i una altra carpeta a dins d'aquesta que tingui per nom el dia actual (en el format any de 4 xifres, mes de 2 xifres, dia de 2 xifres, tot seguit sense separadors, per exemple, 20200318).

```
[85]: import datetime

def create_folder(data_folder='mem_data'):
    today = datetime.datetime.now().strftime("%Y%m%d")
    full_path = os.path.join(data_folder, today, "")
    # Creem tant la carpeta data_folder com la subcarpeta amb el dia
    # i evitem obtenir errors si ja existeixen
    os.makedirs(full_path, exist_ok=True)
    return full_path
```

1.2 Implementeu una funció que rebi com a paràmetre el *path* amb la carpeta de la data actual (que s'ha creat a l'apartat anterior) i hi escrigui un fitxer amb les dades de consum de memòria del sistema actuals. El fitxer ha de tenir per nom l'hora actual (en el format hora\_minut\_segons, amb els ítems separats per guions baixos, per exemple, 14\_45\_55). El contingut del fitxer seran els dos valors (memòria total i utilitzada) en megabytes separats per comes (per exemple, 15571, 4242).

Per obtenir les dades del consum de memòria, recordeu que podeu executar comandes del sistema amb el mòdul `subprocess` (segurament us caldrà buscar informació sobre com obtenir aquestes dades amb comandes de *unix*).

```
[86]: def save_current_data(folder_name):
    # Construïm el path del fitxer a escriure
    file_name = datetime.datetime.now().strftime("%H_%M_%S")
    full_path = os.path.join(folder_name, file_name)
    with open(full_path, 'w') as f:
        # Recuperem les dades de memòria executant free -tm i quedant-nos
        # únicament amb les dades de memòria RAM (línia 1), i formatem
        # la sortida com demana l'enunciat (amb els valors separats per comes)
        exe_out = sp.run(['free', '-tm'], encoding='utf-8', stdout=sp.PIPE)
        text = ", ".join(exe_out.stdout.split("\n")[1].split()[1:3])
        # Escrivim el resultat al fitxer
        f.write(text)
```

1.3 Implementeu una funció que rebi com a paràmetres el número de mostres a capturar i l'interval de temps entre cadascuna de les mostres (en segons), i capturi les dades del consum de memòria tantes vegades com s'hagi especificat, esperant el temps indicat entre captures. La funció farà ús de les dues funcions definides anteriorment.



```
[87]: import time
```

```
def store_data(num_samples, interval):  
    full_path = create_folder()  
    for _ in range(num_samples):  
        save_current_data(full_path)  
        time.sleep(interval)
```

1.4 Crideu a la funció definida a l'apartat 1.3 i captureu 20 mostres de consum de memòria, fent servir un interval de 3 segons entre cada captura.

```
[88]: store_data(num_samples=20, interval=3)
```

1.5 Implementeu una funció que llegeixi totes les dades que s'han capturat, emmagatzemades en una carpeta que rebrà com a paràmetre (i que, de nou, prendrà com a valor per defecte `mem_data`), i mostri les següents dades: \* El número de mostres llegides. \* La mitjana de la memòria total i utilitzada. \* La mediana de la memòria total i utilitzada. \* La desviació estàndard de la memòria total i utilitzada. \* La data i hora de la primera i última captures de què en tenim dades.

Crideu a la funció anterior per tal d'obtenir un resum de les dades capturades.

```
[89]: import numpy as np
```

```
def read_data(data_folder='mem_data'):  
    # Recuperem tots els fitxers de dades  
    all_files = glob.glob(data_folder + '/*/*', recursive=True)  
    # Carreguem les dades de tots els fitxers en llistes  
    # (fem servir una llista per cada tipus de dades a desar)  
    total_mem, used_mem, dts = [], [], []  
    for file_name in all_files:  
        with open(file_name) as f:  
            # Llegim la línia del fitxer  
            line = f.readline()  
            # Separem els dos valors (memòria total i memòria utilitzada)  
            t_mem, u_mem = line.split(",")  
            # Afegim les dades a les llistes  
            total_mem.append(int(t_mem))  
            used_mem.append(int(u_mem))  
            # Afegim també la data i hora del fitxer a la llista dts  
            dt = datetime.datetime.strptime(file_name[-17:], "%Y%m%d/%H_%M_%S")  
            dts.append(dt)  
  
    # Mostrem el resum de les dades llegides  
    print("{} samples read\n".format(len(total_mem)))  
    print("Total memory:\t\t{} (avg), {} (median), {} std".format(  
        np.mean(total_mem), np.median(total_mem), np.std(total_mem)))  
    print("Used memory:\t\t{} (avg), {} (median), {} std".format(  
        np.mean(used_mem), np.median(used_mem), np.std(used_mem)))
```

```

        np.mean(used_mem), np.median(used_mem), np.std(used_mem)))
    print("First sample is from:\t{}".format(min(dts)))
    print("Last sample is from:\t{}".format(max(dts)))

read_data()

```

20 samples read

```

Total memory:          1993.0 (avg), 1993.0 (median), 0.0 std
Used memory:           1202.2 (avg), 1202.0 (median), 0.4 std
First sample is from:   2020-03-21 10:15:00
Last sample is from:    2020-03-21 10:15:58

```

1.6 Implementeu una funció que crei un fitxer comprimit amb totes les dades emmagatzemades per cada dia. La funció rebrà com a argument el nom de la carpeta de dades (per defecte, `mem_data`) i crearà tants fitxers comprimits com dies dels quals en disposem dades. Cada fitxer comprimit contindrà tots els fitxers de dades d'aquell dia.

Crideu a la funció anterior i comproveu que es generen els fitxers comprimits correctament.

```

[90]: def compress_data(data_folder='mem_data'):

    date_folders = glob.glob(data_folder + '/*')
    for date_folder in date_folders:
        # Creem el fitxer .zip
        zip_file = os.path.join(date_folder + '.zip')
        # Omplim el fitxer .zip amb tots els fitxers que hi ha a la carpeta
        # date_folder que estem processant
        with zf.ZipFile(zip_file, 'w', compression=zf.ZIP_DEFLATED) as zip_f:
            zip_f.write(date_folder)
            sample_files = glob.glob(date_folder + '/*')
            for sample_file in sample_files:
                zip_f.write(sample_file)

compress_data()

```

## 9 8.- Bibliografia

### 9.1 8.1.- Bibliografia bàsica

La codificació és un dels detalls importants a considerar quan cal llegir i/o escriure un fitxer i, sovint, és l'origen de mal de caps en molts programadors (sobretot en llenguatges de més baix nivell que el Python). Per a entendre què és la codificació de caràcters, conèixer quines són les codificacions de caràcters més habituals i saber com gestiona Python 3 la codificació, llegiu ara la [guia d'aquest enllaç](#).

## 9.2 8.2.- Bibliografia addicional - Ampliació de coneixements

Aquesta unitat presenta una introducció a com interactuar amb el sistema de fitxers i, en general, amb el sistema operatiu, des de Python. Així, com a introducció, presenta algunes qüestions de manera inicial i obre la porta a explorar-les en més profunditat. A continuació es llisten alguns enllaços que us serviran per seguir explorant alguns dels temes que treballem a la unitat, ja siguin purament de programació en Python com del sistema operatiu:

- **El sistema de fitxers de Linux:** En la unitat parlem d'interactuar amb el sistema de fitxers de Linux, però no entrem a explicar com és aquest sistema de fitxers. Si voleu llegir-ne una introducció, aquest [Overview](#) us pot ser molt útil.
- **Permisos sobre els fitxers en unix:** Si teniu curiositat per saber com funcionen els bits de permís dels fitxers en unix, us recomanem llegir les tres parts de la sèrie d'articles sobre els permisos ([1](#), [2](#), i [3](#)).
- **Obertura de fitxers des de Python:** La funció `open` accepta altres arguments opcionals que no hem presentat, i que gestionen detalls com el *buffering* de dades, la codificació, la gestió dels errors, la gestió del salt de línia, etc. El lector interessat pot consultar la [documentació oficial de la funció open](#) per descobrir com funcionen aquests arguments i quines opcions es troben disponibles.
- **Compressió de fitxers:** Hi ha altres formats de compressió de dades a part dels que hem vist en aquesta unitat. El lector interessat pot llegir la documentació del mòdul [gzip](#) per conèixer les funcions que permeten treballar amb fitxers gzip des de Python.
- **Lectura de fitxers amb pandas:** Més enllà dels fitxers csv, hi ha altres formats que també es fan servir sovint per intercanviar o desar dades. Pandas disposa de diverses funcions per a carregar dades provinents dels formats de dades més populars, com ara, json ([read\\_json](#)) o excel ([read\\_excel](#)).

També us recomanem revisar la documentació oficial de les funcions i classes descrites en aquesta unitat, que trobareu enllaçades en cada un dels apartats que les descriuen, per conèixer quins paràmetres permeten ajustar-ne el seu funcionament.