

Analiza impactului programei de informatică bazată exclusiv pe limbajul Python asupra înțelegerii conceptelor de bază de programare

Context

România deține un avantaj competitiv solid, construit istoric și confirmat internațional, fiind recunoscută ca un important furnizor de ingineri software cu pregătire riguroasă în algoritmică, sisteme și optimizare. Menținerea și consolidarea acestui avantaj presupun dezvoltarea continuă a unei mase critice de specialiști cu profil autentic de *engineer*. Învățământul preuniversitar de informatică a avut un rol determinant în această direcție, programele actuale contribuind semnificativ la formarea gândirii computaționale a absolvenților de la specializarea matematică-informatică.

În contextul în care România dezvoltă tehnologie în multiple sectoare, iar industria IT&C generează peste 8% din PIB și prezintă perspective clare de creștere, investiția în formarea specialiștilor din domeniu reprezintă o opțiune strategică. Aceasta este esențială pentru asigurarea securității economice a țării pe termen lung.

Elemente de noutate în programa clasei a IX-a disciplina Informatică

Analizând comparativ programele actuale cu programele lansate în dezbateră publică pentru clasa a IX-a, specializările matematică-informatică, matematică-informatică intensiv și științele naturii, **elementul definitoriu de noutate** propus constă în **utilizarea limbajului Python ca unic instrument** pentru introducerea conceptelor de programare, inclusiv liste, subprograme, fișiere, interfețe grafice și programare orientată pe obiecte (POO).

Deși Python este un limbaj popular și versatil, pe care îl considerăm extrem de util și potrivit pentru a construi exemple complexe de programe cu scop specific (de exemplu jocuri cu grafică sau inteligență artificială), considerăm că acesta trebuie să apară într-o etapă ulterioară a procesului de învățare, în care **conceptele de bază de programare** (de exemplu, variabile, tipuri de date, tablouri – vectori și matrici, fișiere, etc) **sunt deja sedimentate**, iar **gândirea critică** predomină în rezolvarea problemei date.

În **Anexa 1** prezentăm o analiză detaliată a programei puse în dezbateră, care să demonstreze faptul că alegerea acestui limbaj pentru a preda conceptele în forma și ordinea din această programă ridică probleme didactice semnificative, împiedicând înțelegerea naturală a conceptelor de bază ale programării, dar și limitarea sau lipsa abordării unora dintre concepte, ceea ce duce la superficialitate.

În continuare rezumăm din perspectivă psihopedagogică principalele argumente.

1. Lipsa alinierii la stadiul de dezvoltare cognitivă al elevilor de 14–15 ani

Conform modelelor consacrate ale dezvoltării cognitive (ex. Inhelder & Piaget, 1958; Case, 1992; Sweller, 1994), elevii din clasele de început ale liceului se află în faza de consolidare a gândirii logico-formale, nu în faza de manipulare naturală a conceptelor abstracte de nivel înalt. Programa introduce elemente care necesită:

- manipularea simultană a mai multor niveluri de abstracție (tipuri dinamice, obiecte, instanțe, referințe, structuri dinamice);

- operarea cu modele mentale asupra unor mecanisme invizibile (gestionarea automată a memoriei, tipuri implicite, conversii).

Aceste cerințe depășesc capacitatea cognitivă tipică pentru acest nivel de vârstă, generând suprasolicitare (*cognitive overload*), învățare superficială și confuzie conceptuală.

Didactica programării, fundamentată în ultimele două decenii (Robins, Rountree & Rountree, 2003; Lister, 2011; Luxton-Reilly, 2016), recomandă introducerea programării prin: construcție graduală a conceptelor, transparență a mecanismelor, reducerea complexității inițiale, înțelegerea execuției programului înainte de abstractizare.

2. Caracteristicile limbajului Python maschează concepte fundamentale

Python este accesibil, dar ascunde mecanisme esențiale pentru formarea gândirii algoritmice. De exemplu:

- citirea datelor produce întotdeauna text, impunând înțelegerea conversiilor de tip înainte ca elevul să înțeleagă noțiunea de tip de date;
- gestionarea automată a numerelor mari elimină percepția asupra limitelor calculatoarelor;
- orice variabilă este un obiect, chiar dacă elevul încă nu cunoaște noțiunile de referință, instanțiere sau spațiu de memorie.

Această opacitate face dificilă construirea unor modele mentale corecte, aspect esențial în primele etape ale învățării programării.

3. Introducerea prematură a Programării Orientate pe Obiecte

POO este un cadru conceptual care presupune abstractizare, clasificare, modelare și relaționare între entități – procese cognitive de nivel înalt.

Literatura de specialitate subliniază că POO este dificilă chiar și la nivel universitar pentru studenții începători. Introducerea sa la clasa a IX-a, fără un fundament solid al programării structurate, contravine recomandărilor din literatura de specialitate și generează suprasolicitare cognitivă. Elevii aflați încă în procesul de învățare a funcțiilor nu au resursele necesare pentru a integra noțiuni precum clase, metode și instanțe.

4. Suprasarcina curriculară în raport cu timpul disponibil

Două ore pe săptămână sunt insuficiente pentru parcurgerea unui volum atât de complex, respectiv consolidarea cunoștințelor și exersarea prin activități practice.

Prea multe concepte introduse prea rapid duc la fragmentare, pierderea coerenței și imposibilitatea asigurării unei învățări semnificative.

La nivel introductiv, o înțelegere solidă a conceptelor fundamentale de programare presupune expunerea elevilor la numeroase activități aplicative și la exersarea sistematică a strategiilor de rezolvare a problemelor în contexte variate. Programa aflată în dezbatere include însă un volum considerabil de concepte teoretice — adesea implicite sau insuficient explicitate — în timp ce oferă relativ puțini algoritmi și exemple practice care să faciliteze transferul acestor concepte în activități concrete de învățare.

Trebuie să menționăm că **suprasarcina curriculară** este semnificativ mai mare pentru specializarea **Științele naturii**. Deși în planul cadru disciplina Informatică are alocată o singură oră la această specializare, în timp ce la specializarea Matematică-informatică alocarea este de 2 ore, diferențele dintre programele celor două specializări sunt minore (necesitând cel mult 5 ore), singurele elemente de conținut suplimentare la matematică-informatică fiind:

- noțiunile de coadă și stivă (doar la nivel de concept, fără operații specifice cu aceste structuri de date);
- transformarea unui număr dintr-o bază de numerație în altă bază de numerație;
- sortarea elementelor unei liste prin metoda bulelor.

5. Abordare excesiv de orientată teoretic, în detrimentul aplicabilității practice

Structura actuală a programei este excesiv de orientată spre componenta teoretică, în detrimentul aplicabilității practice. Chiar și sugestiile metodologice — care precizează că *"activitățile din cadrul instruirii teoretice se desfășoară, de regulă, în săli de clasă, dotate cu tablă interactivă, pentru exemplificarea programelor pe calculator, iar activitățile din cadrul instruirii practice se desfășoară, de regulă, în laboratorul de informatică"* — subliniază indirect această disproporție. În predarea informaticii, fiecare concept teoretic trebuie consolidat prin aplicare imediată în contexte diverse, astfel încât învățarea să devină funcțională și relevantă.

Demersul didactic trebuie să fie centrat pe elev, vizând formarea capacității acestuia de a transfera și aplica conceptele teoretice în situații practice. Dacă accentul lecției rămâne predominant pe demonstrațiile profesorului, rezultatul va fi, în cel mai bun caz, dezvoltarea competenței elevilor de a înțelege soluțiile prezentate, nu și de a le reproduce sau adapta autonom în contexte noi. Din acest motiv, **desfășurarea orelor de informatică în laborator este esențială** pentru formarea competențelor specifice domeniului.

Concluzie

Din perspectivă pedagogică și cognitivă, ordinea și nivelul de complexitate propuse de programa actuală nu sunt optim adaptate profilului elevilor de clasa a IX-a. Un demers didactic care introduce rapid concepte avansate, fără fundamentare, riscă să formeze **modele mentale incorecte despre funcționarea programelor**.

Construcția competențelor de programare are nevoie de o abordare secvențială, transparentă și adecvată vârstei, în care conceptele fundamentale sunt consolidate înaintea celor avansate. Reechilibrarea programei în această direcție ar crește semnificativ eficiența procesului de învățare și ar asigura premisele pentru o formare autentică în domeniul informaticii.

Soluția propusă – Progresia $C^{++} \rightarrow C^{++}$ și Python $\rightarrow$ Python și SQL

Anexa 2 conține o propunere schematică de programă pentru disciplina informatică pentru specializarea matematică-informatică, cu diferențiere pentru matematică informatică intensiv pentru clasele IX-XII. Documentul include competențele generale vizate și conținuturile propuse, cu o detaliere orientativă a alocării orare, pentru a valida fezabilitatea atingerii competențelor specifice în bugetul de timp alocat.

Principiile programei propuse

- Propunerea de programă pe care o prezentăm este structurată pe principiul progresiei graduale și al relevanței în raport cu finalitățile educaționale și cu cerințele pieței muncii.
- Programa urmărește formarea unei gândiri computaționale și o înțelegere profundă a conceptelor fundamentale ale programării pentru toți elevii.
- Programa are o construcție organică, pornind de la fundamente și evoluând natural spre aplicabilitate practică.

Structură generală a programei propuse, din perspectiva limbajelor de programare

Clasele a IX-a și a X-a: limbajul C++

Limbajul C++ este utilizat pentru formarea riguroasă a gândirii algoritmice și pentru înțelegerea aprofundată a mecanismelor fundamentale ale programării: variabile, tipuri de date, structuri de control, funcții, pointeri, recursivitate, structuri de date.

Clasa a XI-a: C++ și Python

În această etapă, se continuă aprofundarea algoritmicii și a structurilor de date complexe prin C++. În egală măsură, se introduce limbajul Python, deoarece, la ora actuală, Python este un limbaj dominant în domenii moderne: Inteligență Artificială, Data Science, Cloud, automatizare. Abordarea limbajului Python va fi accesibilă pentru elevii de clasa a XI-a, care au deja formate competențe în domeniul algoritmicii și programării prin intermediul limbajului C++.

Clasa a XII-a: Python și SQL

În ultimul an de liceu, accentul se mută de la algoritmica fundamentală spre aplicabilitate și interdisciplinaritate. Python este utilizat pentru introducerea în domeniul Inteligenței Artificiale, prin biblioteci specializate (NumPy, pandas, Matplotlib, scikit-learn), care permit experimentarea directă cu concepte precum regresie, clasificare, rețele neuronale. În paralel, limbajul SQL este introdus pentru studiul bazelor de date, reprezentând standardul industriei și un instrument indispensabil în orice carieră din domeniul IT.

Progresia C++ → C++ și Python → Python și SQL asigură echilibrul dintre rigoarea algoritmică și aplicabilitatea practică, oferindu-le elevilor atât competențe fundamentale (structuri de date, optimizare, gândire algoritmică), cât și abilități moderne, aliniate cu cerințele actuale ale pieței muncii și ale învățământului superior de profil.

C++ versus Python – avantaje versus riscuri

Concept	C++	Python	Impact educațional
Citirea datelor	Explicită, tipuri fixe	Totul ca șir de caractere	În C++, elevul învață structuri și tipuri înainte de concepte avansate.
Rigoare	Strict, erori obligatoriu rezolvate	Flexibil, codul poate funcționa chiar cu erori	În C++, atenție la detalii și disciplină algoritmică. În Python, elevii dezvoltă mentalitatea „merge și așa”, ceea ce slăbește atenția și ordonarea gândirii.
Structuri de date	Tablouri fixe, clare	Liste de liste, dicționare, clase	C++ dezvoltă înțelegerea logicii interne. În Python concepte fundamentale ale programării orientate pe obiect sau structurilor de date avansate sunt introduse prea devreme, la un nivel superficial, ceea ce împiedică înțelegerea solidă a fundamentelor disciplinei.
Mutabilitate	Clară, explicită	Amestec de mutabile și imutabile	C++ învață gestionarea memoriei și referințelor. Python poate crea confuzie în manipularea datelor și referințelor.

Perspective pe termen lung

Limbajele de programare sunt volatile. Python este un limbaj apărut recent comparativ cu C-ul, în timp ce limbajele C și C++ sunt folosite în industrie și în educație de peste 50 de ani. Ele au rezistat de-a lungul timpului și ne oferă certitudinea că vor rămâne relevante și peste 15–20 de ani. Informatica la liceu nu trebuie să urmărească moda sau limbajele «la modă». Scopul nostru este să formăm concepte solide și gândire algoritmică. Dacă construim pe un teren schimbător, cum este Python-ul, riscăm să formăm elevi care înțeleg limbajul de azi, dar nu au fundamentele necesare pentru viitor.

Un alt avantaj al C și C++ este că formează rapid capacitatea de adaptare. Majoritatea limbajelor moderne – Java, C#, Go sau Rust – sunt inspirate direct din C/C++. Elevul care învață C++ înțelege ușor aceste limbaje și poate să se adapteze rapid la cerințele industriei.

Formarea solidă începe cu un fundament stabil. Alegerea limbajului C/C++ nu este despre nostalgie, ci despre responsabilitate: este despre a pregăti programatori care pot înțelege, construi și inova, indiferent de ce limbaj va fi la modă mâine.

Într-un timp relativ scurt limbajele vor fi înlocuite de interfețele conversaționale. Aplicațiile vor fi realizate fără a scrie cod într-un limbaj, ci adresându-ne AI-ului în limbaj natural. Întrebarea ce urmează firesc este “Cum să conversezi cu AI pentru crearea unor aplicații dacă nu ai o înțelegere profundă a mecanismelor fundamentale ale programării?”

Rezumând, C++ oferă rigurozitate, claritate și disciplină, iar Python trebuie folosit complementar, nu ca înlocuitor. Susținem utilizarea adecvată, utilă și eficientă a limbajului Python pentru dezvoltarea de aplicații în domeniile pentru care acesta a fost conceput. În același timp, considerăm că limbajul C++ trebuie valorificat în mod prioritar pentru consolidarea fundamentelor programării.

Matematică-informatică intensiv vs neintensiv

Programele școlare aflate în prezent în dezbatere publică prevăd introducerea limbajului C++ în clasa a IX-a la specializarea Informatică intensiv, în paralel cu studiul limbajului Python. Această abordare generează o suprasarcină cognitivă pentru elevii de la profilul intensiv, chiar dacă aceștia beneficiază de un număr dublu de ore față de colegii lor de la neintensiv. Este important de subliniat că ponderea elevilor înscriși în clase cu regim intensiv este relativ redusă la nivel național.

În prezent, facultățile de profil din România (informatică, automatică și calculatoare, cibernetică) formează aproximativ 10.000 de studenți anual. Totuși, pentru perioada 2026–2035, proiecțiile arată că economia digitală va necesita un număr semnificativ mai mare de specialiști IT cu formare de nivel *engineer*. În acest context, devine esențial ca învățământul preuniversitar să consolideze dezvoltarea gândirii computaționale în rândul elevilor de la specializarea matematică-informatică neintensiv. Formarea competențelor specifice disciplinei Informatică este nu doar o premisă pentru accesul elevilor la cariere în domeniul IT, ci și o resursă intelectuală valoroasă pentru orice domeniu tehnic, precum și pentru alte domenii, de exemplu științele medicale.

Concluzie – misiunea educațională

România trebuie să formeze tineri capabili să definească viitorul tehnologic, nu doar utilizatori de software. Folosind exclusiv Python, riscăm să formăm elevi care „se descurcă azi”, dar nu au baza necesară pentru viitor, formăm simpli utilizatori de cod din biblioteci.

În cele din urmă, adevărata provocare este aceasta: ce vrem ca școala să construiască pentru România – simpli utilizatori de software sau tineri capabili să definească viitorul acestei țări prin inovație sau, și mai mult, creatori de tehnologie?

Anexa 1.

Analiza detaliată a impactului programei de informatică bazată exclusiv pe limbajul Python asupra înțelegerii conceptelor de bază de programare

Programa de informatică pentru clasa a IX-a (specializarea matematică-informatică - neintensiv și intensiv) propune utilizarea limbajului Python pentru introducerea conceptelor de programare, inclusiv liste, subprograme, fișiere, interfețe grafice și programare orientată pe obiecte (POO). Deși Python este un limbaj popular și versatil, pe care îl considerăm extrem de util și potrivit pentru a construi exemple complexe de programe cu scop specific (de exemplu jocuri cu grafică sau inteligență artificială), considerăm că acesta trebuie să apară într-o etapă ulterioară a procesului de învățare, în care **conceptele de bază de programare** (de exemplu, variabile, tipuri de date, tablouri - vectori și matrici, fișiere, etc) **sunt deja sedimentate**, iar **gândirea critică** predomină în rezolvarea problemei date.

Propunem o analiză detaliată a programei publicate, care să demonstreze faptul că alegerea acestui limbaj pentru a preda conceptele în forma și ordinea din această programă ridică probleme didactice semnificative, împiedicând înțelegerea naturală a conceptelor de bază ale programării, dar și limitarea sau lipsa abordării unora dintre concepte, ceea ce duce la superficialitate.

1. Absența unei introduceri solide în limbajul de programare suport

Programa propusă **nu** evidențiază sau **nu** introduce sistematic elementele fundamentale ale limbajului, ceea ce poate genera confuzie pentru elevii aflați la început de drum. În mod particular, sunt neglijate următoarele aspecte:

a) Concepte / noțiuni fundamentale:

- **Conceptul de variabilă și modul de lucru al memoriei** - elevii trebuie să înțeleagă că variabilele nu sunt simple etichete, ci spații în memorie care stochează valori.
- **Conceptele despre tipurile de bază și caracteristicile lor** - diferențierea între `int`, `float`, `string` etc., precum și comportamentul specific al fiecărui tip în operații.
- **Diferența dintre valoare și referință în Python** - înțelegerea modului în care obiectele mutable și imutabile sunt manipulate în memorie.
- **Domeniul de valabilitate al variabilelor (local vs global)** - impactul asupra vizibilității și duratei de viață a valorilor stocate.
- **Erori de sintaxă și comportamentul codului la rulare** - elevii trebuie să înțeleagă că orice greșeală sintactică împiedică rularea programului și generează mesaje de eroare explicite, care indică tipul de problemă și locația acesteia. Aceasta oferă oportunitatea de a învăța debugging-ul de bază și importanța respectării regulilor limbajului.
- **Mecanismul de execuție în Python** - cum interpretorul citește și execută codul linie cu linie, și ce înseamnă ca programul să fie interpretat în timp real, evidențiind diferențele - avantaje și dezavantaje - față de limbajele compilate.

Această abordare le permite elevilor să înțeleagă nu doar **cum se scrie codul**, ci și ce **se întâmplă** când codul nu respectă regulile limbajului, oferind context pentru **erori** și **debugging** înainte de a trece la concepte mai abstracte, precum liste, clase și obiecte.

b) Instrucțiuni de control:

Structurile fundamentale de control (**while**, **for**, **if-else**) trebuie să fie **centrul dezvoltării gândirii algoritmice**, nu elemente secundare sau introduse derivativ. Acestea reprezintă baza gândirii algoritmice, deoarece permit elevilor să definească fluxul logic al programului. Fără o înțelegere solidă a acestor structuri, elevii riscă să scrie cod care „funcționează” doar în cazuri specifice, fără să înțeleagă principiul general.

În plus, elevii trebuie să fie familiarizați cu modul corect de utilizare a instrucțiunilor de control și cu subtilitățile acestora:

- **Bucle **for** și **while**:** cum se scrie sintactic corect o buclă, ce reprezintă expresia de condiție și cum se modifică variabila de control; de ce bucla poate include un **else** și ce înseamnă acesta în contextul Python (executat dacă bucla se termină normal, fără **break**).
- **Instrucțiunile **break** și **continue**:** cum **break** oprește imediat bucla curentă, iar **continue** sare peste iterația curentă și trece la următoarea, și cum utilizarea lor influențează fluxul de execuție.
- **Instrucțiunea **if-else**:** cum se definesc condițiile logice, cum se interpretează ramura **else** și cum poate fi utilizată pentru a acoperi toate cazurile posibile; diferența dintre **if-elif-else** și mai multe instrucțiuni **if** independente.
- **Structuri combinate:** cum se pot include bucle în interiorul condițiilor și condiții în interiorul buclilor, pentru a construi algoritmi mai complecși.

Astfel, elevii nu doar că învață sintaxa, dar și înțeleg logica fiecărei structuri de control și modul în care instrucțiunile speciale (**break** și **continue**) modifică fluxul algoritmului, dezvoltând gândirea algoritmică necesară pentru programe corecte și eficiente.

Astfel, învățarea acestor structuri trebuie să includă nu doar sintaxa, ci și **comportamentul programului la rulare**, precum și **modalitățile de diagnosticare a erorilor**.

c) Sintaxa vs semantica limbajului:

Pe lângă instrucțiunile de control, elevii trebuie să înțeleagă diferența dintre **sintaxa limbajului** și **semnificația conceptuală a codului**. Aceasta include:

- **Utilizarea și instalarea pachetelor externe**, pentru a vedea cum codul poate fi extins și modularizat;
- **Organizarea codului în mai multe fișiere**, pentru a înțelege principiile de modularizare și reutilizare;
- **Principiile de naming, indentare și stil (coding style)**, care, deși nu sunt tratate implicit în programă, sunt esențiale pentru dezvoltarea unui stil de programare corect și coerent care asigură lizibilitatea codului.

Fără o prezentare clară a acestor fundamente, elevii riscă să învețe doar **sintaxa izolată**, fără să înțeleagă **conceptele** care stau la baza funcționării limbajului și fără să poată aplica cunoștințele în contexte noi.

Lipsa unei introduceri metodice și progresive asupra instrucțiunilor de control și a semanticii limbajului are efecte directe asupra procesului de învățare:

- *Elevii memorează comenzi și sintaxă* fără a înțelege principiile sau logica de funcționare;
- *Capacitatea de a transfera cunoștințele către alte limbaje* este **redușă**, deoarece nu au dezvoltat schemă conceptuală și gândirea critică;

- Apar *dificultăți* în debugging și în rezolvarea problemelor, deoarece elevii nu au exemple și explicații clare pentru elementele fundamentale de sintaxă și structură.

Consolidarea acestor noțiuni fundamentale înainte de a introduce concepte mai abstracte, precum clasele și obiectele, este esențială pentru succesul pedagogic în predarea programării.

2. Citirea datelor în Python

Un aspect problematic în predarea Python este modul în care limbajul tratează tipurile de date. Toate datele citite de la tastatură prin funcția `input()` sunt inițial șiruri de caractere (*str* - strings), chiar dacă elevul introduce valori numerice. Aceasta generează o serie de dificultăți pedagogice concrete:

- `input()` returnează întotdeauna un string, nu un număr.
- Pentru a efectua operații numerice, este necesară conversia explicită: `int(input())` sau `float(input())`.
- Conceptul de conversie de tip nu este intuitiv și rămâne neclar dacă elevul nu a discutat anterior despre tipurile de date și despre reprezentarea internă a acestora.

Astfel, problema nu este doar de sintaxă, ci de semnificație fundamentală: elevul trebuie să înțeleagă ce reprezintă fiecare tip și de ce anumite operații nu sunt permise fără conversie, mai ales în contextul unui limbaj precum Python, care nu impune declararea explicită a tipului de date pentru variabile. Această flexibilitate sintactică poate masca diferențele reale dintre tipuri și poate genera confuzie.

Necesitatea unei fundamentări prealabile

Înainte de a introduce citirea și procesarea datelor în Python, este esențial ca elevii să aibă o înțelegere conceptuală clară:

- **Tipurile de date:** ce este un *șir de caractere*? ce este un *întreg*? Cum se diferențiază și care sunt comportamentele caracteristice ale fiecăruia?
- **Reprezentarea internă:** De ce sunt diferite aceste tipuri și ce înseamnă conversia de tip (*type casting*)?
- **Domeniul de aplicabilitate:** Când folosim *strings* și când folosim numere?
- **Operații specifice:** De ce putem concatena *strings*, dar nu putem adăuga un număr la un *string* fără conversie?

Compararea cu alte limbaje de programare

Dificultatea implicită a tipurilor în Python contrastează cu alte limbaje tradiționale:

- **C/C++:** Tipurile sunt explicite la declarare (`int x = 5; char s[] = "hello";`), ceea ce evidențiază imediat diferențele între date.
- **Pascal:** Tipurile sunt declarate vizibil (`var x: integer; s: string;`), facilitând înțelegerea conceptuală pentru programatorul începător.
- **Python:** `x = 5` și `s = "hello"` par identice din punct de vedere sintactic, mascând diferențele fundamentale dintre tipuri.
- În contextul prelucrării cifrelor unui număr, programa actuală poate crea confuzii semnificative. Citirea numerelor ca stringuri prin `input()` face ca orice număr introdus să fie inițial un șir de caractere. Aceasta poate duce la confuzie atunci când elevii încearcă să facă operații aritmetice sau să acceseze cifre individuale.

```
a = input("a = ")
b = input("b = ")
print(a + b)
a = 4
b = 5
45
```

Prin urmare, un **concept fundamental și simplu**, precum **citirea datelor** devine strâns interconectat de altele mai **complexe** și aduce un **efort cognitiv** nenecesar în primele programe pe care considerăm că trebuie să le facă un elev.

3. Introducerea Programării Orientate pe Obiecte în clasa a IX-a

Programarea Orientată pe Obiecte (POO) este prezentată în literatura de specialitate drept o paradigma de programare avansată care necesită baze solide de programare imperativă pentru a putea fi înțeleasă. Studiile lui Bennedsen și Caspersen [1, 2] arată că nivelul de abstractizare al studenților nu este un predictor clar al succesului în învățarea POO la nivel introductiv, ceea ce sugerează că inclusiv persoanele cu abilități cognitive dezvoltate pot întâmpina dificultăți semnificative în înțelegerea conceptelor cheie din POO, dacă nu stăpânesc conceptele de bază ale programării. Acest rezultat poate fi interpretat ca o dovadă că introducerea POO la clasa a IX-a, când elevii abia încep să învețe bazele programării și să-și dezvolte gândirea algoritmică, riscă să fie prematură. Capacitatea de abstractizare a elevilor nu garantează înțelegerea corectă a conceptelor, iar dificultatea de a aplica noțiuni precum clase, obiecte, metode o să apară în rândul unui număr foarte mare de elevi. Mai mult, studiile lui Bennedsen și Caspersen [1, 2] sugerează că succesul în programare depinde de alți factori, precum abilitatea practică de a scrie cod și înțelegerea logicii programării. Astfel, pentru elevii de liceu, o abordare progresivă, care să consolideze mai întâi conceptele programării imperative și gândirea algoritmică, urmată apoi de explicarea conceptelor de Programare Orientată pe obiecte abia în clasa a XI-a, ar putea fi mult mai eficientă și mai realistă decât introducerea directă a unor concepte POO pe care nu le vor înțelege.

Clasele și obiectele necesită o înțelegere prealabilă a tipurilor de date, a structurii memoriei și a modului de lucru al limbajului de programare. Elevii de 14-15 ani se află, conform lui Piaget, la stadiul operațional concret, iar gândirea abstractă nu este suficient dezvoltată pentru a asimila rapid conceptele de clasă, membri ai clasei (date și metode) sau obiecte. De asemenea, abia în clasa a IX-a sunt predate subprogramele (funcțiile), iar elevii nu vor avea timp să experimenteze suficient cu acestea, astfel că nu ne putem aștepta să facă distincția între o funcție și o metodă sau să înțeleagă pe deplin rolul încapsulării datelor și motivul pentru care este util să lucrăm cu clase. Fără această experiență practică anterioară, introducerea POO riscă să fie confuză și să nu atingă obiectivele de învățare.

Modelul conceptual liniar - listă

Programa solicită lucrul cu **liste** (secțiunile 1.1 și 3.5) și cu clasa **list** din Python (secțiunea 3.5), însă evită explicarea explicită a naturii orientate pe obiecte a acestora. O abordare pedagogică mai eficientă ar fi introducerea conceptului de tablou pe baza următoarelor variante:

- Utilizarea pseudocodului care permite lucrul cu tablouri abstracte de valori, fără detalii de implementare sau reprezentare
- Utilizarea unui limbaj procedural (C, C++) unde tablourile sunt structuri fundamentale, cu semantică bine definită.

Pe lângă liste, **tuplurile** ar putea fi prezentate în paralel, deoarece sunt utile în multe contexte practice în Python. De exemplu, o funcție poate returna mai multe valori simultan folosind un tuplu, cum ar fi:

```
def min_max(lista):  
    return min(lista), max(lista)
```

```
rezultat = min_max([4, 7, 1, 9]) # conține tuplul (1, 9)
```

Tuplurile sunt imutabile și permit elevilor să înțeleagă diferența dintre **date mutabile și imutabile**, **concept** important în proiectarea programelor și în înțelegerea comportamentului funcțiilor.

De asemenea, programa omite conceptul de **range**, **concept** care este foarte util pentru generarea de secvențe și pentru construirea de bucle **for** simple, clare și eficiente:

```
for i in range(1, 10, 2):  
    print(i)
```

Un alt element lipsă este **conceptul de excepție**, deși unele metode ale listelor predefinite pot genera erori (de exemplu, `list.index()` poate arunca excepția `ValueError` dacă elementul nu există). Explicarea modului de prindere și tratare a excepțiilor (**try - except**) ar fi utilă pentru:

- Înțelegerea erorilor și prevenirea opririi neașteptate a programului;
- Dezvoltarea gândirii defensive în programare;
- Introducerea treptată a unui concept fundamental al limbajelor moderne: manipularea controlată a situațiilor neprevăzute.

4. Lipsa conștientizării limitărilor tehnologice ale unui calculator

Un alt efect major al utilizării exclusive a limbajului Python în programă este pierderea unei componente esențiale a **alfabetizării informatice**: **conștientizarea limitărilor reale ale resurselor pe un calculator** și a modului în care acestea influențează logica și stilul programării.

Python ascunde aproape în totalitate detalii legate de: gestionarea memoriei, tipurile primitive, limitele numerice, structurile de date statice, reprezentarea internă a datelor, **costul real al operațiilor**.

În special, **modul în care Python gestionează numerele întregi creează impresia falsă că acestea sunt “infinite”**, întrucât:

- În locul unui tip `int` pe 32 sau 64 de biți, Python folosește un tip *de dimensiune arbitrară* (arbitrary-precision integer).
- Elevul **nu** se lovește niciodată de **overflow**, depășiri de interval sau reprezentări limitate,
- Elevul nu înțelege că în computerele reale orice număr are un **cost de stocare** și un **cost de procesare** dependent de mărimea lui.

Această proprietate, deși convenabilă în proiecte avansate, **distorsionează în clasa a IX-a înțelegerea naturală a limitelor tehnice ale unui calculator**, ducând la convingerea că:

- Orice număr, oricât de mare, poate fi procesat instantaneu.
- Operațiile matematice nu se degradează odată cu mărimea datelor.
- Resursele de memorie sunt nelimitate și gratuite.

Prin contrast, în limbaje precum C și C++, elevii descoperă natural și firesc:

- că un `int` are o limită;
- că depășirea limitei produce overflow;
- că alegerea tipului potrivit este o decizie esențială;
- că mulțimea de valori reprezintă o constrângere tehnică reală;
- că datele ocupă memorie fizică concretă.

Aceste situații formează **reflexe corecte** despre cum funcționează un calculator și reprezintă baza oricărei înțelegeri ulterioare despre performanță, optimizare, structuri de date și arhitectură.

Folosirea exclusivă a Python în clasa a IX-a elimină orice contact cu limitările numerice și de memorie ale unui calculator real, ceea ce poate crea un **model mental eronat și greu de corectat în anii următori**.

5. Complexitatea nealinată a programei propuse

Programa de informatică pentru clasa a IX-a introduce simultan un **număr foarte mare de concepte avansate**, într-un ritm accelerat și într-un limbaj care **ascunde** mecanisme fundamentale de programare. Această abordare generează o **complexitate nealinată** cu nivelul de dezvoltare cognitivă al elevilor de 14-15 ani și cu volumul limitat al orelor (2 ore/săptămână).

În mod concret, elevii sunt puși în situația de a gestiona în paralel:

- noțiuni fundamentale despre variabile, tipuri de date și memorie;
- manipularea obiectelor complexe precum listele Python;
- conversii explicite între tipuri pentru citirea datelor;
- structurarea unui program în funcții și fișiere;
- comportamente interpretate linie cu linie;
- noțiuni de programare orientată pe obiecte (clase, obiecte, metode);
- interfețe grafice și alte concepte care presupun un grad ridicat de abstractizare.

Această suprapunere de niveluri conceptuale profund diferite conduce la o **suprasarcină cognitivă** documentată în literatura de specialitate (“extraneous cognitive load” [3]), ceea ce afectează în mod direct capacitatea elevilor de a învăța în mod semnificativ. În loc să **evolueze progresiv** de la noțiuni simple la concepte mai avansate, elevii sunt expuși simultan la elemente care, în mod firesc, ar trebui distribuite pe parcursul a doi sau chiar trei ani de studiu

Menționăm că anumite lucruri nu pot evitate a fi explicate, pentru a putea înțelege sau folosi un concept sau o construcție în Python. Dăm următorul exemplu: un element suplimentar care complică procesul de învățare este faptul că **în Python nu există cu adevărat tipuri de bază simple**.

Chiar și operații aparent banale, precum:

```
x = 5
```

nu reprezintă stocarea unui simplu număr întreg, așa cum ar fi în limbaje precum C sau C++.

În realitate, această instrucțiune:

- Creează un **obiect** al clasei `int`.
- Instanțiază o **structură complexă** cu funcționalitate internă bogată.

- **Alocă dinamic memorie** pentru un obiect cu dimensiune arbitrară.
- Creează o **referință** către acest obiect în spațiul de nume curent.

Așadar, **cea mai simplă operație din programare** – definirea unei variabile întregi – implică în Python un mecanism orientat pe obiecte, pe care **elevul nu îl conștientizează și nu îl poate înțelege** în primele săptămâni de liceu. Menționăm că a evita explicația comportamentului din spate pentru aceste construcții fundamentale, reprezintă **o greșeală pedagogică fundamentală**.

Referințe bibliografice:

- [1] Bennedsen, J., & Caspersen, M. E. (2006). Abstraction ability as an indicator of success for learning object-oriented programming?. *ACM Sigcse Bulletin*, 38(2), 39-43.
- [2] BENNEDSEN, Jens; CASPERSEN, M. A Model-First Approach to Teaching Introductory Object-Orientation. In: *Workshop on Learning and Teaching Object-Orientation-Scandinavian Perspectives*. Oslo. 2003.
- [3] Studiu la nivelul UE: “Extraneous cognitive load”, <https://data.europa.eu/apps/data-visualisation-guide/extraneous-cognitive-load>
- [4] WATSON, Christopher; FREDERICK, W.B. Li Failure Rates in Introductory Programming Revisited,
https://drive.google.com/file/d/13HjokoQiFZozg9wucyT2sazSPtZDDWEE/view?usp=share_link

Anexa 2.

Propunere schematică de programă pentru disciplina informatică

Clasele IX-XII

Curriculum de specialitate (CS)

Filiera teoretică, profilul real, specializarea matematică-
informatică și matematică-informatică, intensiv informatică
construită pe paradigma

$C^{++} \rightarrow C^{++}$ și *Python* $\rightarrow$ *Python* și *SQL*

COMPETENȚE GENERALE (CG)

CG1	Descrierea caracteristicilor esențiale ale principalelor concepte, structuri de date, metode și strategii utilizate în informatică
CG2	Explicarea mecanismelor de funcționare și a modului de utilizare a principalelor concepte, structuri de date, metode și strategii informatice în contexte diverse
CG3	Aplicarea algoritmilor fundamentali, utilizând un limbaj de programare adecvat, în funcție de natura problemelor de rezolvat
CG4	Analizarea unor activități din lumea reală în vederea organizării optime a datelor relevante, a descompunerii problemei date în subprobleme, precum și distingerea pașilor necesari prelucrării datelor pentru rezolvarea acestora
CG5	Evaluarea corectitudinii și a eficienței unui program, în raport cu restricțiile specifice problemei de rezolvat
CG6	Dezvoltarea de soluții informatice originale pentru probleme din contexte științifice sau cotidiene, cu testarea acestora pe date concrete

CONȚINUTURI ALE ÎNVĂȚĂRII

Clasa a IX-a

Domenii de conținut	Conținuturi	Nr. ore (orientativ) MI/Intensiv
1. Fundamente ale programării	1.1 Elemente de bază ale dezvoltării programelor – Analiza unei probleme în scopul identificării datelor de intrare, datelor de ieșire și a modului de organizare eficientă a datelor – Principiile programării structurate și modalitatea de reprezentare a algoritmilor în pseudocod – Etapele dezvoltării aplicațiilor utilizând un mediu de programare (editarea programului, compilarea programului și corectarea erorilor de sintaxă, execuția programului și depanarea programului – <i>debug</i>) – Principii și metode în elaborarea testelor de evaluare pentru un program – Elemente de analiză a complexității algoritmilor	1/1 8/10 1/1 1/1
2. Structuri de date	2.1. Tablouri unidimensionale (vectori) – Caracteristici ale tablourilor unidimensionale – Modalități de declarare a tablourilor unidimensionale – Modalitatea de calcul a dimensiunii spațiului de memorie alocat unui tablou unidimensional – Modalitatea de acces la elementele unui tablou unidimensional – Repere pentru realizarea parcurgerii tablourilor unidimensionale	1/1

Domenii de conținut	Conținuturi	Nr. ore (orientativ) MI/Intensiv
	– Repere pentru modificarea unui tablou prin inserări/ștergeri de elemente – Repere pentru verificarea unei proprietăți valabile pentru toate elementele unui tablou unidimensional, respectiv a existenței unui element care are o anumită proprietate – Repere pentru utilizarea vectorilor caracteristici în rezolvarea problemelor – Repere pentru utilizarea vectorilor de frecvență în rezolvarea problemelor – Repere pentru prelucrarea secvențelor de valori citite succesiv, cu sau fără memorare în tablouri unidimensionale (minime/maxime, secvențe maxime cu o anumită proprietate)	1/1 1/1 2/2 2/2 2/4
	Suplimentar, pentru studiul intensiv al disciplinei informatică: 2.1. Tablouri unidimensionale (vectori) – completare – Repere pentru utilizarea sumelor parțiale în tablouri unidimensionale în rezolvarea problemelor – Repere pentru utilizarea tablourilor de diferențe (<i>difference-arrays</i>) în rezolvarea problemelor – Repere pentru prelucrarea secvențelor de lungime fixată – Repere pentru implementarea operațiilor pe numere mari (adunare, scădere, produsul dintre un număr mare și un număr mic, determinarea câtului și restului împărțirii unui număr mare la un număr mic, produsul a două numere mari) – Repere pentru aplicarea tehnicii <i>Two-pointers</i> în rezolvarea problemelor	/4 /4 /4 /8 /6
3. Algoritmi fundamentali	3.1. Algoritmi de prelucrare a datelor numerice – Algoritmi de prelucrare a cifrelor numerelor naturale scrise în baza 10 (extragerea tuturor cifrelor unui număr natural, accesarea, modificarea, inserarea, respectiv eliminarea unei cifre de pe o poziție specificată, construirea unui număr prin adăugarea unor cifre într-o ordine specificată) – Algoritmi de prelucrare a divizorilor numerelor naturale (algoritmul de identificare a divizorilor unui număr natural, algoritmul de descompunere a unui număr natural în factori primi, algoritmul de testare a primalității numerelor naturale, algoritmul de determinare a celui mai mare divizor comun, respectiv a celui mai mic multiplu comun) – Algoritmi de calcul pentru a unor expresii matematice (sume, factoriale, puteri) – Algoritmi de generare a șirurilor definite prin relație de recurență 3.2. Algoritmi de prelucrare a tablourilor unidimensionale – Algoritmul de căutare secvențială – Algoritmul de căutare binară – Algoritmul de sortare prin selecția minimului/maximului (<i>Selection sort</i>) – Algoritmul de interclasare a două tablouri unidimensionale – Algoritmul de determinare a numerelor prime mai mici decât o valoare dată utilizând <i>Ciurul lui Eratostene</i> – Algoritmul de sortare utilizând vectori de frecvență	5/5 6/6 1/1 2/4 1/1 1/1 1/1 2/4 1/2
	Suplimentar, pentru studiul intensiv al disciplinei informatică 3.1. Algoritmi de prelucrare a datelor numerice – completare – Sisteme de numerație poziționale. Reguli de conversie între baza 10 și altă bază de numerație	/2

Domenii de conținut	Conținuturi	Nr. ore (orientativ) MI/Intensiv
	– Aritmetică modulară – Algoritmul de exponențiere rapidă	/2 /2
	3.2. Algoritmi de prelucrare a tablourilor unidimensionale – completare – Algoritmul de sortare prin metoda bulelor (<i>Bubble sort</i>) – Algoritmul de sortare prin inserție (<i>Insertion sort</i>) – Algoritmul liniar de determinare a unei secvențe de sumă maximă (<i>Kadane</i>)	/1 /1 /2
4. Elemente de limbaj de programare – C++	4.1. Fișiere text – caracteristici, operații de bază 4.2. Funcții – Rolul și avantajele utilizării funcțiilor – Sintaxa declarației, definiției și apelului unei funcții – Modalități de transfer al parametrilor la apel (transfer prin valoare și transfer prin referință) – Modalitatea de returnare a unei valori – Analiză comparativă a variabilelor locale și variabilelor globale din punctul de vedere al modului de declarare, clasei de memorare, duratei de viață, accesibilității, posibilității de inițializare automată	1/1 10/16

Clasa a X-a

Domenii de conținut	Conținuturi	Nr. ore (orientativ) MI/Intensiv
1. Structuri de date	1.1. Structuri de date neomogene (tipul struct) – Caracteristici ale structurilor de date neomogene – Modalități de declarare a unui tip de date pentru reprezentarea unei structuri de date neomogene și a variabilelor de acest tip – Modalitatea de acces la un câmp al unei variabile de tip structură 1.2. Structura de date abstractă Stiva – Caracteristici ale structurii de date abstracte Stiva – Implementarea statică a unei stive utilizând tablouri unidimensionale – Analiza problemelor în scopul identificării necesității utilizării unei stive 1.3. Structura de date abstractă Coda – Caracteristici ale structurii de date abstracte Coda – Implementarea statică a unei cozi utilizând tablouri unidimensionale – Analiza problemelor în scopul identificării necesității utilizării unei cozi 1.4. Tablouri bidimensionale – Caracteristici ale tablourilor bidimensionale – Modalități de declarare a tablourilor bidimensionale – Modalitatea de acces la elementele unui tablou bidimensional – Analiza problemelor în scopul identificării necesității utilizării tablourilor bidimensionale – Repere pentru realizarea unor operații specifice tablourilor bidimensionale (parcure pe linii, parcure pe coloane, bordare) – Modalități de parcure a tuturor vecinilor unui element al unui tablou bidimensional utilizând vectori de direcție – Caracteristici ale tablourilor bidimensionale pătratice – Repere pentru realizarea unor operații specifice tablourilor bidimensionale pătratice (parcurea diagonalelor, parcurea zonelor delimitate de diagonale, parcure pe chenare concentrice) 1.5. Șiruri de caractere – Caracteristici ale șirurilor de caractere	3/4 3/5 3/5 8/10 6/6 8/12

Domenii de conținut	Conținuturi	Nr. ore (orientativ) MI/Intensiv
	– Repere pentru prelucrarea șirurilor de caractere la nivel de caracter – Repere pentru prelucrarea șirurilor de caractere utilizând funcțiile din <i>cstring</i>	
	Suplimentar, pentru studiul intensiv al disciplinei informatică 1.4. Tablouri bidimensionale – completare – Repere pentru utilizarea sumelor parțiale în tablouri bidimensionale 1.6. Tablouri multidimensionale – Caracteristici ale tablourilor multidimensionale – Modalități de declarare a tablourilor multidimensionale – Modalitatea de acces la elementele unui tablou multidimensional – Repere pentru realizarea parcurgerii tablourilor multidimensionale	/4 /4
2. Algoritmi fundamentali	2.1. Algoritmi de umplere (<i>fill</i>) – Caracteristici ale problemelor pentru care se poate aplica un algoritm de umplere – Forma generică a unui algoritm de umplere	4/4
	Suplimentar, pentru studiul intensiv al disciplinei informatică 2.2. Algoritmul lui Lee – Caracteristici ale problemelor pentru care se poate aplica algoritmul lui Lee – Forma generică a algoritmului lui Lee 2.3. Algoritmi simpli de criptare/decriptare a șirurilor de caractere – caracteristici ale criptării prin metoda cifrului Caesar și repere pentru aplicarea metodei; – caracteristici ale criptării prin metoda substituției simple și repere pentru aplicarea metodei – caracteristici ale criptării prin metoda transpoziției și repere pentru aplicarea metodei	/6 /4
3. Strategii și tehnici de programare	3.1. Metoda Greedy – Caracteristici ale problemelor pentru care se poate aplica metoda Greedy – Forma generică a unui algoritm de tip Greedy – Repere pentru aplicarea metodei Greedy în rezolvarea problemelor	7/8
4. Elemente de limbaj de programare – C++	4.1. Pointeri – Caracteristici ale pointerilor – Modalitatea de declarare a unei variabile de tip pointer – Operatori care se pot utiliza asupra variabilelor de tip pointer – Legătura dintre pointeri și tablouri 4.2. Funcții recursive – Caracteristici ale subprogramelelor recursive – Mecanismul de realizare a recursivității – Modalități recursive de implementare a unor algoritmi elementari de prelucrare a cifrelor numerelor, de prelucrare a divizorilor numerelor, de exponențiere rapidă, de parcurgere a tablourilor unidimensionale	2/4 7/8
	Suplimentar, pentru studiul intensiv al disciplinei informatică 4.2. Funcții recursive – completare – Modalități recursive de generare a unor elemente combinatoriale (secvențe binare, submulțimi, produs cartezian, permutări, aranjamente, combinații, partiții număr, partiții mulțimi)	/8 /10

Domenii de conținut	Conținuturi	Nr. ore (orientativ) MI/Intensiv
	○ aplicarea metodelor specifice dicționarelor (<i>keys, values, items, get, update</i>) în prelucrări de date – Structura de date mulțime ○ definirea mulțimilor ca structuri de date neordonate ce conțin elemente unice ○ aplicarea operațiilor fundamentale asupra mulțimilor (reuniune, intersecție, diferență, apartenență) ○ utilizarea metodelor specifice mulțimilor (<i>add, remove, union, intersection, difference</i>) în prelucrări de date – Tipuri de memorie în Python (mutabilă/imutabilă). <i>Shallow copy</i> și <i>deep copy</i> – Programarea orientată pe obiecte ○ Clase și obiecte, câmpuri și metode ○ Constructori și destructor ○ Încapsulare ○ Moștenire simplă – Interfețe grafice în limbajul Python	6/12 14/22
	Suplimentar, pentru studiul intensiv al disciplinei informatică 4.1. Limbajul Python. Programarea orientată pe obiecte – completare ○ Accesul la membrii unei clase (public, private, protected) ○ Moștenire multiplă ○ Supradefinirea metodelor și a operatorilor. Polimorfism 4.2. Limbajul C++. Biblioteca STL – Clasele <i>set</i> – Clasele <i>map</i> – Clasa <i>priority queue</i>	/6 /6

Clasa a XII-a

Domenii de conținut	Conținuturi	Nr. ore (orientativ) MI/Intensiv
1. Elemente de bază în Inteligența Artificială	1.1. Concepte de bază ale Inteligenței Artificiale – Domeniile Inteligenței Artificiale – Învățare Automată – concepte de bază. Tipuri de învățare și tipuri de probleme asociate ○ Învățare nesupervizată. Probleme de tip clusterizare. Algoritmul <i>K-means</i> ○ Învățare supervizată. Probleme de tip regresie și clasificare. Algoritmul de regresie liniară. Arborele de decizie. Algoritmul <i>KNN (K-Nearest Neighbors)</i> ○ Introducere în rețele neurale – concepte de bază. Modelul neuronului artificial. Rețele perceptron – caracteristicile modelului și algoritmul de învățare – Implicații etice și sociale ale Inteligenței Artificiale ○ Pericole și riscuri asociate IA (bias, confidențialitate, securitate, autonomie) ○ Avantaje și riscuri ale IA generative ○ Impactul asupra pieței muncii și carierei ○ Aspecte etice și responsabilitatea utilizării IA ○ Reglementări și bune practici	22/28
2. Structuri de date	2.1. Modelul conceptual al unei activități din lumea reală – Analiza unei activități din lumea reală în scopul identificării datelor relevante și a relațiilor dintre acestea	18/24

Domenii de conținut	Conținuturi	Nr. ore (orientativ) MI/Intensiv
	– Entități și instanțe – Atribute. Opționalitate – Identificatori unici – Relații între entități. Cardinalitate și opționalitate ○ Clasificarea relațiilor în funcție de cardinalitate (1:1, 1:M, M:M) ○ Transferabilitate ○ Relații recursive – Convenții de reprezentare în diagrama ERD a entităților, atributelor, identificatorilor unici, relațiilor – Rezolvarea relațiilor M:M – Normalizarea datelor: 1NF, 2NF, 3NF – Transformarea modelului conceptual în model fizic ○ Tabele, coloane, cheie primară, chei străine (externe), chei unice, constrângeri ○ Convenții de descriere a modelului fizic – Modele de baze de date	
	Suplimentar, pentru studiul intensiv al disciplinei informatică 2.1. Modelul conceptual al unei activități din lumea reală – completare – Arce – Subtipuri – Modelarea datelor din punct de vedere istoric	/8
3. Dezvoltarea aplicațiilor web	Suplimentar, pentru studiul intensiv al disciplinei informatică 3.1 Elemente de dezvoltare a aplicațiilor web care utilizează baze de date – Concepte de bază ale dezvoltării aplicațiilor web ○ Analiza unei activități din lumea reală pentru identificare nevoilor de gestionare ○ Diferența între <i>backend</i> și <i>frontend</i> ○ Formatul de text JSON – Elemente de <i>Frontend</i> ○ Noțiunea de interfață grafică ○ Alcătuirea schemelor de experiență utilizatorului (<i>user experience</i> și <i>user stories</i>) ○ Proiectarea elementelor vizuale în concordanță cu schemele alcătuite ○ Implementarea unei interfețe grafice adecvate în limbajul Python – Elemente de <i>Backend</i> ○ Identificarea funcționalităților necesare aplicației ○ Implementarea eficientă a acestor funcționalități în limbajul Python ○ Gestionarea datelor aplicației cu ajutorul unei baze de date ○ Integrarea inteligenței artificiale într-o aplicație	/30
4. Elemente de limbaj de programare	4.1. Biblioteci Python pentru Inteligența artificială – Analizarea datelor în Python. Biblioteca <i>pandas</i> – citirea și scrierea seturilor de date în formate uzuale (CSV, JSON) – accesarea și modificarea elementelor, coloanelor și rândurilor (iloc, loc, at, iat) – operații de bază cu <i>DataFrame</i>-uri: selectare, filtrare, sortare, adăugare și ștergere de coloane sau rânduri – calculul indicatorilor statistici pentru variabilele numerice (minimul, maximul, media, mediana, abaterea standard, varianța) – calculul corelațiilor dintre variabile – detectarea valorilor extreme – normalizarea și standardizarea datelor	12/28

Domenii de conținut	Conținuturi	Nr. ore (orientativ) MI/Intensiv
	– Vizualizarea datelor în Python. Bibliotecile <i>Matplotlib</i> și <i>Seaborn</i> – trasarea graficului unei funcții matematice – realizarea diagramelor simple (liniare, cu bare, circulare) – reprezentarea vizuală a distribuțiilor prin histograme – adăugarea de etichete, legendă și titlu pentru claritatea graficelor – afișarea și exportarea graficelor – Algoritmi pentru învățare automată. Biblioteca <i>scikit-learn</i> – împărțirea datelor în seturi de antrenare și testare (<i>train_test_split</i>) – scalarea și transformarea datelor cu <i>StandardScaler</i>, <i>MinMaxScaler</i>, <i>LabelEncoder</i>, <i>OneHotEncoder</i> – crearea modelelor de învățare supervizată și nesupervizată din biblioteca <i>scikit-learn</i> – antrenarea și testarea modelelor (<i>fit</i>, <i>predict</i>, <i>score</i>) – evaluarea performanței modelelor	
	4.2. Limbajul SQL – Transformarea modelului fizic într-o bază de date relațională – Crearea tabelor (CREATE TABLE) – Modificarea structurii unei tabele (ALTER TABLE) – Ștergerea tabelor (DROP TABLE) – Definirea cheilor primare și externe, constrângeri (PRIMARY KEY, FOREIGN KEY, UNIQUE, CHECK) – Manipularea datelor (DML) – Inserarea de date (INSERT) – Modificarea datelor existente (UPDATE) – Ștergerea datelor (DELETE) – Interogări simple. Selecție, proiecție, ordonare – Expresii și funcții – Expresii aritmetice și logice – Funcții pe rânduri individuale (<i>single-row</i>): text, numerice, date – Subinterogări în clauza WHERE – Interogări avansate (JOIN)	20/28
	Suplimentar, pentru studiul intensiv al disciplinei informatică 4.1. Biblioteci Python pentru Inteligența artificială – completare – Elemente de algebră liniară. Biblioteca <i>NumPy</i>. Operații cu tablouri în <i>NumPy</i> 4.2. Limbajul SQL – completare – Agregare și grupare – Funcții de agregare (<i>multiple row</i>): COUNT, SUM, AVG, MIN, MAX – Gruparea datelor (GROUP BY) – Filtrarea grupurilor (HAVING) – Subinterogări corelate – Crearea și utilizarea secvențelor – Indecși și sinonime – Gestionarea drepturilor de acces la baza de date – Gestionarea tranzacțiilor	/8
	4.3. Biblioteci Python pentru Dezvoltarea Aplicațiilor Web – Interacțiunea cu o bază de date. Biblioteca <i>sqlite</i> – crearea și conectarea la o bază de date SQLite – executarea interogărilor SQL (SELECT, INSERT, UPDATE, DELETE) din Python – manipularea rezultatelor interogărilor și gestionarea erorilor – utilizarea operațiilor CRUD pentru persistarea datelor – integrarea bazei de date într-o aplicație web	/16

Domenii de conținut	Conținuturi	Nr. ore (orientativ) MI/Intensiv
	– Adăugarea unei interfețe grafice. Biblioteca <i>gradio</i> – crearea interfețelor web interactive pentru aplicații Python – conectarea funcțiilor Python care interacționează cu baza de date la interfața Gradio – operații tipice: adăugarea de date noi, afișarea datelor existente, actualizarea sau ștergerea unor înregistrări, folosind interfața web	
5. Dezvoltarea profesională în domeniul IT	5.1. Repere pentru dezvoltarea profesională în domeniul IT și managementul proiectelor – Cariere în IT. Conexiunea dintre industrie și informatica studiată în liceu – Principii ale lucrului în echipă – Managementul unui proiect în domeniul IT ○ Rolul managementului de proiect ○ Etapele unui proiect (ciclul de viață) ○ Roluri în managementul de proiect ○ Instrumente și metode – Reguli pentru susținerea unei prezentări de proiect	4/8