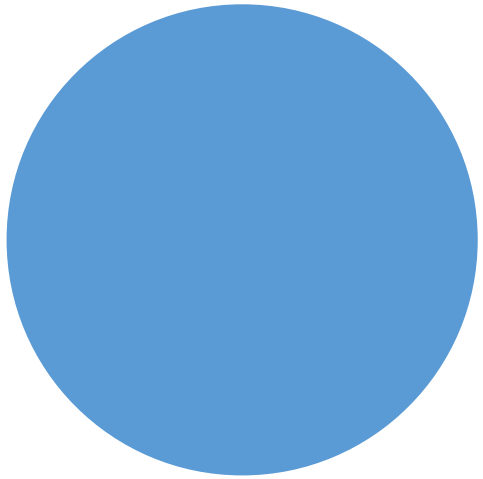
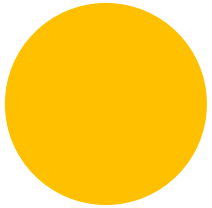



Programare 1

Fisiere

Cursul 9

Despre ce am discutat în cursul precedent?

- Testare
- Depanare
- Excepții
- Aserțiuni

Despre ce o să discutăm astăzi?

Fisiere

- Operatii

Tipuri de fisiere

- Text
- JSON
- CSV
- binare

Definitie

• Conceptual un fișier reprezintă o secvență de date stocate în memoria secundară (în mod tipic un mediu fizic: disc magnetic, SSD, etc)
• Pot să conțină orice tip de date dar este mai ușor să lucrăm cu fișiere care conțin text
    - Fișierele text au avantajul că pot fi citite și interpretate de către oameni, facilitând astfel dezvoltarea și depanarea

•Totodată fișierele ne asigură persistență (atât timp cât suportul fizic ne oferă acest lucru)
• De asemenea acestea ne oferă posibilitatea de a lucra cu cantități mai mari de date, nefiind dependenți de dimensiunea memorie principale

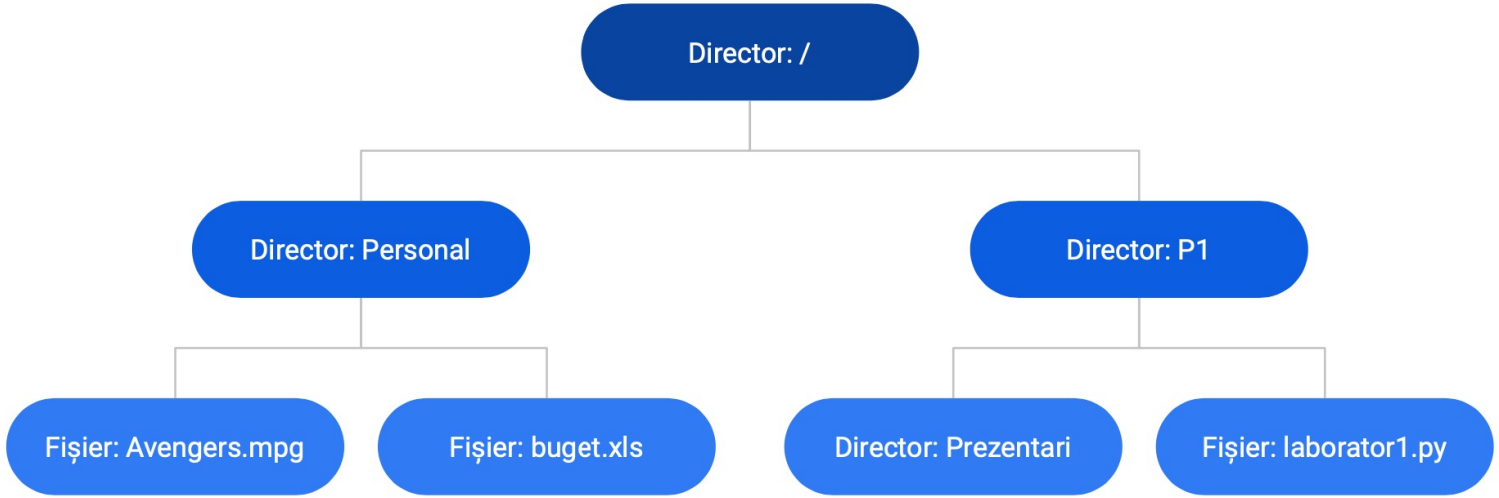
Ce este un sistem de fisiere?

- Suportul fizic ne oferă, de obicei, prin intermediul sistemului de operare o abstractizare liniară, sub formă de blocuri:

- Această abstractizare este formată dintr-o secvență liniară de octeți, fără nici o altă „organizare” expusă explicit
- Sistemul de operare, prin intermediul sistemului de fișiere, este cel care ne ajută să vedem o structură în cadrul acestei secvențe de octeți

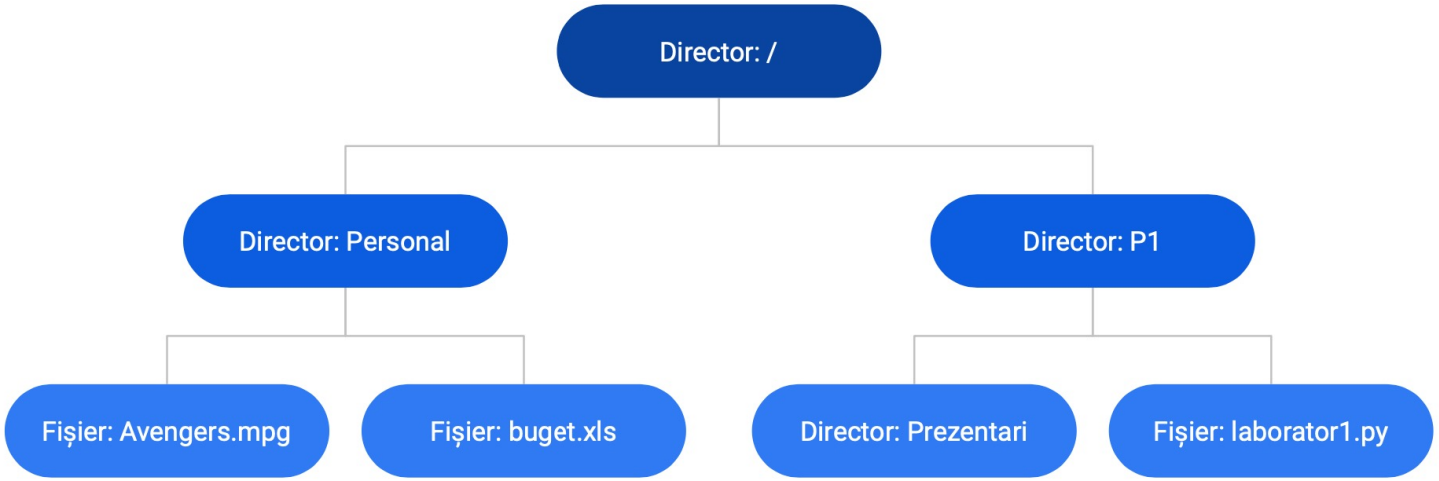
Sistem de fisiere = grupare logica

- O structură ierarhică ce ne permite adresarea și organizarea fiecărui fișier (o grupare logică a datelor)
- Gestionează distribuția datelor corespunzătoare fișierelor pe suportul fizic
  - datele fișierelor nu sunt neapărat secvențiale pe suport dar sistemul de fișiere de asigură „o vedere” secvențială asupra acestora



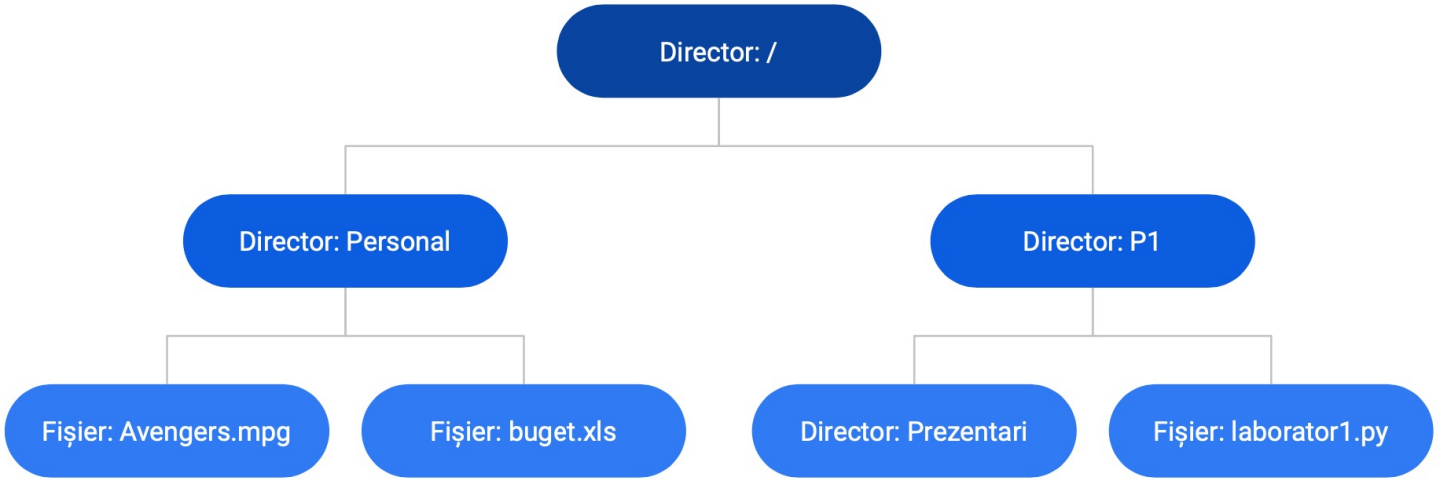
Sistem de fisiere = grupare logica

- Această structură este expusă aplicațiilor de către sistemul de operare prin intermediul VFS (Virtual File System)
- Ne ascunde detaliile de implementare și ne permite să ne concentrăm asupra fișierelor și conținutul acestora



Sistem de fisiere = grupare logica

- Elementele de bază sunt:
  - fișierele = uzual o unitate atomică: nu avem diviziuni din perspectiva sistemului de fișiere
  - directoarele = o colecție de fișiere și directoare
- Ne oferă operații pentru gestiunea acestora



Operații oferite de sistemul de fișiere

- Gestiunea structurii:Reprezentată de operații de gestiune a obiectelor (fișiere sau directoare)
  - Creare: (creare fișier), (creare director)
  - Ștergere: / (ștergere obiect)
  - Redenumire/Mutare: (redenumire obiect)
- Obiectele sunt identificate într-un sistem de fișiere prin intermediul unui nume și căi către acesta în interiorul sistemului de fișiere.
- De exemplu:
  - ’/Users/userul_meu/Cursuri/Curs9.ppt’
- Pe sistemele moderne caracterul / separă căile. Pe Windows este folosit \

Operații primitive pe fișiere

creare (create): reprezintă operația de creare efectivă a unui fișier, alocarea resurselor necesare în cadrul sistemului de fișiere

deschidere (open): reprezintă operația de deschidere a unui fișier și asocierea unui identificator logic (denumit și „file handle”)

citire (read): reprezintă operația de transfer din fișier în memoria principală a unui unități de date

scriere (write): reprezintă operația de transfer de date din memoria principală în fișier

căutare/deplasare (seek): permite determinarea poziției din fișier unde sunt citite/scrise datele

închidere (close/save): presupune sincronizarea tuturor datelor în fișier și eliberarea resurselor asociate

Ce stim despre date si fisiere?

- Ce tipuri de date avem?
- Ce tipuri de fisiere cunoasteti?

Fisiere Text

shutterstock.com · 1453518842

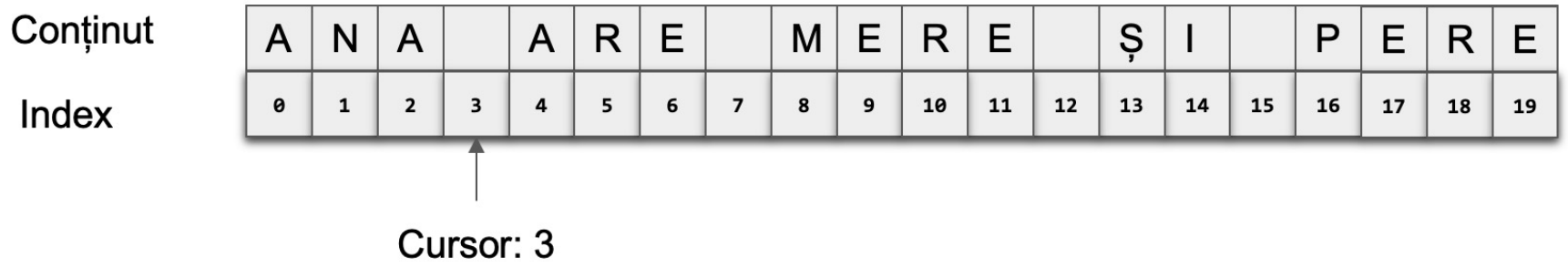
Operații cu fișiere, în Python - OPEN

- **Crearea și deschiderea fișierelor**: avem o singură operație
- fisier = open("/tmp/my_file.txt", "r")
- Operația open deschide fișierul de la calea specificată ca prim argument, în modul specificat în al doilea argument și ne returnează un obiect special Python
- Argumentul al doilea îi spune modul în care să deschidă fișierul:

`r`	Deschide pentru citire. Nu creează fișierul. Reprezintă valoarea implicită	`w`	Deschide pentru scriere. Fișierul este trunchiat dacă există altfel este creat un fișier gol
`r+`	Deschide fișierul pentru citire și scriere. Nu creează fișierul și se poziționează la începutul fișierului	`w+`	Deschide pentru citire și scriere. Fișierul este trunchiat dacă există altfel este creat un fișier gol
`a`	Fișierul este deschis doar pentru scriere. Este creat dacă nu există. **Cursorul este poziționat la final.**	`a+`	Fișierul este deschis pentru citire și scriere. Este creat dacă nu există. **Cursorul este poziționat la final.**

Operații cu fișiere în Python - TELL

- Dar oare ce este cursorul de care am menționat mai devreme ?
- Este un identificator care ne spune poziția din fișier unde încep operațiile de scriere sau citire



- În exemplul de mai sus operațiile de citire sau scriere vor începe cu poziția 3. Indecșii sunt indexați începând cu 0
- Valoarea actuală a cursorului este returnată de metoda `tell`.

Operații cu fișiere în Python – CLOSE

- **Operația de închidere a unui fișier**
  - metoda close oferită de obiectul returnat de funcția open
  - se asigură că toate datele menite a fi scrise au fost transmise sistemului de operare
  - notifică SO că a terminat lucrul cu fișierul
  - Pentru eficientizarea operațiilor de intrare/ieșire este posibil ca Python (sau SO) să nu scrie imediat datele în fișier ci să le țină într-o zonă tampon pana se adună suficiente date.
  - close asigura că aceste zone tampon au fost golite și că datele din ele au fost transmise pentru a fi scrise pe disc
  - După operația de închidere nici o operație nu mai este posibilă asupra fișierului

```
fisier.close()
```

Operații cu fișiere în Python- READ

Operația de citire:

- metoda `read` oferită de obiectul returnat de funcția open.

```
read(size=-1)
```

- Funcția citește și returnează maxim `size` caractere.
- Citește din fișier caractere pană „strânge” `size` caractere sau pana ajunge la sfârșitul fișierului.
- Dacă `size` este -1 atunci funcția citește toate caracterele până la sfârșitul fișierului.
- Aceasta este valoarea implicită.
- Funcția `read` returnează, în cazul fișierelor text, un șir de caractere
- Operația de citire este relativă la poziția cursorului

Operații cu fișiere în Python
–
READ din fisiere TEXT

- În cazul fișierelor text avem la dispoziție o metodă specială pentru citirea a câte unei linii de text din fișier:

```
readline(size=-1)
```

- Funcția citește din fișier pana la întâlnirea caracterului de **linie nouă** sau până la sfârșitul fișierului, care apare primul.
- Argumentul `size` ne permite să limităm numărul de caractere citite

```
readlines(hint=-1)
```

- Funcția `**readlines**` care permite citirea mai multor linii (cu posibilitatea limitării numărului maxim de caractere folosind argumentul `hint`)

Operații cu fișiere în Python: WRITE

- **Operația de scriere**:
- folosind metoda `write` oferită de obiectul returnat de funcția open.

```
write(text)
```

- Funcția scrie șirul de caractere primit în argumentul `text`
- Funcția returnează întotdeauna numărul de caractere scrise, mai exact lungimea șirului de caractere
- Operația de scriere va suprascrie caracterele deja existente în fișier
- În situația în care este atins sfârșitul de fișier atunci acesta este mărit în așa fel încât să poată fi scrise datele
- Nu putem insera date fără suprascriere. Dacă dorim acest lucru atunci suntem răspunzători să *deplasăm datele în cadrul fișierului*

Există sisteme de fișiere care permit inserția datelor dar nu este în scopul acestui curs.

Operații cu fișiere în Python: Deplasare

- **Operația de deplasare**: este realizată folosind metoda `seek` oferită de obiectul returnat de funcția open.

```
seek(cookie, whence=0)
```

- Funcția deplasează cursorul la poziția specificată de argumentul `cookie` (denumit și offset sau deplasament)
- În funcție de valoarea argumentului `whence` această poziționare este relativă la:
    - 0: începutul fișierului. În acestă situație deplasamentul trebuie să fie pozitiv și este relativ la începutul fișierului
    - 1: poziția actuala. În această situație deplasamentul poate să fie și negativ și este relativ la poziția actuală a cursorului
    - 2: sfârșitul fișierului. În această situație deplasamentul este de obicei negativ și este relativ la sfârșitul fișierului. Dacă este pozitiv duce la creșterea fișierului cu valoarea zero.

Operații cu fișiere în Python: Trunchiere

- **Operația de trunchiere**: este realizată folosind metoda `truncate` oferită de obiectul returnat de funcția open.

```
truncate(pos=None)
```

- Funcția trunchiază/redimensioneaza fișierul pana la poziția indicată de argumentul `pos`
- Argumentul `pos` este relativ la începutul fișierului. În situația în care este omis valoarea acestuia este dată de poziția curentă a cursorului.
- Operația de trunchiere șterge toate datele după `pos`

Operații cu fișiere

OPEN

TELL

READ

WRITE

SEEK

TRUNCATE

CLOSE

Exemplu de lucru cu fisiere

Ce se întâmplă dacă apare o excepție la deschiderea fișierului ?

```
try:
    f = open("fisier.txt", "w+")
    sir = f.read() # Citim tot fisierul
    sir = sir.upper()
    f.seek() # Sarim la inceputul fisierului
    f.write(sir)
finally:
    f.close()
```

- deschidem fișierul pentru **citire și scriere** (modul w+),
- **citim** tot conținutul fișierului (funcția `read` nu primește nici un argument),
- convertim tot textul în majuscule,
- ne **deplasăm** din nou la începutul fișierului folosind `seek` (cursorul s-a deplasat în urma operației de citire)
- scriem noul conținut.
- Fișierul închis în cadrul blocului `try: finally` pentru a ne asigura că fișierul este închis chiar și în cazul în care este aruncată o excepție

Operații cu fișiere în Python. Instrucțiunea **with**

- Blocul `try: … finally: …` poate fi înlocuit cu instrucțiunea `with`, instrucțiune care se va asigura că operația închidere va fi executată tot timpul

```
with open("fisier.txt", "w+") as f:
    sir = f.read() # Citim tot fisierul
    sir = sir.upper()
    f.seek() # Ne deplasam la inceputul fisierului
    f.write(sir)
```

- În acest caz instrucțiunea `with` simplifică lucrul cu fișierele
- Instrucțiunea `with` funcționează și cu alt fel de obiecte, nu doar fișiere dar nu o să discutăm momentan despre acestea (vedeți PEP 343 pentru detalii)

Fișiere binare

```
0000000 0000 0001 0001 1010 0010 0001 0004 0128
0000010 0000 0016 0000 0028 0000 0010 0000 0020
0000020 0000 0001 0004 0000 0000 0000 0000 0000
0000030 0000 0000 0000 0010 0000 0000 0000 0204
0000040 0004 8384 0084 c7c8 00c8 4748 0048 e8e9
0000050 00e9 6a69 0069 a8a9 00a9 2828 0028 fdfc
0000060 00fc 1819 0019 9898 0098 d9d8 00d8 5857
0000070 0057 7b7a 007a bab9 00b9 3a3c 003c 8888
0000080 8888 8888 8888 8888 288e be88 8888 8888
0000090 3b83 5788 8888 8888 7667 778e 8828 8888
00000a0 d61f 7abd 8818 8888 467c 585f 8814 8188
00000b0 8b06 e8f7 88aa 8388 8b3b 88f3 88bd e988
00000c0 8a18 880c e841 c988 b328 6871 688e 958b
00000d0 a948 5862 5884 7e81 3788 1ab4 5a84 3eec
00000e0 3d86 dcb8 5cbb 8888 8888 8888 8888 8888
00000f0 8888 8888 8888 8888 8888 8888 8888 0000
0000100 0000 0000 0000 0000 0000 0000 0000 0000
*
0000130 0000 0000 0000 0000 0000 0000 0000
000013e
```

Fișiere binare

- Pana acum am discutat doar despre fișiere text dar este important de știut că Python oferă și suport pentru fișiere binare
- Fișierele binare se deschid similar cu fișierele text, adăugând modul **"b"** pentru al doilea argument al funcției `open`
- Majoritatea operațiilor prezentate mai înainte pentru fișiere text rămân valabile cu **excepția `readline` și `readlines`.**
- De asemenea operația de scriere `write` nu mai primește șiruri de caractere ci obiecte de tipul `bytes`, reprezentând o secvență de octeți. Similar funcția de citire `read` returnează obiecte de tip `bytes` și nu șiruri de caractere

Tipurile bytes și bytearray

- Tipul `**bytes**` este folosit pentru a reprezenta o secvență **imutabilă** de octeți
- Tipul `**bytearray**` este folosit pentru a reprezenta una **mutabilă**
- Amândouă tipurile de date pot fi inițializate cu:
  - Un șir de caractere: `b = bytes("Ana", "utf8")`.
    - În această variantă șirul de caractere este convertit într-o succesiune de octeți folosind convenția `utf8`.
    - În loc de `utf8` putem avea și alte codificări precum `ascii`, `utf16`, `iso-8859-1`, etc
  - Un `int`, situație în care se crează o secvență de dimensiunea respectivă, inițializată cu 0
  - Un obiect iterabil de valori între 0 și 255
  - Un alt obiect de acest tip
- În cazul în care este posibil valorile de tip `bytes` pot fi convertite în șir de caractere folosind metoda `decode(encoding)` unde `encoding` reprezintă codificarea.
- În situația în care nu este posibilă conversia este aruncată excepția UnicodeDecodeError

Tipuri Speciale de Fișiere

Tipuri speciale de fișiere

Până acum am discutat despre fișiere text și fișiere binare. În cazul amândurora noi suntem responsabili să ne gestionăm structura datelor, să le **convertim din șir de caractere în tipuri native sau invers**

Acest proces poate să fie foarte **laborios** și complicat. Din fericire există un mecanism care ne stă la dispoziție: **procesul de serializare**.

Serializarea ne permite să convertim între tipurile de date Python și o reprezentare bine determinată, formă în care putem să salvăm pe disc sau să transmitem peste rețea.

Există mai multe mecanisme de serializare disponibile în Python: pickling, JSON sau CSV. Noi vom discuta în acest curs doar despre ultimele două.

Tipuri Speciale de Fișiere (JSON)

```
[
    {
      "description": "quarter",
      "mode": "REQUIRED",
      "name": "qtr",
      "type": "STRING"
    },
    {
      "description": "sales representative",
      "mode": "NULLABLE",
      "name": "rep",
      "type": "STRING"
    },
    {
      "description": "total sales",
      "mode": "NULLABLE",
      "name": "sales",
      "type": "INTEGER"
    }
]
```

JSON

- JSON (JavaScript Object Notation) este un format simplu pentru schimb de date inspirat de sintaxa limbajului JavaScript
- Python ne oferă posibilitatea de a reprezenta tipuri elementare de date în format JSON. Această funcționalitatea este oferită de pachetul `json`:

```
import json
studenti = [{ 'nume': 'marian', 'nota': 5 }, {'nume': 'ion',
'nota': 7}]
sir = json.dumps(studenti)
studenti2 = json.loads(sir)
```

- `**dumps**` va returna șirul de caractere: '[{"nume": "marian", "nota": 5}, {"nume": "ion", "nota": 7}]'
- Operația inversă este `**loads**` și primește un șir de caractere în format JSON și returnează un obiect Python. În exemplu de mai `studenti2` va fi echivalentul lui `studenti`

Tipuri Speciale de Fișiere (CSV)

```
1,"F. Rosu","Numerical simulation algorithm for fractional-order systems implemented in
CUDA",2020,"Proceedings - 2020 22nd International Symposium on Symbolic and Numeric Algorithms for
Scientific Computing, SYNASC 2020","","","https://www.scopus.com/inward/
citedby.uri?partnerID=HzOxMe3b&scp=85098785408&origin=inward",5,"2021-11-08 13:57:00","Conference
Paper","10.1109/SYNASC51798.2020.00021","","https://api.elsevier.com/content/abstract/
scopus_id/85098785408",,,63,66,1,1.00,1,1,1,"","",""
1,"C. Bonchis","Information theory over multisets",2008,"Computing and
Informatics","","","https://www.scopus.com/inward/
citedby.uri?partnerID=HzOxMe3b&scp=56249103434&origin=inward",6,"2021-11-08 13:57:00","Conference
Paper","","1335-9150","https://api.elsevier.com/content/abstract/
scopus_id/56249103434",27,,441,451,1,0.08,1,1,13,"","",""
1,"C. Bonchiş","Number encodings and arithmetics over multisets",2006,"Proceedings of the 8th
International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, SYNASC
2006","","","https://www.scopus.com/inward/
citedby.uri?partnerID=HzOxMe3b&scp=46449098615&origin=inward",7,"2021-11-08 13:57:00","Conference
Paper","10.1109/SYNASC.2006.58","","https://api.elsevier.com/content/abstract/
scopus_id/46449098615",,,354,361,1,0.07,1,1,15,"","",""
0,"G. Istrate","It's not whom you know, it's what you, or your friends, can do: Coalitional
frameworks for network centralities",2020,"Proceedings of the International Joint Conference on
Autonomous Agents and Multiagent Systems, AAMAS","","","https://www.scopus.com/inward/
citedby.uri?partnerID=HzOxMe3b&scp=85096700570&origin=inward",8,"2021-11-08 13:57:00","Conference
Paper","","1548-8403","https://api.elsevier.com/content/abstract/
scopus_id/85096700570",2020,,566,574,0,0.00,0,1,1,"","",""
0,"G. Istrate","Attacking power indices by manipulating player reliability",2019,"Proceedings of
the International Joint Conference on Autonomous Agents and Multiagent Systems,
AAMAS","","","https://www.scopus.com/inward/
citedby.uri?partnerID=HzOxMe3b&scp=85077061754&origin=inward",9,"2021-11-08 13:57:00","Conference
Paper","","1548-8403","https://api.elsevier.com/content/abstract/
scopus_id/85077061754",1,,538,546,0,0.00,0,1,2,"","",""
0,"F. Turcu","Vector partitions, multi-dimensional Faà di Bruno formulae and generating
algorithms",2020,"Discrete Applied Mathematics","","https://api.elsevier.com/content/article/
eid/1-s2.0-S0166218X18304773","https://www.scopus.com/inward/
citedby.uri?partnerID=HzOxMe3b&scp=85055246406&origin=inward",10,"2021-11-08
13:57:00","Article","10.1016/j.dam.2018.09.012","0166-218X","https://api.elsevier.com/content/
abstract/scopus_id/85055246406",272,,90,99,0,0.00,0,1,1,"","",""
0,"C. Bonchis","Compositional asynchronous membrane systems ?",2006,"Pre-Proceedings of the
International Conference on Bio-Inspired Computing - Theory and Applications: Membrane Computing
Section, BIC-TA 2006","","","https://www.scopus.com/inward/
citedby.uri?partnerID=HzOxMe3b&scp=84883234893&origin=inward",11,"2021-11-08 13:57:00","Conference
Paper","","","https://api.elsevier.com/content/abstract/
scopus_id/84883234893",,,51,59,0,0.00,0,1,15,"","",""
```

CSV

```
# importing csv module
import csv

filename = ”my_export.csv"

# initializing the titles and rows list
header = []
rows = []

# reading csv file
with open(filename, 'r') as csvfile:
    # creating a csv reader object
    csvreader = csv.reader(csvfile)

    # extracting header field names through first row
    header = next(csvreader)

    # extracting each data row one by one
    for row in csvreader:
        rows.append(row)

# printing the header field names
print(‘Column names are:' + ', '.join(col for col in
header))

#  printing first 10 rows
print('\nFirst 10 rows are:\n')
for row in rows[:10]:
    for col in row:
        print("%10s"%col),
    print('\n')
```

Fisiere

- Text
- Binare
- Speciale: JSON, CSV