

Python de la zero

Caiet de laboratoare și proiect

Ediția completă 19 octombrie 2026. Exemple fictive și programe de laborator. Python 3.12.3 / Linux; requests 2.31.0 pentru lecția HTTP. Lectura este estimată separat de practică. Comenzile Windows sunt documentate, fără verificarea instalatorului sau activării în Windows.

Cuprins

1. Instalare și primul script
2. Tipuri și operatori
3. Liste și dicționare
4. Funcții și module
5. Fișiere și CSV
6. Cereri web cu requests
7. Un script util cap-coadă

Cum folosești caietul

Parcurge mai întâi explicațiile din manual. Rezolvă exercițiile, compară rezultatele și păstrează dovada verificării în registrul CSV. Anexa conține programele principale pentru reproducerea laboratoarelor; fragmentele introductive sunt în manual. Folosește un director de lucru nou și date fictive.

Instalare și primul script

Laborator: demonstrează contractul comenzii

Înainte de testare, scrie pe hârtie rezultatele așteptate pentru cinci comenzi: Ana fără opțiuni, Ana cu trei sesiuni, Ana cu zece sesiuni, Ana cu zero sesiuni și Ana cu textul trei în locul numărului. Primele trei trebuie să reușească și să producă 25, 75 și 250 de minute. Ultimele două trebuie să fie respinse. Nu este suficient să verifici numai exemplul fericit cu valoarea trei.

Adaugă două verificări pentru nume: Ana Maria transmis într-un singur argument și un argument care conține numai spații. Primul trebuie păstrat integral, al doilea trebuie respins. Dacă ai omis ghilimelele, corectează comanda, nu programul. Eroarea aparține modului de lansare. Dacă ai eliminat strip și spațiile trec validarea, corectează programul și repetă cazul pentru a demonstra remediarea.

Cazul de la limită este deosebit de valoros. Valoarea zece trebuie acceptată, iar unsprezece trebuie respinsă. Dacă ai scris accidental < 10 , zece eșuează și testul îți arată că implementarea diferă de regula formulată în cuvinte. Dacă ai verificat doar că numărul este pozitiv, unsprezece va trece. Aceeași disciplină va fi folosită pentru sume, stocuri și rânduri CSV.

Pentru exercițiul de modificare, schimbă convenția la 30 de minute pe sesiune. Predicția pentru trei sesiuni devine 90 de minute. Actualizează și descrierea parserului, nu doar înmulțirea, apoi verifică --help. Un program poate calcula corect și totuși să inducă utilizatorul în eroare dacă documentația lui afirmă altceva. Acest exercițiu verifică acordul dintre comportament și explicație.

Rezolvare orientativă și fișa de predare

Versiunea inițială a laboratorului folosește 25 de minute. Pentru modificare, cele două locuri care se schimbă sunt textul description și operandul 25 din calcul. Nu modifica limitele de la una la zece: enunțul nu a cerut acest lucru. Reexecută toate cazurile, inclusiv cele invalide; o schimbare a duratei nu trebuie să transforme zero sau unsprezece în intrări acceptate.

Într-un README.txt notează versiunea interpretorului, calea relativă a executabilului mediului, comanda de creare a mediului și trei comenzi de verificare. Include un rezultat de succes și unul de eșec, cu codurile observate. Menționează că programul nu necesită pachete externe. Păstrează sursele salut.py, diagnostic.py, intreaba.py și plan.py lângă acest document.

Ultima probă este să închizi terminalul și să reproduci rularea din instrucțiunile tale. Dacă ai folosit activarea fără să o documentezi, poți descoperi că python indică altă instalare. Folosirea căii explicite către interpretorul din .venv elimină această ambiguitate în laborator. Nu arhiva mediul în locul surselor: un coleg are nevoie de program și de pașii de recreare, nu de instalarea ta locală.

Ai terminat lecția când poți explica de ce programul acceptă trei sesiuni, de ce respinge zero, de ce Ana Maria trebuie transmis ca un argument și cum demonstrezi ce interpretor l-a executat. Aceste explicații sunt mai utile decât o captură cu un singur mesaj de succes. În următoarea lecție vom analiza ce înseamnă un număr, un text și o valoare booleană, astfel încât validarea să fie o decizie explicită.

Studiu de caz: terminalul merge, butonul editorului nu

Maria deschide plan.py în editor și îl rulează din terminal cu numele Ana și trei sesiuni. Obține rezultatul corect. Apasă apoi butonul Run, iar programul spune că lipsește numele. Prima ipoteză ar putea fi că editorul nu înțelege argparse, dar rezultatul arată altceva: editorul a lansat scriptul fără argumentul obligatoriu. Codul și parserul funcționează conform contractului.

Pentru diagnostic, Maria compară cele două lansări. În terminal poate vedea explicit numele fișierului, Ana și opțiunea --sesiuni 3. În configurația butonului există doar calea fișierului. Soluția este să configureze argumentele pentru rulare sau să folosească terminalul documentat în README. Nu adaugă un nume implicit în program doar ca să ascundă diferența: aceea ar fi o schimbare a cerinței, nu repararea lansării.

Un al doilea coleg obține mesajul vechi, deși a schimbat durata la 30. El verifică mai întâi dacă a salvat. Apoi rulează diagnostic.py și compară calea fișierului cu directorul deschis în editor. Descoperă două copii ale proiectului, una într-un director descărcat și alta în directorul de lucru. Comanda lui execută copia veche. Corectura constă în lansarea sursei potrivite și eliminarea ambiguității din notițe, nu în reinstalarea Python.

Aceste situații ilustrează o ordine utilă a verificărilor: confirmă comanda, fișierul salvat, calea executată și intrările, apoi investighează algoritmul. Un rezultat greșit poate proveni dintr-un program corect lansat în alt context. În registru, scrie cauza precisă și proba care a confirmat-o. Formularea „am reinstalat și acum merge” nu explică incidentul și nu ajută când aceeași problemă reapare.

Tipuri și operatori

Laborator: cazul obișnuit nu este suficient

Pregătește un tabel cu intrarea, rezultatul așteptat și rezultatul observat. Pentru 12.50, 3 și 10, totalul este 33.75. Pentru aceleași preț și cantitate cu reducere zero, totalul este 37.50. Cu reducere o sută, totalul este 0.00. Pentru preț zero și cantitate trei, subtotalul și totalul sunt zero, iar comanda reușește. Astfel demonstrezi că zero este tratat potrivit sensului fiecărui câmp.

Testează apoi cantitatea zero, cantitatea 101 și procentul 100.01: toate trebuie respinse. Prețul 10000.00 este acceptat, iar 10000.01 este respins. Nu este de ajuns să testezi doar numere foarte mari; valorile imediat lângă limită găsesc greșeli precum $<$ în loc de $\leq$. Pentru fiecare respingere verifică atât mesajul, cât și absența liniei Total în ieșirea standard.

Adaugă intrările 12,50, NaN, Infinity și 1e2 ca preț. Contractul le respinge, chiar dacă unele au sens pentru alte parsere. Adaugă 3.0 și +3 pentru cantitate. Dacă un coleg dorește să le accepte, cere mai întâi schimbarea regulii și a testelor. O modificare a parserului este o schimbare a interfeței, nu o simplă corectură cosmetică.

Pentru rotunjire verifică 0.05, 1 și 10. Rezultatul trebuie să fie economia 0.01 și totalul 0.04. Apoi verifică 0.15, 1 și 10: economia 0.02 și totalul 0.13. Aceste cazuri expun regula ROUND_HALF_UP. Compară rezultatele cu calculul manual și explică de ce rotunjirea economiei nu este același lucru cu rotunjirea totalului calculat printr-o altă formulă.

Exercițiu de diagnostic cu rezolvare

Un coleg propune trei simplificări: înlocuiește Decimal cu float, validează prețul cu if not pret și convertește cantitatea prin `int(float(text))`. Analizează-le separat. Prima schimbă modelul numeric. A doua respinge incorect un preț zero. A treia poate transforma o cantitate fracționară în întreg și poate accepta forme pe care interfața inițială le refuza.

Rezolvarea nu este să adaugi rotunjiri până când un exemplu arată bine. Păstrează Decimal pornit din text, verifică explicit limitele și convertește cantitatea numai după validarea formei. Rulează din nou cazurile pentru zero, 3.5 și rotunjirea de jumătate de ban. O corectură este demonstrată când elimină cazul greșit și păstrează comportamentul celor valide.

Ca extensie, adaugă o taxă fixă fictivă de ambalare de 2.00, aplicată după reducere numai dacă totalul produselor este pozitiv. Scrie întâi așteptările: exemplul 33.75 devine 35.75, iar o reducere de o sută de procente lasă totalul zero și nu adaugă taxa. Aceasta este regula exercițiului, nu o practică comercială universală. Implementarea poate folosi o condiție `total > Decimal("0")` și o adunare explicită.

În registrul de lucru păstrează programul de bază și extensia separat, pentru a nu amesteca rezultatele. Explică în cuvinte unde se aplică reducerea și unde taxa. Dacă altcineva nu poate reproduce suma din regulile scrise, problema este încă incomplet definită, chiar dacă programul rulează fără excepții.

Lecția este încheiată când poți urmări traseul unei intrări de la text până la total, poți spune de ce fiecare respingere este justificată și poți reproduce calculele de control. Listele și dicționarele din lecția următoare vor permite aplicarea acestor reguli mai multor rânduri. Tipurile și limitele rămân aceleași responsabilități, chiar dacă volumul datelor crește.

De ce păstrăm separat calculul și mesajul

Să presupunem că vrei să afișezi totalul cu moneda lei. Expresia `f"{total:.2f} lei"` produce text pentru cititor. Dacă reasociezi numele total cu acel șir și încerci apoi să adaugi taxa de ambalare, ai pierdut valoarea numerică din acel nume. Păstrează total pentru Decimal și folosește un nume precum `mesaj_total` pentru reprezentarea destinată ecranului. Astfel, testele pot compara numerele, iar interfața poate schimba formularea fără să schimbe aritmetica.

Aceeași separare ajută când rezultatul ajunge într-un CSV. Dacă scrii „33.75 lei” într-o coloană pe care alt program o așteaptă numerică, ai schimbat contractul fișierului. O variantă clară este o coloană total cu reprezentarea 33.75 și o coloană moneda cu RON, atunci când proiectul chiar cere acea monedă. În laborator nu exportăm încă fișiere, dar pregătim această distincție pentru lecția despre CSV.

Un alt caz este comparația cu un rezultat așteptat. Pentru totalul demonstrativ verifică valoarea `Decimal("33.75")`, nu doar prezența textului 33 într-un mesaj. Un test textual prea permisiv poate accepta și 133.75 sau un mesaj de eroare care conține aceleași cifre. Când verifici comanda întreagă, compară liniile complete, codul de ieșire și canalul pe care este afișat mesajul. Când verifici calculul izolat, compară valorile numerice.

Pentru o probă de regresie, păstrează intrarea 12.50, 3, 10 și rezultatul exact înainte să adaugi taxa de ambalare. După extensie, rezultatul final se schimbă, dar subtotalul și economia trebuie să rămână identice. Această observație îți permite să localizezi o greșeală: dacă economia se schimbă, ai aplicat taxa prea devreme sau ai modificat formula reducerii. Un rezultat intermediar bine numit devine o probă despre ordinea operațiilor.

Exercițiul suplimentar este să afișezi patru linii: subtotal, economie, taxă și total. Pentru cazul de bază, valorile sunt 37.50, 3.75, 2.00 și 35.75. Pentru reducere integrală, sunt 37.50, 37.50, 0.00 și 0.00. Explică de ce taxa zero din al doilea caz este calculată conform regulii, nu o valoare lipsă. Ai astfel o verificare care leagă tipurile, condițiile și prezentarea rezultatului.

Liste și dicționare

Laborator: modifică datele, păstrează regulile

Începe cu programul exact și notează cele cinci rezultate principale. Înlocuiește apoi suma invalidă a lui a3 cu 500. Vei avea patru valide și o respingere, iar totalul devine 35500 bani. Statusul nouă va avea totalul 10500. Nu schimba simultan și duplicatul: vrem să putem atribui diferența unei singure modificări a intrării.

Revino la datele inițiale și adaugă un rând cu suma True. Acesta trebuie respins ca sumă invalidă. Dacă ai înlocuit `type(...) is int` cu `isinstance(..., int)`, testul îți arată de ce contractul strict necesită atenție. Adaugă apoi o sumă negativă, un id numai din spații și un status necunoscut. Fiecare trebuie să producă un motiv distinct potrivit primei reguli încălcate.

Testează lista goală. Numărul de valide și respinse este zero, iar totalurile pe cele două stări rămân zero. Nu trebuie să accesezi primul element pentru a inițializa raportul. Acum testează un dicționar fără suma_bani și unul cu o cheie suplimentară. Ambele sunt respinse pentru structură, deoarece am ales un contract cu exact trei chei. Dacă vrei extensibilitate, schimbă explicit regula și documentează noile câmpuri.

Pentru duplicate, introdu mai întâi o înregistrare invalidă cu id a9, apoi una validă cu același id. A doua este acceptată. Inversează situația: două înregistrări valide a9, cu sume diferite. Prima este acceptată și a doua respinsă. Explică de ce schimbarea ordinii poate schimba totalul conform politicii alese. Repetabilitatea cere aceeași intrare ordonată și aceeași regulă, nu doar aceleași elemente într-o ordine arbitrară.

Rezolvare și extensie: număr de cereri pe status

Extensia cerută este să afișezi și câte cereri valide există în fiecare stare. Creează un dicționar numere cu aceleași două chei și valori zero. În bucla care agregă sumele, mărește `numere[rand["status"]]` cu unu. Pentru datele inițiale vei obține două cereri noi și una închisă. Totalul numerelor este trei, chiar dacă suma uneia dintre cererile noi este zero.

Nu încerca să deduci numărul cererilor din suma banilor și nu număra rândurile respinse într-o categorie validă. Raportul de erori este o ieșire separată. O extensie utilă pentru operator este gruparea motivelor de respingere într-un alt dicționar, dar păstrează și înregistrările individuale: o statistică nu arată singură ce trebuie corectat în sursă.

Predă `lot.py`, datele inițiale, două variante de intrare și un tabel cu rezultatele așteptate. Adaugă o explicație pentru zero, duplicate și copiile de dicționare. Dacă poți urmări manual fiecare rând până în ieșire și programul reproduce acea urmărire, ai o bază solidă pentru următoarea lecție. Acolo vom extrage regulile în funcții importabile, astfel încât testele să poată furniza alte date fără editarea programului principal.

Studiu de caz: două rapoarte cu aceeași sumă

Un operator primește două rapoarte care arată ambele 35000 bani. Primul conține a1, a2 și a4. Al doilea conține a1, a2 și o înregistrare a3 transformată în zero. La nivelul sumei par echivalente, dar al doilea a pierdut o cerere validă și a acceptat una invalidă. Pentru a le deosebi, comparăm identificatorii, numărul înregistrărilor și motivele respingerilor. Acest exemplu arată de ce testarea numai a rezultatului agregat este insuficientă.

Începe investigația prin a construi o listă cu id-urile acceptate. Păstrează ordinea dacă raportul promite ordinea sursei. Dacă ordinea nu este relevantă pentru o anumită verificare, compară mulțimile, dar verifică separat numărul elementelor. O mulțime elimină duplicatele și poate ascunde faptul că același id apare de două ori în lista rezultatului. Structura aleasă pentru test poate pierde exact informația pe care doreai să o verifici.

Acum presupune că sursa conține două cereri cu id-urile a1 și A1. Contractul lecției le tratează ca diferite. Un coleg propune să transforme toate id-urile în litere mici. Această schimbare nu este doar prezentare: poate crea duplicate noi și poate modifica totalul. Înainte de implementare trebuie stabilit dacă identificatorii sunt sensibili la majuscule în sistemul care îi produce. Normalizarea fără contract poate distruge informație validă.

Un alt coleg propune să elimine orice înregistrare cu sumă zero pentru a face raportul mai scurt. Dar o cerere gratuită poate conta în numărul cererilor și în distribuția pe status. Filtrarea pentru afișare trebuie deosebită de validare. Poți oferi o vedere separată cu sume pozitive, fără să modifice lista completă a cererilor valide sau să ascunzi faptul că ai aplicat un filtru.

Cum verifici independența rezultatului

Păstrează sursa înainte de procesare și compar-o după. Pentru datele simple ale lecției, poți construi copii ale dicționarilor pentru o comparație de valori. Apoi modifică deliberat suma din primul dicționar al rezultatului și verifică sursa. Dacă se modifică și ea, ai reutilizat obiectul original în loc să construiești înregistrarea normalizată. Revino la construirea explicită a dicționarului nou.

Această probă este diferită de verificarea ordinii. sorted întoarce o listă nouă, dar elementele ei pot fi aceleași dicționare ca în lista din care ai pornit. Sortarea fără modificarea ordinii sursei nu garantează independența obiectelor interioare. Când ai nevoie doar să afișezi o altă ordine, partajarea poate fi acceptabilă. Când rezultatul va fi editat, trebuie să decizi ce date trebuie copiate.

În fișa de lucru scrie separat ce păstrează programul: valorile sursei, ordinea inițială și identitatea obiectelor. Nu folosi formularea generală „face o copie” fără să precizezi nivelul. Această precizie te ajută ulterior când listele conțin alte liste, când dicționarele sunt construite din JSON și când un modul primește date de la altul.

Funcții și module

Laborator: o matrice de conversii

Testează `bani_din_text` cu 0, 12, 12.5, 12.50 și spații în jurul ultimei valori. Rezultatele sunt 0, 1200, 1250, 1250 și 1250. Formatarea lor produce 0.00, 12.00 și 12.50 pentru celelalte trei. Observă că normalizarea poate schimba reprezentarea fără să schimbe valoarea. Nu promitem să păstrăm zerourile inițiale ale unei sume, deoarece suma nu este un identificator.

Intrările 12,50, -1, +1, 1e2, NaN, șirul gol și 0.001 trebuie să producă `ValueError`. Un întreg 1250 și valoarea `True` trebuie să producă `TypeError` în `bani_din_text`, fiindcă interfața cere text. Pentru `text_din_bani`, `True` este de asemenea respins. Verifică limita 999999.99 și respingerea lui 1000000: aceasta din urmă are șapte cifre înainte de punct.

Pentru `valideaza_rand` folosește o intrare cu diacritice, una fără coloana suma, una cu o coloană suplimentară și una cu nume format numai din spații. Prima trebuie acceptată, celelalte respinse. Verifică și un id de 32 de caractere versus unul de 33. Dacă toate testele tale folosesc numai id-uri de două litere, o greșeală la limită poate rămâne neobservată.

Într-un fișier de test, importă modulul și verifică faptul că importul nu afișează 12.50. Apoi rulează modulul direct și verifică exact această ieșire. Cele două probe sunt diferite și ambele sunt necesare pentru contractul de utilizare. Repetă importul într-un proces nou când verifici efectele, deoarece Python poate reutiliza un modul deja încărcat în același proces.

Rezolvare orientativă și criterii de încheiere

Pentru verificarea unei excepții, folosește un bloc `try` care conține numai apelul testat. În `except ValueError` confirmă cazul, iar în `else` ridică o eroare de test deoarece intrarea a fost acceptată greșit. Nu folosi `except` fără tip: ar accepta și o problemă fără legătură cu regula. Pentru valori valide, compară rezultatul exact cu întregul așteptat.

Extensia lecției este o funcție `total_bani` care primește o listă de înregistrări deja validate și întoarce suma câmpurilor `suma_bani`. Pentru lista goală rezultatul este zero. Pentru valori 10000, 25000 și zero este 35000. Documentează faptul că funcția primește forma validată, nu dicționarele text din CSV. Dacă vrei să accepți ambele forme, vei complica inutil contractul și testele.

Predă `reguli.py`, `demo_reguli.py` și rezultatele testelor. Adaugă în README regulile pentru sumă, id și nume, plus explicația diferenței dintre `print` și `return`. Lecția este încheiată când poți folosi modulul dintr-un fișier nou fără să modifice sursa lui și fără să pornească accidental o citire de fișiere. Următorul pas este să aduci datele din CSV și să păstrezi aceeași validare.

O funcție pură și granița de intrare

Funcțiile de conversie din această lecție sunt predictibile: pentru același argument valid întorc aceeași valoare și nu modifică fișiere sau stări globale. Acest lucru le face ușor de verificat. Dacă funcția `bani_din_text` ar citi singură un fișier de configurare ascuns pentru separatorul zecimal, același apel ar putea produce rezultate diferite pe două calculatoare. O astfel de dependență ar trebui transmisă explicit sau documentată ca parte din contract.

Nu înseamnă că orice aplicație trebuie construită numai din funcții pure. Citirea unui CSV și trimiterea unei cereri HTTP au efecte și depind de mediu. Organizarea utilă este să păstrezi granițele vizibile: o funcție citește datele, alta validează valorile, alta scrie rezultatul. Când un test eșuează, poți determina dacă problema aparține transportului, regulilor sau prezentării.

De exemplu, `bani_din_text` primește text și întoarce un întreg. Nu trebuie să știe din ce coloană CSV a venit textul și nici ce număr de înregistrare are. Bucula de import adaugă acel context când prinde excepția. Astfel, aceeași funcție poate fi folosită într-o comandă de terminal sau într-un client HTTP fără să producă un mesaj fals despre o linie CSV care nu există.

Verificări care exprimă relații între rezultate

Pe lângă exemple individuale, poți testa o relație generală pentru un set de valori controlate: dacă formatezi un număr de bani acceptat și apoi îl convertești înapoi, trebuie să obții aceeași valoare. Pentru zero, unu, cincizeci, o sută douăzeci și cinci și limita maximă a unei intrări, verifică `bani_din_text(text_din_bani(valoare)) == valoare`. Nu extinde proba peste limita parserului fără să schimbi regula.

Această relație găsește greșeli precum completarea fracției în partea greșită. Dacă 12.5 devine 1205 în loc de 1250, unele teste cu două zecimale încă ar putea trece. Testele pentru o singură cifră fracționară și relația de conversie fac problema vizibilă. Nu te baza numai pe 12.50, fiindcă forma lui nu exercită completarea cu zero.

Pentru `valideaza_rand`, o altă relație utilă este independența rezultatului. Două apeluri cu aceeași sursă trebuie să producă valori egale, dar dicționare noi. Dacă modifici rezultatul primului apel, al doilea rezultat și sursa trebuie să rămână neschimbate. Nu este obligatoriu să testezi identitatea fiecărui șir imutabil; ne interesează containerele pe care codul le-ar putea modifica.

Cum refaci o funcție fără să schimbi interfața

Presupune că reorganizezi implementarea lui `bani_din_text` în două funcții interne, una pentru forma textului și alta pentru calcul. Apelantii ar trebui să primească aceleași rezultate și aceleași categorii de erori. Păstrează testele existente și rulează-le după reorganizare. Dacă schimbi simultan formatul acceptat, nu mai poți spune dacă diferența provine din refactorizare sau din noua cerință.

O schimbare de cerință trebuie documentată separat. Dacă decizi să accepți virgula zecimală, definește ce se întâmplă cu 1,234 și cu un șir care conține și punct, și virgulă. Adaugă probe pentru formele ambigue înainte de a modifica parserul. O înlocuire globală a caracterelor poate părea convenabilă, dar poate transforma o intrare greșită într-o sumă diferită de intenția operatorului.

La predare, păstrează numele modulului și al funcțiilor folosite în celelalte lecții. O îmbunătățire internă nu ar trebui să oblige cititorul să caute și să înlocuiască importuri în toate fișierele proiectului. Aceasta este valoarea unei interfețe mici și explicite: poți îmbunătăți implementarea, iar restul aplicației continuă să folosească aceeași regulă verificată.

Fișiere și CSV

Laborator: verifică structura și fidelitatea

Începe cu fișierul demonstrativ și rezultatele de control. Schimbă numele primei cereri într-un text cu virgulă, ghilimele și diacritice folosind un writer CSV, apoi citește-l înapoi. Valoarea trebuie să rămână identică după citire, cu excepția spațiilor exterioare eliminate deliberat de validare. O comparație a numărului de coloane nu dovedește singură păstrarea textului.

Creează un fișier care are doar antetul corect. Rezultatul este zero valide, zero respinse și total zero. Creează apoi un fișier gol și unul cu antetul id,suma,nume. Ambele trebuie să oprească importul. Dacă programul le acceptă, nu mai aplică schema documentată și poate asocia o sumă cu numele unei persoane fictive.

Adaugă o înregistrare cu două câmpuri și una cu patru. Ambele trebuie raportate ca număr incorect de câmpuri. Adaugă o linie fizică goală între înregistrări: parserul o expune ca înregistrare goală, pe care politica noastră o respinge. Nu o elimina în tăcere fără să schimbi documentația și testele. Un câmp citat cu o linie nouă în interior este alt caz și trebuie păstrat ca parte a aceleiași înregistrări.

Verifică un fișier cu BOM UTF-8 și unul cu octeți invalizi pentru UTF-8. Primul trebuie acceptat, al doilea respins înaintea procesării rândurilor. Verifică și un fișier mai mare decât limita: programul trebuie să refuze importul, nu să proceseze arbitrar numai începutul. Limita protejează laboratorul și face explicit volumul pentru care a fost proiectat.

Rezolvare și investigația unui total greșit

Dacă obții 355.00 pentru fișierul inițial, este probabil că ai acceptat și duplicatul a1 cu suma cinci. Verifică momentul adăugării în vazute și comparația id-ului normalizat. Dacă obții 350.00, dar patru valide, este posibil să fi transformat suma invalidă în zero. Același total nu dovedește aceeași selecție de rânduri; compară identificatorii acceptați și motivele respingerilor.

Dacă numele Atelier, demo este împărțit în două câmpuri, verifică dacă ai folosit parserul CSV sau split. Dacă apar caractere greșite în Ștefan, verifică mai întâi codificarea fișierului și setările de afișare. Nu elimina diacriticele ca remediere implicită: ai schimba datele în loc să rezolvi interpretarea lor. Sursa trebuie păstrată pentru a putea compara octeții și textul rezultat.

Predă cele trei fișiere Python, CSV-ul inițial și cel puțin patru fișiere de test, cu rezultatele explicate. Notează convenția de numerotare a înregistrărilor, politica pentru duplicate și limitele volumului. Următoarea lecție schimbă sursa datelor cu un răspuns HTTP; vei vedea că transportul adaugă alte erori, dar nu elimină nevoia de validare a structurii și valorilor.

Studiu de caz: exportul arată bine în tabel, dar importul eșuează

Un coleg trimite un fișier care se deschide corect în aplicația lui de calcul tabelar. Scriptul spune că antetul este invalid. Când deschizi fișierul într-un editor de text, vezi id;nume;suma. Aplicația a folosit punct și virgulă ca separator, în timp ce contractul laboratorului cere virgulă. Nici aplicația, nici parserul nu sunt neapărat defecte; cele două părți folosesc formate diferite.

Rezolvarea este să exportezi în formatul convenit sau să proiectezi o opțiune explicită de delimiter. Dacă adaugi opțiunea, aplic-o atât la citire, cât și la documentația fișierului și testează ambele variante. Nu încerca să înlocuiești toate punctele și virgulele din text: caracterul poate apărea legitim într-un câmp citat. Schimbarea structurii trebuie făcută de un parser, nu printr-o substituție oarbă.

Un al doilea export are antetul corect, dar o sumă este scrisă 1.250,00. Regula noastră nu acceptă separator de mii și nici virgulă zecimală. Operatorul trebuie să producă forma 1250.00 dacă acesta este sensul confirmat al datelor. Nu elimina automat punctul și virgula fără să cunoști convenția sursei. În alte contexte, aceleași semne pot avea altă interpretare.

Al treilea export conține identificatorul 0012, dar aplicația de calcul tabelar îl afișează ca 12. Importul Python păstrează câmpul textual; pierderea poate să fi avut loc înainte, în aplicația care a generat fișierul. Verifică octeții sursei și setările de import ale aplicației respective. Un program nu poate recrea sigur zerourile care au fost deja eliminate fără o regulă externă despre lungimea identificatorului.

Rând gol, câmp gol și valoare lipsă

O linie goală, trei câmpuri goale separate prin virgule și o înregistrare cu numai două câmpuri sunt trei situații distincte. Prima este expusă de parser ca listă goală, a doua ca listă de trei șiruri goale, iar a treia are o lungime insuficientă. În politica noastră, prima și a treia sunt respinse pentru număr de câmpuri; a doua ajunge la validarea valorilor și este respinsă pentru identificator.

Acest nivel de precizie ajută persoana care repară fișierul. Un mesaj generic „rând invalid” o obligă să ghicească dacă lipsește un separator sau dacă valoarea are format greșit. Totuși, nu trebuie să copiezi în mesaj toate datele brute. Pentru lotul didactic, numărul înregistrării și motivul sunt suficiente ca să găsești problema în sursa păstrată.

Dacă un nume citat conține o linie nouă, nu îl trata ca două cereri. Construiește acest caz prin `csv.writer`, citește-l și compară șirul rezultat. Numărul înregistrărilor logice trebuie să rămână același. Dacă numerotarea raportului pare diferită de cea din editor, folosește convenția documentată și explică diferența dintre înregistrare și linie fizică, în loc să schimbi contorul până când un exemplu arată convenabil.

Ce înseamnă să procesezi un fișier mare

Limita de octeți și limita de înregistrări controlează lucruri diferite. Un fișier cu puține câmpuri foarte lungi poate depăși limita de octeți. Un fișier cu multe rânduri scurte poate depăși limita de înregistrări. Ambele verificări sunt utile și niciuna nu o înlocuiește automat pe cealaltă. Ele țin experimentul într-un volum previzibil.

În implementarea noastră există mai multe reprezentări în memorie: octeții, textul decodat și listele rezultatelor. Limita de două milioane de octeți nu înseamnă că procesul consumă exact două milioane de octeți de memorie. Pentru volume mari ar trebui să procesăm incremental și să scriem rezultatele într-un spațiu intermediar, păstrând o strategie pentru erori și pentru publicarea rezultatului complet.

Un parser care citește incremental nu face singur întregul program eficient dacă apoi adaugi fiecare rând într-o listă uriașă. Urmărește unde sunt păstrate datele până la final. Pentru laborator, păstrarea listelor simplifică verificarea și este compatibilă cu limitele alese. Pentru un import cu milioane de înregistrări, designul ar necesita o altă analiză, inclusiv pentru memoria folosită de mulțimea identificatorilor.

O verificare independentă a sursei

Înainte și după rularea verficatorului, poți calcula hash-ul fișierului sau poți compara exact octeții unei copii de test. Ei trebuie să fie identici, deoarece funcția noastră doar citește. Dacă ai deschis sursa în modul w înainte de citire, ea poate fi deja trunchiată, chiar dacă programul se oprește ulterior cu o eroare. De aceea verificăm protejarea sursei printr-o probă, nu numai prin intenție.

La sfârșitul laboratorului, alege un caz care a eșuat și scrie explicația completă: formatul primit, regula încălcată, rezultatul observat și corectura sursei sau a codului. Nu modifica și sursa, și parserul în aceeași încercare. O singură schimbare controlată face diferența dintre depanare și încercări întâmplătoare care

produc, în cele din urmă, un fișier aparent acceptabil.

Cereri web cu requests

Laborator: matricea răspunsurilor

Rulează /cereri și /gol. Prima trebuie să producă trei cereri și 350.00, a doua zero cereri și 0.00. Ambele au cod de ieșire zero. Apoi rulează /schema, /duplicat, /invalid și /html. Toate trebuie să se încheie cu cod unu și fără mesaj de succes. Nu este acceptabil să întoarcă [] pentru fiecare eroare: ai ascunde diferența dintre lipsa datelor și o integrare stricată.

Rulează /eroare și o rută inexistentă. Statusurile 500 și 404 trebuie să fie tratate ca eșecuri HTTP. Rulează /lent și verifică oprirea clientului printr-o eroare de așteptare. Oprește serverul și repetă /cereri pentru eroarea de conexiune. Păstrează rezultatele separat; ele demonstrează probleme diferite, chiar dacă interfața simplifică unele mesaje.

Modifică temporar serverul astfel încât una dintre sume să fie numărul JSON 100, nu șirul "100". Contractul nostru cere text pentru a aplica aceeași conversie ca în CSV, deci răspunsul este respins. Modifică apoi doar numele pentru a include diacritice și o virgulă: acesta trebuie acceptat. Nu orice schimbare de reprezentare este invalidă; regula exactă decide.

Pentru limita de volum, construiește un răspuns cu 101 cereri și verifică respingerea. Pentru limita de octeți, trimite un corp mai mare de 65536 de octeți într-un mediu local și verifică faptul că nu se ajunge la raportul de succes. Testul trebuie să depășească efectiv limita relevantă, nu doar să folosească o listă cu câteva obiecte mari despre care presupui că ar fi suficientă.

Rezolvare și extensie controlată

Dacă /schema este acceptată, probabil ai presupus existența unei chei cereri fără să păstrezi contractul inițial. Decide dacă vrei să schimbi API-ul sau clientul, apoi actualizează toate probele. Dacă /invalid devine total zero, caută un except care întoarce o listă goală. Dacă /duplicat mărește totalul, verificarea identificatoarelor nu este aplicată înainte de agregare.

Extensia recomandată este să afișezi identificatorii acceptați după validarea întregului lot. Pentru exemplul inițial, ordinea este a1, a2, a4. Nu afișa rândurile pe măsură ce le validezi dacă apoi vrei să declari lotul invalid integral: ai produce o ieșire parțială ambiguă. Construiește rezultatul, verifică tot lotul și abia apoi publică raportul.

Predă server_demo.py, client_http.py, versiunea requests folosită și rezultatele matricei de teste. Explică de ce statusul 200 nu este suficient și de ce timeout-ul ales nu promite durată totală maximă. În proiectul final vom folosi sursa CSV pentru un flux repetabil fără rețea; clientul HTTP rămâne un adaptor separat pe care îl poți integra ulterior, păstrând aceleași reguli validate.

Studiu de caz: datele lipsesc sau integrarea este stricată?

Un raport zilnic arată zero cereri. Operatorul presupune că nu au venit solicitări, dar serviciul extern a schimbat structura răspunsului din listă într-un obiect cu cheia cereri. Clientul vechi a prins orice eroare și a întors o listă goală. Rezultatul pare legitim, însă ascunde o incompatibilitate. În clientul nostru, /gol demonstrează absența legitimă a datelor, iar /schema demonstrează încălcarea contractului.

Diferența trebuie păstrată și într-un sistem automat. Un rezultat gol valid poate continua spre generarea unui raport zero. O incompatibilitate trebuie să oprească publicarea sau să marcheze explicit că datele nu au fost obținute. Nu introduce aceeași valoare pentru „nu există date” și „nu știm câte date există”. Această distincție este una dintre cele mai importante reguli pentru rapoarte care depind de servicii externe.

Pentru investigație, notează momentul cererii, ruta, statusul și categoria erorii. Nu salva automat întregul corp dacă poate conține date confidențiale. Un identificator de cerere furnizat de serviciu poate ajuta la corelare, dacă documentația lui definește acel mecanism. În laborator nu avem asemenea identificatori, deci folosim ruta deterministă care a provocat cazul de test.

Paginarea este un protocol, nu doar o buclă

Multe API-uri întorc numai o parte din înregistrări. Documentația poate folosi numere de pagină, cursoare sau o adresă pentru următoarea pagină. Nu presupune că lista goală este mereu singurul semnal de final. Înainte să scrii o buclă, identifică forma răspunsului, semnalul de continuare și condiția de oprire. Clientul acestei lecții acceptă un singur lot de cel mult o sută de cereri; nu implementează paginare.

Dacă extinzi exemplul, stabilește o limită maximă de pagini și verifică repetarea unui cursor. Un server defect poate întoarce aceeași pagină la nesfârșit. Păstrează verificarea identificatorilor între pagini, nu doar în interiorul fiecărei pagini. Altfel, aceeași cerere poate fi numărată de două ori dacă datele se schimbă sau ordinea nu este stabilă în timpul parcurgerii.

Și consistența temporală trebuie discutată. Dacă noi cereri apar în timp ce citești paginile, un API bazat pe poziții poate muta înregistrări între pagini. O sumă rezultată din pagini diferite nu reprezintă neapărat un instantaneu al sistemului. Pentru un raport exact, caută un mecanism de snapshot sau o perioadă de raportare stabilă oferită de serviciu. Nu promite consistență pe care API-ul nu o garantează.

Într-un laborator de extensie poți simula două pagini fixe și un cursor repetat. Rezultatul așteptat pentru paginile fixe este reuniunea fără duplicate. Pentru cursorul repetat, clientul trebuie să se oprească printr-o eroare explicată. Nu introduce în proiectul final o buclă de descărcare netestată doar pentru a spune că aplicația folosește un API.

Formate numerice și contractul dintre sisteme

JSON poate reprezenta numere, texte, valori booleene, obiecte, liste și null. Prezența aceleiași informații vizuale nu înseamnă că tipul este același. În laborator, suma "100" este text și trece prin parserul nostru zecimal. Valoarea numerică 100 este respinsă de contractul rândului. Această decizie permite reutilizarea regulilor CSV și evită o conversie implicită diferită pentru fiecare sursă.

Un API real poate avea un contract mai bun pentru altă situație, de exemplu suma exprimată direct în bani întregi. Atunci adaptezi granița HTTP și verifici tipul exact, inclusiv excluderea valorilor booleene dacă nu sunt permise. Nu converți totul cu str doar ca să treacă validarea. Acea transformare poate ascunde faptul că furnizorul a schimbat tipul unui câmp obligatoriu.

Aceeași problemă apare pentru id. Un id numeric poate pierde zerourile inițiale înainte de a ajunge la client. Dacă identificatorul este definit ca text, păstrează-l ca text pe tot traseul. Dacă ai nevoie de comparație fără diferență de majuscule, acea regulă trebuie să vină din contractul serviciului, nu dintr-o presupunere a clientului care agregă datele.

Resursele se închid și la respingere

În client, verificarea Content-Type poate ridica o eroare înainte să citești corpul complet. Contextul with închide răspunsul și în această situație. Același lucru se întâmplă când limita de dimensiune este depășită. Dacă ai lăsa răspunsurile deschise în multe apeluri, ai putea acumula resurse și ai obține un comportament diferit după numeroase execuții față de prima probă.

Pentru mai multe cereri către același serviciu, o Session poate reutiliza conexiuni și setări, dar trebuie gestionată explicit. Nu este necesară pentru demonstrația cu o singură cerere și nu rezolvă automat retry-urile, validarea sau termenul total. Adaugă o astfel de optimizare după ce ai un contract și teste, apoi

verifică faptul că închiderea și tratarea erorilor rămân corecte.

În registrul de lucru separă ce ai executat de ce ai doar analizat. Rutele locale și erorile lor sunt probe executabile ale acestei lecții. Paginarea, autentificarea și integrarea cu un furnizor real sunt direcții de extindere, nu funcționalități deja demonstrate de program. Această delimitare face proiectul credibil și îl ajută pe următorul dezvoltator să știe ce mai trebuie construit.

Un script util cap-coadă

Laborator: probe de acceptare pentru proiect

Prima probă verifică exact rezultatele demonstrative și identitatea rândurilor acceptate. A doua rulează aceeași intrare într-un alt director nou și compară fișierele de date. În acest laborator nu introducem ora curentă în rezultate, astfel încât execuțiile cu aceeași octeți și aceeași versiune pot fi comparate direct. Calea directorului diferă, dar valorile și amprente fișierelor trebuie să coincidă în același mediu.

A treia probă încearcă reutilizarea directorului rezultat-01. Programul trebuie să eșueze fără să modifice fișierele existente. Calculează hash-urile înainte și după. Nu este suficient să verifici că s-a afișat o eroare: o implementare greșită ar putea modifica un fișier și abia apoi să descopere coliziunea. În codul nostru, crearea exclusivă a directorului precedă toate scrierile.

A patra probă folosește sursa însăși ca argument de ieșire. Fișierul existent nu poate deveni director, deci programul eșuează și sursa trebuie să rămână identică. A cincea probă folosește un antet greșit. Nu trebuie creat niciun director, fiindcă validarea întregii intrări precedă scrierea. A șasea probă folosește numai antetul corect și produce CSV-uri cu antet și fără rânduri, plus total zero.

Testează separat modul strict cu respingeri și cu o intrare complet validă. Verifică codurile unu, doi și trei prin situațiile potrivite: fișier lipsă, argumente lipsă și lot respins în modul strict. Nu cere unui singur test să demonstreze toate aceste cazuri. Fiecare probă trebuie să aibă o intrare și un rezultat așteptat clar.

Simulează un eșec fără să umpli discul

Pentru testare poți înlocui temporar funcția `scrie_json` într-un proces de test cu una care ridică `OSError` la scrierea rezumatului. Aceasta este o simulare controlată, nu un incident real de disc. Rulează generează într-un director temporar și verifică absența unui marcaj final valid. Păstrează această probă în test, nu în versiunea destinată utilizatorului.

Dacă programul afișează succes după eroarea simulată, ai prins excepția prea devreme sau ai ignorat-o. Dacă există marcajul înaintea fișierelor, ordinea de publicare este greșită. Dacă o nouă rulare suprascrie directorul parțial fără să avertizeze, ai pierdut protecția promisă. Corectează cauza și repetă probele de succes, ca remedierea să nu strice fluxul normal.

Pentru un fișier care depășește limita de octeți, așteptarea este oprirea înainte de scriere. Pentru mai mult de zece mii de înregistrări, parserul oprește lotul. Aceste limite sunt parte din contract și trebuie menționate în README. Nu promite procesarea oricărui export comercial doar pentru că fișierul de cinci rânduri a funcționat.

README-ul este parte din livrabil

Documentează versiunea Python testată, cele trei fișiere necesare, schema CSV, codificarea și comenzile pentru ambele moduri. Explică politica primei apariții valide a identificatorului, acceptarea sumei zero și respingerea formatelor cu virgulă sau exponent. Include un exemplu de intrare și cele patru fișiere așteptate, folosind numai date fictive.

Adaugă semnificația codurilor de ieșire și regula marcajului `FINALIZAT.json`. Scrie că o eroare la scriere poate lăsa un director parțial și că următoarea încercare trebuie să folosească o destinație nouă după investigare. Nu recomanda ștergerea automată a unui director necunoscut. Persoana care operează scriptul trebuie să știe ce verifică înainte să consume rezultatul.

Pentru programare automată, folosește căi explicite către interpretor, script, sursă și destinație. Asigură-te că fiecare execuție primește un nume de ieșire unic și că două procese nu încearcă să publice același director. Nu activa programarea înainte să poți reproduce manual o rulare și o eroare. Automatizarea repetă și

greșelile, nu doar pași corecți.

Evaluarea finală: explică o diferență de rezultat

Imaginează-ți că un coleg obține totalul 355.00 în loc de 350.00. Cere mai întâi hash-ul sursei și numărul de înregistrări, nu schimba imediat formula. Poate a corectat a3, poate a acceptat duplicatul sau poate a folosit o altă versiune a regulilor. Compară apoi identificatorii validați și erorile. Această ordine separă schimbarea datelor de o problemă a algoritmului.

Dacă hash-ul sursei este identic, dar ieșirea diferă, verifică versiunea codului și regulile. Dacă numai afișarea diferă, compară total_bani înaintea șirului formatat. Dacă amprenta unui fișier nu corespunde marcajului, rezultatul a fost modificat sau nu este complet. Fiecare dovadă răspunde unei întrebări diferite; niciuna nu trebuie prezentată ca dovadă universală a corectitudinii.

Ai încheiat proiectul când altcineva poate rula programul din README, obține rezultatele de control, provoacă o eroare documentată și verifică faptul că sursa a rămas neschimbată. Mai presus de lungimea codului, ai construit un flux care poate fi explicat: intrare identificabilă, reguli într-un singur loc, respingeri vizibile, rezultate controlate și limite declarate.

O predare repetată într-un mediu nou

Copiază numai sursele, fișierul de intrare și README-ul într-un director de laborator nou. Creează un mediu virtual nou și rulează raportul fără să te bazezi pe fișierele generate anterior. Dacă proiectul funcționează numai în directorul în care l-ai construit, poate depinde de un modul uitat, de un fișier temporar sau de o cale implicită. Proba de recreare găsește aceste dependențe ascunse.

Pentru raportul final, nu trebuie instalat requests. Dacă importul bibliotecii de rețea este necesar doar ca să citești CSV, ai legat inutil componentele. Păstrează client_http.py ca adaptor separat. Un proiect mic este mai ușor de operat când fiecare comandă are numai dependențele necesare funcției ei, iar documentația arată clar care exerciții cer pachete externe.

Verifică importarea lui raport într-un proces nou. Nu trebuie să citească argumentele terminalului, să creeze un director sau să afișeze un total. Apoi apelează generează cu căi dintr-un director temporar. Această probă demonstrează că aceeași logică poate fi folosită de o altă interfață fără să pornească automat comanda originală.

Ce verifică amprentele și ce rămâne în afara lor

Dacă modifichi un singur caracter în valide.csv după generare, hash-ul lui trebuie să difere de cel din FINALIZAT.json. Acesta este un test simplu pentru verificatorul rezultatului. Dacă modifichi și marcajul în mod intenționat, comparația locală poate trece din nou. De aceea amprentele din același director nu protejează împotriva unui actor care poate rescrie întregul rezultat; ele sunt folosite aici pentru detectarea schimbărilor și a incompletitudinii.

Pentru audit mai puternic, o aplicație reală poate păstra metadatele într-un sistem separat cu drepturi și politici proprii. Aceasta este o altă cerință, care nu trebuie sugerată ca fiind rezolvată de exemplul de curs. În laborator, criteriul este mai restrâns și verificabil: fișierele finale corespund intrării și regulilor documentate, iar modificările ulterioare pot fi observate prin comparația cu marcajul păstrat.

La autoevaluare, alege o limită a programului și explică ce ar necesita extinderea ei. Pentru fișiere mai mari, discută memoria și stocarea intermediară. Pentru publicare atomică, discută promovarea rezultatului și recuperarea după întreruperi. Pentru surse HTTP, discută snapshot-ul, paginarea și erorile de transport. Nu adăuga aceste funcționalități prin câteva linii neverificate doar pentru a face lista de caracteristici mai lungă.

Anexă: fișierele executabile de laborator

salut.py

```
nume = "Ana"  
print(f"Salut, {nume}!")
```

diagnostic.py

```
from pathlib import Path
import sys

print("Interpreter:", sys.executable)
print("Director curent:", Path.cwd())
print("Fişier:", Path(__file__).resolve())
print("Mediu virtual:", sys.prefix != sys.base_prefix)
```

intreaba.py

```
nume = input("Cum te numești? ").strip()
if not nume:
    raise SystemExit("Introdu un nume nevid.")
print(f"Salut, {nume}!")
```

plan.py

```
import argparse

parser = argparse.ArgumentParser(
    description="Planifică sesiuni de lectură de 25 minute."
)
parser.add_argument("nume")
parser.add_argument("--sesiuni", type=int, default=1)
args = parser.parse_args()

nume = args.nume.strip()
if not nume:
    parser.error("Numele trebuie să conțină text.")
if not 1 <= args.sesiuni <= 10:
    parser.error("Sesiunile trebuie să fie între 1 și 10.")

minute = args.sesiuni * 25
print(f"{nume}: {args.sesiuni} sesiuni, {minute} minute.")
```

calcul.py

```
import argparse
import re
from decimal import Decimal, ROUND_HALF_UP

parser = argparse.ArgumentParser(description="Calcul de atelier.")
parser.add_argument("pret")
parser.add_argument("cantitate")
parser.add_argument("procent")
args = parser.parse_args()

pret_text = args.pret.strip()
cantitate_text = args.cantitate.strip()
procent_text = args.procent.strip()

if not re.fullmatch(r"[0-9]{1,5}(\.[0-9]{1,2})?", pret_text):
    parser.error("Preț: cifre și opțional punct cu 1-2 zecimale.")
if not re.fullmatch(r"[0-9]{1,3}", cantitate_text):
    parser.error("Cantitate: 1-3 cifre, fără semne.")
if not re.fullmatch(r"[0-9]{1,3}(\.[0-9]{1,2})?", procent_text):
    parser.error("Procent: cifre și cel mult două zecimale.")

pret = Decimal(pret_text)
cantitate = int(cantitate_text)
procent = Decimal(procent_text)
if not pret.is_finite() or not Decimal("0") <= pret <= Decimal("10000"):
    parser.error("Prețul trebuie să fie între 0 și 10000.")
if not 1 <= cantitate <= 100:
    parser.error("Cantitatea trebuie să fie între 1 și 100.")
if not procent.is_finite() or not Decimal("0") <= procent <= Decimal("100"):
    parser.error("Procentul trebuie să fie între 0 și 100.")

subtotal = pret * cantitate
economie = (subtotal * procent / Decimal("100")).quantize(
    Decimal("0.01"), rounding=ROUND_HALF_UP
)
total = subtotal - economie
print(f"Subtotal: {subtotal:.2f}")
print(f"Economie: {economie:.2f}")
print(f"Total: {total:.2f}")
```

reguli.py

```
# reguli.py
import re

SUMA = re.compile(r"[0-9]{1,6}(\.[0-9]{1,2})?")
IDENT = re.compile(r"[A-Za-z0-9_]{1,32}")

def bani_din_text(text: str) -> int:
    """Convertește o sumă nenegativă cu cel mult două zecimale."""
    if not isinstance(text, str):
        raise TypeError("Suma trebuie primită ca text")
    text = text.strip()
    if SUMA.fullmatch(text) is None:
        raise ValueError("Suma trebuie scrisă cu cifre și punct opțional")
    integ, separator, fractie = text.partition(".")
    return int(integ) * 100 + int(fractie.ljust(2, "0"))

def text_din_bani(bani: int) -> str:
    """Formatează bani întregi nenegativi fără conversie la float."""
    if type(bani) is not int:
        raise TypeError("Banii trebuie să fie întregi, nu bool")
    if bani < 0:
        raise ValueError("Banii nu pot fi negativi")
    return f"{bani // 100}.{bani % 100:02d}"

def valideaza_rand(rand: dict) -> dict:
    """Întoarce un dicționar nou; sursa nu este modificată."""
    if not isinstance(rand, dict) or set(rand) != {"id", "nume", "suma"}:
        raise ValueError("Coloane invalide")
    if not all(isinstance(v, str) for v in rand.values()):
        raise ValueError("Câmpurile trebuie să fie text")
    ident = rand["id"].strip()
    nume = rand["nume"].strip()
    if IDENT.fullmatch(ident) is None:
        raise ValueError("ID: 1-32 litere ASCII, cifre, _ sau -")
    if not 1 <= len(nume) <= 80:
        raise ValueError("Numele trebuie să aibă 1-80 caractere")
    return {
        "id": ident,
        "nume": nume,
        "suma_bani": bani_din_text(rand["suma"]),
    }

if __name__ == "__main__":
    print(text_din_bani(bani_din_text("12.50")))
```

demo_reguli.py

```
# demo_reguli.py
from reguli import valideaza_rand, text_din_bani

sursa = {"id": " a1 ", "nume": " Ștefan ", "suma": "12.50"}
rezultat = valideaza_rand(sursa)
print(rezultat["id"], rezultat["nume"])
print(text_din_bani(rezultat["suma_bani"]))
print(sursa["id"] == " a1 ")
```

import_csv.py

```
# import_csv.py
import csv
import io
from pathlib import Path
from reguli import valideaza_rand

ANTET = ["id", "nume", "suma"]
MAX_OCTETI = 2_000_000
MAX_RANDURI = 10_000

def citeste_cereri(caie: Path) -> tuple[list, list]:
    with caie.open("rb") as fisier:
        brut = fisier.read(MAX_OCTETI + 1)
    if len(brut) > MAX_OCTETI:
        raise ValueError("Fișierul depășește limita de octeți")
    try:
        text = brut.decode("utf-8-sig")
    except UnicodeDecodeError as exc:
        raise ValueError("Fișierul nu este UTF-8 valid") from exc

    cititor = csv.reader(io.StringIO(text, newline=""), strict=True)
    valide, erori = [], []
    vazute = set()
    try:
        if next(cititor, None) != ANTET:
            raise ValueError("Antetul trebuie să fie id,nume,suma")
        for numar, valori in enumerate(cititor, start=2):
            if numar - 1 > MAX_RANDURI:
                raise ValueError("Prea multe înregistrări")
            try:
                if len(valori) != len(ANTET):
                    raise ValueError("Număr incorect de câmpuri")
                rand = valideaza_rand(dict(zip(ANTET, valori)))
                if rand["id"] in vazute:
                    raise ValueError("Identificator duplicat")
            except ValueError as exc:
                erori.append({"inregistrare": numar, "motiv": str(exc)})
                continue
            vazute.add(rand["id"])
            valide.append(rand)
    except csv.Error as exc:
        raise ValueError("Sintaxă CSV invalidă") from exc
    return valide, erori
```

verifica_csv.py

```
# verifica_csv.py
import argparse
import sys
from pathlib import Path
from import_csv import citeste_cereri
from reguli import text_din_bani

parser = argparse.ArgumentParser(description="Verifică un CSV de cereri.")
parser.add_argument("intrare", type=Path)
args = parser.parse_args()
try:
    valide, erori = citeste_cereri(args.intrare)
except (OSError, ValueError) as exc:
    print(f"Import oprit: {exc}", file=sys.stderr)
    raise SystemExit(1)
print("Valide:", len(valide))
print("Respinse:", len(erori))
print("Total:", text_din_bani(sum(r["suma_bani"] for r in valide)))
for eroare in erori:
    print(eroare["inregistrare"], eroare["motiv"])
```

exemplu_scriere.py

```
# exemplu_scriere.py
import csv
from pathlib import Path

cale = Path("exemplu-nou.csv")
with cale.open("x", encoding="utf-8", newline="") as fisier:
    writer = csv.DictWriter(fisier, fieldnames=["id", "nume", "suma"])
    writer.writeheader()
    writer.writerow({"id": "t1", "nume": 'Ștefan, "demo"', "suma": "12.50"})
with cale.open(encoding="utf-8", newline="") as fisier:
    inapoi = list(csv.DictReader(fisier))
assert inapoi[0]["nume"] == 'Ștefan, "demo"'
print(inapoi[0]["suma"])
```

server_demo.py

```
# server_demo.py
import json
import time
from http.server import BaseHTTPRequestHandler, ThreadingHTTPServer
from urllib.parse import urlsplit

CERERI = [
    {"id": "a1", "nume": "Ștefan", "suma": "100"},
    {"id": "a2", "nume": "Atelier, demo", "suma": "250.00"},
    {"id": "a4", "nume": "Ioana", "suma": "0"},
]

class Handler(BaseHTTPRequestHandler):
    def do_GET(self):
        ruta = urlsplit(self.path).path
        status, tip, date = 200, "application/json", CERERI
        if ruta == "/gol":
            date = []
        elif ruta == "/schema":
            date = {"cereri": CERERI}
        elif ruta == "/duplicat":
            date = CERERI + [CERERI[0]]
        elif ruta == "/lent":
            time.sleep(0.6)
        elif ruta == "/eroare":
            status, date = 500, {"eroare": "Simulare"}
        elif ruta == "/html":
            tip, date = "text/html", "<p>Pagina de test</p>"
        elif ruta == "/invalid":
            date = None
        elif ruta != "/cereri":
            status, date = 404, {"eroare": "Resursa lipsește"}
        if ruta == "/invalid":
            corp = b'{"neterminat": '
        elif tip == "text/html":
            corp = date.encode("utf-8")
        else:
            corp = json.dumps(date, ensure_ascii=False).encode("utf-8")
        self.send_response(status)
        self.send_header("Content-Type", tip + "; charset=utf-8")
        self.send_header("Content-Length", str(len(corp)))
        self.end_headers()
        try:
            self.wfile.write(corp)
        except (BrokenPipeError, ConnectionResetError):
            pass # Clientul poate abandona deliberat cererea lentă.

    def log_message(self, format, *args):
        return

if __name__ == "__main__":
    ThreadingHTTPServer(("127.0.0.1", 8765), Handler).serve_forever()
```

client_http.py

```
# client_http.py
import argparse
import json
import sys
import requests
from reguli import valideaza_rand, text_din_bani

def descarca_cereri(url: str) -> list:
    with requests.get(
        url, timeout=(1, 0.2), stream=True, allow_redirects=False
    ) as raspuns:
        raspuns.raise_for_status()
        if raspuns.status_code != 200:
            raise ValueError("Se acceptă numai status 200")
        tip = raspuns.headers.get("Content-Type", "").split("; ", 1)[0]
        if tip.strip().lower() != "application/json":
            raise ValueError("Se așteaptă application/json")
        corp = bytearray()
        for fragment in raspuns.iter_content(chunk_size=4096):
            corp.extend(fragment)
            if len(corp) > 65536:
                raise ValueError("Răspuns prea mare")
        date = json.loads(corp.decode("utf-8"))
        if not isinstance(date, list) or len(date) > 100:
            raise ValueError("Se așteaptă o listă de cel mult 100 cereri")
        rezultat, vazute = [], set()
        for rand in date:
            valid = valideaza_rand(rand)
            if valid["id"] in vazute:
                raise ValueError("Identificator duplicat în răspuns")
            vazute.add(valid["id"])
            rezultat.append(valid)
        return rezultat

def main() -> int:
    parser = argparse.ArgumentParser(description="Client HTTP de laborator.")
    parser.add_argument("url")
    args = parser.parse_args()
    try:
        cereri = descarca_cereri(args.url)
    except requests.Timeout:
        print("Timp de așteptare depășit", file=sys.stderr)
        return 1
    except requests.RequestException:
        print("Cerere HTTP eșuată", file=sys.stderr)
        return 1
    except (ValueError, UnicodeError):
        print("Răspunsul nu respectă formatul cerut", file=sys.stderr)
        return 1
    total = sum(r["suma_bani"] for r in cereri)
    print(f"Cereri: {len(cereri)}; total: {text_din_bani(total)}")
    return 0

if __name__ == "__main__":
    raise SystemExit(main())
```

raport.py

```
# raport.py
import argparse
import csv
import hashlib
import json
import sys
import tempfile
from pathlib import Path
from import_csv import citeste_cereri, MAX_OCTETI
from reguli import text_din_bani

VERSIUNE = "1.0"

def scrie_json(cale: Path, date) -> None:
    with cale.open("x", encoding="utf-8", newline="\n") as fisier:
        json.dump(date, fisier, ensure_ascii=False, indent=2)
        fisier.write("\n")

def scrie_csv(cale: Path, coloane: list, randuri: list) -> None:
    with cale.open("x", encoding="utf-8", newline="") as fisier:
        writer = csv.DictWriter(fisier, fieldnames=coloane)
        writer.writeheader()
        writer.writerows(randuri)

def genereaza(intrare: Path, iesire: Path, strict: bool = False) -> dict:
    with intrare.open("rb") as fisier:
        brut = fisier.read(MAX_OCTETI + 1)
    if len(brut) > MAX_OCTETI:
        raise ValueError("Fișierul depășește limita de octeți")
    with tempfile.TemporaryDirectory(prefix="cereri-") as temporar:
        copie = Path(temporar) / "intrare.csv"
        copie.write_bytes(brut)
        valide, erori = citeste_cereri(copie)
    if strict and erori:
        return {"oprit_strict": True, "respinse": len(erori)}

    total = sum(rand["suma_bani"] for rand in valide)
    rezumat = {
        "versiune": VERSIUNE,
        "strict": strict,
        "sursa_sha256": hashlib.sha256(brut).hexdigest(),
        "inregistrari": len(valide) + len(erori),
        "valide": len(valide),
        "respinse": len(erori),
        "total_bani": total,
        "total": text_din_bani(total),
    }
    export = [
        {"id": r["id"], "nume": r["nume"],
         "suma": text_din_bani(r["suma_bani"])}
        for r in valide
    ]
    iesire.mkdir(exist_ok=False)
    scrie_csv(iesire / "valide.csv", ["id", "nume", "suma"], export)
    scrie_csv(iesire / "erori.csv", ["inregistrare", "motiv"], erori)
    scrie_json(iesire / "rezumat.json", rezumat)
    fisiere = ["valide.csv", "erori.csv", "rezumat.json"]
    amprente = {
        nume: hashlib.sha256((iesire / nume).read_bytes()).hexdigest()
        for nume in fisiere
    }
    scrie_json(iesire / "FINALIZAT.json", {"sha256": amprente})
    return rezumat

def main() -> int:
    parser = argparse.ArgumentParser(description="Raport CSV din cereri fictive.")
    parser.add_argument("intrare", type=Path)
```

```

parser.add_argument("iesire", type=Path)
parser.add_argument("--strict", action="store_true")
args = parser.parse_args()
try:
    rezultat = genereaza(args.intrare, args.iesire, args.strict)
except (OSError, ValueError, csv.Error) as exc:
    print(f"Raport nefinalizat: {exc}", file=sys.stderr)
    return 1
if rezultat.get("oprit_strict"):
    print(f"Oprit: {rezultat['respins']} înregistrări respins.",
          file=sys.stderr)
    return 3
print(f"Valide: {rezultat['valide']}; respins: {rezultat['respins']}; "
      f"total: {rezultat['total']}")
return 0

if __name__ == "__main__":
    raise SystemExit(main())

```

cereri.csv și comanda finală

```
id,nume,suma
a1,"Atelier, demo",100
a2,Ștefan,250.00
a3,Mara,invalid
a4,Ioana,0
a1,Duplicat,5
```

Rulează raport.py cereri.csv rezultat-01 prin interpretorul mediului virtual. Directorul rezultat-01 nu trebuie să existe. Cu --strict, intrarea de mai sus este respinsă înainte de scriere. Pentru laboratorul HTTP, pornește server_demo.py separat și rulează client_http.py <http://127.0.0.1:8765/cereri>.