ТОВ «ТЕХНІЧНИЙ УНІВЕРСИТЕТ
«МЕТІНВЕСТ ПОЛІТЕХНІКА»

КОМП'ЮТЕРНА ТЕХНІКА, АЛГОРИТМІЗАЦІЯ ТА ПРОГРАМУВАННЯ:

методичні вказівки до виконання індивідуальних завдань

Запоріжжя 2026

metinvest polytechnic

УДК 004.021:004.42:004.382 (072)
К63

Рекомендовано Науково-методичною радою
ТОВ «ТЕХНІЧНИЙ УНІВЕРСИТЕТ
«МЕТІНВЕСТ ПОЛІТЕХНІКА»
(протокол № 5 від 27.02.2026 р.)

Укладачі

Мірошниченко В.І., канд. техн. наук, доцент

Мірошниченко С.О., старший викладач,

Полупан В.Г., старший викладач

К63 Комп'ютерна техніка, алгоритмізація та програмування : методичні рекомендації до виконання індивідуальних завдань / уклад.: В. І. Мірошниченко, С. О. Мірошниченко, В. Г. Полупан. Запоріжжя : ТОВ «ТЕХНІЧНИЙ УНІВЕРСИТЕТ «МЕТІНВЕСТ ПОЛІТЕХНІКА», 2026. 60 с.

У методичних рекомендаціях наведено тематику індивідуальних завдань, методичні пояснення щодо порядку та приклади їх виконання, критерії оцінювання, вимоги до оформлення звітів, питання для самоперевірки тощо.

Рекомендовано для здобувачів бакалаврського рівня вищої освіти за освітньо-професійною програмою «Інжиніринг механічного обладнання та систем», спеціальність 131 Прикладна механіка.

УДК 004.021:004.42:004.382 (072)

ЗМІСТ

ВСТУП

Метою виконання індивідуальних завдань з дисципліни «Комп'ютерна техніка, алгоритмізація та програмування» є формування у здобувачів вищої освіти за освітньо-професійною програмою «Інжиніринг механічного обладнання та систем» алгоритмічного мислення та здатності програмувати лінійні та розгалужені алгоритми з використання мови програмування Python шляхом створення консольного додатку для розв’язання математичної задачі, а також формування практичних навичок опрацювання текстових даних мовою Python, зокрема виконання стандартних перетворень рядків, застосування лексикографічних порівнянь, підрахунків за ознаками та сортування за різними критеріями.

Методичні рекомендації містять теоретичні відомості щодо основних понять, визначень, операцій, необхідні для розробки алгоритму розв’язання поставленої задачі та реалізації відповідного програмного коду, а також приклади розв’язання типових задач та оформлення звітів.

1 КРИТЕРІЇ ОЦІНЮВАННЯ

Виконання кожного з індивідуальних завдань в повному обсязі може зайняти до 180 хв. Підготовлений звіт з індивідуального у вигляді файлу *docx, або *pdf розміщується у відповідному розділі дисципліни в Moodle і перевіряється протягом тижня після завершення терміну подачі. Оскарження оцінки може бути здійснене на останньому практичному занятті модуля.

Здобувач(ка) може отримати максимальну кількість балів (10) за кожне індивідуальне завдання, з них

- 7 балів за умов:

– вибору відповідного (стандартного) завдання (для ІЗ №1) та продемонстрованих під час захисту набутих навичок алгоритмічного мислення та розроблення програми на мові Python у чіткій відповідності до завдання,

– розроблена під час роботи над завданням програма проходить тестування;

– завантажено оформлений за вимогами звіт у форматі *.docx(pdf) та файл *.txt (*.py) з кодом розробленої програми в Moodle згідно з семестровим графіком;

- 3 бали як міра оцінювання ініціативності при захисті розроблених блок-схеми та програми, логічності та структурованості відповіді, здатності комунікувати у команді та під впливом негативних факторів, в т.ч. під тиском викладача та/або групи, вміння вести дискусію та бути критичним та самокритичним.

Якщо здобувач(ка) при виконанні індивідуального завдання №1 обрав(ла) спрощене завдання, результат його виконання, що відповідає варіанту та оформлений у вигляді звіту (див. п. 2), може бути оцінений у 6 балів максимально.

2 ВИМОГИ ДО ОФОРМЛЕННЯ ЗВІТІВ

Звіт має бути оформлений відповідно до вимог щодо оформлення технічної документації на аркушах формату А4 у вигляді файлу *.docx або *.pdf. До складу звіту повинні входити:

1. Титульна сторінка.
2. Назва та мета індивідуального завдання.
3. Постановка завдання відповідно варіанту.
4. Блок-схема алгоритму розв’язання задачі.
5. Лістінг програми.
6. Результати роботи програми.
7. Висновки.

3 ІНДИВІДУАЛЬНЕ ЗАВДАННЯ №1

3.1 Теоретичні відомості

3.1.1 Основні поняття, визначення, операції

Приклади оголошення змінних:

```
n1 = 0
k2 = 0
max_value = 0
a = 0.0
```

Над змінними числових типів виконують такі операції:

+, -, *, / – арифметичні операції – відповідно додавання, віднімання, множення та ділення (операція / завжди повертає число з плаваючою точкою);

// – цілочисельне ділення (ділення націло);

% – остача від ділення; ** – піднесення до степеня.

У Python для введення та виведення використовуються такі засоби:

= – присвоєння – запис до змінної значення арифметичного виразу;

input() – введення – зчитування рядка з клавіатури; за потреби перетворюємо у int, float тощо;

print() – виведення значень змінних на екран; для форматування використовуються f-рядки.

Приклади присвоєння й арифметичних операцій:

```
n1 = 1  # присвоїти 1 змінній n1
n1 = n1 + 3  # збільшити n1 на 3
k2 = n1 * (n1 - 1) + 5  # дужки змінюють пріоритет
операцій
k2 = 999 % 100  # k2 = 99
q = 7 // 3  # цілочисельне ділення: q = 2
p = 2 ** 5  # піднесення до степеня: p = 32
```

Правила використання input()/print():

```
# input: зчитує рядок (string). Необхідні типи
перетворюємо явно
text = input("Введіть значення: ")

# print: виводить значення; за замовчуванням розділяє
```

```
# пробілами
print("Результат:", text)
```

де перелік змінних у print() розділяється комами і виводиться зі вставленням пробілів.

Приклади інструкції введення (через пробіл):

```
# зчитування однієї змінної
n1 = int(input("Введіть n1: "))

# зчитування кількох змінних, розділених пробілом
a, k2, max_value = input("Введіть a k2 max_value:
").split()
a = float(a); k2 = int(k2); max_value =
int(max_value)

# або використання map для числового перетворення
n1, a = map(float, input("Введіть n1 a: ").split())
```

Приклад інструкції введення (через кому):

```
line = input("Введіть n1,a (через кому): ")
n1_str, a_str = line.split(",")
n1 = int(n1_str)
a = float(a_str)
```

Приклади інструкції виведення:

```
print(a)  # виведення однієї змінної

# виведення кількох змінних з пробілами
print(k2, n1)

# виведення мітки та значення
print("max_value =", max_value)
```

Форматоване виведення з використанням f-рядків:

```
a = 3.1415; n1 = 100
print(f"a = {a:4.2f}, n1 = {n1}.")

# перенос рядка можна додати у рядок або керувати
параметром end
print(f"a = {a:4.2f},\n n1 = {n1}.")
```

Таблиця 3.1 – Інкрементні та декрементні операції

Операція	Опис	Приклад	Еквівалент
`+= n`	збільшення змінної на n	`i += n`	`i = i + n`
`-= n`	зменшення змінної на n	`i -= n`	`i = i - n`
`*= n`	множення змінної на n	`i *= n`	`i = i * n`
`/= n`	ділення змінної на n	`i /= n`	`i = i / n`
`//= n`	цілочисельне ділення на n	`i //= n`	`i = i // n`
`%= n`	остача від ділення на n	`i %= n`	`i = i % n`
`**= n`	піднесення до степеня n	`i **= n`	`i = i ** n`

3.1.2 Умови

Мова програмування Python має механізми для роботи з алгеброю висловлювань та умовами в цілому. Логічні значення представлені булевим типом `bool` із двома значеннями: `True` та `False`. Числові значення в умовах інтерпретуються так: 0 вважається хибним, будь-яке інше число – істинним.

Для запису умов у Python використовуються:

- операції відношення (==, !=, <, >, >=, <=)
  - == – дорівнює;
  - != – не дорівнює;
  - < – менше;
  - > – більше;
  - <= – менше або дорівнює;
  - >= – більше або дорівнює;
- умовні операції (or, and, not)
  - `or` – диз'юнкція;
  - `and` – кон'юнкція;
  - `not` – заперечення.

Наприклад, умовою того, що рівняння *ax+b = 0* з коефіцієнтами *a* та *b* має єдиний розв'язок, є нерівність *a ≠ 0*. Залежно від значення умови `True` або `False` можна обчислити розв'язок рівняння або з'ясувати, чи має воно нескінченно багато розв'язків, чи не має жодного (для цього може знадобитися умова b == 0).

Найпростіші умови дуже часто мають вигляд порівнянь. Наприклад, умовою того, що значення цілої змінної *a* парне (ділиться на 2 без остачі), є те, що воно дає остачу 0 від ділення на 2: *a%2==0*.

Складніші умови формулюють як системи або сукупності, складені з простіших умов. Системи умов записують мовою Python за допомогою операції and, сукупності – за допомогою or.

Наприклад, умовою того, що число x належить проміжку $[a; b]$, є математичний вираз $a \leq x \leq b$, якій аналогічно записується у Python.

Аналогічно умову того, що значення змінної $c: str$ є десятковою цифрою, можна записати як $"0" \leq c \leq "9"$.

```
c: str = '10'
print("0" <= c <= "9")  # True
```

Той факт, що цілі ненульові числа *a*, *b*, *c* утворюють геометричну прогресію, математично можна виразити як ***b/a = c/b***. Ділення цілих у Python може повертати число з плаваючою точкою, а порівняння дійсних значень на рівність не є надійним. Щоб уникнути цього, запишемо умову у математично еквівалентній формі: *bb = ac*. Отже, умова того, що числа *a*, *b, c* утворюють геометричну прогресію, у Python має вигляд ***b*b==a*c***.

Умовний вираз у мові Python записується так

```
a if F else b
```

та називається «тернарний оператор».

Значенням виразу «`x if x >= 0 else -x`» є модуль числового значення змінної *x*.

Значенням виразу «`a if a < b else b`» є мінімальне зі значень числових змінних *a* та *b*.

3.1.3 Розгалуження

Структура керування, що виконує певну дію (перелік дій), залежно від істинності деякої умови F, називається **розгалуженням**.

Для програмування розгалужень у Python реалізовано умовний оператор, що має такий синтаксис

```
if F:
    # P – інструкція або послідовність інструкцій для випадку істинності
    ...
else:
    # Q – інструкція або послідовність інструкцій для випадку хибності
    ...
```

де F – умова, P – інструкція (або набір інструкцій), що виконується у випадку істинності умови F, Q – інструкція (або набір інструкцій), що виконується у випадку хибності умови F.

Блок розгалуження `else` є необов'язковим, якщо у випадку хибності умови F програма не має виконувати жодних дій.

Для багатоваріантного вибору використовується `elif`:

```
if x < 0:
    # гілка 1
    ...
elif x == 0:
    # гілка 2
    ...
else:
    # гілка 3
    ...
```

3.1.4 Точність обчислень. Правила обчислення математичних виразів

Відкидання молодших двійкових розрядів при переході до внутрішнього двійкового поданням призводить до появи специфічної "машинної" похибки. Похибка відтворення дійсних чисел залежить від кількості бітів мантиси. У Python тип `float` зазвичай має подвійну точність (IEEE 754, 64 біти).

Фактично машинну похибку можна визначити як деяке максимальне число ε, яке, при додаванні до дійсного числа, не змінює його (через обмежену розрядну сітку), тобто виконується рівність

$$A + \varepsilon = A. \quad (1)$$

Приклад програми, що використовує в циклі рівність (1) як умову виходу з циклу:

```
x = 1e-9
t = 0.5
while x != x + t:
    t = t / 2.0
    print(f"x = {x:e} t = {t:e}")
```

У таблиці 3.2 наведені результати роботи програми для різних значень *x*.

Таблиця 3.2 - Результати роботи програми

X	Машинна
0,0000000001 (10^{-10})	≈$3.23 \cdot 10^{-27}$
0.0000001 (10^{-7})	≈$6.61 \cdot 10^{-24}$
0.0001 (10^{-4})	≈$3.39 \cdot 10^{-21}$
0.001 (10^{-3})	≈$1,08 \cdot 10^{-19}$
0.01 (10^{-2})	≈$4,33 \cdot 10^{-19}$
0.1 (10^{-1})	≈$6.94 \cdot 10^{-18}$
1	≈$1.11 \cdot 10^{-16}$

10	$\approx 8.88 \cdot 10^{-16}$
100 (10^2)	$\approx 7.1 \cdot 10^{-15}$
1000 (10^3)	$\approx 5.68 \cdot 10^{-14}$
1000000 (10^6)	$\approx 5.82 \cdot 10^{-11}$
1000000000 (10^9)	$\approx 5.96 \cdot 10^{-8}$

Програма для обчислення оцінки ε для кожного значення x з таблиці 3.2:

```
# обчислення оцінки машинної похибки epsilon для
набору x
xs = [1e-10, 1e-7, 1e-4, 1e-3, 1e-2, 1e-1, 1.0, 10.0,
100.0, 1000.0, 1e6, 1e9]

print("x\t\tepsilon")
for x in xs:
    t = 0.5  # початкове значення кроку
    # зменшуємо t, доки додавання t ще змінює число x
    while x + t != x:
        t = t / 2.0
    eps = t  # мінімальний крок, що ще змінює x
    print(f"{x:>14.10g}\t{eps:.3e}")
```

Таким чином, при програмуванні необхідно враховувати існування «машинної» похибки. У зв'язку з цим повинен змінюватися підхід до використання дійсних чисел.

У практиці обчислень доцільно вводити в програму константу або змінну, значення якої не менше «машинної похибки». Позначимо цю константу, наприклад, t.

Для Python: машинна точність (epsilon) — це найменше число `eps`, для якого `1.0 + eps > 1.0`. Для типу `float` у Python (IEEE 754 double precision) стандартне значення: `sys.float_info.epsilon = 2**-52 ≈ 2.220446049250313e-16`.

Можна перевірити:

```
import sys
t = sys.float_info.epsilon
print(f"{t = }")  # t = 2.220446049250313e-16
t1 = 2**(-52)
print(f"{t1 = }")  # t1 = 2.220446049250313e-16
```

Також, аналогічно:

```
import sys

# вбудоване значення машинної точності для float
```

```
(IEEE 754 double)
eps_builtin = sys.float_info.epsilon
print(f"sys.float_info.epsilon = {eps_builtin:.18e}")

# перевірка визначення: 1.0 + eps > 1.0 і 1.0 + eps/2
== 1.0
one = 1.0
eps = eps_builtin
print("1.0 + eps > 1.0 ->", one + eps > one)
print("1.0 + eps/2 == 1.0 ->", one + eps/2 == one)

# наближене обчислення epsilon шляхом ділення навпіл
t = 1.0
while one + t > one:
    t = t / 2.0
eps_est = t * 2.0  # попереднє значення t —
мінімальний крок, що змінює 1.0
print(f"estimated epsilon = {eps_est:.18e}")
```

Рекомендовано застосовувати для порівняння дійсних чисел:

`math.isclose(a, b, rel_tol=..., abs_tol=...)`.

Пояснення: `math.isclose(a, b, rel_tol, abs_tol)` повертає `True`, якщо виконується:

`|a - b| ≤ max(rel_tol * max(|a|, |b|), abs_tol)`.

- `rel_tol` — відносний допуск (корисний для великих за модулем значень).
- `abs_tol` — абсолютний допуск (обов'язковий, коли значення поблизу нуля).

Рекомендації:

для значень порядку одиниці беріть `rel_tol` ~ `1e-9…1e-12`;
коли можливі значення близькі до нуля, задавайте ще й `abs_tol` ~ `1e-12…1e-15`.

Приклад (лінійний скрипт) використання `math.isclose`:

```
import math

# приклад 1: класична похибка 0.1 + 0.2 != 0.3 при
прямому порівнянні
a = 0.1 + 0.2
b = 0.3
print("a == b ->", a == b)  # очікувано False через
двійкове подання
```

```
print("isclose(a,b, rel_tol=1e-12) ->",
math.isclose(a, b, rel_tol=1e-12))

# приклад 2: порівняння з нулем — потрібен абсолютний
допуск
c = 1e-12
d = 0.0
print("c == d ->", c == d)  # False
print("isclose(c,d, abs_tol=1e-12) ->",
math.isclose(c, d, abs_tol=1e-12))  # True

# приклад 3: велике число з малою добавкою —
відносний допуск зручний
e = 1_000_000.0 + 1e-8
f = 1_000_000.0
print("e == f ->", e == f)  # False (різниця не нуль)
print("isclose(e,f, rel_tol=1e-12) ->",
math.isclose(e, f, rel_tol=1e-12))  # True

# приклад 4: дуже малі значення — один rel_tol не
спрацьовує, потрібен abs_tol
g = 1e-15
h = 0.0
print("isclose(g,h, rel_tol=1e-12) ->",
math.isclose(g, h, rel_tol=1e-12))       # False
print("isclose(g,h, rel_tol=1e-12, abs_tol=1e-15) -
>", math.isclose(g, h, rel_tol=1e-12, abs_tol=1e-15))
# True
```

Таблиця 3.3 – Запис операцій порівняння з дійсними змінними

Математичний запис операції	Запис в програмі на Python	Математичний запис операції	Запис в програмі на Python
x = 0	abs(x) < t або math.isclose(x, 0.0, abs_tol=t)	x = k	abs(x - k) < t або math.isclose(x, k, abs_tol=t)
x ≠ 0	abs(x) > t	x ≠ k	abs(x - k) > t
x < 0	x < -t	x < k	x < k - t
x > 0	x > t	x > k	x > k + t
x ≤ 0	x < t	x ≤ k	x < k + t
x ≥ 0	x > -t	x ≥ k	x > k - t

3.2 Завдання

Нижче наведені завдання з різним ступенем складності та відповідною максимальною кількістю балів, яку здобувач зможе отримати, виконавши обраний рівень складності у відповідності до критеріїв оцінювання.

ПОВНЕ ЗАВДАННЯ (максимум 10 балів):

Розробити блок-схему алгоритму розв’язання поставленої задачі та відповідну програму. Математична постановка задачі наведена в Додатку А. Номер варіанту обирається здобувачами згідно з номером в журналі академічної групи. Результати оформити у вигляді звіту. Приклад оформлення виконаного завдання наведений у Додатку Б.

СПРОЩЕНЕ ЗАВДАННЯ (максимум 6 балів): Постановка завдання, хід виконання та приклад коду неведені в Додатку В.

3.3 Питання для самоперевірки

1. Дайте визначення «лінійній» програмі.
2. Правила використання input()/print() та форматування результатів у Python.
3. Принципи читання та парсингу введення з роздільниками (пробіл, кома).
4. Оголошення змінних, констант. Основні типи даних у Python.
5. Порядок обчислень математичних виразів у Python (пріоритети операцій).
6. Дайте визначення «розгалуженій» програмі.
7. Правила використання if/elif/else.
8. Умовний вираз (тернарний оператор) у Python.
9. Дайте визначення поняттю «машинна точність».
10. Правила порівняння дійсних чисел у Python: abs, math.isclose, використання допусків.

4 ІНДИВІДУАЛЬНЕ ЗАВДАННЯ №2

4.1 Основні теоретичні положення

4.1.1 Тип *str*, представлення тексту, кодування Unicode

Основні властивості рядків:

– Рядок у Python має тип *str* і є незмінним (immutable) об'єктом. Це означає, що після створення рядка його не можна змінити. Будь-які операції, що здаються зміною рядка, насправді створюють новий рядок.

– Python 3 використовує Unicode (UCS/UTF-8), що дає можливість коректно працювати з українськими літерами, емодзі тощо. Кожен символ у рядку є кодовою точкою Unicode.

– Довжина рядка: *len(s)* — повертає кількість кодових точок Unicode у рядку.

Способи створення рядків. У Python існує кілька способів створення рядків залежно від потреб: одинарні або подвійні лапки для звичайних рядків, потрійні лапки для багаторядкових текстів, та raw-рядки для збереження спеціальних символів без обробки.

```
# Подвійні лапки (еквівалентні одинарним)
s2 = "Привіт"

# Потрійні лапки для багаторядкових рядків
s3 = """Багаторядковий
рядок з кількома
рядками тексту"""

# Потрійні одинарні лапки (також працюють)
s4 = '''Також
багаторядковий'''

# Raw-рядки (екрануючі послідовності не обробляються)
s5 = r"C:\new\folder\test.txt"

# Явне використання Unicode-символів
s6 = "Українська мова \u0456 емодж\u00ed \U0001F600"
```

Особливості незмінності. Оскільки рядки є незмінними, будь-яка спроба змінити окремий символ призведе до помилки. Для "зміни" рядка потрібно створити новий рядок на основі старого.

```
s = "Hello"
# s[0] = "h"  # Помилка: TypeError: 'str' object does
```

```
not support item assignment

# Щоб "змінити", створюємо новий рядок
s = "h" + s[1:]  # "hello"
```

4.1.2 Індексація, зрізи (slicing), ітерування

Індексація дозволяє отримати доступ до окремих символів рядка. Python підтримує як прямі індекси (від початку), так і від'ємні (від кінця).

– *Прямі індекси:* `s[i]` де `i` від `0` до `len(s)-1`

– *Від'ємні індекси:* від кінця рядка, `s[-1]` – останній символ, `s[-2]` – передостанній тощо.

```
s = "алгоритмика"
print(s[0])    # "а" (перший символ)
print(s[4])    # "р" (п'ятий символ, індекс 4)
print(s[-1])   # "а" (останній символ)
print(s[-3])   # "и" (третій з кінця)
```

Результат:

```
а
р
а
и
```

Зрізи (Slicing) дозволяють витягувати підрядки з рядка, вказуючи початок, кінець та крок. Це потужний інструмент для роботи з частинами рядків.

Синтаксис: `s[start:end:step]`

– `start` — початок зрізу (включно), за замовчуванням 0
– `end` — кінець зрізу (не включно), за замовчуванням len(s)
– `step` — крок, за замовчуванням 1

```
s = "алгоритмика"

# Базове використання зрізів
print(s[0:4])     # "алго" (символи з індексами 0,1,2
,3)
print(s[2:7])     # "горит"
print(s[:5])      # "алгор" (від початку до індексу 5
)
print(s[5:])      # "итмика" (від індексу 5 до кінця)

# Від'ємні індекси у зрізах
```

```
print(s[-3:])     # "ика" (останні 3 символи)
print(s[:-3])     # "алгоритм" (всі крім останніх 3)

# Параметр step (крок)
print(s[::2])     # "агртиа" (кожен другий символ)
print(s[1::2])    # "лоимк" (кожен другий, починаючи
з індексу 1)

# Розворот рядка (негативний крок)
print(s[::-1])    # "акимтирогла" (розворот рядка)
```

Результат:

```
алго
горит
алгор
итмика
ика
алгоритм
агртиа
лоимк
акимтирогла
```

Ітерування по символах. Цикли дозволяють обробляти кожен символ рядка по черзі. Існує кілька способів ітерування залежно від потреб: прямий обхід, обхід з індексами, або зворотний обхід.

```
s = "Python"

# Спосіб 1: Прямий обхід символів
for ch in s:
    print(ch, end=' ')  # P y t h o n

print()  # новий рядок

# Спосіб 2: Обхід з індексами через range
for i in range(len(s)):
    print(f"Індекс {i}: {s[i]}")

# Спосіб 3: Зворотний обхід за допомогою while
i = len(s) - 1
while i >= 0:
    print(s[i], end=' ')  # n o h t y P
    i -= 1
```

Результат:

```
P y t h o n
Індекс 0: P
Індекс 1: y
Індекс 2: t
Індекс 3: h
Індекс 4: o
Індекс 5: n
n o h t y P
```

4.1.3 Основні операції та методи рядків

Конкатенація та повторення. Конкатенація (з'єднання) рядків виконується оператором +, а повторення — оператором *. Для об'єднання списку рядків ефективніше використовувати метод *join()*.

```
# Конкатенація (з'єднання)
s1 = "Hello"
s2 = "World"
s3 = s1 + " " + s2  # "Hello World"

# Повторення
s4 = "Python! " * 3  # "Python! Python! Python! "

# Побудова рядка зі списку (ручне об'єднання)
words = ["Hello", "World", "from", "Python"]
sentence = ""
for i in range(len(words)):
    if i > 0:
        sentence += " "
    sentence += words[i]
print(sentence)  # "Hello World from Python"

# Використання вбудованого методу join (ефективніше)
sentence = " ".join(words)
```

Результат:

```
Hello World from Python
```

Методи пошуку дозволяють знаходити підрядки всередині рядка. Метод *find()* повертає індекс або -1, метод *count()* підраховує входження, а оператор *in* перевіряє наявність підрядка.

```
s = "Hello, World! Hello, Python!"

# find() - повертає індекс або -1
print(s.find("World"))      # 7
print(s.find("Java"))       # -1 (не знайдено)
print(s.find("Hello", 1))   # 14 (пошук з індексу 1)

# rfind() - пошук з кінця
print(s.rfind("Hello"))     # 14 (останнє входження)

# count() - підрахунок входжень
print(s.count("Hello"))     # 2
print(s.count("o"))         # 4

# Оператор in - перевірка наявності підрядка
if "World" in s:
    print("Знайдено 'World'")

if "Java" not in s:
    print("'Java' не знайдено")
```

Результат:

```
7
-1
14
14
2
4
Знайдено 'World'
'Java' не знайдено
```

Методи заміни та видалення. Метод *replace()* замінює всі входження підрядка на інший рядок. Можна також обмежити кількість замін. Для видалення підрядка замінюємо його на порожній рядок.

```
s = "Hello, World!"

# replace() - заміна всіх входжень
s2 = s.replace("World", "Python")  # "Hello, Python!"

# replace з обмеженням кількості замін
s3 = "aaa bbb aaa ccc aaa"
s4 = s3.replace("aaa", "xxx", 2)   # "xxx bbb xxx ccc
aaa"
```

```
# Видалення підрядка (заміна на порожній рядок)
s5 = s.replace(",", "")              # "Hello World!"
```

Методи розбиття. Метод *split()* розбиває рядок на список підрядків за роздільником. За замовчуванням роздільником є пробільні символи. Це корисно для обробки слів у реченні або даних у CSV-форматі.

```
# split() - розбиття за пробілами
s = "Hello World from Python"
words = s.split()  # ['Hello', 'World', 'from', 'Pyth
on']

# Доступ до окремих слів
print(words[0])    # "Hello"
print(words[1])    # "World"

# split() з вказаним роздільником
csv = "apple,banana,orange"
fruits = csv.split(',')  # ['apple', 'banana', 'orang
e']

# split() з обмеженням кількості розбиттів
text = "one:two:three:four"
parts = text.split(':', 2)  # ['one', 'two', 'three:f
our']

# Підрахунок слів
word_count = len(words)
print(f"Кількість слів: {word_count}")
```

Результат:

```
Hello
World
Кількість слів: 4
```

Методи зміни регістру дозволяють перетворювати літери рядка у верхній або нижній регістр, що корисно для порівнянь без урахування регістру або форматування тексту.

```
s = "Hello World"

# Нижній регістр
print(s.lower())        # "hello world"

# Верхній регістр
print(s.upper())        # "HELLO WORLD"
```

```
# Перша літера у верхньому регістрі
print(s.capitalize())   # "Hello world"

# Кожне слово з великої літери (title case)
print(s.title())        # "Hello World"

# Інвертування регістру
print(s.swapcase())     # "hELLO wORLD"
```

Результат:

```
hello world
HELLO WORLD
Hello world
Hello World
hELLO wORLD
```

Методи очищення пробілів. Методи *strip()*, *lstrip()* та *rstrip()* видаляють пробільні символи з країв рядка. Це корисно при обробці введених даних користувача.

```
s = "  Hello, World!  "

# strip() - видалення пробілів з обох кінців
print(s.strip())        # "Hello, World!"

# lstrip() - видалення зліва
print(s.lstrip())       # "Hello, World!  "

# rstrip() - видалення справа
print(s.rstrip())       # "  Hello, World!"

# strip з вказаними символами
s2 = "***Hello***"
print(s2.strip('*'))    # "Hello"
```

Результат:

```
Hello, World!
Hello, World!
  Hello, World!
Hello
```

Методи перевірки (is-methods) повертають *True* або *False* залежно від властивостей символів рядка. Вони корисні для валідації введених даних.

```
# Перевірка: всі символи - літери
print("abc".isalpha())          # True
print("abc123".isalpha())       # False

# Перевірка: всі символи - цифри
print("123".isdigit())          # True
print("12.3".isdigit())         # False

# Перевірка: всі символи - літери або цифри
print("abc123".isalnum())       # True
print("abc 123".isalnum())      # False

# Перевірка: всі символи - пробільні
print("   ".isspace())          # True
print(" a ".isspace())          # False

# Перевірка: всі літери малі
print("hello".islower())        # True
print("Hello".islower())        # False

# Перевірка: всі літери великі
print("HELLO".isupper())        # True
print("Hello".isupper())        # False

# Перевірка: рядок друкований
print("Hello".isprintable())    # True
print("Hello\n".isprintable()) # False
```

Результат:

```
True
False
True
False
True
False
True
False
True
False
True
False
True
False
```

Методи перевірки префіксів та суфіксів: *startswith()* та *endswith()* перевіряють, чи починається або закінчується рядок вказаним підрядком. Це зручно для фільтрації рядків за певними ознаками.

```
s = "Hello, World!"

# startswith() - перевірка префікса
print(s.startswith("Hello"))    # True
print(s.startswith("World"))    # False

# endswith() - перевірка суфікса
print(s.endswith("World!"))     # True
print(s.endswith("World"))      # False
```

Результат:

```
True
False
True
False
```

Форматування рядків дозволяє вставляти значення змінних у рядки. F-рядки (f-strings) — це найсучасніший та найзручніший спосіб форматування в Python 3.6+.

```
# f-рядки (рекомендовано)
name = "Анна"
age = 25
print(f"{name} має {age} років")  # "Анна має 25 рокі
в"
print(f"{name:>10}")  # Вирівнювання праворуч в 10 си
мволів
print(f"{age:05d}")   # Доповнення нулями: "00025"

# Метод format()
print("{} має {} років".format(name, age))
print("{name} має {age} років".format(name=name, age=
age))
```

Результат:

```
Анна має 25 років
      Анна
00025
Анна має 25 років
Анна має 25 років
```

Приклад: Базові перетворення

Нижче наведено комплексний приклад використання різних методів рядків для типових операцій обробки тексту.

```
s = "  Привіт, світ!  "

print(f"Довжина: {len(s)}") # Довжина з пробілами
print(f"Обрізано: '{s.strip()}'") # Видалення пробілів
print(f"Нижній регістр: {s.lower()}") # Перетворення
до нижнього регістру
print(f"Заміна: {s.replace('світ', 'Python')}") # Заміна
підрядка
print(f"Позиція 'Привіт': {s.find('Привіт')}") # Пошук
позиції
print(f"Кількість '!': {s.count('!')}") # Підрахунок
символів
```

Результат:

```
Довжина: 17
Обрізано: 'Привіт, світ!'
Нижній регістр:   привіт, світ!
Заміна:   Привіт, Python!
Позиція 'Привіт': 2
Кількість '!': 1
```

4.1.4 Порівняння рядків, лексикографічний порядок

Лексикографічне порівняння. Порівняння рядків в Python відбувається **лексикографічно** (як у словнику), за кодовими точками Unicode символ за символом зліва направо. Це означає, що порівняння відбувається посимвольно до першої відмінності.

```
# Базові порівняння
print("apple" < "banana")   # True (a < b)
print("cat" > "car")        # True (t > r)
print("123" < "45")         # True (1 < 4)

# Порівняння Unicode-кодів
print("a" < "b")            # True (ord('a') = 97, or
d('b') = 98)
print("A" < "a")            # True (ord('A') = 65, or
d('a') = 97)
```

```
# Оператори порівняння
print("abc" == "abc")     # Рівність: True
print("abc" != "xyz")     # Нерівність: True
print("abc" < "abd")      # Менше: True
print("abc" <= "abc")     # Менше або дорівнює: True
print("xyz" > "abc")      # Більше: True
print("xyz" >= "xyz")     # Більше або дорівнює: True
```

Результат:

```
True
True
True
True
True
True
True
True
True
True
True
```

Вплив регістру. При сортуванні рядків регістр літер має значення, оскільки великі літери мають менші коди Unicode, ніж малі. За замовчуванням великі літери йдуть перед малими.

```
strings = ["banana", "Apple", "cherry", "apple"]

# Сортування з урахуванням регістру (за замовчуванням
)
# Великі літери йдуть першими в ASCII/Unicode
sorted_strings = sorted(strings)
print(sorted_strings)  # ['Apple', 'apple', 'banana',
'cherry']

# Для сортування без урахування регістру треба перетв
орювати до нижнього регістру
# Ми розглянемо це вручну в наступних прикладах
```

Результат:

```
['Apple', 'apple', 'banana', 'cherry']
```

Сортування списків рядків: Функція *sorted()* повертає новий відсортований список, не змінюючи оригінальний. Параметр *reverse=True* дозволяє сортувати у зворотному порядку.

```
# Сортування у зростаючому порядку (лексикографічно)
words = ["cat", "elephant", "dog", "butterfly"]
sorted_words = sorted(words)
print(sorted_words)  # ['butterfly', 'cat', 'dog', 'e
lephant']

# Сортування у спадаючому порядку
sorted_words_desc = sorted(words, reverse=True)
print(sorted_words_desc)  # ['elephant', 'dog', 'cat'
, 'butterfly']
```

Результат:

```
['butterfly', 'cat', 'dog', 'elephant']
['elephant', 'dog', 'cat', 'butterfly']
```

4.1.5 Підрахунок символів за категоріями

Підрахунок цифр у рядку. Для підрахунку цифр використовуємо цикл та метод *isdigit()*, який повертає *True* для символів-цифр.

```
s = "Hello123World456"

# Підрахунок цифр
digit_count = 0
for ch in s:
    if ch.isdigit():
        digit_count += 1

print(f"Цифр: {digit_count}")  # 6
```

Результат:

```
Цифр: 6
```

Підрахунок літер у різних регістрах. Для підрахунку літер у різних регістрах використовуємо методи *islower()*, *isupper()* та *isalpha()*. Це корисно для аналізу та валідації текстових даних.

```
s = "Hello World Python"

# Підрахунок малих літер
lowercase_count = 0
for ch in s:
    if ch.islower():
```

```
            lowercase_count += 1

    # Підрахунок великих літер
    uppercase_count = 0
    for ch in s:
        if ch.isupper():
            uppercase_count += 1

    # Підрахунок всіх літер
    letter_count = 0
    for ch in s:
        if ch.isalpha():
            letter_count += 1

    print(f"Малих літер: {lowercase_count}")  # 13
    print(f"Великих літер: {uppercase_count}")  # 3
    print(f"Всього літер: {letter_count}") # 16
```

Результат:

```
Малих літер: 13
Великих літер: 3
Всього літер: 16
```

Підрахунок пунктуації. ASCII-пунктуація включає стандартні розділові знаки. Для перевірки приналежності символу до пунктуації використовуємо оператор *in* з рядком пунктуаційних символів.

```
    # Символи ASCII-пунктуації
    punctuation = "!\"#$%&'()*+,-./:;<=>?@[\\]^_`{|}~"

    s = "Hello, World! How are you?"

    punct_count = 0
    for ch in s:
        if ch in punctuation:
            punct_count += 1

    print(f"Пунктуації: {punct_count}")  # 3
```

Результат:

```
Пунктуації: 3
```

Підрахунок недрукованих символів. Недруковані символи (наприклад, *\n*, *\t*) можна виявити за допомогою методу *isprintable()*. Це важливо при обробці файлів та введених даних.

```
s = "Hello\nWorld\t!"

non_printable_count = 0
for ch in s:
    if not ch.isprintable():
        non_printable_count += 1

print(f"Недрукованих: {non_printable_count}")  # 2 (\
n та \t)
```

Результат:

```
Недрукованих: 2
```

Підрахунок слів. Існує два основні способи підрахунку слів: простий (через *split()*) та ручний (через перевірку кожного символу). Ручний спосіб дає більше контролю над визначенням "слова".

```
s = "Hello World from Python"

# Спосіб 1: Використання split()
words = s.split()
word_count = len(words)
print(f"Слів: {word_count}")  # 4

# Спосіб 2: Ручний підрахунок
word_count = 0
in_word = False

for ch in s:
    if ch.isalpha():
        if not in_word:
            word_count += 1
            in_word = True
    else:
        in_word = False

print(f"Слів (вручну): {word_count}")  # 4
```

Результат:

```
Слів: 4
Слів (вручну): 4
```

4.2 Приклади програм

4.2.1 Базові операції з рядками

Приклад 1: Довжина рядка

Функція `len()` повертає кількість символів (кодових точок Unicode) у рядку. Це одна з найбазовіших операцій при роботі з рядками, яка часто використовується для перевірки введених даних або організації циклів.

```
# Програма для визначення довжини рядка

s = input("Введіть рядок: ")
length = len(s)
print(f"Довжина: {length}")
```

Результат:

```
Введіть рядок: Hello, World!
Довжина: 13
```

Приклад 2: Підрахунок входжень символу

Для підрахунку кількості входжень символу у рядок можна використати вбудований метод `count()` або реалізувати власний підрахунок через цикл. Метод `count()` швидший, але цикл демонструє базову логіку підрахунку.

Спосіб 1: Використання методу count()

```
# Програма для підрахунку входжень символу (використання методу count)

s = input("Введіть рядок: ")
ch = input("Введіть символ для підрахунку: ")

if len(ch) != 1:
    print("Будь ласка, введіть рівно один символ")
else:
    count = s.count(ch)
    print(f"Кількість входжень: {count}")
```

Спосіб 2: Використання циклу

```
# Програма для підрахунку входжень символу (використання циклу)

s = input("Введіть рядок: ")
ch = input("Введіть символ для підрахунку: ")
```

```
if len(ch) != 1:
    print("Будь ласка, введіть рівно один символ")
else:
    count = 0
    for c in s:
        if c == ch:
            count += 1
    print(f"Кількість входжень: {count}")
```

Результат:

```
Введіть рядок: Hello, World!
Введіть символ для підрахунку: o
Кількість входжень: 2
```

Приклад 3: Розворот рядка

Розворот рядка — це створення нового рядка з символами у зворотному порядку. Найпростіший спосіб — використати зріз з негативним кроком [::-1]. Альтернативний спосіб — побудувати новий рядок через цикл, що проходить по символах у зворотному порядку.

Спосіб 1: Використання зрізів

```
# Програма для розвороту рядка (використання зрізів)

s = input("Введіть рядок: ")
reversed_s = s[::-1]
print(f"Розвернутий: {reversed_s}")
```

Спосіб 2: Використання циклу

```
# Програма для розвороту рядка (використання циклу)

s = input("Введіть рядок: ")
reversed_s = ""

for i in range(len(s) - 1, -1, -1):
    reversed_s += s[i]

print(f"Розвернутий: {reversed_s}")
```

Результат:

```
Введіть рядок: Python
Розвернутий: nohtyP
```

Приклад 4: Кількість слів у рядку

Підрахунок слів можна виконати методом `split()`, який розбиває рядок за пробілами, або вручну, відстежуючи переходи між літерними та нелітерними символами. Ручний спосіб дає краще розуміння алгоритму обробки тексту.

Спосіб 1: Простий split()

```
# Програма для підрахунку слів (використання split)

s = input("Введіть рядок: ")
words = s.split()
word_count = len(words)
print(f"Кількість слів: {word_count}")
```

Спосіб 2: З використанням циклу

```
# Програма для підрахунку слів (використання циклу)

s = input("Введіть рядок: ")
count = 0
in_word = False

for ch in s:
    if ch.isalpha():
        if not in_word:
            count += 1
            in_word = True
    else:
        in_word = False

print(f"Кількість слів: {count}")
```

Результат:

```
Введіть рядок: Hello World from Python
Кількість слів: 4
```

Приклад 5: Кількість різних символів

Для підрахунку унікальних символів будуємо список, який містить лише символи, що зустрілися вперше. Перед додаванням кожного символу перевіряємо, чи він вже є у списку.

```
# Програма для підрахунку різних символів

s = input("Введіть рядок: ")

# Побудова списку унікальних символів
unique_chars = []
for ch in s:
    if ch not in unique_chars:
        unique_chars.append(ch)

print(f"Кількість унікальних символів: {len(unique_ch
ars)}")
print(f"Унікальні символи: {unique_chars}")
```

Результат:

```
Введіть рядок: Hello, World!
Кількість унікальних символів: 10
Унікальні символи: ['H', 'e', 'l', 'o', ',', ' ', 'W'
, 'r', 'd', '!']
```

4.2.2 Пошук та заміна підрядків

Приклад 6: Позиція першого входження підрядка

Метод `find()` повертає індекс першого входження підрядка або -1, якщо підрядок не знайдено. Це базова операція для пошуку тексту всередині рядка.

```
# Програма для пошуку позиції першого входження підрядка

s = input("Введіть рядок: ")
sub = input("Введіть підрядок для пошуку: ")

pos = s.find(sub)
if pos == -1:
    print("Підрядок не знайдено")
else:
    print(f"Підрядок знайдено на позиції: {pos}")
```

Результат:

```
Введіть рядок: Hello, World!
Введіть підрядок для пошуку: World
Підрядок знайдено на позиції: 7
```

Приклад 7: Заміна всіх входжень підрядка

Метод `replace()` створює новий рядок, де всі входження вказаного підрядка замінені на інший рядок. Оригінальний рядок залишається незмінним через властивість незмінності рядків.

```python
# Програма для заміни всіх входжень підрядка

s = input("Введіть рядок: ")
old = input("Підрядок для заміни: ")
new = input("Замінити на: ")

result = s.replace(old, new)
print(f"Результат: {result}")
```

Результат:

```
Введіть рядок: Hello, Hello, World!
Підрядок для заміни: Hello
Замінити на: Hi
Результат: Hi, Hi, World!
```

Приклад 8: Видалення всіх входжень підрядка

Видалення підрядка реалізується як заміна на порожній рядок. Це ефективний спосіб очищення тексту від небажаних елементів.

```python
# Програма для видалення всіх входжень підрядка

s = input("Введіть рядок: ")
sub = input("Підрядок для видалення: ")

result = s.replace(sub, "")
print(f"Результат: {result}")
```

Результат:

```
Введіть рядок: Hello, Hello, World!
Підрядок для видалення: Hello,
Результат: World!
```

4.2.3 Перевірка паліндромів

Приклад 9: Перевірка паліндрому (базова)

Паліндром — це рядок, який читається однаково зліва направо і справа наліво. Найпростіша перевірка — порівняти рядок з його розворотом.

```
# Програма для перевірки паліндрому (базова)

s = input("Введіть рядок: ")
reversed_s = s[::-1]

if s == reversed_s:
    print("Рядок є паліндромом")
else:
    print("Рядок не є паліндромом")
```

Результат:

```
Введіть рядок: radar
Рядок є паліндромом
```

Приклад 10: Перевірка паліндрому (без пробілів і регістру)

Для коректної перевірки паліндромів часто ігнорують пробіли та регістр літер. Спочатку очищаємо рядок від пробілів та перетворюємо до нижнього регістру, потім порівнюємо з розворотом.

```
# Програма для перевірки паліндрому (без пробілів і р
егістру)

s = input("Введіть рядок: ")

# Видалення пробілів та перетворення до нижнього регі
стру
cleaned = ""
for ch in s:
    if ch != " ":
        cleaned += ch
cleaned = cleaned.lower()

# Розворот очищеного рядка
reversed_cleaned = cleaned[::-1]

if cleaned == reversed_cleaned:
```

```
        print("Рядок є паліндромом")
    else:
        print("Рядок не є паліндромом")
```

Результат:

```
Введіть рядок: А роза упала на лапу Азора
Рядок є паліндромом
```

Приклад 11: Перевірка паліндрому (тільки літери)

У найбільш строгій версії перевірки паліндрому враховуються тільки літери, ігноруючи всі інші символи (пробіли, пунктуацію). Перевірка виконується порівнянням символів з обох кінців очищеного рядка.

```
# Програма для перевірки паліндрому (тільки літери)

s = input("Введіть рядок: ")

# Залишаємо тільки літери та перетворюємо до нижнього регістру
cleaned = ""
for ch in s:
    if ch.isalpha():
        cleaned += ch.lower()

# Перевірка паліндрому
is_palindrome = True
length = len(cleaned)
for i in range(length // 2):
    if cleaned[i] != cleaned[length - 1 - i]:
        is_palindrome = False
        break

if is_palindrome:
    print("Рядок є паліндромом")
else:
    print("Рядок не є паліндромом")
```

4.2.4 Робота з кількома рядками

Приклад 12: Найдовший спільний префікс

Спільний префікс — це найдовший підрядок, який знаходиться на початку обох рядків. Алгоритм порівнює символи з однаковими індексами доти, доки вони збігаються або доки не досягнуто кінця коротшого рядка.

```
# Програма для пошуку найдовшого спільного префікса

a = input("Введіть перший рядок: ")
b = input("Введіть другий рядок: ")

# Знаходження спільного префікса
min_len = len(a) if len(a) < len(b) else len(b)
prefix_len = 0

for i in range(min_len):
    if a[i] == b[i]:
        prefix_len += 1
    else:
        break

if prefix_len > 0:
    prefix = a[:prefix_len]
    print(f"Спільний префікс: {prefix}")
else:
    print("Немає спільного префікса")
```

Результат:

```
Введіть перший рядок: flower
Введіть другий рядок: flow
Спільний префікс: flow
```

Приклад 13: Найдовший спільний суфікс

Спільний суфікс — це найдовший підрядок, який знаходиться в кінці обох рядків. Алгоритм порівнює символи з кінця обох рядків, рухаючись до початку.

```
# Програма для пошуку найдовшого спільного суфікса

a = input("Введіть перший рядок: ")
b = input("Введіть другий рядок: ")

# Знаходження спільного суфікса
len_a = len(a)
len_b = len(b)
```

```
suffix_len = 0

i = 1
while i <= len_a and i <= len_b:
    if a[-i] == b[-i]:
        suffix_len += 1
        i += 1
    else:
        break

if suffix_len > 0:
    suffix = a[len_a - suffix_len:]
    print(f"Спільний суфікс: {suffix}")
else:
    print("Немає спільного суфікса")
```

Результат:

```
Введіть перший рядок: testing
Введіть другий рядок: running
Спільний суфікс: ing
```

4.2.5 Підрахунок символів за категоріями

Приклад 14: Підрахунок різних типів символів

Для аналізу складу рядка корисно підрахувати різні категорії символів: цифри, малі/великі літери, пробіли, пунктуацію. Це виконується за допомогою циклу та методів перевірки типу символів.

```
# Програма для підрахунку різних типів символів

s = input("Введіть рядок: ")

# Підрахунок різних категорій
digits = 0
lowercase = 0
uppercase = 0
letters = 0
spaces = 0
punctuation = "!\"#$%&'()*+,-./:;<=>?@[\\]^_`{|}~"
punct = 0

for ch in s:
    if ch.isdigit():
```

```
            digits += 1
        if ch.islower():
            lowercase += 1
        if ch.isupper():
            uppercase += 1
        if ch.isalpha():
            letters += 1
        if ch.isspace():
            spaces += 1
        if ch in punctuation:
            punct += 1

print(f"Цифр: {digits}")
print(f"Малих літер: {lowercase}")
print(f"Великих літер: {uppercase}")
print(f"Всього літер: {letters}")
print(f"Пробілів: {spaces}")
print(f"Пунктуації: {punct}")
```

Результат:

```
Введіть рядок: Hello, World! Python3 123
Цифр: 3
Малих літер: 11
Великих літер: 3
Всього літер: 17
Пробілів: 2
Пунктуації: 2
```

Приклад 15: Частота символів

Для підрахунку частоти кожного символу використовуємо два паралельні списки: один для унікальних символів, інший для їхніх лічильників. При кожному символі шукаємо його в списку унікальних: якщо знайдено — збільшуємо лічильник, якщо ні — додаємо новий символ з лічильником 1.

```
# Програма для підрахунку частоти символів

s = input("Введіть рядок: ")

# Побудова списків унікальних символів та їхніх лічил
ьників
unique_chars = []
char_counts = []
```

```
for char in s:
    # Перевірка, чи символ вже є у списку
    found = False
    for i in range(len(unique_chars)):
        if unique_chars[i] == char:
            char_counts[i] += 1
            found = True
            break

    # Якщо не знайдено, додаємо новий символ
    if not found:
        unique_chars.append(char)
        char_counts.append(1)

# Виведення частот
print("Частота символів:")
for i in range(len(unique_chars)):
    print(f"'{unique_chars[i]}': {char_counts[i]}")
```

Результат:

```
Введіть рядок: Hello
Частота символів:
'H': 1
'e': 1
'l': 2
'o': 1
```

4.2.6 Приклади сортування рядків

Приклад 16: Алфавітне (лексикографічне) сортування

Лексикографічне сортування упорядковує рядки за алфавітом, порівнюючи символи посимвольно за їхніми Unicode-кодами. Функція `sorted()` виконує таке сортування за замовчуванням, враховуючи регістр літер (великі літери йдуть перед малими).

```
# Програма для алфавітного сортування рядків

# Введення рядків
n = int(input("Кількість рядків: "))
strings = []
for i in range(n):
```

```
    s = input(f"Рядок {i+1}: ")
    strings.append(s)

# Сортування за алфавітом (з урахуванням регістру)
sorted_strings = sorted(strings)

print("\nВідсортовано (з урахуванням регістру):")
for i in range(len(sorted_strings)):
    print(f"{i+1}. {sorted_strings[i]}")

# Сортування у зворотному порядку
sorted_strings_desc = sorted(strings, reverse=True)

print("\nВідсортовано (в зворотному порядку):")
for i in range(len(sorted_strings_desc)):
    print(f"{i+1}. {sorted_strings_desc[i]}")
```

Результат:

```
Кількість рядків: 4
Рядок 1: яблуко
Рядок 2: абрикос
Рядок 3: Груша
Рядок 4: банан

Відсортовано (з урахуванням регістру):
1. Груша
2. абрикос
3. банан
4. яблуко

Відсортовано (в зворотному порядку):
1. яблуко
2. банан
3. абрикос
4. Груша
```

Приклад 17: Сортування за кількістю слів

Для сортування за користувацьким критерієм (наприклад, кількістю слів) спочатку обчислюємо цей критерій для кожного рядка у окремий список. Потім застосовуємо алгоритм бульбашкового сортування (Bubble Sort), який порівнює критерії сусідніх елементів і міняє їх місцями, якщо вони в неправильному порядку. При обміні також міняємо місцями відповідні рядки.

```python
# Програма для сортування за кількістю слів

# Введення рядків
n = int(input("Кількість рядків: "))
strings = []
for i in range(n):
    s = input(f"Рядок {i+1}: ")
    strings.append(s)

# Обчислення кількості слів для кожного рядка
word_counts = []
for s in strings:
    words = s.split()
    word_counts.append(len(words))

# Бульбашкове сортування за кількістю слів
for i in range(len(strings) - 1):
    for j in range(len(strings) - 1 - i):
        if word_counts[j] > word_counts[j + 1]:
            # Обмін рядків
            strings[j], strings[j + 1] = strings[j +
1], strings[j]
            # Обмін лічильників
            word_counts[j], word_counts[j + 1] = word
_counts[j + 1], word_counts[j]

print("\nВідсортовано за кількістю слів (зростання):"
)
for i in range(len(strings)):
    print(f"{i+1}. ({word_counts[i]} слів) {strings[i
]}")
```

Результат:

```
Кількість рядків: 4
Рядок 1: Hello World
Рядок 2: Python
Рядок 3: Programming in Python is fun
Рядок 4: Test string here

Відсортовано за кількістю слів (зростання):
1. (1 слів) Python
2. (2 слів) Hello World
3. (3 слів) Test string here
4. (5 слів) Programming in Python is fun
```

Приклад 18: Сортування за кількістю цифр

Сортування за кількістю цифр демонструє той самий підхід: обчислюємо критерій (кількість цифр) для кожного рядка циклом з перевіркою `isdigit()`, потім застосовуємо бульбашкове сортування за цим критерієм.

```
# Програма для сортування за кількістю цифр

# Введення рядків
n = int(input("Кількість рядків: "))
strings = []
for i in range(n):
    s = input(f"Рядок {i+1}: ")
    strings.append(s)

# Обчислення кількості цифр для кожного рядка
digit_counts = []
for s in strings:
    count = 0
    for ch in s:
        if ch.isdigit():
            count += 1
    digit_counts.append(count)

# Бульбашкове сортування за кількістю цифр (зростання
)
for i in range(len(strings) - 1):
    for j in range(len(strings) - 1 - i):
        if digit_counts[j] > digit_counts[j + 1]:
            # Обмін рядків
            strings[j], strings[j + 1] = strings[j +
1], strings[j]
            # Обмін лічильників
            digit_counts[j], digit_counts[j + 1] = di
git_counts[j + 1], digit_counts[j]

print("\nВідсортовано за кількістю цифр (зростання):"
)
for i in range(len(strings)):
    print(f"{i+1}. ({digit_counts[i]} цифр) {strings[
i]}")
```

Результат:

```
Відсортовано за кількістю цифр (зростання):
1. (0 цифр) test
2. (3 цифр) abc123
3. (3 цифр) Python3.11
4. (4 цифр) 2024 year
Кількість рядків: 4
Рядок 1: abc123
Рядок 2: test
Рядок 3: Python3.11
Рядок 4: 2024 year

Відсортовано за кількістю цифр (зростання):
1. (0 цифр) test
2. (3 цифр) Python3.11
3. (3 цифр) abc123
4. (4 цифр) 2024 year
```

Приклад 19: Сортування за довжиною першого слова

Для сортування за довжиною першого слова розбиваємо кожен рядок методом `split()` та беремо довжину першого елемента (якщо він існує). Потім виконуємо бульбашкове сортування за цими довжинами.

```python
# Програма для сортування за довжиною першого слова

# Введення рядків
n = int(input("Кількість рядків: "))
strings = []
for i in range(n):
    s = input(f"Рядок {i+1}: ")
    strings.append(s)

# Обчислення довжини першого слова для кожного рядка
first_word_lengths = []
for s in strings:
    words = s.split()
    if len(words) > 0:
        first_word_lengths.append(len(words[0]))
    else:
        first_word_lengths.append(0)

# Бульбашкове сортування за довжиною першого слова (з
ростання)
```

```
        for i in range(len(strings) - 1):
            for j in range(len(strings) - 1 - i):
                if first_word_lengths[j] > first_word_lengths
        [j + 1]:
                    # Обмін рядків
                    strings[j], strings[j + 1] = strings[j +
        1], strings[j]
                    # Обмін довжин
                    first_word_lengths[j], first_word_lengths
        [j + 1] = first_word_lengths[j + 1], first_word_lengt
        hs[j]

        print("\nВідсортовано за довжиною першого слова (зрос
        тання):")
        for i in range(len(strings)):
            print(f"{i+1}. (довжина {first_word_lengths[i]})
        {strings[i]}")
```

4.3 Завдання

Розробити програму, що виконує введення, сортування та виведення рядків згідно з варіантом таблиці Додатку Г.

Вимоги:

- програма запитує кількість рядків n;
- зчитує n рядків від користувача;
- запитує номер варіанту;
- обчислює необхідний критерій для кожного рядка;
- сортує рядки відповідно до критерію варіанту методом "бульбашки" (Bubble Sort);
- виводить відсортовані рядки.

Обмеження:

- використовувати тільки прості цикли (for, while);
- використовувати умовні оператори (if, elif, else);
- використовувати списки для зберігання рядків та обчислених критеріїв;
- для сортування використовувати метод "бульбашки" (Bubble Sort).

4.4 Питання для самоперевірки

Основи роботи з рядками:

1. Що таке тип *str* у Python та яка його основна властивість?
2. Що означає незмінність (immutability) рядків та які наслідки це має для програмування?
3. Які існують способи створення рядків у Python? Коли використовувати потрійні лапки?
4. Що таке Unicode та як Python 3 працює з ним?
5. Як отримати довжину рядка? Що повертає функція *len()*?
6. Як отримати доступ до окремого символу в рядку?
7. Що таке індексація рядків? Як працюють від'ємні індекси?
8. Що таке зрізи (slicing) та як вони працюють?
9. Як перевернути рядок за допомогою зрізів?
10. Як ітерувати по символах рядка за допомогою циклу *for*?

Методи рядків:

1. Для чого використовується метод *find()* та яке значення він повертає?
2. Для чого використовується метод *replace()* та як він працює? Чи змінює він оригінальний рядок?
3. Для чого використовується метод *split()* та що він повертає?
4. Які методи використовуються для зміни регістру рядка?
5. Що робить метод *strip()* та чим він відрізняється від *lstrip()* і *rstrip()*?
6. Для чого використовуються методи *startswith()* та *endswith()*?
7. Що таке метод *count()* та що він повертає?
8. Що робить оператор *in* при роботі з рядками?
9. Як об'єднати список рядків в один рядок?
10. Чим відрізняються методи *lower()* та *upper()*?

Перевірка символів:

1. Які методи використовуються для перевірки типу символів?
2. Що повертає метод *isalpha()* і коли він повертає *True*?
3. Що повертає метод *isdigit()* і які символи він розпізнає?
4. Що повертає метод *isspace()* і які символи він розпізнає як пробільні?
5. Що повертає метод *isalnum()* і для чого він використовується?
6. Як перевірити, чи складається рядок тільки з малих літер?
7. Як перевірити, чи складається рядок тільки з великих літер?
8. Що таке метод *isprintable()* та які символи він вважає друкованими?

Порівняння та сортування:

1. Як порівнюються рядки в Python? Що таке лексикографічне порівняння?
2. Чи впливає регістр літер на порівняння рядків?
3. Які оператори використовуються для порівняння рядків?
4. Як відсортувати список рядків в алфавітному порядку?
5. Що робить параметр *reverse=True* у функції *sorted()*?
6. Як працює метод бульбашкового сортування (Bubble Sort)?
7. Що таке критерій сортування?
8. Як відсортувати рядки за довжиною, використовуючи Bubble Sort?
9. Як відсортувати рядки за кількістю певних символів (наприклад, цифр)?
10. У чому перевага використання функції *sorted()* порівняно з ручним сортуванням?

Цикли та списки:

1. Як створити порожній список у Python?
2. Як додати елемент до списку?
3. Як отримати доступ до елемента списку за індексом?
4. Як ітерувати по списку за допомогою циклу *for*?
5. Як використати цикл *for* з *range()* для обходу списку за індексами?
6. Як поміняти місцями два елементи списку?
7. Що таке вкладені цикли і коли вони використовуються?

Підрахунок та обробка рядків:

1. Як підрахувати кількість цифр у рядку за допомогою циклу?
2. Як підрахувати кількість літер у рядку?
3. Як підрахувати кількість малих/великих літер у рядку?
4. Що таке пунктуація і як її підрахувати?
5. Як підрахувати кількість слів у рядку?
6. Як видалити всі пробіли з рядка?
7. Як знайти спільний префікс двох рядків?
8. Як знайти спільний суфікс двох рядків?
9. Як перевірити, чи є рядок паліндромом (ігноруючи пробіли та регістр)?
10. Як знайти кількість унікальних символів у рядку за допомогою списку?

5 ПЕРЕЛІК РЕКОМЕНДОВАНИХ ДЖЕРЕЛ

1. Das U., Lawson A., Norouzi N. Introduction to Python Programming. OpenStax, 2024. https://read.kortext.com/library/books/3213515
2. Sundnes J. Introduction to Scientific Programming with Python. Cham : Springer International Publishing, 2020. URL: https://doi.org/10.1007/978-3-030-50356-7 https://read.kortext.com/library/books/977939
3. Mayer C. Art of Clean Code: Best Practices to Eliminate Complexity and Simplify Your Life. No Starch Press, Incorporated, 2022.
4. Васильєв О. Програмування мовою Python. Навч. кн. - Богдан, 2019. 504 с.
5. Висоцька В. А., Оборська О. В. Python : алгоритмізація та програмування : навч. посібник. Львів : «Новий Світ-2000», 2025. 514 с.
6. Ceder N. Quick Python Book. 4th ed. Manning Publications Co. LLC, 2025. 580 p.
7. Lutz M. Learning Python. 5th ed. Sebastopol : O'Reilly Media, 2013. 1648 p.
8. Downey A. B. Think Python: How to Think Like a Computer Scientist. 2nd ed. Sebastopol : O'Reilly Media, 2015. 292 p.
9. Sweigart A. Automate the Boring Stuff with Python: Practical Programming for Total Beginners. 2nd ed. San Francisco : No Starch Press, 2019. 592 p.
10. Beazley D., Jones B. K. Python Cookbook. 3rd ed. Sebastopol : O'Reilly Media, 2013. 706 p.

Офіційна документація Python

11. Python 3 Standard Library — Text Sequence Type (str). *Python Documentation*. URL: https://docs.python.org/3/library/stdtypes.html#text-sequence-type-str (дата звернення: 10.02.2026).
12. Python 3 String Methods. *Python Documentation*. URL: https://docs.python.org/3/library/stdtypes.html#string-methods (дата звернення: 10.02.2026).
13. Python 3 — Built-in Functions. *Python Documentation*. URL: https://docs.python.org/3/library/functions.html (дата звернення: 10.02.2026).
14. Python 3 Tutorial — Strings. *Python Documentation*. URL: https://docs.python.org/3/tutorial/introduction.html#strings (дата звернення: 10.02.2026).

Навчальні ресурси

15. Strings and Character Data in Python. *Real Python*. URL: https://realpython.com/python-strings/ (дата звернення: 10.02.2026).
16. Python String Formatting Best Practices. *Real Python*. URL: https://realpython.com/python-string-formatting/ (дата звернення: 10.02.2026).
17. Python String Methods. *W3Schools*. URL: https://www.w3schools.com/python/python_ref_string.asp (дата звернення: 10.02.2026).
18. Python Loops. *W3Schools*. URL: https://www.w3schools.com/python/python_for_loops.asp (дата звернення: 10.02.2026).
19. Python String. *GeeksforGeeks*. URL: https://www.geeksforgeeks.org/python-string/ (дата звернення: 10.02.2026).
20. Python String Methods. *Programiz*. URL: https://www.programiz.com/python-programming/methods/string (дата звернення: 10.02.2026).

Алгоритми сортування

21. Python Sorting. *W3Schools*. URL: https://www.w3schools.com/python/python_lists_sort.asp (дата звернення: 10.02.2026).
22. Bubble Sort. *GeeksforGeeks*. URL: https://www.geeksforgeeks.org/bubble-sort/ (дата звернення: 10.02.2026).
23. Sorting Algorithms Visualization. *Visualgo*. URL: https://visualgo.net/en/sorting (дата звернення: 10.02.2026).

Додаткові матеріали

24. Visualize Python Code Execution. *Python Tutor*. URL: https://pythontutor.com/ (дата звернення: 10.02.2026).
25. Unicode and Character Encodings in Python. *Real Python*. URL: https://realpython.com/python-encodings-guide/ (дата звернення: 10.02.2026).
26. Python PEP 8 — Style Guide for Python Code. *Python Enhancement Proposals*. URL: https://peps.python.org/pep-0008/ (дата звернення: 10.02.2026).

Практика та задачі

27. String Problems. *LeetCode*. URL: https://leetcode.com/tag/string/ (дата звернення: 10.02.2026).
28. Python String Challenges. *HackerRank*. URL: https://www.hackerrank.com/domains/python?filters%5Bsubdomains%5D%5B%5D=py-strings (дата звернення: 10.02.2026).
29. Python String Kata. *Codewars*. URL: https://www.codewars.com/kata/search/python?q=string (дата звернення: 10.02.2026).

ДОДАТОК А.
МАТЕМАТИЧНА ПОСТАНОВКА ЗАДАЧІ ІНДИВІДУАЛЬНОГО ЗАВДАННЯ №1

Номер варіанту	Функція
1	$Y = \frac{\sqrt{\lvert \ln x - e^{2.3x} \rvert}}{\sqrt[3]{tg^2 x - 2^x}}$ + 1.3 x7
2	$Y = \sqrt[3]{\frac{\log_3 6x - e^x}{x^2 - 0.8} + x^9}$
3	$Y = \frac{\sqrt[3]{\ln x - 0.85x^{0.1}}}{3^{-x} + \ln\lvert \sin 2x \rvert}$
4	$Y = 3^{\cos x} - 7.5x - \sqrt[3]{\sin x - 2^x} + ctg^3 x/3$
5	$Y = \frac{7.3x^3 + \sqrt[5]{tgx}}{x^3 - 0.8x - 0.75}$+ ln (2x-3)
6	$Y = \frac{x^{-x\sin^2 x} + \ln 6.23x^2}{\sqrt[3]{x^2 - 24.8}}$ - ctg (2x/3)
7	$Y = \frac{\sqrt{\lvert 1.5\ln x - 2^x \rvert}}{\sqrt[3]{tg3x + 2x}}$
8	$Y = e^{-\cos x} - \sqrt{e^x - tg^3 x} - \frac{1}{\lg 0.5x}$
9	$y = \sqrt[5]{ctg^3 4x} \times \frac{3^{-x\cos x} + \lg\lvert 6.3 - x \rvert}{x^2 + \sin 2x}$
10	$y = \frac{\sqrt{\lg\lvert x - 0.8 \rvert + e^x}}{e^{-x} - ctg\ x} + \sqrt[3]{e^5 - 1/x}$
11	$y = \frac{2^{\sin x} - \sqrt[3]{e^x - 2.85}}{e^{-x} - \sin x}$ +sin9x
12	$Y = \frac{(6.35x^3 - e^x)(x - ctg4x)}{\sqrt[5]{3.5x - \ln x}}$
13	$Y = \frac{\sin 2x \cdot \cos 2x \cdot e^x}{\sqrt[7]{x^3 - 0.58\sin x}}$ + log4(tgx-3)
14	$Y = \frac{\sqrt{e^x + 5\sin x}}{x^2 - 0.8x - 0.25} + \sqrt[3]{tg0.1x}$
15	$Y = \frac{\sqrt[5]{\ln(x - 0.8)}}{2^{-\sin 2x} - e^x}$ -ctg3x7

Номер варіанту	Функція
16	$Y = \frac{3.5^{-x} - \lg 2\lvert x\rvert}{\sqrt[3]{3x^2 - 27.5e^x}} + ctg4x/3$
17	$Y = \frac{\sqrt[3]{\ln x + e^{2.5x} - \sin^2 x}}{\sqrt[7]{(x-0.8)(x+9.3)}}$
18	$Y = \frac{e^{-\cos x} + 3^{0.5x}}{\sqrt[3]{\ln x - 0.7x^2}} + \sqrt{\sin^3 x}$
19	$Y = \frac{2^{-\cos x} + \sqrt[3]{(0.75 - x^3)\cdot x}}{2^x - 3\cos x}$
20	$Y = \frac{\sqrt[3]{9.5x^3 - 2^{(x^2)}}}{(2^x - 3\cos^2 x)^3}$
21	$Y = \frac{\sqrt[3]{21\lg(x-3) - e^x}}{\sec^3 x - 2ctg^2 2x}$
22	$Y = \frac{\sqrt[5]{2^{3x} - tg^3 x + e^x}}{\sqrt{2x^3 - \sin^2 x}}$
23	$Y = \frac{(\lvert\ln x - e^x\rvert)^x}{\sqrt[3]{(tgx - 2^x)(x+1)}}$

ДОДАТОК Б.
ПРИКЛАД ОФОРМЛЕННЯ ІНДИВІДУАЛЬНОГО ЗАВДАННЯ №1

Написати програму, яка запитує значення *x* і обчислює функцію:

$$Y = \frac{2^{-\cos 2x} - x\sqrt[3]{0.75 - x^2}}{\ln|0.8 - x| - x^2}$$

Визначимо ОДЗ функції:

1) $0.8-x \neq 0$;

2) $\ln|0.8-x|-x^2 \neq 0$.

Блок-схема алгоритму розв'язання задачі наведена на рис. Б.1.

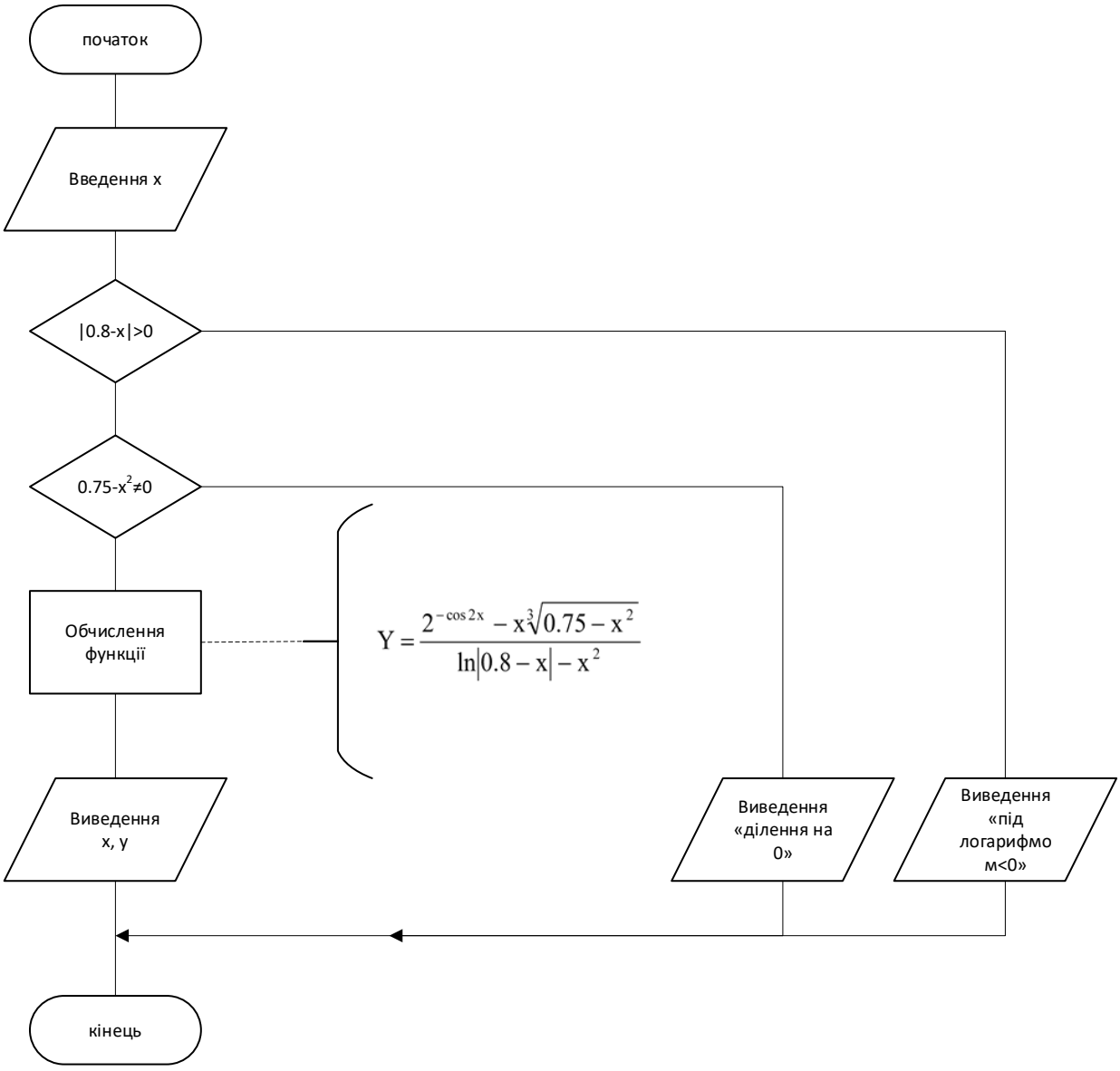

Рисунок Б.1 – Блок-схема алгоритму розв'язання задачі

Код програми:

```
    import math

def cbrt(value: float) -> float:
    # кубічний корінь з урахуванням знака
    return math.copysign(abs(value) ** (1.0 / 3.0),
value)

def main() -> None:
    # зчитування вхідних даних
    x = float(input("Введіть x: "))

    t = 1e-12  # допуск для перевірок на нуль

    # перевірка ОДЗ: вираз під логарифмом має бути > 0
    expr_log = abs(0.8 - x)
    if not (expr_log > 0):
        print("вираз під логарифмом не додатній")
        return

    denom = math.log(expr_log) - x * x
    if math.isclose(denom, 0.0, abs_tol=t):
        print("ділення на нуль")
        return

    numerator = (2 ** (-math.cos(2 * x))) - x *
cbrt(0.75 - x * x)
    y = numerator / denom

    print(f"x = {x}  y = {y}")

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

ДОДАТОК В.
СПРОЩЕНЕ ІНДИВІДУАЛЬНЕ ЗАВДАННЯ №1 З ПАРАМЕТРАМИ

Функція (загальний вигляд):

$$Y = \frac{a \cdot \ln|x - b| + c \cdot x^p}{\sqrt{d \cdot x - e} + f \cdot x - g}$$

де параметри a, b, c, p, d, e, f, g задаються згідно з таблицею варіантів.

Визначення ОДЗ (умови коректності обчислень):
$|x - b| > 0$ — вираз під логарифмом додатній (x ≠ b);
$d \cdot x - e \geq 0$ — вираз під квадратним коренем не від'ємний;
$\sqrt{d \cdot x - e} + f \cdot x - g \neq 0$ — знаменник не дорівнює нулю.

Зауваження до чисельної перевірки: у програмі використовують малий допуск t (наприклад, 10^{-12}) замість точних рівностей/нерівностей, щоб уникнути впливу округлень:

$$|x - b| > t$$
$$d \cdot x - e \geq -t$$
$$\sqrt{\{d\,x - e\}} + f\,x - g| > t$$
$$\left|\sqrt{d \cdot x - e} + f \cdot x - g\right| > t$$

Примітки щодо реалізації в Python:

– використовувати `abs(x - b)` для модуля, `math.log(abs(x - b))` для натурального логарифма;

– ступінь p задається як дійсний показник: `x ** p`;

– перевірки ОДЗ виконувати ДО обчислення виразу; для порівнянь з нулем — з урахуванням допуску t.

Постановка завдання

1. Обрати номер варіанту згідно з журналом групи.
2. За Таблицею С.1 визначити коефіцієнти a, b, c, p, d, e, f, g для обраного варіанту.
3. Для введеного дійсного x перевірити виконання ОДЗ, для перевірок використовувати допуск t.
4. Обчислити значення

$$Y = \frac{a \cdot \ln|x - b| + c \cdot x^p}{\sqrt{d \cdot x - e} + f \cdot x - g}$$

та вивести обраний варіант, коефіцієнти, x і результат Y. У випадку порушення будь-якої з умов ОДЗ — скорегувати значення x (див. п.2).

5. Результати оформити у вигляді звіту: мета, постановка, лістинг, результати, висновки.

Таблиця В.1 — Коефіцієнти для 23 варіантів

№	a	b	c	p	d	e	f	g
1	1	0,8	0,5	2	2	1	0,5	1
2	1,2	0,7	0,4	1,5	1,8	0,9	0,6	1,1
3	0,8	0,6	0,6	2,5	2,2	1,2	0,4	0,9
4	1,5	1	0,3	1,2	2,5	1	0,7	1,3
5	0,9	1,2	0,7	2,2	1,6	0,8	0,5	0,8
6	1,1	0,5	0,5	1,8	2,1	0,7	0,6	1,2
7	1,3	0,9	0,4	2	1,9	1,1	0,4	1
8	0,7	0,4	0,6	2,8	2,3	0,6	0,7	1,4
9	1,4	1,1	0,5	1,3	1,7	1,3	0,5	0,7
10	1	0,3	0,5	2,1	2,4	0,5	0,6	1,1
11	1,2	0,2	0,4	2,3	2	0,4	0,5	1,2
12	0,9	1,3	0,7	1,7	1,5	1,4	0,4	0,9
13	1,1	0,1	0,6	2,6	2,2	0,3	0,7	1,3
14	0,8	0	0,3	1,4	2,6	0,2	0,6	1
15	1,3	1,4	0,5	2,4	1,8	1,5	0,5	0,8
16	1	-0,2	0,4	1,6	2,1	0,1	0,6	1,4
17	1,5	-0,5	0,6	2,2	1,9	0	0,4	1,1
18	0,7	0,2	0,7	1,9	2,3	-0,2	0,7	0,9
19	1,4	-0,8	0,3	2,7	2	-0,3	0,5	1,2
20	0,9	0,5	0,6	1,5	2,4	-0,4	0,6	1
21	1,1	-0,3	0,5	2,9	2,2	-0,5	0,4	1,3
22	1,2	0,6	0,4	1,1	1,7	-0,6	0,7	0,7

Код програми для обчислення за номером варіанту:

```
import math

# табличні коефіцієнти для 23 варіантів (a, b, c, p, d, e, f, g)
coeffs = [
    (1.0, 0.8, 0.5, 2.0, 2.0, 1.0, 0.5, 1.0),
    (1.2, 0.7, 0.4, 1.5, 1.8, 0.9, 0.6, 1.1),
    (0.8, 0.6, 0.6, 2.5, 2.2, 1.2, 0.4, 0.9),
    (1.5, 1.0, 0.3, 1.2, 2.5, 1.0, 0.7, 1.3),
    (0.9, 1.2, 0.7, 2.2, 1.6, 0.8, 0.5, 0.8),
    (1.1, 0.5, 0.5, 1.8, 2.1, 0.7, 0.6, 1.2),
    (1.3, 0.9, 0.4, 2.0, 1.9, 1.1, 0.4, 1.0),
```

```
    (0.7, 0.4, 0.6, 2.8, 2.3, 0.6, 0.7, 1.4),
    (1.4, 1.1, 0.5, 1.3, 1.7, 1.3, 0.5, 0.7),
    (1.0, 0.3, 0.5, 2.1, 2.4, 0.5, 0.6, 1.1),
    (1.2, 0.2, 0.4, 2.3, 2.0, 0.4, 0.5, 1.2),
    (0.9, 1.3, 0.7, 1.7, 1.5, 1.4, 0.4, 0.9),
    (1.1, 0.1, 0.6, 2.6, 2.2, 0.3, 0.7, 1.3),
    (0.8, 0.0, 0.3, 1.4, 2.6, 0.2, 0.6, 1.0),
    (1.3, 1.4, 0.5, 2.4, 1.8, 1.5, 0.5, 0.8),
    (1.0, -0.2, 0.4, 1.6, 2.1, 0.1, 0.6, 1.4),
    (1.5, -0.5, 0.6, 2.2, 1.9, 0.0, 0.4, 1.1),
    (0.7, 0.2, 0.7, 1.9, 2.3, -0.2, 0.7, 0.9),
    (1.4, -0.8, 0.3, 2.7, 2.0, -0.3, 0.5, 1.2),
    (0.9, 0.5, 0.6, 1.5, 2.4, -0.4, 0.6, 1.0),
    (1.1, -0.3, 0.5, 2.9, 2.2, -0.5, 0.4, 1.3),
    (1.2, 0.6, 0.4, 1.1, 1.7, -0.6, 0.7, 0.7),
    (1.0, -0.1, 0.5, 2.0, 2.5, -0.7, 0.5, 1.1),
]

# вхідні дані
variant = int(input("Введіть номер варіанту (1-23): "))
x = float(input("Введіть x: "))

# перевірка коректності номера варіанту
if variant < 1 or variant > 23:
    print("невірний номер варіанту")
else:
    a, b, c, p, d, e, f, g = coeffs[variant - 1]
    t = 1e-12  # допуск для перевірок на нуль

    # перевірки ОДЗ
    if abs(x - b) <= t:
        print("порушено ОДЗ: x не повинен дорівнювати b
(|x-b|>0)")
    else:
        radicand = d * x - e
        if radicand < -t:
            print("порушено ОДЗ: вираз під коренем
від’ємний")
        else:
            denom = math.sqrt(radicand if radicand > 0
else 0.0) + f * x - g
            if abs(denom) <= t:
                print("порушено ОДЗ: знаменник дорівнює
```

```
нулю")
            else:
                numerator = a * math.log(abs(x - b)) + c *
(x ** p)
                y = numerator / denom
                print(f"Варіант {variant}: a={a} b={b}
c={c} p={p} d={d} e={e} f={f} g={g}")
                print(f"x = {x}")
                print(f"y = {y}")
```

ДОДАТОК Г.
ВАРІАНТИ ЗАВДАНЬ ДЛЯ ІНДИВІДУАЛЬНОГО ЗАВДАННЯ №2

Таблиця Г.1 – Варіанти завдань і критерії сортування

№ варіанту	Критерій сортування	Напрям сортування
1	Лексикографічно (алфавітно), чутливо до регістру	зростання
2	Лексикографічно (алфавітно), чутливо до регістру	спадання
3	За кількістю слів у рядку	зростання
4	За кількістю слів у рядку	спадання
5	За довжиною першого слова	спадання
6	За довжиною першого слова	зростання
7	За кількістю цифр у рядку	зростання
8	За кількістю цифр у рядку	спадання
9	За кількістю малих літер у рядку	спадання
10	За кількістю малих літер у рядку	зростання
11	За кількістю великих літер у рядку	спадання
12	За кількістю великих літер у рядку	зростання
13	За кількістю знаків пунктуації (ASCII)	спадання
14	За кількістю знаків пунктуації (ASCII)	зростання
15	За кількістю літер у рядку	зростання
16	За кількістю літер у рядку	спадання
17	За довжиною другого слова	спадання
18	За довжиною другого слова	зростання
19	За кількістю недрукованих символів	зростання

Навчально-методичне видання

Мірошниченко Вікторія Ігорівна
Мірошниченко Сергій Олександрович
Полупан Віталій Геннадійович

КОМП'ЮТЕРНА ТЕХНІКА, АЛГОРИТМІЗАЦІЯ ТА ПРОГРАМУВАННЯ

методичні вказівки до виконання індивідуальних завдань

Самостійне електронне мережеве видання

Публікується в авторський редакції