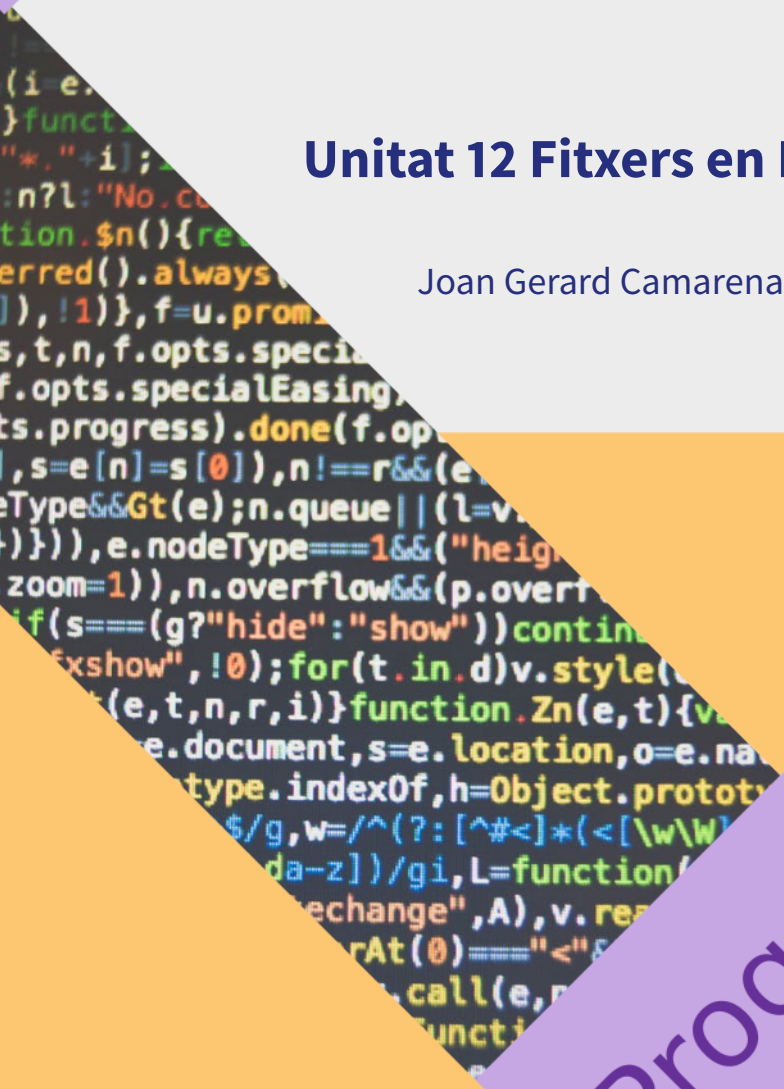
Curs 2019-2020

Unitat 12 Fitxers en Python

Joan Gerard Camarena Estruch



Programació

IES Jaume II El Just
Tavernes de la Valldigna

Continguts

https://www.guru99.com/reading-and-writing-files-in-python.html https://www.geeksforgeeks.org/file-objects-python/ https://www.tutorialspoint.com/python3/python_files_io.htm

1 Sistema de fitxers

El primer pas per a treballar amb fitxers, sigui en el llenguatge que sigui és el accés al sistema de fitxers, per a recòrrer l'arbre de directoris, estudiar les propietats dels fitxers (tamany, permisos, etc). L'equivalent a la classe `File` de Java. En Python qui ens proporciona totes aquestes funcions de llibreria és el nòdul `os` i `os.path`:

- `os.getcwd()` → aquesta funció ens retorna el *current working directory*, és a dir, la carpeta des de la que hem executat el programa. No te perquè coincidir en la carpeta on està el programa o el fitxer *.py* que estem executant.
- `__file__` → aquesta constant simbòlica fa referència al fitxer actual que estem executant, mostrant-ho com una ruta completa
- `os.path.realpath(__file__)` → molt semblant a l'anterior, però ens dona la ruta real (desfent link simbòlics)
- `path, filename = os.path.split(full_path)` → ens separa la carpeta i el fitxer que estem executant
- `os.path.dirname(full_path)` → si volem obtenir sols la carpeta.

Imaginem ara que volem comprovar diverses coses sobre una ruta absoluta: pertany a un fitxer o directori, etc. Per això tenim la variable `ruta="/users/joange/proves.txt"`

- `os.sep` → retorna el separador de carpetes. `\` en Windows o `/` en sistemes basats en Unix.
- `os.path.exists(ruta)` → ens indica si existeix eixe element o no (independentment del que sigui).
- `os.path.isdir(ruta)` → `True` si la ruta és un directori i `False` en cas contrari.
- `os.path.isfile(ruta)` → `True` si la ruta és un fitxer i `False` en cas contrari.
- `os.path.join(ruta,subcarpeta)` → serveix per a combinar noves rutes (accedir a una subcarpeta dins d'una ruta) sense preocupar-nos de la barra o contrabarra. Podriem fer-ho de manera manual amb `ruta + os.sep + subcarpeta`.
- `os.path.getsize(ruta)` → retorna el tamany en bytes de l'objecte ruta.
- `os.access(ruta,permis)` → ens diu si tenim o no el permís que li preguntem. Aquesta funció ho mostra. Hi han molts més permisos que podem comprovar

```
1 def mostrarPermisos(ruta):
2     print("Existeix  ", ruta,":",os.access(ruta,os.F_OK))
3     print("Permís de lectura ", ruta,":",os.access(ruta,os.R_OK))
4     print("Permís d'escriptura ", ruta,":",os.access(ruta,os.W_OK))
```

```
5       print("Permís d'execució ", ruta,":",os.access(ruta,os.X_OK))
```

- `os.listdir(ruta)` → retorna el llistat d'elements que conté una carpeta, com un vector de string. Vigilar que `ruta` no sigui un fitxer, ja que sinó donarà error.
- `os.chdir(nova_ruta)` → canvia la carpeta de treball a la `nova_ruta`.
- `os.walk(ruta)` → retorna un llistat amb tots els elements d'una carpeta i subcarpetes, per a fer un recorregut recursiu. Cada element del llistat és una tupla que conté `<nomDeLaCarpeta>` , `<LlistaDeCarpetes>` , `<LlistaDeFitxer>`.

Podeu consultar tota la llibreria completa (apartat del sistema de fitxers) `os` a https://docs.python.org/3/library/os.html#files-and-directories

2. Llegir i escriure en fitxers de text

Un cop vist com podem menejar-se pel sistema d'arxius, anem ja a obrir fitxers per a manipular el seu contingut.

2.1 Modes d'accés

El primer que hem de fer per a accedir a un fitxer és obrir-lo. Anem a suposar que ja sabem que existeix i que tenim permisos, tot això comprobable amb les eines vistes al punt 1.

Per a obrir un fitxer es fa amb la funció `open(fitxer,modeAcces)`, sent els dos arguments string.

- El `fitxer` és la ruta absoluta o relativa al fitxer al que volem accedir:
  - absoluta → ruta completa des de l'arrel del sistema (`/home/usuari/fitxer.txt`)
  - relatica → sols el nom del fitxer, que haurà d'estar en la carpeta actual d'execució (`os.getcwd()`)
- El `modeAcces` indica el que anem a fer amb el fitxer, i pot ser una combinació de lletres tal i com es veu:
  - `r` → Obri el fitxer existent per a lectura. Situa el cursor al principi del fitxer.
  - `r+` → Obri el fitxer existent per a lectura i escriptura. Situa el cursor al principi del fitxer.
  - `w` → Obri el fitxer per a escriptura. Si existeix el matxaca, sinó el crea. Situa el cursor al principi del fitxer.
  - `w+` →: Obri el fitxer per a escriptura i lectura. Si existeix el matxaca, sinó el crea. Situa el cursor al principi del fitxer.

  - `a` → Obri el fitxer per a escriptura. Si no existeix el crea. Situa el cursor al **final** del fitxer. No el matxaca
  - `a+`→ Obri el fitxer per a escriptura i lectura. Si no existeix el crea. Situa el cursor al **final** del fitxer. No el matxaca

- A qualsevol mode d'accés podem afegir una `b` per a accedir als fitxers en mode binari, ja que per defecte l'apertura és en mode *text*.

2.2 Llegir i escriure

`read()`

Suposem un fitxer ja obert. Per a llegir disposem de la funció `f.read([tamany])`. Aquesta funció llig tants caràcters com indiquem en el tamany. Cas de no posar-lo, llegirà fins arribar al `EOF`, constant simbòlica de *End Of File*. Aquesta funció retorna **`null`** si no pot llegir-se o s'ha arribat al final. Exemples:

```
1 f=open("quijote.txt,"r")
2 text=f.read(1)          # llig una lletra
3
4 text=f.read(10)         # llig 10 lletres
5
6 text=f.read()           # llig fins al final del fitxer. Tot. Linies
     incloses
7
8 f.close()
```

`readline()`

Suposem un fitxer obert. Si llegim amb `readline([tamamy])` ens llegirà una línia del fitxer o fins arribar al `EOF`. Si posem el *tamamy* ens llegirà eixa quantitat de caracters, que pot contenir un `\n` al seu interior, amb la qual cosa ens deixaria linies a mitges (no sol fer-se servir, a menys que ho vullguem en blocs).

> **NOTA**. A diferència de Java, la linia que llegim inclou el `\n`.

`readlines()`

Llig totes les linies de un fitxer retornat-les en un arrray (una linia per casella)

apertura amb `with`

És important que hem de recordar de tancar el fitxer sempre (`f.close()`) quan acabem. Python ha incorporat una instruciió que ens permet obrir el fitxer i el tanca automàticament, per la qual cosa podem prevenir error. Vegem l'exemple:

Algorisme habitual de treball:

```
f=open("quijote.txt","r")   #obrim

# llegir informació
# manipular incormació

f.close()                    # tanquem
```

Amb `with`:

```
with open("quijote.txt","r") as f:
    # llegir informació
    # manipular incormació

# fora del bloc with el fitxer ja s'ha tancat
```

Amb tractament d'errors.

```
with open("/etc/passwd", "a") as (f, err):
    if err:
        print "IOError:", err
    else:
        # manipular fitxer
```

`write()`

Per a l'escriptura de dades al fitxer de text, disposem de la funció `write()`, totalment sobrecarregada. Vegem les opcions mitjançant aquest exemple:

```
with open(fitxer,"w") as f:
    f.write("Hola\n")
    f.writelines("proves"+"\n")
    numeros=[3,6,8,4,6]
    f.write(str(numeros)+"\n")
    f.write(str(True)+"\n")
    f.write(str(3.1415)+"\n")

    # opció molt interessant
    f.write("m=%d i x=%2.2f i text=%s\n"%(2,2.0,"2.0"))
```

El fitxer generat és:

```
Hola
proves
[3, 6, 8, 4, 6]
True
3.1415
m=2 i x=2.00 i text=2.0
```

> NOTA: Fixar-se que ha de ser tot `str`, i hi ha que afegir el bot de linia

2.3 Posicionament

En Python, a l'igual que en la majoria de llenguatges, l'accés és seqüencial, i el cursor conforme anem llegint va avançant pel fitxer. Si volem saber en quina posició estem del fitxer, que també equivaldrà a la quantitat de bytes que em llegit, podem fer servir la funció `file.tell()`. Si volem menejar el cursor, farem servir `seek(quantitat,[posInicial])`. D'aquesta manera avancem el cursor *quantitat* bytes a partir d'on està el cursor, o be a partir de la posició *posInicial* cas de posar-ho. Atenció: - `posInicial` val 0, llavors es mou a partir de l'inici del fitxe. `quantitat` ha de ser positiu - `posInicial` val 1, llavors es mou a partir de la posició actual del fitxer. *Sols en fitxers binaris*. - `posInicial` val 2, llavors es mou a partir de la posició final del fitxer. *Sols en fitxers binaris*. `quantitat` ha de ser negatiu.

```
with open(fitxer,"rb") as f:
    linea=f.readline()
    print(linea,len(linea))
    print("El fitxer està en la posicó",f.tell())

    f.seek(10,1)      # avancem 10 bytes
    linea=f.readline()
    f.seek(0,0)     # al principi del fitxer
```

3. Guardant dades binàries. Empaquetat. `struct`

Per a guardar dades binàries és molt *fàcil* (no, és broma). Sols hem d'obrir el fitxer, vigilant que tinga la `b` en el mode d'apertura, i escriure les dades en binari. Exemple:

```
f=open(fitxer,"wb")
f.write(b'\x01\x0a')
f.close
```

El problema és que hem de posar el `str` de write com a dades en binari , la qual cosa com sembla evident no és massa clarificador.

Per a procedir el que farem és fer servir el modul `struct` que ens permet trasnformar de qualssevol tipus bàsics a bytes i desfer-ho també. El problema és que hem de conèixer el tamany de cada tipus de dades així com el tipus de representació interna de la nostra màquina.

Representació de les dades: endian, litle, etc. https://es.wikipedia.org/wiki/Endianness

Formats i tamanys de les dades que guardarem, per al format:

Format	C Type	Python type	Standard size
`x`	pad byte	no value	
`c`	`char`	bytes of length 1	1
`b`	`signed char`	integer	1
`B`	`unsigned char`	integer	1
`?`	`_Bool`	bool	1
`h`	`short`	integer	2
`H`	`unsigned short`	integer	2
`i`	`int`	integer	4
`I`	`unsigned int`	integer	4
`l`	`long`	integer	4
`L`	`unsigned long`	integer	4
`q`	`long long`	integer	8
`Q`	`unsigned long long`	integer	8
`n`	`ssize_t`	integer	
`N`	`size_t`	integer	
`e`	(6)	float	2
`f`	`float`	float	4
`d`	`double`	float	8
`s`	`char[]`	bytes	
`p`	`char[]`	bytes	
`P`	`void *`	integer	

Sintaxi:

- `struct.pack(format,variables,..)` → empaqueta les *variables* passades com arguments d'acord a la cadena de *format* segons la taula anterior. La sortida de pack és el que escriurem al fitxer binari
- `struct.unpack(format,bytes)` → retorna una tupla amb les variables que estaven contingudes en els *bytes* atesos al *format* indicat. Els *bytes* que passem és el que llegim del fitxer
- `struct.calcsize(format)` → ens retorna els bytes necessaris per a guardar dir *format* Exemple explicat:

Més informació a https://docs.python.org/3/library/struct.html

```
f=open(fitxer,"wb")
numero=10
real=5.345
text="Hola"

f.write(struct.pack("i",numero))  # convertim i escrivim un int
f.write(struct.pack("f",real))    # convertim i escrivim un float
f.write(struct.pack("i",len(text))) # guardem el tamany de la cadena
# guardem ara tants Bytes com te la cadena convertida en bytes
f.write(struct.pack("%dB"%len(text),*text.encode()))
f.close

f=open(fitxer,"rb")

# anem llegint un a un. Fixar-se que
#  - llegim tants bytes com el tamany del tipus
#  - ho desempaquetem segons el tipus
#  - ens quedem en l'element 0 de la tupla
numero=struct.unpack("i",f.read(struct.calcsize("i")))[0]
real=struct.unpack("f",f.read(struct.calcsize("f")))[0]
tam=struct.unpack("i",f.read(struct.calcsize("i")))[0]
text=struct.unpack("%ds"%tam,f.read(tam))[0].decode()

print("numero=",numero,"real=",real,"text=",text)
f.close()
```

4. Guardant objectes. Serialització. `pickle`

Com que el que hem vist anteriorment funciona, però és una mica costòs, ens apareix el mòdul `pickle` per a facilitar-nos la feina. Aquest mòdul ens permet guardar de manera senzilla tipus bàsic com objectes, mitjançant una serialitació dels mateixos. Així doncs, `pickle` ho tracta tot com a objectes, i ja que en `Python` tot són objectes, podem fer-lo servir per a guardar tot el que considerem a un arxiu.

Exemple:

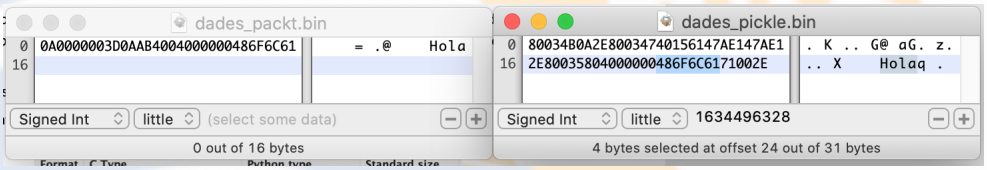
```
f=open(fitxer,"wb")
numero=10
real=5.345
text="Hola"
                # guardem les dades
pickle.dump(numero,f)
pickle.dump(real,f)
pickle.dump(text,f)
f.close()
```

```
10
11                  # recuperem les dades
12  f=open(fitxer,"rb")
13  numero=pickle.load(f)
14  real=pickle.load(f)
15  text=pickle.load(f)
16
17  print("numero=",numero,"real=",real,"text=",text)
18  f.close()
```

Bàsicament te dos mètodes:

- `pickle.dump(objecte,fixter)` → serializa l'objecte i l'escriu al fitxer
- `objecte=pickle.load(fitxer)` → llig del fitxer l'objecte que te a continuació i el guarda a l'objecte

Com és evident, ha d'afegit metainformació per saber els tipus d'objectes que està guardant, i aiò incrementarà el tamany del fitxer, com es veu a continuació:



Exemple que afig objectes a un fitxer i després el recupera:

```
1   f=open(fitxer,"ab")
2   for p in array:        # Array ple d'objectes
3       pickle.dump(p,f)
4
5   f.close()
6
7   f=open(fitxer,"rb")
8   i=0   # contador
9   while True:
10      try:
11          p=pickle.load(f)  # llegim objecte
12      except EOFError:      # quan no podem llegir acabem
13          break
14      i+=1
15      print(p)
16
17  print("S'han llegit",i,"objectes")
18  f.close()
```

5. Investigació. Serialitzant amb `json`

Vos deixe aquest apartat per a completar, donat que es treballarà en futures assignatures