test-e.ro

CAIET ELEV

Lecția 30

Biblioteca Tkinter. Interfețe grafice simple

Interfețe grafice simple cu biblioteca Tkinter, privită ca ansamblu de clase predefinite: fereastra principală (Tk) și componentele Label, Button, Entry, Text, Frame, Canvas, Listbox; așezarea cu pack(), grid() și place(); programarea condusă de evenimente cu mainloop(), command= și bind(); citirea datelor cu get() (întotdeauna text) și afișarea cu config() sau messagebox; Listbox cu insert(), delete(), curselection() pentru afișarea unei liste Python.

DISCIPLINA Informatică **CLASA** a IX-a · mate-info **LIMBAJ** Python 3

NUMELE ELEVULUI .....

CUM FOLOSEȘTI CAIETUL

Rezolvă exercițiile pe hârtie: urmărești execuția pas cu pas, completezi, asociezi și scrii singur soluțiile, apoi le verifici pe platformă. Nou este biblioteca tkinter (import tkinter as tk): programele cu ferestre le rulezi în laborator, în IDLE sau Thonny; în rest exersezi logica din spatele interfeței, cu funcții def și input()/print(). Ce ai voie să folosești: def/return, atribuirea multiplă, min/max/sum, slicing, listele cu metodele lor, for pe liste, metodele clasei str. Variabilele globale sunt permise, dar problemele se rezolvă prin parametri și return. Fără lambda, fără sorted()/.sort(), fără clase proprii. Fără diacritice în codul Python; culorile Tkinter se scriu în engleză (red, green, yellow).

CUPRINS

1 Ce știm deja .....
2 Prima fereastră .....
3 Butonul care calculează .....
4 Unde stă fiecare buton .....
5 Desene, mesaje și taste .....
6 Coșul de cumpărături .....
7 Cod ghidat · șase probleme .....

SECȚIUNEA 1 · REACTUALIZARE

1 Ce știm *deja*

Înainte de lecția nouă, ne încălzim cu trei unelte pe care le vom folosi din plin: subprogramele care primesc parametri și întorc un rezultat, listele cu metodele lor și obiectele ale căror metode le apelăm cu operatorul punct.

Un **subprogram** (funcție) se definește cu `def`, primește valori prin **parametri** și trimite înapoi un rezultat cu `return`. Rezultatul întors poate fi un număr, dar și un **șir de caractere** — de exemplu un mesaj gata de afișat. Apelul `salut("Ana")` se înlocuiește, în expresie, cu valoarea întoarsă.

De exemplu:

```
1  def mesaj(nume):
2      return "Salut, " + nume + "!"
3
4  print(mesaj("Ana"))
```

O **listă** Python este un obiect al clasei `list`: `append(x)` adaugă la final, `pop(i)` scoate elementul de pe poziția `i` și îl întoarce, `len(lista)` spune câte elemente are. Pozițiile se numără **de la 0**.

La lecția 29 am văzut că șirurile, listele și fișierele sunt **obiecte**: au date și **metode**, iar metodele se apelează cu operatorul punct — `nume.upper()`, `produse.pop(0)`, `fisier.read()`.

> **TEXT SAU NUMĂR?**
>
> `input()` întoarce întotdeauna un **șir de caractere**. `"3" + "4"` lipește textele și dă `"34"`; ca să aduni numerele, convertește întâi cu `int()`: `int("3") + int("4")` dă `7`.

A EXERCIȚIUL A · GRILĂ

Recapitulare: funcții, liste, obiecte

GRILĂ

Încercuiește răspunsul corect la fiecare întrebare.

1. Ce afișează programul? 9 6 12 36

2.

După instrucțiunile <code>produse = ["paine", "lapte", "oua"]</code> și <code>produse.pop(0)</code>, ce conține lista?

<code>["lapte", "oua", "paine"]</code> <code>["paine", "oua"]</code>

<code>["lapte", "oua"]</code> <code>["paine", "lapte"]</code>

3. În expresia <code>nume.upper()</code>, ce reprezintă <code>upper</code>?

<code>O variabilă nouă, creată automat de Python pentru <code>nume</code></code>

<code>O metodă a obiectului <code>nume</code>, apelată cu operatorul punct</code>

<code>O funcție din biblioteca <code>math</code>, aplicată asupra șirului</code>

<code>Un atribut al clasei <code>list</code>, care păstrează lungimea șirului</code>

4. Ce afișează instrucțiunea <code>print("3" + "4")</code>?

<code>o eroare, pentru că nu se pot aduna șiruri</code> <code>7</code> <code>34</code> <code>3 + 4</code>

B

EXERCIȚIUL B · DRY-RUN

Câți divizori are 8?

DRY-RUN

Urmărește apelul numara_divizori(8) pas cu pas. Coloana d este dată; tu completezi restul împărțirii lui n la d și valoarea contorului nr după fiecare pas. Toate valorile de completat sunt numerice.

```
1  def numara_divizori(n):
2      nr = 0
3      d = 1
4      while d <= n:
5          if n % d == 0:
6              nr = nr + 1
7          d = d + 1
8      return nr
9
10 print(numara_divizori(8))
```

d	n % d	nr
1	0	1
2		
3		
4		
5		
6		
7		
8		
9 — condiția 9 <= 8 e falsă, bucla se oprește	—	4

HINT *La fiecare pas calculezi restul n % d; când restul este 0, d este divizor și nr crește cu 1, altfel nr rămâne neschimbat.*

SPRIJIN · SCRIE CODUL

Mesajul potrivit orei

SCRIERE COD

Se citesc, pe două rânduri, un nume și o oră ora (număr natural între 0 și 23). Scrie funcția salut(nume, ora) care întoarce mesajul Buna dimineata, <nume>! dacă ora este mai mică decât 12, Buna ziua, <nume>! dacă ora este între 12 și 17 (inclusiv), și Buna seara, <nume>! în rest. Programul afișează mesajul întors de funcție. Ex.: Pentru numele Ana și ora 9 se afișează Buna dimineata, Ana!; pentru Maria și ora 18 se afișează Buna seara, Maria!.

HINT **Observație.** *Funcția nu afișează nimic: ea întoarce mesajul cu return, iar print se face în afara ei.*

SOLUȚIA TA · PYTHON

SPRIJIN · SCRIE CODUL

Scot un produs din listă

SCRIERE COD

Se citește un număr natural n (n ≥ 1), apoi n produse, câte unul pe rând, și la final o poziție p (numerotată de la 0, 0 ≤ p < n). Scoate din listă produsul de pe poziția p. Afișează pe primul rând sters: urmat de produsul scos, iar pe al doilea rând ramase: urmat de numărul de produse rămase. Ex.: Pentru n = 3, produsele paine, lapte, oua și p = 1 se afișează sters: lapte și apoi ramase: 2.

HINT **Atenție.** *Pozițiile se numără de la 0: p = 0 înseamnă primul produs.*

SOLUȚIA TA · PYTHON

SECȚIUNEA 2 · PRIMA FEREASTRĂ: TK, LABEL, BUTTON

2 Prima *fereastră*

Biblioteca Tkinter este un ansamblu de clase predefinite pentru ferestre, butoane și etichete. Construim prima fereastră și aflăm de ce un program cu interfață nu se mai termină după ultima instrucțiune.

Tkinter vine instalată odată cu Python pe Windows și macOS; pe unele distribuții Linux se instalează separat (pachetul `python3-tk`). O importăm cu un nume scurt: `import tkinter as tk`, iar apoi fiecare clasă se scrie cu prefixul `tk.` (`tk.Tk`, `tk.Label`, `tk.Button`).

Orice aplicație începe cu **fereastra principală**, un obiect al clasei `Tk`. Pe ea se pun **componente** (în engleză *widgets*): etichete (`Label`) care afișează text, butoane (`Button`) pe care se face click. La crearea unei componente, **primul argument este părintele** — fereastra în care stă. Restul argumentelor au nume: `text="..."`, `width=...`, `bg="..."` (culoarea de fundal).

Un program complet:

```
1  import tkinter as tk
2
3  fereastra = tk.Tk()
```

```
4  fereastra.title("Prima mea fereastra")
5  fereastra.geometry("300x150")
6
7  eticheta = tk.Label(fereastra, text="Salut, Tkinter!")
8  eticheta.pack()
9
10 buton = tk.Button(fereastra, text="Inchide",
11                  command=fereastra.destroy)
12 buton.pack()
13
14 fereastra.mainloop()
```

Să citim programul cu ochii lecției 29. `tk.Tk()` **instanțiază** clasa `Tk` și obține obiectul `fereastra`. `title(...)` și `geometry("300x150")` sunt **metode** ale ferestrei: prima schimbă textul din bara de sus, a doua stabilește lățimea și înălțimea în pixeli. `pack()` este metoda care **așază** componenta în fereastră. `command=fereastra.destroy` spune butonului ce metodă să apeleze la click — aici, închiderea ferestrei.

FĂRĂ PACK(), NIMIC PE ECRAN

O etichetă creată cu `tk.Label(...)` există ca obiect în memorie, dar **nu apare** în fereastră până nu o așezi cu `pack()` (sau cu `grid()` / `place()`, pe care le vedem la S4). Nu primești nicio eroare — pur și simplu nu vezi nimic.

PROGRAMARE CONDUSĂ DE EVENIMENTE

Până acum, programele se executau de sus în jos și se terminau. `mainloop()` schimbă regula: pornește o buclă care **așteaptă** acțiunile utilizatorului (click, tastă apăsată, închiderea ferestrei) și reacționează la fiecare. Programul se încheie abia când fereastra este închisă. De aceea `mainloop()` este **ultima** instrucțiune a programului.

UNDE RULEZI PROGRAMELE CU FERESTRE

Pagina lecției rulează Python în browser (Pyodide), unde nu se pot deschide ferestre Tkinter. Programele complete din teorie le rulezi în laborator, în IDLE sau Thonny. Pe pagină exersezi **logica** din spatele interfeței.

UNDE SE FOLOSEȘTE ÎN VIAȚA REALĂ

Tot ce apeși pe telefon sau pe calculator funcționează prin evenimente: aplicația de mesaje stă „în buclă" și reacționează când atingi butonul *Trimite*; un bancomat așteaptă să apeși o tastă înainte să facă ceva; chiar și IDLE, editorul în care scrii Python, este construit cu Tkinter.

A

EXERCIȚIUL A · GRILĂ

Fereastra și componentele ei

GRILĂ

Încercuiește răspunsul corect la fiecare întrebare.

1. În instrucțiunea <code>fereastra = tk.Tk()</code>, ce este <code>fereastra</code>?

   - O metodă a clasei <code>Label</code> care afișează textul pe ecran
   - Un obiect (o instanță) al clasei <code>Tk</code>
   - O funcție care deschide fereastra de fiecare dată când e apelată
   - Numele bibliotecii tkinter, prescurtat pentru scriere rapidă

2. Ce se întâmplă dacă scrii <code>eticheta = tk.Label(fereastra, text="Salut")</code>, dar uiți <code>eticheta.pack()</code>?

   - Eticheta apare în colțul din stânga sus, fără nicio aliniere
   - Fereastra nu se mai deschide deloc, iar programul se încheie
   - Programul se oprește cu o eroare de sintaxă la linia etichetei
   - Eticheta există ca obiect, dar nu apare în fereastră

3. Ce face instrucțiunea <code>fereastra.mainloop()</code>?

   - Repetă de la început, la infinit, toate instrucțiunile programului
   - Desenează o singură dată fereastra, apoi programul se încheie imediat
   - Pornește bucla care așteaptă și tratează evenimentele până la închiderea ferestrei
   - Închide automat fereastra după ce utilizatorul apasă primul buton

4. În <code>tk.Label(fereastra, text="Salut")</code>, ce reprezintă primul argument, <code>fereastra</code>?

   - Numele variabilei în care se salvează eticheta creată
   - Fereastra-părinte în care se pune eticheta
   - Textul care se afișează pe etichetă
   - Culoarea de fundal pe care o primește eticheta

B

EXERCIȚIUL B · ORDONARE

Scheletul oricărui program Tkinter

ORDONARE

Numerotează instrucțiunile de la 1 la 5, în ordinea în care trebuie să apară într-un program care afișează o fereastră cu o etichetă. Instrucțiunile îți apar amestecate.

- [ ] fereastra = tk.Tk()

- [ ] import tkinter as tk
- [ ] eticheta = tk.Label(fereastra, text="Salut!")
- [ ] fereastra.mainloop()
- [ ] eticheta.pack()

C

EXERCIȚIUL C · COMPLETARE

Cele trei piese esențiale

COMPLETARE

Completează golurile folosind cuvintele din rezervă (trei cuvinte rămân nefolosite).

`pack` `mainloop` `Tk` `Label` `input` `print`

Fereastra principală este un obiect al clasei _____; o componentă apare în fereastră abia după apelul metodei _____, iar programul rămâne deschis, așteptând acțiunile utilizatorului, datorită metodei _____.

3 Butonul care *calculează*

SECȚIUNEA 3 · EVENIMENTE ȘI DATE: GET(), COMMAND, CONFIG()

Un buton devine util abia când face ceva. Legăm un buton de o funcție, citim ce a scris utilizatorul într-o casetă și afișăm rezultatul într-o etichetă — și învățăm să despărțim calculul de interfață.

Caseta în care utilizatorul scrie o valoare pe un singur rând este un obiect al clasei `Entry`. Ce a scris utilizatorul se citește cu metoda `get()`. Pentru a reacționa la un click, butonul primește argumentul `command=` urmat de **numele** unei funcții — funcția pe care Tkinter o apelează la fiecare click. Rezultatul se afișează schimbând textul unei etichete cu metoda `config(text=...)`.

Iată un calculator care adună două numere. Observă că **calculul** stă într-o funcție separată, care nu știe nimic despre ferestre:

```
1  import tkinter as tk
2
3  def suma_text(text_a, text_b):
4      return "Suma: " + str(int(text_a) + int(text_b))
5
6  def la_apasare():
```

```
    rezultat.config(text=suma_text(caseta_a.get(), caseta_b.get()))

fereastra = tk.Tk()
fereastra.title("Calculator")
caseta_a = tk.Entry(fereastra)
caseta_a.pack()
caseta_b = tk.Entry(fereastra)
caseta_b.pack()
buton = tk.Button(fereastra, text="Aduna", command=la_apasare)
buton.pack()
rezultat = tk.Label(fereastra, text="Suma: ?")
rezultat.pack()
fereastra.mainloop()
```

Dacă utilizatorul scrie 12 și 7 și apasă „Aduna", eticheta devine `Suma: 19`. Funcția `la_apasare` nu primește parametri: Tkinter o apelează singur, la click, iar ea citește casetele, apelează `suma_text` și afișează rezultatul.

COMMAND=FUNCTIE, FĂRĂ PARANTEZE

Scrii `command=la_apasare`, nu `command=la_apasare()`. Cu paranteze, funcția se execută **o singură dată, chiar în momentul creării butonului**, iar butonului i se dă ca „funcție" rezultatul ei (`None`) — click-urile ulterioare nu mai fac nimic.

GET() ÎNTOARCE ÎNTOTDEAUNA TEXT

Chiar dacă utilizatorul scrie cifre, `caseta.get()` întoarce un șir de caractere: `"12"`, nu `12`. `"12" + "7"` dă `"127"`. Convertește cu `int()` înainte de calcule și cu `str()` înainte de a lipi numărul într-un mesaj. Dacă o casetă este goală, `get()` întoarce `""`, iar `int("")` produce eroarea `ValueError`.

SEPARAREA LOGICII DE INTERFAȚĂ

Funcția care calculează (`suma_text`) primește valori și întoarce un rezultat, fără să atingă vreo casetă sau etichetă. Funcția legată de buton (`la_apasare`) doar citește casetele, o apelează și afișează. Avantajul: logica se poate verifica fără fereastră, cu `print(suma_text("12", "7"))`. Exact așa sunt construite problemele de pe această pagină — tu scrii logica, iar nota „Observație" îți arată cum o legi de interfață în laborator.

UNDE SE FOLOSEȘTE ÎN VIAȚA REALĂ

Formularul de autentificare dintr-o aplicație citește ce ai scris în casetele *utilizator* și *parolă* abia când apeși *Conectare*; aplicația de calculator a telefonului face exact ce face programul de mai sus; la magazinele online, butonul *Adaugă în coș* este legat de o funcție care actualizează

totalul afișat.

A

EXERCIȚIUL A · GRILĂ

Evenimente și date

GRILĂ

Încercuiește răspunsul corect la fiecare întrebare.

1. Ce se întâmplă la instrucțiunea <code>tk.Button(fereastra, text="OK", command=salut())</code>?

   - Programul se oprește imediat cu o eroare, pentru că lipsesc ghilimelele
   - <code>salut()</code> se execută de fiecare dată când utilizatorul apasă butonul OK
   - <code>salut()</code> se execută de două ori: la crearea butonului și la primul click
   - <code>salut()</code> se execută o singură dată, la crearea butonului, iar click-ul nu mai face nimic

2. Utilizatorul scrie 25 în caseta <code>varsta</code> (un obiect <code>Entry</code>). Ce întoarce <code>varsta.get()</code>?

   - Numărul real 25.0, pentru că Entry lucrează cu zecimale
   - Șirul de caractere <code>"25"</code>
   - Numărul întreg 25, gata de folosit în calcule
   - Lista <code>[2, 5]</code>, cu cifrele scrise de utilizator

3. Cum schimbi textul afișat de eticheta <code>rezultat</code> în „Gata"?

   - <code>rezultat.insert(0, "Gata")</code>
   - <code>rezultat.config(text="Gata")</code>
   - <code>rezultat.get(text="Gata")</code>
   - <code>print(rezultat, "Gata")</code>

4. De ce separăm funcția care calculează de funcția legată de buton?

   - Pentru că logica se poate testa fără fereastră, iar interfața doar o apelează
   - Pentru că programele cu interfață nu au voie să folosească <code>return</code>
   - Pentru că altfel butoanele nu s-ar mai afișa în fereastra principală
   - Pentru că Tkinter nu acceptă funcții cu parametri în niciun program

5. Aceeași aplicație de divizori există în două variante: una în consolă, cu <code>input()</code> și <code>print()</code>, și una cu ferestre Tkinter. Care comparație este corectă?

   - Cele două variante au nevoie de funcții de calcul diferite, pentru că Tkinter nu acceptă return
   - Varianta în consolă nu poate primi date de la utilizator, ci doar afișează rezultate pe ecran

Varianta cu ferestre e mai ușor de folosit, dar are mai mult cod; cea în consolă e mai scurtă și ușor de testat

Varianta cu ferestre calculează mai repede, pentru că Tkinter folosește alt procesor decât consola

B

EXERCIȚIUL B · SCRIE CODUL

Calculator simplu

SCRIERE COD

Scrie funcția calculeaza(text_a, text_b, operatie), care primește două numere întregi scrise ca text (exact cum le întoarce get()) și o operație (+, -, * sau :). Funcția întoarce Rezultat: urmat de rezultatul operației, unde : înseamnă câtul împărțirii întregi. Dacă operația este : și al doilea număr este 0, funcția întoarce Impartire la zero. Programul citește cele trei valori, câte una pe rând, și afișează ce întoarce funcția. Ex.: Pentru 12, 7 și + se afișează Rezultat: 19; pentru 17, 5 și : se afișează Rezultat: 3; pentru 8, 0 și : se afișează Impartire la zero.

HINT Atenție. *Parametrii sunt text. "12" + "7" ar da "127" — convertește-i cu int() înainte de calcul și rezultatul cu str() înainte de a-l lipi în mesaj.*

HINT Observație. *În laborator, funcția se leagă de interfață așa: două casete Entry, câte un buton pentru fiecare operație și o etichetă rezultat; butonul + apelează o funcție care face rezultat.config(text=calculeaza(caseta_a.get(), caseta_b.get(), "+")).*

SOLUȚIA TA · PYTHON

C

EXERCIȚIUL C · SCRIE CODUL

Mesaj cu numărul de divizori

SCRIERE COD

Scrie funcția mesaj_divizori(text), care primește un număr natural n scris ca text (n ≥ 1) și întoarce mesajul <n> are <k> divizori, unde k este numărul divizorilor lui n. Dacă n are un singur divizor, mesajul este <n> are 1 divizor (la singular). Programul citește un număr natural q (q ≥ 1), apoi q numere, câte unul pe rând, și afișează, pe câte un rând, mesajul întors de funcție pentru fiecare număr. Ex.: Pentru q = 2 și numerele 12 și 1 se afișează 12 are 6 divizori și, pe rândul următor, 1 are 1 divizor (divizorii lui 12 sunt 1, 2, 3, 4, 6 și 12).

HINT **Atenție.** *Singularul contează: o interfață care scrie „1 are 1 divizori" arată neglijent. Numără întâi divizorii, apoi alege forma mesajului.*

HINT **Observație.** *În laborator, mesajul se afișează într-o fereastră separată: messagebox.showinfo("Divizori", mesaj_divizori(caseta.get())) (vezi S5).*

SOLUȚIA TA · PYTHON

D

EXERCIȚIUL D · SCRIE CODUL

Divizori comuni — aplicație cu fereastră

SCRIERE COD

În editor este o aplicație Tkinter completă: are o casetă și un buton Calculeaza, iar la click afișează într-o fereastră-mesaj câți divizori are numărul scris în casetă. Modific-o astfel: adaugă a doua casetă Entry, iar la click fereastra-mesaj (cu titlul Divizori) să afișeze a si b au k divizori comuni, unde a și b sunt numerele din cele două casete. Dacă numerele au un singur divizor comun, mesajul este a si b au 1 divizor comun. Ex.: Pentru 12 și 18 fereastra-mesaj arată 12 si 18 au 4 divizori comuni (1, 2, 3 și 6); pentru 7 și 5 arată 7 si 5 au 1 divizor comun.

HINT **Cum se verifică.** *Pe pagină nu se pot deschide ferestre, așa că verificatorul folosește o fereastră simulată și joacă rolul utilizatorului: scrie numere în casete (în ordinea în care le-ai creat), apasă butonul Calculeaza și citește fereastra-mesaj. În rezultatele verificării, fereastra-mesaj apare scrisă așa: [showinfo] Divizori: 12 si 18 au 4 divizori comuni. Dacă uiți pack() sau mainloop(), verificatorul nu „vede” caseta sau butonul — exact ca un utilizator adevărat.*

HINT **Atenție.** *Singularul contează și aici, ca la problema anterioară. Fereastra-mesaj se afișează cu messagebox.showinfo(titlu, mesaj) — o vei întâlni pe larg în S5.*

SOLUȚIA TA · PYTHON

PERFORMANȚĂ · SCRIE CODUL

Calculatorul cu o singură casetă — exercițiu de performanță

SCRIERE COD

Un calculator real are o singură casetă, în care utilizatorul scrie toată expresia, de exemplu 12 + 7. Scrie funcția evalueaza(expresie), care primește textul casetei, format din două numere întregi și un operator, separate prin spații. Funcția întoarce expresia urmată de = și rezultat (de exemplu 12 + 7 = 19). Dacă operatorul nu este +, - sau *, întoarce Operatie necunoscuta. Dacă textul nu are exact trei bucăți separate prin spații, întoarce Expresie invalida. Programul citește expresia și afișează ce întoarce funcția. Ex.: Pentru 30 - 45 se afișează 30 - 45 = -15; pentru 8 / 2 se afișează Operatie necunoscuta; pentru 15 + se afișează Expresie invalida.

HINT **Observație.** *Față de problema anterioară, aici tu trebuie să despachetezi singur textul. Metoda split() a clasei str (lecția 29) împarte textul la spații și întoarce o listă de bucăți.*

HINT **Atenție.** *Verifică numărul de bucăți înainte de a le folosi: la 15 + lista are doar 2 elemente, iar parti[2] ar produce o eroare.*

SOLUȚIA TA · PYTHON

SECȚIUNEA 4 · AȘEZAREA COMPONENTELOR ȘI FRAME

4 Unde stă fiecare *buton*

O interfață bună are fiecare componentă la locul ei. Tkinter oferă trei metode de așezare și un container, Frame, cu care construim, de exemplu, tastatura unui calculator.

Tkinter are trei **metode de așezare** (în engleză *geometry managers*):

PACK()

Pune componentele **una sub alta**, în ordinea în care apelezi `pack()`. E cea mai simplă și e potrivită pentru ferestre cu puține componente.

GRID(ROW=R, COLUMN=C)

Pune componenta într-un **tabel invizibil**, pe rândul `r` și coloana `c`. Rândurile și coloanele se numără **de la 0**. E potrivită pentru formulare și tastaturi.

PLACE(X=..., Y=...)

Pune componenta la **coordonate exacte**, în pixeli, măsurate de la colțul din stânga sus al părintelui: `eticheta.place(x=20, y=40)` o așază la 20 de pixeli de marginea stângă și 40 de pixeli de marginea de sus.

Pentru a combina metodele, folosim `Frame`: un container invizibil care grupează alte componente. Frame-ul se așază în fereastră ca orice componentă, iar componentele din interiorul lui pot folosi alt sistem de așezare.

Tastatura unui calculator, cu butoanele 1–9 pe 3 coloane, într-un Frame:

```
import tkinter as tk

fereastra = tk.Tk()
fereastra.title("Tastatura")
ecran = tk.Label(fereastra, text="0", width=12)
ecran.pack()
tastatura = tk.Frame(fereastra)
tastatura.pack()
for k in range(1, 10):
    buton = tk.Button(tastatura, text=str(k), width=4)
    buton.grid(row=(k - 1) // 3, column=(k - 1) % 3)
fereastra.mainloop()
```

Formula de poziționare este cea cunoscută de la împărțirea cu rest: butonul `k` ajunge pe rândul `(k - 1) // 3` și coloana `(k - 1) % 3`. Scădem 1 pentru că butoanele se numără de la 1, iar rândurile și coloanele de la 0. Butonul 5 ajunge pe rândul 1, coloana 1 — adică exact în mijloc.

NU AMESTECA PACK() ȘI GRID() ÎN ACELAȘI PĂRINTE

Dacă în fereastră ai așezat o componentă cu `pack()` și încerci să așezi alta, tot direct în fereastră, cu `grid()`, Tkinter oprește programul cu eroarea `TclError`. Soluția este Frame-ul: în programul de mai sus, eticheta și Frame-ul folosesc `pack()` în fereastră, iar butoanele folosesc `grid()` **în interiorul** Frame-ului.

UȘOR DE FOLOSIT („USER-FRIENDLY”)

Aceeași logică poate avea o interfață clară sau una încurcată — diferența o face așezarea. Într-un formular, etichetele stau pe coloana 0 și casetele pe coloana 1, ca să fie aliniate; butoanele înrudite stau împreună, într-un `Frame`; butonul principal vine la final, unde ajunge privirea după completarea casetelor; iar textele spun exact ce face fiecare componentă („Calculează”, nu „OK”). Când alegi între `pack()`, `grid()` și `place()`, întreabă-te întâi ce îi e mai ușor utilizatorului.

UNDE SE FOLOSEȘTE ÎN VIAȚA REALĂ

Tastatura numerică a telefonului și cea a bancomatului sunt grile de 3 coloane, exact ca în formula de mai sus; formularele de înscriere așază etichetele pe coloana 0 și casetele pe coloana 1, ca să fie aliniate; galeria de poze a telefonului pune imaginile într-o grilă, iar numărul de coloane se schimbă când rotești ecranul.

A

EXERCIȚIUL A · GRILĂ

Metode de așezare

GRILĂ

Încercuiește răspunsul corect la fiecare întrebare.

1.

Butoanele sunt numerotate de la 1 și așezate cu <code>grid(row=(k - 1) // 3, column=(k - 1) % 3)</code>. Unde ajunge butonul 8?

`Rândul 2, coloana 1` `Rândul 1, coloana 2` `Rândul 3, coloana 2` `Rândul 2, coloana 2`

2.

În fereastra principală, o etichetă e așezată cu <code>pack()</code>. Ce se întâmplă dacă o a doua etichetă, tot din fereastra principală, e așezată cu <code>grid()</code>?

`Tkinter oprește programul cu eroarea <code>TclError</code>`

A doua etichetă apare sub prima, la fel ca la <code>pack()</code>

Prima etichetă dispare și rămâne doar cea așezată cu <code>grid()</code>

A doua etichetă este ignorată, fără niciun mesaj de eroare

3. Ce face instrucțiunea <code>eticheta.place(x=20, y=40)</code>?

Așază eticheta la 20 de pixeli de marginea de sus și 40 de pixeli de cea stângă

Stabilește lățimea etichetei la 20 de pixeli și înălțimea ei la 40 de pixeli

Așază eticheta pe rândul 20 și coloana 40 ale unui tabel invizibil

Așază eticheta la 20 de pixeli de marginea stângă și 40 de pixeli de cea de sus

4. La ce folosește un obiect <code>Frame</code>?

Desenează un chenar colorat în jurul ferestrei principale a aplicației

Grupează mai multe componente, care pot fi apoi așezate împreună, cu alt sistem

Creează o a doua fereastră principală, separată de prima

Afișează un mesaj într-o fereastră mică, peste fereastra principală

B

EXERCIȚIUL B · ASOCIERE

Cine ce face?

ASOCIERE

Fiecărui element din stânga îi corespunde o singură descriere din dreapta. Scrie în căsuță litera descrierii potrivite. Fiecare literă se folosește o singură dată.

	<code>pack()</code>
	<code>grid(row=..., column=...)</code>
	<code>place(x=..., y=...)</code>
	<code>Frame</code>

A Așază componenta pe un rând și o coloană, ca într-un tabel

B Așază componenta la coordonate exacte, în pixeli

C Așază componentele una sub alta, în ordinea apelurilor

D Container invizibil care grupează alte componente

C

EXERCIȚIUL C · DRY-RUN

Tastatura cu 3 coloane

DRY-RUN

Urmărește bucla care așază 7 butoane pe o grilă cu 3 coloane. Coloana k este dată; tu completezi rândul și coloana în care ajunge fiecare buton. Valorile de completat sunt numerice.

```
1  import tkinter as tk
2
3  fereastra = tk.Tk()
4  coloane = 3
5  k = 1
6  while k <= 7:
7      rand = (k - 1) // coloane
8      col = (k - 1) % coloane
9      buton = tk.Button(fereastra, text=str(k))
10     buton.grid(row=rand, column=col)
11     k = k + 1
12 fereastra.mainloop()
```

k	rand = (k - 1) // coloane	col = (k - 1) % coloane
1	0	0
2		
3		
4		
5		
6		
7		
8 — condiția 8 <= 7 e falsă, bucla se oprește	—	—

HINT *Rândul este câtul împărțirii lui k - 1 la 3, iar coloana este restul aceleiași împărțiri. Scade 1 înainte de a împărți: butoanele se numără de la 1, rândurile și coloanele de la 0.*

D

EXERCIȚIUL D · SCRIE CODUL

Harta tastaturii

SCRIERE COD

Scrie funcția pozitie(k, coloane), care întoarce rândul și coloana butonului cu numărul k (butoanele se numără de la 1, rândurile și coloanele de la 0) într-o grilă cu numărul de coloane dat de parametrul coloane. Programul citește numărul de coloane coloane (coloane ≥ 1) și numărul de butoane n (n ≥ 0). Pentru fiecare buton, de la 1 la n, afișează un rând de forma k: rand r, coloana c. La final afișează randuri: urmat de numărul de rânduri ocupate de grilă. Ex.: Pentru coloane = 2 și n = 5 se afișează 1: rand 0, coloana 0, 2: rand 0, coloana 1, 3: rand 1, coloana 0, 4: rand 1, coloana 1, 5: rand 2, coloana 0 și randuri: 3.

HINT **Atenție.** *Numărul de rânduri nu este n // coloane: pentru 5 butoane pe 2 coloane, 5 // 2 = 2, dar ultimul buton stă singur pe al treilea rând. Gândește-te cum rotunjești în sus, doar cu //. Iar pentru n = 0 grila nu are niciun rând.*

HINT **Observație.** *Funcția poate întoarce două valori deodată: return rand, col, iar la apel le primești cu rand, col = pozitie(k, coloane). În laborator, exact aceste valori se dau lui buton.grid(row=rand, column=col).*

SOLUȚIA TA · PYTHON

SECȚIUNEA 5 · DESENE, MESAJE ȘI TASTE: BIND()

5 Desene, mesaje și *taste*

Completăm trusa de componente: caseta Text pe mai multe rânduri, suprafața de desen Canvas, ferestrele-mesaj din messagebox și legarea tastelor cu bind(). Totul se adună într-un mini-proiect: semaforul interactiv.

`Text` este o casetă pe **mai multe rânduri** (un mic editor). Pozițiile în ea se scriu ca text de forma `"rand.coloana"`, cu rândurile numerotate de la 1 și coloanele de la 0: `"1.0"` înseamnă începutul textului, iar `tk.END` sfârșitul lui.

```
jurnal = tk.Text(fereastra, height=5, width=30)
jurnal.pack()
jurnal.insert("1.0", "Linia 1\nLinia 2")
continut = jurnal.get("1.0", tk.END)
```

Aici `continut` devine `"Linia 1\nLinia 2\n"` — Text adaugă mereu **un salt de rând la final**, spre deosebire de Entry.

`Canvas` este o **suprafață de desen**. Metoda `create_oval(x1, y1, x2, y2, fill="red")` desenează un cerc (o elipsă) încadrat în dreptunghiul cu colțurile `(x1, y1)` și `(x2, y2)` și întoarce un **număr** care identifică desenul. Cu acest număr, desenul se poate modifica ulterior: `panou.itemconfig(bec, fill="green")` îi schimbă culoarea, iar `panou.itemcget(bec, "fill")` citește culoarea curentă.

Modulul `messagebox` afișează ferestre-mesaj mici, peste fereastra principală. Se importă separat: `from tkinter import messagebox`.

> **CELE PATRU FERESTRE-MESAJ**
>
> `messagebox.showinfo(titlu, mesaj)` — informare; `messagebox.showwarning(titlu, mesaj)` — avertizare; `messagebox.showerror(titlu, mesaj)` — eroare; `messagebox.askyesno(titlu, intrebare)` — întrebare cu butoanele Da/Nu, care **întoarce** `True` sau `False`, după răspunsul utilizatorului.

Pe lângă `command=`, pe care îl au butoanele (și alte câteva componente de control), orice componentă poate reacționa la un eveniment cu metoda `bind(eveniment, functie)`: `fereastra.bind("<space>", la_tasta)` apelează `la_tasta` la fiecare apăsare a tastei Space, `caseta.bind("<Return>", la_enter)` la fiecare Enter.

Semaforul interactiv (mini-proiectul din planificare). Culoarea curentă nu o ținem într-o variabilă separată, ci o citim chiar din desen:

```python
import tkinter as tk
from tkinter import messagebox

def urmatoarea(culoare):
    if culoare == "red":
        return "green"
    elif culoare == "green":
        return "yellow"
    else:
        return "red"

def schimba():
    noua = urmatoarea(panou.itemcget(bec, "fill"))
    panou.itemconfig(bec, fill=noua)
    if noua == "red":
        messagebox.showinfo("Semafor", "Stop! Culoarea este rosie.")

def la_tasta(event):
    schimba()

fereastra = tk.Tk()
fereastra.title("Semafor")
panou = tk.Canvas(fereastra, width=120, height=120, bg="black")
panou.pack()
bec = panou.create_oval(20, 20, 100, 100, fill="red")
buton = tk.Button(fereastra, text="Schimba", command=schimba)
buton.pack()
fereastra.bind("<space>", la_tasta)
fereastra.mainloop()
```

Becul pornește roșu; fiecare click pe „Schimba" sau fiecare apăsare pe Space îl trece în ciclul roșu → verde → galben → roșu, iar la revenirea pe roșu apare mesajul „Stop!". Culorile se scriu în engleză (`red`, `green`, `yellow`), pentru că așa le recunoaște Tkinter.

> **FUNCȚIA LEGATĂ CU BIND PRIMEȘTE UN PARAMETRU**
>
> Tkinter îi trimite automat un obiect cu detaliile evenimentului (ce tastă, unde s-a dat click), numit de obicei `event`. Dacă scrii `def la_tasta():` fără parametru, la prima apăsare pe Space apare eroarea `TypeError`. În schimb, funcțiile legate cu `command=` **nu** primesc nimic. De aceea, în program, `la_tasta(event)` doar apelează `schimba()`.

UNDE SE FOLOSEȘTE ÎN VIAȚA REALĂ

Fereastra „Doriți să salvați modificările?" care apare când închizi un document este un `askyesno`; în jocuri, apăsarea tastelor săgeți este legată cu `bind` de funcțiile care mișcă personajul; aplicațiile de hărți desenează traseul pe o suprafață de tip Canvas, iar semafoarele inteligente dintr-o intersecție rulează un ciclu de stări exact ca funcția `urmatoarea`.

A

EXERCIȚIUL A · GRILĂ

Text, Canvas, messagebox, bind

GRILĂ

Încercuiește răspunsul corect la fiecare întrebare.

1.

Caseta <code>jurnal</code> (de tip <code>Text</code>) conține <code>Linia 1</code> pe primul rând și <code>Linia 2</code> pe al doilea. Ce întoarce <code>jurnal.get("1.0", tk.END)</code>?

- <code>"Linia 1 Linia 2"</code>
- <code>"Linia 1\nLinia 2"</code>
- <code>"Linia 1\nLinia 2\n"</code>
- <code>["Linia 1", "Linia 2"]</code>

2.

Ce fereastră-mesaj pune o întrebare cu butoanele Da/Nu și întoarce <code>True</code> sau <code>False</code>?

- <code>messagebox.askyesno</code>
- <code>messagebox.showinfo</code>
- <code>messagebox.showerror</code>
- <code>messagebox.showwarning</code>

3.

Ce trebuie să aibă funcția <code>la_enter</code>, legată cu <code>caseta.bind("<Return>", la_enter)</code>?

- Un parametru, de obicei numit <code>event</code>, primit automat de la Tkinter
- Doi parametri: tasta apăsată și fereastra în care s-a apăsat
- Niciun parametru, exact ca funcțiile legate cu <code>command=</code>
- Obligatoriu o instrucțiune <code>return True</code> la sfârșitul funcției

4.

După <code>bec = panou.create_oval(20, 20, 100, 100, fill="red")</code>, ce se memorează în variabila <code>bec</code>?

- O copie a cercului desenat, ca obiect de clasa <code>Oval</code>
- Coordonatele cercului, sub forma listei <code>[20, 20, 100, 100]</code>
- Culoarea cercului, adică șirul de caractere <code>"red"</code>
- Un număr întreg care identifică desenul pe suprafața Canvas

B

EXERCIȚIUL B · COMPLETARE

Mesaje și evenimente

COMPLETARE

Completează golurile folosind cuvintele din rezervă (trei cuvinte rămân nefolosite).

`command` `showinfo` `showerror` `event` `get` `askyesno`

Funcția messagebox. _____ afișează un mesaj de informare, funcția messagebox. _____ întoarce True sau False după răspunsul utilizatorului, iar o funcție legată cu bind primește automat parametrul _____.

C

EXERCIȚIUL C · SCRIE CODUL

Ciclul semaforului

SCRIERE COD

Scrie funcția urmatoarea(culoare), care primește culoarea curentă a unui semafor (rosu, verde sau galben) și întoarce culoarea următoare, în ciclul roșu → verde → galben → roșu. Programul citește culoarea inițială și numărul k de apăsări pe butonul „Schimba" (k ≥ 0). Afișează culoarea inițială, apoi, pe câte un rând, culoarea de după fiecare apăsare. Ex.: Pentru culoarea rosu și k = 3 se afișează, pe rânduri separate, rosu, verde, galben, rosu.

HINT **Atenție.** *Pentru k = 0 nu se apasă niciodată butonul, deci se afișează doar culoarea inițială.*

HINT **Observație.** *În programul din teorie aceeași funcție lucrează cu red, green, yellow, pentru că Tkinter recunoaște culorile după numele lor în engleză; aici folosim numele românești, fără diacritice.*

SOLUȚIA TA · PYTHON

SECȚIUNEA 6 · LISTBOX: COȘUL DE CUMPĂRĂTURI

6 Coșul de *cumpărături*

Listbox afișează o listă Python rând cu rând și îi permite utilizatorului să aleagă. Construim un coș de cumpărături: produsele apar în listă, se elimină cu un buton, iar totalul se confirmă printr-o fereastră-mesaj.

`Listbox` este o componentă care afișează **rânduri de text**, dintre care utilizatorul poate selecta unul cu un click. Rândurile se numerotează **de la 0**, ca elementele unei liste Python. Metodele de bază:

> **METODELE CLASEI LISTBOX**
>
> `insert(poz, text)` — adaugă un rând pe poziția `poz` (`tk.END` înseamnă la final); `delete(i)` — șterge rândul `i`; `delete(0, tk.END)` — golește toată lista; `get(i)` — întoarce textul rândului `i`; `size()` — întoarce numărul de rânduri; `curselection()` — întoarce pozițiile rândurilor selectate.

Metoda `curselection()` întoarce un **tuplu** — o secvență asemănătoare cu lista, scrisă între paranteze rotunde, care se indexează la fel, dar nu se poate modifica. Dacă e selectat al doilea rând, `curselection()` întoarce `(1,)` (virgula arată că e un tuplu cu un singur element), iar poziția se obține cu `curselection()[0]`. Dacă nu e selectat nimic, întoarce tuplul gol `()`, cu `len(...) == 0`.

Pentru a afișa o listă Python într-un Listbox, o parcurgem și inserăm fiecare element la final. Coșul de cumpărături din planificare ține produsele și prețurile în două liste paralele, pe care le actualizează odată cu Listbox-ul:

```
import tkinter as tk
from tkinter import messagebox

produse = ["paine", "lapte", "oua"]
preturi = [5, 8, 12]

def elimina():
    selectie = lista.curselection()
    if len(selectie) == 0:
        messagebox.showwarning("Cos", "Selecteaza intai un produs.")
    else:
        i = selectie[0]
```

```
        lista.delete(i)
        produse.pop(i)
        preturi.pop(i)

def confirma():
    messagebox.showinfo("Cos",
                        "Total de plata: " + str(sum(preturi)) + " lei")

fereastra = tk.Tk()
fereastra.title("Cos de cumparaturi")
lista = tk.Listbox(fereastra)
lista.pack()
for i in range(len(produse)):
    lista.insert(tk.END,
                 produse[i] + " - " + str(preturi[i]) + " lei")
buton_elimina = tk.Button(fereastra, text="Elimina produsul",
                          command=elimina)
buton_elimina.pack()
buton_confirma = tk.Button(fereastra, text="Confirma",
                           command=confirma)
buton_confirma.pack()
fereastra.mainloop()
```

Dacă utilizatorul selectează „lapte - 8 lei" și apasă „Elimina produsul", apoi „Confirma", apare mesajul `Total de plata: 17 lei`. Dacă apasă „Elimina produsul" fără să fi selectat ceva, primește avertizarea în loc de o eroare.

> **LISTBOX-UL ȘI LISTA PYTHON TREBUIE ȚINUTE SINCRONIZATE**
>
> Listbox-ul arată doar text; calculele (totalul) se fac pe listele Python. De aceea, la eliminare, poziția `i` se șterge **din toate trei**: `lista.delete(i)`, `produse.pop(i)` și `preturi.pop(i)`. Dacă uiți una, totalul afișat nu mai corespunde cu ce vede utilizatorul.

> **UNDE SE FOLOSEȘTE ÎN VIAȚA REALĂ**
>
> Lista de contacte din telefon, din care alegi pe cine suni; lista de melodii dintr-o aplicație de muzică, din care selectezi ce redai sau ce ștergi; coșul unui magazin online, în care fiecare produs are un buton *Elimină* și totalul se recalculează imediat.

A

EXERCIȚIUL A · GRILĂ

Listbox

GRILĂ

Încercuiește răspunsul corect la fiecare întrebare.

1.

Un Listbox conține rândurile <code>paine</code>, <code>lapte</code>, <code>oua</code>, iar utilizatorul a selectat <code>lapte</code>. Ce întoarce <code>lista.curselection()</code>?

<code>[1]</code> <code>(1,)</code> <code>"lapte"</code> <code>2</code>

2. Ce întoarce <code>lista.curselection()</code> dacă utilizatorul nu a selectat niciun rând?

Poziția primului rând, adică <code>(0,)</code> Tuplul gol <code>()</code>

Valoarea <code>None</code>, care înseamnă „nimic"

Numărul <code>-1</code>, ca poziție inexistentă

3. Cum ștergi toate rândurile unui Listbox numit <code>lista</code>?

<code>lista.delete(tk.END, 0)</code> <code>lista.insert(0, tk.END)</code>

<code>lista.delete(0, tk.END)</code> <code>lista.pop(0, tk.END)</code>

4.

Un Listbox conține rândurile <code>paine</code>, <code>oua</code>. Ce conține după <code>lista.insert(0, "mere")</code>?

<code>paine</code>, <code>mere</code>, <code>oua</code>

<code>mere</code>, <code>paine</code>, <code>oua</code>

<code>paine</code>, <code>oua</code>, <code>mere</code>

<code>mere</code>, <code>oua</code>

B

EXERCIȚIUL B · DRY-RUN

Ce e în coș?

DRY-RUN

Pornești cu un Listbox gol, cos, și execuți instrucțiunile de mai jos, în ordine. Instrucțiunea este dată pe fiecare rând; tu completezi conținutul Listbox-ului după ea (produsele de sus în jos, separate prin virgulă și spațiu — valoare de tip text) și numărul de rânduri, cos.size() (valoare numerică).

```
1  import tkinter as tk
2
3  fereastra = tk.Tk()
4  cos = tk.Listbox(fereastra)
```

```
cos.pack()
cos.insert(tk.END, "paine")
cos.insert(tk.END, "lapte")
cos.insert(0, "mere")
cos.insert(tk.END, "oua")
cos.delete(1)
cos.insert(2, "unt")
cos.delete(0)
fereastra.mainloop()
```

Instrucțiunea	Conținutul Listbox-ului după instrucțiune	cos.size()
cos.insert(tk.END, "paine")	paine	1
cos.insert(tk.END, "lapte")		
cos.insert(0, "mere")		
cos.insert(tk.END, "oua")		
cos.delete(1)		
cos.insert(2, "unt")		
cos.delete(0)		

HINT *insert(0, ...) pune rândul primul, insert(tk.END, ...) îl pune ultimul, iar după delete(i) rândurile de dedesubt urcă cu un loc. Scrie produsele de sus în jos, separate prin virgulă și spațiu.*

C

EXERCIȚIUL C · SCRIE CODUL

Coșul de cumpărături

SCRIERE COD

Se citește un număr natural n (n ≥ 1), apoi n rânduri de forma produs pret (un cuvânt și un preț întreg, în lei), și la final poziția p a rândului selectat în Listbox (numerotată de la 0), sau -1 dacă nu e selectat nimic. Dacă p nu este -1, elimină produsul de pe poziția p. Afișează apoi produsele rămase, fiecare pe un rând, în forma produs - pret lei, iar pe ultimul rând Total: urmat de suma prețurilor și lei. Dacă nu a rămas niciun produs, afișează Cos gol înainte de total. Ex.: Pentru produsele paine 5, lapte 8, oua 12 și p = 1 se afișează paine - 5 lei, oua - 12 lei și Total: 17 lei.

HINT **Atenție.** *Produsul și prețul de pe poziția p trebuie scoase din ambele liste; altfel totalul include un preț al unui produs care nu mai apare.*

HINT **Observație.** *Valoarea -1 joacă rolul tuplului gol () întors de curselection() când nu e selectat nimic: nu se elimină nimic, dar totalul se afișează.*

SOLUȚIA TA · PYTHON

7 SECȚIUNEA 7 · COD GHIDAT
Scrie cod care *trece testele*

1 PROBLEMA 7.1 · COD GHIDAT
Verificatorul de parolă

SCRIERE COD

Scrie funcția verifica(parola), care întoarce PUTERNICA dacă parola are cel puțin 8 caractere, cel puțin o cifră și cel puțin o literă mare, și SLABA în caz contrar. Programul citește un număr natural n (n ≥ 1), apoi n parole, câte una pe rând. Pentru fiecare parolă afișează, pe câte un rând, parola, textul -> și verdictul întors de funcție. Ex.: Pentru n = 2 și parolele Parola123 și parola123 se afișează, pe rânduri separate, Parola123 -> PUTERNICA și parola123 -> SLABA (a doua parolă nu are literă mare).

HINT **Observație.** *Un caracter c este cifră dacă c >= "0" and c <= "9" și literă mare dacă c >= "A" and c <= "Z" — comparațiile folosesc codurile ASCII (lecția 13). Parcurge parola caracter cu caracter, cu for c in parola.*

HINT Curiozitate. *În formularele reale de înscriere, această funcție este legată cu bind("<KeyRelease>", ...) de caseta parolei, iar eticheta de sub casetă se schimbă din „slabă" în „puternică" pe măsură ce tastezi.*

SOLUȚIA TA · PYTHON

2 PROBLEMA 7.2 · COD GHIDAT

Contorul cu trei butoane

SCRIERE COD

O aplicație are o etichetă care arată un număr, pornind de la 0, și trei butoane: + (crește numărul cu 1), - (scade numărul cu 1, dar numai dacă este mai mare decât 0) și R (readuce numărul la 0). Scrie funcția apasa(valoare, buton), care întoarce valoarea de după apăsarea butonului. Programul citește numărul k de apăsări (k ≥ 0), apoi k butoane, câte unul pe rând, și afișează valoarea finală a etichetei. Ex.: Pentru apăsările +, +, -, +, + valoarea trece prin 1, 2, 1, 2, 3, deci se afișează 3.

HINT Atenție. *Butonul - nu coboară sub 0: pentru apăsările -, -, + valoarea rămâne 0, 0 și devine 1.*

HINT Observație. *În laborator, fiecare buton are propria funcție legată cu command=, de exemplu def la_plus(): eticheta.config(text=str(apasa(int(eticheta.cget("text")), "+"))) — valoarea curentă se citește chiar din etichetă, cu cget("text").*

SOLUȚIA TA · PYTHON

3

PROBLEMA 7.3 · COD GHIDAT

Media din catalog

SCRIERE COD

Scrie funcția media(note), care primește o listă de note (numere naturale de la 1 la 10) și întoarce media lor aritmetică, rotunjită la 2 zecimale cu round. Programul citește numărul de note n (n ≥ 1), apoi cele n note, câte una pe rând, și afișează Media: urmat de valoarea întoarsă de funcție. Ex.: Pentru notele 9, 8, 9 se afișează Media: 8.67.

HINT **Observație.** *În aplicația de catalog, notele stau într-un Listbox, iar un buton „Calculeaza media" apelează funcția și afișează rezultatul cu messagebox.showinfo. Poți folosi sum și len (lecția 21).*

SOLUȚIA TA · PYTHON

4

PROBLEMA 7.4 · REPARĂ CODUL

Calculatorul care lipește numerele

SCRIERE COD

Un coleg a scris logica pentru un calculator cu două casete și un buton „Aduna". Programul lui, de mai jos, primește ce întoarce get() (două rânduri de text) și ar trebui să afișeze Suma: urmat de suma numerelor. Pentru 12 și 7, programul afișează însă Suma: 127. Repară programul astfel încât să afișeze suma corectă.

Programul de corectat:

```
1  def aduna(text_a, text_b):
2      return text_a + text_b
3
4  a = input()
5  b = input()
6  print("Suma: " + aduna(a, b))
```

Ex.: Pentru 12 și 7, programul reparat afișează Suma: 19.

HINT **Atenție.** *Nu e suficient să adaugi int() într-un singur loc: dacă funcția întoarce un număr, lipirea lui cu "Suma: " produce altă eroare. Urmărește tipul valorii la fiecare pas: text → număr pentru adunare, număr → text pentru afișare.*

HINT **Observație.** *Este exact capcana din teoria S3: get() întoarce întotdeauna text, iar + între două texte le lipește.*

PROGRAMUL CORECTAT · PYTHON

PROBLEMA 7.5 · PROVOCARE

Căutarea în listă, pe măsură ce tastezi

SCRIERE COD

Scrie funcția filtreaza(nume, prefix), care primește o listă de nume și un text prefix și întoarce o listă nouă cu numele care încep cu prefix, în ordinea în care apar, fără să țină cont de literele mari sau mici. Programul citește un număr natural n (n ≥ 0), apoi n nume, câte unul pe rând, și la final prefixul (cel puțin un caracter). Afișează numele găsite, câte unul pe rând, sau Niciun rezultat dacă nu există niciunul. Ex.: Pentru numele Ana, Andrei, Bogdan, Anca și prefixul an se afișează, pe rânduri separate, Ana, Andrei, Anca.

HINT **Observație.** *Un nume x începe cu prefixul p dacă primele len(p) caractere ale lui sunt egale cu p: x[0:len(p)] == p (slicing, lecția 22). Ca să nu conteze literele mari, compară variantele cu lower() ale ambelor.*

HINT **Curiozitate.** *Așa funcționează căutarea din lista de contacte a telefonului: caseta de căutare este legată cu bind("<KeyRelease>", ...), iar la fiecare tastă apăsată Listbox-ul se golește cu delete(0, tk.END) și se reumple cu rezultatul funcției filtreaza.*

SOLUȚIA TA · PYTHON

PROBLEMA 7.6 · PROVOCARE CU FEREASTRĂ

Șirul care crește

SCRIERE COD

Șirul $a_0 = 3$, $a_n = 2 \cdot a_{n-1}$ are termenii 3, 6, 12, 24, … Scrie o aplicație Tkinter cu o etichetă cu fundal colorat, care arată la început 3, și două butoane, cu textele Start și Stop. După apăsarea pe Start, eticheta trece la fiecare secundă la termenul următor al șirului. Stop oprește schimbarea, iar un nou Start continuă de la termenul afișat. Apăsarea repetată pe Start nu trebuie să accelereze șirul.

HINT Idee. *fereastra.after(1000, pas) programează un apel al funcției pas peste 1000 de milisecunde și întoarce un identificator; fereastra.after_cancel(identificator) anulează apelul. Funcția pas schimbă eticheta și se programează din nou pe ea însăși. Ca să păstrezi identificatorul între apeluri fără global, ține-l într-o listă cu un singur element: programat = [None].*

HINT Cum se verifică. *Verificatorul folosește o fereastră simulată: apasă butoanele, lasă să treacă timpul (de exemplu 3 secunde) și citește textul etichetei. După Start și 3 secunde, eticheta trebuie să arate 24.*

HINT Atenție. *Dacă la Stop doar „știi minte” că șirul e oprit, un apel after() deja programat rămâne în așteptare. Un Stop urmat imediat de Start poate porni atunci două lanțuri de apeluri, iar șirul crește de două ori pe secundă. Gândește-te ce trebuie anulat la Stop.*

SOLUȚIA TA · PYTHON