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

Context

România deţine un avantaj competitiv solid, construit istoric şi confirmat internaţional, fiind recunoscută ca un important furnizor de ingineri software cu pregătire riguroasă în algoritmică, sisteme şi optimizare. Menţinerea şi consolidarea acestui avantaj presupun dezvoltarea continuă a unei mase critice de specialişti cu profil autentic de *engineer*. Învăţământul preuniversitar de informatică a avut un rol determinant în această direcţie, programele actuale contribuind semnificativ la formarea gândirii computaţionale a absolvenţilor de la specializarea matematică-informatică.

În contextul în care România dezvoltă tehnologie în multiple sectoare, iar industria IT&C generează peste 8% din PIB şi prezintă perspective clare de creştere, investiţia în formarea specialiştilor din domeniu reprezintă o opţiune strategică. Aceasta este esenţială pentru asigurarea securităţii economice a ţării pe termen lung.

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

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

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

În **Anexa 1** prezentăm o analiză detaliată a programei puse în dezbatere, care să demonstreze faptul că alegerea acestui limbaj pentru a preda conceptele în forma şi ordinea din această programă ridică probleme didactice semnificative, împiedicând înţelegerea naturală a conceptelor de bază ale programării, dar şi 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 abstracţie (tipuri dinamice, obiecte, instanţe, referinţe, structuri dinamice);

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

Aceste cerinţe depăşesc capacitatea cognitivă tipică pentru acest nivel de vârstă, generând suprasolicitare (*cognitive overload*), învăţare superficială şi confuzie conceptuală.

Didactica programării, fundamentată în ultimele două decenii (Robins, Rountree & Rountree, 2003; Lister, 2011; Luxton-Reilly, 2016), recomandă introducerea programării prin: construcţie graduală a conceptelor, transparenţă a mecanismelor, reducerea complexităţii iniţiale, înţelegerea execuţiei programului înainte de abstractizare.

2. Caracteristicile limbajului Python maschează concepte fundamentale

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

- citirea datelor produce întotdeauna text, impunând înţelegerea conversiilor de tip înainte ca elevul să înţeleagă noţiunea de tip de date;
- gestionarea automată a numerelor mari elimină percepţia asupra limitelor calculatoarelor;
- orice variabilă este un obiect, chiar dacă elevul încă nu cunoaşte noţiunile de referinţă, instanţiere sau spaţiu de memorie.

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

3. Introducerea prematură a Programării Orientate pe Obiecte

POO este un cadru conceptual care presupune abstractizare, clasificare, modelare şi relaţionare între entităţi – procese cognitive de nivel înalt.

Literatura de specialitate subliniază că POO este dificilă chiar şi la nivel universitar pentru studenţii începători. Introducerea sa la clasa a IX-a, fără un fundament solid al programării structurate, contravine recomandărilor din literatura de specialitate şi generează suprasolicitare cognitivă. Elevii aflaţi încă în procesul de învăţare a funcţiilor nu au resursele necesare pentru a integra noţiuni precum clase, metode şi instanţe.

4. Suprasarcina curriculară în raport cu timpul disponibil

Două ore pe săptămână sunt insuficiente pentru parcurgerea unui volum atât de complex, respectiv consolidarea cunoştinţelor şi exersarea prin activităţi practice.

Prea multe concepte introduse prea rapid duc la fragmentare, pierderea coerenţei şi imposibilitatea asigurării unei învăţări semnificative.

La nivel introductiv, o înţelegere solidă a conceptelor fundamentale de programare presupune expunerea elevilor la numeroase activităţi aplicative şi 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 puţini algoritmi şi exemple practice care să faciliteze transferul acestor concepte în activităţi concrete de învăţare.

Trebuie să menţionăm că **suprasarcina curriculară** este semnificativ mai mare pentru specializarea **Ştiinţele naturii**. Deşi 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, diferenţele dintre programele celor două specializări sunt minore (necesitând cel mult 5 ore), singurele elemente de conţinut suplimentare la matematică-informatică fiind:

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

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

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

Demersul didactic trebuie să fie centrat pe elev, vizând formarea capacităţii acestuia de a transfera şi aplica conceptele teoretice în situaţii practice. Dacă accentul lecţiei rămâne predominant pe demonstraţiile profesorului, rezultatul va fi, în cel mai bun caz, dezvoltarea competenţei elevilor de a înţelege soluţiile prezentate, nu şi de a le reproduce sau adapta autonom în contexte noi. Din acest motiv, **desfăşurarea orelor de informatică în laborator este esenţială** pentru formarea competenţelor specifice domeniului.

Concluzie

Din perspectivă pedagogică şi cognitivă, ordinea şi 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 funcţionarea programelor**.

Construcţia competenţelor de programare are nevoie de o abordare secvenţială, transparentă şi adecvată vârstei, în care conceptele fundamentale sunt consolidate înaintea celor avansate. Reechilibrarea programei în această direcţie ar creşte semnificativ eficienţa procesului de învăţare şi ar asigura premisele pentru o formare autentică în domeniul informaticii.

Soluţia propusă – *Progresia C++ → C++ şi Python → Python şi SQL*

Anexa 2 conţine o propunere schematică de programă pentru disciplina informatică pentru specializarea matematică-informatică, cu diferenţiere pentru matematică informatică intensiv pentru clasele IX-XII. Documentul include competenţele generale vizate şi conţinuturile propuse, cu o detaliere orientativă a alocării orare, pentru a valida fezabilitatea atingerii competenţelor specifice în bugetul de timp alocat.

Principiile programei propuse

- Propunerea de programă pe care o prezentăm este structurată pe principiul progresiei graduale şi al relevanţei în raport cu finalităţile educaţionale şi cu cerinţele pieţei muncii.
- Programa urmăreşte formarea unei gândiri computaţionale şi o înţelegere profundă a conceptelor fundamentale ale programării pentru toţi elevii.
- Programa are o construcţie organică, pornind de la fundamente şi evoluând natural spre aplicabilitate practică.

Structură generală a programei propuse, din perspectiva limbajelor de programare

Clasele a IX-a şi a X-a: limbajul C++

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

Clasa a XI-a: C++ şi Python

În această etapă, se continuă aprofundarea algoritmicii şi 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: Inteligenţă Artificială, Data Science, Cloud, automatizare. Abordarea limbajului Python va fi accesibilă pentru elevii de clasa a XI-a, care au deja formate competenţe în domeniul algoritmicii şi programării prin intermediul limbajului C++.

Clasa a XII-a: Python şi SQL

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

Progresia C++ → C++ şi Python → Python şi SQL asigură echilibrul dintre rigoarea algoritmică şi aplicabilitatea practică, oferindu-le elevilor atât competenţe fundamentale (structuri de date, optimizare, gândire algoritmică), cât şi abilităţi moderne, aliniate cu cerinţele actuale ale pieţei muncii şi ale învăţământului superior de profil.

C++ versus Python – avantaje versus riscuri

Concept	C++	Python	Impact educaţional
Citirea datelor	Explicită, tipuri fixe	Totul ca şir de caractere	În C++, elevul învaţă structuri şi tipuri înainte de concepte avansate.
Rigoare	Strict, erori obligatoriu rezolvate	Flexibil, codul poate funcţiona chiar cu erori	În C++, atenţie la detalii şi disciplină algoritmică. În Python, elevii dezvoltă mentalitatea „merge şi aşa”, ceea ce slăbeşte atenţia şi ordonarea gândirii.
Structuri de date	Tablouri fixe, clare	Liste de liste, dicţionare, clase	C++ dezvoltă înţelegerea 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 înţelegerea solidă a fundamentelor disciplinei.
Mutabilitate	Clară, explicită	Amestec de mutabile şi imutabile	C++ învaţă gestionarea memoriei şi referinţelor. Python poate crea confuzie în manipularea datelor şi referinţelor.

Perspective pe termen lung

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

Un alt avantaj al C şi C++ este că formează rapid capacitatea de adaptare. Majoritatea limbajelor moderne – Java, C#, Go sau Rust – sunt inspirate direct din C/C++. Elevul care învaţă C++ înţelege uşor aceste limbaje şi poate să se adapteze rapid la cerinţele 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 înţelege, construi şi inova, indiferent de ce limbaj va fi la modă mâine.

Într-un timp relativ scurt limbajele vor fi înlocuite de interfeţele conversaţionale. Aplicaţiile vor fi realizate fără a scrie cod într-un limbaj, ci adresându-ne AI-ului în limbaj natural. Întrebarea ce urmează firesc este "*Cum sa conversezi cu AI pentru crearea unor aplicaţii dacă nu ai o înţelegere profundă a mecanismelor fundamentale ale programării*?"

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

Matematică-informatică intensiv vs neintensiv

Programele şcolare 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 aceştia beneficiază de un număr dublu de ore faţă de colegii lor de la neintensiv. Este important de subliniat că ponderea elevilor înscrişi în clase cu regim intensiv este relativ redusă la nivel naţional.

În prezent, facultăţile de profil din România (informatică, automatică şi calculatoare, cibernetică) formează aproximativ 10.000 de studenţi anual. Totuşi, pentru perioada 2026–2035, proiecţiile arată că economia digitală va necesita un număr semnificativ mai mare de specialişti IT cu formare de nivel *engineer*. În acest context, devine esenţial ca învăţământul preuniversitar să consolideze dezvoltarea gândirii computaţionale în rândul elevilor de la specializarea matematică-informatică neintensiv. Formarea competenţelor specifice disciplinei Informatică este nu doar o premisă pentru accesul elevilor la cariere în domeniul IT, ci şi o resursă intelectuală valoroasă pentru orice domeniu tehnic, precum şi pentru alte domenii, de exemplu ştiinţele medicale.

Concluzie – misiunea educaţională

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 şcoala să construiască pentru România – simpli utilizatori de software sau tineri capabili să definească viitorul acestei ţări prin inovaţie sau, şi mai mult, creatori de tehnologie?

Anexa 1.

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

Programa de informatică pentru clasa a IX-a (specializarea matematică-informatică - neintensiv şi intensiv) propune utilizarea limbajului Python pentru introducerea conceptelor de programare, inclusiv liste, subprograme, fişiere, interfeţe grafice şi programare orientată pe obiecte (POO). Deşi Python este un limbaj popular şi versatil, pe care îl considerăm extrem de util şi potrivit pentru a construi exemple complexe de programe cu scop specific (de exemplu jocuri cu grafică sau inteligenţă artificială), considerăm că acesta trebuie să apară într-o etapă ulterioară a procesului de învăţare, în care ***conceptele de bază de programare*** (de exemplu, variabile, tipuri de date, tablouri - vectori şi matrici, fişiere, 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 şi ordinea din această programă ridică probleme didactice semnificative, împiedicând înţelegerea naturală a conceptelor de bază ale programării, dar şi limitarea sau lipsa abordării unora dintre concepte, ceea ce duce la superficialitate.

1. Absenţa unei introduceri solide în limbajul de programare suport

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

a) Concepte / noţiuni fundamentale:

- **Conceptul de variabilă şi modul de lucru al memoriei** - elevii trebuie să înţeleagă că variabilele nu sunt simple etichete, ci spaţii în memorie care stochează valori.
- **Conceptele despre tipurile de bază şi caracteristicile lor** - diferenţierea între int, float, string etc., precum şi comportamentul specific al fiecărui tip în operaţii.
- **Diferenţa dintre valoare şi referinţă în Python** - înţelegerea modului în care obiectele mutable şi imutabile sunt manipulate în memorie.
- **Domeniul de valabilitate al variabilelor (local vs global)** - impactul asupra vizibilităţii şi duratei de viaţă a valorilor stocate.
- **Erori de sintaxă şi comportamentul codului la rulare** - elevii trebuie să înţeleagă că orice greşeală sintactică împiedică rularea programului şi generează mesaje de eroare explicite, care indică tipul de problemă şi locaţia acesteia. Aceasta oferă oportunitatea de a învăţa debugging-ul de bază şi importanţa respectării regulilor limbajului.
- **Mecanismul de execuţie în Python** - cum interpretorul citeşte şi execută codul linie cu linie, şi ce înseamnă ca programul să fie interpretat în timp real, evidenţiind diferenţele - avantaje şi dezavantaje - faţă de limbajele compilate.

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

b) Instrucţiuni 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 înţelegere solidă a acestor structuri, elevii riscă să scrie cod care „funcţionează" doar în cazuri specifice, fără să înţeleagă principiul general.

În plus, elevii trebuie să fie familiarizaţi cu modul corect de utilizare a instrucţiunilor de control şi cu subtilităţile acestora:

- **Bucle for şi while**: cum se scrie sintactic corect o buclă, ce reprezintă expresia de condiţie şi cum se modifică variabila de control; de ce bucla poate include un else şi ce înseamnă acesta în contextul Python (executat dacă bucla se termină normal, fără break).
- **Instrucţiunile break şi continue**: cum break opreşte imediat bucla curentă, iar continue sare peste iteraţia curentă şi trece la următoarea, şi cum utilizarea lor influenţează fluxul de execuţie.
- **Instrucţiunea if-else**: cum se definesc condiţiile logice, cum se interpretează ramura else şi cum poate fi utilizată pentru a acoperi toate cazurile posibile; diferenţa dintre if-elif-else şi mai multe instrucţiuni if independente.
- **Structuri combinate**: cum se pot include bucle în interiorul condiţiilor şi condiţii în interiorul buclelor, pentru a construi algoritmi mai complecşi.

Astfel, elevii nu doar că învaţă sintaxa, dar şi înţeleg logica fiecărei structuri de control şi modul în care instrucţiunile speciale (break şi continue) modifică fluxul algoritmului, dezvoltând gândirea algoritmică necesară pentru programe corecte şi eficiente.

Astfel, învăţarea acestor structuri trebuie să includă nu doar sintaxa, ci şi ***comportamentul programului la rulare***, precum şi ***modalităţile de diagnosticare a erorilor***.

c) Sintaxa vs semantica limbajului:

Pe lângă instrucţiunile de control, elevii trebuie să înţeleagă diferenţa dintre **sintaxa limbajului** şi **semnificaţia conceptuală a codului**. Aceasta include:

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

Fără o prezentare clară a acestor fundamente, elevii riscă să înveţe doar **sintaxa izolată**, fără să înţeleagă ***conceptele*** care stau la baza funcţionării limbajului şi fără să poată aplica cunoştinţele în contexte noi.

Lipsa unei introduceri metodice şi progresive asupra instrucţiunilor de control şi a semanticii limbajului are efecte directe asupra procesului de învăţare:

- *Elevii memorează comenzi şi sintaxă* fără a înţelege principiile sau logica de funcţionare;
- *Capacitatea* de a transfera cunoştinţele către alte limbaje este **redusă**, deoarece nu au dezvoltat schemă conceptuală şi gândirea critică;

- Apar *dificultăţi* în debugging şi în rezolvarea problemelor, deoarece elevii nu au exemple şi explicaţii clare pentru elementele fundamentale de sintaxă şi structură.

Consolidarea acestor noţiuni fundamentale înainte de a introduce concepte mai abstracte, precum clasele şi obiectele, este esenţială 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 funcţia input() sunt iniţial şiruri de caractere (*str* - strings), chiar dacă elevul introduce valori numerice. Aceasta generează o serie de dificultăţi pedagogice concrete:

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

Astfel, problema nu este doar de sintaxă, ci de semnificaţie fundamentală: elevul trebuie să înţeleagă ce reprezintă fiecare tip şi de ce anumite operaţii 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 diferenţele reale dintre tipuri şi poate genera confuzie.

Necesitatea unei fundamentări prealabile

Înainte de a introduce citirea şi procesarea datelor în Python, este esenţial ca elevii să aibă o înţelegere conceptuală clară:

- **Tipurile de date**: ce este un şir de caractere? ce este un întreg? Cum se diferenţiază şi care sunt comportamentele caracteristice ale fiecăruia?
- **Reprezentarea internă**: De ce sunt diferite aceste tipuri şi ce înseamnă conversia de tip (type casting)?
- **Domeniul de aplicabilitate**: Când folosim strings şi când folosim numere?
- **Operaţii 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 tradiţionale:

- **C/C++**: Tipurile sunt explicite la declarare (int x = 5; char s[] = "hello";), ceea ce evidenţiază imediat diferenţele între date.
- **Pascal**: Tipurile sunt declarate vizibil (var x: integer; s: string;), facilitând înţelegerea conceptuală pentru programatorul începător.
- **Python**: x = 5 şi s = "hello" par identice din punct de vedere sintactic, mascând diferenţele 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 iniţial un şir de caractere. Aceasta poate duce la confuzie atunci când elevii încearcă să facă operaţii 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 şi simplu***, precum ***citirea datelor*** devine strâns interconectat de altele mai ***complexe*** şi 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 înţeleasă. Studiile lui Bennedsen şi Caspersen [1, 2] arată că nivelul de abstractizare al studenţilor nu este un predictor clar al succesului în învăţarea POO la nivel introductiv, ceea ce sugerează că inclusiv persoanele cu abilităţi cognitive dezvoltate pot întâmpina dificultăţi semnificative în înţelegerea 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 înveţe bazele programării şi să-şi dezvolte gândirea algoritmică, riscă să fie prematură. Capacitatea de abstractizare a elevilor nu garantează înţelegerea corectă a conceptelor, iar dificultatea de a aplica noţiuni precum clase, obiecte, metode o să apară în rândul unui număr foarte mare de elevi. Mai mult, studiile lui Bennedsen şi Caspersen [1, 2] sugerează că succesul în programare depinde de alţi factori, precum abilitatea practică de a scrie cod şi înţelegerea logicii programării. Astfel, pentru elevii de liceu, o abordare progresivă, care să consolideze mai întâi conceptele programării imperative şi gândirea algoritmică, urmată apoi de explicarea conceptelor de Programare Orientată pe obiecte abia în clasa a XI-a, ar putea fi mult mai eficientă şi mai realistă decât introducerea directă a unor concepte POO pe care nu le vor înţelege.

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

Modelul conceptual liniar - listă

Programa solicită lucrul cu **liste** (secţiunile 1.1 şi 3.5) şi cu clasa list din Python (secţiunea 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 funcţie 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 şi permit elevilor să înţeleagă diferenţa dintre **date mutabile şi imutabile**, ***concept*** important în proiectarea programelor şi în înţelegerea comportamentului funcţiilor.

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

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

Un alt element lipsă este **conceptul de excepţie**, deşi unele metode ale listelor predefinite pot genera erori (de exemplu, list.index() poate arunca excepţia ValueError dacă elementul nu există). Explicarea modului de prindere şi tratare a excepţiilor (try - except) ar fi utilă pentru:

- Înţelegerea erorilor şi prevenirea opririi neaşteptate a programului;
- Dezvoltarea gândirii defensive în programare;
- Introducerea treptată a unui concept fundamental al limbajelor moderne: manipularea controlată a situaţiilor neprevăzute.

4. Lipsa conştientizării limitărilor tehnologice ale unui calculator

Un alt efect major al utilizării exclusive a limbajului Python în programă este pierderea unei componente esenţiale a ***alfabetizării informatice***: **conştientizarea limitărilor reale ale resurselor pe un calculator** şi a modului în care acestea influenţează logica şi 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 operaţiilor***.

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 biţi, Python foloseşte un tip *de dimensiune arbitrară* (arbitrary-precision integer).
- Elevul ***nu*** se loveşte niciodată de ***overflow***, depăşiri de interval sau reprezentări limitate,
- Elevul nu înţelege că în computerele reale orice număr are un **cost de stocare** şi un **cost de procesare** dependent de mărimea lui.

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

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

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

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

Aceste situaţii formează **reflexe corecte** despre cum funcţionează un calculator şi reprezintă baza oricărei înţelegeri ulterioare despre performanţă, optimizare, structuri de date şi arhitectură.

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

5. Complexitatea nealiniată a programei propuse

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

În mod concret, elevii sunt puşi în situaţia de a gestiona în paralel:

- noţiuni fundamentale despre variabile, tipuri de date şi memorie;
- manipularea obiectelor complexe precum listele Python;
- conversii explicite între tipuri pentru citirea datelor;
- structurarea unui program în funcţii şi fişiere;
- comportamente interpretate linie cu linie;
- noţiuni de programare orientată pe obiecte (clase, obiecte, metode);
- interfeţe grafice şi 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 învăţa în mod semnificativ. În loc să ***evolueze progresiv*** de la noţiuni simple la concepte mai avansate, elevii sunt expuşi simultan la elemente care, în mod firesc, ar trebui distribuite pe parcursul a doi sau chiar trei ani de studiu

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

Chiar şi operaţii aparent banale, precum:

```
x = 5
```

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

În realitate, această instrucţiune:

- Creează un ***obiect*** al clasei int.
- Instanţiază o ***structură complexă*** cu funcţionalitate 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 şi matematică-informatică, intensiv informatică

construită pe paradigma

C++ → C++ şi Python → Python şi 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 şi a eficienței unui program, în raport cu restricţiile specifice problemei de rezolvat
CG6	Dezvoltarea de soluții informatice originale pentru probleme din contexte ştiinţifice 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 ieşire şi a modului de organizare eficientă a datelor – Principiile programării structurate şi modalitatea de reprezentare a algoritmilor în pseudocod – Etapele dezvoltării aplicaţiilor utilizând un mediu de programare (editarea programului, compilarea programului şi corectarea erorilor de sintaxă, execuţia programului şi depanarea programului – *debug*) – Principii şi metode în elaborarea testelor de evaluare pentru un program – Elemente de analiză a complexităţii algoritmilor	**1/1** **8/10** **1/1** **1/1**
2. Structuri de date	**2.1. Tablouri unidimensionale (vectori)** – Caracteristici ale tablourilor unidimensionale – Modalităţi de declarare a tablourilor unidimensionale – Modalitatea de calcul a dimensiunii spaţiului 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/ştergeri de elemente	**1/1**
	– Repere pentru verificarea unei proprietăţi valabile pentru toate elementele unui tablou unidimensional, respectiv a existenţei unui element care are o anumită proprietate	**1/1**
	– Repere pentru utilizarea vectorilor caracteristici în rezolvarea problemelor	**2/2**
	– Repere pentru utilizarea vectorilor de frecvenţă în rezolvarea problemelor	**2/2**
	– Repere pentru prelucrarea secvenţelor de valori citite succesiv, cu sau fără memorare în tablouri unidimensionale (minime/maxime, secvenţe maximale cu o anumită proprietate)	**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	**/4**
	– Repere pentru utilizarea tablourilor de diferenţe (*difference-arrays*) în rezolvarea problemelor	**/4**
	– Repere pentru prelucrarea secvenţelor de lungime fixată	**/4**
	– Repere pentru implementarea operaţiilor pe numere mari (adunare, scădere, produsul dintre un număr mare şi un număr mic, determinarea câtului şi restului împărţirii unui număr mare la un număr mic, produsul a două numere mari)	**/8**
	– Repere pentru aplicarea tehnicii *Two-pointers* în rezolvarea problemelor	**/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 poziţie specificată, construirea unui număr prin adăugarea unor cifre într-o ordine specificată)	**5/5**
	– 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 primalităţii numerelor naturale, algoritmul de determinare a celui mai mare divizor comun, respectiv a celui mai mic multiplu comun)	**6/6**
	– Algoritmi de calcul pentru a unor expresii matematice (sume, factoriale, puteri)	**1/1**
	– Algoritmi de generare a şirurilor definite prin relaţie de recurenţă	**2/4**
	3.2. Algoritmi de prelucrare a tablourilor unidimensionale	
	– Algoritmul de căutare secvenţială	**1/1**
	– Algoritmul de căutare binară	
	– Algoritmul de sortare prin selecţia minimului/maximului (*Selection sort*)	**1/1**
	– Algoritmul de interclasare a două tablouri unidimensionale	**1/1**
	– Algoritmul de determinare a numerelor prime mai mici decât o valoare dată utilizând *Ciurul lui Eratostene*	**2/4**
	– Algoritmul de sortare utilizând vectori de frecvenţă	**1/2**
	Suplimentar, pentru studiul intensiv al disciplinei informatică **3.1. Algoritmi de prelucrare a datelor numerice – completare**	
	– Sisteme de numeraţie poziţionale. Reguli de conversie între baza 10 şi altă bază de numeraţie	**/2**

Domenii de conținut	Conținuturi	Nr. ore (orientativ) MI/Intensiv
	– Aritmetică modulară – Algoritmul de exponenţiere rapidă **3.2. Algoritmi de prelucrare a tablourilor unidimensionale – completare** – Algoritmul de sortare prin metoda bulelor (*Bubble sort*) – Algoritmul de sortare prin inserţie (*Insertion sort*) – Algoritmul liniar de determinare a unei secvenţe de sumă maximă (*Kadane*)	**/2** **/2** **/1** **/1** **/2**
4. Elemente de limbaj de programare – C++	**4.1. Fişiere text – caracteristici, operaţii de bază** **4.2. Funcţii** – Rolul şi avantajele utilizării funcţiilor – Sintaxa declaraţiei, definiţiei şi apelului unei funcţii – Modalităţi de transfer al parametrilor la apel (transfer prin valoare şi transfer prin referinţă) – Modalitatea de returnare a unei valori – Analiză comparativă a variabilelor locale şi variabilelor globale din punctul de vedere al modului de declarare, clasei de memorare, duratei de viaţă, accesibilităţii, posibilităţii de iniţializare 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 – Modalităţi de declarare a unui tip de date pentru reprezentarea unei structuri de date neomogene şi a variabilelor de acest tip – Modalitatea de acces la un câmp al unei variabile de tip structură	**3/4**
	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 necesităţii utilizării unei stive	**3/5**
	1.3. Structura de date abstractă Coada – Caracteristici ale structurii de date abstracte Coada – Implementarea statică a unei cozi utilizând tablouri unidimensionale – Analiza problemelor în scopul identificării necesităţii utilizării unei cozi	**3/5**
	1.4. Tablouri bidimensionale – Caracteristici ale tablourilor bidimensionale – Modalităţi de declarare a tablourilor bidimensionale – Modalitatea de acces la elementele unui tablou bidimensional – Analiza problemelor în scopul identificării necesităţii utilizării tablourilor bidimensionale – Repere pentru realizarea unor operaţii specifice tablourilor bidimensionale (parcurgere pe linii, parcurgere pe coloane, bordare) – Modalităţi de parcurgere a tuturor vecinilor unui element al unui tablou bidimensional utilizând vectori de direcţie – Caracteristici ale tablourilor bidimensionale pătratice – Repere pentru realizarea unor operaţii specifice tablourilor bidimensionale pătratice (parcurgerea diagonalelor, parcurgerea zonelor delimitate de diagonale, parcurgere pe chenare concentrice)	**8/10** **6/6**
	1.5. Şiruri de caractere – Caracteristici ale şirurilor de caractere	**8/12**

Domenii de conținut	Conținuturi	Nr. ore (orientativ) MI/Intensiv
	– Repere pentru prelucrarea şirurilor de caractere la nivel de caracter – Repere pentru prelucrarea şirurilor de caractere utilizând funcţiile din *cstring*	
	Suplimentar, pentru studiul intensiv al disciplinei informatică **1.4. Tablouri bidimensionale – completare** – Repere pentru utilizarea sumelor parţiale în tablouri bidimensionale **1.6. Tablouri multidimensionale** – Caracteristici ale tablourilor multidimensionale – Modalităţi 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 (*fill*)** – 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 transpoziţiei 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 şi tablouri **4.2. Funcţii recursive** – Caracteristici ale subprogramelor recursive – Mecanismul de realizare a recursivităţii – Modalităţi recursive de implementare a unor algoritmi elementari de prelucrare a cifrelor numerelor, de prelucrare a divizorilor numerelor, de exponenţiere rapidă, de parcurgere a tablourilor unidimensionale	**2/4** **7/8**
	Suplimentar, pentru studiul intensiv al disciplinei informatică **4.2. Funcţii recursive – completare** – Modalităţi recursive de generare a unor elemente combinatoriale (secvenţe binare, submulţimi, produs cartezian, permutări, aranjamente, combinări, partiţii număr, partiţii mulţimi)	**/8** **/10**

Domenii de conținut	Conținuturi	Nr. ore (orientativ) MI/Intensiv
	4.3. Biblioteca STL – Noţiuni de bază necesare utilizării unor clase predefinite: clase, obiecte, constructori, iteratori, accesare membri – Funcţii din biblioteca STL care implementează algoritmi de sortare şi de căutare – Clasa *vector* – Clasa *string* – Clasa *stack* – Clasa *queue*	

Clasa a XI-a

Domenii de conținut	Conținuturi	Nr. ore (orientativ) MI/Intensiv
1. Structuri de date	1.1. **Grafuri neorientate şi grafuri orientate** – Terminologie (graf neorientat, graf orientat, lanţ, lanţ elementar, drum, drum elementar, ciclu, ciclu elementar, circuit, circuit elementar, grad, graf parţial, subgraf, graf transpus, conexitate, tare conexitate, arbore, arbore parţial, graf ponderat, arbore parţial de cost minim, graf orientat aciclic) – Tipuri speciale de grafuri (graf complet, graf hamiltonian, graf eulerian, graf bipartit, graf turneu) – Reprezentarea grafurilor (matrice de adiacenţă, liste de adiacenţă, lista muchiilor) – Reprezentarea grafurilor ponderate (matricea costurilor, liste de adiacenţă cu costuri, lista muchiilor cu costuri)	**6/6**
	Suplimentar, pentru studiul intensiv al disciplinei informatică **1.2. Structuri de date liniare alocate dinamic** – Caracteristici ale listelor simplu înlănţuite. Operaţii specifice – Caracteristici ale listelor simplu înlănţuite circulare – Caracteristici ale listelor dublu înlănţuite **1.3. Structuri de date arborescente** – Caracteristici ale arborilor cu rădăcină – Modalităţi de reprezentare a arborilor cu rădăcină (reprezentare cu referinţe ascendente – vector de taţi, reprezentare cu referinţe descendente) – Caracteristici ale arborilor binari cu rădăcină – Tipuri speciale de arbori binari cu rădăcină (arbore binar plin, arbore binar complet, arbore binar strict) – Modalităţi de reprezentare a arborilor binari cu rădăcină – Parcurgerea arborilor binari cu rădăcină: preordine, inordine, postordine – *Heap*-uri o Caracteristici ale *heap*-urilor o Inserarea unui nod într-un *heap* o Combinarea a două *heap*-uri o Crearea unui heap prin combinări repetate o Extragerea elementului minim/maxim din *heap* o Analiza complexităţii operaţiilor – Arbori binari de căutare o Caracteristici ale arborilor binari de căutare o Inserarea unui nod într-un arbore binar de căutare o Ştergerea unui nod dintr-un arbore binar de căutare	**/4** **/3** **/4** **/4**

Domenii de conținut	Conținuturi	Nr. ore (orientativ) MI/Intensiv
	○ Analiza complexităţii operaţiilor în cazul cel mai defavorabil **1.4 Tabele de dispersie (*hash*)**	**/2**
2. Strategii şi tehnici de programare	**2.1. Metoda de programare *Divide et impera*** – Caracteristici ale problemelor pentru care se poate aplica metoda *Divide et impera* – Forma generică a unui algoritm de tip *Divide et impera* – Repere pentru aplicarea metodei *Divide et impera* în rezolvarea problemelor **2.2. Metoda de programare *backtracking*** – Caracteristici ale problemelor pentru care se poate aplica metoda *backtracking* – varianta elementară – Forma generică a unui algoritm de tip *backtracking* elementar – Repere pentru aplicarea metodei *backtracking* în variantă elementară în rezolvarea problemelor – Caracteristici ale problemelor pentru care se poate aplica metoda *backtracking* – varianta în plan – Forma generică a unui algoritm de tip *backtracking* în plan – Repere pentru aplicarea metodei *backtracking* în plan în rezolvarea problemelor	**12/12** **16/16**
	Suplimentar, pentru studiul intensiv al disciplinei informatică **2.3 Metoda programării dinamice** – Caracteristici ale problemelor pentru care se poate aplica metoda programării dinamice – Etape ale abordării unei probleme prin metoda programării dinamice – Repere pentru aplicarea metodei programării dinamice în rezolvarea problemelor – Tehnici de optimizare a spaţiului de memorie utilizat – Tehnica memoizării	**/14**
3. Algoritmi fundamentali	**3.1. Algoritmi de sortare care utilizează metoda Divide et Impera** – Algoritmul de sortare prin interclasare (*Merge Sort*) **3.2. Algoritmi de prelucrare a grafurilor** – Algoritmul de parcurgere a unui graf în adâncime (DFS – *Depth First Search*) – Algoritmul de parcurgere a unui graf în lăţime (BFS – *Breadth First Search*). – Algoritmul de descompunere a unui graf în componente conexe folosind o metodă de parcurgere – Algoritmul de determinare a câte unui lanț/drum de lungime minimă de la un vârf specificat la fiecare dintre celelalte vârfuri ale grafului utilizând parcurgerea BFS – Algoritmul *Roy-Warshall* de determinare a matricii drumurilor/lanţurilor dintr-un graf. Descompunerea unui graf în componente tare-conexe folosind matricea drumurilor – Algoritmul lui *Prim* de determinare a unui arbore parţial de cost minim – Algoritmul *Roy-Floyd* care determină pentru oricare două vârfuri ale unui graf ponderat costul drumului de cost minim – Algoritmul lui *Dijkstra* care determină pentru un vârf specificat, câte un drum de cost minim către fiecare dintre celelalte vârfuri ale grafului	**2/2** **2/2** **2/2** **1/1** **1/1** **1/1** **2/2** **1/1** **4/4**

Domenii de conținut	Conținuturi	Nr. ore (orientativ) MI/Intensiv
	Suplimentar, pentru studiul intensiv al disciplinei informatică **3.1. Algoritmi de sortare care utilizează metoda Divide et Impera – completare** – Sortarea rapidă (*Quick sort*) **3.2. Algoritmi de prelucrare a grafurilor – completare** – Algoritmul de sortare topologică a vârfurilor unui graf orientat aciclic, utilizând parcurgerea DFS – Algoritmul lui *Kruskal* de determinare a unui arbore parţial de cost minim – Reprezentarea mulţimilor disjuncte. Algoritmii *Union-Find*	**/2** **/2** **/2** **/2**
4. Elemente de limbaj de programare – C++/Python	**4.1. Limbajul de programare Python** – Elemente de bază ale limbajului o Lucrul în modul linie de comandă o Utilizarea unui mediu de programare (IDE) pentru lucrul în limbajul Python o Convenții de scriere: indentare; denumirea variabilelor, funcțiilor și claselor; comentarea codului o Tipuri de date de bază: *int*, *float*, *bool*, *str* o Citirea datelor de la tastatură și afişarea datelor pe ecran o Operatori aritmetici, logici și de comparație o Sintaxa structurilor de control: *if*, *elif*, *else*, *for*, *while*, *break*, *continue*, *pass* o Modalități de tratare a erorilor o Modalități de citire și scriere utilizând fișiere text o Biblioteci standard şi biblioteci externe – Funcții în limbajul Python o Definirea funcțiilor o Apelarea funcțiilor o Returnarea valorilor. Elementul *None* o Funcții cu parametri impliciți – Structurile de date listă şi şir de caractere o definirea listei ca o structură de date secvențială, eterogenă și modificabilă o prezentarea metodelor uzuale pentru tipul de date listă: *append*, *insert*, *remove*, *sort*, *reverse*, *count*, *index* o utilizarea operațiilor fundamentale pentru liste: accesarea elementelor prin index, parcurgerea prin instrucțiuni repetitive, adăugarea și eliminarea de elemente, concatenarea listelor și copierea unei liste o definirea noțiunii de șir de caractere ca secvență imutabilă de caractere o prezentarea operațiilor uzuale pentru șiruri de caractere: *find*, *replace*, *split*, *join*, *upper*, *lower* o *List comprehension* – Structura de date tuplu o definirea tuplurilor ca structuri de date secvențiale și imutabile o utilizarea indexării pentru accesarea elementelor unui tuplu o aplicarea despachetării (*unpacking*) pentru atribuirea multiplă de valori – Structura de date dicționar o definirea dicționarelor ca structuri de date bazate pe perechi de tip cheie-valoare o utilizarea cheilor pentru accesarea și modificarea valorilor dintr-un dicționar	**51/89** **6/8** **6/8** **20/25**

Domenii de conținut	Conținuturi	Nr. ore (orientativ) MI/Intensiv
	o aplicarea metodelor specifice dicționarelor (*keys*, *values*, *items*, *get*, *update*) în prelucrări de date – Structura de date mulțime o definirea mulțimilor ca structuri de date neordonate ce conțin elemente unice o aplicarea operațiilor fundamentale asupra mulțimilor (reuniune, intersecție, diferență, apartenență) o utilizarea metodelor specifice mulțimilor (*add*, *remove*, *union*, *intersection*, *difference*) în prelucrări de date – Tipuri de memorie în Python (mutabilă/imutabilă). *Shallow copy* şi *deep copy* – Programarea orientată pe obiecte o Clase și obiecte, câmpuri şi metode o Constructori şi destructor o Încapsulare o 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** o Accesul la membrii unei clase (public, private, protected) o Moștenire multiplă o Supradefinirea metodelor şi a operatorilor. Polimorfism **4.2. Limbajul C++. Biblioteca STL** – Clasele *set* – Clasele *map* – Clasa *priority_queue*	**/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 învăţare și tipuri de probleme asociate o Învățare nesupervizată. Probleme de tip clusterizare. Algoritmul *K-means* o Învățare supervizată. Probleme de tip regresie și clasificare. Algoritmul de regresie liniară. Arborele de decizie. Algoritmul *KNN (K-Nearest Neighbors)* o 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 o Pericole și riscuri asociate IA (bias, confidențialitate, securitate, autonomie) o Avantaje şi riscuri ale IA generative o Impactul asupra pieței muncii și carierei o Aspecte etice și responsabilitatea utilizării IA o Reglementări și bune practici	**22/28**
2. Structuri de date	**2.1. Modelul conceptual al unei activităţi din lumea reală** – Analiza unei activităţi din lumea reală în scopul identificării datelor relevante şi a relaţiilor dintre acestea	**18/24**

Domenii de conținut	Conținuturi	Nr. ore (orientativ) MI/Intensiv
	– Entităţi şi instanţe – Atribute. Opţionalitate – Identificatori unici – Relaţii între entităţi. Cardinalitate şi opţionalitate o Clasificarea relaţiilor în funcţie de cardinalitate (1:1, 1:M, M:M) o Transferabilitate o Relaţii recursive – Convenţii de reprezentare în diagrama ERD a entităţilor, atributelor, identificatorilor unici, relaţiilor – Rezolvarea relaţiilor M:M – Normalizarea datelor: 1NF, 2NF, 3NF – Transformarea modelului conceptual în model fizic o Tabele, coloane, cheie primară, chei străine (externe), chei unice, constrângeri o Convenţii de descriere a modelului fizic – Modele de baze de date	
	Suplimentar, pentru studiul intensiv al disciplinei informatică **2.1. Modelul conceptual al unei activităţi 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 o Analiza unei activități din lumea reală pentru identificare nevoilor de gestionare o Diferența între *backend* și *frontend* o Formatul de text JSON – Elemente de *Frontend* o Noțiunea de interfață grafică o Alcătuirea schemelor de experiența utilizatorului (*user experience* și *user stories*) o Proiectarea elementelor vizuale în concordanță cu schemele alcătuite o Implementarea unei interfețe grafice adecvate în limbajul Python – Elemente de *Backend* o Identificarea funcționalităților necesare aplicației o Implementarea eficientă a acestor funcționalități în limbajul Python o Gestionarea datelor aplicației cu ajutorul unei baze de date o 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 *pandas* – 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 *DataFrame*-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 *Matplotlib* și *Seaborn* – 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 învăţare automată. Biblioteca *scikit-learn* – împărțirea datelor în seturi de antrenare și testare (*train_test_split*) – scalarea și transformarea datelor cu *StandardScaler*, *MinMaxScaler*, *LabelEncoder*, *OneHotEncoder* – crearea modelelor de învățare supervizată și nesupervizată din biblioteca *scikit-learn* – antrenarea și testarea modelelor (*fit*, *predict*, *score*) – evaluarea performanței modelelor **4.2. Limbajul SQL** – Transformarea modelului fizic într-o bază de date relaţională – Crearea tabelelor (CREATE TABLE) – Modificarea structurii unei tabele (ALTER TABLE) – Ștergerea tabelelor (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. Selecţie, proiecţie, ordonare – Expresii și funcții – Expresii aritmetice și logice – Funcții pe rânduri individuale (*single-row*): text, numerice, date – Subinterogări în clauza WHERE – Interogări avansate (JOIN)	**20/28**
	Suplimentar, pentru studiul intensiv al disciplinei informatică **4.1. Biblioteci Python pentru Inteligenţa artificială – completare** – Elemente de algebră liniară. Biblioteca *NumPy*. Operații cu tablouri în *NumPy* **4.2. Limbajul SQL – completare** – Agregare și grupare – Funcții de agregare (*multiple row*): COUNT, SUM, AVG, MIN, MAX – Gruparea datelor (GROUP BY) – Filtrarea grupurilor (HAVING) – Subinterogări corelate – Crearea şi utilizarea secvenţelor – Indecşi şi sinonime – Gestionarea drepturilor de acces la baza de date – Gestionarea tranzacţiilor **4.3. Biblioteci Python pentru Dezvoltarea Aplicațiilor Web** – Interacțiunea cu o bază de date. Biblioteca *sqlite* – 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	**/8** **/16**

Domenii de conținut	Conținuturi	Nr. ore (orientativ) MI/Intensiv
	– Adăugarea unei interfețe grafice. Biblioteca *gradio* – 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 şi informatica studiată în liceu – Principii ale lucrului în echipă – Managementul unui proiect în domeniul IT o Rolul managementului de proiect o Etapele unui proiect (ciclul de viață) o Roluri în managementul de proiect o Instrumente și metode – Reguli pentru susținerea unei prezentări de proiect	**4/8**