МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ВАСИЛЯ СТУСА
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ І ПРИКЛАДНИХ ТЕХНОЛОГІЙ

Р. М. Бабаков

МЕТОДИЧНІ РЕКОМЕНДАЦІЇ ДО ВИКОНАННЯ ЛАБОРАТОРНИХ РОБІТ З ДИСЦИПЛІНИ Основи програмування

для здобувачів освіти ОС «Бакалавр» денної форми навчання
спеціальності 122 Комп’ютерні науки

Вінниця
2024

УДК 004.42 (076.5)

Б12

Рекомендовано до друку вченою радою факультету інформаційних і прикладних технологій
Донецького національного університету імені Василя Стуса
(протокол № 1 від 27 серпня 2024 р.)

Автор: *Р. М. Бабаков,* д-р техн. наук, доцент, доцент кафедри інформаційних технологій Донецького національного університету імені Василя Стуса;

Рецензенти: *О. В. Зелінська,* канд. техн. наук, доцент, завідувач кафедри інформаційних технологій Донецького національного університету імені Василя Стуса;
Т. В. Січко, канд. техн. наук, доцент, доцент кафедри інформаційних технологій Донецького національного університету імені Василя Стуса.

Р. М. Бабаков
Б12 Методичні рекомендації до виконання лабораторних робіт з дисципліни «Основи програмування» для здобувачів освіти СО «Бакалавр» денної форми навчання спеціальності 122 Комп'ютерні науки. Вінниця: ДонНУ імені Василя Стуса, 2024. 133 с.

Методичні рекомендації є навчально-методичним документом, який містить рекомендації для отримання практичних навичок написання програм мовою Python під час виконання практикуму з дисципліни «Основи програмування».

Для здобувачів ступеня освіти «Бакалавр» спеціальності 122 Комп'ютерні науки факультету інформаційних і прикладних технологій ДонНУ імені Василя Стуса.

УДК 004.42 (076.5)

ЗМІСТ

ВСТУП

В основі сучасного світу інформаційних технологій лежить використання комп'ютерного програмного забезпечення (ПЗ). Здатність проєктувати та розробляти ПЗ є однією з фахових компетенцій Стандарту вищої освіти України першого (бакалаврського) рівня за спеціальністю 122 «Комп'ютерні науки». Різноманітність типів ПЗ та складність його розробки вимагають особливого підходу до викладання програмування для здобувачів вищої освіти. Складниками цього підходу можна визначити такі:

- щосеместрове викладання дисциплін, пов'язаних із програмуванням;
- тісний міждисциплінарний зв'язок цих дисциплін;
- перевага практичного складника дисциплін над теоретичним;
- значна частка самостійної роботи здобувачів освіти.

Оволодіння основами програмування доцільно розпочинати з вивчення мови програмування, яка, з одного боку, є відносно простою (має низький поріг входження), а з іншого боку, є універсальною за переліком вирішуваних прикладних задач. Сьогодні такою мовою є Python. Її перевагами з погляду використання у навчальному процесі є безкоштовність, велика кількість середовищ розробки, простий синтаксис, наявність великої кількості бібліотечних модулів (фреймворків), можливість використання браузерних онлайн-середовищ розробки, кросплатформовість та інші. Недоліком мови Python є відносно низька швидкодія, що пов'язана з використанням інтерпретатора, однак для оволодіння базовими основами і методами програмування цей фактор не виглядає критичним.

У цих методичних рекомендаціях здобувачам освіти пропонується двосеместровий цикл лабораторних робіт з програмування мовою Python. У кожному семестрі передбачається виконання восьми лабораторних робіт та індивідуального творчого завдання, метою якого є агрегація знань, отриманих у навчальному семестрі. Методичний матеріал до кожної

лабораторної роботи включає набір індивідуальних завдань, що обираються згідно з номером здобувача освіти в журналі групи, та детальні пояснення щодо виконання роботи. Навчальний курс «Основи програмування» викладається в Донецькому національному університеті імені Василя Стуса для здобувачів освіти СО «Бакалавр» спеціальності 122 Комп’ютерні науки.

Лабораторна робота № 1

Прості типи даних у мові програмування Python

Метою лабораторної роботи є отримання студентами навичок програмного опису об’єктів навколишнього світу за допомогою простих типів даних мови програмування Python.

Кожен студент отримує індивідуальний варіант завдання із табл. 1.1. Номер рядка таблиці відповідає номеру студента в загальному списку студентів групи (не в підгрупі). Якщо номер студента перебільшує кількість рядків таблиці, рух по таблиці починається знову з першого рядка. Наприклад, якщо в таблиці 30 рядків, а у студента номер варіанта дорівнює 55, він обирає 55 – 30 = 25 номер варіанта.

Варіантом завдання є три об’єкти з навколишнього світу, що вказані у відповідному рядку таблиці. Для заданого варіанта завдання необхідно розробити програму мовою Python, у якій виконати таке:

1. Описати об’єкт 1 за допомогою множини змінних, серед яких мають бути:

– не менш ніж 5 змінних цілого типу (int);

– не менш ніж 5 змінних дійсного типу (float);

– не менш ніж 5 змінних символьного (рядкового) типу (str).

2. Задати значення змінних так, щоб вони були більш-менш реальними. Значення задавати константами, без вводу з клавіатури.

3. Вивести значення змінних на екран у порядку «важливості» даних. Наприклад, якщо це людина, то спочатку повинні виводитись прізвище, ім’я та по батькові, потім – вік, стать, рік народження, потім – посада, професія та ін. Порядок створення цих змінних у коді програми може бути довільним, але виведення на екран – у порядку важливості.

4. У коді програми біля кожної команди зі створенням змінної додати коментар, що пояснює призначення цієї змінної.

5. Виконати пункти 1–4 для об’єкта 2.

6. Виконати пункти 1–4 для об’єкта 3.

7. Підготувати та захистити звіт з лабораторної роботи.

Вміст звіту з лабораторної роботи

1. Титульний аркуш, оформлений за стандартом університету.
2. Індивідуальний варіант завдання (рядок з табл. 1.1).
3. Лістинг програми.
4. Знімки екрана з результатами роботи програми.
5. Висновки до лабораторної роботи.

Таблиця 1.1 – Варіанти завдань до лабораторної роботи № 1

Приклад оформлення титульного аркуша лабораторної роботи (оформлюється на всю сторінку, рамка не потрібна):

Міністерство освіти і науки України
Донецький національний університет імені Василя Стуса
Факультет інформаційних і прикладних технологій

Кафедра інформаційних технологій

З В І Т

з лабораторної роботи № 1
з дисципліни «Основи програмування»
на тему:
«Прості типи даних в мові Python»

Виконав: студент гр. КН-24-А
Губчакевич В. Р.

Перевірив: доц. Бабаков Р. М.

Вінниця – 2024

Приклад виконання лабораторної роботи (для одного об’єкта)

Здійснимо програмний опис об’єкта «Лекція».

За допомогою символьних змінних можна задати такі властивості об’єкта:

– назва університету, де читається лекція;

– назва навчального предмета;

– прізвище, ім’я та по батькові викладача;

– назва лекції;

– назва групи, якій читається лекція.

За допомогою змінних цілого типу можна описати такі властивості об’єкта:

– номер семестру, в якому читається лекція (від 1 до 8);

– порядковий номер лекції;

– кількість сторінок у файлі з лекцією;

– кількість рисунків;

– кількість таблиць.

За допомогою змінних дійсного типу можна описати такі властивості об’єкта:

– розмір у кілобайтах, який займає файл з лекцією (наприклад, 37.2);

– очікувана середня кількість студентів на лекції (наприклад, 19.8);

– розмір лекції, виражений у друкованих аркушах (наприклад, 0.4);

– частка матеріалу лекції, яка може бути винесена на самостійне вивчення (наприклад, 0.2);

– ймовірність зустріти матеріал цієї лекції в екзаменаційному білеті (наприклад, 0.1).

Опишемо ці параметри набором змінних та надамо їм початкові значення:

```
vnz = "ДонНУ імені Василя Стуса"      # Назва вишу
predmet = "Дискретна математика"      # Назва предмета
vikladach = "Бабаков Роман Маркович" # Викладач
name = "Алгебра множин"               # Назва лекції
group = "КН-д23"                      # Студентська група

semestr = 1                  # Номер семестру
number = 2                   # Номер лекції
pages = 7                    # Кількість сторінок
figures = 4                  # Кількість рисунків
tables = 0                   # Кількість таблиць
```

```
file_size = 37.2     # Розмір файла в Кб
students = 19.8      # Середня кількість студентів
arkush = 0.4         # Розмір у друкованих аркушах
samost = 0.2         # Частка лекції на самостійне вивчення
p_exam = 0.1         # Ймовірність матеріалу в екзам. білеті
```

Додамо команди виведення змінних на екран:

```
print("Предмет:", predmet)
print("Назва лекції:", name)
print("Викладач", vikladach)
print("Група:", group)
print("Назва вишу:", vnz)

print()

print("Семестр:", semestr)
print("Лекція №", number)
print("Сторінок:", pages, end="")
print(", рисунків:", figures, end="")
print(", таблиць:", tables)

print()

print("Розмір файла:", file_size, "Кб")
print("Середня кількість студентів на лекції:", students)
print("Кількість друкованих аркушів:", arkush)
print("Частка лекції на самостійне вивчення:", samost*100, "%")
print("Зустрічається в", p_exam*100, "% екзаменаційних білетів")
```

Результат роботи програми:

```
Предмет: Дискретна математика
Назва лекції: Алгебра множин
Викладач Бабаков Роман Маркович
Група: КН-д23
Назва вишу: ДонНУ імені Василя Стуса

Семестр: 1
Лекція № 2
Сторінок: 7, рисунків: 4, таблиць: 0

Розмір файла: 37.2 Кб
Середня кількість студентів на лекції: 19.8
Кількість друкованих аркушів: 0.4
Частка лекції на самостійне вивчення: 20.0 %
Зустрічається в 10.0 % екзаменаційних білетів
```

Приклад висновків до лабораторної роботи

Висновки

Внаслідок виконання лабораторної роботи я навчився використовувати прості типи даних мови програмування Python для опису об’єктів навколишнього світу, а також засвоїв виведення інформації на екран за допомогою функції print.

Лабораторна робота № 2

Оператори і вирази

Метою лабораторної роботи є отримання студентами навичок побудови арифметичних виразів з урахуванням пріоритетів операцій.

Завдання до лабораторної роботи

Варіантом завдання є алгебраїчний вираз, який обирається з табл. 2.1 відповідно до номера студента в журналі групи. Для заданого виразу необхідно виконати таке:

1. Написати програму, яка вводить значення необхідних змінних із клавіатури, обчислює вираз відповідно до пріоритету операцій та виводить результат на екран.
2. Спростити вираз, якщо це можливо (на папері).
3. Додати в програму обчислення спрощеного виразу та виведення результату на екран. Для спрощеного виразу використовувати ті ж самі значення змінних, що й для не спрощеного виразу (значення, введені з клавіатури на початку програми).
4. Підрахувати кількість операцій у виразі до та після спрощення.
5. Зробити висновки про правильність та ефективність спрощення.

Таблиця 2.1 – Індивідуальні варіанти завдань

№	Алгебраїчний вираз
1	$\left(\sqrt{\sqrt{m}-\sqrt{\frac{m^2-9}{m}}}+\sqrt{\sqrt{m}+\sqrt{\frac{m^2-9}{m}}}\right)^2 \cdot \sqrt[4]{\frac{m^2}{4}}$
2	$\frac{t^5+64t^{-1}}{t^3-4t+16t^{-1}} : \frac{t^2+4}{2}$
3	$\frac{a^2b+b^2a}{a+b} \cdot \left(1+\frac{1}{a^2-ab}\right) - \frac{b}{a-b}$
4	$\frac{1}{(m+n)^2}\left(\frac{1}{m^2}+\frac{1}{n^2}\right)+\frac{2}{(m+n)^3}\left(\frac{1}{m}+\frac{1}{n}\right)$
5	$\left(\frac{2a-b}{a+b}-\frac{2b+a}{b-a}\right) \cdot \left(\frac{a^2-b^2}{3} : \left(a^2+b^2\right)\right)$

6	$\frac{c}{a+b}:\left(\frac{a^2-b^2}{c^3}\right)^{-1}\cdot\frac{c^2}{a-b}$
7	$\frac{a^2-b^2}{a^2+ac}:\frac{3a+3b}{3ac+3c^2}\cdot\frac{1}{a-b}$
8	$\frac{r^3+s^3}{r+s}:\left(r^2-s^2\right)-\frac{rs}{r^2-s^2}+\frac{2s}{r+s}$
9	$\left(\frac{x-y}{xy}+\frac{3x+y}{x^2-xy}+\frac{3y+x}{xy-y^2}\right):\frac{2(x+y)}{xy}+\frac{2x}{y-x}$
10	$\left(\frac{x}{xy+y^2}+\frac{x-y}{x^2-xy}\right):\left(\frac{y^2}{x^3-xy^2}+\frac{1}{x-y}\right)$
11	$\frac{2b+a-\frac{4a^2-b^2}{a}}{b^3+2ab^2-3a^2b}\cdot\frac{a^3b-2a^2b^2+ab^3}{a^2-b^2}$
12	$\frac{\left(\frac{a}{b}+1\right)^2}{\frac{a}{b}-\frac{b}{a}}\cdot\frac{\frac{a^3}{b^3}-1}{\frac{a^2}{b^2}+\frac{a}{b}+1}:\frac{\frac{a^3}{b^3}+1}{\frac{a^2}{b^2}-\frac{a}{b}+1}$
13	$\frac{3a^2+2ax-x^2}{(3x+a)(a+x)}-2+10\cdot\frac{ax-3x^2}{a^2-9x^2}$
14	$\frac{a^{-1}-b^{-1}}{a^{-3}+b^{-3}}:\frac{a^2b^2}{(a+b)^2-3ab}\cdot\left(\frac{a^2-b^2}{ab}\right)^{-1}$
15	$\left(\frac{\sqrt{a}+2}{\left(\sqrt{a}+1\right)^2}-\frac{\sqrt{a}-2}{a-1}\right)\cdot\frac{\sqrt{a}+1}{\sqrt{a}}$
16	$\frac{16\sqrt{ab}-16a-4b}{4a-b}+\frac{18\sqrt{a}+\sqrt{b}}{2\sqrt{a}+\sqrt{b}}$
17	$\frac{\sqrt{x^3}+\sqrt{xy^2}-\sqrt{x^2y}-\sqrt{y^3}}{\sqrt[4]{y^5}+\sqrt[4]{x^4y}-\sqrt[4]{xy^4}-\sqrt[4]{x^5}}$

18	$\left(\frac{\sqrt{a}}{2}-\frac{1}{2\sqrt{a}}\right)^2\cdot\left(\frac{\sqrt{a}-1}{\sqrt{a}+1}-\frac{\sqrt{a}+1}{\sqrt{a}-1}\right)$
19	$\left(p+\frac{\sqrt{q^3}}{\sqrt{p}}\right)^{\frac{2}{3}}\cdot\left(\frac{\sqrt{p}-\sqrt{q}}{\sqrt{p}}+\frac{\sqrt{q}}{\sqrt{p}-\sqrt{q}}\right)^{-\frac{2}{3}}$
20	$\left(\sqrt{m}-\frac{\sqrt{mn}+n}{\sqrt{m}+\sqrt{n}}\right)^2\cdot\left(\frac{\sqrt{m}}{\sqrt{m}+\sqrt{n}}+\frac{\sqrt{n}}{\sqrt{m}-\sqrt{n}}+\frac{2\sqrt{mn}}{m-n}\right)$
21	$\left(\frac{\sqrt{u}+3\sqrt{v}}{\left(\sqrt{u}-\sqrt{v}\right)^2}+\frac{\sqrt{u}-3\sqrt{v}}{u-v}\right)\cdot\frac{\sqrt{u}-\sqrt{v}}{2}$
22	$\frac{\left(x^2-y^2\right)\left(\sqrt[3]{x}+\sqrt[3]{y}\right)}{\sqrt[3]{x^5}+\sqrt[3]{x^2y^3}-\sqrt[3]{x^3y^2}-\sqrt[3]{y^5}}-\left(\sqrt[3]{xy}+\sqrt[3]{y^2}\right)$
23	$\left(\frac{\sqrt{x^3}+\sqrt{y^3}}{\sqrt{x}+\sqrt{y}}-\sqrt{xy}\right)\cdot\frac{1}{x-y}+\frac{2\sqrt{y}}{\sqrt{x}+\sqrt{y}}$
24	$\frac{\left(a^2-b^2\right)\left(a^2+\sqrt[3]{b^2}+a\sqrt[3]{b}\right)}{a\sqrt[3]{b}+a\sqrt{a}-b\sqrt[3]{b}-\sqrt{ab^2}}:\frac{a^3-b}{a\sqrt[3]{b}-\sqrt[6]{a^3b^2}-\sqrt[3]{b^2}+a\sqrt{a}}$
25	$\frac{\sqrt[4]{x^5}+\sqrt[4]{xy^4}-\sqrt[4]{x^4y}-\sqrt[4]{y^5}}{\sqrt{x}+\sqrt{y}}\left(\sqrt[4]{x}+\sqrt[4]{y}\right)$
26	$\frac{\sqrt[3]{ab}\left(\sqrt[3]{b^2}-\sqrt[3]{a^2}\right)+\sqrt[3]{a^4}-\sqrt[3]{b^4}}{\sqrt[3]{a^4}+\sqrt[3]{a^2b^2}-\sqrt[3]{a^3b}}\cdot\sqrt[3]{a^2}$
27	$\frac{(a-b)^3\left(\sqrt{a}+\sqrt{b}\right)^{-3}+2a\sqrt{a}+b\sqrt{b}}{a\sqrt{a}+b\sqrt{b}}+\frac{3\left(\sqrt{ab}-b\right)}{a-b}$
28	$\frac{\left(\sqrt[4]{m}+\sqrt[4]{n}\right)^2+\left(\sqrt[4]{m}-\sqrt[4]{n}\right)^2}{2(m-n)}:\frac{1}{\sqrt{m^3}-\sqrt{n^3}}-3\sqrt{mn}$

29	$\left(a^{\frac{1}{3}}+b^{\frac{1}{3}}\right):\left(2+\sqrt[3]{\frac{a}{b}}+\sqrt[3]{\frac{b}{a}}\right)-\frac{\sqrt[3]{ab}}{\sqrt[3]{a}+\sqrt[3]{b}}$
30	$\frac{a^{\frac{1}{2}}+ab^{=1}}{a^{-\frac{1}{3}}-a^{-\frac{1}{6}}b^{-\frac{1}{3}}+b^{-\frac{2}{3}}}-\frac{a}{\sqrt[3]{b}}$
31	$\left(\frac{2x^{1/3}}{x^{4/3}-4x^{1/3}}+\frac{1}{x^{1/2}-4x^{-1/2}}\right)^{-2}-x+4$
32	$\frac{\left(\sqrt[5]{a^{4/3}}\right)^{3/2}}{\left(\sqrt[5]{a^4}\right)^3}\cdot\frac{\left(\sqrt{a\sqrt[3]{a^2b}}\right)^4}{\left(\sqrt[4]{a\sqrt{b}}\right)^6}$
33	$\left(\frac{\sqrt[4]{a^3}-1}{\sqrt[4]{a}-1}+\sqrt[4]{a}\right)^{1/2}\cdot\left(\frac{\sqrt[4]{a^3}+1}{\sqrt[4]{a}+1}-\sqrt{a}\right)\cdot\left(a-\sqrt{a^3}\right)^{-1}$
34	$\frac{x-y}{x^{\frac{3}{4}}+x^{\frac{1}{2}}y^{\frac{1}{4}}}\cdot\frac{x^{\frac{1}{2}}y^{\frac{1}{4}}-x^{\frac{1}{4}}y^{\frac{1}{2}}}{x^{\frac{1}{2}}+y^{\frac{1}{2}}}\cdot\frac{x^{\frac{1}{4}}y^{\frac{1}{4}}}{x^{\frac{1}{2}}-2x^{\frac{1}{4}}y^{\frac{1}{4}}+y^{\frac{1}{2}}}$
35	$\frac{x^{\frac{1}{6}}-y^{\frac{1}{6}}}{x^{\frac{1}{2}}+x^{\frac{1}{3}}y^{\frac{1}{6}}}\cdot\frac{\left(x^{\frac{1}{3}}+y^{\frac{1}{3}}\right)^2-4\sqrt[3]{xy}}{x^{\frac{5}{6}}y^{\frac{1}{3}}-x^{\frac{1}{2}}y^{\frac{2}{3}}}+2x^{-\frac{2}{3}}y^{-\frac{1}{6}}$
36	$\frac{\sqrt{a}\left(\sqrt{a}-\sqrt{b}\right)^2}{\sqrt{b}}:\left(\sqrt{\frac{a}{b}}+\sqrt{\frac{b}{a}}-2\right)$
37	$\frac{m^{4/3}-27m^{1/3}n}{m^{2/3}+3\sqrt[3]{mn}+9n^{2/3}}:\left(1-3\sqrt[3]{\frac{n}{m}}\right)-\sqrt[3]{m^2}$

38	$\dfrac{a^3-a-2b-\dfrac{b^2}{a}}{\left(1-\sqrt{\dfrac{1}{a}+\dfrac{b}{a^2}}\right)\cdot\left(a+\sqrt{a+b}\right)}:\left(\dfrac{a^3+a^2+ab+a^2b}{a^2-b^2}+\dfrac{b}{a-b}\right)$
39	$\dfrac{a^{\frac{7}{3}}-2a^{\frac{5}{3}}b^{\frac{2}{3}}+ab^{\frac{4}{3}}}{a^{\frac{5}{3}}-a^{\frac{4}{3}}b^{\frac{1}{3}}-ab^{\frac{2}{3}}+a^{\frac{2}{3}}b}:a^{\frac{1}{3}}$
40	$\sqrt[n]{y^{\frac{2n}{m-n}}}:\sqrt[m]{y^{\frac{(m-n)^2+4mn}{m^2-n^2}}}$
41	$\dfrac{x-1}{x^{3/4}+x^{1/2}}\cdot\dfrac{x^{1/2}+x^{1/4}}{x^{1/2}+1}\cdot x^{1/4}+1$
42	$\dfrac{1-x^{-2}}{x^{1/2}-x^{-1/2}}-\dfrac{2}{x^{3/2}}+\dfrac{x^{-2}-x}{x^{1/2}-x^{-1/2}}$
43	$t\cdot\dfrac{1+\dfrac{2}{\sqrt{t+4}}}{2-\sqrt{t+4}}+\sqrt{t+4}+\dfrac{4}{\sqrt{t+4}}$
44	$\left(\dfrac{\sqrt{a}+\sqrt{b}}{\sqrt{a+b}}-\dfrac{\sqrt{a+b}}{\sqrt{a}+\sqrt{b}}\right)^{-2}-\left(\dfrac{\sqrt{a}-\sqrt{b}}{\sqrt{a+b}}-\dfrac{\sqrt{a+b}}{\sqrt{a}-\sqrt{b}}\right)^{-2}$
45	$\left(\dfrac{1}{\sqrt{a}+\sqrt{a+1}}+\dfrac{1}{\sqrt{a}-\sqrt{a-1}}\right):\left(1+\sqrt{\dfrac{a+1}{a-1}}\right)$
46	$\dfrac{\left(\sqrt[3]{\left(r^2+4\right)\cdot\sqrt{1+\dfrac{4}{r^2}}}-\sqrt[3]{\left(r^2-4\right)\cdot\sqrt{1-\dfrac{4}{r^2}}}\right)^2}{r^2-\sqrt{r^4-16}}$
47	$\dfrac{\left(\sqrt{a^2+a\sqrt{a^2-b^2}}-\sqrt{a^2-a\sqrt{a^2-b^2}}\right)^2}{2\sqrt{a^3b}}:\left(\sqrt{\dfrac{a}{b}}+\sqrt{\dfrac{b}{a}}-2\right)$

48	$\left(\frac{\sqrt{x-a}}{\sqrt{x+a}+\sqrt{x-a}}+\frac{x-a}{\sqrt{x^2-a^2}-x+a}\right):\sqrt{\frac{x^2}{a^2}-1}$
49	$\frac{\sqrt{1-x^2}-1}{x}\cdot\left(\frac{1-x}{\sqrt{1-x^2}+x-1}+\frac{\sqrt{1+x}}{\sqrt{1+x}-\sqrt{1-x}}\right)$
50	$\frac{\sqrt{\frac{1+a}{1-a}}+\sqrt{\frac{1-a}{1+a}}}{\sqrt{\frac{1+a}{1-a}}-\sqrt{\frac{1-a}{1+a}}}-\frac{1}{a}$
51	$\left(\frac{t\sqrt{t+2}}{\sqrt{t-2}}-\frac{2\sqrt{t-2}}{\sqrt{t+2}}-\frac{4t}{\sqrt{t^2-4}}\right)^{1/2}:\sqrt[4]{t^2-4}$
52	$\left(\frac{\sqrt[3]{x+y}}{\sqrt[3]{x-y}}+\frac{\sqrt[3]{x-y}}{\sqrt[3]{x+y}}-2\right):\left(\frac{1}{\sqrt[3]{x-y}}-\frac{1}{\sqrt[3]{x+y}}\right)$
53	$\frac{(x+2a)(x+2b)}{(c+a)(c+b)}+\frac{(x+2b)(x-2c)}{(a-b)(a+c)}+\frac{(x-2c)(x+2a)}{(b+c)(b-a)}$
54	$\frac{\frac{3x+4}{9x^2+12x+16}+\frac{3x-4}{9x^2-12x+16}}{\frac{3x+4}{9x^2+12x+16}-\frac{3x-4}{9x^2-12x+16}}$
55	$\frac{3x^2-6x}{x^2+bx-ax-ab}\cdot\frac{x^2-b^2}{x^2-4}\cdot\frac{x^3-a^2x+2x^2-2a^2}{x^2-bx}$
56	$\left(2-x+4x^2+\frac{5x^2-6x+3}{x-1}\right):\left(2x+1+\frac{2x}{x-1}\right)$
57	$\frac{\sqrt[8]{ab}\,(\sqrt[3]{b^2}-\sqrt[8]{a^2})+\sqrt[8]{a^4}-\sqrt[3]{b^4}}{\sqrt[3]{a^4}+\sqrt[8]{a^2b^2}-\sqrt[3]{a^3b}}\cdot\sqrt[3]{a^2}$

58	$\frac{1}{b\,(abc+a+c)} - \frac{1}{a+\dfrac{1}{b+\dfrac{1}{c}}} : \frac{1}{a+\dfrac{1}{b}}$
59	$\left(\frac{t\,(t+2)}{\sqrt{t-2}} - \frac{2\sqrt{t-2}}{\sqrt{t+2}} - \frac{4t}{\sqrt{t^2-4}}\right)^{\frac{1}{2}} : \sqrt[4]{t^2-4}.$
60	$\frac{\left(\frac{1}{a^m} - a^{\frac{1}{n}}\right)^2 + 4a^{\frac{m+n}{mn}}}{\left(a^{\frac{2}{m}} - a^{\frac{2}{n}}\right)\left(\sqrt[m]{a^{m+1}} + \sqrt[n]{a^{n+1}}\right)}$

Приклад виконання лабораторної роботи

Нехай нам заданий такий алгебраїчний вираз:

$$b + \frac{a^2 - b^2}{a+b} + 5\,.$$

Програма для обчислення виразу матиме такий вигляд:

```
a = input("Введіть a: ")
b = input("Введіть b: ")

a = float(a)
b = float(b)

print()

y = b + (a**2 - b**2) / (a + b) + 5
print("Результат:", y)
```

Спростимо вираз:

$$b + \frac{a^2 - b^2}{a+b} + 5 = b + \frac{(a-b)(a+b)}{a+b} + 5 = b + (a-b) + 5 = a + 5\,.$$

Додамо у програму обчислення та виведення на екран результату спрощеного виразу:

```
a = input("Введіть a: ")
b = input("Введіть b: ")

a = float(a)
b = float(b)

print()

y1 = b + (a**2 - b**2) / (a + b) + 5
print("Результат:", y1)

y2 = a + 5
print("Результат після спрощення:", y2)
```

Результат роботи програми:

```
Введіть a: 25
Введіть b: -10

Результат: 30.0
Результат після спрощення: 30.0
```

Рекомендації до виконання лабораторної роботи

Бажано, щоб у програмі довжина одного рядка не перевищувала 80 символів. Якщо вираз є надто довгим, можна розбити його на кілька частин та обчислювати кожну частину окремою командою. Розглянемо приклад:

$$\left(\frac{a-\sqrt{a^2-b^2}}{a+\sqrt{a^2-b^2}} - \frac{a+\sqrt{a^2-b^2}}{a-\sqrt{a^2-b^2}} \right) : \frac{4\sqrt{a^4-a^2b^2}}{(5b)^2} .$$

Обчислення цього виразу краще виконати за окремими частинами, наприклад, так:

```
a = input("Введіть a: ")
b = input("Введіть b: ")

a = float(a)
b = float(b)

x1 = (a-(a*a-b*b)**0.5)/(a+(a*a-b*b)**0.5)
x2 = (a+(a*a-b*b)**0.5)/(a-(a*a-b*b)**0.5)
x3 = x1 - x2
x4 = 4*(a**4-a*a*b*b)**0.5/((5*b)**2)
x5 = x3 / x4

print("Результат:", x5)
```

Можна було б записати увесь вираз в один рядок, однак у цьому випадку розуміння програми було б значно ускладненим:

```
a = input("Введіть a: ")
b = input("Введіть b: ")

a = float(a)
b = float(b)

x =((a-(a*a-b*b)**0.5)/(a+(a*a-b*b)**0.5)-(a+(a*a-b*b)**0.5)/(a-
(a*a-b*b)**0.5))/(4*(a**4-a*a*b*b)**0.5/((5*b)**2))

print("Результат:", x)
```

Зміст звіту з лабораторної роботи

1. Титульний аркуш.
2. Індивідуальний варіант завдання.
3. Спрощення виразу.
4. Лістинг програми для обчислення спрощеного та неспрощеного виразів.
5. Результати роботи програми (3–4 приклади).
6. Висновки до лабораторної роботи.

Лабораторна робота № 3

Класи у мові програмування Python

Метою лабораторної роботи є отримання студентами навичок програмного опису об’єктів навколишнього світу за допомогою класів.

Варіант завдання вибирається з табл. 3.1, номер рядка таблиці відповідає номеру *N* у загальному списку студентів групи (не в підгрупі). Якщо номер студента перевищує кількість рядків таблиці, рух по таблиці починається знову з першого рядка. Наприклад, якщо в таблиці 30 рядків, а у студента номер варіанта дорівнює 55, він обирає 55 – 30 = 25 номер варіанта.

Варіантом завдання є об’єкт із навколишнього світу, що вказаний у відповідному рядку таблиці. Для заданого варіанта завдання необхідно розробити програму мовою Python, у якій виконати таке:

1. Розробити п’ять класів, які описують типові елементи заданого об’єкта. У кожному класі має бути не менш ніж 10 атрибутів, тип і початкові значення яких студент обирає на свій розсуд. Тип атрибута може бути символьним, цілим або дійсним.

2. Для кожного з класів створити три екземпляри. Усі атрибути кожного екземпляра ініціалізувати значеннями, що характеризують реальних представників цього класу та відрізняються від значень, вказаних під час опису класу.

3. Вивести на екран у зручній формі усю інформацію, що міститься у створених екземплярах класів. Зробити висновок про те, що мають спільного та чим відрізняються окремі екземпляри одного класу.

4. Підготувати та захистити звіт з лабораторної роботи.

Вміст звіту з лабораторної роботи

1. Титульний аркуш, оформлений за стандартом університету.
2. Індивідуальний варіант завдання.
3. Лістинг програми з докладними коментарями.
4. Знімки екрана з результатами роботи програми.
5. Висновки до лабораторної роботи.

Таблиця 3.1 – Варіанти завдань до лабораторної роботи № 3

Приклад виконання лабораторної роботи (для одного класу)

Створимо набір класів для об’єкта «Автовокзал».

Для автовокзалу можна визначити, наприклад, такі класи:

– транспортний засіб;

– маршрут;

– квиток;

– водій;

– розклад.

Опишемо клас «транспортний засіб»:

```
class Transport:       # Транспортний засіб
    type = "автобус"  # Тип (автобус, мікроавтобус тощо)
    marka = "ПАЗ"      # Марка
    num = 0            # Номер
    places = 30        # Кількість місць для сидіння
    cost = 10000       # Ціна в доларах США
    age = 12           # Вік (кількість років після виготовлення)
    speed = 70         # Середня швидкість
    mileage = 3000     # Пробіг, км
    doors = 2          # Кількість дверей
    TO = 2019          # Рік останнього технічного огляду
```

Створимо для цього класу три екземпляри:

```
a1 = Transport()      # Перший екземпляр класу Transport
a2 = Transport()      # Другий екземпляр класу Transport
a3 = Transport()      # Третій екземпляр класу Transport
```

Ініціалізуємо усі атрибути кожного екземпляра початковими значеннями:

```
a1.type = "мікроавтобус"
a1.marka = "Mersedes"
a1.num = "ВА 5476 ЕК"
a1.places, a1.cost, a1.age, a1.mileage, a1.doors = 12, 12000, 5, 7000, 1
a1.TO = 2018

a2.type = "тролейбус"
a2.marka = "Богдан"
a2.num = "253"
a2.places, a2.cost, a2.age, a2.mileage, a2.doors = 28, 30000, 2, 100000, 3
a2.TO = 2020

a3.type = "автобус"
a3.marka = "Iveco"
a3.num = "ВН 5476"
a3.places, a3.cost, a3.age, a3.mileage, a3.doors = 32, 18000, 10, 290300, 2
a3.TO = 2019
```

Виведемо на екран цю інформацію:

```
print("Екземпляр a1 класу Transport:")
print(" ", a1.type, "марки", a1.marka, end="")
print(" на", a1.places, "посадкових місць,")
print("  номер реєстрації: ", a1.num, ",", sep="")
print("  вартість:", a1.cost, "у.о.,", a1.age, end="")
print(" років, пробіг", a1.mileage, "км,")
print("  кількість дверей:", a1.doors, end="")
print(", рік проходження ТО: ", a1.TO, ".", sep="")
print()

print("Екземпляр a2 класу Transport:")
print(" ", a2.type, "марки", a2.marka, end="")
print(" на", a2.places, "посадкових місць,")
print("  номер реєстрації: ", a2.num, ",", sep="")
```

```
print("  вартість:", a2.cost, "у.о.,", a2.age, end="")
print(" років, пробіг", a2.mileage, "км,")
print("  кількість дверей:", a2.doors, end="")
print(", рік проходження ТО: ", a2.TO, ".", sep="")
print()

print("Екземпляр a3 класу Transport:")
print(" ", a3.type, "марки", a3.marka, end="")
print(" на", a3.places, "посадкових місць,")
print("  номер реєстрації: ", a3.num, ",", sep="")
print("  вартість:", a3.cost, "у.о.,", a3.age, end="")
print(" років, пробіг", a3.mileage, "км,")
print("  кількість дверей:", a3.doors, end="")
print(", рік проходження ТО: ", a3.TO, ".", sep="")
```

Результат роботи програми:

```
Екземпляр a1 класу Transport:
  мікроавтобус марки Mersedes на 12 посадкових місць,
  номер реєстрації: ВА 5476 ЕК,
  вартість: 12000 у.о., 5 років, пробіг 7000 км,
  кількість дверей: 1, рік проходження ТО: 2018.

Екземпляр a2 класу Transport:
  тролейбус марки Богдан на 28 посадкових місць,
  номер реєстрації: 253,
  вартість: 30000 у.о., 2 років, пробіг 100000 км,
  кількість дверей: 3, рік проходження ТО: 2020.

Екземпляр a3 класу Transport:
  автобус марки Iveco на 32 посадкових місць,
  номер реєстрації: ВН 5476,
  вартість: 18000 у.о., 10 років, пробіг 290300 км,
  кількість дверей: 2, рік проходження ТО: 2019.
```

Лабораторна робота № 4

Організація розгалужень

Метою лабораторної роботи є отримання студентами навичок написання програм з розгалуженнями.

Під час лабораторної роботи студент повинен виконати три завдання, кожне з яких оцінюється від 0 до 3 балів залежно від якості виконання (правильності алгоритму, наявності помилок тощо). Додатковий десятий бал ставиться у випадку, коли усі три завдання виконані правильно, і в програмах виведення результату зроблені охайно і у зручному форматі.

Завдання 1

Задані два цілі числа *A* і *B*. Проаналізувати числа згідно з варіантом завдання з таблиці 4.1. Номер рядка таблиці відповідає номеру *№* у загальному списку студентів групи (не в підгрупі). Якщо номер студента більший за кількість рядків таблиці, рух по таблиці починається з першого рядка. Наприклад, якщо в таблиці 14 рядків, а у студента номер варіанта дорівнює 55, він обирає 55 – 14 – 14 – 14 = 13 номер варіанта з таблиці 4.1.

Під час написання програми дозволяється використовувати будь-яку кількість додаткових змінних. Значення змінних *A* і *B* змінювати не можна. Вбудовані функції мови Python, як-от *max*, *min* тощо, дозволяється використовувати лише в тому випадку, якщо їх не можна замінити операторами *if*.

Таблиця 4.1 – Варіанти для завдання 1

Завдання 2

Задана послідовність із п’яти цілих чисел *x1*, ..., *x5*. Проаналізувати дану послідовність згідно з варіантом завдання із таблиці 4.2. Номер рядка таблиці відповідає номеру *№* у загальному списку студентів групи (не в підгрупі). Якщо номер студента більший за кількість рядків таблиці, рух по таблиці починається з першого рядка. Наприклад, якщо в таблиці 15 рядків, а у студента номер варіанта дорівнює 55, він обирає 55 – 15 – 15 – 15 = 10 номер варіанта з таблиці 4.2.

Під час написання програми дозволяється використовувати будь-яку кількість додаткових змінних. Значення змінних *x1*, ..., *x5* змінювати не можна.

Таблиця 4.2 – Варіанти для завдання 2

Завдання 3

П’ять змінних *a1*, *a2*, *a3*, *a4*, *a5* задають послідовність цифр числа *A* в десятковій системі числення (наприклад, *a1*=3, *a2*=0, *a3*=8, *a4*=5, *a5*=2 відповідають числу 30 852). Змінні *b1*, *b2*, *b3*, *b4*, *b5* задають послідовність цифр числа *B*. Необхідно проаналізувати дані числа відповідно до варіанта завдання із таблиці 4.3.

Номер рядка таблиці 4.3 відповідає номеру *№* у загальному списку студентів групи (не в підгрупі). Якщо номер студента більший за кількість рядків таблиці, рух по таблиці починається з першого рядка. Наприклад, якщо в таблиці 16 рядків, а у студента номер варіанта дорівнює 55, він обирає 55 – 16 – 16 – 16 = 7 – номер варіанта з таблиці 4.3.

Під час написання програми дозволяється використовувати будь-яку кількість додаткових змінних. Значення змінних *a1–a5* та *b1–b5* змінювати не можна.

Таблиця 4.3 – Варіанти для завдання 3

Вміст звіту з лабораторної роботи

1. Титульний аркуш, оформлений за стандартом університету.
2. Завдання 1 (варіант завдання, блок-схема алгоритму, лістинг програми, результати роботи).
3. Завдання 2 (варіант завдання, блок-схема алгоритму, лістинг програми, результати роботи).
4. Завдання 3 (варіант завдання, блок-схема алгоритму, лістинг програми, результати роботи).
5. Висновки до лабораторної роботи.

Приклад виконання завдання 1

Враховуючи відносно низьку складність цього завдання, студентам треба орієнтуватись на приклади, наведені у лекціях.

Приклад виконання завдання 2

Для заданої послідовності чисел *x1*, ..., *x5* визначити кількість ненульових чисел, які з обох боків обмежені нулями.

Нехай усі п’ять чисел вводяться з клавіатури:

```
x1 = input("Введіть x1: ")
x2 = input("Введіть x2: ")
x3 = input("Введіть x3: ")
x4 = input("Введіть x4: ")
x5 = input("Введіть x5: ")

x1 = int(x1)
x2 = int(x2)
x3 = int(x3)
x4 = int(x4)
x5 = int(x5)
```

Для послідовності з п’яти чисел можливі такі ситуації, коли число обмежене нулями:

x1	0			
0	*x2*	0		
	0	*x3*	0	
		0	*x4*	0
			0	*x5*

Перший і останній випадки відповідають крайнім елементам послідовності, які можуть обмежуватись нулем тільки з одного боку. Інакше кажучи, в завданні було сказано знайти кількість чисел, обмежених *нулями*, а не *нулем*. Для деяких задач це може бути принциповим, і випадки з обмеженням числа з одного боку можуть порушити статистику і привести до поганих наслідків.

Але ми все ж розглянемо випадки для крайніх чисел. Отже, у нас можливі п’ять ситуацій, коли якесь число, що не дорівнює нулю, обмежене нулями. Для вирішення задачі ми повинні послідовно розглянути кожну із ситуацій та підрахувати кількість ситуацій, що відповідають умові.

Для підрахунку кількості ситуацій будемо використовувати окрему змінну. Через її призначення таку змінну зазвичай називають лічильником. Дамо цій змінній ім’я *k* і початкове значення нуль:

```
k = 0
```

Початкове значення дорівнює нулю тому, що на початку роботи програми ми ще не знаємо, чи знайдемо хоча б одну ситуацію, коли число обмежене нулями. Наприклад, якщо усі змінні більші за одиницю, жодної такої ситуації не буде. У цьому випадку змінна *k* залишиться рівною нулю, що є правильним результатом роботи програми.

Тепер перейдемо до перевірки кожної із ситуацій, розглянутих на рисунку вище.

Аналіз сусідів змінної x1

Змінна *x1* обмежена нульовими значеннями у випадку, якщо вона не дорівнює нулю, і водночас *x2*=0. Якщо обидві умови виконуються, ми

знайшли чергову ситуацію, яку ми шукали, і можемо збільшити лічильник *k* на одиницю:

```
if x1!=0 and x2==0:
    k = k + 1
```

Аналіз сусідів змінної x2

Змінна *x2* обмежена нульовими значеннями у випадку, якщо вона не дорівнює нулю, і водночас *x1*=0 та *x3*=0:

```
if x2!=0 and x1==0 and x3==0:
    k = k + 1
```

Аналіз сусідів змінної x3

Змінна *x3* обмежена нульовими значеннями у випадку, якщо вона не дорівнює нулю, і водночас *x2*=0 та *x4*=0:

```
if x3!=0 and x2==0 and x4==0:
    k = k + 1
```

Аналіз сусідів змінної x4

Змінна *x4* обмежена нульовими значеннями у випадку, якщо вона не дорівнює нулю, і водночас *x3*=0 та *x5*=0:

```
if x4!=0 and x3==0 and x5==0:
    k = k + 1
```

Аналіз сусідів змінної x5

Змінна *x5* обмежена нульовими значеннями у випадку, якщо вона не дорівнює нулю, і водночас *x4*=0:

```
if x5!=0 and x4==0:
    k = k + 1
```

Після того, як програма послідовно виконала усі п’ять операторів *if*, змінна *k* буде містити результат роботи програми, тобто кількість ненульових змінних заданої послідовності, що обмежені з обох боків нулями. Нам залишилось вивести значення змінної *k* на екран.

Отже, увесь лістинг роботи програми має такий вигляд:

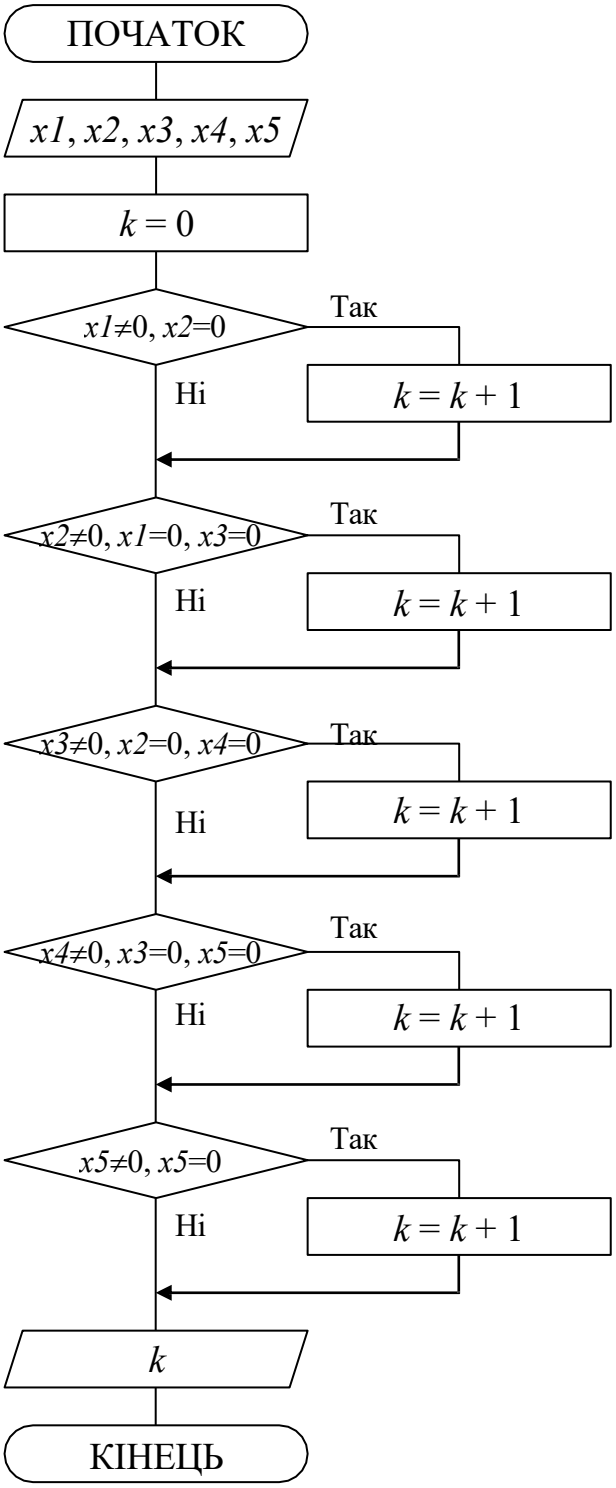
```
x1 = input("Введіть x1: ")
x2 = input("Введіть x2: ")
x3 = input("Введіть x3: ")
x4 = input("Введіть x4: ")
x5 = input("Введіть x5: ")

x1 = int(x1)
x2 = int(x2)
x3 = int(x3)
x4 = int(x4)
x5 = int(x5)

k = 0                                # Обнулення лічильника

if x1!=0 and x2==0:                  # Аналіз змінної x1
    k = k + 1

if x2!=0 and x1==0 and x3==0:        # Аналіз змінної x2
    k = k + 1

if x3!=0 and x2==0 and x4==0:        # Аналіз змінної x3
    k = k + 1

if x4!=0 and x3==0 and x5==0:        # Аналіз змінної x4
    k = k + 1

if x5!=0 and x4==0:                  # Аналіз змінної x5
    k = k + 1

print("Кількість змінних =", k) # Результат
```

Приклад результату роботи програми:

```
Введіть x1: 0
Введіть x2: 0
Введіть x3: 2
Введіть x4: 0
Введіть x5: 5
Кількість змінних = 2
```

Блок-схема алгоритму програми доволі проста, хоча й трохи довга (дивись рисунок нижче).

Розглянутий спосіб вирішення задачі – не єдиний можливий. Студентам пропонується самостійно пошукати інші алгоритми та порівняти результати їх роботи.



Приклад виконання завдання 3

Для чисел *A* і *B*, заданих змінними *a1–a5* та *b1–b5*, визначити, чи можна із цифр числа *B* скласти число *A*.

Задамо цифри чисел константами. Можна було б вводити їх з клавіатури, але наразі це не є принциповим.

```
a1, a2, a3, a4, a5 = 3, 1, 5, 2, 4
b1, b2, b3, b4, b5 = 5, 3, 4, 1, 2
```

Найбільш простим рішенням поставленої задачі є розгляд усіх можливих варіантів розташування цифр числа *B* і порівняння їх із відповідними цифрами числа *A*. Для п'ятирозрядного числа існують такі комбінації цифр:

*b*1 *b*2 *b*3 *b*4 *b*5
*b*1 *b*2 *b*3 *b*5 *b*4
*b*1 *b*2 *b*4 *b*3 *b*5
*b*1 *b*2 *b*4 *b*5 *b*3

і так далі. Водночас треба розуміти, що кількість перестановок без повторів (пертурбацій) дорівнює *n*!, тобто в нашому випадку 5! = 120. Нескладно зрозуміти, що аналіз усіх 120 комбінацій буде ускладнений через необхідність побудови усіх можливих комбінацій та написання складних операторів *if* для кожної з комбінацій. Код програми буде мати приблизно такий вигляд:

```
a1, a2, a3, a4, a5 = 3, 1, 5, 2, 4
b1, b2, b3, b4, b5 = 1, 2, 3, 4, 5

k = 0  # Ознака – чи можливе вирішення задачі (1 – так, 0 – ні)

if a1==b1 and a2==b2 and a3==b3 and a4==b4 and a5==b5:
    k = 1

if a1==b1 and a2==b2 and a3==b3 and a4==b5 and a5==b4:
    k = 1

if a1==b1 and a2==b2 and a3==b4 and a4==b3 and a5==b5:
    k = 1

if a1==b1 and a2==b2 and a3==b4 and a4==b5 and a5==b3:
    k = 1

...
...
...

if k==1:
    print("Можливо.")
else:
    print("Неможливо.")
```

У цій програмі ми використовуємо спеціальну змінну *k*, яка буде містити результат роботи програми. Спочатку *k* = 0, тобто ми припускаємо, що рішення не буде знайдене. Якщо хоча б один оператор *if* «спрацює», змінна *k* стане рівною 1 і залишатиметься такою до кінця програми. У кінці програми ми аналізуємо значення змінної *k* і виводимо результат на екран.

Розглянуте рішення є простим, але вимагає величезних витрат часу на написання програми (і до речі, на її роботу). А якщо уявити, що числа *A* і *B* мають по 10 розрядів кожне, написання програми стає практично неможливим (10! = 3 628 800).

Тому бажано знайти якийсь інший алгоритм розв'язання задачі. Таких алгоритмів існує декілька. До того ж навряд чи математики і програмісти колись ставили задачу знайти усі можливі алгоритми саме для цієї задачі.

Розглянемо розв'язання цієї задачі за допомогою розташування цифр числа в порядку їх збільшення. Нехай *A* = 43512, *B* = 15243. Оскільки ми маємо можливість працювати з окремими цифрами чисел, розташуємо значення цифр кожного числа в порядку їх збільшення:

A = 12345,

B = 12345.

Тепер, порівнявши числа, ми бачимо, що вони однакові. Це означає, що обидва числа складаються з одного й того ж набору цифр. Як наслідок, запис числа *A* за допомогою цифр числа *B* можливий.

Якщо ж числа *A* і *B* після впорядкування розрядів будуть відрізнятись, це означатиме, що відрізняються і їх набори цифр. Тобто запис числа *A* за допомогою цифр числа *B* неможливий.

Отже, нам залишилось реалізувати алгоритм розташування цифр кожного із чисел у порядку збільшення. Тут також існують різні способи, але ми розглянемо лише один:

1. Серед п'яти чисел *a1–a5* знайдемо найменше і обміняємо його значення з першим числом *a1*:

```
if a2<a1 and a2<a3 and a2<a4 and a2<a5: # Перевіряємо число a2
    a1, a2 = a2, a1                     # Обмін чисел місцями

if a3<a1 and a3<a2 and a3<a4 and a3<a5: # Перевіряємо число a3
    a1, a3 = a3, a1                     # Обмін чисел місцями

if a4<a1 and a4<a2 and a4<a3 and a4<a5: # Перевіряємо число a4
    a1, a4 = a4, a1                     # Обмін чисел місцями

if a5<a1 and a5<a2 and a5<a3 and a5<a3: # Перевіряємо число a5
    a1, a5 = a5, a1                     # Обмін чисел місцями
```

Внаслідок виконання цих команд у змінній *a1* буде записана найменша цифра числа *A*, а початкове значення змінної *a1* буде перенесене у змінну, що містила найменшу цифру. Наприклад, якщо *a1*=5, *a2*=3, *a3*=2, *a4*=1, *a5*=4, після виконання цих команд матимемо *a1*=1, *a4*=5. Змінні *a2*, *a3*, *a5* залишаться незмінними.

2. Тепер, коли найменша цифра числа A у нас визначена і поставлена на перше місце, будемо працювати лише зі змінними *a2–a5*. Серед них знайдемо найменшу і обміняємо її значення зі змінною *a2*:

```
if a3<a2 and a3<a4 and a3<a5:              # Перевіряємо число a3
    a2, a3 = a3, a2

if a4<a2 and a4<a3 and a4<a5:              # Перевіряємо число a4
    a2, a4 = a4, a2

if a5<a2 and a5<a3 and a5<a4:              # Перевіряємо число a5
    a2, a5 = a5, a2
```

Після виконання цих трьох команд у змінній *a2* буде знаходитись найменше значення серед чисел *a2–a5*. Оскільки найменша цифра числа *A* раніше вже була покладена нами у змінну *a1*, у змінній *a2* ми отримали другу найменшу цифру числа *A*.

3. Серед змінних *a3–a5* знайдемо найменшу і обміняємо її значення зі змінною *a3*:

```
if a4<a3 and a4<a5:                        # Перевіряємо число a4
    a3, a4 = a4, a3

if a5<a3 and a5<a3:                        # Перевіряємо число a5
    a3, a5 = a5, a3
```

Можна помітити, що ми в цих командах намагаємось поставити на місце змінної *a3* змінну *a4* або змінну *a5*. Якщо ці змінні будуть більші, ніж *a3*, її значення виявиться мінімальним серед них, і тому залишиться незмінним.

4. Тепер, коли перші три цифри числа *A* впорядковані, нам залишилося впорядкувати змінні *a4* та *a5*. Це робиться такою командою:

```
if a5<a4:
    a4, a5 = a5, a4
```

Після виконання усіх операторів *if* значення у змінних *a1–a5* будуть розташовані за збільшенням. Переконатись у цьому можна, вивівши їх на екран:

```
print(a1, a2, a3, a4, a5)
```

Для чисел *a1*, *a2*, *a3*, *a4*, *a5* = 3, 1, 5, 2, 4 буде отриманий такий результат:

```
1 2 3 4 5
```

Отже, нам вдалося впорядкувати розряди числа *A* в порядку збільшення значень. Для впорядкування розрядів числа *B* можна використати той самий код, замінивши імена змінних *a1–a5* на відповідні імена *b1–b5*:

```
b1, b2, b3, b4, b5 = 5, 3, 4, 1, 2

if b2<b1 and b2<b3 and b2<b4 and b2<b5: # Перевіряємо число b2
    b1, b2 = b2, b1                    # Обмін чисел місцями

if b3<b1 and b3<b2 and b3<b4 and b3<b5: # Перевіряємо число b3
    b1, b3 = b3, b1                    # Обмін чисел місцями

if b4<b1 and b4<b2 and b4<b3 and b4<b5: # Перевіряємо число b4
    b1, b4 = b4, b1                    # Обмін чисел місцями

if b5<b1 and b5<b2 and b5<b3 and b5<b3: # Перевіряємо число b5
    b1, b5 = b5, b1                    # Обмін чисел місцями

if b3<b2 and b3<b4 and b3<b5:          # Перевіряємо число b3
    b2, b3 = b3, b2

if b4<b2 and b4<b3 and b4<b5:          # Перевіряємо число b4
    b2, b4 = b4, b2

if b5<b2 and b5<b3 and b5<b4:          # Перевіряємо число b5
    b2, b5 = b5, b2

if b4<b3 and b4<b5:                    # Перевіряємо число b4
    b3, b4 = b4, b3

if b5<b3 and b5<b3:                    # Перевіряємо число b5
    b3, b5 = b5, b3

if b5<b4:
    b4, b5 = b5, b4

print(b1, b2, b3, b4, b5)
```

Тепер ми маємо дві групи чисел – *a1–a5* та *b1–b5*, впорядкованих за зростанням. Нам залишилось порівняти їх між собою. Це можна зробити, порівнявши між собою відповідні розряди:

```
if a1==b1 and a2==b2 and a3==b3 and a4==b4 and a5==b5:
    print("Можливо.")
else:
    print("Неможливо.")
```

Можна зробити інакше: побудувати зі значень розрядів числа *A* і *B* та порівняти їх між собою:

```
A = a1*10000 + a2*1000 + a3*100 + a4*10 + a5
B = b1*10000 + b2*1000 + b3*100 + b4*10 + b5

if A==B:
    print("Можливо.")
else:
    print("Неможливо.")
```

Увесь лістинг програми наведений нижче. Студентам пропонується самостійно перевірити дієвість цієї програми для різних випадків, а також побудувати блок-схему алгоритму.

```
a1, a2, a3, a4, a5 = 3, 1, 5, 2, 4
b1, b2, b3, b4, b5 = 5, 3, 4, 1, 2

if a2<a1 and a2<a3 and a2<a4 and a2<a5: # Перевіряємо число a2
    a1, a2 = a2, a1                    # Обмін чисел місцями

if a3<a1 and a3<a2 and a3<a4 and a3<a5: # Перевіряємо число a3
    a1, a3 = a3, a1                    # Обмін чисел місцями

if a4<a1 and a4<a2 and a4<a3 and a4<a5: # Перевіряємо число a4
    a1, a4 = a4, a1                    # Обмін чисел місцями

if a5<a1 and a5<a2 and a5<a3 and a5<a3: # Перевіряємо число a5
    a1, a5 = a5, a1                    # Обмін чисел місцями

if a3<a2 and a3<a4 and a3<a5:           # Перевіряємо число a3
    a2, a3 = a3, a2

if a4<a2 and a4<a3 and a4<a5:           # Перевіряємо число a4
    a2, a4 = a4, a2

if a5<a2 and a5<a3 and a5<a4:           # Перевіряємо число a5
    a2, a5 = a5, a2

if a4<a3 and a4<a5:                     # Перевіряємо число a4
    a3, a4 = a4, a3
```

```
if a5<a3 and a5<a3:                          # Перевіряємо число a5
    a3, a5 = a5, a3

if a5<a4:
    a4, a5 = a5, a4


if b2<b1 and b2<b3 and b2<b4 and b2<b5: # Перевіряємо число b2
    b1, b2 = b2, b1                          # Обмін чисел місцями

if b3<b1 and b3<b2 and b3<b4 and b3<b5: # Перевіряємо число b3
    b1, b3 = b3, b1                          # Обмін чисел місцями

if b4<b1 and b4<b2 and b4<b3 and b4<b5: # Перевіряємо число b4
    b1, b4 = b4, b1                          # Обмін чисел місцями

if b5<b1 and b5<b2 and b5<b3 and b5<b3: # Перевіряємо число b5
    b1, b5 = b5, b1                          # Обмін чисел місцями

if b3<b2 and b3<b4 and b3<b5:            # Перевіряємо число b3
    b2, b3 = b3, b2

if b4<b2 and b4<b3 and b4<b5:            # Перевіряємо число b4
    b2, b4 = b4, b2

if b5<b2 and b5<b3 and b5<b4:            # Перевіряємо число b5
    b2, b5 = b5, b2

if b4<b3 and b4<b5:                          # Перевіряємо число b4
    b3, b4 = b4, b3

if b5<b3 and b5<b3:                          # Перевіряємо число b5
    b3, b5 = b5, b3

if b5<b4:
    b4, b5 = b5, b4

A = a1*10000 + a2*1000 + a3*100 + a4*10 + a5
B = b1*10000 + b2*1000 + b3*100 + b4*10 + b5

if A==B:
    print("Можливо.")
else:
    print("Неможливо.")
```

Лабораторна робота № 5

Організація циклів у мові Python

Метою цієї лабораторної роботи є одержання студентами практичних навичок з організації циклів у програмах мовою Python.

Завдання до лабораторної роботи

Під час лабораторної роботи студент повинен виконати два завдання, кожне з яких оцінюється від 0 до 5 балів залежно від якості виконання (правильності алгоритму, наявності помилок тощо).

Завдання 1

Задані два дійсні числа *A* і *B*, A < B. Написати програму мовою Python, яка для заданих чисел вирішує завдання, наведене в таблиці 5.1. Номер варіанта таблиці відповідає номеру студента в журналі групи (не підгрупи). Якщо номер студента в журналі групи більший за номер варіанта в таблиці, студент повинен вчинити так, як у попередніх лабораторних роботах. Завдання повинні вирішуватись за допомогою циклів *while*.

Таблиця 5.1 – Варіанти для завдання 1

Завдання 2

Натуральне тризначне число *A* записане у десятковій системі числення у вигляді окремих розрядів *a1*, *a2*, *a3*. Написати програму мовою Python, яка вирішує завдання, наведене у таблиці 5.2. Номер варіанта таблиці відповідає номеру студента в журналі групи (не підгрупи). Якщо номер студента в журналі групи більший за номер варіанта в таблиці, студент повинен вчинити так, як у попередніх лабораторних роботах. Задачі повинні розв’язуватись за допомогою кількох вкладених циклів *while*.

Таблиця 5.2 – Варіанти для завдання 2

Вміст звіту з лабораторної роботи

1. Титульний аркуш.
2. Індивідуальний варіант завдання.
3. Блок-схема алгоритму.
4. Лістинг програми з докладними коментарями.
5. Приклад роботи програми (знімки екрана).
6. Висновки до лабораторної роботи.

Приклад виконання завдання 1

Знайдемо суму цілих чисел у діапазоні (*A*; *B*].

Для розв'язання задачі треба побудувати такий цикл, у якому змінна циклу (позначимо її іменем *x*) буде змінюватись від початкового значення *x1* до кінцевого значення *x2* з якимось кроком *dx*. Для цього спочатку треба визначити, чому будуть дорівнювати *x1*, *x2* і *dx*.

Насамперед визначимо крок *dx*, із яким будемо перебирати числа в заданому діапазоні. Оскільки нам треба працювати з цілими числами, *dx* буде дорівнювати одиниці, оскільки відстань між двома цілими числами дорівнює 1.

Тепер визначимо початкове значення *x1*. Можна зазначити, що:

– *x1* має бути цілим числом, бо ми перебираємо лише цілі числа;

– *x1* має бути строго більшим за *A*.

Отже, нам треба знайти найближче ціле число, більше за *A*. Число *A* може бути як цілим, так і дробовим. Якщо *A* – ціле (наприклад, –12), значення *x1* буде на одиницю більшим, оскільки число *A* не входить у діапазон пошуку. Якщо число A – дробове, треба відкинути від нього дробову частину і додати одиницю.

Наприклад, *A* = –5,7. Відкидаємо дробову частку: отримуємо –5. Додаємо одиницю: отримуємо –4. Тобто *x1* = –4.

Якщо *A* = +8,3, відкидання дробової частини дасть нам +8, а додавання одиниці дасть нам шукане значення *x1* = +9.

Отже, у нас є два випадки:

1) якщо *A* – дробове число, ми відкидаємо дробову частину і додаємо одиницю;
2) якщо *A* – ціле число, ми тільки додаємо одиницю.

Звернемо увагу, що ці два випадки можна об'єднати. Якщо придивитись уважно, другий випадок насправді «входить» у перший. Якщо *A* = –12, його дробова частина дорівнює нулю, і її відкидання знов дасть нам число –12. Якщо тепер додати до нього одиницю, ми отримаємо те, що шукали: *x1* = – 11.

Отже, для будь-якого значення числа *A* значення першого числа, з якого будемо починати цикл, визначається шляхом відкидання від *A* дробової частини і додавання одиниці.

Для відкидання дробової частини можна скористатись функцією *trunc*, яка міститься у стандартному модулі *math* і розглядалась у лекції № 5. Програма, що обчислює значення *x1*, виглядає так:

```
import math

A, B = -12.35, 20.74            # Заданий діапазон

x1 = math.trunc(A) + 1          # Відкидання дробової частини + 1
print(x1)
```

Результат:

```
-11
```

Кінцеве значення змінної циклу *x2*, яке повинне бути оброблене, має відповідати таким критеріям:

– бути цілим числом, бо ми в нашій задачі працюємо лише з цілими числами;

– бути меншим або рівним *B*.

Округлення числа до найбільшого меншого цілого виконує функція *floor* із модуля math. Ось кілька прикладів її роботи:

```
print(math.floor(-5.4))    # Має бути -6
print(math.floor(-5))      # Має бути -5
print(math.floor(0))       # Має бути 0
print(math.floor(7.2))     # Має бути 7
print(math.floor(7))       # Має бути 7
```

Результат:

```
-6
-5
0
7
7
```

Отже, фрагмент програми, в якому визначаються значення змінних *x1*, *x2*, *dx* матиме такий вигляд:

```
import math

A, B = -12.35, 5.1         # Заданий діапазон

x1 = math.trunc(A) + 1     # Визначення початкового значення циклу
x2 = math.floor(B)         # Визначення кінцевого значення циклу
dx = 1                     # Визначення кроку циклу
```

Тепер нам ніщо не заважає організувати цикл за допомогою оператора *while*. Оскільки нашою задачею є підрахунок суми усіх цілих чисел у заданому діапазоні, перед початком циклу треба створити змінну *S*, рівну нулю. У цій змінній буде накопичуватися сума чисел.

Програма матиме такий вигляд:

```
import math              # Підключення модуля math

A, B = 0, 10             # Діапазон пошуку

x1 = math.trunc(A) + 1 # Початкове значення змінної циклу
x2 = math.floor(B        # Кінцеве значення змінної циклу
dx = 1                   # Крок зміни змінної циклу

S = 0                    # Початкове значення суми

x = x1                 # Кладемо в змінну циклу початкове значення
while x <= x2:           # Цикл, поки не досягнемо x2
    S = S + x            # Збільшуємо суму
    x = x + dx           # Наступне значення змінної циклу

print("Сума цілих чисел в діапазоні (", A, ", ", B, "] =", S)
```

Результат:

```
Сума цілих чисел в діапазоні ( 0 ,  10 ] = 55
```

Отже, програма працює правильно.

Приклад виконання завдання 2

Серед усіх чисел діапазону [0; *A*] знайти найменше і найбільше числа, які одночасно є парними, є квадратом цілого числа та дільником числа *A*.

Спочатку напишемо загальний «каркас» програми. У цьому каркасі будуть перебиратись усі числа від нуля до *A*, причому перебиратись будуть за окремими цифрами. Розглянемо структуру «каркасу» докладно.

```
a1, a2, a3 = 9, 8, 5      # Задані розряди числа A

A = a1*100 + a2*10 + a3   # Саме число A

i1 = 0                    # Початкове значення змінної циклу по сотнях
while i1 <= 9:            # Цикл по тисячах
    i2 = 0                # Початкове значення змінної циклу по десятках
    while i2 <= 9:        # Цикл по сотнях
        i3 = 0            # Початкове значення змінної циклу по одиницях
        while i3 <= 9:    # Цикл по десятках
            print(i1, i2, i3)  # Виводимо поточне число на екран

            i3 = i3 + 1   # Збільшення змінної циклу по одиницях
        i2 = i2 + 1       # Збільшення змінної циклу по десятках
    i1 = i1 + 1           # Збільшення змінної циклу по сотнях
```

У першому рядку задаються початкові дані програми – значення розрядів числа *A*. У другому рядку на основі своїх розрядів обчислюється саме число *A*.

Програма буде складатись із трьох вкладених циклів. Зовнішній цикл перебиратиме другий (старший) розряд числа, який відповідає сотням. Середній цикл перебиратиме перший розряд числа, який відповідає десяткам.

Внутрішній цикл перебирає нульовий (молодший) розряд числа, який відповідає одиницям. Оскільки число *A* записане в десятковій системі числення, кожен розряд перебиратиметься в діапазоні від 0 до 9.

У кожного циклу буде своя змінна циклу. Нехай у зовнішнього циклу буде змінна *i1*, у середнього – змінна *i2*, у внутрішнього – змінна *i3*. Хоча такі імена і не відповідають правильним номерам розрядів числа, але відповідають іменам змінних *a1–a3*.

У третьому рядку програми іде підготовка до початку зовнішнього циклу програми – у змінну *i1* записується початкове значення, рівне нулю. Оскільки за завданням діапазон пошуку чисел починається з нуля, ми будемо починати перебір діапазону з числа, у якому старший розряд *i1* = 0.

У четвертому рядку програми ми починаємо зовнішній цикл програми. Умовою входу в цикл є значення старшої цифри *i2*, не більше дев'яти. Така умова дає змогу перебрати значення *i2* від 0 до 9 включно.

Тіло зовнішнього циклу показане синім фоном. Воно містить три команди:

1. Команда підготовки до середнього циклу «*i2* = 0». Вона означає, що за кожного значення цифри *i1* цифра *i2* буде перебиратись, починаючи з нуля.
2. Заголовок середнього циклу програми, який забезпечує перебір змінної *i2* до 9 включно.
3. Команда збільшення змінної зовнішнього циклу «*i1* = *i1* + 1». Після виконання цієї команди програма автоматично перейде до заголовка зовнішнього циклу (рядок 4).

Тіло середнього циклу показане жовтим фоном. Його вміст аналогічний вмісту тіла зовнішнього циклу і складається з трьох команд:

1. Команда підготовки до внутрішнього циклу «*i3* = 0». Вона означає, що при кожній парі значень *i1* та *i2* цифра *i3* буде перебиратись, починаючи з нуля.
2. Заголовок внутрішнього циклу програми, який забезпечує перебір змінної *i3* до 9 включно.
3. Команда збільшення змінної середнього циклу «*i2* = *i2* + 1». Після виконання цієї команди програма автоматично перейде до заголовку середнього циклу.

Тіло внутрішнього циклу наразі містить лише дві команди.

Перша команда – це те, заради чого ми створили три цикли. У нашому прикладі команда просто виводить значення змінних *i1*, *i2*, *i3* на екран. Трохи пізніше ми замінимо її на команди, які потрібні для розв'язання нашої задачі. Наразі ж ця команда дає змогу протестувати працездатність нашого «каркасу» і побачити на екрані, чи дійсно змінні *i1*, *i2*, *i3* змінюються в діапазоні від [0; 0; 0] до [9; 9; 9].

Друга команда – збільшення змінної внутрішнього циклу «*i3* = *i3* + 1». Після виконання цієї команди програма автоматично перейде до заголовка внутрішнього циклу.

Розглянутий каркас програми виводить на екран рядки чисел від «0 0 0» до «9 9 9». Отже, програма коректно перебирає числа, утворені змінними *i1*, *i2*, *i3*, в діапазоні від 0 до 999.

Недоліком цього каркасу є те, що він перебирає числа в діапазоні від [0; 0; 0] до [9; 9; 9], тоді як за умовою задачі треба перебирати числа від [0; 0; 0] до [*a1*; *a2*; *a3*]. Через це треба додати у програму обмеження: як тільки значення змінних *i1*, *i2*, *i3* досягають значень *a1*, *a2*, *a3*, треба переривати виконання усіх трьох циклів.

Заведемо спеціальну булеву змінну, яка показує, чи досягли змінні *i1*, *i2*, *i3* значень *a1*, *a2*, *a3*. Таку змінну, яка вказує на якусь ознаку, у програмуванні називають прапорцем (англ. *flag*). Назвемо наш прапорець іменем *f*. До початку усіх циклів у цю змінну покладемо значення False, яке означає, що змінні *i1*, *i2*, *i3* поки ще не досягли значень *a1*, *a2*, *a3*. У кінці тіла внутрішнього циклу ми кожного разу будемо перевіряти: якщо *i1*, *i2*, *i3* вже досягли значень *a1*, *a2*, *a3* (тобто *i1=a1*, *i2=a2*, *i3=a3*), змінна *f* стає рівною True. Ці дії показані в лістингу сірим фоном:

```
a1, a2, a3 = 9, 8, 5     # Задані розряди числа A

A = a1*100 + a2*10 + a3 # Саме число A

f = False                # Ознака необхідності завершення циклу

i1 = 0                   # Початкове значення змінної циклу по сотнях
while i1 <= 9:           # Цикл по тисячах

    i2 = 0               # Початкове значення змінної циклу по десятках
    while i2 <= 9:       # Цикл по сотнях

        i3 = 0           # Початкове значення змінної циклу по одиницях
        while i3 <= 9:  # Цикл по десятках

            print(i1, i2, i3)  # Виводимо поточне число на екран

            if i1==a1 and i2==a2 and i3==a3:
                f = True # Цикли треба переривати

            i3 = i3 + 1  # Збільшення змінної циклу по одиницях

        i2 = i2 + 1      # Збільшення змінної циклу по десятках

    i1 = i1 + 1          # Збільшення змінної циклу по сотнях
```

Тепер у кінці кожного циклу ми повинні додати перевірку: якщо *f* = True, треба виконати команду переривання циклу *break*. Така перевірка повинна бути в кінці кожного тіла циклу.

Додамо у програму перевірки необхідності переривання циклів (показані сірим фоном):

```
a1, a2, a3 = 9, 8, 5      # Задані розряди числа A

A = a1*100 + a2*10 + a3   # Саме число A

f = False                 # Ознака необхідності завершення циклу

i1 = 0                    # Початкове значення змінної циклу по сотнях
while i1 <= 9:            # Цикл по тисячах

    i2 = 0                # Початкове значення змінної циклу по
десятках
    while i2 <= 9:        # Цикл по сотнях

        i3 = 0            # Початкове значення змінної циклу по
одиницях
        while i3 <= 9:    # Цикл по десятках

            print(i1, i2, i3)  # Виводимо поточне число на екран

            if i1==a1 and i2==a2 and i3==a3:
                f = True # Цикли треба переривати

            if f == True: break
             i3 = i3 + 1  # Збільшення змінної циклу по одиницях

        if f == True: break
         i2 = i2 + 1      # Збільшення змінної циклу по десятках

    if f == True: break
    i1 = i1 + 1           # Збільшення змінної циклу по сотнях
```

Як тільки спрацьовує внутрішній оператор *if* з перевіркою «*f* == True» (перший сірий рядок), програма перериває внутрішній цикл і продовжує виконання тіла середнього циклу.

У тілі середнього циклу наступною командою є також оператор *if*, що перевіряє умову «*f* == True» (другий сірий рядок). Оскільки значення змінної *f* все ще дорівнює True, цей оператор *if* теж спрацює, програма вийде з середнього циклу і продовжить виконання тіла зовнішнього циклу.

У тілі зовнішнього циклу також стоїть оператор *if*, що перевіряє умову «*f* == True» (другий сірий рядок). Оскільки значення змінної *f* все ще дорівнює True, цей оператор *if* теж спрацює, програма вийде з зовнішнього циклу і продовжить виконання основної гілки програми. Оскільки після зовнішнього циклу у нас немає інших команд, програму буде завершено.

Тепер наша програма буде виводити на екран числа в діапазоні від 0 до *A*. Але виведення чисел на екран не є тією задачею, яку нам потрібно вирішити. Тому замість команди *print* ми повинні написати набір команд для пошуку найбільшого і найменшого чисел, які одночасно є парними, є

квадратами цілих чисел та дільниками числа *A*. Будемо вирішувати цю задачу частинами.

Спочатку спробуємо знайти усі такі числа і вивести їх на екран. Для цього в тілі внутрішнього циклу нам треба зробити таке:

1) вирахувати число, цифри якого дорівнюють поточним значенням змінних *i1*, *i2*, *i3*;
2) перевірити це число на відповідність заданим умовам.

Фрагмент програми, що виконує ці дії, має такий вигляд:

```
        n = i1*100 + i2*10 + i3   # Поточне число

        if (n > 0 and             # Якщо воно більше нуля
            n % 2 == 0 and        # і є парним
            n ** (1/2) == math.trunc(n ** (1/2)) and # і є квадратом
            A % n == 0):          # і є дільником A
                    print(n)      # Виведемо його на екран
```

Умова «$n > 0$» додана для того, щоб запобігти виконанню операції «A % n» при $n = 0$, оскільки ділення на нуль неможливе. За відсутності умови «$n > 0$» програма видає помилку. Цей фрагмент треба додати у попередню програму замість команди *print*:

```
import math               # Потрібен для використання функції trunc

a1, a2, a3 = 8, 6, 4      # Задані розряди числа A
A = a1*100 + a2*10 + a3   # Саме число A
f = False                 # Ознака необхідності завершення циклу

i1 = 0                    # Початкове значення змінної циклу по сотнях
while i1 <= 9:            # Цикл по тисячах

    i2 = 0                # Початкове значення змінної циклу по
десятках
    while i2 <= 9:        # Цикл по сотнях

        i3 = 0            # Початкове значення змінної циклу по
одиницях
        while i3 <= 9:    # Цикл по десятках

            n = i1*100 + i2*10 + i3   # Поточне число

            if (n > 0 and
                n % 2 == 0 and
                n ** (1/2) == math.trunc(n ** (1/2)) and
                A % n == 0):
                        print(n)

            if i1==a1 and i2==a2 and i3==a3:
                f = True # Цикли треба переривати

            if f == True: break
            i3 = i3 + 1  # Збільшення змінної циклу по одиницях
```

```
        if f == True: break
        i2 = i2 + 1       # Збільшення змінної циклу по десятках

    if f == True: break
    i1 = i1 + 1           # Збільшення змінної циклу по сотнях
```

За умови значеннь *a1*, *a2*, *a3* = 8, 6, 4 матимемо такий результат роботи програми:

```
4
16
36
144
```

Отже, програма працює правильно. Нам залишилося знайти перше і останнє з цих чисел. Для зберігання кожного з них нам буде потрібна окрема змінна. Дамо цим змінним імена *first* і *last* («перший» і «останній»).

Перше число будемо шукати за таким алгоритмом:

1. До початку усіх циклів у змінну *first* покладемо значення –1, яке буде означати, що таке число взагалі не знайдене. Якщо числа, що задовольняють задані умовами, знайдені не будуть, у змінній *first* залишиться значення –1, за яким ми зможемо зрозуміти, що число не знайдене. Якщо число буде знайдене, воно обов'язково буде додатним.
2. У внутрішньому циклі програми замість теперішньої команди *print* додамо перевірку: якщо змінна *first* дорівнює –1, покласти в неї поточне значення числа *n*, інакше нічого не робити. Отже, у змінну *first* нове значення зможе бути записане лише один раз – коли *first* == –1. Отже, це буде перше знайдене число. Далі *first* уже не буде дорівнювати –1, і нове значення туди покладене не буде.

Друге число будемо шукати за таким алгоритмом:

1. До початку усіх циклів у змінну *last* покладемо значення –1, яке буде означати, що таке число взагалі не знайдене. Якщо числа, що задовольняють заданим умовам, знайдені не будуть, у змінній *last* залишиться значення –1, за яким ми зможемо зрозуміти, що число не знайдене. Якщо число буде знайдене, воно обов'язково буде додатним.
2. У внутрішньому циклі програми замість теперішньої команди *print* додамо команду, яка заносить значення змінної *n* у змінну *last*. Ця команда буде виконуватись без усяких умов і оновлюватиме змінну *last* кожного разу, коли буде знайдене чергове значення *n*. Внаслідок цього в кінці роботи програми змінна *last* дорівнюватиме останньому знайденому числу *n*.

```
import math                 # Потрібен для використання функції trunc

a1, a2, a3 = 8, 6, 4        # Задані розряди числа A
A = a1*100 + a2*10 + a3     # Саме число A
f = False                   # Ознака необхідності завершення циклу
first, last = -1, -1        # Перше та останнє числа

i1 = 0                      # Початкове значення змінної циклу по сотнях
while i1 <= 9:              # Цикл по тисячах

    i2 = 0                  # Початкове значення змінної циклу по
десятках
    while i2 <= 9:          # Цикл по сотнях

        i3 = 0              # Початкове значення змінної циклу по
одиницях
        while i3 <= 9:      # Цикл по десятках
            n = i1*100 + i2*10 + i3   # Поточне число
            if (n > 0 and
                n % 2 == 0 and
                n ** (1/2) == math.trunc(n ** (1/2)) and
                A % n == 0):
                      if first == -1: first = n # Перше число
                      last = n                  # Останнє число
                      print(n)                  # Про всяк випадок

            if i1==a1 and i2==a2 and i3==a3:
                f = True # Цикли треба переривати

            if f == True: break
            i3 = i3 + 1  # Збільшення змінної циклу по одиницях

        if f == True: break
        i2 = i2 + 1      # Збільшення змінної циклу по десятках

    if f == True: break
    i1 = i1 + 1          # Збільшення змінної циклу по сотнях

print("Перше число:", first)
print("Останнє число:", last)
```

Після роботи програми знайдені значення змінних *first* і *last* треба вивести на екран. На наступній сторінці наведений лістинг програми, в якому додані команди показані сірим фоном. При значеннях *a1*, *a2*, *a3* = 8, 6, 4 матимемо такий результат роботи програми:

```
4
16
36
144
Перше число: 4
Останнє число: 144
```

Отже, програма працює правильно.

Лабораторна робота № 6
Обробка рядків у мові Python

Метою цієї лабораторної роботи є одержання студентами практичних навичок роботи з рядками символів у програмах мовою Python.

Завдання до лабораторної роботи

Під час лабораторної роботи студент повинен виконати два завдання, кожне з яких оцінюється від 0 до 5 балів залежно від якості виконання (правильності алгоритму, наявності помилок тощо).

Заданий рядок довжиною не менш 100 символів (включно з пробілами та знаками пунктуації), що містить текст українською, російською або англійською мовамою, який складається мінімум із трьох речень. Треба написати програму мовою Python, яка для заданого рядка вирішує два завдання, вказані у табл. 6.1 (номер рядка таблиці відповідає номеру студента в журналі групи). Якщо номер студента в журналі групи більший за номер варіанта в таблиці, студент повинен вчинити так, як у попередніх лабораторних роботах. Під час виконання завдання треба працювати з рядком посимвольно, без використання спеціальних методів для обробки рядків. Кожне завдання виконується незалежно від іншого.

Таблиця 6.1 – Індивідуальні варіанти завдань

Вміст звіту з лабораторної роботи

1. Титульний аркуш.
2. Індивідуальний варіант завдання.
3. Блок-схеми алгоритмів.
4. Лістинг програми з докладними коментарями.
5. Приклади роботи програми (знімки екрана).
6. Висновки до лабораторної роботи.

Рекомендації до виконання лабораторної роботи

Розглянемо деякі базові алгоритми, які можуть бути корисні під час розв'язання багатьох задач.

Приклад 1. Підрахунок кількості літер «а»

Знайдемо, скільки разів у заданому рядку зустрічається маленька або більша кирилична літера «а».

У нижченаведеній програмі спочатку задається змінна *s*, яка містить аналізований рядок. Можна вводити рядок із клавіатури за допомогою команди:

```
s = input()
```

Лістинг програми:

```
s = "Хлюпоче синя річка — ой річка, ой ріка! \
Юрба нас невеличка, зате ж бо гомінка! \
І співи тут, і крики, і радість тут, і сміх. \
А вколо, як музики, - гурти пташок дзвінких."

n_a = 0                          # Кількість знайдених літер

i = 0                            # Починаємо з індексу 0
l = len(s)                       # Обчислюємо довжину рядка s

while i < l:                     # Цикл до останнього індексу

    c = s[i]                     # Беремо черговий символ

    if c == "а" or c == "А":     # Якщо це літера «а» або «А»
        n_a += 1                 # Збільшуємо лічильник на 1

    i = i + 1                    # Переходимо до наступного
індексу

print(n_a)                       # Виводимо кількість літер на
екран
```

Потім ми створюємо змінну *n_a*, яка повинна містити відповідь задачі. Спочатку ми заносимо у *n_a* нульове значення (можливо, що рядок *s* не містить жодної літери «а»).

Для забезпечення роботи циклу, який перебиратиме усі символи рядка,

ми створюємо змінну циклу *i*, якій надаємо початкове значення 0. Також ми кладемо довжину рядка у змінну *l*, яку потім використовуємо в якості кінцевого значення циклу.

У середині циклу *while* черговий (*i*-й) символ рядка прочитується в змінну *c*, яка потім аналізується на рівність шуканому символу. У випадку рівності змінна *n_a* збільшується на одиницю.

Після аналізу чергового символа ми збільшуємо на одиницю значення змінної циклу *i*, переходячи в такий спосіб до наступного символа.

Те ж саме можна було б зробити за допомогою циклу *for*:

```
s = "Хлюпоче синя річка — ой річка, ой ріка! \
Юрба нас невеличка, зате ж бо гомінка! \
І співи тут, і крики, і радість тут, і сміх. \
А вколо, як музики, - гурти пташок дзвінких."

n_a = 0

for c in s: # Перебір усіх символів рядка s за допомогою змінної
c
    if c == "а" or c == "А":
        n_a += 1

print(n_a)
```

Цикл *for* може здаватись більш простим і зручним, оскільки нам не треба самостійно створювати і змінювати змінну циклу. Однак у випадку циклу *for* ми не маємо можливості контролювати змінну циклу. Наприклад, ми не можемо перебирати символи рядка з кінця в початок або з кроком, не рівним одиниці. У випадку циклу *while* ми цілком контролюємо кількість повторів циклу і діапазон змінної циклу.

Також цю задачу можна розв’язати за допомогою стандартного рядкового методу *count*:

```
s = "Хлюпоче синя річка — ой річка, ой ріка! \
Юрба нас невеличка, зате ж бо гомінка! \
І співи тут, і крики, і радість тут, і сміх. \
А вколо, як музики, - гурти пташок дзвінких."

n_a = s.count("а") + s.count("А")
print(n_a)
```

Розглянуті способи можна застосовувати для будь-якої посимвольної обробки рядка.

Приклад 2. Підрахунок кількості слів

Знайдемо кількість слів у рядку *s*. Словом будемо вважати послідовність символів рядка (підрядок), який:

– починається з літерного символа;

– містить лише літерні символи (заголовні або рядкові);

– після слова розташований нелітерний символ або кінець рядка.

Нижче наведена програма для вирішення цієї задачі.

```
1  s = "Хлюпоче синя річка — ой річка, ой ріка! \
2  Юрба нас невеличка, зате ж бо гомінка! \
3  І співи тут, і крики, і радість тут, і сміх. \
4  А вколо, як музики, - гурти пташок дзвінких."
5
6  rus1 = "абвгдеёжзийклмнопрстуфхцчшщъыьэюя"
7  rus2 = "АБВГДЕЁЖЗИЙКЛМНОПРСТУФХЦЧШЩЪЫЬЭЮЯ"
8  ukr1 = "абвгґдеєжзиіїйклмнопрстуфхцчшщьюя"
9  ukr2 = "АБВГҐДЕЄЖЗИІЇЙКЛМНОПРСТУФХЦЧШЩЬЮЯ"
10 eng1 = "abcdefghijklmnopqrstuvwxyz"
11 eng2 = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
12 letters = rus1 + rus2 + ukr1 + ukr2 + eng1 + eng2
13
14 flag = 0     # Набір слова не розпочатий
15 k = 0        # Поки слів не знайдено
16
17 i = 0        # Початкове значення індексу
18 l = len(s)   # Довжина рядка
19 while i < l: # Цикл по всіх символах рядка s
20
21     c = s[i] # Беремо черговий символ
22
23     # Якщо знайдено літеру і набір слова ще не ведеться
24     if c in letters and flag == 0:
25         w = c      # Додаємо у слово перший символ
26         flag = 1   # Починаємо набір слова
27         i = i + 1
28         continue   # Перехід до заголовка циклу
29
30     # Якщо знайдена літера і набір слова вже ведеться
31     if c in letters and flag == 1:
32         w =  w + c       # Додаємо символ до слова
33         i = i + 1
34         continue         # Перехід до заголовка циклу
35
36     # Якщо це нелітерний символ і набір слова не ведеться
37     if c not in letters and flag == 0:
38         i = i + 1
39         continue   # Просто пропускаємо цей символ
40
41     # Якщо це нелітерний символ і набір слова вже ведеться
42     if c not in letters and flag == 1:
43         flag = 0   # Зупиняємо набір слова
44         k = k + 1  # Збільшуємо кількість знайдених слів
45         print("Слово", k, ":", w) # Виведення слова на екран
46         i = i + 1
47
48 print("Усього знайдено", k, "слів.") # Відповідь задачі
```

Розглянемо програму.

У рядках 1–4 задається символьна змінна *s*.

У рядках 6–11 задаються рядкові змінні, що зберігають символи російського, українського та латинського алфавітів. У рядку 12 формується змінна *letters*, яка містить заголовні й малі літери трьох алфавітів, записані підряд в одному рядку.

У рядку 14 задається змінна *flag*, яка буде використовуватися в тілі циклу. Вона встановлюється в одиницю в той момент, коли ми знайшли першу літеру чергового слова. Із цього моменту вона залишається рівною одиниці доти, поки ми не дійдемо до кінця слова (поки не зустрінемо нелітерний символ). Коли нелітерний символ знайдений, змінна *flag* стає рівною нулю і залишається такою до того, як буде знайдений літерний символ (початок наступного слова).

У рядку 15 змінна *k*, що містить кількість знайдених слів, стає рівною нулю. В загальному випадку змінна *s* може не містити жодного слова – бути порожньою або містити нелітерні символи. У цьому випадку після виконання циклу змінна *k* залишиться рівною нулю.

У рядках 17–18 здійснюється підготовка до циклу *while*, який перебиратиме усі символи рядка *s*. У якості початкового значення індексу рядка береться нуль. Також у змінній *l* обчислюється довжина рядка *s*.

У рядку 19 організований цикл *while*, у якому змінна *c* послідовно перебирає усі символи рядка *s*. Для чергового прочитаного символа можливі чотири ситуації, які перевіряються в тілі циклу (рядки 24–46):

1. Символ є літерою (*c* in letters), і набір літер слова у цей момент не виконується (*flag*=0). Це значить, що ми знайшли першу літеру чергового слова й повинні зробити такі дії:

– рядкову змінну *w* зробити рівною знайденому символу *c*;

– вказати програмі, що набір слова розпочатий (*flag* = 1).

Після цих дій ми можемо припинити виконання інших операторів тіла циклу й перейти до його початку за допомогою оператора *continue*, збільшивши перед цим індекс *i* на одиницю (рядки 27, 28).

2. Символ є літерою (*c* in letters), і набір літер слова у цей момент уже ведеться (*flag*=1). Це означає, що ми знайшли чергову (не першу) літеру слова і повинні додати до символьної змінної *w* знайдений символ *c*. Після цього символ *c* можна вважати обробленим. Ми збільшуємо змінну циклу на одиницю і за допомогою оператора *continue* переходимо до наступного повтору циклу (рядки 33, 34).

3. Символ не є літерою (*c* not in letters), і набір літер слова у цей момент не виконується (*flag*=0). Це означає, що ми знайшли якийсь нелітерний символ, розташований між словами. Оскільки за завданням ми шукаємо лише слова, то нелітерний символ ми просто пропускаємо й переходимо до наступного проходу циклу за допомогою оператора *continue*, збільшивши перед цим змінну циклу на одиницю (рядки 38, 39).

4. Символ не є літерою (*c* not in letters), і набір літер слова у цей момент уже ведеться (*flag*=1). Значення *flag*=1 вказує на те, що на попередньому проході циклу була знайдена чергова літера слова й додана до змінної *w*.

Тепер ми знайшли нелітерний символ, розташований відразу після знайденого слова. Виконуємо такі дії:

– скидаємо в нуль змінну *flag*, показуючи, що набір слова зупинений (рядок 43);

– збільшуємо кількість знайдених слів на одиницю (рядок 44);

– виводимо слово і його номер на екран (рядок 45);

– збільшуємо індекс символа на одиницю (рядок 46).

Після цих дій ми повинні перейти до наступного проходу циклу. Оскільки оператор *if* у рядку 42 є останнім оператором тіла циклу, оператор *continue* можна не використовувати – програма автоматично перейде до заголовка циклу.

У рядку 48 виводиться на екран загальна кількість знайдених слів.

Результат роботи програми:

```
Слово 1: Хлюпоче
Слово 2: синя
Слово 3: річка
Слово 4: ой
Слово 5: річка
Слово 6: ой
Слово 7: ріка
Слово 8: Юрба
Слово 9: нас
Слово 10: невеличка
Слово 11: зате
Слово 12: ж
Слово 13: бо
Слово 14: гомінка
Слово 15: І
Слово 16: співи
Слово 17: тут
Слово 18: і
Слово 19: крики
Слово 20: і
Слово 21: радість
Слово 22: тут
Слово 23: і
Слово 24: сміх
Слово 25: А
Слово 26: вколо
Слово 27: як
Слово 28: музики
Слово 29: гурти
Слово 30: пташок
Слово 31: дзвінких
Усього знайдено 31 слів.
```

Як можна бачити, основні дії зі знайденим словом здійснюються в рядках 44 та 45. У нашому прикладі це збільшення кількості знайдених слів і виведення поточного знайденого слова на екран. Якщо зі знайденим словом потрібно виконати якісь додаткові дії (знайти довжину слова, підрахувати кількість голосних букв, знайти парні букви тощо), ці дії повинні виконуватися всередині четвертого оператора *if* замість рядків 44 та 45.

Оскільки ми перебираємо символи рядка *s* послідовно, замість циклу *while* може бути використаний цикл *for*. Це дасть змогу скоротити програму на декілька команд:

```
s = "Хлюпоче синя річка — ой річка, ой ріка! \
Юрба нас невеличка, зате ж бо гомінка! \
І співи тут, і крики, і радість тут, і сміх. \
А вколо, як музики, - гурти пташок дзвінких."

rus1 = "абвгдеёжзийклмнопрстуфхцчшщъыьэюя"
rus2 = "АБВГДЕЁЖЗИЙКЛМНОПРСТУФХЦЧШЩЪЫЬЭЮЯ"
ukr1 = "абвгґдеєжзиіїйклмнопрстуфхцчшщьюя"
ukr2 = "АБВГҐДЕЄЖЗИІЇЙКЛМНОПРСТУФХЦЧШЩЬЮЯ"
eng1 = "abcdefghijklmnopqrstuvwxyz"
eng2 = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
letters = rus1 + rus2 + ukr1 + ukr2 + eng1 + eng2

flag = 0     # Набір слова не розпочатий
k = 0        # Поки слів не знайдено

for c in s:  # Цикл по всіх символах рядка s

    # Якщо знайдено літеру і набір слова ще не ведеться
    if c in letters and flag == 0:
        w = c      # Додаємо у слово перший символ
        flag = 1   # Починаємо набір слова
        continue   # Перехід до заголовка циклу

    # Якщо знайдена літера і набір слова вже ведеться
    if c in letters and flag == 1:
        w =  w + c       # Додаємо символ до слова
        continue         # Перехід до заголовка циклу

    # Якщо це нелітерний символ і набір слова не ведеться
    if c not in letters and flag == 0:
        continue   # Просто пропускаємо цей символ

    # Якщо це нелітерний символ і набір слова вже ведеться
    if c not in letters and flag == 1:
        flag = 0   # Зупиняємо набір слова
        k = k + 1  # Збільшуємо кількість знайдених слів
        print("Слово", k, ":", w) # Виведення слова на екран

print("Усього знайдено", k, "слів.") # Відповідь задачі
```

Приклад 3. Підрахунок кількості цілих чисел

Ціле число може починатися із цифри, зі знака «+» або знака «–».

Лістинг програми наведений нижче та загалом подібний до програми з попереднього прикладу.

```
s = "Хлюпоче 245 річка — ой річка, ой ріка! \
Юрба нас невеличка, зате ж -982 гомінка! \
І співи тут, і крики, і радість 0, і сміх. \
А вколо, як музики, - -01230 пташок дзвінких."

numbers = "0123456789"

flag = 0
k = 0

for c in s:

    if flag == 0 and (c == "+" or c == "-"):
        w = c
        flag = 1
        continue

    if flag == 0 and c in numbers:
        w = c
        flag = 2
        continue

    if flag == 1 and c not in numbers:
        flag = 0
        continue

    if flag == 1 and c in numbers:
        flag = 2
        w = w + c
        continue

    if flag == 2 and c in numbers:
        w = w + c
        continue

    if flag == 2 and c not in numbers:
        k = k + 1
        print("Число", k, ":", w)
        flag = 0

print("Усього знайдено", k, "чисел.")
```

Відмінності з попередньою програмою полягають у такому.

У попередній програмі (під час знаходження окремих слів) програма могла перебувати у двох станах: у процесі пошуку першої літери слова та у процесі політерного набору слова. Перша ситуація позначалася прапорцем *flag* = 0, друга – *flag* = 1. Якщо при *flag* = 0 ми знаходили літеру, ми точно знали, що це початок наступного слова, і встановлювали *flag* в одиницю.

Під час пошуку чисел ситуація трохи ускладнюється.

1. Нехай значення *flag* = 0 означає, що ми поки не знайшли початок числа і не почали процес додавання символів числа в окремий рядок *w*. Загалом число може починатися не з цифри, а зі знака «+» або «–». Якщо ми при *flag* = 0 зустріли знак «+» або «–», то можливо, це є початком чергового числа. Але необов'язково: якщо після знака йде не цифра, а інший символ, то це не початок числа, а щось інше.

Тому коли ми зустрічаємо знак «+» або «–» при *flag* = 0, ми даємо змінній *flag* значення 1. Це означає, що ми знайшли знак числа та чекаємо, що далі будуть іти цифрові символи. Ці дії виконуються в рядках 13–16.

Якщо ж при *flag* = 0 ми зустріли цифру, то це однозначно початок чергового числа. Навіть якщо число складається з однієї цифри – це все одно початок числа. У цьому випадку ми встановлюємо змінну *flag* у значення 2 та робимо змінну *w* рівною знайденому символу *c* (рядки 18–21).

2. У рядках 23–30 аналізується ситуація, коли *flag* = 1, тобто коли ми на попередньому кроці циклу знайшли символ «+» або «–».

Якщо на цьому кроці знайдений нецифровий символ (*c* not in numbers), то раніше знайдений знак не належить до числа. Нам потрібно знов перейти до пошуку першого символу числа. Для цього в рядках 24–25 ми переводимо змінну *flag* у значення 0 і повторюємо цикл. Змінну *w*, у якій зараз перебуває знак, можна не очищувати – вона буде оновлена на наступних повторах циклу разом зі зміною *flag* з нуля в одиницю або двійку.

Якщо ж був знайдений цифровий символ (*c* in numbers), ми приступаємо до набору цифрових символів числа. Щоб програма пам'ятала про це, ми задаємо *flag* = 2 і додаємо знайдений символ *c* до рядка *w*, у якому вже перебуває знак числа, знайдений на попередньому кроці. Це робиться в рядках 27–30.

3. У рядках 32–39 аналізується ситуація, коли *flag* = 2, тобто коли ми на попередньому кроці циклу знайшли цифровий символ і виконуємо набір цифрових символів у рядок *w*. Тут можливі такі ситуації.

Якщо поточний символ є цифрою (рядок 32), то ми додаємо його в кінець рядка *w* і переходимо до наступного повтору циклу.

Якщо поточний символ не є цифрою, то ми знайшли символ, розташований після числа. Тобто до поточного числа нові цифри додаватися не будуть, і число можна вважати цілком сформованим. Ми збільшуємо кількість знайдених чисел на одиницю (рядок 37), виводимо знайдене число на екран (рядок 38) і скидаємо змінну *flag* у нуль (рядок 39). Значення *flag*=0 переводить програму в режим пошуку першого символу чергового числа (цим символом можуть бути знак числа або цифра).

Далі тіло циклу закінчується, оскільки ніякі інші ситуації в програмі виникнути не можуть. Винятком є ситуація, коли число перебуває в самому кінці рядка (наприклад, «абвґд 235»). У цьому випадку число не буде виведене на екран, оскільки цикл перерветься відразу після досягнення кінця рядка. Для виправлення помилки достатньо після циклу перевірити, чи було знайдене чергове число, і якщо так – вивести його на екран:

```
if flag == 2:
    k = k + 1
    print("Число", k, ":", w)

print("Усього знайдено", k, "чисел.")
```

Результат роботи програми:

```
Число 1 : 245
Число 2 : -982
Число 3 : 0
Число 4 : -01230
Усього знайдено 4 чисел.
```

Із використанням подібного алгоритму можна виділяти з рядка різні формалізовані типи даних: http-адреси, адреси електронної пошти, числа із плаваючою комою, дати, час тощо. Головне – мати чіткі правила запису цих даних.

Студентам пропонується самостійно переробити цю програму так, щоб замість циклу *for* був використаний цикл *while*.

Приклад 4. Вивести на екран усі фрагменти рядка, укладені в круглі дужки

Програма повністю аналогічна двом попереднім програмам, але має більш простий алгоритм.

```
s = "Хлюпоче (синя) річка — ой річка, ой ріка! \
Юрба (((нас))) невеличка, зате( )ж бо гомінка! \
І співи тут, і крики, (і) радість тут, і сміх. \
(А вколо, як музики, - гурти пташок дзвінких.)"

flag = 0

for c in s:

    if flag == 0 and c == "(":
        w = ""
        flag = 1
        continue

    if flag == 1 and c != ")":
        w = w + c
        continue

    if flag == 1 and c == ")":
        print(w)
        flag = 0
```

До початку циклу змінна *flag* робиться рівною нулю. Це переводить програму в режим пошуку відкриваючої дужки.

У тілі циклу перевіряються три умови:

1) якщо *flag* = 0 і знайдено відкриваючу дужку, ми переходимо в режим додавання символів до рядка *w*. Для цього ми створюємо порожню рядкову змінну *w* і заносимо у змінну *flag* одиницю;

2) якщо *flag* = 1 і знайдений символ не є закриваючою дужкою, ми додаємо цей символ до рядка *w*;

3) якщо *flag* = 1 і знайдено символ «) », ми виводимо рядок *w* на екран і знову переходимо в режим пошуку відкриваючої дужки (задаємо *flag* = 0).

Результат роботи програми:

```
синя
((нас

і
А вколо, як музики, - гурти пташок дзвінких.
```

Із результатів роботи програми видно, що під час аналізу підрядка "**(((нас)))**" програма знаходить першу відкриваючу дужку, після чого доходить до першої закриваючої дужки й виводить на екран символи між цими дужками: "**((нас**". Можливо, комусь такий результат здасться неправильним, однак він точно відповідає умові завдання.

Зазначимо, що сучасні компілятори, які виконують символьний аналіз рядків на зразок нижчеподаного, використовують більш складні алгоритми аналізу рядків.

```
((a+0.456)*(12*(((d+3)*2-1)/c)+b)
```

Приклад 5. Виведення на екран усіх речень

Незважаючи на те, що речення в загальному випадку є довшим за слово, пошук речень подібний до пошуку слів. Для початку потрібно визначити ознаки початку та кінця речення. Початком речення будемо вважати будь-який літерний символ, кінцем речення – один із символів «.», «!» «?».

Такий підхід не є цілком правильним, оскільки речення не обов’язково має починатися з літери. Воно може починатися з числа, із символа лапок або дужки, зі знака тире тощо. Закінченням речення можуть бути не тільки знаки «.», «!» «?», але й лапки, трикрапка, закриваюча дужка, набори з декількох знаків «?!», «...», «???», «!!!» та інші ознаки. Однак у межах нашого прикладу будемо підходити до цього питання спрощено: початком речення будемо вважати тільки літеру, закінченням речення – тільки знаки «.», «!», «?».

Таке спрощення значно знижує складність алгоритму і робить його схожим на алгоритми попередніх програм:

```
s = "Хлюпоче синя річка — ой річка, ой ріка! \
Юрба нас невеличка, зате ж бо гомінка! \
І співи тут, і крики, і радість тут, і сміх. \
А вколо, як музики, - гурти пташок дзвінких."

rus1 = "абвгдеёжзийклмнопрстуфхцчшщъыьэюя"
rus2 = "АБВГДЕЁЖЗИЙКЛМНОПРСТУФХЦЧШЩЪЫЬЭЮЯ"
ukr1 = "абвгґдеєжзиіїйклмнопрстуфхцчшщьюя"
ukr2 = "АБВГҐДЕЄЖЗИІЇЙКЛМНОПРСТУФХЦЧШЩЬЮЯ"
eng1 = "abcdefghijklmnopqrstuvwxyz"
eng2 = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"

letters = rus1 + rus2 + ukr1 + ukr2 + eng1 + eng2
fin = ".?!"

flag = 0

for c in s:

    if flag == 0 and c in letters:
        w = c
        flag = 1
        continue

    if flag == 1 and c not in fin:
        w = w + c
        continue

    if flag == 1 and c in fin:
        w = w + c
        print(w)
        flag = 0
```

Під час аналізу чергового символа використовуються два допоміжні рядки: змінна *letters* містить усі літерні символи на трьох мовах, змінна *fin* містить символи, що вважаються закінченням речення.

Змінна *flag* використовується в такий спосіб. При *flag*=0 програма перебуває в режимі пошуку початку речення. При *flag*=1 програма перебуває в режимі пошуку кінця речення.

Якщо *flag*=0 і була знайдена літера, то знайдений початок речення. Ми робимо рядок *w* рівним знайденому символу та переводимо програму в режим пошуку кінця речення (*flag*=1).

Якщо *flag*=1 і був знайдений символ, що не входить до *fin*, то це черговий символ поточного речення. Ми додаємо його до змінної *w* і переходимо до наступного кроку циклу.

Якщо *flag*=1 і знайдений символ входить до *fin*, то це останній символ речення. Ми додаємо його до змінної *w* і виводимо її на екран. Після цього ми кладемо у змінну *flag* значення 0 і в такий спосіб переводимо програму в режим пошуку початку наступного речення.

Якщо нам потрібно знайти речення і в якийсь спосіб його обробити, алгоритм обробки повинен розташовуватися всередині четвертого оператора *if* замість команди:

```
        print(w)
```

Інша структура програми має бути збережена.

Приклад 6. В усіх словах розташувати літери у зворотному порядку

Розташування літер слова у зворотному порядку легко робиться за допомогою операції зрізу, виконуваного із кроком –1 від початкового до кінцевого символа:

```
w = "12345"
w1 = w[::-1]
```

Пошук окремих слів також нескладний і докладно розглянутий вище у прикладі 2.

Особливістю цієї задачі, на відміну від попередніх, є те, що ми не просто аналізуємо вміст рядка, а формуємо новий рядок на основі вхідного. У новому рядку переверненими мають бути лише слова, а інші символи повинні зберегти свій початковий порядок.

У програмі із прикладу 1 ми діяли в такий спосіб:

– якщо символ не є літерою, він відкидається;

– якщо символ є літерою, він додається до слова;

– якщо слово сформоване, воно виводиться на екран.

Модифікуємо програму в такий спосіб:

– якщо символ не є літерою, він додається до створюваного рядка;

– якщо символ є літерою, він додається до поточного слова;

– якщо слово сформоване, воно «перевертається» і додається до створюваного рядка.

Лістинг програми має такий вигляд:

```
s = "Хлюпоче синя річка — ой річка, ой ріка! \
Юрба нас невеличка, зате ж бо гомінка! \
І співи тут, і крики, і радість тут, і сміх. \
А вколо, як музики, - гурти пташок дзвінких."

rus1 = "абвгдеёжзийклмнопрстуфхцчшщъыьэюя"
rus2 = "АБВГДЕЁЖЗИЙКЛМНОПРСТУФХЦЧШЩЪЫЬЭЮЯ"
ukr1 = "абвгґдеєжзиіїйклмнопрстуфхцчшщьюя"
ukr2 = "АБВГҐДЕЄЖЗИІЇЙКЛМНОПРСТУФХЦЧШЩЬЮЯ"
eng1 = "abcdefghijklmnopqrstuvwxyz"
eng2 = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"

letters = rus1 + rus2 + ukr1 + ukr2 + eng1 + eng2
```

```
flag = 0  # Набір слова не розпочатий
s1 = ""   # Новий рядок (поки що пустий)

for c in s: # Цикл по всіх символах рядка s

    # Якщо було знайдено літеру і набір слова не ведеться
    if c in letters and flag == 0:
        w = c       # Додаємо до слова w перший символ
        flag = 1    # Ознака початку набору слова
        continue    # Перехід до заголовка циклу

    # Якщо було знайдено літеру і набір слова вже ведеться
    if c in letters and flag == 1:
        w =  w + c      # Додаємо поточний символ до слова
        continue        # Перехід до заголовка циклу

    # Якщо знайдений нелітерний символ і набір слова не ведеться
    if c not in letters and flag == 0:
        s1 = s1 + c     # Додаємо символ до нового рядка
        continue

    # Якщо знайдений нелітерний символ і набір слова вже
ведеться
    if c not in letters and flag == 1:
        flag = 0        # Зупиняємо набір слова
        w = w[::-1]     # Перевертаємо рядок
        s1 = s1 + w     # Додаємо слово до нового рядка
        s1 = s1 + c     # Додаємо поточний символ (після слова)

print(s1)
```

Результат роботи програми:

```
ечопюлХ янис акчір — йо акчір, йо акір! абрЮ сан акчилевен, етаз
ж об акнімог! І ивіпс тут, і икирк, і ьтсідар тут, і хімс. А
олокв, кя икизум, - итруг кошатп хикнівзд.
```

Якщо зі знайденим словом потрібно виконати не «переворот», а інші операції, це має виконуватися в останньому операторі *if* замість рядка:

```
        w = w[::-1]
```

Лабораторна робота № 7

Створення користувацьких функцій у мові Python

Метою цієї лабораторної роботи є одержання студентами практичних навичок роботи з користувацькими функціями у програмах мовою Python.

Завдання до лабораторної роботи

Лабораторна робота містить два завдання, кожне з яких оцінюється окремо від іншого.

Завдання 1

Написати функцію, яка розв’язує задачу відповідно до варіанта, заданого в таблиці 7.1. Варіант завдання відповідає номеру студента в журналі групи (не підгрупи). Якщо номера за журналом перевищує кількість варіантів у таблиці, треба від номеру за журналом відняти кількість варіантів.

Під час виконання завдання треба самостійно обрати кількість та імена аргументів функції, а також продумати, чи можна зробити значення якихось аргументів «за замовчуванням». В основній програмі продемонструвати декілька способів виклику функції. Обов’язково навести словесний опис та блок-схему алгоритму. У разі виникнення труднощів під час розробки алгоритму варто звернутися до профільної літератури або мережі Інтернет.

Таблиця 7.1 – Варіанти до завдання 1

Завдання 2

Початковими даними для цього завдання є лабораторна робота № 4. Вона містить три завдання, присвячені операціям розгалуження.

Необхідно реалізувати кожне завдання у вигляді окремої функції. Треба самостійно обрати кількість та імена аргументів функцій, а також продумати, чи можна зробити значення якихось аргументів «за замовчуванням». Усі функції мають бути самостійні і незалежні одна від одної. В основній програмі продемонструвати декілька прикладів виклику функцій.

Вміст звіту з лабораторної роботи

1. Титульний аркуш.
2. Індивідуальний варіант завдання.
3. Опис, блок-схема, лістинг і результати роботи програми для завдання 1.
4. Лістинг і результати роботи програми для завдання 2.
5. Висновки до лабораторної роботи.

Лабораторна робота № 8
Генерація псевдовипадкових чисел

Метою лабораторної роботи є одержання студентами практичних навичок роботи з функціями генерації псевдовипадкових чисел модуля *random* мови Python.

Завдання до лабораторної роботи

Варіантом завдання до лабораторної роботи є варіант із лабораторної роботи № 3, яка присвячена побудові класів. Отже, вхідними даними для цієї лабораторної роботи є певна предметна область.

Під час лабораторної роботи треба виконати таке:

1. Розробити клас, який належить до заданої предметної області і містить не менш ніж 5 числових атрибутів цілого типу і не менш ніж 5 атрибутів дійсного (дробового) типу.
2. Створити 5 екземплярів цього класу (цикли та списки не використовувати).
3. Заповнити кожен атрибут усіх екземплярів класу псевдовипадковими числами у діапазоні значень, що припустимі для цього атрибуту (цикли та списки не використовувати).
4. Вивести на екран у зручній формі значення атрибутів усіх екземплярів класу. Для цього доцільно розробити окрему функцію, в яку буде передаватись екземпляр класу.

Вміст звіту з лабораторної роботи

1. Титульний аркуш.
2. Індивідуальний варіант завдання.
3. Докладний опис класу і лістинг його завдання мовою Python.
4. Лістинг програми для виконання пунктів 2–4.
5. Результати роботи програми.
6. Висновки до лабораторної роботи.

Приклад виконання лабораторної роботи

Створимо клас, що описує студента. Для спрощення в нашому класі будуть лише три атрибути:

– номер курсу;
– середній бал;
– номер спеціальності.

Клас описується таким фрагментом коду:

```
class student:     # Клас "студент"
    kurs = 1       # Номер курсу
    ball = 78.5    # Середній бал
    spec = 122     # Номер спеціальності
```

Створимо 5 екземплярів цього класу:

```
e1 = student() # Створюємо перший екземпляр класу
e2 = student() # Створюємо другий екземпляр класу
e3 = student() # Створюємо третій екземпляр класу
e4 = student() # Створюємо четвертий екземпляр класу
e5 = student() # Створюємо п'ятий екземпляр класу
```

Заповнимо атрибути усіх екземплярів класу псевдовипадковими числами, для чого підключимо модуль *random*:

```
import random
```

Заповнимо атрибут, що відповідає за курс студента.

```
e1.kurs = random.randint(1, 4)     # Курс від 1 до 4
e2.kurs = random.randint(1, 4)
e3.kurs = random.randint(1, 4)
e4.kurs = random.randint(1, 4)
e5.kurs = random.randint(1, 4)
```

У цих командах ми безпосередньо використовуємо функцію *randint* і передаємо в неї два параметри, які задають діапазон генерованих цілих чисел [1; 4]. Отже, в атрибут *kurs* можуть бути покладені числа 1, 2, 3 або 4.

Атрибут *ball* заповнимо дійсними (дробовими) випадковими значеннями в діапазоні [60; 100]. Для цього замість функції *randint* використаємо функцію *uniform*:

```
e1.ball = random.uniform(60, 100)   # Бал від 60 до 100
e2.ball = random.uniform(60, 100)
e3.ball = random.uniform(60, 100)
e4.ball = random.uniform(60, 100)
e5.ball = random.uniform(60, 100)
```

Функція *uniform* генерує дробові числа з високою точністю, наприклад, 94.47905594488954. Для нашого випадку така точність не потрібна, нам достатньо одного знака після десяткової крапки. Щоб скоротити точність числа, можна вчинити так: взяти псевдовипадкове ціле число в діапазоні [600; 1000], а потім поділити його на 10:

```
e1.ball = random.randint(600, 1000) / 10
e2.ball = random.randint(600, 1000) / 10
e3.ball = random.randint(600, 1000) / 10
e4.ball = random.randint(600, 1000) / 10
e5.ball = random.randint(600, 1000) / 10
```

Тепер атрибут *ball* буде набувати таких значеннь: 87.5, 71.4, 96.7 тощо. Замість такого способу можна було б скористатись функціями округлення чисел, що наявні у модулі *math* (але модуль *math* треба було б підключати).

Щоб обрати спеціальність, треба діяти трохи інакше. Нехай на факультеті є три умовно «комп'ютерні» спеціальності: 122 Комп'ютерні науки, 113 Прикладна математика та 125 Кібербезпека. Ці числа не розташовані підряд, тому використовувати для їх вибору функцію *randint* неможливо.

Хоча чому неможливо? Скористаємось функцією *randint*, щоб генерувати цілі числа в діапазоні [1; 3]. Після генерації кожне число перетворимо за допомогою оператора *if* у номер спеціальності. Число 1 буде замінене на 122, число 2 – на 113, число 3 – на 125. Ось як це буде виглядати.

```
x = random.randint(1, 3)
if x == 1:
    e1.spec = 122
if x == 2:
    e1.spec = 113
if x == 3:
    e1.spec = 125
```

Подібний фрагмент коду треба повторити для екземплярів *e2*, *e3*, *e4*, *e5*. Замість функції *randint* і оператора *if* можна скористатись функцією *choice*, попередньо зробивши список допустимих номерів спеціальностей.

Індивідуальне творче завдання (перший семестр)

Програмування знаходить застосування майже в усіх сферах людської діяльності. Людина, що є фахівцем у програмуванні, у своїй роботі постійно зіштовхується з особливостями тієї галузі, в якій працює вона або її підприємство. Якщо програміст працює у сфері металургії, він, окрім самого програмування, повинен певною мірою розуміти особливості організації металургійного виробництва, етапи та нормативи виготовлення продукції, професійну термінологію, завдання тих чи інших професій тощо. Програмісту не треба бути досвідченим металургом, але треба розуміти цю сферу на рівні, достатньому для розробки, впровадження та супроводу відповідного програмного забезпечення.

Те ж саме стосується роботи програміста в інших галузях, як-от залізничний транспорт, торгівля, медицина, освіта, туризм та інші. Програміст ніколи не повинен обмежувати себе лише знанням програмування – інакше він не зможе бути корисним у сучасному суспільстві. Якщо програміст потрапляє в нову для нього галузь, він швидко вивчає її базові основи та приступає до своєї роботи.

Індивідуальне творче завдання полягає в написанні програми, яка буде корисною в повсякденній діяльності людини певної професії. Кожному студенту, згідно з його варіантом завдання, надається одна із професій, для якої він повинен розробити «корисну» програму. Студент повинен з’ясувати, яку допомогу він, як програміст, може надати фахівцеві цієї професії. Це можуть бути якісь розрахунки, довідкові дані, перевірка коректності якихось значень чи щось інше. Загалом функціонал програми може бути корисним не лише фахівцю заданої професії, але й фахівцям суміжних спеціальностей. Головна вимога програми – вона повинна дійсно робити щось корисне.

Загальна структура програми зображена у вигляді блок-схеми на рисунку нижче.

Після запуску програми на екрані повинен з’являтися перелік пунктів меню, вибір яких користувач робить за допомогою клавіатури. Залежно від обраного пункту меню програма виконує одну зі своїх функцій, що передбачає введення вхідних даних, їх обробку та виведення результату. Після виконання будь-якої зі своїх функцій програма повертається до головного меню. Головне меню повинне передбачати пункт «Вихід», під час вибору якого програма завершує роботу.

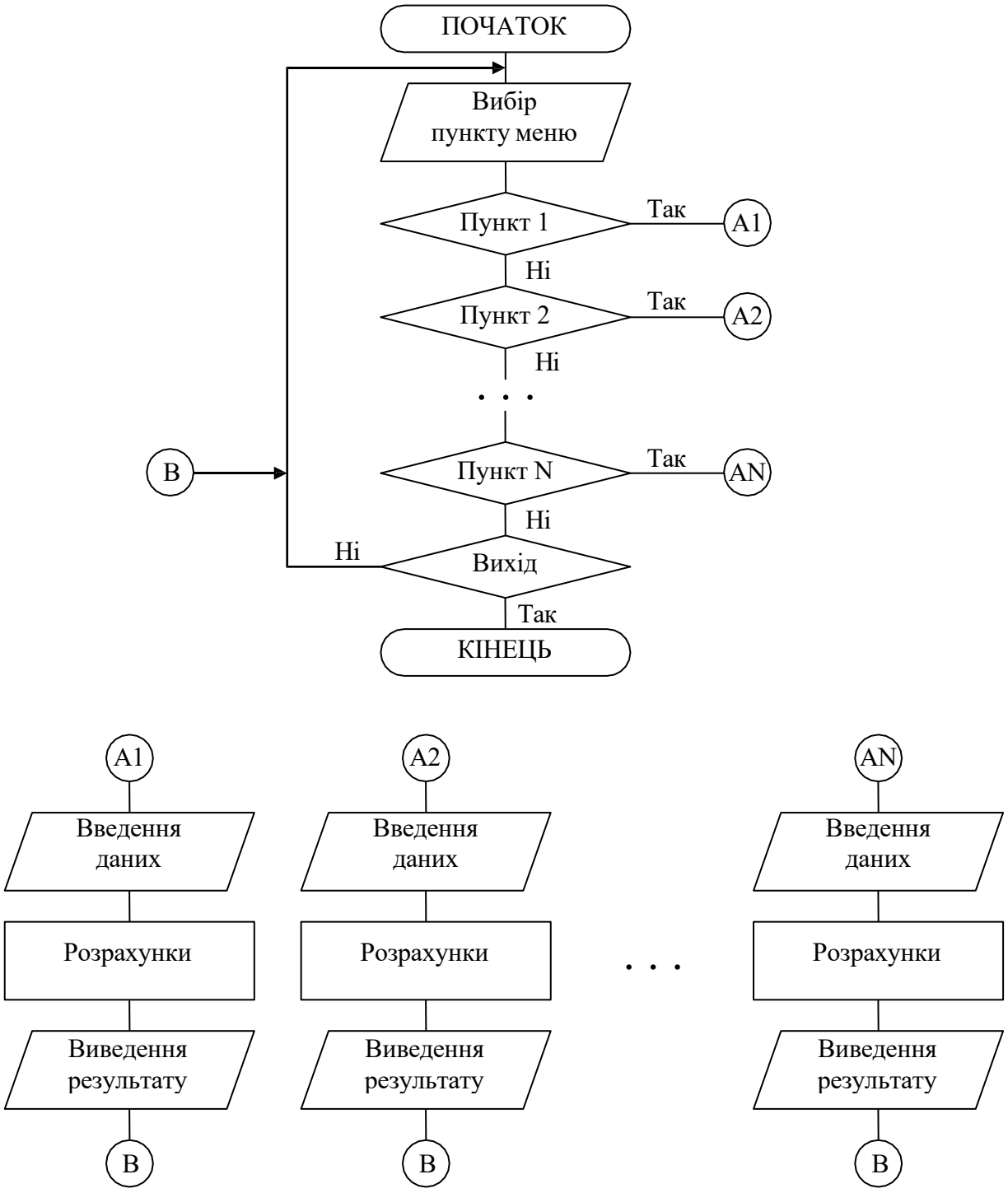


Рисунок І-1 – Загальна структура програми

Вимоги для розроблюваної програми

1. Реалізація не менш ніж семи функцій (корисних дій).
2. Наявність пункту меню «Про програму», під час вибору якого виводитиметься інформація про розробника програми.
3. Наявність пункту меню «Вихід», під час вибору якого здійснюється вихід із програми.
4. Перевірка на належність введених даних до припустимого діапазону.

5. Використання циклів хоча б в одній із функцій програми.
6. Використання хоча б одного класу.
7. Використання хоча б однієї користувацької функції.
8. Охайний інтерфейс програми.

Індивідуальні варіанти завдань

Варіант завдання вибирається в межах своєї підгрупи (не в межах загальної групи), для чого студент повинен знати свій номер у журналі своєї підгрупи. Студенти підгруп А, Б обирають варіанти із таблиці 1, студенти підгруп В, Г і Д обирають варіанти із таблиці 2. У таблицях стовбець *N* відповідає номеру студента в журналі підгрупи.

Таблиця І-1 – Варіанти завдань для підгруп А, Б

N	Професія	*N*	Професія
1	Фермер	8	Лікар
2	Менеджер банку	9	Збірник меблів
3	Викладач	10	Мисливець
4	Ріелтор	11	Геолог
5	Фізик	12	Перекладач
6	Тренер	13	Секретар
7	Кухар	14	Ювелір

Таблиця І-2 – Варіанти завдань для підгруп В, Г, Д

N	Професія	*N*	Професія
1	Касир	8	Економіст
2	Геолог	9	Хімік
3	Астроном	10	Архітектор
4	Туроператор	11	Водій
5	Фармацевт	12	Агроном
6	Археолог	13	Кінолог
7	Логіст	14	Лоцман

Структура звіту з індивідуального творчого завдання

1. Титульний аркуш, оформлений за вимогами університету.
2. Індивідуальний варіант завдання.
3. Блок-схема програми із деталізацією усіх функцій (режимів роботи).
4. Лістинг програми з докладними коментарями.
5. Знімки екрана, що демонструють усі режими роботи програми.

Приклад виконання завдання

Розглянемо скорочений приклад виконання завдання для варіанта «школяр». Хоча школяр – це не професія, для нього теж існує багато задач, у яких може допомогти комп'ютерна програма. В якості таких задач обрані:

– розв'язання квадратного рівняння;
– обчислення гіпотенузи трикутника за значеннями катетів;
– виведення деяких корисних математичних констант.

Треба зауважити, що в цьому прикладі реалізовані лише три задачі, тоді як у роботі потрібно реалізувати ***мінімум сім задач*** для заданої професії.

```
# Індивідуальне творче завдання
# Варіант завдання: школяр

while True:
    print("/----------------------------\\")
    print('| Програма "Помічник школяра" |')
    print("\\----------------------------/")
    print()
    print("1. Вирішення квадратного рівняння")
    print("2. Обчислення гіпотенузи")
    print("3. Базові математичні константи")
    print("4. Про програму")
    print("0. Вихід")
    print()

    s = input("Оберіть пункт меню -> ")

    if s == "0":   # Вихід з програми
        print("Програму завершено.")
        exit()

    if s == "4":   # Про програму
        print("Розробив: Бабаков Роман Маркович")
        print("          група КН-Д23")
        print()
        print("Натисніть Enter для продовження.")
        input()

    if s == "1":   # Квадратне рівняння
        print("Вирішення квадратного рівняння ax^2+bx+c=0")
        print()
        a = input("Введіть a : ")
        b = input("Введіть b : ")
        c = input("Введіть c : ")
        a, b, c = int(a), int(b), int(c)

        D = b*b-4*a*c   # Дискримінант
        print("Дискримінант D =", D)
```

```
        if D < 0:       # Якщо дискримінант від'ємний
            print("Дискримінант  < 0, коренів немає.")

        if D == 0:      # Якщо дискримінант дорівнює нулю
            x = (-b + D**0.5) / (2*a)
            print("Корені x1 = x2 =", x)

        if D > 0:
            x1 = (-b + D**0.5) / (2*a)
            x2 = (-b + D**0.5) / (2*a)
            print("Корені: x1 =", x1, ", x2 =", x2)

        print()
        print("Натисніть Enter для продовження.")
        input()

    if s == "2":   # Обчислення гіпотенузи
        print("Обчислення гіпотенузи за довжинами катетів")
        print()

        a = input("Введіть довжину катета a : ")
        b = input("Введіть довжину катета b : ")

        a, b = int(a), int(b)

        if a > 0  and b > 0:
            g = (a*a + b*b)**0.5
            print("Гіпотенуза =", g)
        else:
            print("Помилка: значення катетів мають бути
додатні!")

        print()
        print("Натисніть Enter для продовження.")
        input()

    if s == "3":   # Математичні константи
        print()
        print("Базові математичні константи:")
        print("pi = 3.14")
        print("e = 2.71")
        print("tau = 2*pi = 6.28")
        print("Корінь із 2 = 1.41")
        print("Корінь із 3 = 1.73")
        print()
        print("Натисніть Enter для продовження.")
        input()
```

Нижче наведені результати роботи програми в різних режимах:

```
/----------------------------\
| Програма "Помічник школяра" |
\----------------------------/

1. Розв'язання квадратного рівняння
2. Обчислення гіпотенузи
3. Базові математичні константи
4. Про програму
0. Вихід

Оберіть пункт меню -> 1
Розв'язання квадратного рівняння ax^2+bx+c=0

Введіть a : 1
Введіть b : 4
Введіть c : 3
Дискримінант D = 4
Корені: x1 = -1.0 , x2 = -1.0

Натисніть Enter для продовження.
```

```
/----------------------------\
| Програма "Помічник школяра" |
\----------------------------/

1. Розв'язання квадратного рівняння
2. Обчислення гіпотенузи
3. Базові математичні константи
4. Про програму
0. Вихід

Оберіть пункт меню -> 2
Обчислення гіпотенузи за довжинами катетів

Введіть довжину катету a : 3
Введіть довжину катету b : 4
Гіпотенуза = 5.0

Натисніть Enter для продовження.
```

```
/----------------------------\
| Програма "Помічник школяра" |
\----------------------------/

1. Розв'язання квадратного рівняння
2. Обчислення гіпотенузи
3. Базові математичні константи
4. Про програму
0. Вихід

Оберіть пункт меню -> 3
```

```
Базові математичні константи:
pi = 3.14
e = 2.71
tau = 2*pi = 6.28
Корінь із 2 = 1.41
Корінь із 3 = 1.73

Натисніть Enter для продовження.
```

```
/----------------------------\
| Програма "Помічник школяра" |
\----------------------------/

1. Розв'язання квадратного рівняння
2. Обчислення гіпотенузи
3. Базові математичні константи
4. Про програму
0. Вихід

Оберіть пункт меню -> 4
Розробив: Бабаков Роман Маркович
          група КН-Д23

Натисніть Enter для продовження.
```

```
/----------------------------\
| Програма "Помічник школяра" |
\----------------------------/

1. Розв'язання квадратного рівняння
2. Обчислення гіпотенузи
3. Базові математичні константи
4. Про програму
0. Вихід

Оберіть пункт меню -> 0
Програму завершено.
```

Лабораторна робота № 9

Обробка одновимірних масивів

Метою лабораторної роботи є одержання студентами практичних навичок роботи з одновимірними масивами, побудованими за допомогою списків Python.

Завдання до лабораторної роботи

Лабораторна робота містить два завдання, кожне з яких оцінюється окремо від іншого.

Завдання 1

Це завдання присвячене виконанню завдань пошуку елементів у масиві без внесення в масив будь-яких змін. Необхідно написати програму, яка виконує завдання відповідно до варіанта, заданого в таблиці 8.1. Варіант завдання відповідає номеру студента в журналі групи (не підгрупи). Якщо номер за журналом перевищує кількість варіантів у таблиці, треба вчинити так, як у випадку лабораторних робіт минулого семестру.

Під час виконання завдання треба обов’язково враховувати, що масив може бути довільного розміру, зокрема й порожнім (містити нуль елементів). Програма має містити перевірки: якщо довжина заданого масиву не дає змогу розв’язати поставлену задачу, треба вивести на екран відповідне повідомлення. У звіті з лабораторної роботи навести 3–5 прикладів роботи програми для різних масивів. Обов’язково навести словесний опис та блок-схему алгоритму.

Таблиця 8.1 – Варіанти до завдання 1

Завдання 2

Завдання присвячене розв’язанню задач, що передбачають зміну вмісту масиву. Необхідно написати програму, яка розв’язує задачу відповідно до варіанта, заданого в таблиці 8.2. Варіант завдання відповідає номеру студента в журналі групи (не підгрупи). Якщо номер за журналом перевищує кількість варіантів у таблиці, треба вчинити так, як у випадку лабораторних робіт минулого семестру.

Під час виконання задачі треба обов’язково враховувати, що масив може бути довільного розміру, зокрема і порожнім (містити нуль елементів). Програма має містити перевірки: якщо довжина заданого масиву не дає змогу розв’язати поставлену задачу, треба вивести на екран відповідне повідомлення. У звіті з лабораторної роботи навести 3–5 прикладів роботи програми для різних масивів. Обов’язково навести словесний опис та блок-схему алгоритму.

Таблиця 8.2 – Варіанти до завдання 2

Вміст звіту з лабораторної роботи

1. Титульний аркуш.
2. Індивідуальний варіант завдання.
3. Опис, блок-схема, лістинг і результати роботи програми для завдання 1.
4. Опис, блок-схема, лістинг і результати роботи програми для завдання 2.
5. Висновки до лабораторної роботи.

Приклад виконання

Розглянемо простий приклад виконання завдання 2. Нехай у заданому одномірному масиві треба видалити усі послідовності нульових елементів, коротші від заданого числа *K*.

Перш ніж писати програму, нам треба зрозуміти, як виконувати завдання усно. Нехай у нас заданий такий масив:

```
mas = [0, 0, 3, 0, 0, -5, 2, 0, 1, 0, 7, 8, 9, 0, 0, 0, 10, 0]
```

У ньому можна бачити декілька ланцюжків нулів:

- від самого початку довжиною два елементи;
- після числа 3 довжиною два елементи;
- після числа 2 довжиною один елемент;

– після числа 1 довжиною один елемент;
– після числа 9 довжиною три елементи;
– у кінці масиву довжиною один елемент.

Задамо число $K = 3$. Воно означає, що з масиву треба видалити усі ланцюжки нулів, довжина яких менша за K. У нашому випадку це будуть усі ланцюжки, окрім передостаннього ланцюжка довжиною 3. Він повинен залишитись, а усі інші нульові елементи мають бути видалені. Внаслідок цього масив матиме такий вигляд:

```
[3, -5, 2, 1, 7, 8, 9, 0, 0, 0, 10]
```

Отже, відбудеться зменшення розміру масиву через видалення низки елементів.

Нижче наведений повний лістинг програми. Розберемо його докладно.

```
mas = [0, 0, 3, 0, 0, -5, 2, 0, 1, 0, 7, 8, 9, 0, 0, 0, 10, 0]

K = 3      # Задане число K

i = 0      # Початковий індекс масиву

flag = 0   # 1 - ми рухаємось по ланцюжку нулів, 0 - ні

while i < len(mas):
     if mas[i]==0:     # Якщо зустріли нульовий елемент
          if flag==0:         # Якщо ми знайшли початок ланцюжка нулів
               start_i = i          # Ланцюжок починається з індексу start_i
               flag = 1             # Ознака, що ми рухаємось по ланцюжку

     else:             # Якщо зустріли елемент, не рівний нулю
          if flag==1:         # Якщо ми рухались по ланцюжку нулів
               flag = 0             # Рух по ланцюжку нулів завершено
               end_i = i-1          # Ланцюжок закінчується індексом end_i
               l = end_i - start_i + 1  # Довжина ланцюжка
               print("Знайдено послідовність нулів довжиною", l, end=" ")
               print("від індексу", start_i, "до індексу", end_i)
               if l < K:     # Якщо довжина ланцюжка менша K, видаляємо ланцюжок
                    del(mas[start_i : end_i+1])      # Видаляємо ланцюжок
                    print("Ланцюжок [", start_i, ":", end_i, "] видалено", sep="")
                    i = i - l       # Зменшуємо індекс на довжину видаленого ланцюжка

     i = i + 1      # Продовжуємо цикл

if flag==1:
     end_i = i-1    # Ланцюжок закінчується індексом end_i
     l = end_i - start_i + 1  # Довжина ланцюжка
     print("Знайдено послідовність нулів довжиною", l, end=" ")
     print("від індексу", start_i, "до індексу", end_i)
     if l < K:      # Якщо довжина ланцюжка менша K, видаляємо ланцюжок
          del(mas[start_i : end_i+1])      # Видаляємо ланцюжок
          print("Ланцюжок [", start_i, ":", end_i, "] видалено", sep="")

print(mas)
```

Задамо масив у вигляді константи. Такий спосіб рекомендується тоді, коли нам треба протестувати працездатність алгоритму програми. Далі можна замінити завдання масиву константою якимось іншим способом,

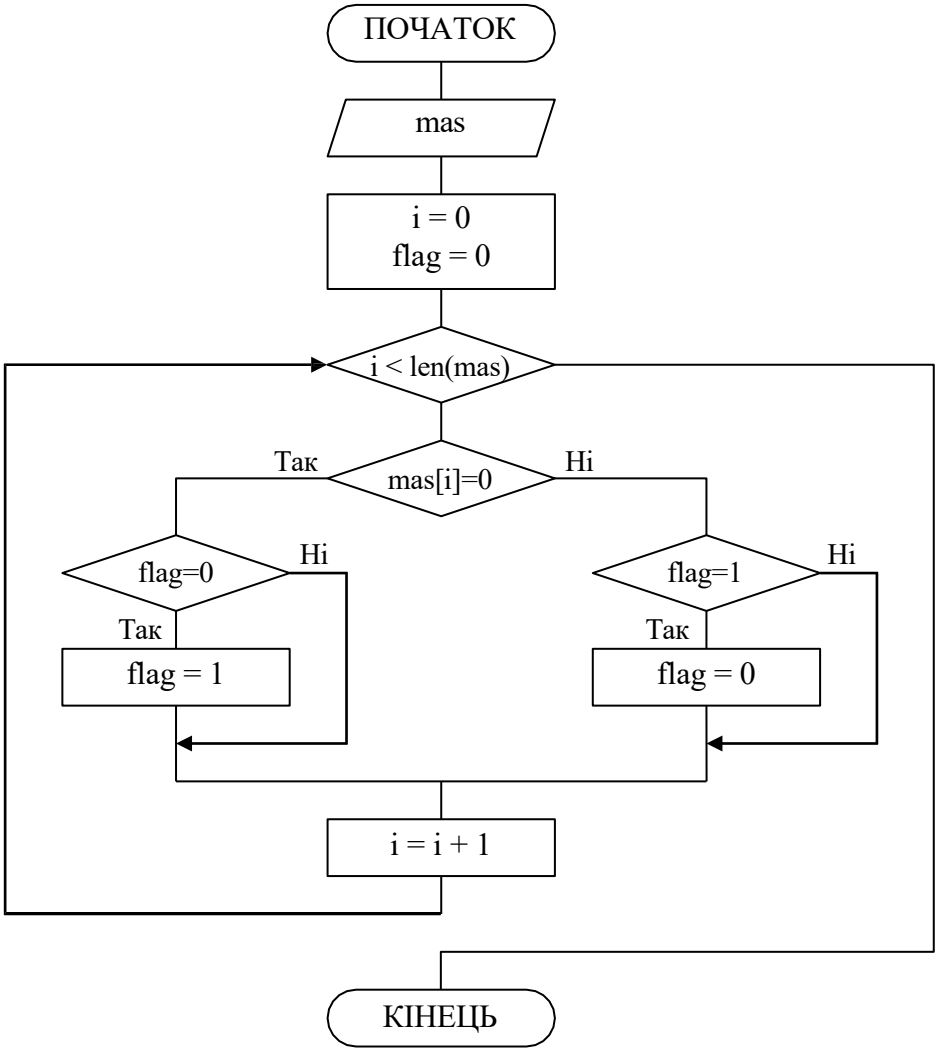
наприклад, читанням з файла, генерацією псевдовипадкових чисел, введенням з клавіатури тощо.

Підготуємось до обробки масиву. Задамо такі змінні:

```
K = 3       # Задане число K
i = 0       # Початковий індекс масиву
flag = 0    # 1 - ми рухаємось по ланцюжку нулів, 0 - ні
```

Тут змінна *flag* виступає в якості ознаки (прапорця) і показує, чи знайшли ми початок ланцюжка нулів і рухаємось по ланцюжку, чи ні. На початку роботи програми ця змінна дорівнює нулю. Як тільки буде знайдений нульовий елемент, змінна *flag* перемкнеться у значення 1 (прапорець буде піднято) і буде зберігати це значення, поки будуть зустрічатись нульові елементи (поки ми йдемо по ланцюжку нулів). Як тільки ми зустрінемо ненульовий елемент, змінна *flag* перемкнеться у значення 0 (прапорець буде скинуто) і залишатиметься такою до знаходження чергового нульового елементу (до початку нового ланцюжка нулів).

Цей процес можна описати за допомогою блок-схеми:



Цей алгоритм робить лише одне: вмикає прапорець, коли знайдений перший елемент нульового ланцюжка, і вимикає прапорець, коли знайдений перший ненульовий елемент після ланцюжка.

Програмний код, що відповідає цій блок-схемі, має такий вигляд:

```
mas = [0, 0, 3, 0, 0, -5, 2, 0, 1, 0, 7, 8, 9, 0, 0, 0, 10, 0]

K = 3
i = 0
flag = 0

while i < len(mas):
     if mas[i]==0:
          if flag==0:
               flag = 1
     else:
          if flag==1:
               flag = 0
     i = i + 1
```

Щоб було трохи веселіше, можна додати команди виведення на екран повідомлень про знаходження початку і кінця ланцюжка:

```
mas = [0, 0, 3, 0, 0, -5, 2, 0, 1, 0, 7, 8, 9, 0, 0, 0, 10, 0]

K = 3       # Задане число K
i = 0       # Початковий індекс масиву
flag = 0    # 1 - ми рухаємось по ланцюжку нулів, 0 - ні

while i < len(mas):
     if mas[i]==0:
          if flag==0:
               flag = 1
               print("[", i, "] - початок ланцюжка, ", sep="", end="")
     else:
          if flag==1:
               flag = 0
               print("[", i-1, "] - кінець ланцюжка\n", sep="")
     i = i + 1
```

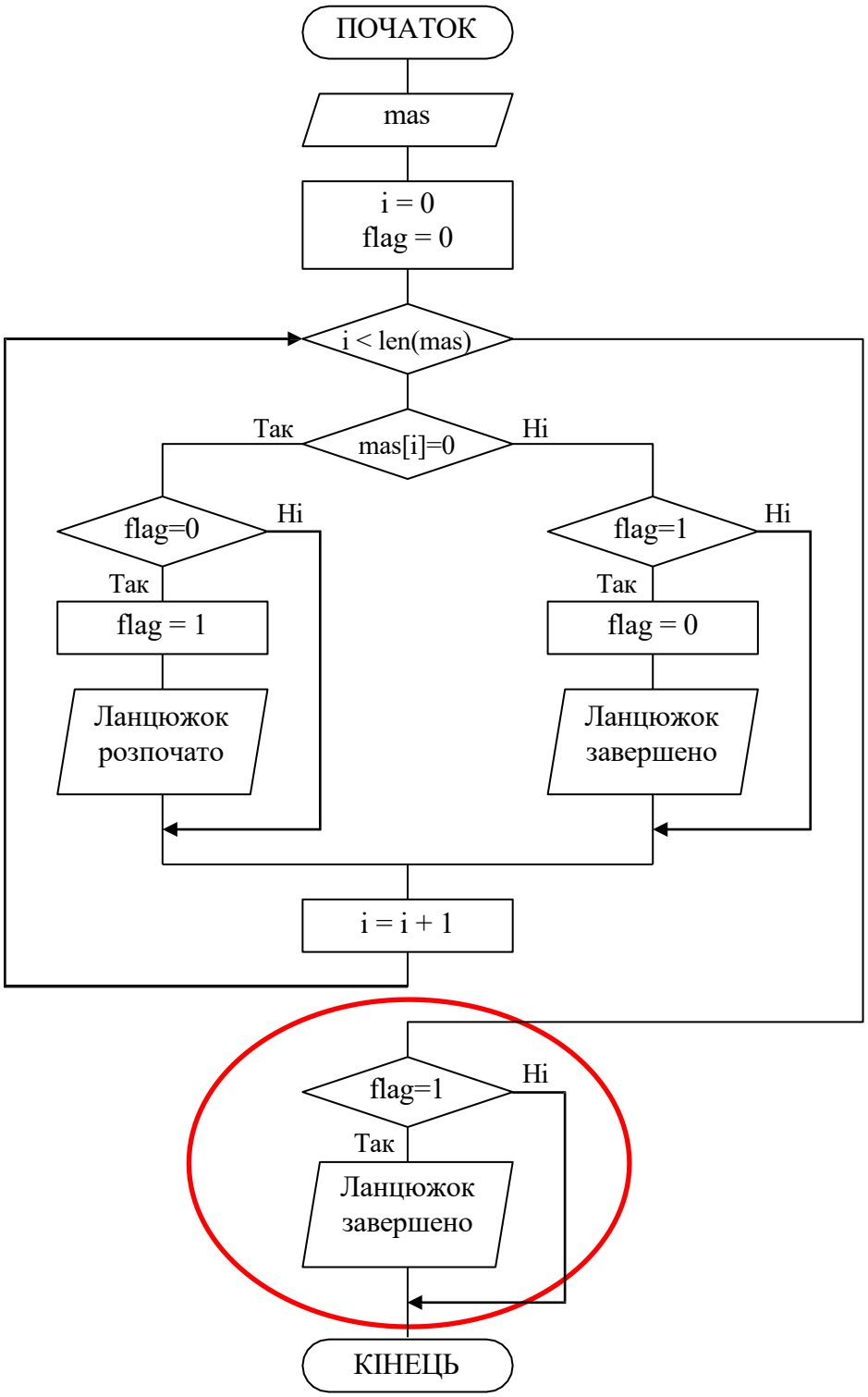
Результат роботи програми:

```
[0] - початок ланцюжка, [1] - кінець ланцюжка
[3] - початок ланцюжка, [4] - кінець ланцюжка
[7] - початок ланцюжка, [7] - кінець ланцюжка
[9] - початок ланцюжка, [9] - кінець ланцюжка
[13] - початок ланцюжка, [15] - кінець ланцюжка
[17] - початок ланцюжка,
```

Як можна помітити, для останнього ланцюжка кінцевий елемент не був знайдений. Чому так сталося? Якщо подивитись код програми, для «закриття» ланцюжка нам треба зустріти ненульовий елемент. Але якщо ланцюжок знаходиться в самому кінці списку, у нас не буде ненульового елементу, що стоїть після ланцюжка.

У цьому випадку ми повинні зробити таке. Після того, як цикл завершено, перевіримо: чи залишився у нас відкритий ланцюжок (чи дорівнює змінна *flag* одиниці). Якщо так, останнім елементом ланцюжка завжди буде останній елемент масиву, незалежно від довжини ланцюжка. Тому ми можемо вивести останній індекс масиву в якості кінця ланцюжка.

На блок-схемі це виглядатиме так:



Програмний код матиме такий вигляд:

```
mas = [0, 0, 3, 0, 0, -5, 2, 0, 1, 0, 7, 8, 9, 0, 0, 0, 10, 0]

K = 3       # Задане число K
i = 0       # Початковий індекс масиву
flag = 0    # 1 - ми рухаємось по ланцюжку нулів, 0 - ні

while i < len(mas):
     if mas[i]==0:
          if flag==0:
               flag = 1
               print("[", i, "] - початок ланцюжка, ", sep="", end="")
     else:
          if flag==1:
               flag = 0
               print("[", i-1, "] - кінець ланцюжка", sep="")
     i = i + 1

if flag==1:
     print("[", i-1, "] - кінець ланцюжка", sep="")
```

Результат:

```
[0] - початок ланцюжка, [1] - кінець ланцюжка
[3] - початок ланцюжка, [4] - кінець ланцюжка
[7] - початок ланцюжка, [7] - кінець ланцюжка
[9] - початок ланцюжка, [9] - кінець ланцюжка
[13] - початок ланцюжка, [15] - кінець ланцюжка
[17] - початок ланцюжка, [17] - кінець ланцюжка
```

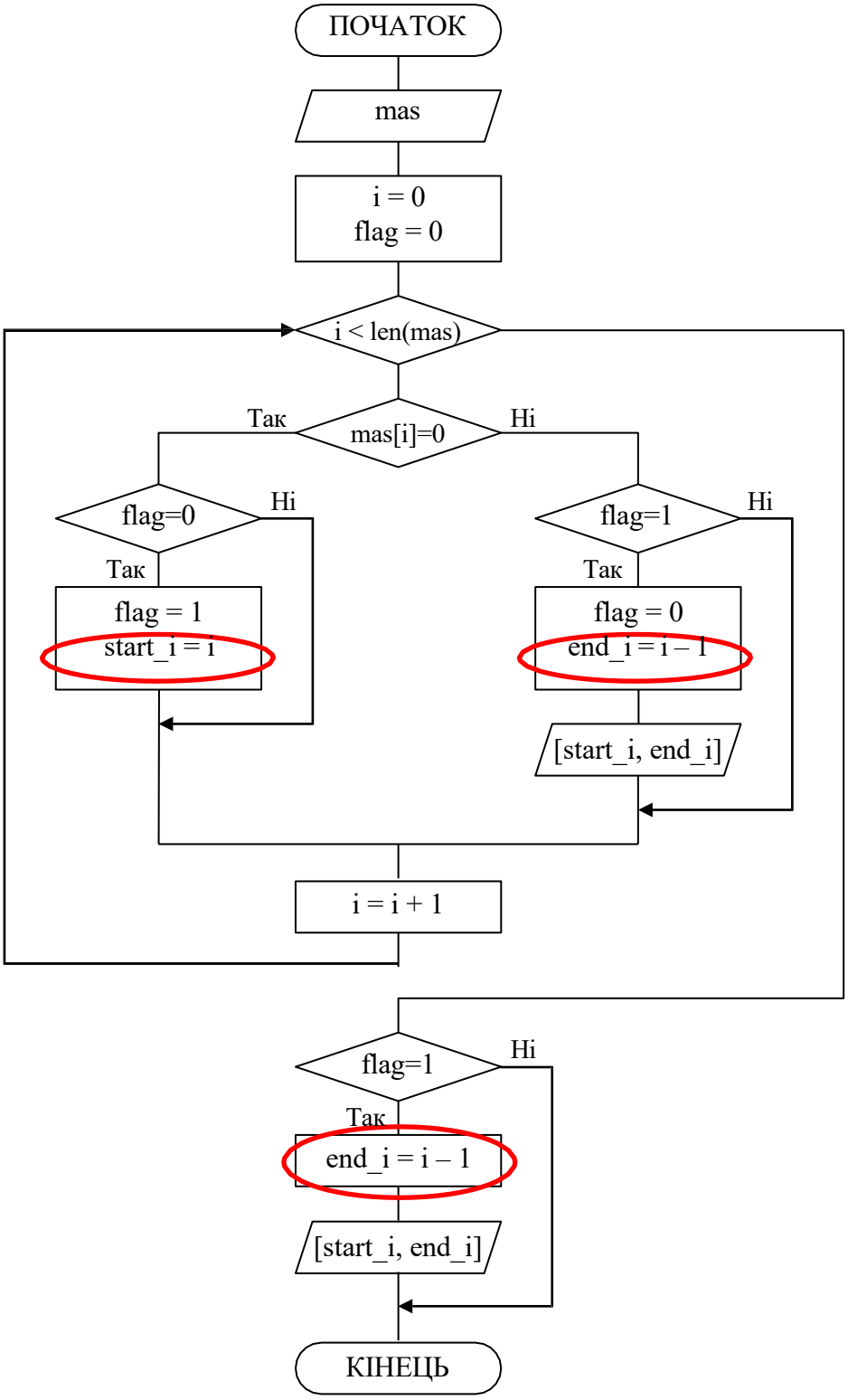
Тепер наша програма вміє знаходити початок і кінець кожного ланцюжка. Звернемо увагу на те, що під час виведення індексу, що вказує на останній елемент ланцюжка, ми використовуємо значення (i–1), а не значення i. Річ у тім, що коли ми дізнаємось, що ланцюжок завершився, змінна i вже дорівнює індексу елементу, наступного після ланцюжка. Це означає, що індекс останнього елементу ланцюжка буде на одиницю меншим від поточного значення i.

Так само і у випадку закінчення циклу. Коли змінна i стає рівною значенню $len(mas)$, цикл завершується. Але індекс останнього елементу масиву завжди дорівнює $len(mas)$ – 1 або ж (i–1). Тому індекс кінця ланцюжка, який знаходиться в самому кінці масиву, теж дорівнює (i–1).

У цій програмі ми виводимо інформацію про початок і кінець ланцюжків на екран. Команда *print*, яка виводить інформацію на екран, сама по собі не запам’ятовує індекси початку і кінця ланцюжка і «забуває» їх відразу після виведення. Щоб мати можливість пам’ятати початок і кінець ланцюжка, треба додати у програму дві змінні.

Нехай початок ланцюжка зберігається у змінній *start_i*, кінець ланцюжка – у змінній *end_i*. Змінна *start_i* приймає значення i тоді, коли ми

виводимо інформацію про початок ланцюжка. Змінна *end_i* приймає значення (*i*–1) тоді, коли ми виводимо інформацію про кінець ланцюжка. Це можна зобразити так:



Програма, що відповідає блок-схемі:

```
mas = [0, 0, 3, 0, 0, -5, 2, 0, 1, 0, 7, 8, 9, 0, 0, 0, 10, 0]

K = 3       # Задане число K
i = 0       # Початковий індекс масиву
flag = 0    # 1 - ми рухаємось по ланцюжку нулів, 0 - ні

while i < len(mas):
     if mas[i]==0:
          if flag==0:
               flag = 1
               start_i = i
     else:
          if flag==1:
               flag = 0
               end_i = i - 1
               print("Знайдено ланцюжок [", start_i, ":", end_i, "]", sep="")
     i = i + 1

if flag==1:
     end i = i - 1
     print("Знайдено ланцюжок [", start_i, ":", end_i, "]", sep="")
```

Результат:

```
Знайдено ланцюжок [0:1]
Знайдено ланцюжок [3:4]
Знайдено ланцюжок [7:7]
Знайдено ланцюжок [9:9]
Знайдено ланцюжок [13:15]
Знайдено ланцюжок [17:17]
```

Може виникнути питання: навіщо класти початок і кінець ланцюжків у змінні, якщо попередня програма і так непогано виводила результат? Річ у тім, що наша задача – не просто вивести на екран початок і кінець усіх нульових ланцюжків, а видалити їх з масиву. У найпростіший спосіб це можна зробити за допомогою зрізу командою:

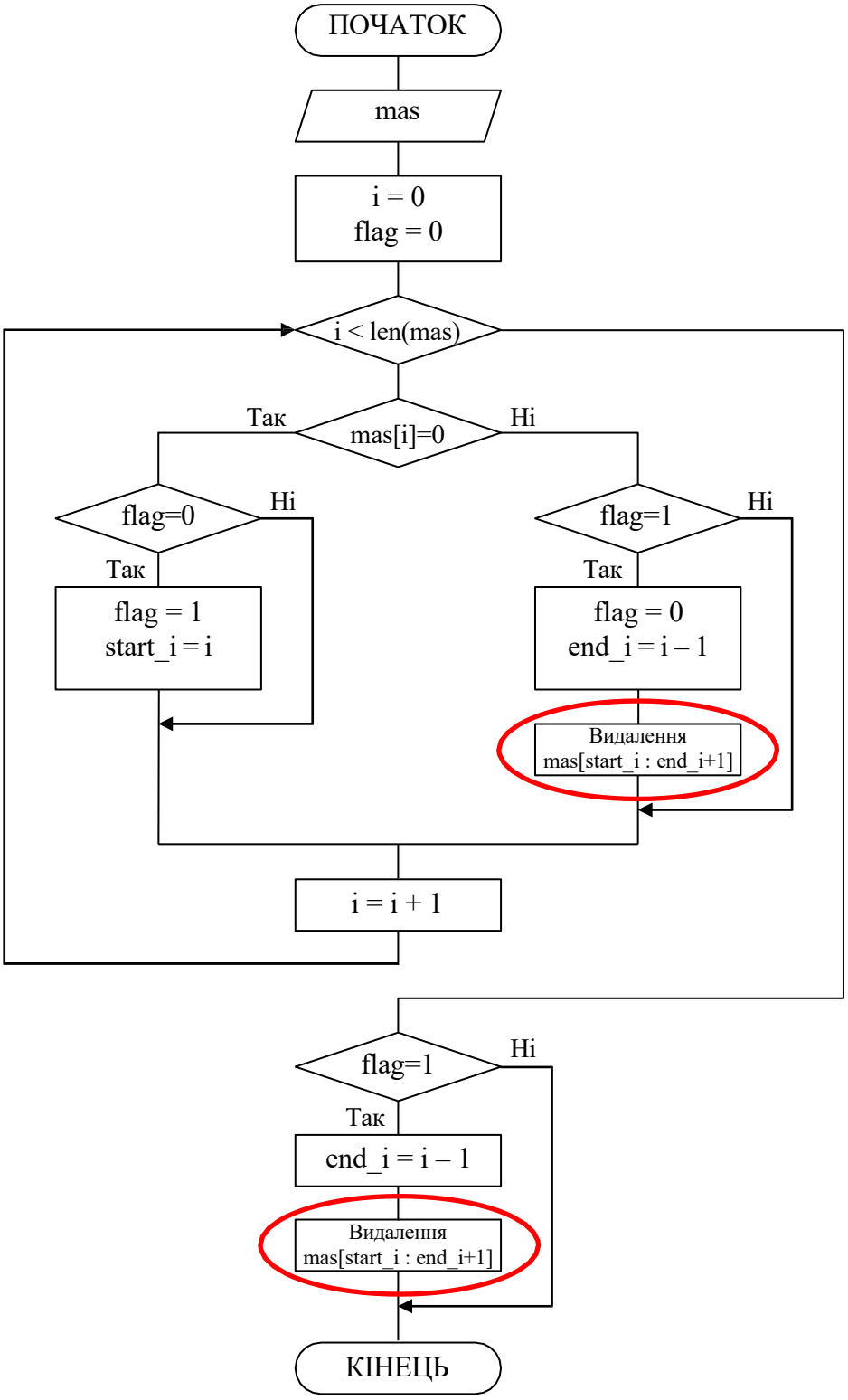
```
del(mas[a:b+1])
```

Тут *a* – індекс першого елементу ланцюжка, *b* – індекс останнього елементу ланцюжка. У нашому випадку індекс першого елементу ланцюжка зберігається у змінній start_i, індекс останнього елементу ланцюжка зберігається у змінній end_i. З урахуванням цього команда видалення ланцюжка матиме вигляд:

```
del(mas[start_i:end_i+1])
```

Саме для цієї команди нам потрібно пам’ятати початок і кінець ланцюжка в окремих змінних. Число «+1» потрібне для того, щоб були видалені елементи аж до індексу *ind_i* включно (так працює зріз у мові Python).

Разом із командою видалення ланцюжка блок-схема алгоритму матиме такий вигляд:



Ця блок-схема є майже готовою, за винятком однієї команди. Коли ми видаляємо шматок масиву, загальна кількість елементів зменшується на розмір цього шматка. Наприклад, якщо масив був таким:

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
2	–5	3	0	0	0	0	0	1	10

то після видалення ланцюжка довжиною 5 нулів він матиме такий вигляд:

[0]	[1]	[2]	[3]	[4]
2	–5	3	1	10

Тобто елемент, який був розташований відразу після ланцюжка і мав індекс [8], посунеться ближче до початку масиву і матиме індекс 3. До речі, посунуться усі елементи масиву (скільки б їх не було), розташовані після видаленого ланцюжка.

Сама по собі ця ситуація цілком нормальна. Проблема в тому, що після видалення цього ланцюжка ми повинні продовжити перевірку елементів масиву з того елементу, на якому ми зупинились до цього. Не з того ж індексу (він змінився), а з того самого елементу.

У початковому масиві елементом, розташованим після ланцюжка, є елемент зі значенням «1». Оскільки ми вже перевірили, що цей елемент не рівний нулю, нам він теж не цікавий. Його можна не перевіряти, а відразу перейти до наступного елементу. Тобто (у нашому випадку) розпочати перевірку не з індексу 3, а з індексу 4.

Як це зробити? Необхідно скорегувати змінну *i* в такий спосіб, щоб вона дорівнювала індексу, з якого ми продовжимо перевірку. Для цього треба підібрати потрібну нам формулу для зменшення змінної *i*, яка врахує довжину видаленого ланцюжка.

У нашому прикладі відразу після видалення у нас будуть такі значення змінних:

$i = 8$,

$start\_i = 3$,

$end\_i = 7$.

Нам треба, щоб було $i = 4$. Це можна зробити за допомогою наступного виразу:

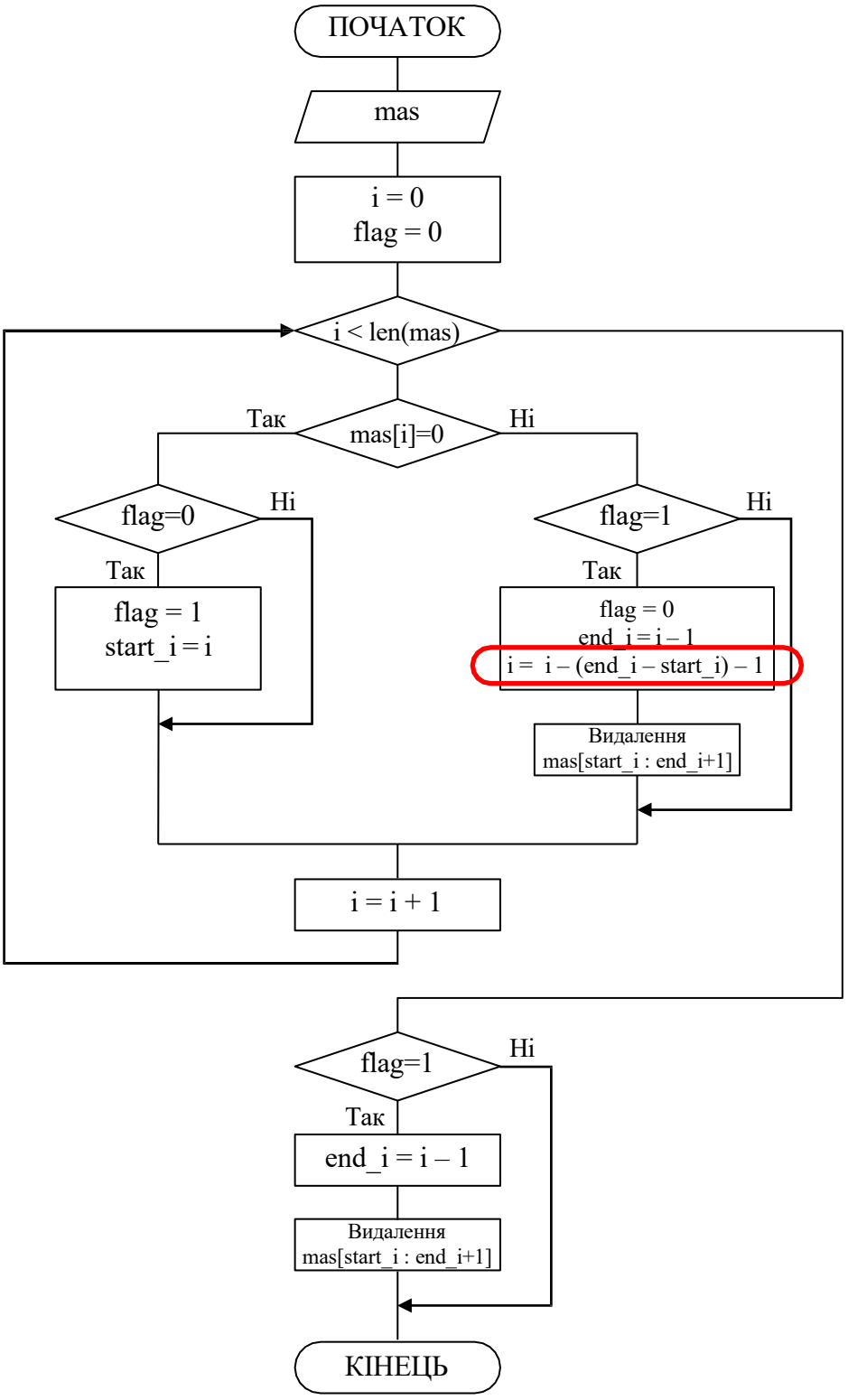
$$i = i - (end\_i - start\_i).$$

Дійсно, $8 - (7 - 3) = 4$. Але не будемо забувати, що в нашій блок-схемі в кінці тіла циклу розташована команда $i = i + 1$, яка обов'язково виконається і збільшить змінну *i* на одиницю. Щоб компенсувати це збільшення, додамо до нашої формули примусове віднімання одиниці:

$$i = i - (end\_i - start\_i) - 1.$$

Тепер за умови значень значень i=8, $start\_i$=3, end_i=7 значення змінної *i* дорівнюватиме 3, але воно відразу буде збільшене на одиницю командою $i = i + 1$, і в наступному проході циклу дорівнюватиме чотирьом.

Ось як це виглядає на блок-схемі алгоритму:



Нарешті наша програма працює добре. Вона коректно знаходить усі ланцюжки нулів, зокрема й розташовані з початку та з кінця масиву, і видаляє їх. Лістинг програми має такий вигляд:

```
mas = [0, 0, 3, 0, 0, -5, 2, 0, 1, 0, 7, 8, 9, 0, 0, 0, 10, 0]

K = 3
i = 0
flag = 0

while i < len(mas):
     if mas[i]==0:
          if flag==0:
               flag = 1
               start_i = i
     else:
          if flag==1:
               flag = 0
               end_i = i - 1
               del(mas[start_i:end_i+1])
               i = i - (end_i-start_i) - 1
     i = i + 1

if flag==1:
     end_i = i - 1
     del(mas[start_i:end_i+1])

print(mas)
```

Результат:

```
[3, -5, 2, 1, 7, 8, 9, 10]
```

Але все ж таки єдиний недолік у програми залишився. За завданням нам треба видаляти не усі ланцюжки нулів, а лише ті, довжина яких менша за значення *K*. У цій програмі це ніяк не реалізовано. Рішення дуже просте: перед командами видалення ланцюжка (їх у нас дві – всередині циклу та після циклу) треба додати перевірку: якщо довжина ланцюжка менша за *K*, то видаляти, інакше не видаляти. Ця перевірка є в першому лістингу програми.

Лабораторна робота № 10
Обробка двовимірних масивів

Метою лабораторної роботи є одержання студентами практичних навичок роботи з двовимірними масивами даних у мові Python.

Завдання до лабораторної роботи

Заданий двовимірний масив розміром *m* на *n* елементів, заповнений псевдовипадковими цілими числами в діапазоні [–1000; 1000]. Необхідно над цим масивом виконати два завдання відповідно до таблиць 10.1 і 10.2. Номер рядка таблиці визначається номером *N* у журналі групи (не підгрупи). Якщо значення *N* перевищує кількість рядків таблиці, треба обрати рядок з номером *N mod K*, де *K* – кількість рядків таблиці.

У звіті з лабораторної роботи треба для кожного завдання навести блок-схему алгоритму, лістинг програми (у вигляді тексту, не скриншот) та результати роботи програми для декількох прикладів (для різних розмірів масиву і різного вмісту масиву).

Таблиця 10.1 – Варіанти для першого завдання лабораторної роботи

Таблиця 10.2 – Варіанти для другого завдання лабораторної роботи

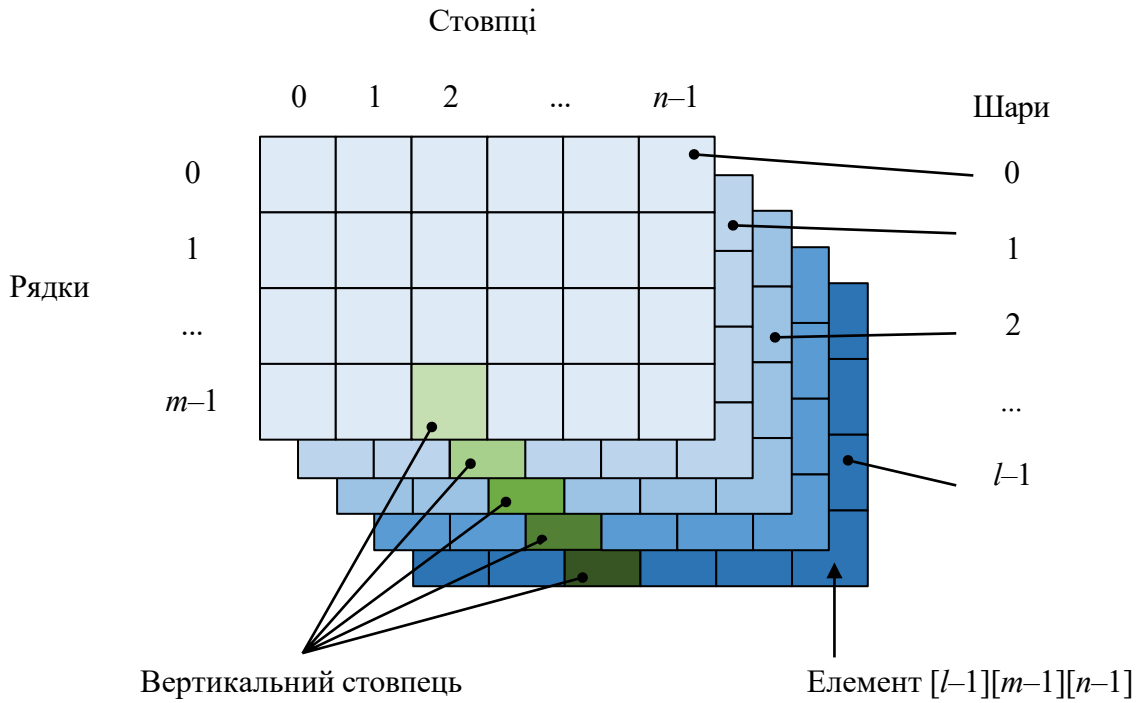
Лабораторна робота № 11
Обробка тривимірних масивів

Метою лабораторної роботи є одержання студентами практичних навичок роботи з тривимірними масивами даних у мові Python.

Завдання до лабораторної роботи

Інститут геологічних досліджень проводить пошук корисних копалин. Територія пошуку (об'єм ґрунту) має площу *m* * *n* сотень метрів, глибина пошуку – *l* сотень метрів. Попередні геологічні дослідження дали змогу розробити докладну карту вмісту усього обсягу ґрунту з точністю до сотні метрів та представити її у вигляді тривимірного масиву цілих чисел розміром *l* * *m* * *n* елементів. Перший вимір відповідає глибині (0 – поверхня, *l* – найбільша глибина), два останні виміри відповідають площині (*m* – кількість рядків, *n* – кількість стовпців).

Вертикальний стовпець масиву – це колона, що йде «в глибину» і складається з елементів, що мають однакові координати рядка і стовпця. У цій задачі не треба плутати стовпець масиву і вертикальний стовпець масиву. Стовпець – це не якась частина масиву, а просто третя координата його елементів. Вертикальний стовпець – це вже частина масиву: набір елементів, які розташовані вертикально один під іншим і мають спільні другу і третю координати (спільний рядок і стовпець масиву):



Кожен елемент масиву (тип ґрунту) може мати таке значення:

0 – корисні копалини відсутні, гірська порода м’яка;
1 – корисні копалини відсутні, гірська порода тверда;
2 – корисні копалини відсутні, гірська порода надтверда;
3 – вода;
4 – нафта;
5 – кам’яне вугілля;
6 – залізна руда;
7 – золото.

Видобуток корисних копалин дає прибуток. Прибуток від розробки одного елементу масиву, що містить нафту або кам’яне вугілля – однаковий і дорівнює одиниці. Прибуток від розробки залізної руди дорівнює двом. Прибуток від видобутку золота дорівнює десяти. Прибуток від видобутку води та гірських порід відсутній (дорівнює нулю).

Складність видобутку корисних копалин визначається кількістю елементів масиву, які треба пройти вертикально вниз (вздовж першого виміру масиву), щоб дістатись елементу, який містить потрібний ресурс. Заглиблення на один елемент, що містить м’які гірські породи або воду, коштує одну одиницю трудовитрат; тверді гірські породи – дві одиниці трудовитрат; будь-які корисні копалини – три одиниці трудовитрат; надтверді гірські породи – п’ять одиниць трудовитрат. Вважається, що одна одиниця прибутку покриває одну одиницю трудовитрат.

Необхідно провести аналіз території пошуку корисних копалин: відповідно до номера N студента в журналі групи розробити програму, що виконує завдання відповідно до таблиці 11.1. Якщо номер студента перевищує кількість рядків таблиці, треба вчинити так, як у попередніх лабораторних роботах.

Розміри масиву m, n, k вводяться з клавіатури або задаються константами на початку програми. Вміст масиву заповнюється за допомогою генератора псевдовипадкових чисел. Бажано зробити генерацію так, щоб якісь типи ґрунту зустрічались частіше (наприклад, м’які гірські породи), якісь – рідше (наприклад, золото).

Таблиця 11.1 – Індивідуальні варіанти завдань

Приклад виконання лабораторної роботи

Виконаємо завдання пошуку середньої кількості золота у вертикальних стовпцях ґрунту, що не містять води.

Спочатку створимо масив і заповнимо його псевдовипадковими числами в діапазоні від 0 до 7. Заповнювати будемо так, щоб золота було трохи більше, ніж усіх інших типів ґрунту. Для цього будемо генерувати псевдовипадкові числа не в діапазоні від 0 до 7, а в діапазоні від 0 до 9. Якщо згенероване число дорівнює від 0 до 6, ми його просто записуємо в масив – це буде один з типів ґрунту, окрім золота. Якщо згенероване число дорівнює від 7 до 9, ми будемо записувати в масив число 7, що відповідає золоту. Оскільки генератор псевдовипадкових чисел видає числа в діапазоні [0; 9] приблизно рівномірно, золото буде з'являтись у трьох випадках із десяти (коли генеруються числа 7, 8 або 9), а усі інші типи ґрунту – в одному випадку з десяти, тобто втричі рідше.

```
import random

l = 5          # Кількість шарів ґрунту
m = 3          # Кількість рядків
n = 2          # Кількість стовпців

mas = []                          # Порожня заготовка для масиву
for k in range(l):                # Цикл по виміру 0 (по шарах
ґрунту)

    mas1 = []
    for i in range(m):            # Цикл по виміру 1 (по рядках)
```

```
        mas2 = []
        for j in range(n):    # Цикл по виміру 2 (по стовпцях)
            v = random.randint(0, 9)
            if v<=6:
                mas2.append(v)
            else:
                mas2.append(7)

        mas1.append(mas2)     # Додаємо новий рядок

    mas.append(mas1)          # Додаємо новий шар ґрунту
```

Для перевірки виконаємо пошарове виведення масиву на екран:

```
for k in range(l):          # Цикл по шарах
    print("Шар", k, ":")
    for i in range(m):      # Цикл по рядках
        print(mas[k][i])    # Виведення i-го рядка k-го шару
    print()                 # Пустий рядок між шарами
```

Результат:

```
Шар 0 :
[0, 3]
[2, 1]
[7, 2]

Шар 1 :
[1, 5]
[2, 7]
[1, 7]

Шар 2 :
[7, 7]
[7, 1]
[7, 1]

Шар 3 :
[6, 1]
[7, 5]
[7, 7]

Шар 4 :
[7, 7]
[7, 6]
[5, 1]
```

Тепер розробимо алгоритм обробки масиву згідно з нашим завданням. Розіб’ємо усю роботу на кроки:

1) організація циклу, що перебирає усі вертикальні стовпці масиву;

2) пошук кількості золота в окремому вертикальному стовпці ґрунту з одночасним пошуком води. Підрахунок суми золота і кількості тих вертикальних стовпців, що не містять води;
3) розрахунок середнього арифметичного золота для вертикальних стовпців, що не містять води.

1. Організуємо цикл, що перебирає усі вертикальні стовпці масиву.

```
for i in range(m):              # Цикл по рядках
    for j in range(n):          # Цикл по стовпцях
        ТІЛО ЦИКЛУ
```

Цей фрагмент коду містить два вкладені цикли. Зовнішній цикл використовує змінну *i* та застосовується для перебору рядків. Внутрішній цикл використовує змінну *j* та застосовується для перебору стовпців. Для кожної пари чисел <*i*, *j*> маємо вертикальний стовпець, який будемо аналізувати в тілі внутрішнього циклу.

2. Тіло циклу матиме такий вигляд:

```
stovb = 0    # Кількість шуканих вертикальних стовпців
summa = 0    # Загальна кількість золота у шуканих верт. стовпцях

for i in range(m):              # Цикл по рядках
    for j in range(n):          # Цикл по стовпцях
        water = 0  # 0 - воду не знайдено, 1 - знайдено
        gold = 0   # Кількість золота у вертикальному стовпці

        for k in range(l):          # Цикл по шарах ґрунту
            if mas[k][i][j] == 3: # Якщо знайдено воду
                water = 1               # Ознака, що воду знайдено
                break                   # Перериваємо цикл
            if mas[k][i][j] == 7: # Якщо знайдено золото
                gold += 1               # Збільшуємо кількість золота

        if water == 0:        # Якщо воду не знайдено
            stovb += 1        # Збільшуємо кількість верт. стовпців
```

Перед початком головних циклів створюються змінні *stovb* та *summa*, які зберігатимуть кількість знайдених вертикальних стовпців без води та загальну суму золота в цих стовпцях. Перед початком циклів обидві ці змінні ініціюються нулями.

Всередині внутрішнього циклу «*for j*» ми готуємось до аналізу чергового вертикального стовпця. Спочатку ми встановлюємо змінну *water* у значення 0. Якщо в процесі аналізу стовпця в одній з його комірок виявиться вода, у змінну *water* буде записана одиниця, тобто ознака того, що в цьому стовпці є вода. Також ми обнулюємо змінну *gold*, у якій будемо накопичувати кількість «золотих» елементів стовпця.

Далі йде цикл перебору елементів вертикального стовпця. В якості змінної циклу використовується змінна *k*, яка в циклі приймає значення від 0 до (*l*–1).

У тілі циклу розташовані дві перевірки (два оператори *if*):

Перший оператор *if* перевіряє, чи є поточний елемент стовпця (елемент з координатами [*k*][*i*][*j*]) водою. Якщо так, то у змінну *water* записується значення 1, і цикл «*for k*» переривається, оскільки ми з'ясували, що поточний вертикальний стовпець містить воду, нам він не цікавий і перевіряти його до кінця немає сенсу). Таке переривання дає змогу економити час роботи програми.

Якщо переривання не відбулося, виконується другий оператор *if*. Він перевіряє, чи дорівнює поточний елемент значенню 7. Якщо так, то це золото, і ми додаємо до змінної *gold* одиницю. Після перегляду вертикального стовпця змінна *gold* дорівнюватиме кількості елементів стовпця, рівних 7.

Далі тіло циклу «*for k*» завершується, оскільки наступні команди записані у програмі з меншим відступом. Програма буде повторювати цикл «*for k*» доти, поки він не буде повністю завершений або перерваний оператором *break*.

Після циклу «*for k*» іде аналіз змінної *water*. Якщо вона виявилась рівною нулю, в поточному вертикальному стовпці воду не було знайдено. Тож ми можемо розглядати цей стовпець як один із тих, що ми шукали. Збільшуємо кількість *stovp* знайдених стовпців, а також додаємо до загальної кількості золота *suma* кількість золота в поточному вертикальному стовпці (змінну *gold*).

3. Виведемо на екран основні результати роботи програми:

```
print("Вертикальних стовпців без води:", stovp)
print("Загальна кількість золота:", suma)
seredne = suma / stovp
print("В середньому в одному вертикальному стовпці", seredne,
"золота.")
```

Змінна *seredne* (середнє арифметичне) розраховується лише для тих вертикальних стовпців, у яких не було води. Навіть якщо такий стовпець не містив жодної комірки з золотом, він враховується під час цього розрахунку.

Загальний лістинг програми має такий вигляд:

```
import random

l = 5          # Кількість шарів ґрунту
m = 3          # Кількість рядків
n = 2          # Кількість стовпців

mas = []
for k in range(l):             # Цикл по нульовому виміру (по шарах ґрунту)

    mas1 = []
    for i in range(m):         # Цикл по першому виміру (по рядках)

        mas2 = []
        for j in range(n):     # Цикл по другому виміру (по стовпцях)
            v = random.randint(0, 9)
            if v<=6:
                mas2.append(v)
            else:
                mas2.append(7)

        mas1.append(mas2)      # Додаємо новий рядок

    mas.append(mas1)           # Додаємо новий шар ґрунту


for k in range(l):
    print("Шар", k, ":")
    for i in range(m):
        print(mas[k][i])
    print()


stovp = 0    # Кількість шуканих вертикальних стовпців
suma = 0    # Загальна кількість золота у шуканих верт. стовпцях

for i in range(m):             # Цикл по рядках
    for j in range(n):         # Цикл по стовпцях

        water = 0  # 0 - воду не знайдено, 1 - знайдено
        gold = 0   # Кількість золота у вертикальному стовпці

        for k in range(l):         # Цикл по шарах ґрунту
            if mas[k][i][j] == 3: # Якщо знайдено воду
                water = 1              # Ознака, що воду знайдено
                break                  # Перериваємо цикл
            if mas[k][i][j] == 7: # Якщо знайдено золото
                gold += 1              # Збільшуємо кількість золота

        if water == 0:       # Якщо воду не знайдено
            stovp += 1       # Збільшуємо кількість верт. стовпців
            suma += gold    # Збільшуємо загальну кількість золота

print("Вертикальних стовпців без води:", stovp)
print("Загальна кількість золота:", suma)
seredne = suma / stovp
print("В середньому в одному вертикальному стовпці", seredne, "золота.")
```

Результат роботи програми має такий вигляд:

```
Шар 0 :
[2, 7]
[6, 7]
[1, 4]

Шар 1 :
[3, 5]
[6, 2]
[7, 5]

Шар 2 :
[7, 7]
[1, 7]
[7, 4]

Шар 3 :
[6, 7]
[7, 7]
[7, 6]

Шар 4 :
[1, 0]
[7, 7]
[0, 1]

Вертикальних стовпців без води: 5
Загальна кількість золота: 12
В середньому в одному вертикальному стовпці 2.4 золота.
```

Цей приклад дає змогу зробити висновок, що програма працює правильно і вирішує поставлене завдання.

Лабораторна робота № 12
Робота з текстовими файлами

Метою лабораторної роботи є одержання студентами практичних навичок роботи з текстовими файлами у мові Python.

Завдання до лабораторної роботи

Варіантом завдання є завдання з лабораторної роботи № 10, що присвячена обробці двовимірних масивів.

Необхідно модифікувати програму з лабораторної роботи № 10 згідно з такими вимогами:

1) розміри та вміст масиву повинні прочитуватись із текстового файла;
2) результати роботи програми повинні записуватись у текстовий файл;
3) алгоритм обробки масиву повинен залишатись незмінним;
4) на екран нічого виводитись не повинно.

Усі вхідні дані повинні бути розташовані в одному файлі. Програма обов'язково повинна перевіряти правильність прочитаних даних і виводити повідомлення у разі виявлення помилкових значень. Наприклад, розміри масиву обов'язково повинні бути цілими числами, більшими за нуль. Елементи масиву повинні бути цілими числами в діапазоні від –1000 до 1000 (згідно з завданням до лабораторної роботи № 10). Кількість елементів масиву в файлі повинна бути такою, щоб відповідати розмірам масиву. Якщо умови задачі передбачають використання додаткових чисел (наприклад, діапазону чисел [A; B], ці числа також мають читатись із файла.

Усі вихідні дані, зокрема й інформаційні повідомлення про помилки, повинні послідовно записуватись в один вихідний файл. Необхідно, щоб окрім числових результатів, у файл виводились також відповідні текстові коментарі. Наприклад: «Середнє арифметичне елементів масиву: 125». Загалом вміст файла має бути таким, як вміст екрана після виконання лабораторної роботи № 10.

У звіті з лабораторної роботи треба навести варіанти завдань з лабораторної роботи № 10. Наводити блок-схеми не потрібно. Обов'язково мають бути 3–4 приклади роботи програми (вміст вхідного і вихідного файлів). Вміст цих файлів необхідно показувати не текстом, а скриншотом вікна з відкритим текстовим файлом (наприклад, у програмі «Блокнот»). Також обов'язково має бути повний лістинг програми у форматі тексту (не картинки).

Лабораторна робота № 13

Базові елементи бібліотеки tkinter

Метою лабораторної роботи є ознайомлення з технологією побудови візуальних користувацьких інтерфейсів за допомогою бібліотеки *tkinter* мовою програмування Python.

Індивідуальним варіантом завдання є одна із професій, що наведені в табл. 13.1. Для заданої професії треба розробити програму з графічним інтерфейсом, яка повинна допомагати спеціалісту в цій галузі під час розв'язання його задач. Номер рядка таблиці відповідає номеру студента в журналі групи.

В окремих випадках допускається узгодження з викладачем варіанта завдання, який відсутній в цій таблиці. Наприклад, якщо у студента варіант «Фізик», але студент дуже добре розуміється в хімії, він може домовитись з викладачем використовувати варіант «Хімік», якого немає в табл. 1. Але не можна просто змінювати свій варіант на інший з табл. 13.1. Тобто не можна змінити, наприклад, варіант «Фізик» на варіант «Водій».

Таблиця 13.1 – Індивідуальні варіанти завдань

№	Професія	№	Професія
1	Фермер	11	Логіст
2	Менеджер банку	12	Збирач меблів
3	Викладач	13	Кіберспортсмен
4	Лікар	14	Фармацевт
5	Ріелтор	15	Економіст
6	Фізик	16	Касир
7	Кухар	17	Водій
8	Тренер	18	Перекладач
9	Мережевий адміністратор	19	Автомеханік
10	Електрик	20	Моряк

Згідно з індивідуальним варіантом завдання треба зробити таке:

1. Проаналізувати та сформулювати задачі, з якими стикається людина заданої спеціальності. Задачі зазвичай мають бути обчислювального або консультативного характеру.

2. Розробити програму мовою Python із використанням бібліотеки *tkinter*, яка лає змогу розв'язувати задачі, сформульовані в п. 1. Програма має відповідати вимогам, наведеним нижче.

3. Підготувати та захистити звіт з лабораторної роботи у встановлені терміни.

Вимоги до розроблювальної програми

1. Програма має розроблятися за допомогою саме бібліотеки *tkinter*. Не можна використовувати інші подібні бібліотеки.

2. У програмі мають обов'язково використовуватись такі віджети:

– кнопка (Button);

– радіокнопка (Radiobutton);

– чекбокс (Checkbutton);

– мітка (Label).

Інші віджети в цій лабораторній роботі використовувати не можна. Їх використання – це наступна лабораторна робота.

3. Віджети у межах вікна мають бути розміщені за допомогою методу-пакувальника *pack*.

4. Вікно програми та віджети повинні мати кольорову гаму, що асоціюються з предметною областю (професією).

5. У програмі не повинно бути «некорисних» віджетів, які нічого не роблять або дублюють своїми функціями інші віджети.

6. У вікні програми не повинно бути вільного місця, що не використовується віджетами.

Вміст звіту з лабораторної роботи

1. Титульний лист, оформлений за вимогами університету.

2. Номер студента в журналі групи та індивідуальний варіант завдання.

3. Опис задач, що вирішуються програмою, та «ручні» приклади їх розв'язання.

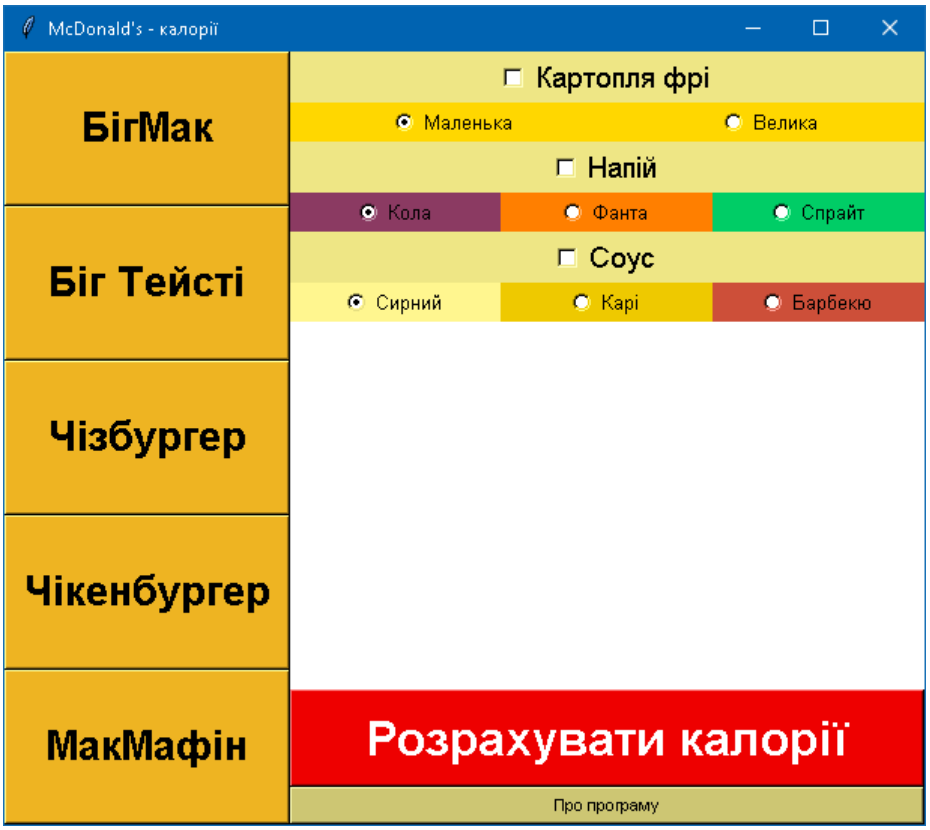
4. Докладний опис лістингу програми з наведенням знімків екрана (приклад наведений нижче).

5. Висновки.

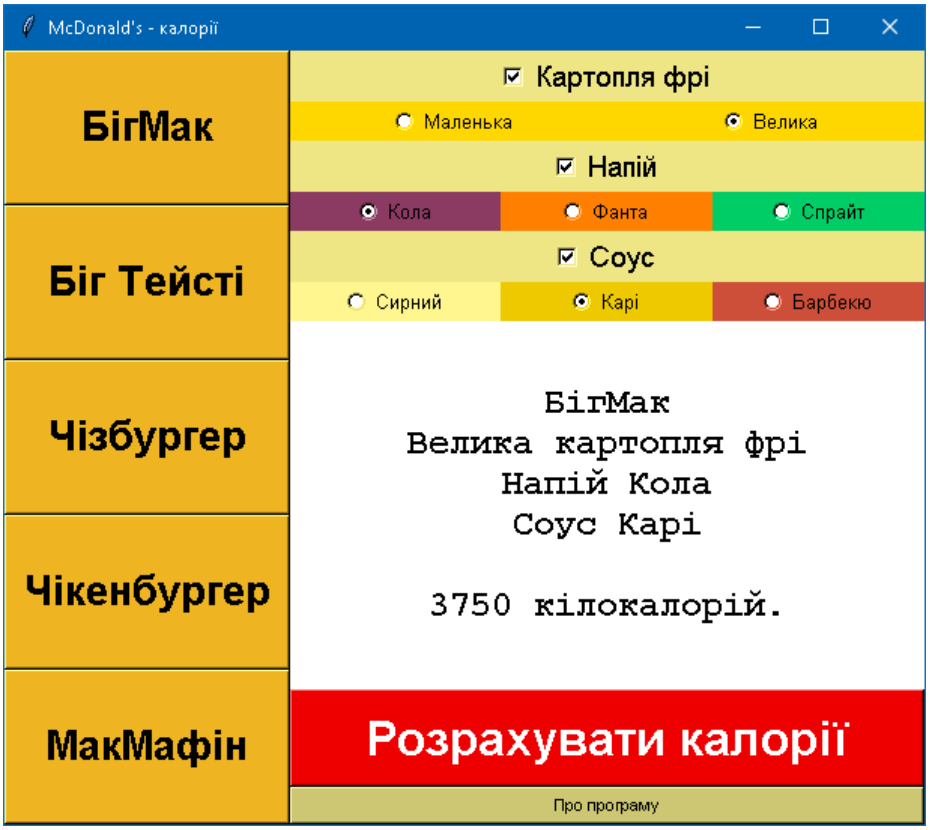
Приклад опису розробленої програми

Нехай індивідуальним варіантом завдання є професія дієтолога. Як відомо, їжа в мережі МакДональдс є доволі шкідливою з погляду дієтології через велику кількість споживаних калорій. Метою програми є розрахунок калорій у різних «фастфудах» компанії МакДональдс. У цій програмі усі розрахунки є умовними і не відповідають дійсності, але в програмах, що розроблять студенти, усі дані і результати мають бути якомога ближчими до реальних.

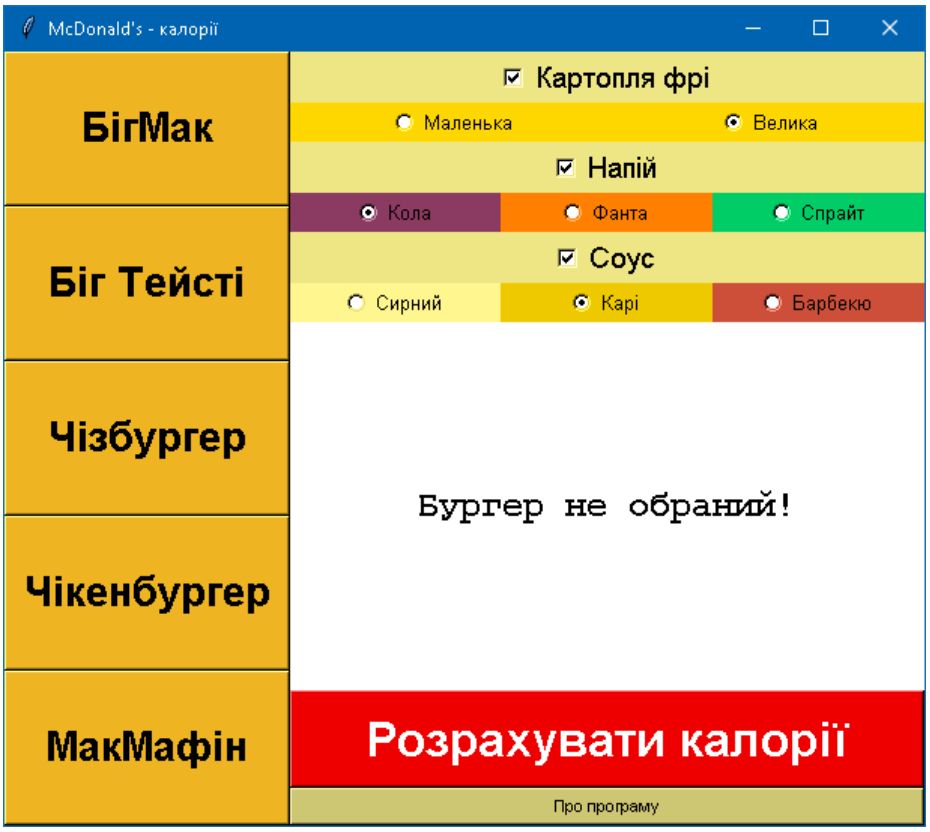
Після запуску вікно програми має такий вигляд:



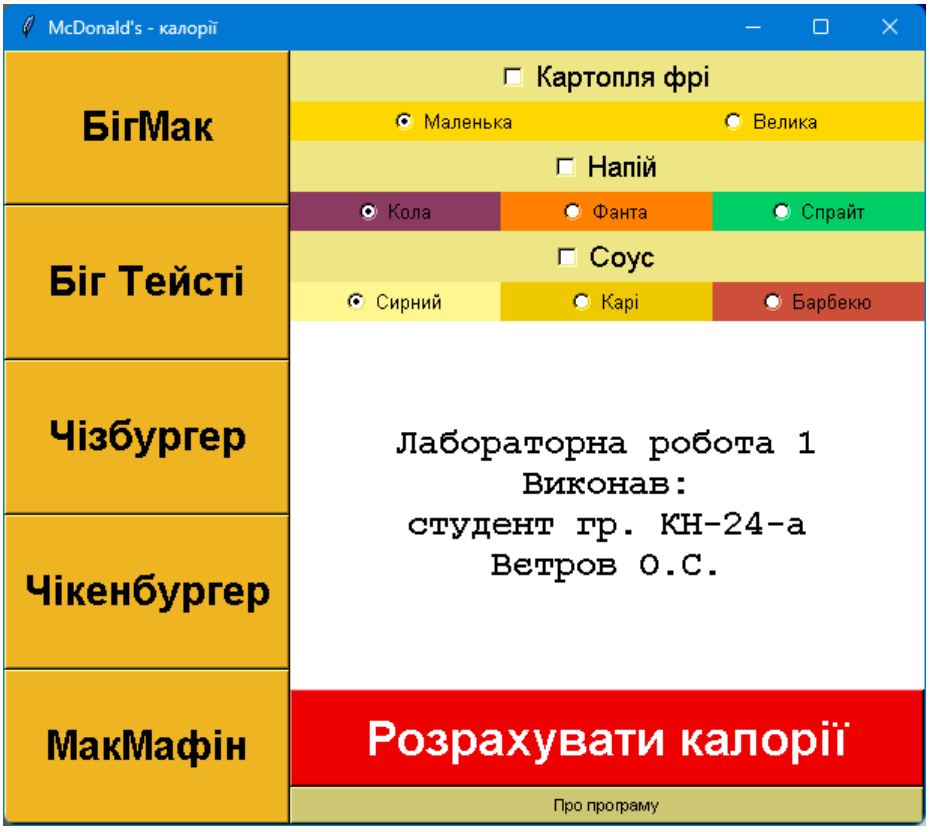
Якщо натиснути одну з кнопок із назвою бургера (в лівій частині вікна), після чого обрати додаткові страви (у верхній частині вікна) і натиснути кнопку «Розрахувати калорії», то в білому інформаційному полі буде виведена така інформація:



Якщо натиснути кнопку «Розрахувати калорії» до того, як буде натиснута будь-яка кнопка з назвою бургера, результат буде таким:



У разі натискання кнопки «Про програму» видається коротка інформація про автора програми:



Розглянемо основні частини програми.

1. Підготовчі рядки

На початку програми розташовані такі чотири рядки:

```
# coding=1251
from tkinter import *
root = Tk()
root.title("McDonald's - калорії")
```

Тут перший рядок підказує інтерпретатору Python, у якій кодовій таблиці розглядати символи лістингу програми. Якщо програму запускати з-під оболонки IDLE, цей рядок не потрібен. У разі самостійного запуску програми в системі Windows цей рядок є необхідним.

2. Завдання глобальних змінних

Під час розрахунку кількості калорій у програмі використовується ряд констант. Дані константи містять значення калорій окремих продуктів. Хоча в цьому прикладі значення констант є вигаданими, заміна їх значень на реальні дасть правильний результат роботи програми.

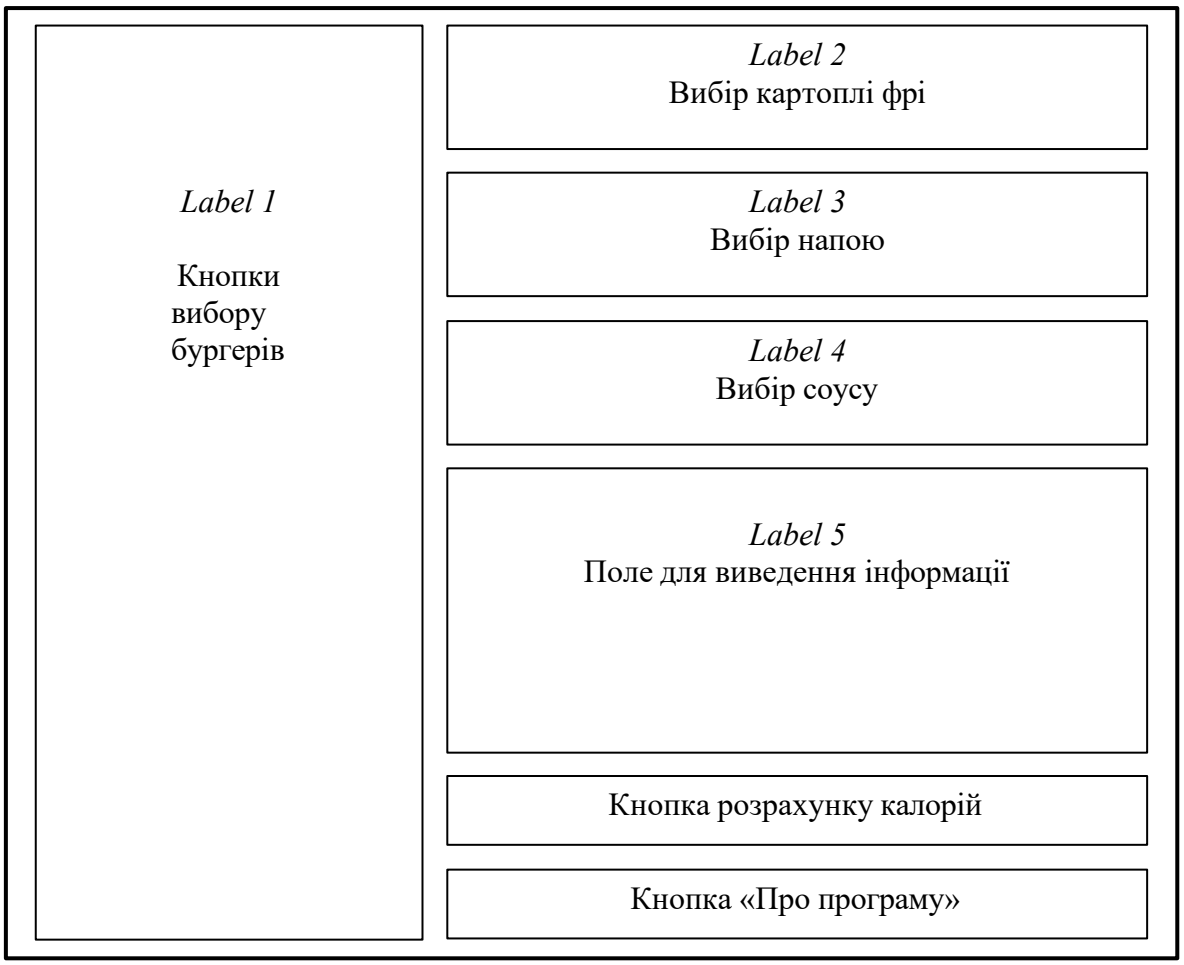
Всі константи оформлені у вигляді глобальних змінних. Це надає до них доступ з будь-якої точки програми.

```
# Глобальні змінні

burger = 0 # Тип бургера: 0 - не обраний, 1 – Біг Мак,
           # 2 - Біг Тейсті, 3 - Чизбургер,
           # 4 - Чікенбургер, 5 - МакМафін.

# Калорійність бургерів (по порядку)
k1, k2, k3, k4, k5 = 1200, 1700, 2300, 1400, 2800

# Калорийність картоплі фрі
p1, p2 = 700, 1400

# Калорійність напоїв
d1, d2, d3 = 1000, 1100, 800

# Калорійність соусів
s1, s2, s3 = 120, 150, 200
```

3. Заповнення вікна програми мітками

Як відомо, бібліотека *tkinter* дає змогу розміщувати всередині віджетів-міток (*Label*) інші віджети. Тому зручно спочатку розмістити у вікні програми мітки, всередині яких потім розмістити кнопки, радіокнопки, чекбокси та інші віджети.

Розташуємо мітки у вікні програми так, як показано на рисунку нижче. В нижній частині вікна, після мітки № 5, розташовані кнопки розрахунку калорій та «Про програму».



У коді програми мітки 1–5 задані змінними *l1–l5*:

```
# Мітки
l1 = Label(root, text="Label 1", bg="khaki3", bd=0)
l2 = Label(root, text="Label 2", bg="khaki3", bd=0)
l3 = Label(root, text="Label 3", bg="khaki3", bd=0)
l4 = Label(root, text="Label 4", bg="khaki3", bd=0)

ryadok = StringVar()
ryadok.set("")
l5 = Label(root, textvariable=ryadok, bg="white", bd=0,
           font="Courier 18 bold", height=9, width=30)
```

Для мітки *l5*, що використовується для виведення інформації, створена змінна *ryadok*, зв’язана з міткою через параметр *textvariable*. Будь-яка зміна вмісту змінної *ryadok* буде приводити до миттєвої зміни напису на мітці.

Щоб розмістити мітки у вікні програми так, як показано на рисунку вище, використовуються наступні виклики функції-пакувальника *pack*:

```
l1.pack(side=LEFT, expand=1, fill=BOTH)
l2.pack(side=TOP, expand=1, fill=BOTH)
l3.pack(side=TOP, expand=1, fill=BOTH)
l4.pack(side=TOP, expand=1, fill=BOTH)
l5.pack(side=TOP, expand=1, fill=BOTH)
```

4. Кнопки вибору бургерів

Під час створення кнопок використано одну особливість: кнопка вибору бургерів прив'язується не до змінної *root* (тобто не до головного вікна програми), а до змінної *l1*, яка відповідає за мітку *l1*. В результаті кнопки бургерів будуть розташовані всередині мітки *l1*, тобто у правій частині вікна програми.

```
# Кнопки з назвами бурегрів (всередині мітки l1)
b1 = Button(l1, text="Біг Мак", font="Arial 20 bold",
            bg="Goldenrod2", activebackground="Goldenrod1")
b2 = Button(l1, text="Біг Тейсті", font="Arial 20 bold",
            bg="Goldenrod2", activebackground="Goldenrod1")
b3 = Button(l1, text="Чизбургер", font="Arial 20 bold",
            bg="Goldenrod2", activebackground="Goldenrod1")
b4 = Button(l1, text="Чікенбургер", font="Arial 20 bold",
            bg="Goldenrod2", activebackground="Goldenrod1")
b5 = Button(l1, text="МакМафін", font="Arial 20 bold",
            bg="Goldenrod2", activebackground="Goldenrod1")
```

У процесі пакування віджети кнопок розташовуються всередині мітки *l1* зверху донизу:

```
b1.pack(side=TOP, expand=1, fill=BOTH)
b2.pack(side=TOP, expand=1, fill=BOTH)
b3.pack(side=TOP, expand=1, fill=BOTH)
b4.pack(side=TOP, expand=1, fill=BOTH)
b5.pack(side=TOP, expand=1, fill=BOTH)
```

5. Прив'язка кнопок вибору бургерів до подій

У пункті 2 було заведено глобальну змінну *burger*, значення якої має визначати тип обраного бургера. Під час натискання однієї з кнопок *b1–b5* відбувається виклик функції, в якій змінна *burger* приймає потрібне значення:

```
# Функциї вибору бургерів
def burger1(event):
    global burger
    burger = 1

def burger2(event):
    global burger
    burger = 2
def burger3(event):
```

```
    global burger
    burger = 3

def burger4(event):
    global burger
    burger = 4

def burger5(event):
    global burger
    burger = 5
```

```
# Прив'язка кнопок бургерів до подій
b1.bind("<Button-1>", burger1)
b2.bind("<Button-1>", burger2)
b3.bind("<Button-1>", burger3)
b4.bind("<Button-1>", burger4)
b5.bind("<Button-1>", burger5)
```

Оскільки всередині цих функцій ми змінюємо глобальну змінну *burger*, ми в кожній функції повинні попередньо вказати, що ця змінна є глобальною:

```
global burger
```

Інакше інтерпретатор створить всередині кожної функції внутрішню (локальну) змінну *burger*, а глобальна змінна *burger* буде залишатись незміненою.

6. Блок вибору картоплі фрі

Під блоком вибору картоплі фрі будемо розуміти чекбокс і дві радіокнопки. Чекбокс відповідає за вибір картоплі фрі, радіокнопки – за тип картоплі фрі (маленька чи велика порція).

Ці віджети будемо розміщувати всередині мітки № 2 (всередині віджета з іменем *l2*).

```
# Чекбокс для картоплі фрі (всередині мітки l2)
var_c1 = IntVar()
var_c1.set(0)
c1 = Checkbutton(l2, text="Картопля фрі", font="Arial 14",
                 bg="khaki2", variable=var_c1,
                 onvalue=1, offvalue=0)

# Радіокнопки для вибору типу картоплі фрі (всередині мітки l2)
var_r1 = IntVar()
var_r1.set(p1)
r11 = Radiobutton(l2, text="Маленька", font="Arial 10",
                  bg="gold", variable=var_r1, value=p1)
r12 = Radiobutton(l2, text="Велика", font="Arial 10",
                  bg="gold", variable=var_r1, value=p2)
```

Звернемо увагу на таке. Віджет *c1* (чекбокс) зв'язаний зі змінною *var_c1*. У разі ввімкненого чекбокса змінна приймає значення 1, у разі вимкненого – значення 0.

Радіокнопки *r11* та *r12* зв'язані зі змінною *var_r1*. Під час натискання першої радіокнопки у змінну *var_r1* кладеться значення змінної *p1* (кількість калорій у маленькій порції картоплі фрі). Під час натискання другої радіокнопки у змінну *var_r1* кладеться значення змінної *p2* (кількість калорій у великій порції картоплі фрі). Такий підхід дає змогу використовувати безпосередньо значення змінної *var_r1* під час розрахунку загальної кількості калорій, не роблячи додаткових перевірок.

Віджети *c1*, *r11*, *r12* упаковуються всередину віджета *l2* (мітки № 2):

```
c1.pack(side=TOP, expand=1, fill=BOTH)
r11.pack(side=LEFT, expand=1, fill=X)
r12.pack(side=LEFT, expand=1, fill=X)
```

Зазначимо, що для віджетів *c1*, *r11*, *r12* немає прив'язки до якихось подій. Єдина задача цих віджетів – зміна значень зв'язаних з ними змінних.

7. Блок вибору напою

Цей блок схожий з розглянутим вище блоком картоплі фрі. Відмінність полягає в тому, що тут використовуються три радіокнопки, що дають змогу обрати один з трьох напоїв.

Віджети цього блока упаковуються в змінну *l3* (мітка № 3). Функцією цих віджетів є зміна значень зв'язаних з ними змінних.

```
# Чекбокс для напою (всередині мітки l3)
var_c2 = IntVar()
var_c2.set(0)
c2 = Checkbutton(l3, text="Напій", font="Arial 14",
                 bg="khaki2", variable=var_c2,
                 onvalue=1, offvalue=0)

# Радіокнопки для вибору напою (всередині мітки l3)
var_r2 = IntVar()
var_r2.set(d1)
r21 = Radiobutton(l3, text="Кола", font="Arial 10",
                  variable=var_r2, value=d1,
                  bg="hotpink4", width=7)

r22 = Radiobutton(l3, text="Фанта", font="Arial 10",
                  variable=var_r2, value=d2,
                  bg="DarkOrange1", width=7)

r23 = Radiobutton(l3, text="Спрайт", font="Arial 10",
                  variable=var_r2, value=d3,
                  bg="springgreen3", width=7)
```

Упаковка віджетів:

```
c2.pack(side=TOP, expand=1, fill=BOTH)
r21.pack(side=LEFT, expand=1, fill=X)
r22.pack(side=LEFT, expand=1, fill=X)
r23.pack(side=LEFT, expand=1, fill=X)
```

8. Блок вибору соусу

Цей блок повністю схожий з блоком вибору напою. Зрозуміло, що для чекбокса, радіокнопок і зв'язаних з ними змінних використовуються інші ідентифікатори. Віджети прив'язуються до мітки *l4* і упаковуються всередину неї:

```
# Чекбокс для соусу (всередині мітки l4)
var_c3 = IntVar()
var_c3.set(0)
c3 = Checkbutton(l4, text="Соус", font="Arial 14",
                 bg="khaki2", variable=var_c3,
                 onvalue=1, offvalue=0)

# Радіокнопки для вибору соусу (всередині мітки l4)
var_r3 = IntVar()
var_r3.set(s1)
r31 = Radiobutton(l4, text="Сирний", font="Arial 10",
                  variable=var_r3, value=s1,
                  bg="khaki1", width=7)

r32 = Radiobutton(l4, text="Кері", font="Arial 10",
                  variable=var_r3, value=s2,
                  bg="gold2", width=7)

r33 = Radiobutton(l4, text="Барбекю", font="Arial 10",
                  variable=var_r3, value=s3,
                  bg="tomato3", width=7)
```

```
c3.pack(side=TOP, expand=1, fill=BOTH)
r31.pack(side=LEFT, expand=1, fill=X)
r32.pack(side=LEFT, expand=1, fill=X)
r33.pack(side=LEFT, expand=1, fill=X)
```

9. Мітка для виведення результату

Ця мітка (змінна *l5*) розглянута в п. 3. Вона розташовується в головному вікні програми нижче мітки *l4*. Через параметр *textvariable* мітка зв'язується з символьною змінною *ryadok*.

10. Функція виведення результату

Для відображення інформації в мітці *l5* використовується функція *show*. Її задачею є відображення інформації, що зберігається у змінних *burger*, *c1-c3*, *r11-r33* у вигляді тексту.

Суть функції полягає в тому, що в процесі її роботи поступово заповнюється символьна змінна *s*, яка в кінці функції поміщується в змінну *ryadok*, що зв'язана з міткою *l5*. Отже, після виклику функції show буде змінений текст всередині мітки *l5*.

```
# Виведення поточного набору продуктів
def show(): # функція не має аргументів
    s=""
    if burger == 0:
        s = "Бургер не обраний!"
    else:
        # Створення рядка з назвою бургера
        if burger == 1: s = "БігМак\n"
        if burger == 2: s = "Біг Тейсті\n"
        if burger == 3: s = "Чизбургер\n"
        if burger == 4: s = "Чікенбургер\n"
        if burger == 5: s = "МакМафін\n"

        # Створення рядка з назвою картоплі фрі
        if var_c1.get()==1:
            if var_r1.get()==p1:
                s+="Маленька картопля фрі\n"
            else:
                s+="Велика картопля фрі\n"
        else:
            s+="Без картоплі фрі\n"

        # Створення рядка з назвою напою
        if var_c2.get() == 1:
            if var_r2.get() == d1:
                s+="Напій Кола\n"
            if var_r2.get() == d2:
                s+="Напій Фанта\n"
            if var_r2.get() == d3:
                s+="Напій Спрайт\n"
        else:
            s+="Без напою\n"

        # Створення рядка з назвою соусу
        if var_c3.get() == 1:
            if var_r3.get() == s1:
                s+="Соус Сирний\n"
            if var_r3.get() == s2:
                s+="Соус Карі\n"
            if var_r3.get() == s3:
                s+="Соус Барбекю\n"
        else:
            s+="Без соусу\n"
```

```
    # Передаємо в мітку l5 сформований рядок s
    ryadok.set(s)
    # На цьому функція show() завершується.
```

11. Кнопка розрахунку калорій

Ця кнопка слуие' для виклику функції *calculator* і упаковується у головне вікно програми (root) нижче мітки *l5*:

```
# Кнопка для розрахунку калорій
b6 = Button(root, text="Розрахувати калорії",
            font="Arial        24        bold",        bg="red2",
activebackground="red1",
            fg="white", activeforeground="white")
b6.bind("<Button-1>", calculator)
```

```
b6.pack(side=TOP, expand=1, fill=BOTH)
```

12. Функція розрахунку калорій

Wz функція зв'язана з кнопкою *b6* «Розрахувати калорії». Алгоритм роботи функції nfrbq:

1. Викликається функція *show*. Вона формує змінну *ryadok*, в якій будуть перелічені назви обраних продуктів. Те, що цей рядок відразу буде відображений у мітці *l5*, зараз неважливо.

2. Обнуляється змінна *kalorii*, після чого до неї додаються калорії обраного бургера.

3. Якщо картопля фрі обрана, до змінної *kalorii* додається відповідна кількість калорій.

4. Якщо напій обраний, до змінної *kalorii* додається відповідна кількість калорій.

5. Якщо соус обраний, до змінної *kalorii* додається відповідна кількість калорій.

6. Перетворення змінної *kalorii* у символьний рядок і додавання її до змінної *ryadok*. В результаті в мітці *l5* будуть автоматично відображені обрані продукти і кількість калорій.

```
# Розрахунок калорій
def calculator(event):
    show() # Спочатку виводимо інформацію про обрані продукти

    if burger == 0: # Якщо бургер не обраний, калорії не
розраховуємо
        return

    kalorii = 0

    # Початкове значення калорій залежить від типу бургера
    if burger == 1: kalorii = k1
```

```
    if burger == 2: kalorii = k2
    if burger == 3: kalorii = k3
    if burger == 4: kalorii = k4
    if burger == 5: kalorii = k5

    if var_c1.get()==1: # Якщо картопля фрі обрана
        kalorii += var_r1.get()

    if var_c2.get()==1: # Якщо напій обраний
        kalorii += var_r2.get()

    if var_c3.get()==1: # Якщо соус обраний
        kalorii += var_r3.get()

    # Додаємо до напису кількість калорій
    s = str(ryadok.get())
    s = s + "\n" + str(kalorii) + " кілокалорій."
    ryadok.set(s)
```

13. Виведення інформації про автора програми

Для цього використовується кнопка *b7* і зв'язана з нею функція *about*:

```
# Натискання кнопки "Про програму"
def about(event):
    s = "Лабораторна робота 1\n"+\
        "Виконав:\nстудент гр. КН-24-а\nВєтров О.С."
    ryadok.set(s)
    burger=0
```

```
# Кнопка "Про програму"
b7 = Button(root, text="Про програму",
     font="Arial 8", bg="khaki3", activebackground="khaki2")
b7.bind("<Button-1>", about)
```

```
b6.pack(side=TOP, expand=1, fill=BOTH)
```

14. Повний лістинг програми

```
# coding=1251
from tkinter import *
root = Tk()
root.title("McDonald's - калорії")
#-----------------------------------------------
# Глобальні змінні

burger = 0 # Тип бургера: 0 - не обраний, 1 - Біг Мак,
           # 2 - Біг Тейсті, 3 - Чизбургер,
           # 4 - Чікенбургер, 5 - МакМафін.

# Калорійність бургерів (по порядку)
```

```
k1, k2, k3, k4, k5 = 1200, 1700, 2300, 1400, 2800

# Калорийність картоплі фрі
p1, p2 = 700, 1400

# Калорійність напоїв
d1, d2, d3 = 1000, 1100, 800

# Калорійність соусів
s1, s2, s3 = 120, 150, 200

#-----------------------------------------------
# Користувацькі функції

# Функції вибору бургерів
def burger1(event):
    global burger
    burger = 1

def burger2(event):
    global burger
    burger = 2

def burger3(event):
    global burger
    burger = 3

def burger4(event):
    global burger
    burger = 4

def burger5(event):
    global burger
    burger = 5

# Виведення поточного набору продуктів
def show(): # функція не має аргументів
    s=""
    if burger == 0:
        s = "Бургер не обраний!"
    else:
        # Створення рядка з назвою бургера
        if burger == 1: s = "БігМак\n"
        if burger == 2: s = "Біг Тейсті\n"
        if burger == 3: s = "Чизбургер\n"
        if burger == 4: s = "Чікенбургер\n"
        if burger == 5: s = "МакМафін\n"




        # Створення рядка з назвою картоплі фрі
```

```
        if var_c1.get()==1:
            if var_r1.get()==p1:
                s+="Маленька картопля фрі\n"
            else:
                s+="Велика картопля фрі\n"
        else:
            s+="Без картоплі фрі\n"

        # Створення рядка з назвою напою
        if var_c2.get() == 1:
            if var_r2.get() == d1:
                s+="Напій Кола\n"
            if var_r2.get() == d2:
                s+="Напій Фанта\n"
            if var_r2.get() == d3:
                s+="Напій Спрайт\n"
        else:
            s+="Без напою\n"

        # Створення рядка з назвою соусу
        if var_c3.get() == 1:
            if var_r3.get() == s1:
                s+="Соус Сирний\n"
            if var_r3.get() == s2:
                s+="Соус Карі\n"
            if var_r3.get() == s3:
                s+="Соус Барбекю\n"
        else:
            s+="Без соусу\n"

    # Передаємо в мітку l5 сформований рядок s
    ryadok.set(s)
    # На цьому функція show() завершується.


# Розрахунок калорій
def calculator(event):
    show() # Спочатку виводимо інформацію про обрані продукти

    if burger == 0: # Якщо бургер не обраний,
                    # калорії не розраховуємо
        return

    kalorii = 0

    # Початкове значення калорій залежить від типу бургера
    if burger == 1: kalorii = k1
    if burger == 2: kalorii = k2
    if burger == 3: kalorii = k3
    if burger == 4: kalorii = k4
    if burger == 5: kalorii = k5

    if var_c1.get()==1: # Якщо картопля фрі обрана
```

```
        kalorii += var_r1.get()

    if var_c2.get()==1: # Якщо напій обраний
        kalorii += var_r2.get()

    if var_c3.get()==1: # Якщо соус обраний
        kalorii += var_r3.get()

    # Додаємо до напису кількість калорій
    s = str(ryadok.get())
    s = s + "\n" + str(kalorii) + " кілокалорій."
    ryadok.set(s)

# Натискання кнопки "Про програму"
def about(event):
    s = "Лабораторна робота 1\n"+\
        "Виконав:\nстудент гр. КН-24-а\nВєтров О.С."
    ryadok.set(s)
    burger=0

#-----------------------------------------------
# Створення віджетів

# Мітки
l1 = Label(root, text="Label 1", bg="khaki3", bd=0)
l2 = Label(root, text="Label 2", bg="khaki3", bd=0)
l3 = Label(root, text="Label 3", bg="khaki3", bd=0)
l4 = Label(root, text="Label 4", bg="khaki3", bd=0)

ryadok = StringVar()
ryadok.set("")
l5 = Label(root, textvariable= ryadok, bg="white", bd=0,
             font="Courier 18 bold", height=9, width=30)

# Кнопки з назвами бургерів (всередині мітки l1)
b1 = Button(l1, text="Біг Мак", font="Arial 20 bold",
            bg="Goldenrod2", activebackground="Goldenrod1")
b2 = Button(l1, text="Біг Тейсті", font="Arial 20 bold",
            bg="Goldenrod2", activebackground="Goldenrod1")
b3 = Button(l1, text="Чизбургер", font="Arial 20 bold",
            bg="Goldenrod2", activebackground="Goldenrod1")
b4 = Button(l1, text="Чікенбургер", font="Arial 20 bold",
            bg="Goldenrod2", activebackground="Goldenrod1")
b5 = Button(l1, text="МакМафін", font="Arial 20 bold",
            bg="Goldenrod2", activebackground="Goldenrod1")

# Прив'язка кнопок бургерів до подій
b1.bind("<Button-1>", burger1)
b2.bind("<Button-1>", burger2)
b3.bind("<Button-1>", burger3)
b4.bind("<Button-1>", burger4)
b5.bind("<Button-1>", burger5)
```

```
# Чекбокс для картоплі фрі (всередині мітки l2)
var_c1 = IntVar()
var_c1.set(0)
c1 = Checkbutton(l2, text="Картопля фрі", font="Arial 14",
                 bg="khaki2", variable=var_c1,
                 onvalue=1, offvalue=0)

# Радіокнопки для вибору типу картоплі фрі (всередині мітки l2)
var_r1 = IntVar()
var_r1.set(p1)
r11 = Radiobutton(l2, text="Маленька", font="Arial 10",
                  bg="gold", variable=var_r1, value=p1)
r12 = Radiobutton(l2, text="Велика", font="Arial 10",
                  bg="gold", variable=var_r1, value=p2)


# Чекбокс для напою (всередині мітки l3)
var_c2 = IntVar()
var_c2.set(0)
c2 = Checkbutton(l3, text="Напій", font="Arial 14",
                 bg="khaki2", variable=var_c2,
                 onvalue=1, offvalue=0)

# Радіокнопки для вибору напою (всередині мітки l3)
var_r2 = IntVar()
var_r2.set(d1)
r21 = Radiobutton(l3, text="Кола", font="Arial 10",
                  variable=var_r2, value=d1,
                  bg="hotpink4", width=7)

r22 = Radiobutton(l3, text="Фанта", font="Arial 10",
                  variable=var_r2, value=d2,
                  bg="DarkOrange1", width=7)

r23 = Radiobutton(l3, text="Спрайт", font="Arial 10",
                  variable=var_r2, value=d3,
                  bg="springgreen3", width=7)

# Чекбокс для соусу (всередині мітки l4)
var_c3 = IntVar()
var_c3.set(0)
c3 = Checkbutton(l4, text="Соус", font="Arial 14",
                 bg="khaki2", variable=var_c3,
                 onvalue=1, offvalue=0)

# Радіокнопки для вибору соусу (всередині мітки l4)
var_r3 = IntVar()
var_r3.set(s1)
r31 = Radiobutton(l4, text="Сирний", font="Arial 10",
                  variable=var_r3, value=s1,
                  bg="khaki1", width=7)
r32 = Radiobutton(l4, text="Кері", font="Arial 10",
                  variable=var_r3, value=s2,
```

```
                    bg="gold2", width=7)

r33 = Radiobutton(l4, text="Барбекю", font="Arial 10",
                    variable=var_r3, value=s3,
                    bg="tomato3", width=7)

# Кнопка для розрахунку калорій
b6 = Button(root, text="Розрахувати калорії",
             font="Arial 24 bold",
             bg="red2", activebackground="red1",
             fg="white", activeforeground="white")
b6.bind("<Button-1>", calculator)

# Кнопка "Про програму"
b7 = Button(root, text="Про програму",
             font="Arial 8", bg="khaki3",
             activebackground="khaki2")
b7.bind("<Button-1>", about)

# Упаковка віджетів
l1.pack(side=LEFT, expand=1, fill=BOTH)
l2.pack(side=TOP, expand=1, fill=BOTH)
l3.pack(side=TOP, expand=1, fill=BOTH)
l4.pack(side=TOP, expand=1, fill=BOTH)
l5.pack(side=TOP, expand=1, fill=BOTH)

b1.pack(side=TOP, expand=1, fill=BOTH)
b2.pack(side=TOP, expand=1, fill=BOTH)
b3.pack(side=TOP, expand=1, fill=BOTH)
b4.pack(side=TOP, expand=1, fill=BOTH)
b5.pack(side=TOP, expand=1, fill=BOTH)

c1.pack(side=TOP, expand=1, fill=BOTH)
r11.pack(side=LEFT, expand=1, fill=X)
r12.pack(side=LEFT, expand=1, fill=X)

c2.pack(side=TOP, expand=1, fill=BOTH)
r21.pack(side=LEFT, expand=1, fill=X)
r22.pack(side=LEFT, expand=1, fill=X)
r23.pack(side=LEFT, expand=1, fill=X)

c3.pack(side=TOP, expand=1, fill=BOTH)
r31.pack(side=LEFT, expand=1, fill=X)
r32.pack(side=LEFT, expand=1, fill=X)
r33.pack(side=LEFT, expand=1, fill=X)

b6.pack(side=TOP, expand=1, fill=BOTH)
b7.pack(side=LEFT, expand=1, fill=BOTH)

root.mainloop()
```

Додаткові елементи бібліотеки *tkinter*

Метою лабораторної роботи є вивчення та одержання навичок практичного використання додаткового набору віджетів бібліотеки *tkinter*: *Entry*, *Text*, *Listbox*, *Scale*, *Spinbox*, *Optionmenu*, *Frame*.

Завдання до лабораторної роботи

Модифікувати програму, розроблену в лабораторній роботі № 13, з урахуванням таких вимог:

1. В інтерфейсі програми хоча б один раз має бути використаний віджет *Entry* (однорядкове текстове поле). Віджет може або замінювати деякі віджети з лабораторної роботи № 13, або використовуватися паралельно з ними. Віджет *Text* (багаторядкове текстове поле) також може бути використаний, але у випадку об'єктивної необхідності.

2. Хоча б один раз має бути використаний віджет *Listbox* (список) або віджет *Scale* (повзунок). У тих випадках, коли потрібен вибір одного з параметрів, записаних у вигляді тексту, повинен використовуватися віджет *Listbox*. Якщо потрібно вибрати одне із числових значень, може використовуватися будь-який з цих віджетів залежно від зручності. Віджет може як замінювати деякі віджети з лабораторної роботи № 13, так і використовуватися паралельно з ними.

3. Хоча б один раз має бути використаний віджет *Spinbox* (поле введення із попередньо встановленими варіантами) або віджет *Optionmenu* (випадний список). Ці віджети рекомендується використовувати за відносно невеликої кількості обираних елементів (до 10). Віджет може або заміняти деякі віджети з лабораторної роботи № 13, або використовуватися паралельно з ними.

4. Усі віджети *Label* (мітка), які використовувалися в лабораторній роботі № 10 для розміщення інших віджетів, замінити на віджети *Frame* (кадр). Якщо віджет *Label* використовувався не для розміщення інших віджетів, а для виводу інформації, то замінювати його на *Frame* не потрібно.

5. Обов’язковою вимогою в цій лабораторній роботі є розширення функціоналу програми за допомогою доданих віджетів. Програма повинна вміти робити щось, чого не вміла програма з лабораторної роботи № 13. Це може бути введення додаткових параметрів, обчислення нових величин, відображення додаткової інформації тощо. Цей додатковий функціонал повинен бути реалізований обов’язково за допомогою віджетів з цієї лабораторної роботи.

Вміст звіту з лабораторної роботи

1. Титульний аркуш.
2. Індивідуальний варіант завдання з лабораторної роботи № 13.
3. Лістинг програми.
4. Приклад роботи програми (знімки екрана).
5. Висновки до лабораторної роботи.

Лабораторна робота № 15

Позиціонування віджетів за допомогою методів *place* і *grid*

Метою лабораторної роботи є вивчення та одержання навичок практичного використання методів розміщення віджетів *place* і *grid* бібліотеки *tkinter*.

Завдання до лабораторної роботи

1. Модифікувати програму, розроблену в лабораторній роботі № 13 або 14, виконавши позиціонування віджетів за допомогою методу *place*.
2. Модифікувати програму, розроблену в лабораторній роботі № 13 або 14, виконавши позиціонування віджетів за допомогою методу *grid*.

Вміст звіту з лабораторної роботи

1. Титульний аркуш.
2. Індивідуальний варіант завдання з лабораторної роботи № 13.
3. Лістинг програми.
4. Приклад роботи програми (знімки екрана).
5. Висновки до лабораторної роботи.

Лабораторна робота № 16

Побудова графіків функцій засобами віджета *Canvas*

Метою лабораторної роботи є вивчення та одержання навичок практичного використання віджета *Canvas* з бібліотеки *tkinter*.

Завдання до лабораторної роботи

За допомогою віджета *Canvas* побудувати для заданої параметричної функції її графік. Крім графіка, на зображенні мають бути наявні:

– осі координат;

– розмітка осей;

– задані формули для побудови графіка;

– прізвище та ініціали студента (дрібним шрифтом у правому нижньому куті полотна).

Якщо графік не може бути побудований на екрані повністю, треба відобразити на екрані найбільш важливі ділянки графіка, що дають уявлення про його форму.

Індивідуальний варіант завдання

У наступній таблиці в першому стовпці вказаний номер студента в журналі групи, у другому стовпці – задані функції $x(t)$ та $y(t)$.

Рекомендації до виконання лабораторної роботи

Під час побудови графіків можна використовувати програму-заготовку, наведену нижче. Подібна програма докладно розглянута в лекції, присвяченій побудові графіків за допомогою віджета *Canvas*.

Відповідно до індивідуального варіанта завдання в заготовку треба внести такі зміни:

1) Змінити функції $x(t)$ та $y(t)$ на задані.
2) Підібрати діапазон зміни аргументу t (значення змінних *t1* та *t2*) так, щоб відобразити всі важливі частини графіка.
3) Підібрати крок *dt* зміни змінної t так, щоб одержати графік достатньої щільності за мінімальний час.
4) Підібрати коефіцієнти *kx* та *ky* так, щоб графік містився на екрані в найбільш зручному масштабі.
5) У разі потреби змінити змінні *xx0* та *yy0*, які є координатами центру осей координат, а також розміри самого віджета.
6) На осях координат додати розмітку значень (достатньо 2–3 позначок на кожній півосі).
7) Додати прізвище та ініціали студента.

```
from tkinter import *
from math import *
root = Tk()

xx_max, yy_max = 800, 600        # Розміри віджета Canvas
xx0, yy0 = xx_max/2, yy_max/2

c1 = Canvas(root, width=xx_max, height=yy_max, bg="white")
c1.create_line(0, yy0, xx_max, yy0, width=1,
               arrow=LAST, arrowshape=(20, 25, 10))
c1.create_line(xx0, yy_max, xx0, 0, width=1,
               arrow=LAST, arrowshape=(20, 25, 10))

t1, t2 = -10, 10
dt = 0.001
kx, ky = 100, 250

t = t1
while (t<t2):

    x = sin(2*t)              # Функція x(t)
    y = cos(0.5*t-1)          # Функція y(t)

    xx = kx*x+xx0
    yy = ky*y*(-1)+yy0
    c1.create_oval(xx, yy, xx, yy, fill="black")
    t=t+dt

c1.pack()
root.mainloop()
```

Вміст звіту з лабораторної роботи

1. Титульний аркуш.
2. Індивідуальний варіант завдання.
3. Лістинг програми.
4. Приклад роботи програми (знімки екрана) із зазначенням основних параметрів побудови графіка.
5. Висновки до лабораторної роботи.

Індивідуальне творче завдання
(другий семестр)

Індивідуальне творче завдання другого семестру загалом повторює завдання першого семестру з урахуванням окремих вимог, що стосуються навчального матеріалу другого семестру.

Завдання полягає в розробці програми, якою може користуватись людина заданої професії. Професія, з якою працює студент, залишається такою самою, як і в індивідуальному творчому завданні першого семестру. Важливим є написання не просто програми, а корисної програми, яка поліпшує повсякденну роботу людини-фахівця.

Нижче наведений перелік вимог, які обов'язково мають бути виконані тою чи іншою мірою. Вимоги, які ставились до ІТЗ в першому семестрі, тут не діють, але більшість із них рекомендується до виконання.

1. Дозволяється або писати нову програму «з нуля», або дороблять програму, розроблену в межах ІТЗ першого семестру. Результуюча програма повинна розв'язувати не менш ніж три задачі для фахівця заданої професії, які не повинні перетинатись із задачами з ІТЗ першого семестру.
2. Програма повинна мати графічний інтерфейс, розроблений за допомогою бібліотеки *tkinter*.
3. Хоча б в одній із задач має бути робота з масивами даних (на основі списків).
4. Має бути робота з файлами. Можна або прочитувати початкові дані з файла, або записувати у файл результати роботи, або і те, і інше. Тип файлів, а також дані, що зберігатимуться у файлі, визначаються на розсуд студента відповідно до розв'язуваної задачі. В інтерфейсі програми мають бути окремі елементи (віджети), що стосуються роботи з файлами. Наприклад: «Читання з файла», «Запис у файл». Деякі віджети можуть «не працювати», поки дані з файла не прочитані.
5. Початковий вміст масивів може заповнюватись різними способами: або вводитися з клавіатури, або генеруватись у псевдовипадковий спосіб, або прочитуватись із файла. Можливе використання різних способів початкового заповнення масиву. Для початкового заповнення масиву доцільно передбачити окремий віджет.

Усі інші вимоги (щодо оформлення, критеріїв оцінювання, термінів захисту) – як у першому семестрі.

СПИСОК РЕКОМЕНДОВАНОЇ ЛІТЕРАТУРИ

Основна література

1. Васильєв О. Програмування мовою Python. Тернопіль: «Навчальна книга Богдан», 2019. 504 с.
2. Руденко В. Д., Жугастров О. О. Основи алгоритмізації і програмування мовою Python. Харків: Ранок, 2019. 192 с.
3. Вступ до алгоритмів. Томас Г. Кормен, Чарлз Е. Лейзерсон, Роланд Л. Рівест, Кліфорд Стайн. Київ: К.І.С., 2019. 1288 с.
4. Бородкіна І., Бородкін Г. Інженерія програмного забезпечення: навчальний посібник. Київ: Центр навчальної літератури, 2018. 204 с.
5. Лавріщева К. М. Програмна інженерія. Підручник. Київ: Видавничий дім «Академперіодика» НАН України, 2008. 319 с.

Допоміжна література

1. Слипченко В. Задачі з програмування мовою Python. Київ: Шкільний світ, 2019. 120 с.
2. Кривий С. Л. Вступ до методів створення програмних продуктів. Чернівці: Букрек, 2012. 424 с.
3. Анісімов А. В. Програмування числових методів мовою Python: підручник / А. В. Анісімов, А. Ю. Дорошенко, С. Д. Погорілий, Я. Ю. Дорогий; за ред. А. В. Анісімова. Київ: Видавничо-поліграфічний центр «Київський університет», 2014. 640 с.

Інформаційні ресурси в мережі Інтернет

1. Підручник з Python. URL: https://docs.python.org/uk/3/tutorial/index.html
2. Путівник мовою програмування Python. URL: https://pythonguide.rozh2sch.org.ua/
3. Python Basics. URL: https://www.pythontutorial.net/python-basics/
4. Python tkinter manual. URL: https://docs.python.org/uk/3/library/tkinter.html

Навчально-методичне видання

Бабаков *Роман Маркович*

МЕТОДИЧНІ РЕКОМЕНДАЦІЇ НАВЧАЛЬНОЇ ДИСЦИПЛІНИ

ОСНОВИ ПРОГРАМУВАННЯ