# Інформатика

## Профільний рівень

підручник для 10 класу
закладів загальної середньої освіти

Харків
Видавництво «Ранок»
2018

УДК [004:37.016](075.3)
Р83

**Руденко В. Д.**
Р83 Інформатика (профільний рівень) : підруч. для 10 кл. закл. загал. серед. освіти /В. Д. Руденко, Н. В. Речич, В. О. Потієнко. — Харків : / Вид-во «Ранок», 2018.
ISBN

**УДК [004:37.016](075.3)**

**Інтернет-підтримка**
Електронні матеріали
до підручника розміщено на сайті
interactive.ranok.com.ua

ISBN

# Зміст

**Шановні десятикласники та десятикласниці!**

За п'ять років ви опанували базові знання та сформували компетенції, маєте первинні навички практичної роботи з багатьма програмними засобами. Ви досягли певного рівня інформаційної культури і здатні самостійно оволодівати сучасними інформаційними технологіями. Та інформатика — дуже динамічна наука. Її подальші напрямки й темпи розвитку значною мірою визначатимуться рівнем підготовки людей, які мають ґрунтовні знання в цій галузі. Імовірно, багатьох із вас, адже ви вивчатимете інформатику на профільному рівні і, сподіваємося, в майбутньому пов'яжете з нею свою професійну діяльність.

У 10 класі ви будете працювати з новими програмними засобами. Ви зануритеся у світ статистики, фінансових розрахунків, інфографіки; навчитеся автоматизувати процес оформлення текстового документа, створювати та опрацьовувати графічні зображення у векторному та растровому графічних редакторах.

Під час опанування курсу багато навчального часу приділяється мовам програмування та структуруванню даних. Пропонований підручник — серед перших, де вивчаються основи об'єктно-орієнтованого програмування. Оволодіння основами алгоритмізації і програмування здійснюється на основі мови Python і середовища програмування IDLE.

Бажаємо вам успіхів,
*авторський колектив*

Підручник, який ви тримаєте в руках, — ваш надійний помічник. У ньому ви знайдете завдання для самостійного виконання — виконуйте їх на комп'ютері з натхненням, повторюйте теоретичний матеріал і викладайте основні положення на папері.

Описи практичних робіт, запропонованих до курсу інформатики, ви знайдете на сайті «Інтерактивне навчання» (interactive.ranok.com.ua).

Скориставшись цим посиланням, ви також зможете пройти комп'ютерне тестування з автоматичною перевіркою результату.

Різнорівневі питання для перевірки знань і завдання для самостійного виконання відповідають рівням навчальних досягнень:

— початковий і середній рівні

— достатній рівень

— високий рівень

У тексті використано також позначки:

 — означення, висновок

 — питання на повторення

 — зверніть увагу

 — цікаво знати

 — завдання для виконання та обговорення в парах або групах

 — вправи для домашнього виконання

# Розділ 1. МОВА ПРОГРАМУВАННЯ ТА СТРУКТУРИ ДАНИХ

## 1. Структура і способи виконання проектів мовою Python

### 1.1. Класифікація і складові мов програмування

*Із якими мовами програмування ви ознайомилися в попередніх класах? Які задачі розв'язували за їх допомогою?*

Історію комп'ютерних наук до певної міри можна подати як історію мов програмування, початок розвитку яких припадає на XIX ст., коли англійський учений Чарльз Беббідж розробив механічну обчислювальну машину. Програму для неї, як вам відомо, написала леді Ада Лавлейс. Мови програмування в сучасному розумінні фактично почали розвиватися з появою електронних обчислювальних машин.

Ада Августа Лавлейс, дочка лорда Байрона, розробила перші програми для аналітичної машини Беббіджа, заклавши тим самим теоретичні основи програмування. На її честь названо мову програмування ADA.

**Мова програмування** (англ. *Programming language*) — це штучна мова, створена для розробки програм, які призначено для виконання на комп'ютері.

**Комп'ютерна програма** (англ. *Computer Program*) — це послідовність команд (інструкцій), що забезпечує реалізацію на комп'ютері конкретного алгоритму.

**Команда (інструкція)** — це вказівка, що визначає, яку дію (операцію) слід виконувати.

Сьогодні можна нарахувати понад 2 тис. різних мов програмування та їх модифікацій, проте лише окремі набули широкого визнання. Усі мови програмування можна умовно класифікувати за деякими *основними ознаками* (рис. 1).

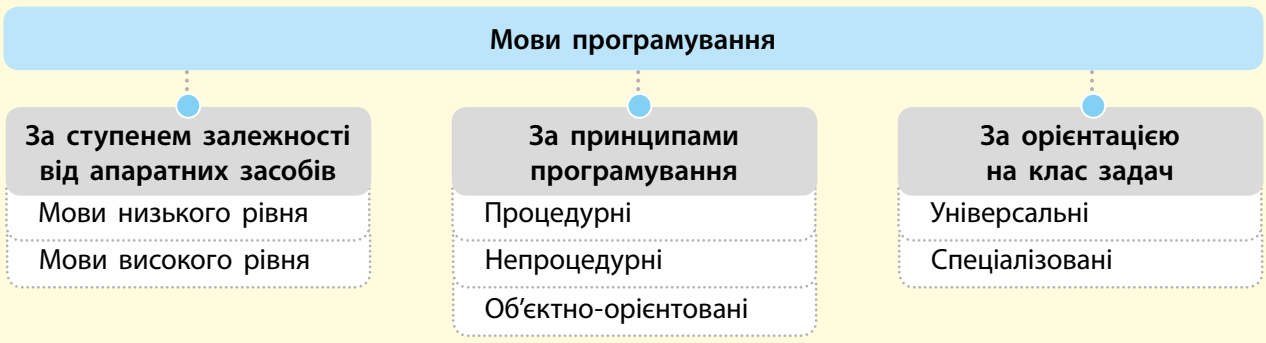


Рис. 1. Класифікація мов програмування

- За ступенем залежності від апаратних засобів розрізняють *мови програмування низького* і *високого рівнів*.

Мови програмування низького рівня (машинно-орієнтовані) — мови, у яких команди та дані враховують архітектуру комп'ютера. Такі мови орієнтовані на конкретний тип комп'ютера і враховують його особливості.

Практично кожний тип комп'ютера мав власну мову програмування низького рівня. Одна й та сама програма не могла виконуватися на комп'ютері іншого типу, що суттєво обмежувало можливість обміну програмами.

Програми для перших ЕОМ розробляли саме «машинними» мовами. Це був складний процес, тому невдовзі з'явилися мови символьного кодування. Команди подавалися вже не двійковим кодом, а символами. Перетворення символьного коду в машинні команди здійснюється автоматично.

Зазвичай команди сучасних мов програмування записують англійськими літерами з використанням символів, які містяться на клавіатурі. Але в комп'ютері зберігаються й виконуються команди, які подано фізичними сигналами (наприклад, двома рівнями остаточної магнітної індукції, двома значеннями електричної напруги, наявністю та відсутністю світлового променя тощо). Значення фізичних сигналів ототожнюються з математичними значеннями 0 і 1, тобто двійковими символами.

Програми, що подано сукупністю 0 і 1, називають машинними, або машинним кодом. Він указує, яку саме дію слід виконати процесору.

Використовуються різні структури команд. Найчастіше команди складаються з операційної та адресної частин. В операційній частині зазначається яку дію (операцію) слід виконати, а в адресній — виконати над якими даними (приклад).

У нашому випадку код A3 може бути операційною частиною й означати, наприклад, операцію Додати, а B7 і C5 — адресною частиною, яка визначає місце збереження даних, над якими слід виконати операцію.

Уже на перших етапах розвитку обчислювальної техніки почалося розроблення мов, доступних для широкого кола користувачів і не пов'язаних із конкретним комп'ютером. Першою мовою високого рівня, яка набула визнання програмістів, була Fortran.

Процес розроблення програм дещо полегшився, коли до мов символьного кодування почали включати макрокоманди, що реалізуються послідовністю з кількох машинних команд. До різновидів мов символьного кодування належать мови асемблера й автокода.

Мови програмування високого рівня (машинно-незалежні) — мови, на яких програми можуть використовуватися на комп'ютерах різних типів і які більш доступні людині, ніж мови низького рівня.

На різних етапах розвитку комп'ютерної техніки популярністю користувалися різні мови програмування.


**Приклад.**
Спрощено команду двійковим або шістнадцятковим кодом можна записати так:

10100011 10110111 11000101
або
A3 B7 C5

Вагомий внесок у розвиток теорії мов програмування зробила Катерина Логвинівна Ющенко. Вона написала перші програми для першої ЕОМ, створеної у НАН України під керівництвом С. О. Лебедєва.

- За принципами програмування розрізняють *процедурні*, *непроцедурні* мови та мови *об'єктно-орієнтованого програмування*.

Процедурні мови ґрунтуються на описі послідовної зміни стану комп'ютера, тобто значення комірок пам'яті, стану процесора й інших пристроїв. Вони маніпулюють даними в покроковому режимі, використовуючи послідовні інструкції. У процедурних мовах витримано чітку структуризацію програм, тому їх ще називають *мовами структурного програмування*. До таких мов належать Fortran, Algol, Pascal, BASIC тощо.

Процедурні мови повністю задовольняють потреби розроблення невеликих програм і програм середньої складності. Але на початку 80-х років XX ст. обсяг і складність програм досягли рівня, який вимагав нових концептуальних підходів до програмування.

Непроцедурні мови є ефективними для програмування пошуку даних у великих обсягах, а також для програмування задач, процес розв'язування яких неможливо описати точно (переклад, розпізнавання образів). У цих мовах саму процедуру пошуку розв'язку вбудовано в інтерпретатор мови. До таких мов належать мови *функціонального* і *логічного програмування*.

Наприкінці XX ст. було презентовано нову методику програмування, що отримала назву об'єктно-орієнтованого програмування (ООП). Тобто почали розвиватися мови, що містять конструкції, які дають змогу визначати об'єкти, що належать класам і мають властивості роботи з абстрактними типами даних. До таких мов належать C++, Java, C#, Python та ін. Нині мови ООП практично витіснили з ринку професійного програмування процедурні мови.

- За орієнтацією на клас задач мови програмування поділяють на *універсальні* та *спеціалізовані*.

Універсальні мови призначені для розв'язування широкого класу задач. До таких мов належать PL/1, Algol, Pascal, C тощо. Особливим класом універсальних мов є візуальні середовища програмування: VisualBasic, Delphi й ін.

Спеціалізовані мови враховують специфіку предметної галузі. На цей час існують десятки спеціалізованих мов програмування, наприклад, мови веб-програмування, мови скриптів тощо. *Мова скриптів* використовується для створення невеликих допоміжних програм, мова Javascript — для створення динамічних об'єктів на веб-сторінках. *Мови розмітки* містять шаблони та засоби опису вмісту, структури й формату електронних документів, наприклад мова HTML забезпечує розмітку гіпертекстового документа. *Мови для роботи з базами даних* забезпечують створення й супровід баз даних.

Зазначимо, що не всі з перелічених мов у класичному розумінні є мовами програмування. Так, мова HTML є мовою розмітки гіпертексту, але її також часто помилково називають мовою програмування.

Першою мовою високого рівня, яка набула широкого визнання серед програмістів світу, була **Fortran**. Її було розроблено корпорацією IBM (США) у 1954 році. Мова **Fortran** наближена до мови алгебри та орієнтована на розв'язування обчислювальних задач.
У 1960 році групою вчених різних країн створено мову **Algol-60**, теж орієнтовану на розв'язування обчислювальних задач.


День програміста святкують у 256-й день року (у високосний рік це 12 вересня, а в невисокосний — 13 вересня). Як ви думаєте, чому обрано саме цей день?

Вибір пояснюється тим, що це число символічне, воно тісно пов'язане з комп'ютерами, але не асоціюється з конкретними особами чи кодами спеціальностей. Число 256 відповідає кількості символів, які можна подати за допомогою одного байта.

Починаючи з 60-х років XX ст. розвиток мов програмування відбувається як шляхом спеціалізації, так і шляхом універсалізації.

Однією з перших спеціалізованих мов була мова COBOL, розроблена в США 1961 року й орієнтована на розв'язування економічних задач. Згодом з'явилися десятки різних спеціалізованих мов, наприклад, Simula — мова моделювання, LISP — мова для інформаційно-логічних задач, RPG — мова для розв'язування навчальних задач тощо.

Будь-яка мова програмування високого рівня, як і будь-яка інша мова, має основні складові (рис. 2): *алфавіт*, *синтаксис*, *семантику*.

> Найкращий спосіб у чомусь розібратися до кінця — це спробувати навчити цього комп'ютер.
>
> *Дональд Ервін Кнут*

**Складові мови програмування**

**Алфавіт**

Набір символів, із яких утворюються команди програми й інші конструкції мови.

Кожна мова має власний алфавіт. Але більшість із них містить англійські літери, цифри, знаки арифметичних операцій (+, *, –, /), знаки відношень (більше, дорівнює й ін.), синтаксичні знаки (крапка, крапка з комою тощо)

**Синтаксис**

Сукупність правил запису команд та інших конструкцій мови.

Порушення правил синтаксису виявляється автоматично, про що програміст отримує повідомлення

**Семантика**

Сукупність правил тлумачення та виконання конструкцій мови програмування.

Наприклад, два коди, наведені далі, мають однакову логіку (виконують однакові дії), результати їх виконання теж однакові. Але семантично коди різні:

```
i = 0; while (i<5) {i++;}
```

та

```
i = 0; do {i++;} while (i<=4)
```

Рис. 2. Основні складові мови програмування

Мова програмування має словник — певну кількість слів, правила вживання яких визначено певною мовою і які мають строго визначене призначення. Такі слова називають *зарезервованими (ключовими)*, наприклад, for, input, if, print.

## Запитання для перевірки знань

1. Що таке мова програмування?
2. Які мови називають машинно-орієнтованими?
3. Які мови називають мовами програмування високого рівня?
4. За якими ознаками класифікують мови програмування?
5. Як класифікуються мови програмування за орієнтацією на клас задач?
6. Як класифікуються мови програмування за принципами програмування?
7. Чому виникла потреба в об'єктно-орієнтованому програмуванні?
8. Назвіть основні складові мов програмування.
9. Поясніть на прикладах сутність синтаксису мови програмування.

# 1.2. Призначення і склад середовища програмування

*Які середовища програмування ви використовували в попередніх класах? Назвіть їх переваги й недоліки.*

Для зручної розробки програм існують спеціальні засоби їх створення, — середовища (системи) програмування, які забезпечують весь цикл роботи з програмою — від її розроблення до виконання й отримання необхідних результатів.

Вважається, що першим пристроєм із програмним керуванням був ткацький верстат, побудований Жозефом Марі Жаккаром у 1804 році. Верстат здійснив революцію в ткацькій промисловості: Жаккар віднайшов можливість за допомогою перфокарт програмувати візерунки на тканинах.

**Середовище програмування** — це комплекс програмних засобів, які призначено для автоматизації процесу підготовки та виконання програм користувача.

Розглянемо основні складові середовища програмування (рис. 1).

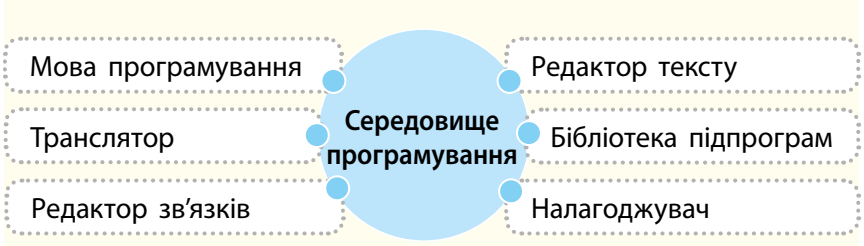


Рис. 1. Складові середовища програмування

Для свідомого розуміння призначення складових середовища програмування опишемо етапи процесу розробки програми, пов'язані з використанням комп'ютера.

**Етап 1**

Уводимо текст розробленої програми, яку називають *початковим кодом*, у комп'ютер і зберігаємо в пам'яті. Для цього середовище програмування має редактор тексту, який забезпечує введення й редагування початкового коду.

**Етап 2**

Після введення програми та виправлення помилок, які могли статися під час уведення, здійснюється перетворення програми з мови програмування високого рівня у двійковий код.

Таке перетворення здійснюється за допомогою транслятора програм.

Розрізняють два типи трансляторів: *компілятори* й *інтерпретатори*.

У процесі *інтерпретації* з початкового програмного коду послідовно кожна команда (інструкція) перетворюється у двійковий код і відразу виконується, — на екрані висвітлюється результат її виконання. Після завершення виконання однієї команди виконується наступна і так до останньої команди. Але результат перетворення не зберігається, і кожного запуску програма починається спочатку.

У процесі *компіляції* здійснюється перетворення всього тексту програмного коду у двійковий код. Отриману після компіляції програму називають *об'єктним модулем*. Така програма ще не готова до виконання.

Початковий код зазвичай містить посилання на інші модулі (підпрограми), які містяться в бібліотеці підпрограм (наприклад, модуль обчислення квадратного кореня). Таким чином, до програмного модуля потрібно додати коди необхідних підпрограм, щоб підготувати програму для виконання.
Компільовані програми виконуються швидше за інтерпретовані. Режим інтерпретації потребує додаткової основної пам'яті, оскільки інтерпретатор повинен увесь час зберігатися разом із кодом. Але інтерпретація в роботі зручніша. Особливо для програмістів, які лише починають працювати з середовищем програмування, оскільки контролюється результат кожної команди.

**Етап 3**

Після компіляції редактор зв'язків «склеює» окремі двійкові модулі в єдину програму, яка називається програмою, що виконується, і яка вже призначена для виконання.
Цей процес (*етапи 1–3*) подано схемою (рис. 2).

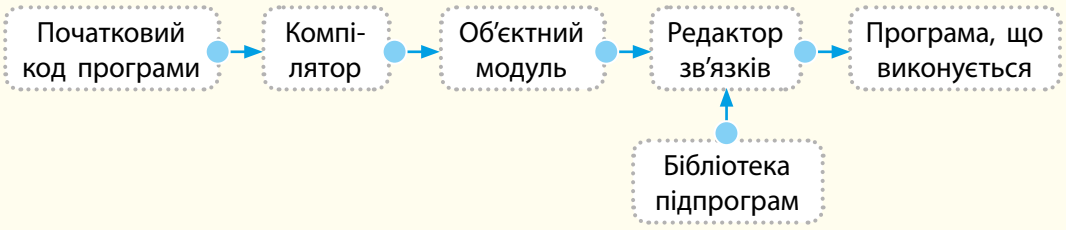


Рис. 2. Процес опрацювання початкового коду програми

Для подальшого виконання програмного коду, що виконується, компілятор не потрібен. Отже, після компіляції програма подана двійковими символами 1 і 0.

**Етап 4**

Отримана програма, що виконується, не гарантує, що немає логічних помилок. Вона може виконуватися, але результат виконання може бути неправильним. Тож потрібно здійснити тестування (випробування) програми на предмет виявлення й усунення в ній логічних помилок.
*Тестування* — досить відповідальний етап. У великих ІТ-компаніях над розробленням програм, які називають проектами, працюють десятки й навіть сотні програмістів різних напрямків. Одні з них розробляють проекти, інші займаються тестуванням програм, економічним обґрунтуванням тощо.
На цьому етапі застосовується налагоджувач програм, який дає змогу покроково аналізувати програму. Налагоджувач дозволяє виконувати трасирування програми, встановлювати й видаляти контрольні точки в програмах, умову призупинення виконання програми тощо.

Описаний процес розробки програм є класичним для процедурних мов програмування. Для програм, розроблених мовою ООП, є відмінності. Їх сутність полягає в тому, що після компіляції отримується не машинний, а проміжний код, так званий байт-код. За допомогою спеціального програмного забезпечення він потім перетворюється на машинний.

**Байт-код** — це проміжний код між початковим кодом і кодом, що виконується.

Такий підхід зумовлений тим, що в Інтернеті вільно переміщуються дані та програми (*аплети* — невеликі програми, призначені для передавання через Інтернет і виконання

Аплет переносить окремі функції із сервера до клієнта. У разі клацання посилання, що містить аплет, він автоматично завантажується на комп'ютер і запускається в браузері.

Потребу в розробці мов ООП із використанням байт-коду на початку 1990-х років зумовлено розвитком виробництва побутових приладів зі вбудованими контролерами на різних типах процесорів.

Спочатку виникла потреба в програмах і компіляторах для кожного типу процесора. Та розробка компілятора виявилася справою доволі дорогою. Так з'явилася ідея створити мову для розробки коду, придатного для виконання на будь-якому типі процесора.

За рік виникла потреба в мові, яку можна було б використовувати на будь-якому типі процесора. Це було зумовлене розвитком Всесвітньої павутини й Інтернету, які об'єднали різні типи комп'ютерів із різними процесорами й ОС.

в браузері, сумісному з мовою програмування). Їх потрібно захистити від вірусів та інших шкідливих програм, а також реалізувати переносимість програм.

Під переносимістю розуміють можливість завантаження й виконання аплета на комп'ютерах із будь-яким типом процесора, будь-якою операційною системою та браузером, що під'єднані до Інтернету. Саме ці проблеми й дозволяє розв'язати байт-код.

Зрозуміло, що використання будь-якого проміжного коду, у тому числі й байт-коду, знижує швидкість виконання програм і потребує додаткових апаратних засобів. Утім, ці втрати незначні порівняно з отриманим виграшем. Якби ООП-програма одразу компілювалася в машинний код, то для кожного комп'ютера зі своїм типом процесора необхідно було б мати окрему версію тієї самої програми, що економічно вкрай невигідно.

Інколи використовуються так звані динамічні компілятори. Їх сутність полягає в тому, що байт-код компілюється в машинний код не весь одразу, а окремими фрагментами, у міру необхідності. Інші частини коду можуть виконуватися в режимі інтерпретації. Цим самим досягається висока ефективність роботи з кодом.

Середовища (системи) програмування часто іменуються за назвою мови, яка в них реалізується, наприклад середовище Pascal, середовище Delphi. Інколи назва середовища містить префікс, який вказує на розробника середовища: назва середовища Turbo-C означає, що її розробником є фірма Borland.

Нині все частіше використовуються інтегровані середовища програмування, які забезпечують роботу з кількома мовами. Такими середовищами є, наприклад, IntelliJ IDEA, Eclipse. Варіант Ultimate Edition середовища IDEA забезпечує роботу з мовами програмування Java, PHP, Python.

Деякі середовища підтримують як режим інтерпретації, так і режим компіляції програм.

Далі в процесі опису мови програмування Python ми будемо застосовувати середовище IDLE.

## Запитання для перевірки знань

1. Для чого призначено середовища програмування?
2. Назвіть складові середовища програмування.
3. Які функції виконує редактор тексту?
4. Поясніть сутність інтерпретації програм.
5. Які переваги та дефекти мають компілятори й інтерпретатори програм?
6. Для чого призначено редактор зв'язку?
7. Які основні дії виконуються в процесі налагодження програм?
8. Назвіть сучасні інтегровані середовища програмування.
9. Що називають об'єктним кодом?
10. Які особливості мають середовища об'єктно-орієнтованого програмування?
11. Для чого застосовують байт-код?
12. Які переваги мають інтегровані середовища програмування?

# 1.3. Основні можливості мови Python і структура проекту

*Яку методику програмування підтримує мова, яку ви вивчали? Чи користувалися ви об'єктно-орієнтованими мовами програмування?*

Матеріал нашого підручника зорієнтовано на роботу з мовою програмування Python*, яка підтримує об'єктно-орієнтований і процедурний методи програмування з інтерпретацією команд (інструкцій).

Мова Python підтримується всіма операційними системами і дозволяє розв'язувати складні математичні задачі, створювати графічні зображення, розробляти веб-сайти, працювати з реляційними базами даних.

Мова Python має потужну стандартну бібліотеку, яку користувач може розширювати власними бібліотеками й бібліотеками інших користувачів. Наприклад, розширення .NumPy містить реалізацію різноманітних математичних обчислень, модуль tkinter дає змогу реалізувати графічний інтерфейс користувача.

Програми можуть розроблятися в консольному режимі (такі програми мають розширення .py) і з графічним інтерфейсом (програми мають розширення .pyw).

Програма мовою Python — це звичайний текстовий файл, інструкції (команди) якого виконуються інтерпретатором для кожного рядка. Під час першого запуску програми створюється байт-код, який зберігається у файлі з розширенням .pyw. Якщо після цього програма не змінювалася, то в процесі наступних її запусків буде виконуватися байт-код.

Усі дані мови, у тому числі прості типи даних (числа, рядки) є об'єктами. У змінній зберігається не сам об'єкт, а посилання на нього, тобто адреса пам'яті, у якій зберігається об'єкт.

Структура проекту мовою Python складається з окремих модулів. Модуль — це будь-який файл із програмним кодом. Кількість таких модулів не обмежена. Один модуль може бути вкладений в інший модуль, тобто застосовується багатоієрархічна структура модулів. Модулі можуть групуватися в пакети.

Модулі можуть розроблятися самим програмістом, а можуть використовуватися вже наявні в стандартній бібліотеці мови. Один із модулів є *головним*, з нього запускається проект на виконання.

Щоб запустити один модуль з іншого, перший необхідно під'єднати до останнього (імпортувати).

* Мова програмування Python розповсюджується вільно за посиланням: http://python.org Наразі існують дві версії Python — 2.x (застаріла) та 3.x (перспективна), які не сумісні між собою. У цьому підручнику наведені приклади, розглянуті у версії 3.4.

Нідерландський програміст Гвідо ван Россум — щирий прихильник скетч-серіалу «Літаючий цирк Монті Пайтона» (англ. Monty Python's Flying Circus). На честь цієї програми він назвав створену ним мову програмування — **Python**.
Існують версії для Linux, Windows, MacOS.

Програму мовою **Python** можна створювати й редагувати за допомогою будь-якого редактора, наприклад, Notepad++, Eclipse, PythonWin та ін.

Мова **Python** підтримує динамічну типізацію даних. Це означає, що оголошувати типи даних не потрібно, мова самостійно слідкує та визначає їх тип на основі їх зовнішнього вигляду. Автоматично звільняється пам'ять для тих даних, які стають непотрібними.

На рис. 1 подано варіант структури проекту мовою Python.

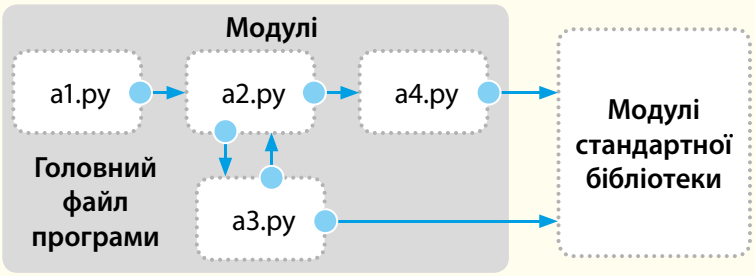


Рис. 1. Варіант структури проекту мовою Python

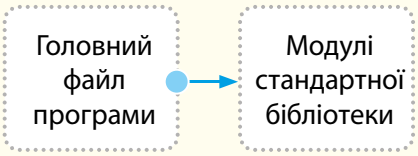


Рис. 2. Найпростіша структура програми мовою Python

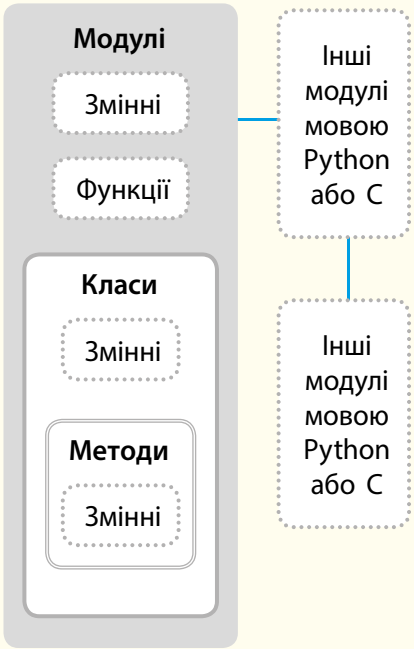


Рис. 3. Структура проекту модуля мовою Python

Оскільки ми розглядатимемо лише найпростіші навчальні проекти, то далі їх можемо називати просто програмами.

Наведена структура програми складається з чотирьох файлів: a1.py, a2.py, a3.py, a4.py, які розробляє програміст, а також модулів стандартної бібліотеки мови Python. Файл a1.py є головним, із нього запускається програма. Цей файл, так само як і інші три модулі, складається з інструкцій мови Python. Фактично це звичайні текстові файли, які нами будуть розроблятися в середовищі IDLE.

Файли a2.py, a3.py і a4.py самостійно не запускаються. Файл a2.py запускається з головного файла. Але для запуску його слід спочатку імпортувати у файл a1.py. Файли a3.py і a4.py запускаються з файла a2.py, але для цього їх також слід імпортувати у файл a2.py. Із файлів a3.py і a4.py запускаються модулі стандартної бібліотеки, які також потрібно імпортувати у ці файли.

Із рис. 1 видно, що мова Python реалізує багатоієрархічну структуру вкладеності модулів. Але в підручнику буде застосовуватися найпростіша архітектура програми, яка містить лише головний файл, та програму, яка містить головний файл і модулі стандартної бібліотеки (рис. 2).

У такій архітектурі модулі стандартної бібліотеки імпортуються безпосередньо в головний файл програми. Реально програмний модуль може складатися не тільки з інструкцій мови Python, а й зі змінних, функцій і класів (рис. 3).

В ООП мовою Python використовуються класи і методи (основи описано нижче). Із рис. 3 видно, що модуль може імпортувати інші модулі, які написано не тільки мовою Python, але й мовою C. Він також може бути імпортований в інші модулі.

## Запитання для перевірки знань

1. Якими операційними системами підтримується мова Python?
2. Яке розширення мають файли програм, які створено в консольному режимі?
3. Назвіть основні переваги мови **Python**.
4. Який тип трансляції застосовується у **Python**?
5. Із якими мовами може інтегруватися мова Python?
6. Що називають динамічною типізацією даних?
7. Поясніть структуру проекту мовою Python.
8. Яку структуру може мати модуль мовою Python?

# 1.4. Режими виконання програмного коду в середовищі IDLE

У середовищі IDLE програмний код мовою Python можна виконувати в *інтерактивному режимі* та *режимі виконання файлів програм*.

## 1.4.1. Виконання програмного коду в інтерактивному режимі

*Пригадайте, який вид трансляції програми використовувався вами у 8 і 9 класах. Які, на вашу думку, його переваги й недоліки?*

В *інтерактивному режимі* результати виконання інструкцій користувача виводяться одразу після їх уведення. Тобто вводиться перша інструкція, яка одразу виконується, потім уводиться друга інструкція і т. д.

Розглянемо способи запуску інтерактивного режиму роботи інтерпретатора IDLE.

- За допомогою командного рядка ОС Windows (щоб його відкрити, слід клацнути правою кнопкою миші кнопку Пуск і виконати команду Командний рядок). До командного рядка слід увести команду python.

  Відкриється вікно, яке зображене на рис. 1.

Інтерактивний режим доцільно використовувати на етапі вивчення синтаксису мови, коли необхідно переконатися в правильності виконання окремих інструкцій після їх уведення. А також для тестування коду, що зберігається у файлах.


Рис. 1. Вікно інтерпретатора, яке відкрите за допомогою командного рядка, версія Python3.4

- Із робочого стола. Якщо ярлик мови Python розміщено на робочому столі, то слід клацнути цей ярлик.
- Із головного вікна середовища програмування IDLE. Для цього слід виконати команди:

  Пуск → Усі програми → Python3.x* → IDLE (Python3.xGUI — 32 біт). Відкриється вікно, яке зображене на рис. 2.

  Саме цей спосіб запуску застосовуватиметься нами далі.

Розглянемо порядок виконання найпростішої програми мовою Python в інтерактивному режимі (рис. 2).

Мова Python розповсюджується вільно на підставі ліцензії GNU General Public License.

* Замість «x» укажіть номер версії Python, що встановлено на вашому комп'ютері — 3.4 або 3.6.

**PEP8** — стиль написання програм мовою Python. Цей документ описує угоду про те, як писати код для мови Python. Це список рекомендацій, яких добровільно дотримуються Python програмісти.

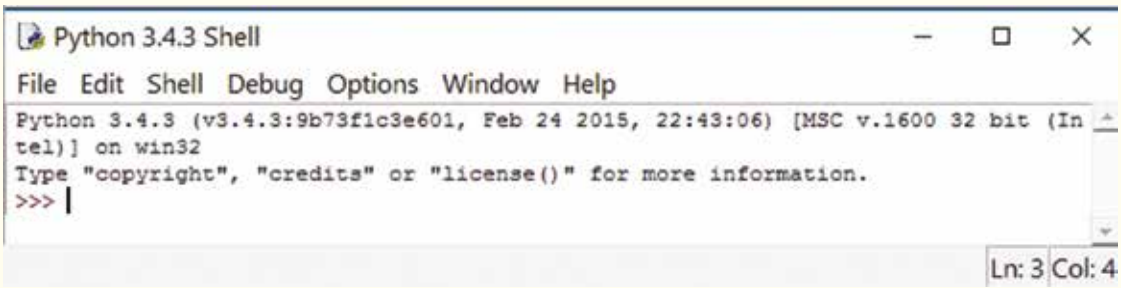


Рис. 2. Вікно інтерпретатора, яке відкрито в середовищі IDLE

**Приклад**

Припустимо, що програма повинна містити повідомлення про стилі програмування та побажання успіхів. Цю програму можна подати в такому вигляді:

```
#Починаємо вивчати мову Python
print ("Python підтримує ООП
і процедурний стиль")                #повідомлення
print ("Найбільших успіхів")         #побажання
```

Тут знак **#** — це коментар, який жодним чином не впливає на роботу інтерпретатора (все, що пишеться після нього, не виконується). Він може бути розташований як на початку рядка програми, так і наприкінці кожної команди.

Команда **print** забезпечує виведення на екран повідомлення, що міститься в круглих дужках у подвійних лапках.

**Хід виконання**

1. Для введення програми запускаємо інтерпретатор мови Python середовища IDLE.
2. Одразу після знака запрошення (>>>) уводимо перший рядок програми (#Починаємо вивчати мову Python) і натискаємо клавішу Enter.
3. Знак запрошення з'явиться на наступному рядку, після якого уводимо команду другого рядка програми (print ("Python підтримує ООП і процедурний стиль") #повідомлення і натискаємо клавішу Enter.
4. Система виведе повідомлення "Python підтримує ООП і процедурний стиль", а в наступному рядку з'явиться знак запрошення, після якого уводимо print ("Найбільших успіхів") #побажання.

Динаміку описаного процесу введення та виконання програми в інтерактивному режимі зображено на рис. 3.

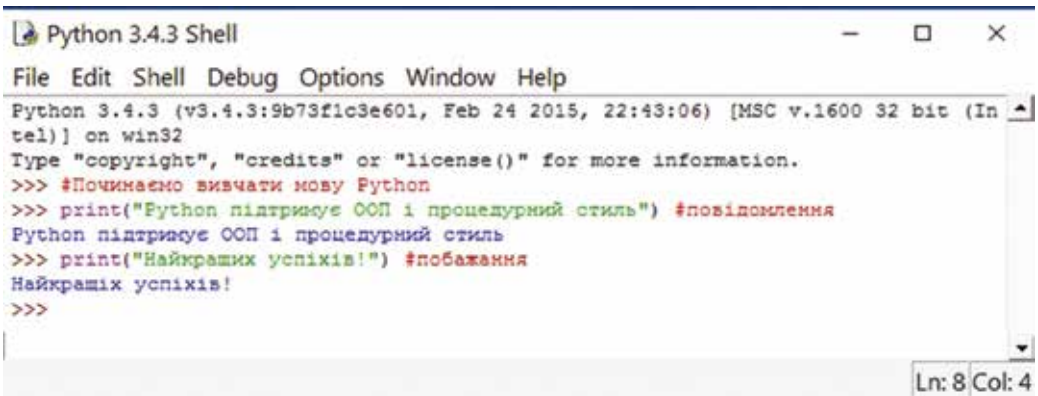


Рис. 3. Перша програма мовою Python (результат роботи)

Після знаку запрошення (>>>) не повинно бути пробілів (але є невеличкий відступ для кращого сприйняття), інакше видаватиметься повідомлення про синтаксичну помилку.

Команди відокремлюються крапкою з комою (;), якщо в одному рядку уводяться дві або більше команд.

Кожну команду рекомендується вводити в окремому рядку, але команди можна розміщувати й у кількох рядках. Наприклад, команду x=130+82+45 можна розмістити у два рядки за такими варіантами:

```
x = 130 + 82 \
+ 45
```

або

```
x = (130 + 82
+ 45)
```

У мові Python відсутні спеціальні засоби для виділення блоків, наприклад фігурні дужки або конструкція begin…end. Якщо команда містить блок, то кожна команда блока мовою Python відокремлюється від початку рядка спеціальним відступом чотирма пробілами. Якщо кількість пробілів різна, інтерпретатор видає повідомлення про фатальну помилку.

Щоб завершити роботу інтерактивного режиму, необхідно натиснути сполучення клавіш Ctrl+Z.

Після того як Гвідо ван Россум розробив мову Python (приблизно у 1991 році), він виклав її в Інтернет. Мова Python сподобалася програмістам і почала вільно поширюватися.

Таким чином, до розробки приєдналося вже співтовариство програмістів.

У середньому кожні два чи 2,5 роки з'являється нова версія мови.

## Запитання для перевірки знань

1. Для чого застосовується коментар у програмному коді?
2. Яке позначення має знак запрошення в інтерактивному режимі?
3. Як завершується робота інтерпретатора в інтерактивному режимі?
4. Які переваги має інтерактивний режим виконання програмного коду?
5. Як можна запустити інтерактивний режим інтерпретатора IDLE?
6. Поясніть сутність інтерактивного режиму.
7. Як правильно увести кілька команд в одному рядку?
8. Як оформлюється блок команд у мові Python?

## Завдання для самостійного виконання

1. Запустіть інтерактивний режим роботи IDLE за допомогою командного рядка ОС Windows.
2. Запустіть інтерактивний режим інтерпретатора IDLE з головного вікна середовища програмування IDLE.
3. Розробіть і виконайте програмний код в інтерактивному режимі, який виводить на екран два речення:

   У мові Python застосовується байт-код.

   Python забезпечує високу компактність і наочність програмного коду.
4. Розробіть і виконайте програмний код, який в інтерактивному режимі виводить таке повідомлення: "Python підтримує процедурні і об'єктно-орієнтовні методи програмування".
5. Розробіть і виконайте програмний код в інтерактивному режимі для виведення повідомлення: "Python має високий рейтинг" із використанням коментаря.
6. Обчисліть значення виразу p = 123 / 4 + 45 * 3 – 66 * (3 + 87) в інтерактивному режимі.

# 1.4.2. Виконання файлів програмного коду

*Інтерактивний режим роботи інтерпретатора забезпечує простоту і наочність у роботі програміста. Чому не можна обмежитися лише цим режимом?*

Способи запуску файла програмного коду на виконання:
- із командного рядка середовища програмування;
- клацанням ярлика файла;
- за допомогою користувацького інтерфейсу середовища IDLE.

Інтерактивний режим роботи є досить зручним. Але програмний код, який уводиться в інтерактивному режимі, не зберігається: він зникає одразу після того, як інтерпретатор мови Python його виконає. Для повторного виконання програмного коду потрібно увести код заново, що є суттєвим недоліком інтерактивного режиму.

Програмні коди необхідно зберігати у файлах, які, як раніше зазначалося, називають модулями. Файли модулів — це програми, які називають також сценаріями, хоча між ними є невелика різниця. Ім'я файла може не містити розширення. Але якщо передбачається імпортувати програмний код з інших файлів, то ім'я файла має містити розширення .py.

Файл програмного коду інтерпретатор може виконувати необмежену кількість разів.

Після запуску коду на виконання інтерпретатор виконує послідовно одну інструкцію за іншою в порядку їх розташування й видає результат на екран монітора. Якщо в програмному коді буде виявлено синтаксичну помилку, відповідне повідомлення виводиться на екран.

Розглянемо порядок створення файла найпростішого програмного коду.

**Приклад**

Припустимо, що потрібно створити файл із кодом, який виконує складання чисел 65 і 279 та множення числа 21 на число 15.

1. Запускаємо середовище програмування IDLE (Python3.x 32-bit). У відкритому вікні Python3.x Shell виконуємо команду File → New File.

2. У вікно Untitled, яке відкрилося, уводимо програму так, як зображено на рис. 1.

Останньою в програмі є команда input(). Наразі ця команда відіграє допоміжну роль. За її допомогою результат виконання програми буде відображатися на екрані доти, доки не буде натиснуто клавішу Enter.

Можливо, що без цієї команди результат відображатиметься досить короткий час, і ми не зможемо його побачити.

```
#Складання і множення чисел
print("65+279=", 65 + 279)
print("21*15=", 21 * 15)
print("Програма завершена")
input()
```

Рис. 1. Програма додавання і множення двох чисел

3. Зберігаємо програму за допомогою команди File → Save As... У результаті відкриється вікно Збереження файлу (рис. 2).

Звертаємо увагу на те, що за замовчуванням пропонується зберегти файл у папці Python3 диску C*. Далі всі файли програм зберігатимемо саме в ній. Інакше необхідно змінити ім'я папки та місце її розташування.

Уводимо ім'я prog_02 (без розширення .py, оскільки на цьому етапі вивчення мови Python не передбачає імпортування файлів), і натискаємо кнопку Зберегти.

Файл буде збережено.

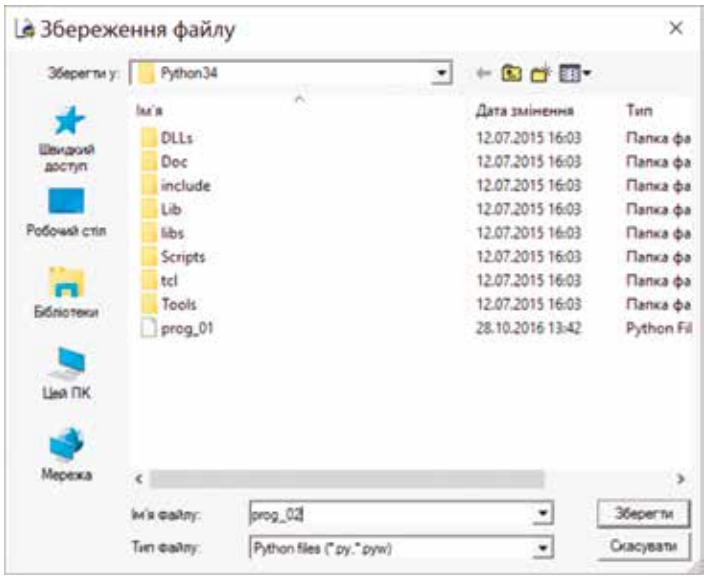


Рис. 2. Вікно **Збереження файлу**

Виконувати програму можна різними способами.

- Якщо програму відкрито в додатковому вікні Python3.4.3 Shell, то достатньо виконати команди Run → Run Module (або натиснути клавішу F5). Результат виконання програми зображено на рис. 3.

Далі будемо користуватися командою Run → Run Module.

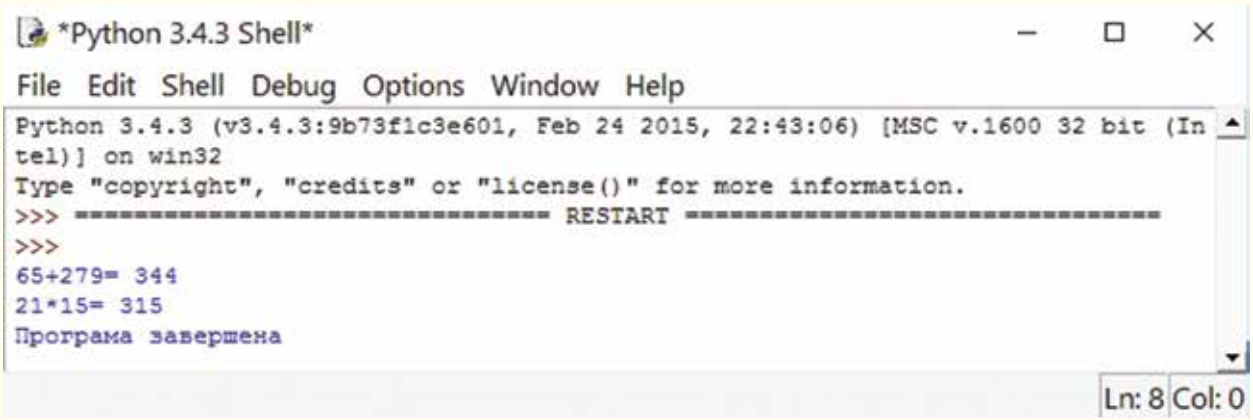


Рис. 3. Результат виконання програми додавання і множення чисел

- Якщо програму створено раніше, її можна** викликати у вікно Python3.4.3 Shell, виконавши команду File → Open...

Виконати програму, створену раніше, можна також подвійним клацанням імені відповідного файла у папці

Нині **Python** — одна з найпопулярніших мов програмування, яка охоплює нові сфери застосування. Останні 5 років входить до п'ятірки найзатребуваніших технологій.

* Цей варіант збереження пропонується за замовчуванням.

** Відкрити в середовищі Python3.x IDLE у вікні Python3.x Shell.

Мову **Python** відрізняє швидкість і простота скриптів. Разом із набором доступних бібліотек для роботи з мережами і файлами це робить її незамінним помічником системного адміністратора.

C:\Python34. Але в цьому випадку останньою командою програми обов'язково повинна бути input(). Інакше результат її виконання можна не побачити.

Зазначимо, що за замовчуванням у Python 3 для кодування команд здійснюється код UTF-8, який буде використовуватися далі.

Для застосування іншого коду необхідно в першому або другому рядку програми вказати назву коду за допомогою інструкції: # –*– coding: <код> –*–.

Наприклад, для коду Windows 1251 інструкція матиме такий зміст: # –*– coding: cp1251 –*–.

## Запитання для перевірки знань

1. Як можна запустити на виконання файл програмного коду?
2. Як можна викликати на екран файл програмного коду?
3. Наведіть приклад найпростішого програмного коду мовою Python.
4. Як зберегти файл програмного коду?
5. Які вади має інтерактивний режим інтерпретатора?
6. Чому доцільно зберігати програмний код у файлі?
7. Для чого в програмному коді останньою зазначається команда input()?

## Завдання для самостійного виконання

1. Складіть програму для обчислення значення виразу 2.3 + 106 * 4 і збережіть її у файлі з іменем f_01. Виконайте програму і переконайтеся, що отриманий результат є правильним.
2. Завантажте середовище IDLE, викличте створену в пункті 1 програму з файла f_01, замініть у програмі операцію множення числа 106 на 4 на операцію ділення. Збережіть програму в тому самому файлі. Виконайте програму та переконайтеся, що отриманий результат є правильним.
3. Складіть програму для обчислення площі трикутника за значеннями його сторін. Збережіть програму у файлі f_02. Виконайте програму та доведіть, що отриманий результат є правильним.
4. Викличте програму, що зберігається у файлі f_02. Внесіть до неї такі зміни, щоб обчислювалася площа прямокутного трикутника за значеннями його катетів.
5. У коло радіусом r вписано правильний трикутник. Розробіть програму визначення площі кола, яку не зайнято трикутником. Збережіть програму у файлі f_03. Виконайте програму та доведіть, що вона функціонує правильно.
6. У правильний трикутник зі стороною a вписано коло. Розробіть програму визначення площі трикутника, яку не зайнято колом. Збережіть програму у файлі f_04. Виконайте програму та доведіть, що вона функціонує правильно.
7. Знайдіть в Інтернеті відстань по шосе між містами Хмельницький і Вінниця. Розробіть проект визначення орієнтовного часу прибуття автобуса до Вінниці, який починає рух із Хмельницького о 8.30.
8. Знайдіть в Інтернеті відомості про найнижчу та найвищу температуру в Києві у лютому за останні 5 років. Розробіть проект визначення різниці між цими показниками.

# 2. Оператори, вирази і засоби опрацювання чисел

Мова Python має багато вбудованих типів даних, які буде описано в наступних розділах. У цьому розділі наводиться лише їх перелік, для того щоб свідомо розуміти необхідність перетворення одного типу даних на інший у процесі програмування арифметичних та інших типів виразів.

## 2.1. Основні елементи мови Python

*Пригадайте елементи мови програмування, яку ви вивчали. Чи всі символи клавіатури комп'ютера ви використовували в процесі введення програм?*

Будь-яка конструкція мови програмування створюється з алфавіту. Алфавіт мови Python містить символи, що розташовані на клавіатурі комп'ютера (рис. 1).

Із символів алфавіту створюються лексеми (token).

**Лексема** — це мінімальна одиниця мови, яка має певне самостійне значення і яку розуміє транслятор. Якщо транслятор її не розуміє, то видається повідомлення про помилку в програмі. За замовчуванням символи кодуються в системі UTF-8.

Розрізняють такі види лексем:

- ключові (зарезервовані) слова (keywords);
- ідентифікатори (identifiers);
- літерали (константи);
- операції;
- знаки пунктуації.

У мові Python використовується кілька десятків ключових (зарезервованих) слів (наприклад, int, list, input, print, float тощо). З їх призначенням будемо знайомитися поступово, у міру потреби в їх застосуванні.

Ідентифікатори (імена) використовуються для позначення змінних, функцій, які створює програміст, та інших об'єктів. Ідентифікатор змінної може складатися з латинських літер, цифр і знака підкреслення. Першим символом імені не може бути цифра або знак підкреслення.

У редакторі IDLE правильні імена змінних висвічуються чорним кольором. Якщо ім'я змінної відображається іншим кольором, його необхідно замінити. Однакові імена з літерами на різних регістрах сприймаються як різні імена. Наприклад, ідентифікатори ster і Ster є різними іменами.

Імена змінних використовуються для доступу до даних. Дані в мові Python подано у формі об'єктів. Тобто *об'єкт* — це ділянка пам'яті з певним значенням і можливими операціями його опрацювання. Кожний об'єкт має свій тип, наприклад int(ціле число), str(рядок) та ін.

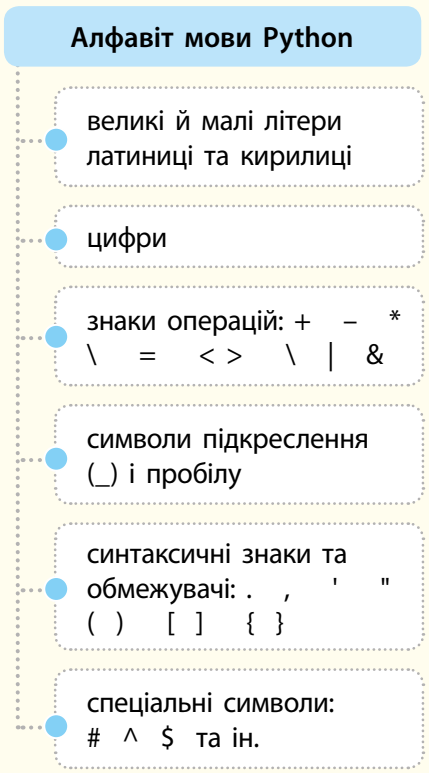


Рис. 1. Алфавіт мови Python

Ключові слова мови, назви функцій та вбудованих у мову об'єктів забороняється використовувати як імена змінних. Наприклад, ідентифікатори змінних dush, cosm_1 і Ftr_2 є правильними, а ідентифікатори 2beta, rnt-sd, _fok, int — неправильними.

Python використовують багато відомих кампаній у всьому світі. Серед них Google, Facebook, Yahoo, NASA, Dropbox, IBM, Red Hat, Pinterest.

У мові Python існують базові об'єктні типи (вбудовані в мову) і ті, що розробляються користувачем засобами самої мови або іншими засобами.

Зазначимо, що змінні зберігають не сам об'єкт, а посилання на об'єкт, тобто адресу об'єкта в пам'яті комп'ютера.

Об'єкт може бути літералом (константою). У табл. 1 наведено найчастіше вживані базові типи об'єктів та приклади їх літералів. Детальніше вони описані в розділі 4.

Таблиця 1. Найчастіше вживані вбудовані типи об'єктів

| Тип об'єкта | Приклад літерала |
|---|---|
| Цілі числа (int) | 45, 365 |
| Дійсні числа (float) | 36.5, 213.75 |
| Рядки (str) | 'Python', "процесор" |
| Логічні дані (bool) | true, false |
| Списки (list) | [40, [3, 'монітор'], 32] |
| Словники (dict) | {'айфон': 'сон', 'doc': 'nm'} |
| Кортежі (tuple) | (5, 'sp', 31, 'M') |
| Множини (set) | set('abc'), {'a', 'b', 'c'} |

Як уже зазначалося, у мові Python застосовується динамічна типізація змінних. Це означає, що не потрібно оголошувати типи змінних, як це робиться в багатьох мовах програмування, оскільки їхній тип визначається автоматично в процесі присвоювання їм значень. Цей тип визначається значенням, розташованим праворуч від оператора присвоювання, який позначається знаком (=) (приклад 1).

В одному рядку можна присвоїти однакові значення кількох змінним, наприклад:

```
>>> x = y = z = 441   #змінні x, y, z мають тип int і значення 441
```

Ще раз зазначимо, що після виконання оператора присвоювання в змінній зберігається не сам об'єкт, а лише посилання на нього, тобто адреса ділянки пам'яті, у якій зберігається об'єкт. Тому слід бути досить уважним під час запису групових операцій. Розглянемо такий фрагмент програми (приклад 2).

Із прикладу 2 видно, що створюється ніби два об'єкти (x і y) типу list, але вони мають одну й ту саму адресу пам'яті, тобто реально створюється один об'єкт, значення якого виводиться двічі. Для підтвердження цього змінимо значення нульового елемента об'єкта y (нумерація елементів у списку починається з нуля) і перевіримо значення об'єктів (приклад 3).

Із прикладу 3 видно, що ми змінили тільки значення y[0], а фактично об'єкти мають однакові значення, оскільки реально і x і y мають однакову адресу, тобто є одним об'єктом.

Щоб отримати два об'єкти x і y, необхідно виконати присвоювання окремо для кожного з них, наприклад:

```
>>> x = [21, 807]
>>> y = [21, 807]
```

**Приклад 1.**

```
>>> x_1 = 87          #змінна x_1 має тип int і значення 87
>>> x_2 = 21.45       #змінна x_2 має тип float і значення 21.45
>>> s_1 = 'проект'    #змінна s_1 має тип str і значення «проект»
>>> s_2 = "байт"      #змінна s_2 має тип str і значення «байт»
```

**Приклад 2.**

```
>>> x = y = [7, 15] #створюється ніби два об'єкти
>>> x, y            #виведення значень x і y
([7,15], [7,15])    #двічі виводяться значення одного об'єкта
```

**Приклад 3.**

```
>>> y[0] = 50       #зміна значення нульового елемента об'єкта y
>>> x, y            #виведення значень x і y
([50, 15], [50, 15])  #значення x і y однакові, хоча x не змінювали
```

Для перевірки, чи посилаються дві змінні на один і той самий об'єкт, використовується оператор is. Якщо змінні посилаються на один об'єкт, то оператор повертає значення True, інакше — значення False (приклад 4).

Одним оператором присвоювання можна присвоїти різні значення кільком змінним.

У такому разі змінні та значення відокремлюються комою, наприклад:

```
>>> x_1, x_2, x_3 = 13, 105, 27
>>> x_1, x_2, x_3
(13, 105, 27)
```

Кількість елементів ліворуч і праворуч від оператора присвоювання має бути однакова, інакше буде видано повідомлення про синтаксичну помилку.

Наприклад, помилку буде видано для такої інструкції:

```
>>> x, y, z = (2, 44, 50, 65)
```

Позбавитися цього явища можна за допомогою символу «зірочка» (*), який розміщується перед однією зі змінних. У такому разі ця змінна містить список з усіх зайвих значень (приклад 5).

**Приклад 4.**

```
>>> x = y = [60, 90]   #змінні x і y посилаються на один об'єкт
>>> x is y
True
>>> x = [23, 20]       #змінні x і y посилаються на різні об'єкти
>>> y = [23, 20]
>>> x is y
False
```

**Приклад 5.**

```
>>> x_1, x_2, *x_3 = (46, 7, 21, 14)
#змінна x_3 набуде значень 21 і 14
>>> x_1, x_2, x_3
(46, 7, [21, 14])
>>> x_1, *x_2, x_3 = (4, 7, 33, 17)
#змінна x_2 набуде значень 7 і 33
>>> x_1, x_2, x_3
(4, [7, 33], 17)
```

## Запитання для перевірки знань

1. Яким ідентифікатором позначаються цілі числа?
2. Яким ідентифікатором позначаються дійсні числа?
3. Які базові типи даних використовуються найчастіше?
4. Сформулюйте правило запису ідентифікаторів змінних.
5. Що зберігається в змінних?
6. Поясніть сутність динамічної типізації змінних.
7. Для чого призначений оператор is?
8. Поясніть сутність поняття «посилання на об'єкт».
9. Як можна одним оператором присвоїти різні значення кільком об'єктам?
10. Що називають лексемою?
11. Які існують види лексем?

## Завдання для самостійного виконання

1. Виявіть помилки в записі ідентифікаторів змінних: ac_1, 2ba, x–1, –z_1.
2. Змінним x, y, z присвойте одним оператором присвоювання, відповідно, такі значення: 3, 37, 45.
3. Доведіть, що після виконання операторів:
   ```
   >>> a1 = [5, 20]
   >>> a2 = [5, 20]
   ```
   змінні будуть посилатися на різні об'єкти.
4. Визначте значення змінних a_1, a_2 і a_3 після виконання оператора:
   ```
   >>> a_1, *a_2, a_3 = (1, 6, 2, 3, 4, 5)
   ```
   і доведіть, що отриманий результат є правильним.
5. Визначте, як реагуватиме середовище програмування на спробу виконати оператор:
   ```
   >>> a, b, c, d = (100, 61, 55).
   ```
6. Одним оператором присвойте змінним такі значення: x = 33, 21, 2; y = 7; z = 66.

# 2.2. Поняття про перетворення типів даних

*Чому, на вашу думку, виникає потреба в перетворенні типів даних? Чи доводилося вам стикатися з цією проблемою під час розв'язування математичних задач?*

Одна й та сама змінна в процесі виконання коду може посилатися на об'єкти з різними типами даних. Наприклад, після виконання двох інструкцій:

```
>>> x_1 = "принтер"        #змінна x_1 посилається на тип str
>>> x_1 = 3.8              #тепер змінна x_1 посилається на тип float
```

Змінна x_1 спочатку посилається на тип str, потім — на тип float.

Для визначення останнього типу даних, на який посилається змінна, призначена функція type (ім'я змінної) (приклад 1).

Отже, тип даних — це характеристика об'єкта, а не змінної. Змінна містить тільки посилання на об'єкт.

Для кожного конкретного типу даних існує строго визначений набір операцій, які можуть виконуватися над ним.

Наприклад, для даних типу int і float можна виконувати арифметичні операції. Спроба виконати, наприклад, операцію додавання цілого числа та рядка:

```
>>> 43 + "25"
```

приведе до виведення повідомлення про синтаксичну помилку.

Для перетворення одного типу даних на інший у мові Python застосовуються спеціальні функції.

Розглянемо основні функції перетворення одного типу даних на інший.

**bool([об'єкт])** — перетворення об'єкта на логічний тип (приклад 2).

Із прикладу видно, що функція bool повертає значення False у випадку, якщо об'єкт дорівнює нулю або порожній, інакше — значення True.

**int([об'єкт [,<система числення>]])** — перетворення об'єкта на ціле число. Система числення, у якій подається об'єкт, може бути десятковою, вісімковою, шістнадцятковою. За замовчуванням — десяткова система (приклад 3).

**float(ціле число або рядок)** — перетворення цілого числа або рядка на число дійсного типу (приклад 4).

**str(об'єкт)** — перетворення об'єкта на рядок (приклад 5).

**list(послідовність)** — перетворення елементів послідовності на список (приклад 6).

**Приклад 1.**
```
>>> x_2 = 300
>>> type(x_2)      #визначення типу змінної x_2
<class ‹int›>      #змінна x_2 посилається на тип int
```

**Приклад 2.**
```
>>> bool(0), bool(1), bool([4,6]), bool(""), bool("файл")
(False, True, True, False, True)
```

**Приклад 3.**
```
>>> int(13.5), int("71"), int("A", 16), int("71", 8)
(13, 71, 10, 57)
```
(пояснення: значення функції int("71", 8) має двійкове значення 111001, що еквівалентно 57)

**Приклад 4.**
```
>>> float(25), float("13.75")
(25.0, 13.75)
```

**Приклад 5.**
```
>>> str(149), str([21, 16, 35]), str({"x" :10})
('149', '[21, 16, 35]', "{'x'10}")
```

**Приклад 6.**
```
>>> list("2468")          #перетворення рядка
['2', '4', '6', '8']      #список
>>> list((2,4,6,8))       #перетворення кортежу
[2, 4, 6, 8]              #список
```

На рис. 1 зображено найпростішу програму додавання двох чисел із перетворенням типу str на тип int, яку збережено з іменем prog_03.

```
# Програма складання чисел з перетворенням типу str у тип int
x = int(input("x= ")) # увести, наприклад, 26
y = int(input("y= ")) # увести, наприклад, 9
print(x + y)          # виведення суми чисел
input()               # очікування натиснення Enter
```

Рис. 1. Програма додавання чисел із перетворенням типу str на тип int

За замовчуванням ці змінні мають тип str, тому їм потрібно встановити відповідний тип.

Після запуску програми на екрані з'явиться запрошення на введення значення змінної *x*. Уведемо, наприклад, число 26 і натиснемо клавішу Enter. З'явиться таке саме запрошення для введення значення змінної *y*. Уведемо, наприклад, число 9 і натиснемо Enter. У результаті отримаємо результат 35.

Після того як змінні в програмі вже не потрібні, їх можна *видалити за допомогою функції* del. Спроба використати змінні *x* або *y* після виконання такого фрагмента (приклад 7) призведе до видачі повідомлення про помилку, оскільки змінну x видалено*.

**Приклад 7.**

```
>>> x = 10; x; y = 20; y
10
20
>>> del x, y
>>> x
```

* Автоматичний збирач сміття в Python.

## Запитання для перевірки знань

1. Яка функція призначена для визначення типу даних?
2. Які операції можуть виконуватися над даними типу int?
3. Для чого призначено функцію bool?
4. На скільки типів даних може посилатися змінна в програмі?
5. Як можна видалити з програми непотрібні імена змінних?
6. Який результат буде отримано після виконання інструкції >>> int("D", 16)?
7. Змініть інструкцію >>> 77 * '89' так, щоб її було правильно виконано.

## Завдання для самостійного виконання

1. Визначте тип змінної sk_1 після виконання двох інструкцій:
   ```
   >>> s = "Python"
   >>> s = 21
   ```
2. Визначте, який буде отримано результат після виконання інструкцій:
   ```
   >>> int(41.5), int("60"), int("F", 16)
   ```
3. Визначте результат виконання інструкції:
   ```
   >>> list("40, 22, 5, 66")
   ```
   Доведіть, що в інструкції немає помилок.
4. Запишіть і виконайте інструкцію перетворення вісімкового числа 47 на десяткове. Доведіть, що інструкцію виконано правильно.
5. Запишіть і виконайте інструкцію перетворення слова «процесор» на логічний тип. Доведіть, що інструкцію виконано правильно.
6. Складіть і виконайте програму додавання чисел 37 і 29.7. Доведіть, що програму виконано правильно.

# 2.3. Оператори і вирази

*Пригадайте, які операції над числами використовували ви в процесі програмування у 8 і 9 класах. Які для цього застосовувалися оператори?*

Операції над об'єктами виконуються за допомогою відповідних операторів. Об'єкти, над якими виконуються операції, називають операндами. Кожний оператор може виконувати операції над строго визначеними для нього типами операндів.

Залежно від типу об'єктів, над якими виконуються операції, оператори групуються в *арифметичні, логічні, порівняння, присвоювання* й ін.

До арифметичних операторів належать такі загальновідомі оператори:

- додавання (+)
- віднімання (–)
- множення (*)
- ділення (/)

Арифметичні оператори призначені для виконання операцій над числами. Якщо операція виконується над цілим і дійсним числами, то ціле число буде спочатку перетворене на дійсний тип, а потім виконуватиметься операція над дійсними числами. Результатом операції в цьому випадку буде число дійсного типу. Результатом операції ділення завжди буде число дійсного типу.

Окрім звичайних арифметичних операцій, у мові Python також застосовуються такі операції (табл. 1).

Таблиця 1. Деякі операції у мові Python

| Операція | Позначка | Приклад |
|---|---|---|
| Цілочислове ділення (без остачі) | // | У результаті виконання операції 10.0 // 3.0 отримаємо результат 3.0 |
| Ділення за модулем (остача від ділення) | % | У результаті виконання операції 10 % 3.0 отримаємо результат 1.0 |
| Піднесення до степеня | ** | У результаті виконання операції 10 ** 2 отримаємо результат 100 |

У мові Python використовуються арифметичні оператори з присвоюванням (табл. 2).

Таблиця 2. Арифметичні оператори з присвоюванням

| Оператор | Позначка | Приклад |
|---|---|---|
| Збільшення значення змінної на вказану величину | += | x += 8 (еквівалентно x = x + 8) |
| Зменшення значення змінної на вказану величину | –= | x –= 8 (еквівалентно x = x – 8) |
| Множення значення змінної на вказану величину | *= | x *= 8 (еквівалентно x = x * 8) |
| Ділення значення змінної на вказану величину | /= | x /= 8 (еквівалентно x = x / 8) |

Один із варіантів імпортування можна реалізувати за допомогою інструкції: **from decimal import Decimal**. Наприклад, можна виконати такі інструкції:

```
>>> from decimal import Decimal          #імпортування
>>> Decimal("1.123") – Decimal("0.5")    #виконання операції
Decimal('0.623')                         #результат
```

Для точного виконання операцій над числами в мові **Python** призначений модуль **decimal**, у якому є функція **Decimal**.

Цей модуль слід імпортувати до коду програми. Можна імпортувати модуль або його частину, наприклад, окрему функцію.

Оператори порівняння порівнюють значення об'єкта, який розташовано ліворуч від оператора, зі значенням об'єкта, який розташовано праворуч від цього оператора. Якщо умова виконується, повертається значення True, інакше — False.

Склад, позначення та приклади використання операторів наведено в табл. 3.

Таблиця 3. Оператори порівняння

| Позначення | Назва | Пояснення | Приклад |
|---|---|---|---|
| == | Дорівнює | Якщо значення операндів однакові, повертається значення True, інакше — False | >>> 23 == 40<br>False |
| != | Не дорівнює | Якщо значення операндів не однакові, повертається значення True, інакше — False | >>> != 13<br>True |
| > | Більше | Якщо значення операнда зліва більше за значення справа, повертається значення True, інакше — False | >>> 5 > 204<br>False |
| < | Менше | Якщо значення операнда зліва менше за значення справа, повертається значення True, інакше — False | >>> 5 < 204<br>True |
| >= | Більше або дорівнює | Якщо значення операнда зліва більше або дорівнює правому, повертається значення True, інакше — False | >>> 17 >= 17<br>True |
| <= | Менше або дорівнює | Якщо значення операнда зліва менше або дорівнює правому, повертається значення True, інакше — False | >>> 43 <= 17<br>False |

У мові Python використовуються такі логічні оператори: not (ні), or (або), and (і). Вони виконуються над даними логічного типу, які мають два значення: True (істинне) і False (хибне). Результати виконання цих операторів наведені в табл. 4.

Таблиця 4. Результати виконання логічних операторів

| X | Y | not x | x or y | x and y |
|---|---|---|---|---|
| False | False | True | False | False |
| True | False | False | True | False |
| False | True | True | True | False |
| True | True | False | True | True |

Із табл. 4 видно, що оператор not є унарним, тобто діє лише для одного операнда. Результатом його виконання є значення, протилежне значенню операнда x. Результатом виконання оператора or є значення False лише в тому випадку, коли обидва

операнди мають значення False. У всіх інших випадках отримується значення True.

Результатом виконання оператора and є значення True лише у випадку, коли обидва операнди мають значення True. У всіх інших випадках отримується значення False.

Оператори над послідовностями виконують операції над списками, кортежами та рядками.

Далі роз'яснено сутність та наведено приклади виконання операцій над рядками (табл. 5) (операції над іншими типами послідовностей описано в розділі 4).

**Python** входить до трійки найбільш значущих мов у сфері машинного навчання й аналізу великих обсягів даних. Як універсальна мова **Python** має застосування практично скрізь, навіть в ігровій індустрії.

Таблиця 5. Операції над рядками

| Операція | Позначення | Приклад |
|---|---|---|
| Об'єднання (призначене для з'єднання послідовностей) | + | У результаті виконання операції над рядками "послі" + "довність" отримаємо результат "послідовність" |
| Повторення (призначене для повторення послідовності задану кількість разів) | * | >>> "ав" * 5<br>'авававававав' |
| Перевірка на входження послідовностей одна до іншої | in | Якщо одна послідовність входить до іншої, видається значення True, інакше — False. Приклад:<br>>>> "виконання" in "Результатом виконання операції"<br>True<br>>>> "оператор" in "Результатом виконання операції"<br>False |
| Перевірка на невходження однієї послідовності до іншої | not in | Якщо одна послідовність входить до іншої, видається повідомлення False, інакше — True. Приклад:<br>>>> "байт" not in "біт"<br>True<br>>>> "байт" not in "байт і біт"<br>False |

Над рядками можуть виконуватися також дві операції з присвоюванням (табл. 6).

Таблиця 6. Операції з присвоюванням

| Операція | Позначення | Приклад |
|---|---|---|
| Об'єднання з присвоюванням | += | >>> s = "арифме"; s += "тичні"; print(s)<br>арифметичні |
| Повторення задану кількість разів | *= | >>> s = "пр"; s *= 6; s<br>'прпрпрпрпрпр' |

**Python** дуже популярна для написання серверної частини веб-сайтів для мобільних і веб-застосунків та побудови різноманітних сервісів.

Оператори використовуються у виразах. Поняття виразу в програмуванні аналогічне поняттю виразу в математиці.

Вираз мовою програмування складається з операндів, операторів і круглих дужок та визначає порядок виконання операцій над даними. Операнди виразу — це змінні, константи, функції, методи. Найпростіший вираз складається з одного операнда, наприклад, константи або змінної.

Залежно від типу операндів і операцій, що використовуються у виразі, розрізняють вирази:
- *арифметичні* (результат арифметичного типу),
- *логічні* (результат логічного типу),
- *рядкові* (результат рядкового типу).

Для кожного типу операцій існують чіткі правила їх запису та виконання, з якими ми будемо знайомитися поступово в процесі їх використання.

Наведемо основні правила для арифметичних виразів.

Для мови **Python** існує велика кількість бібліотек для наукових досліджень, серед яких NumPy, ScPy, Matplotlib.

- Не можна застосовувати підрядкові й надрядкові символи. Наприклад, вираз $2.5*(a-b^2)$ є неправильним.
  Його слід записати так: 2.5*(a–b*b).
- Не можна записувати дві або більше операцій безпосередньо одна за одною. Наприклад, вираз a*–b є некоректним.
  Його слід записати так: a*(–b).
- Кожній дужці, що відкривається, у виразі має відповідати дужка, що закривається.
- Типи операндів виразу мають бути узгоджені. Якщо автоматично вони не узгоджуються, слід використати засоби перетворення типів, що вже були описані раніше.

Операції в арифметичному виразі виконуються з урахуванням їх пріоритету, а операції, що мають однаковий пріоритет, — у порядку їх запису, тобто зліва направо.

## Запитання для перевірки знань

1. Який тип даних отримується після виконання операції ділення?
2. Який тип результату отримується після цілочислового ділення?
3. Поясніть сутність арифметичних операторів із присвоюванням.
4. Які існують типи операторів?
5. У чому полягає сутність ділення за модулем?
6. Поясніть сутність оператора !=.
7. Як виконується логічний оператор and?
8. Поясніть сутність оператора or.
9. Які операції виконуються над рядками?
10. Поясніть сутність операції повторення рядків.
11. Наведіть приклад перевірки входження одного рядка до іншого.
12. Наведіть приклад використання оператора повторення з присвоюванням.

## Завдання для самостійного виконання

1. Дано два числа. Виконайте їх цілочислове ділення та ділення за модулем.
2. Дано два числа. Виконайте їх додавання та множення з використанням відповідних арифметичних операторів із присвоюванням.
3. Дано два числа. Виконайте їх порівняння на рівність і перевірте, чи більше перше число за друге.
4. Виконайте об'єднання рядків слів "опрацю" і "вання".
5. Складіть найпростіший код перевірки входження рядка "Чемпіонат України" в рядок "Чемпіонат України з футболу".
6. Складіть найпростіший код з'єднання рядків "зменшення значення" і "змінної" з використанням оператора об'єднання з присвоюванням.

# 2.4. Модулі, функції і методи для опрацювання числових даних

*Пригадайте, які функції опрацювання числових даних ви використовували у 8 і 9 класах. Чи доводилося вам самостійно розробляти функції опрацювання чисел?*

Числа можуть подаватися в десятковій, двійковій, вісімковій і шістнадцятковій системах числення. У процесі виконання арифметичних операцій над числами в різних системах числення вони автоматично перетворюються на десяткову систему числення.

У мові Python використовуються цілі числа (тип int), дійсні (тип float) і комплексні (тут не розглядаються). Якщо в арифметичній операції використовуються різні типи чисел, то числа типу int автоматично перетворюються на тип float, і результат отримується типу float.

Далі будуть розглядатися операції лише в десятковій системі числення.

Для роботи з числами можуть застосовуватися вбудовані у мову Python функції, основні з яких наведено в табл. 1.

Таблиця 1. Основні функції для роботи з числами

| Функція | Призначення | Приклад |
|---|---|---|
| **int**([<об'єкт>]) | Перетворює об'єкт на ціле число | >>> int(8.7), int("57")<br>(8, 57) |
| **float**([<число або рядок>]) | Перетворює ціле число або рядок на дійсне число | >>> float(23), float("11.7")<br>(23.00, 11.7) |
| **round**(<число>[,<кількість знаків після коми>])) | Повертає найближче ціле, якщо кількість знаків не вказана, а також кількість знаків після коми, якщо вона вказана | >>> round(0.47), round(45.347, 1)<br>(0, 45.3) |
| **abs**(число) | Повертає абсолютне значення | >>> abs(22), abs (–66)<br>(22, 66) |
| **pow**(<число>,<степінь>) | Повертає число в степені | >>> pow(10, 2), pow (2, 4)<br>(100, 16) |
| **max**(<числа через кому>) | Повертає максимальне значення | >>> max(9, 4, 7), max(5, 9.7, 3)<br>(9, 9.7) |
| **min**(<числа через кому>) | Повертає мінімальне значення | >>> min(21, 5, 19), min(7, 2.5, 6)<br>(5, 2.5) |
| **sum**([<числа>],<початкове значення>) | Повертає суму чисел. Якщо вказане початкове значення, то воно додається до суми | >>> sum([4, 5, 7]), sum([12, 3], 33)<br>(16, 48) |

Модуль **math** містить константи:

**число pi**:

```
>>> import math
>>> math.pi
3.14159265389793
```

**число e**:

```
>>> math.e
2.718281828459045
```

Усі типи даних мови Python є класами. Класи містять методи. *Метод* — це програма, яка виконує ту чи іншу функцію. Метод викликається для конкретного об'єкта. Для його виклику спочатку вказується об'єкт, потім крапка, за якою слідує ім'я метода: <об'єкт>.<ім'я методу>. Кожний клас підтримує свої методи.

Модуль math, який містить стандартні константи та функції, використовують для роботи з числами. Для роботи з константами та функціями необхідно імпортувати його в програму за допомогою інструкції import math.

Найуживаніші функції модуля math наведено в табл. 2.

Таблиця 2. Найчастіше вживані функції модуля math

| Функція | Призначення | Приклади |
|---|---|---|
| **sqrt()** | Корінь квадратний | >>> math.sqrt(85)<br>9.219544457292887 |
| **log10()** | Логарифм десятковий | >>> math.log10(15)<br>1.1760912590556813 |
| **ceil()** | Найближче більше ціле | >>> math.ceil(3.213)<br>4 |
| **floor()** | Найближче менше ціле | >>> math.floor(6.79)<br>6 |
| **pow**(число, степінь) | Підносить число до степеня | >>> math.pow(8, 3)<br>512.0 |
| **fmod()** | Остача від ділення | >>> math.fmod(35, 3)<br>2.0 |
| **factorial()** | Факторіал числа | >>> math.factorial(4)<br>24 |

Модуль random містить функції, які генерують випадкові числа. На початку використання функцій необхідно цей модуль імпортувати в програму за допомогою команди import random.

У табл. 3 наведено найуживаніші з них.

Модуль **math** містить:

- стандартні тригонометричні функції: (sin(), cos(), tan());
- обернені тригонометричні функції: asin(), acos(), atan();
- функцію перетворення радіанів на градуси degrees();
- функцію перетворення градусів на радіани radians();
- функцію експоненти exp().

Пам'ятайте, що для виконання наведених у таблиці прикладів спочатку необхідно імпортувати модуль **math** за допомогою команди >>> import math.

Таблиця 3. Деякі функції модуля random

| Функція | Призначення | Приклад |
|---|---|---|
| **random()** | Генерує випадкове число від 0.0 до 1.0 [0.0, 1.0) 1.0 не включено до діапазону | >>> import random<br>>>> random.random()<br>0.4632200164843052 |
| **uniform**(початок,кінець) | Генерує дійсне випадкове число в діапазоні від «початок» до «кінець». 1.0 включено до діапазону | >>> random.uniform(3, 9)<br>4.951275580428769 |
| **randint**(початок,кінець) | Генерує ціле випадкове число в діапазоні від «початок» до «кінець». 1.0 не включено до діапазону | >>> random.randint(3, 8)<br>7 |
| **choice**(послідовність) | Вибирає з послідовності (рядка, списку або кортежу) випадковий елемент | >>> random.choice(«Python»)<br>'h' |

## Запитання для перевірки знань

1. У яких системах числення можуть подаватися числа?
2. Для чого призначено функцію float?
3. Для чого призначено модуль math?
4. Для чого призначено функцію pow?
5. Для чого призначено модуль random?
6. Наведіть приклад використання функції round.
7. Яку структуру має функція sum?

## Завдання для самостійного виконання

1. Визначте максимальне та мінімальне число в послідовності: 38, 20, 5, 40, 13.
2. Початкове значення суми дорівнює 14. Додайте до неї такі числа: 23, 50, 37, 13.
3. Визначте випадкове число в послідовності: 23, 40, 29, 33, 17.
4. Обчисліть суму факторіалів чисел 5 і 7.

# 3. Реалізація базових алгоритмічних конструкцій
# 3.1. Реалізація алгоритмів із розгалуженням

*Відомо, що існує три базові структури алгоритмів: слідування, розгалуження та повторення. Пригадайте, якими операторами мови ви, можливо, реалізовували розгалуження й повторення.*

В алгоритмах із розгалуженням залежно від умови виконуються ті чи інші інструкції.

Існують три типи розгалуження: *одноальтернативне* (неповне галуження), *двоальтернативне* (повне галуження) *і багатоальтернативне* (вибір, варіант).

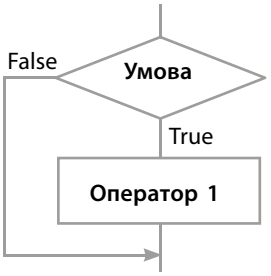


Рис. 1. Графічна схема одноальтернативного розгалуження

- Одноальтернативне розгалуження

Графічну схему одноальтернативного розгалуження зображено на рис. 1.

Одноальтернативне розгалуження виконується так. Перевіряється Умова, і якщо вона істинна (True), то виконується Оператор 1. Інакше буде виконуватися оператор, який розташовано безпосередньо за оператором розгалуження.

У мові Python цей тип розгалуження реалізується оператором умовного переходу в неповній формі:

```
if <логічний вираз>:
    <блок S>
```

**Приклад 1.**

I варіант запису:

```
if x > 4:
    y = 2 * x + 5;
    z = y + x * x
```

II варіант запису:

```
if x > 4:
    y = 2 * x + 5; z = y + x * x
```

В один рядок 2 оператори записують через «;».

У наведеному фрагменті (приклад 1) умовою є логічний вираз x>4. Якщо він має значення True, то обчислюється значення виразу y = 2 * x + 5, потім — виразу z = y + x * x. Після завершення їх обчислення буде виконуватися оператор, який розташовано безпосередньо за оператором умовного переходу.

Якщо ж логічний вираз x>4 має значення False, вказані вирази не обчислюються, а управління буде одразу передано оператору, що слідує за оператором умовного переходу.

Звернемо увагу на те, що в наведеному прикладі в операторі умовного переходу міститься блок із двох операторів, який починається після чотирьох пробілів.

**Приклад 2.**

У банк покладено 5000 грн під 20 % річних. Якщо гроші знімаються раніше зазначеного терміну, відсотки не нараховуються. Програму обчислення реальної суми, яку отримано через *k* днів, зображено на рис. 2.

*Примітка.* У змінній *s* спочатку зберігається початкова сума вкладу, а потім сума, яку отримано з банку.

```
s = 5000                # початкова сума вкладу
k = int(input("Через скільки днів?"))
if k > 366:             # після одного року?
    s = s + s * 0.2     # обчислення суми вкладу
print("s= ", s)         # виведення суми вкладу
input()                 # очікування натиснення Enter
```

Рис. 2. Код обчислення суми вкладу

• Двоальтернативне розгалуження

Графічну схему двоальтернативного розгалуження зображено на рис. 3.

Двоальтернативне розгалуження виконується так: перевіряється Умова і, якщо вона має значення True, то виконується Оператор 1. Інакше виконується Оператор 2.

У мові Python цей тип розгалуження реалізується оператором такої структури:

```
if <логічний вираз>:
    <блок s1>
else:
    <блок s2>
```

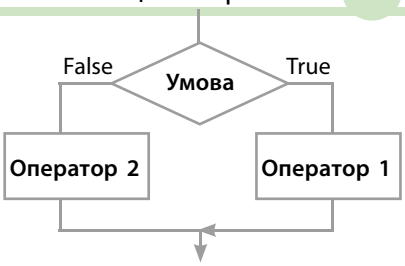


Рис. 3. Графічна схема двоальтернативного розгалуження

**Приклад 3.**

```
if x==10:
    y = "Python"
    print ("y=", y)
else:
    y = "Pascal"
    print ("y=", y)
```

У цьому прикладі, якщо значення змінної *x* дорівнює 10, виводиться повідомлення Python, інакше — Pascal.

**Приклад 4.**

Довжина сторони квадрата дорівнює a, радіус кола — r. Чи можна у квадрат уписати коло?

Програму розв'язування задачі наведено на рис. 4.

```
r = int(input("Увести радіус: "))
a = int(input("Увести сторону квадрата: "))
if r == int(a / 2):    # перевірка умови
    print("Можна вписати")
else:                  # інакше
print("Вписати не можна")
input()
```

Рис. 4. Код визначення можливості вписати коло у квадрат

Звернемо увагу на те, що безпосередньо за ключовими словами if і else може слідувати один оператор або блок операторів. Нагадаємо, оператори блоків починаються від початку рядка через чотири пробіли.

**Приклад 5.**

Із м. Києва до м. Чернігова на велосипеді о 8.00 виїхав Василь й рухався 5,3 год. Потім о 9.00 виїхав Микола і рухався за тим самим маршрутом 4,2 год. Хто з велосипедистів прибув до Чернігова першим?

Програму розв'язування задачі наведено на рис. 5.

```
if 8.0 + 5.3 < 9.0 + 4.2:
    print("Першим буде Василь")
else:
    print("Першим буде Микола")
input()
```

Рис. 5. Код визначення першого велосипедиста

• Багатоальтернативне розгалуження

У цьому типі розгалуження реалізується розгалуження з багатьма варіантами вибору. Графічну схему розгалуження з трьома можливими варіантами зображено на рис. 6.

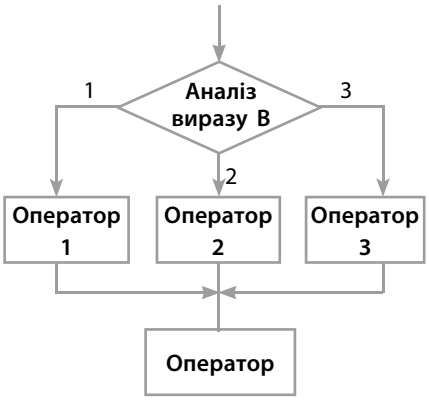


Рис. 6. Графічна схема розгалуження з трьома варіантами вибору

Вираз B повинен мати ціле значення. Якщо його значення дорівнює одиниці, виконується Оператор 1, якщо вираз має значення два, виконується Оператор 2, а якщо три — Оператор 3. Після виконання одного з цих операторів виконується Оператор, що міститься після оператора багатоальтернативного розгалуження.

У випадку якщо вираз B не дорівнює жодному з перелічених значень, одразу виконується Оператор (рис. 6).

У мові Python розгалуження за багатьма варіантами вибору реалізується оператором такої структури:

```
if <вираз> == <значення_1>:
    <блок_1 >
elif <вираз> == <значення_2>:
    <блок_2>
      …
elif <вираз> == <значення_N>:
    <блок_N>
else:
    <блок_N+1>
```

Отже, кожне значення виразу має бути унікальним константним виразом. Значення виразу порівнюється з кожним значенням в операторах elif у порядку їх розташування. Якщо певне значення вираз збігається зі значенням оператора elif, то виконується той блок операторів, що розташований безпосередньо за цим оператором. Якщо значення вираз не збігається з жодним значенням в операторах elif, виконується блок N+1.

**Приклад 6.**

Призерами чемпіонату України з футболу є команди: 1 — «Динамо», 2 — «Шахтар», 3 — «Зоря». За номером призера необхідно визначити назву команди.

Програму розв'язування задачі зображено на рис. 7.

```
nom = int(input("Увести номер призера: "))
if nom == 1:
    print("Команда Динамо")
elif nom == 2:
    print("Команда Шахтар")
elif nom == 3:
    print("Команда Зоря")
else:
    print("Це не номер призера")
input()
```

Рис. 7. Код визначення призера

## Запитання для перевірки знань

1. Які існують типи розгалужень алгоритмів?
2. Яким оператором мови Python реалізується одноальтернативне розгалуження?
3. Сформулюйте сутність двоальтернативного розгалуження.
4. Які помилки є в записі оператора:

```
if x = 3
    y = 2,  p = y = y * 4.1
```

5. Наведіть задачу, для розв'язання якої використовується двоальтернативне розгалуження.
6. Запишіть структуру оператора двоальтернативного розгалуження.
7. Поясніть на прикладі сутність багатоальтернативного розгалуження.
8. Чи можна будь-який алгоритм розгалуження реалізувати тільки оператором одноальтернативного розгалуження? Наведіть приклад.

## Завдання для самостійного виконання

1. Оклад працівника або працівниці дорівнює *s* грн. За якісне й дострокове виконання завдання нараховується премія в розмірі 50 % окладу. Розробіть програму визначення реальної заробітної платні.
2. Дано два дійсних числа. Розробіть програму, за допомогою якої менше число замінюється нулем, а більше — одиницею.
3. Дано ціле число *n*. Розробіть програму, за допомогою якої це число збільшується на 7, якщо воно більше 15, і зменшується на 5, якщо воно менше або дорівнює 15.
4. Тренерка формує команду для гри в баскетбол з учениць, які на зріст не нижчі ніж 180 см. Розробіть програму визначення, чи потрапить у цю команду дівчина зростом *h* см?
5. Дано рівносторонній трикутник зі стороною *b*. Розробіть програму визначення, чи можна в трикутник уписати коло з радіусом r.
6. Відомі три найкращі результати учнівства школи в стрибках у довжину. Розробіть програму, за допомогою якої за результатами стрибків визначаються прізвища учнів.
7. У п'ятницю в 9 класі такі уроки: 1 — історія, 2 — математика, 3 — географія, 4 — інформатика, 5 — фізкультура. Розробіть програму, за допомогою якої визначається назва предмета за його номером у розкладі.
8. Знайдіть в Інтернеті та складіть список із п'яти найбільших міст України. Розробіть програму, за допомогою якої визначається назва міста за його номером у цьому списку.

# 3.2. Вкладені оператори умовного переходу

*На практиці трапляються випадки, коли виконання або невиконання події залежить не від однієї умови, а від кількох. Пригадайте такі випадки. Як, на вашу думку, можна їх реалізувати операторами умовного переходу?*

**Вкладені оператори умовного переходу** — це оператори умовного переходу, які входять до складу інших операторів умовного переходу.

Існують різні конструкції вкладених операторів. Один із варіантів конструкції вкладених операторів умовного переходу подано на графічній схемі (рис. 1).

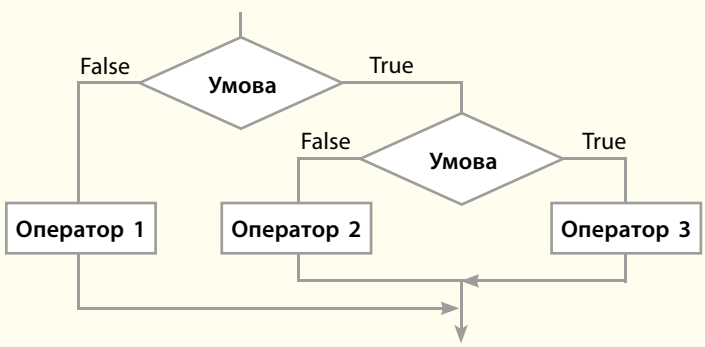


Рис. 1. Графічна схема алгоритму зі вкладеним умовним переходом

Розробники мови Python є прихильниками певної філософії програмування, яку називають «The Zen of Python» («Дзен Пайтона»). Її текст можна отримати в інтерпретаторі Python за допомогою команди import this (лише один раз за сесію). Автором цієї філософії вважається Тім Пейтерс.

**Текст філософії «The Zen of Python» («Дзен Пайтона»)**

- Гарне краще за потворне.
- Явне краще за неявне.
- Просте краще за складне.
- Складне краще за заплутане.
- Плоске краще за вкладене.
- Розріджене краще за щільне.
- Легкість читання має значення.
- Особливі випадки не є настільки особливими, щоб порушувати правила. Хоча практичність є важливішою за бездоганність.
- Помилки ніколи не повинні проходити непомітно. Якщо їх приховування не прописано явно.
- Зустрівши неоднозначність, опирайтеся спокусі вгадати.

Розглянемо на прикладі алгоритм обчислення виразу

$$y=\begin{cases} ax, \text{ якщо } x>0 \text{ і } a\geq 0. \\ 2ax, \text{ якщо } x>0 \text{ і } a<0 \\ 2, \text{ якщо } x\leq 0 \end{cases}$$

На рис. 2 зображено фрагмент схеми алгоритму, що реалізує вираз ax, якщо x>0 і a≥0.

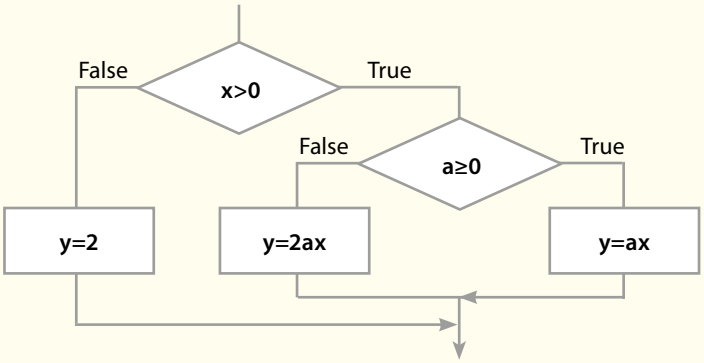


Рис. 2. Графічна схема алгоритму обчислення значення виразу

Мовою Python цей алгоритм можна реалізувати так:

```
If x > 0:                    #перший оператор if
    If a >= 0:               #другий оператор if
      y = a * x
    else:                    #для другого оператора if
      y = 2 * a * x
else:                        #для першого оператора if
    y = 2
```

Алгоритм, наведений на рис. 2, реалізується програмою, яку зображено на рис. 3.

```
a = int(input("Увести a: "))
x = int(input("Увести x: "))
if x > 0:                    # якщо x>0
    if a >= 0:
        y = a * x            # гілка Так другого оператора if
    else:
        y = 2 * a * x        # гілка Ні другого оператора if
else:
    y = 2                    # гілка Ні першого оператора if
print("y = ", y)
input()
```

Рис. 3. Код обчислення значення арифметичного виразу

Інший варіант конструкції з вкладеними операторами умовного переходу подано на рис. 4.

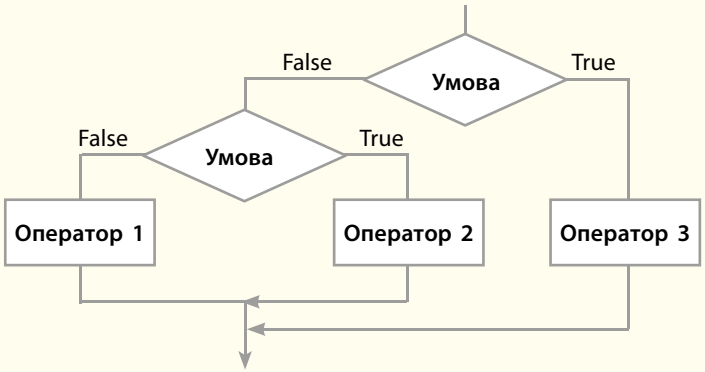


Рис. 4. Графічна схема алгоритму зі вкладеним умовним переходом

Розглянемо ще один приклад застосування конструкції зі вкладеними операторами умовного переходу для розв'язування задачі.

**Python** використовується для розв'язування великої кількості як наукових, так і бізнес-завдань.

У науковій сфері мову **Python** широко використовують західні вчені-непрограмісти (математики, фізики, біологи) завдяки простоті вивчення та наявності популярних додаткових потужних бібліотек.

**Приклад.**

За рейтингом УЄФА футбольні команди розташовано так: «Реал», «Барселона», «Баварія», «Челсі», «Атлетіко», «Бенфіка», «Шальке-04», «Порту», «Арсенал», «Манчестер». На рис. 5 наведено програму, яка за номером команди визначає країну.

Наведено один із варіантів виконання програми:

```
Увести номер команди: 9
країна - Англія
```

```
kom = int(input("Увести номер команди: "))
if kom == 1 or kom == 2 or kom == 5:                    # команда Іспанії
    kr = "Іспанія"
else:
        if kom == 3 or kom == 7:                        # команда Німеччини
            kr = "Німеччина"
        else:
                if kom == 4 or kom == 9 or kom == 10:   # команда Англії
                    kr = "Англія"
                else:
                    if kom == 6 or kom == 8:            # команда Португалії
                        kr = "Португалія"
                    else:
                        kr = "Рейтинг невідомий"
print("країна - ", kr)
input()
```

Рис. 5. Код визначення назви країни

## Запитання для перевірки знань

1. Поясніть сутність вкладеного оператора умовного переходу.
2. Накресліть графічну схему алгоритму зі вкладеним умовним переходом.
3. Поясніть правило запису блоку операторів у вкладеному операторі умовного переходу.
4. Наведіть приклад алгоритму, що реалізується за допомогою вкладеного оператора умовного переходу.
5. Проаналізуйте порядок виконання алгоритму (див. рис. 2) для випадку x>0, a<0.
6. Проаналізуйте порядок виконання коду (див. рис. 3), якщо вводяться такі значення: a=0, x=0.

## Завдання для самостійного виконання

1. Дано три дійсних числа a, b, c. Якщо a>b>c, кожне число збільшується вдвічі, інакше — кожне число зменшується на одиницю. Розробіть програму реалізації цього завдання.
2. Дано три сторони трикутника. Розробіть програму, за допомогою якої визначається, чи є цей трикутник прямокутним.
3. Дано три сторони трикутника. Розробіть програму для визначення, чи є цей трикутник рівнобедреним.
4. Літак вилітає за розкладом із Києва в Мюнхен, якщо швидкість вітру в Києві менше v, а хмарність у Мюнхені не менше h. Розробіть програму для визначення, чи вилетить літак за розкладом.
5. Шкільна команда шахістів із трьох учнів потрапляє до фінальних міських змагань у випадку, якщо у відбірних змаганнях кожен учень набере не менше 5 очок. Розробіть програму, за допомогою якої визначається, чи потрапить команда до фінальних змагань, якщо перший учень набрав a очок, другий — b очок і третій — c очок.
6. У Всеукраїнській учнівській олімпіаді з інформатики з однієї школи беруть участь два учні. Щоб потрапити до української команди для участі в міжнародній олімпіаді, потрібно набрати не менше k балів. Перший учень набрав a балів, а другий — b балів. Розробіть програму, за допомогою якої визначаються всі варіанти потрапляння до цієї команди учнів школи.
7. Перед останнім туром чемпіонату України з футболу склалася така ситуація. Щоб команда «Динамо» (Київ) стала чемпіоном, їй потрібно виграти в команди «Ворскла», і щоб «Шахтар» водночас програв «Дніпру». Розробіть алгоритм зі вкладеним умовним оператором визначення, чи стане «Динамо» чемпіоном.

# 3.3. Реалізація циклічних алгоритмів

*Пригадате, яку алгоритмічну структуру називають повторенням (циклом), які існують види циклів.*

Для запису алгоритмів із повторенням (циклів) мовою Python використовують два види операторів циклу: з параметром та з умовою.

## 3.3.1. Цикли з параметрами

**Повторення (цикл)** — це алгоритмічна структура, за допомогою якої та сама послідовність дій виконується багаторазово для різних значень змінних.

Серію інструкцій, які виконуються багаторазово під час виконання циклу, називають тілом циклу.

У циклах із параметром (циклах зі змінною циклу) кількість повторень інструкцій тіла циклу заздалегідь відома.

Існують різні варіанти циклів із параметрами. Структуру одного з них зображено на рис. 1. Тут змінна i — це параметр циклу (змінна циклу). Блок Оператор є тілом циклу. У блоці може бути одна інструкція або кілька. Змінна n містить кінцеве значення змінної циклу, а змінна a — її початкове значення.

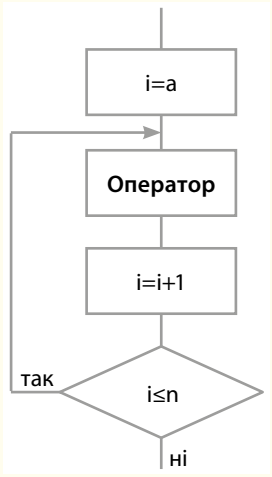


Рис. 1. Фрагмент графічної схеми циклу з параметром

На рис. 2 наведено фрагмент блок-схеми циклічного алгоритму для отримання таблиці множення на 8. Результат множення чисел від 1 до 10 на 8 отримується у змінній p й одразу виводиться. У цьому алгоритмі тілом циклу є інструкції p=8*i, Виведення p, i=i+1, які виконуються 10 разів (можна записати скорочено i+=1). Інструкція i=1 виконує підготовку до реалізації інструкцій тіла циклу, а за допомогою інструкції i≤10 здійснюється перевірка завершення їх виконання.

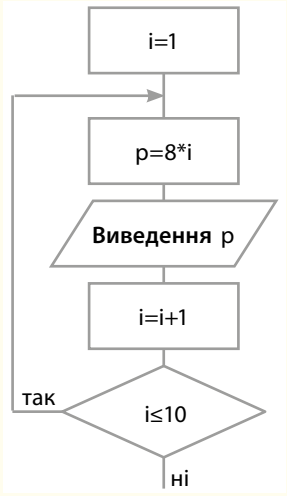


Рис. 2. Фрагмент графічної схеми циклічного алгоритму

Як бачимо, у цій схемі змінна i виконує функцію лічильника циклів. Як тільки значення цієї змінної стане більше 10, виконання циклу завершується. Такий принцип реалізації циклів із параметром застосовується в багатьох мовах програмування, у тому числі найсучасніших.

У мові Python теж застосовується змінна циклу, але її значення послідовно вибираються зі значень об'єкта.

Цикли з параметром у мові Python реалізуються оператором циклу for (для), який має таку загальну структуру:

```
for <змінна циклу> in <об'єкт>:
        <блок інструкцій тіла циклу>
    [ else:
        <блок інструкцій>      # виконується, якщо
                                 не використовується
                                 оператор break
]
```

Інструкції у квадратних дужках є необов'язковими. Якщо всередині циклу не використовується оператор **break** (із ним ми познайомимося пізніше), то після завершення циклу виконуватиметься блок інструкцій, що міститься після слова **else**.

де:

<об'єкт> — може бути рядок, список, словник та інші типи даних, які підтримують реалізацію циклу. Тип об'єкта програміст може також створити самостійно;

<змінна циклу> — поточне значення об'єкта. Початкове її значення — це перший елемент об'єкта. У другому циклі ця змінна набуде значення другого елемента об'єкта і так далі до останнього.

Блок інструкцій тіла циклу буде виконуватися до тих пір, доки змінна циклу послідовно не набуде усіх значень, що містяться в об'єкті. Наприклад, якщо об'єктом є квадрат, сторони якого набувають таких значень: 2, 5, 9, 10, то цикл буде виконуватися 4 рази для значень змінної циклу 2, 5, 9 і 10.

Найпростіша структура оператора for така:

```
for <змінна циклу> in <об'єкт>:
<блок інструкцій тіла циклу>
```

Розглянемо, як *виконується оператор циклу*.

| | |
|---|---|
| Крок 1 | На початку виконання циклу змінна циклу набуває значення першого елемента об'єкта. Оператор **in** генерує логічне значення **True**, і виконується блок інструкцій тіла циклу. |
| Крок 2 | На цьому кроці змінна циклу набуде значення другого елемента об'єкта й також буде генеруватися значення **True**, у результаті чого буде виконано блок інструкцій тіла циклу. |
| Крок 3 | Після того як усі елементи об'єкта будуть перебрані, оператор **in** згенерує значення **False**, блок інструкцій тіла циклу не виконається, а управління буде передано першій інструкції, що розташована безпосередньо за блоком інструкцій тіла циклу. |

### Приклад 1

Розробити програму реалізації алгоритму отримання таблиці множення на 8.

Блок-схему алгоритму отримання таблиці множення на 8 зображено на рис. 2, а програму його реалізації — на рис. 3.

У цьому прикладі застосовується функція range(). Загальна структура цієї функції така:

range ([<початок>,] <кінець> [, <крок> ])

Як бачимо, обов'язковим є лише параметр кінець. Саме така структура оператора використана в наведеному прикладі. За допомогою функції range(10) формується діапазон чисел від 0 до 10, тому змінна циклу спочатку набуде значення нуль. Але нам не потрібно множити число 8 на нуль. Тому значення змінної i одразу збільшується на одиницю (i = i+1). Після виконання трьох інструкцій тіла циклу змінна i набуде чергового значення функції range(), тобто значення одиниці, яке наступним оператором збільшується на одиницю, у результаті чого число 8 буде помножене на 2. Цей процес повторюється до отримання результату 8*10.

```
# об'єктом в операторі for є цілі числа
for i in range(10):
    i = i + 1    # можна записати як i += 1
    p = 8 * i
    print("8 *",i," = ",p)
input()
```

Рис. 3. Код множення чисел на 8

**Приклад 2.**

У банк покладено 10000 грн під 20 % річних. Визначити суму вкладу за кожний із п'яти років. Програму реалізації цього завдання зображено на рис. 4.

```
s = 10000
for i in range(5):
    i = i + 1          # можна записати як i += 1
    s = s + 0.2 * s    # можна записати як s += 0.2 * s
    print("за ",i," рік: ", s )
input()
```

Рис. 4. Код обчислення суми вкладу

**Приклад 3.**

Дано рядок символів. Програму виведення кожного символу з рядка через один пробіл і підрахунку в ньому кількості символів зображено на рис. 5.

У цьому прикладі об'єктом у структурі оператора for є рядок. Змінна циклу s послідовно набуває значень, починаючи з букви м до букви р. Аргумент end=" " в операторі print забезпечує виведення символів рядка через пробіл в одному рядку.

```
k = 0              # початкова кількість
                   символів
                   # об'єктом в операторі
                   for є рядок
for s in "мікропроцесор":
    print(s, end = " ")
    k = k + 1      # можна записати як k += 1
print()            # перехід на новий рядок
print("Кількість символів = ", k)
```

Рис. 5. Код обчислення кількості букв у рядку

Результат виконання програми має такий вигляд:

```
м і к р о п р о ц е с о р
Кількість символів = 13
```

**Приклад 4.**

У заданій послідовності комп'ютерних термінів підрахувати кількість конкретного терміна й вивести його стільки разів, скільки він повторюється.

На рис. 6 неведено код, за допомогою якого в заданій послідовності виводиться слово «біт» і підраховується його кількість.

```
k = 0                          # початкова кількість слів
                               # об'єктом у наступному
                               операторі for є список
for p in ['миша', 'біт', 'біт', 'принтер', 'біт']:
    if p == 'біт':
        print(p, end=" ")      # друк через один пробіл
        k += 1                 # обчислення кількості слів
print()                        # перехід на новий рядок
print("Таких слів: ", k)
```

Рис. 6. Код обчислення кількості заданих слів

Результат виконання програми має такий вигляд:

```
біт біт біт
Таких слів: три
```

# 3.3.2. Цикли з умовою

Існує два види циклів з умовою: цикли з передумовою і цикли з післяумовою.

У циклах з умовою кількість виконання інструкцій тіла циклу заздалегідь невідома: вона завершується після досягнення певної умови.

- Цикли з передумовою

Графічну схему циклу з передумовою зображено на рис. 7.

Блок **Оператор** — це одна або кілька інструкцій тіла циклу, які виконуються доти, доки умова має значення True (гілка **Так**). Умова виконання інструкцій тіла циклу перевіряється перед початком їх виконання. Це означає, що оператори тіла циклу можуть бути не виконані жодного разу.

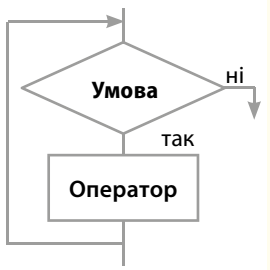


Рис. 7. Графічна схема циклу з передумовою

У мові Python цикли з передумовою реалізуються оператором **while**, який має таку структуру:

<початкове значення>
while <умова>):
    <блок інструкцій тіла циклу>
    <зміна початкового значення>
[ else:
    <інструкції, які не виконуються,
    якщо не використовується оператор break>
]

Тут <умова> — це вираз, який має значення True або False.

**Приклад 5.**

Автомобіль рухається зі швидкістю v км/год і раптом гальмує. За першу секунду його швидкість падає на 10 км/год, а за кожну наступну секунду — зменшується на 10 км/год від значення за попередню секунду. Через скільки секунд він зупиниться?

На рис. 8 наведено код програми моделювання процесу гальмування. Тут змінна k — це кількість секунд гальмування, змінна y — значення швидкості гальмування автомобіля за поточну секунду.

```
v = int(input("Увести швидкість автомобіля v = "))
y = 10          #гальмування за першу секунду
k = 0           #змінна k - кількість секунд
while v > 0:
    v = v - y   #поточне значення швидкості
    y = y + 10 #поточне значення гальмування
    k += 1
print("Автомобіль зупиниться через", k, "сек")
```

Рис. 8. Код обчислення часу гальмування автомобіля

**Приклад 6.**

Мінімальна кількість розрядів (n) для адресації k байт пам'яті визначається нерівністю $2^n>k$.

Програму визначення кількості розрядів зображено на рис. 9.

```
k = int(input("Увести значення k = "))
n = 0              # n - це кількість розрядів
p = 1              # p - поточне значення
                   кількості байтів
while k > p:
    n = n + 1      # можна записати як n += 1
    p = p * 2      # можна записати як p *= 2
print("Мін. кількість розрядів =", n)
```

Рис. 9. Код визначення кількості розрядів

### • Цикли з післяумовою

Графічну схему циклу з післяумовою зображено на рис. 10.

У таких алгоритмах спочатку виконуються оператори тіла циклу, а потім перевіряється умова. Якщо умова має значення True (Так), виконання операторів тіла циклу продовжується.

Як тільки умова набуде значення False (Ні), виконання операторів тіла циклу припиняється й управління передається першому оператору, розташованому за оператором циклу. У мові Python відсутній оператор, який безпосередньо реалізує такий варіант циклу.


Рис. 10. Графічна схема циклу з післяумовою

У мові Python цикли з післяумовою можна реалізувати такою конструкцією оператора while:

```
while True :
        <блок інструкцій тіла циклу>
if <умова> : break
```

**Приклад 7.**

Батискаф заглиблюється в океан. За першу хвилину батискаф заглиблюється на 10 м, а за кожну наступну хвилину він заглиблюється на 10 % менше, ніж за попередню хвилину. Через скільки хвилин батискаф досягне глибини 100 м?

Програмний код моделювання процесу заглиблення батискафа зображено на рис. 11.

Тут k — кількість хвилин заглиблення, h — заглиблення за поточну хвилину, H — поточне значення глибини заглиблення. Оператор break здійснює переривання виконання циклу.

```
h = 10                  # заглиблення за першу хвилину
H = 0                   # початкова глибина заглиблення
k = 0                   # кількість хвилин заглиблення
while True:             # цикл з післяумовою
    H += h              # поточна глибина заглиблення
    k += 1              # поточна кількість хвилин
    h -= 0.01 * h       # заглиблення за поточну хвилину
    if H >= 100: break
print("Через:", k, " хвилин")
```

Рис. 11. Код моделювання процесу заглиблення батискафа

**Приклад 8.**

У банк покладено 10 000 грн під 15 % річних. Кожного року з рахунка знімається 800 грн. Через скільки років сума вкладу перевищить 14 000 грн? Програмний код розв'язання зображено на рис. 12. На екран буде виведено: Через 5 років.

Тут s — поточна сума вкладу, k — кількість років вкладу.

```
s = 10000                   # початкова сума вкладу
k = 0                       # початкова кількість років
while True:                 # цикл з післяумовою
    s = (s + 0.15 * s) - 800 # поточна сума вкладу
    k = k + 1               # поточна кількість років
    if s > 14000: break     # вихід із циклу
print("Через:", k, " років ")
```

Рис. 12. Код обчислення кількості років вкладу

## 3.3.3. Оператори continue і break

Оператори **continue** і **break** застосовуються всередині операторів тіла циклу та призначені для їх переривання.

- Оператор continue перериває цикл і повертає управління на початок циклу. Це дозволяє перейти до наступної ітерації циклу до завершення виконання всіх інструкцій усередині циклу.

На рис. 13 зображено програму виведення усіх парних чисел від 6 до 30, окрім чисел у діапазоні від 13 до 25 включно.

```
for i in range(6, 31, 2):
    if 12 < i < 26:    # якщо числа в діапазоні 12<i<26
        continue       # перехід на нову інтерацію циклу
    print(i)           # виведення чисел 6,8,10,12,26,28,30
```

Рис. 13. Код із використанням оператора continue

- Оператор break здійснює переривання та вихід із циклу навіть у тому випадку, коли всі ітерації ще не виконані. Цей оператор уже використовувався в розглянутих раніше прикладах.

На рис. 14 зображено ще один приклад програми з використанням оператора **break**. У ньому використовується нескінченний цикл уведення чисел та обчислення їх суми. Але цикл переривається, якщо буде уведено слово «кінець».

```
s = 0                                  # початкова сума
while True:                            # нескінченний цикл
    a = input("увести число ")
    if a == "кінець" :                 # якщо введено слово кінець
        break                          # переривання нескінченного циклу
    a = int(a)                         # перетворення рядка в число
    s += a                             # обчислення суми чисел
print("сума чисел = ", s)
input()
```

Рис. 14. Код із використанням оператора continue

## 3.3.4. Вкладені цикли

Циклічні алгоритми, розглянуті раніше, не містять у собі інших циклів, тому їх називають **простими циклами**.

Вкладеними називають цикли, що містяться в іншому циклі.

Цикл, що входить до складу іншого циклу, називають **внутрішнім**, а цикл, який містить інший цикл, — **зовнішнім**.

На рис. 15 наведено приклад блок-схеми алгоритму зі вкладеним циклом, у якому обчислюються площі 30 прямокутників, сторони яких набувають таких значень:

a = 3, 4, 5, 6, 7;
b = 2, 4, 6, 8, 10, 12.

Код програми, що реалізує цей алгоритм, зображено на рис. 16.

```
for a in range(3, 8, 1):        # зовнішній цикл
    for b in range(2, 13, 2): # внутрішній цикл
        s = a * b                 # обчислення площі
print("s = ", s)
input()
```

Рис. 16. Код обчислення площ прямокутників

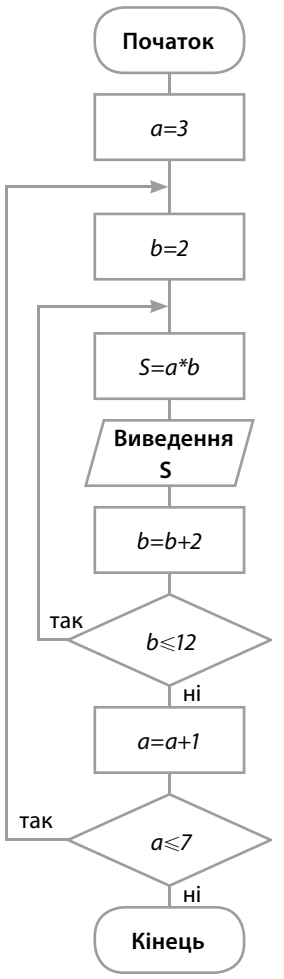


Рис. 15. Графічна схема алгоритму зі вкладеним циклом

## Запитання для перевірки знань

1. Які існують види циклічних алгоритмів?
2. Поясніть на прикладі сутність параметра і тіла циклу.
3. Для чого призначено оператор break, continue?
4. Накресліть графічну схему циклу з параметром.
5. Яку структуру має оператор циклу for?
6. Яку структуру має оператор циклу з передумовою?
7. Наведіть приклад вкладеного циклу.
8. Як у мові Python можна реалізувати цикл із післяумовою?

## Завдання для самостійного виконання

1. Розробіть код обчислення суми для чисел: 2, 7, 21, 9, 33, 13.
2. Розробіть код, обчислення суми непарних чисел, що більші 7, але менші 25.
3. Розробіть код обчислення суми чисел натурального ряду, максимальне значення якого не перевищує 7.
4. Перший член геометричної прогресії 6, а її знаменник — 0.5. Розробіть код обчислення значень членів прогресії, більших 0.6, і визначення номера останнього члена прогресії, що підсумовується.
5. Дано куб, сторони якого набувають п'ять значень: 3, 4.5, 6, 7.5, 9. Розробіть код визначення об'єму для кожного з них.
6. Радіус першої кулі дорівнює 2 см, а радіус кожної наступної збільшується на 0.5 см. Розробіть код для визначення бокових поверхонь перших шести куль.
7. Відомо результати плавання вільним стилем на дистанції 50 м п'яти учнів кожного з трьох 10 класів школи. Розробіть код, за допомогою якого визначається середній час запливу учнями кожного класу.
8. У банк покладено S грн під N відсотків річних. Розробіть код, за допомогою якого визначається кількість років, через які сума вкладу буде не менше N грн.

# 4. Вбудовані типи даних та їх опрацювання

У будь-якій мові програмування дані поділяють на дві основні групи: прості (скалярні) та структуровані.

У мові програмування Python також існує поділ на змінювані (mutable) та незмінювані (inmutable) типи даних. Структуровані дані поділяють на лінійні, табличні й ієрархічні.

У мові програмування Python лінійні й табличні структури вбудовані в саму мову, що суттєво полегшує та спрощує користувачу роботу з ними.

У лінійній структурі адреса елемента даних однозначно визначається його номером (індексом) у цій структурі. Лінійними структурами є рядкові типи даних і всі типи послідовностей.

У табличних структурах адреса елемента даних визначається кількома індексами. Наприклад, у двовимірних масивах його адреса визначається номером рядка й номером стовпця.

В ієрархічних структурах адреса кожного елемента визначається маршрутом доступу від вершини структури до цього елемента.

Далі розглянемо ті структури даних, які передбачені навчальною програмою для 10 класу, решта вивчається в 11 класі.

## 4.1. Списки, стеки, черги

*Ви неодноразово використовували в повсякденному житті термін «список». Спробуйте дати йому означення. Чи доводилося використовувати його в мові програмування, яку ви вивчали раніше?*

### 4.1.1. Структура списків і операції над ними

**Список** у мові Python — це впорядкована колекція* об'єктів будь-якого типу у квадратних дужках, які відокремлюються один від одного комою.

**Приклад 1.**
[5, "файл", "w", 21, [1, 2, 3]].

У прикладі 1 об'єктами списку є числа, рядки та список. У списках можна змінювати значення його елементів, збільшувати та зменшувати кількість елементів, здійснювати пошук потрібних елементів і впорядковувати його елементи. Отже, список є об'єктом, що змінюється, тобто належить до змінюваних типів даних. Самі списки можуть бути вкладені в об'єкти інших типів.

Списки деякою мірою нагадують динамічні масиви вказівників, але в масивах значення елементів можуть бути лише одного типу. Мова Python має значну кількість операцій, функцій і методів опрацювання списків.

Список є одним з основних типів даних. Він використовується в мові Python частіше за інші.

На основі списків у програмі можна створювати та опрацьовувати структури даних довільної складності*.

Далі розглянемо лише основні операції, якими є, наприклад, звернення до елемента за його індексом, отримання зрізу, конкатенація, повторення, перевірка на входження й ін.

* Змінювана впорядкована колекція (тобто індексована послідовність.

Списки можуть бути *одновимірними* і *багатовимірними*.

Розглянемо спочатку одновимірні списки. Позиція елемента в списку задається індексом, який починається з нуля.

Списки можна створювати простим переліченням елементів списку в квадратних дужках (приклад 2).

Розглянемо операції, які можна виконувати над списками.

**Приклад 2.**

```
>>> al_1 = ["файл", 21, 13, "5"]
>>> al_1
['файл', 21, 13, '5']
```

| | | |
|---|---|---|
| Щоб звернутися до елемента списку, необхідно у квадратних дужках зазначити індекс елемента | >>> a_1 = [7, 20, "миша"]<br>>>> a_1[2]<br>**'миша'** | # виведення другого елемента |
| Значення елементів списку можна змінювати шляхом присвоювання їм нових значень | >>> b_1 = [21, 11, 17, 24]<br>>>> b_1 [2] = 15<br>>>> b_1<br>**[21, 11, 15, 24]** | # зміна значення другого елемента<br># виведення списку |
| Елементи списку можна присвоїти кільком змінним | >>> x1, x2, x3 = [3, 7, 5]<br><br>>>> x1, x2, x3<br>**(3, 7, 5)** | # елементи списку присвоюються<br># 3 змінним<br># виведення значень змінних |
| Якщо елементів списку більше кількості змінних, зайві елементи можна присвоїти одній змінній, поміченій зірочкою | >>> x1, *x2, x3 = [1, 3, 7, 5]<br><br>>>> x1, x2, x3<br>**(1, [3, 7], 5)** | # змінній x2 присвоюються два<br># значення: 3 і 7<br># виведення значень змінних |
| Операція зрізу (виділення елементів) має формат: <ім'я списку> [початок: кінець: крок]. Усі параметри є необов'язковими | >>> a = [1, 2, 3, 4, 5, 6]<br>>>> a[1:4]<br><br>**[2, 3, 4]** | # виділення елементів з 1 до 3<br># включно |
| Списки можна об'єднувати | >>> a1 = [1, 2, 3, 4]<br>>>> a2 = [5, 6, 7]<br>>>> a1 + a2<br>**[1, 2, 3, 4, 5, 6, 7]** | # об'єднання списків |

## 4.1.2. Функції і методи опрацювання списків

Розглянемо **основні функції опрацювання списків** на прикладах.

| | | |
|---|---|---|
| Перетворити рядок у список можна за допомогою функції list() | >>> list("процесор")<br>**['п', 'р', 'о', 'ц', 'е', 'с', 'о', 'р']** | # перетворення рядка в список |
| Отримати кількість елементів у списку можна за допомогою функції len() | >>> a = [1, 4, 6,8]<br>>>> len(a)<br>**4** | # довжина списку |

| | | |
|---|---|---|
| Отримати список із випадковими числами з іншого списку можна за допомогою функції sample (послідовність, кількість елементів) з модуля random | >>> import random<br>>>> random.sample(range(51), 6)<br>**[48, 11, 46, 3, 21, 0]** | # імпортування модуля random<br># шість випадкових чисел<br># у діапазоні 0–51 |
| Перетворити список у рядок можна різними засобами. Найпростіший із них — використання функції str | >>> a1 = ["file1", 25, "file2", 21]<br>>>> str(a1)<br>**"['file1', 25, 'file2', 21]"** | # список<br># перетворення списку в рядок |
| Максимальне та мінімальне значення в списку можна знайти за допомогою, відповідно; функцій max() і min() | >>> a1 = [44, 20, 3, 17, 25, 16]<br>>>> max(a1), min(a1)<br><br>**(44, 3)** | <br># виведення максимального<br># та мінімального елементів |
| Для отримання випадкового елемента зі списку слід імпортувати модуль random і скористатися функцією choice (послідовність) або функцією sample (послідовність, кількість елементів) | >>> import random<br>>>> random.choice("a", "ab", "c", "ca", "f"])<br>**'ab'** | # імпортування модуля random<br># виведення випадкового елемента |

Розглянемо **основні методи опрацювання списків** на прикладах.

| | | |
|---|---|---|
| Метод **append(об'єкт)** — додає один об'єкт у кінець списку | >>> a1 = [1, "as", 2]<br>>>> a1.append("bmv")<br><br>>>> a1<br>**[1, 'as', 2, 'bmv']** | # список<br># додавання елемента bmv<br># у кінець списку<br># виведення списку |
| Метод **extend("послідовність")** — додає елементи послідовності в кінець списку | >>> a1 = [1, 2, "web"]<br>>>> a1.extend("file, 7")<br>>>> a1<br>**[1, 2, 'web', 'f', 'i', 'l', 'e', ',', ' ', '7']** | # список<br># додавання елементів у список<br># виведення списку |
| Метод **insert(індекс, об'єкт)** — вставляє один об'єкт у вказану позицію списку | >>> a1 = [3, "vin", 5]<br>>>> a1.insert(1, "file")<br><br>>>> a1<br>**[3, 'file', 'vin', 5]** | # список<br># додавання в 1 позицію списку<br># елемента<br># виведення списку |
| Метод **pop(індекс)** — видаляє елемент зі списку за вказаним індексом. Видалити елемент зі списку можна також за допомогою оператора **del список**[індекс] | >>> b1 = [7, "com", 5]<br>>>> b1.pop(2)<br>**5**<br>>>> b1<br><br>**[7, 'com']**<br>>>> del b1[1]<br>>>><br>**[7]** | # список<br># виведення видаленого елемента<br><br># виведення списку після видалення<br># елемента<br><br># видалення елемента зі списку за індексом<br># виведення списку після видалення елемента |

| | |
|---|---|
| Метод **remove(значення)** — видаляє зі списку перший елемент, який містить вказане значення | >>> a1 = [3, 5, "lad", 7, "lad"]<br>>>> a1.remove ("lad"); a1 # видалення елемента "lad" із другої позиції<br>**[3, 5, 7, 'lad']** |
| Метод **clear()** — видаляє зі списку всі елементи | >>> b1 = [5, 11, 8, 7, 20]<br>>>> b1.clear(); b1<br>**[]** |
| Метод **index(значення [, початок [,кінець]])** — повертає індекс елемента, який має вказане значення. За замовчуванням параметрів **початок** і **кінець** пошук елемента буде виконуватися від початку до кінця списку | >>> b1 = [10, 7, 55, 16, 8, 20]<br>>>> b1.index(16) # виведення індексу елемента зі значенням 16<br>**3** |
| Метод **count(значення)** — використовується для визначення кількості елементів зі вказаним значенням. Якщо елемент відсутній у списку, повертається значення 0 | >>> c1 = [7, 13, 14, 7, 7, 5]<br>>>> c1.count(7) # виведення кількості елементів зі значенням 7<br>**3** |
| Сортування елементів списку реалізується за допомогою методу **sort()**, який має таку загальну структуру: **sort([key = None][,reverse = False])**. Як бачимо, параметри є необов'язковими. За замовчуванням сортування виконується за зростанням значень елементів з урахуванням регістра. Для сортування за зменшенням слід указати другий параметр таким: **reverse=True:** Зазначимо, що **метод sort()** перетворює старий список у новий | >>> a1 = [17, 100, 20, 3, 8, 16, 25] # список<br>>>> a1.sort(reverse=True) # сортування в порядку зменшення значень<br>>>> a1 # виведення впорядкованого списку<br>**[100, 25, 20, 17, 16, 8, 3]** |
| Метод **sorted(послідовність [,key=None][,reverse = False])** — призначений для сортування списку та збереження старого | >>> b1 = [39, 16, 21, 7, 23, 15] # початковий список<br>>>> sorted(b1) # повернення впорядкованого списку<br>**[7, 15, 16, 21, 23, 39]**<br>>>> b1<br>**[39, 16, 21, 7, 23, 15]** # повернення початкового списку |

# 4.1.3. Стек і черга

**Стек** — це виділена область оперативної пам'яті.

Робота зі стеком у програмуванні деякою мірою нагадує роботу зі стосом книжок: на першу книжку кладеться друга, на другу — третя, на третю — четверта і так далі. Щоб узяти першу книжку, покладену в стос, необхідно зняти спочатку четверту, потім третю, далі другу і, нарешті, першу.

Стек працює в порядку LIFO (*Last In, First Out*), тобто останній доданий у стек фрагмент пам'яті буде першим у черзі на вихід зі стеку. Кожного разу, коли функція оголошує нову змінну, вона додається в стек. А коли ця змінна стає неактуальною (наприклад, коли функція припиняє роботу), вона автоматично видаляється зі стеку і область пам'яті стає доступною для інших стекових змінних.

Стек часто використовується для організації виклику підпрограм і повернення в основну програму. Для точки основної програми, з якої здійснюється звернення до підпрограми, у стеку запам'ятовується адреса основної програми, до якої слід повернутися після завершення підпрограми.

Під час кожного звернення до підпрограми в стек додаються нові адреси повернення. Після кожного завершення підпрограми зі стеку знімається адреса повернення в основну програму. Ураховуючи те, що звернення до підпрограм виконується досить часто, стек здебільшого реалізується на апаратному рівні, а не програмному.

Програмно стек реалізується на основі списку або масиву. Якщо використовується масив, то потрібно визначати його розмір, щоб за потреби мати достатньо комірок. Неправильне визначення розміру масиву може призвести до помилок у роботі програми або до неефективного використання пам'яті.

У мові Python стек програмно реалізується здебільшого на основі списку.

У списку для кожного елемента відводиться блок пам'яті, обсяг якого повинен бути достатній для збереження значення елемента та посилання на попередній і наступний елементи стеку.

Для роботи зі стеком застосовуються такі визначені для списків методи:

- **append()** — для додавання нового елемента в стек;
- **pop()** — для вибірки елемента із вершини стеку (вершиною стеку називають останній уведений елемент).

Реалізація стеку на основі **масиву** простіша й потребує меншого обсягу пам'яті, але необхідно заздалегідь знати розмір масиву.
Реалізація стеку на основі **списку** надійніша, але потребує більшого обсягу пам'яті.

**Приклад 3.**

```
>>> stack = [2, 5, 7]      # створення списку
>>> stack.append(12)       # додавання у вершину стеку числа 12
>>> stack                  # виклик стеку
[2, 5, 7, 12]              # значення елементів стеку
>>> stack.pop()            # вибір елемента із вершини стеку
12                         # видалений елемент
>>> stack                  # виклик стеку
[2, 5, 7]                  # значення елементів стеку
>>> stack.pop()            # вибір елемента із вершини стеку
7                          # видалений елемент
```

**Черга** в програмуванні — це структура даних, що працює за принципом «перший прийшов — перший пішов».

Чергу можна порівняти, наприклад, із чергою в залізничну касу: перший клієнт біля каси обслуговується першим. Елемент, що додається до черги, опиняється в її кінці, а елемент, який видаляється із черги, перебуває на її початку.

У мові Python для роботи з чергою також використовуються списки (приклад 4).

**Приклад 4.**

```
>>> zherg = ["Олег", "Олена", "Ліда"]
>>> zherg.append("Сашко")
>>> zherg.append("Ліза")
>>> zherg
['Олег', 'Олена', 'Ліда', 'Сашко', 'Ліза']
>>> zherg.pop(0)
'Олег'
>>> zherg
['Олена', 'Ліда', 'Сашко', 'Ліза']
```

## 4.1.4. Багатовимірні списки

У багатовимірних (або вкладених) списках кожна група елементів списку береться у квадратні дужки. Двовимірний список можна створити, наприклад, так:

```
>>> a1 = [[1, 2, 3], [4, "файл", "s"], [5, 6, 7]]
```

Але для наочності краще записувати так:

```
>>> a1 =  [
          [1, 2, 3],
          [4, "файл", "s"],
          [5, 6, 7]]
>>> a1
[[1, 2, 3], [4, 'файл', 's'], [5, 6, 7]]
```

Для звернення до елемента багатовимірного списку слід у квадратних дужках указати всі його індекси. Наприклад, звернутися до першого елемента першої групи двовимірного списку можна так:

```
>>> a1[1][1]
'файл'
```

Звертатися до всіх або частини елементів списку можна за допомогою операторів циклу for і while, а також функції range(), яка розглядалася раніше. Наприклад, за допомогою оператора циклу for до них можна звернутися так:

```
>>> b1 = ["файл", 5, 25, "ц"]
>>> for i in b1: print (i, end=" ")    # натиснути Enter двічі
файл, 5, 25, ц
```

У генераторах списку разом з операторами циклу можуть використовуватися умовні й інші оператори, що дозволяє виконувати складні перетворення списків. Якщо вираз розміщується всередині не квадратних, а круглих дужок, то буде повертатися не список, а значення певного виразу. Такі конструкції у мові Python називають виразами-генераторами.

Виконаємо, наприклад, підсумовування парних чисел:

```
>>> a1 = [6, 5, 2, 1, 8]
>>> sum ((i for i in a1 if i % 2 == 0))
16   # сума нульового, другого і четвертого елементів
```

За допомогою операторів циклу можна не тільки отримати значення його елементів, а й змінити значення кожного з них або окремих елементів. Такі дії можна виконувати різними способами, у тому числі за допомогою так званих **генераторів списку**. Для їх реалізації оператор циклу розміщується всередині списку:

```
>>> a1 = [2, 3, 8, 5]
>>> a1 = [i * 3 for i in a1]
>>> print(a1)
```

У цьому фрагменті коду кожнен елемент списку множиться на 3. У результаті отримаємо значення списку:
**[6, 9, 24, 15]**

## 4.1.5. Приклади програм опрацювання списків

**Приклад 5.**

У списку [27, 3, 12, 22, 37, 8] знайти максимальний елемент, вилучити елемент на другій позиції, упорядкувати новий список у порядку збільшення значень його елементів зі збереженням попереднього списку, вставити в нього на четверту позицію число 5, потім замінити значення першого елемента на число 10.

Програму, що реалізує це завдання, зображено на рис. 1.

```
a = [27, 3, 12, 22, 37, 8]
print(max(a))     # максимальний елемент списку
a.pop(2)          # вилучити елемент на 2 позиції
print(sorted(a)) # сортувати список зі збереженням старого
print(a)          # виведення старого списку
a.insert(4, 5)    # вставити число 5 на 4 позицію
print(a)          # виведення списку зі змінами
a[1] = 10         # замінити перший елемент на число 10
print(a)          # виведення списку зі змінами
```

Рис. 1. Програма опрацювання списку

Далі наведено результат виконання програми:

```
37
[3, 8, 22, 27, 37]
[27, 3, 22, 37, 8]
[27, 3, 22, 37, 5, 8]
[27, 10, 22, 37, 5, 8]
```

**Приклад 6.**

Дано два списки: ["Python розробив", "1991"] і ["Гвидо ван"]. Розробити програму, за допомогою якої отримується рядок: Мову Python розробив у 1991 році Гвидо Ван Россум. Один із варіантів програми, що реалізує це завдання, зображено на рис. 2.

```
a = ["Python розробив", "1991"]   # перший список
b = ["Гвидо ван"]                 # другий список
a.insert(0, "Мову")               # вставлення слова Мову
a.insert(2, "у")                  # вставлення букви у
a.append("році")                  # додавання слова році
print(a)                          # виведення 1 рядка
b.append("Россум")                # зміна другого списку
c = a + b                         # об'єднання списків
print(c)                          # виведення нового списку
print(" ".join(c))                # перетворення списку в рядок
```

Рис. 2. Програма об'єднання списків і перетворення списку в рядок

Далі наведено результат виконання програми:

```
['Мову', 'Python розробив', 'у', '1991', 'році']
['Мову', 'Python розробив', 'у', '1991', 'році', 'Гвидо ван', 'Россум']
Мову Python розробив у 1991 році Гвидо ван Россум
```

**Приклад 7.**

У послідовності [4, 6, 13, 9, 5, 16, 11] знайти числа, більші 6, збільшити їх удвічі, вивести на екран і обчислити їх суму.

Варіант програми, що реалізує це завдання, зображено на рис. 3.

```
a = [4, 6, 13, 9, 5, 16, 11]   # створення списку
s = 0                          # початкове значення суми
for i in range(len(a)):        # перебрати елементи списку
    if a[i] > 6:               # якщо число більше 6
        a[i] = 2 * a[i]        # збільшення значення елемента
        print(a[i])            # виведення чисел більше 6
        s = s + a[i]           # обчислення суми
print("сума = ", s)
input()
```

Рис. 3. Програма збільшення та обчислення суми чисел, більших 6

Далі наведено результат виконання програми:

```
26
18
32
22
сума=  98
```

## Запитання для перевірки знань

1. Що називають списком?
2. Як створюється список?
3. Як можна замінити значення елемента списку?
4. Як здійснюється об'єднання списків?
5. Для чого призначена функція list()?
6. Як перетворюється список у рядок?
7. Як можна отримати випадкові елементи з іншого списку?
8. Для чого призначений метод append()?
9. Поясніть сутність методу insert().
10. Для чого призначений метод remove()?
11. Поясніть, як виконується сортування елементів списку.

## Завдання для самостійного виконання

1. Дано список [13, 19, 11, 7, 18]. Вилучте елемент на першій позиції і після цього знайдіть максимальний елемент.
2. Дано список міст України ["Херсон", "Житомир", "Ужгород", "Харків"]. Уставте місто Луцьк на другу позицію і після цього відсортуйте список.
3. Дано список [9, 2, 5, 6]. Замініть число на першій позиції числом 12. Відсортуйте новий список зі збереженням старого.
4. Дано список ["В. Глушков", "вчений України"]. Внесіть у нього зміни, щоб отримати такий список ["В. Глушков — великий", "вчений України"]. Перетворіть список на рядок.
5. Дано два списки: ["Мова", "Pascal"] і ["— це мова", "процедурного програмування"]. Об'єднай-те списки в один і перетворіть його на рядок.
6. Дано список [5, 7, 8, 12, 4]. Обчисліть суму елементів, значення яких більші 5.

# 4.2. Кортежі, діапазони, множини

*Ви вже знайомі з деякими структурами даних, зокрема із списками. Які, на вашу думку, ще структури даних доцільно підтримувати сучасним мовам програмування?*

**Кортеж** у математиці — упорядкована та скінченна сукупність елементів (нескінченний кортеж називається сімейством). Кількість елементів у кортежі визначає його довжину. Наприклад, кортеж із двох елементів (тобто довжини 2) називають двійкою, із трьох елементів — трійкою і т. д. Кортеж із n елементів називають n-кою.

**Кортеж** — це незмінна впорядкована колекція об'єктів будь-якого типу в круглих дужках (або без них), які відокремлюються один від одного комою.

Кортежі схожі на списки, але відрізняються від них тим, що кортежі є незмінними послідовностями і замість квадратних дужок застосовуються звичайні дужки.

У мові Python **кортеж** (англ. *tuple*) відрізняється від списку тим, що елементи кортежу після його створення не можна змінювати жодним чином.

Кортежі підтримують функції len(), min(), max(), методи index() і count(), сутність яких розглядалася в процесі опису списків.

Діапазони підтримують доступ до елемента за його індексом, отримання зрізу, перевірку на входження і невходження, функції **len()**, **max()**, **min()**, методи **index()** і **count()**.

**Діапазон** — це незмінна послідовність цілих чисел із початковим, кінцевим значеннями і кроком їх зміни.

Для створення діапазону призначена функція range() такої структури:

range([початок,] кінець[, крок ])

Як бачимо, обов'язковим є лише параметр кінець. За замовчуванням значення параметра початок дорівнює нулю, а крок — одиниці.

**Множина** — це невпорядкована колекція унікальних (тих, що не повторюються) об'єктів будь якого типу.

**Множини** в мові **Python** створюються абсолютно випадковим чином, згодом вони все одно розташуються у випадковому порядку.

Існує два типи множин: змінна (set) і незмінна (frozenset()). Множина змінного типу створюється за допомогою вбудованої функції set, генераторів множин, літералів множин та інших.

Ця функція дозволяє також перетворювати послідовність у множину:

```
>>> set("монітор")                  # перетворення рядка
                                      в множину
{'і', 'т', 'м', 'н', 'о', 'р'}
>>>set([5, 8, 4, 8, 2, 4, 7])      # перетворення списку
                                      в множину
{8, 2, 4, 5, 7}
>>> set()                           # пуста множина може бути
set()                                 задана тільки так
```

Для виконання операцій над множинами в мові Python існують різні оператори, функції і методи.

| | | |
|---|---|---|
| Метод **union()** — призначений для об'єднання множин в одну | >>> a = set([3, 5, 7])<br>>>> a.union(set([7, 9, 11]))<br>**{3, 5, 7, 9, 11}** | # множина a<br># об'єднання 2 множин<br># результат об'єднання записується у множину a |
| Метод **add()** — додає елемент до множини | >>> b = set([1, 2, 3])<br>>>> b.add(4)<br>>>> b<br>**{1, 2, 3, 4}** | # множина b<br># додавання у множину b значення 4<br># виведення множини |
| Функція **len()** — визначає кількість елементів у множині | >>> len(set([4, 5, 7, 9, 10]))<br><br>**5** | # визначення кількості елементів<br># у множині |
| Метод **a.update(b)** — додає елементи множини **b** до множини **a** | >>> a = set([1, 2])<br>>>> a.update(set([5, 6, 7]))<br>>>> a<br>**{1, 2, 5, 6, 7}** | # множина a<br># додавання множини до множини a<br># виведення нової множини |
| Оператор **in** — перевіряє наявність елемента в множині. Повертає значення **True**, якщо множина містить елемент, інакше — значення **False** | >>> a = set([1, 2, 3, 4, 5])<br>>>> 3 in a, 6 in a<br><br>**(True, False)** | # множина a<br># перевірка наявності в множині<br># чисел 3 і 6 |
| Оператор перевірки на рівність **(==)** — повертає значення **True**, якщо множини рівні, інакше — значення **False** | >>> set([1, 2, 3]) ==<br>set([3, 4, 5])<br>**False** | # перевірка на рівність двох множин |
| Метод **discard(елемент)** — видаляє з множини елемент, якщо його значення міститься в множині | >>> a = set([1, 2, 3, 4, 5])<br>>>> a.discard(2)<br>>>> a<br>**{1, 3, 4, 5}** | # множина a<br># видалення елемента зі значенням 2<br># виведення нової множини |
| Метод **remove()** — видаляє елемент зі множини та повертає KeyError, якщо такого елемента не існує | >>>> a = set([0, 1, 4, 81]<br>>>>> remove(6)<br>Keyerror | # множина a<br># видалення елемента зі значенням 6<br># такого елемента не існує |

Мова Python підтримує також генератор множин, синтаксис якого схожий на синтаксис генератора списку, але вираз береться не у квадратні дужки, а у фігурні:

```
>>> {a * 2 for a in [2, 3, 4, 3, 4]}
{4, 6, 8}
```

Із використанням кортежів (приклад 1-4), діапазонів (приклади 5-7), множин (приклад 8) можна ознайомитися на сайті interactive.ranok.com.ua

## Запитання для перевірки знань

1. Що називають діапазоном?
2. Що називають множиною?
3. Як створити кортеж?
4. Як створити діапазон?
5. Як перетворити діапазон на кортеж?
6. Як перетворити список на множину?

# 4.3. Словники. Функції, операції і методи опрацювання словників

*Словником у класичному розумінні ви, звичайно, користувалися. Згадайте, як за терміном (словом) знаходили відповідний пояснювальний текст або його переклад.*

**Словник** у мові Python реалізований у вигляді хеш-таблиці — невпорядкована колекція об'єктів будь-якого типу, доступ до яких здійснюється не за допомогою індексу, а за допомогою ключа.

Для зручності можна уявити словник як невпорядковану множину пар виду ключ і значення. Пари розділяються комами, весь словник перебуває у фігурних {} дужках.

Словники можна створювати різними способами.

- Використати **функцію dict()**. Ця функція має чотири формати. Нижче наведено два основні формати:

1) dict (ключ1=значення1, …, ключN=значенняN):

```
>>> a1 = dict(a2=5, a3=7, a4=8)          # створення словника
>>> a1                                   # виведення словника
{'a4': 8, 'a3': 7, 'a2': 5}
```

2) dict (словник) :

```
>>> b1 = dict({"b2": 5, "b3": 7, "b4": 8})   # створення словника
>>> b1                                       # виведення словника
{'b3': 7, 'b2': 5, 'b4': 8}
```

- **Указати всі елементи словника всередині фігурних дужок:**

```
>>> a1 = {"a2": 5, "a3": 7, "a4": 8}     # створення словника
>>> a1                                   # виведення словника
{'a4': 8, 'a3': 7, 'a2': 5}
```

Цей спосіб застосовується найчастіше. Щоб створити порожній словник, можна використати функцію dict():

```
>>> a0 = dict()                          # створення пустого словника
>>> a0                                   # виведення словника
{}
```

або

```
>>> a0 = {}                              # створення пустого словника
>>> a0                                   # виведення словника
{}
```

- Об'єднати два списки в список кортежів можна за допомогою **функції zip()**. Створимо два списки, об'єднаємо їх у список кортежу, а потім створимо словник:

```
>>> a1 = ["m", "n"]                      # список із ключами
>>> a2 = [3, 6]                          # список зі значеннями
>>> list(zip(a1, a2))                    # створення списку кортежу
[('m', 3), ('n', 6)]
>>> a3 = dict(zip(a1, a2))               # створення словника
```

**Ключем** може бути будь-який незмінний тип даних, наприклад число, рядок або кортеж, тобто будь-який об'єкт із незмінюваним типом. Значення елементів словника можна змінювати. Словники не є послідовностями, тому такі загальні операції, як отримання зрізу, конкатенації та інші вони не підтримують.

**Хеш-таблиця** — структура даних, яка реалізує інтерфейс асоціативного масиву. Вона дозволяє зберігати пари (ключ, значення) і здійснювати три операції: операцію додавання нової пари, операцію пошуку та операцію видалення за ключем.

Під час хешування може статися, що різні значення ключів перетворюються в однаковий двійковий код. Такі випадки називають колізією. Існують різні алгоритми боротьби з колізією. Один із найнадійніших — так званий алгоритм відкритої адресації.

```
>>> a3                                  # виведення словника
{'n': 6, 'm': 3}
```

Ознайомимося з аспектами створення словника докладніше.

| | | |
|---|---|---|
| Щоб звернутися до елемента словника, потрібно вказати його ключ (число, рядок або кортеж) у квадратних дужках. Якщо елемент зі вказаним ключем у словнику відсутній, генерується виняток **KeyError** | >>> a = {2: "файл", "n": "пам'ять", (1, 3): "процесор"}<br>>>> a[2], a["n"], a[(1, 3)]<br>**('файл', 'пам'ять', 'процесор')** | # словник<br><br># звернення до елементів словника |
| Перевірити наявність ключа у словнику можна за допомогою оператора **in**. Якщо ключ у словнику є, генерується значення **True**, інакше — **False** | >>> a = {1: "as", "kl": 5}<br>>>> "kl" in a<br>**True** | # словник<br># перевірка наявності у словнику ключа kl |
| Метод **get(ключ)** повертає значення, яке відповідає ключу. Якщо ключ відсутній, повертається значення **None** | >>> a = {"p": 21, 37: "байт", "sk": "біт"}<br>>>> a.get("p"), a.get("sk"), a.get(38)<br>**(21, "біт", None)** | # словник<br># повернення значень словника<br># елемента з ключем 38 немає |
| Змінити елемент словника або додати новий елемент можна за допомогою його ключа | >>> a = {"b": 25, "c": 100}<br>>>> a["b"] = 300<br><br>>>> a[2] = "astra"<br>>>> a<br>**{2:'astra', 'c': 100, 'b': 300}** | # словник a<br># зміна значення, що має ключ "b"<br># додавання нового елемента<br># виведення оновленого словника |
| За допомогою функції **len()** у словнику можна визначити кількість ключів | >>> a = {1: "один", 2: "два", 3: "три"}<br>>>> len (a)<br>**3** | # словник a<br># визначення кількості ключів у словнику |
| За допомогою оператора **del** можна видалити елемент зі словника, вказавши його ключ у квадратних дужках | >>> a = {"sd": "система", "sf": "принтер", "sh": 25}<br>>>> del a["sf"]<br>>>> a<br>**{'sd': 'система', 'sh': 25}** | # словник a<br><br># видалення елемента з ключем sf<br># виведення нового словника |
| Метод **discard** (елемент) видаляє з множини елемент, якщо його значення міститься в множині | >>> a = set([1, 2, 3, 4, 5])<br>>>> a.discard(2)<br>>>> a<br>**{1, 3, 4, 5}** | |
| Метод keys() повертає об'єкт із ключами словника; його можна використати для виведення всіх елементів словника за допомогою оператора циклу for. | >>> a = {"x":1, "y":2, "z":3}<br>>>> for i in a.keys(): print("({0} : {1})". format(i, a[i]), end=" ")<br><br>**(y:2) (z:3) (x:1)** | |

Звернемо увагу на те, що елементи словника виведені в довільному порядку. Це зумовлене тим, що словники є невпорядкованими структурами. Якщо потрібно вивести елементи словника впорядкованими за значенням ключа, то слід отримати список ключів, а потім скористатися функцією **sorted()**.

| | | |
|---|---|---|
| Метод **update()** додає елементи в словник. Одна з найпростіших структур методу така: update (словник) | >>> x = {1:"a", 2:"b"}<br>>>> x.update({3:"c", 4:"d"})<br><br>>>> x<br>**{1: 'a', 2: 'b', 3: 'c', 4: 'd'}** | # словник x<br># додавання у словник нових елементів<br># виведення нового словника |
| Метод **values()** повертає об'єкт dict_values, який містить усі значення словника | >>> x = {"a": 1, "b": 2}<br>>>> x.values()<br>**dict_values([1, 2])**<br>>>> list(x.values())<br>**[1, 2]** | <br># об'єкт dict_values ()<br><br># отримано список значень |
| Метод **pop**(ключ **[**, значення за замовчуванням**])** вилучає зі словника елемент зі вказаним ключем і повертає його значення. Якщо ключ у словнику відсутній, то повертається значення другого параметра, а якщо відсутній ключ і не вказаний другий параметр, то генерується виняток **KeyError** | >>> x = {"a": 5, "b": 6, "c": 7}<br>>>> x.pop("b"), x.pop("d", 8)<br><br>**(6, 8)**<br>>>> x<br>**{'c': 7, 'a': 5}** | # словник x<br># видалення елементів з указаними ключами<br><br># виведення нового словника |

**Приклад 2.**

```
>>> x = ["a", "b", "c"]
>>> values = [5, 6, 7]
>>> {m: n for (m, n) in zip(x, values)}
{'c': 7, 'a': 5, 'b': 6}
```

Мова Python підтримує також генератор словників, синтаксис якого схожий на синтаксис генератора списку, але вираз заключається не у квадратні, а у фігурні дужки всередині виразу перед циклом **for** (приклад 2).

Із прикладами створення програм опрацювання словників можна ознайомитися на сайті interactive.ranok.com.ua

## Запитання для перевірки знань

1. Який тип даних називають словником?
2. Як оголошуються словники?
3. Як можна звернутися до елемента словника?
4. За допомогою якого оператора перевіряється наявність ключа в словнику?
5. Як можна видалити елемент зі словника?
6. Наведіть приклад використання генератора словника.

## Завдання для самостійного виконання

1. Ринкова ціна на посібники у гривнях така: з інформатики — 45, географії — 53, історії — 40, хімії — 47. Складіть програму визначення мінімальної і максимальної ціни посібників і ціну посібника з хімії.
2. Створіть словник, об'єктами якого є номери потягів і міста, до яких вони слідують: 23 — Львів, 7 — Одеса, 15 — Харків. Додайте до словника два потяги: 37 — Херсон, 29 — Полтава і вилучіть потяг 15 — Харків.

# 4.4. Масиви

Для розв'язування математичних та інших задач у кожній мові програмування досить часто використовують масиви, які в мові Python досить подібні до вбудованого в мову структурованого типу «списки», але з обмеженням на тип даних і розмір кожного елемента.

**Масив** — це структурований тип даних, усі елементи якого мають лише один тип, наприклад **int**, **char** та ін.

Структура масиву може бути *одновимірною* (лінійною), *двовимірною* (табличною) та *багатовимірною*. Доступ до елементів масиву отримується відповідно за одним або кількома індексами.

У мові Python масиви визначено в модулі array, який містить величезну кількість методів і функцій їх опрацювання. Таким чином користувачу не потрібно розробляти масиви, слід просто знати, як їх використовувати.

Далі розглянемо та опишемо класичні засоби опрацювання масивів.

У мові **Python** із масивами можна працювати й без модуля **array**. Власне модуль **array** використовується досить рідко, у випадках коли потрібно досягти високої швидкості роботи. В інших випадках масив типу **array** краще замінити іншими типами даних: списками, кортежами, рядками. Найчастіше для представлення масиву в **Python** використовуються списки.

## 4.4.1. Одновимірні масиви

*Чи можна назвати перелік навчальних предметів у 10 класі масивом? Наведіть приклади масивів, із якими вам доводилося стикатися в процесі вивчення географії. Які операції ви виконували над цими масивами?*

Загальну структуру одновимірного масиву можна позначити так: x[0], x[1], x[2], …, x[n-1], x[n]. У квадратних дужках зазначено **індекси** (номери позицій) елементів у масиві. Індексами елементів масивів можуть бути дані будь-якого типу, у тому числі вирази, але найчастіше ними є цілі числа.

Будь-якому елементу масиву можна надати нове значення за допомогою оператора присвоювання, наприклад, mas[4]=5 — четвертому елементу одновимірного масиву mas присвоєно значення 5 (це можливо лише у випадку, коли елемент із цим індексом уже існував, інакше — помилка).

Створити масив у мові Python можна різними способами, наприклад, можна вводити значення елементів із клавіатури, обчислювати значення за певною формулою та присвоювати їх елементам масиву й ін.

Розглянемо створення, введення і виведення елементів у масивах на прикладах.

**Одновимірний масив** — це нумерована послідовність однотипних елементів. Такий масив можна уявити, наприклад, як таблицю, що містить один рядок або стовпець.
Він є динамічним — ви можете змінювати його довжину, додавати або видаляти елементи й ін. Щоб зберігати групу однотипних об'єктів у **Python**, оптимально використовувати тип даних **list**(список).

**Приклад 1.**

Найпростіш створення масиву є перелічення у квадратних дужках значень його елементів праворуч від оператора присвоювання. Наприклад, у результаті виконання оператора a = [5, 9, 3, 12] елементи масиву з іменем a набудуть таких значень: a[0]=5, a[1]=9, a[2]=3, a[3]=12. Зверніть увагу, що нумерація елементів у масивах мови Python починається з нуля.

**Приклад 2.**

На рис. 1 наведено програмний код створення масиву, елементами якого є рядки. Елементи масиву за допомогою оператора циклу виводяться на екран.

```
# Найпростіший спосіб створення масиву
mas = ['File', 'Edit', 'Run','Windows', 'Help']
n = len(mas)              # довжина масиву
for i in range(n):        # цикл перегляду елементів масиву
    print(mas[i])         # виведення елементів масиву
# або
for i in mas:             # цикл перегляду елементів масиву
    print(mas)            # виведення елементів масиву
```

Рис. 1. Створення масиву в процесі його оголошення

**Приклад 3.**

На рис. 2 зображено програмний код, у якому циклічно обчислюються значення виразу, кожне з яких присвоюється елементу масиву. Після створення масиву елементи виводяться на екран.

```
# Створення масиву шляхом обчислення його елементів
n = int(input('увести довжину масиву: '))
a = int(input('увести число a: '))
g = int(input('увести число g: '))
mas = []                  # порожній масив
for i in range(n):        # цикл формування масиву
    a = a*g               # обчислення виразу
    mas.append(a)         # формування масиву
for i in range(n):        # цикл виведення масиву
    print(mas [i])        # виведення масиву
```

Рис. 2. Створення масиву шляхом обчислення значень його елементів

**Приклад 4.**

На рис. 3 зображено програмний код, у якому масив створюється шляхом уведення значень його елементів із клавіатури. Після введення масиву обчислюється загальна сума й середнє значення елементів масиву.

```
N = int(input('Кількість елементів: '))
a = []                    # порожній масив
for i in range(N):        # цикл уведення
    b = int(input('Увести число: '))
    a.append(b)           # додавання елемента до масиву
s = 0                     # початкове значення суми
for i in range(N):        # цикл виведення та обчислення
    print('Елемент на позиції ', i, '=', a[i])
    s = s + a[i]          # обчислення суми масиву
c = s / N                 # обчислення середнього значення
print('Середнє значення масиву = ', c)
```

Рис. 3. Уведення масиву з клавіатури та обчислення середнього значення

## Запитання для перевірки знань

1. Що називають масивом?
2. Які існують структури масивів?
3. Як можна звернутися до окремого елемента масиву?
4. Яку структуру має одновимірний масив?
5. Якими способами можна створити одновимірний масив?
6. Назвіть порядок створення масиву шляхом уведення даних.

## Завдання для самостійного виконання

1. Створіть масив, елементами якого є рядки: 'звук', 'колонка', 'кодування', 'модель'. Виведіть масив на екран.
2. Створіть масив, елементами якого є цілі числа: 21, 40, 53, 17, 33, що вводяться з клавіатури. Виведіть масив на екран.
3. Створіть масив, елементами якого є 10 випадкових чисел у діапазоні від 4 до 10. Виведіть масив на екран.
4. Створіть масив, елементами якого є 5 перших членів арифметичної прогресії. Перший член 3, її різниця 4. Виведіть масив на екран.
5. Створіть масив із семи випадкових чисел у діапазоні від 2 до 6 обчисліть суму елементів.
6. Обчисліть середнє значення масиву, елементами якого є перші шість членів геометричної прогресії. Перший член 3, знаменник 2.

# 4.4.2. Двовимірні масиви

*Пригадайте, з якими двовимірними масивами ви стикалися під час вивчення хімії. Які операції ви виконували в цих масивах?*

Двовимірний масив, так само як і одновимірний, може містити елементи будь-якого типу, але лише одного. Масив складається з фіксованої кількості рядків і стовпців. У побуті масиви часто називають таблицями, а в математиці — матрицями. Елементи двовимірного масиву беруться у квадратні дужки, елементи кожного рядка також беруться у квадратні дужки, які відокремлюються комою. Усередині рядка елементи також відокремлюються комою. Подання аналога двовимірного масива в Python найкраще реалізується за допомогою списка списків (тип даних list, кожен елемент якого також типу list).

**Двовимірний масив** — це нумерована послідовність однотипних елементів, кожен із яких визначається двома індексами. Такий масив можна подати, наприклад, як таблицю, що містить n рядків і m стовпців.

**Приклад 5.**

На рис. 4 наведено масив цілих чисел із трьома рядками й чотирма стовпцями.

У мові Python нумерація рядків і стовпців починається з нуля. Масив, який наведено на рис. 4, матиме такий вигляд:

```
[[34, 23, 6, 9], [35, 11, 54, 23], [15, 40, 3, 44]].
```

Для наочності краще подати його таким чином:

```
[[34, 23,  6,  9],
[35, 11,  54,  23],
[15, 40,  3,  44]].
```

| Рядки | Стовпці | | | |
|---|---|---|---|---|
| | 0 | 1 | 2 | 3 |
| 0 | 34 | 23 | 6 | 9 |
| 1 | 35 | 11 | 54 | 23 |
| 2 | 15 | 40 | 3 | 44 |

Рис. 4. Масив цілих чисел

**Приклад 6.**

```
>>> mas = [[34, 23 ,6 ,9],
[35, 11, 54, 23],[15 ,40, 3, 44]]
>>> mas [1][2]
54.
```

Із прикладами 7-9 можна ознайомитися на сайті interactive.ranok.com.ua.

У наведених прикладах використано масиви, у яких кількість стовпців у кожному рядку однакова, тобто прямокутні. Деякі мови, наприклад Java, підтримують масиви з різною кількістю стовпців у рядках, — їх називають **нерегулярними**. Крім того, підтримуються **динамічні** масиви — їх розмір (кількість стовпців і рядків) визначається в процесі виконання програми.

Звернення до елемента масиву здійснюється за формою:

<ім'я масиву>[номер рядка][номер стовпця] (приклад 6).

Над елементами двовимірних масивів можуть виконуватися ті самі операції, що й над елементами одновимірних масивів.

- **Обчислення загальної суми й середнього значення елементів масиву.**

Обчислити суму значень елементів масиву можна різними способами. Але найчастіше застосовується алгоритм «накопичення», за яким до початкового значення суми додається елемент, розташований у нульовому рядку нульового стовпця. Потім до отриманої суми поступово додається решта елементів стовпців нульового рядка, здійснюється перехід на наступний рядок, виконуються аналогічні дії і т. д.

- **Обчислення суми значень елементів кожного рядка й загальної суми масиву**

Сума значень елементів рядків двовимірного масиву обчислюють так само, як і для одновимірного. Після завершення обчислення суми елементів одного рядка здійснюється перехід до обчислення суми елементів наступного рядка.

- **Обчислення кількості заданого елемента в масиві**

Алгоритм обчислення кількості заданого елемента у двовимірному масиві відрізняється від аналогічного алгоритму для одновимірного масиву лише тим, що пошук елемента виконується не в одному, а в кількох рядках.

## Запитання для перевірки знань

1. Як у мові Python нумеруються рядки та стовпці двовимірного масиву?
2. Який тип можуть мати елементи двовимірного масиву?
3. Як здійснюється звернення до елементів двовимірного масиву?
4. Які операції можуть виконуватися над елементами двовимірного масиву?
5. Сформулюйте алгоритм обчислення загальної суми двовимірного масиву.
6. Поясніть алгоритм пошуку максимального елемента двовимірного масиву.

## Завдання для самостійного виконання

1. Із двовимірного масиву [['команда','файл','біт'], ['смартфон','миша','байт']] виведіть на екран перший елемент нульового рядка та другий елемент першого рядка.
2. У двовимірному масиві [[5, 3, 12],[13, 7, 7], [21, 6, 8]] визначте загальну суму його чисел.
3. У двовимірному масиві [[34, 11, 23, 19], [18, 19, 37, 51],[77, 20, 35, 55]] визначте мінімальний елемент у кожному рядку.
4. У двовимірному масиві [[77, 32, 23 ,3], [44, 21, 23, 9], [80, 5, 2, 4]] визначте рядок із мінімальною сумою його чисел.
5. Знайдіть в Інтернеті п'ять кращих футбольних команд України за 2015, 2016 і 2017 роки. Створіть двовимірний масив, елементами якого є назви цих команд. Визначте, скільки разів у ньому є команда «Дніпро».

# 4.5. Вказівники

*Вказівники мають фундаментальне значення в будь-якій мові програмування.*

**Вказівник** — це тип даних, який використовується для зберігання адрес змінних і об'єктів.

Адреса містить номери комірок пам'яті або спеціальне значення (часто нульове), яке свідчить про те, що звернення до комірки пам'яті не може бути виконано. Якщо вказівник містить будь-яку адресу, то кажуть, що він посилається на відповідний об'єкт.

Сутність і принципи використання вказівників у типізованих мовах програмування (Pascal, Java, C++ та ін.) майже ідентичні.

У мовах із динамічною типізацією даних, якою є мова Python, наявна суттєва відмінність — змінні не оголошуються, а вказівник присвоюється об'єктам. Вказівники не мають типу, тип мають об'єкти, на які вони посилаються.

Пояснимо сутність вказівників мови Python на прикладі.

У змінній зберігається не сам об'єкт, а посилання на нього, тобто адреса пам'яті, у якій зберігається об'єкт.

**Приклад.**

Припустимо, що для обчислення уводиться вираз 8**3. Спочатку в комірки пам'яті заносяться значення 8 і 3, потім обчислюється вираз 8**3 і значення 512 також буде збережено в комірці. Це значення буде виведено на екран і зміст усіх цих комірок буде стерто. Отже, у мові Python автоматично збирається сміття, тобто пам'ять звільняється від усього, що не використовується.

Щоб зберегти об'єкт у пам'яті, потрібно встановити на нього вказівник. Як уже зазначалося, іменування об'єктів реалізується за допомогою оператора присвоювання (наприклад, zmk = 21). Водночас літерал 21 створює в пам'яті об'єкт типу int(), а zmk є вказівником на цей об'єкт.

Таким чином, 21 — це реально існуючий об'єкт зі власними атрибутами, а zmk — вказівник на нього.

Якщо після цього виконати zmk = '21', то буде створено об'єкт str('21'), на який посилається вказівник zmk, а попередній об'єкт 21 буде видалено з пам'яті.

Розпізнати тип об'єкта, на який посилається цей вказівник, можна за допомогою функції type(zmk).

## Запитання для перевірки знань

1. Дайте означення вказівника мови програмування.
2. Поясніть сутність вказівника мови Python.

# 5. Функції користувача та модулі мови Python

## 5.1. Функції

*Пригадаємо, що в алгоритмізації часто застосовуються допоміжні алгоритми, які в мовах програмування реалізуються підпрограмами. Які види підпрограм ви вивчали у 8 і 9 класах?*

У багатьох мовах програмування підпрограми оформлюються у вигляді функцій і процедур. У мові Python вони реалізуються лише за допомогою функцій.

У мові Python існує чимало убудованих функцій, які вже нами неодноразово використовувалися, наприклад, **del(), len()**. Ці функції не потрібно розробляти — їх слід правильно використовувати; їх може створювати і сам програміст. Такі функції називають **користувацькими**.

**Користувацька функція** — це невеликий, логічно завершений програмний код, до якого можна звертатися багаторазово з різних місць основної програми.

У процесі кожного звернення до функції вона виконує одні й ті самі дії над різними значеннями даних і повертає отримане значення до основної програми. Наприклад, функція обчислення суми членів арифметичної прогресії під час кожного звернення до неї може обчислювати суму її членів із різними значеннями різниці та кількості членів прогресії.

Сутність програми з двома функціями пояснюється схемою (рис. 1).

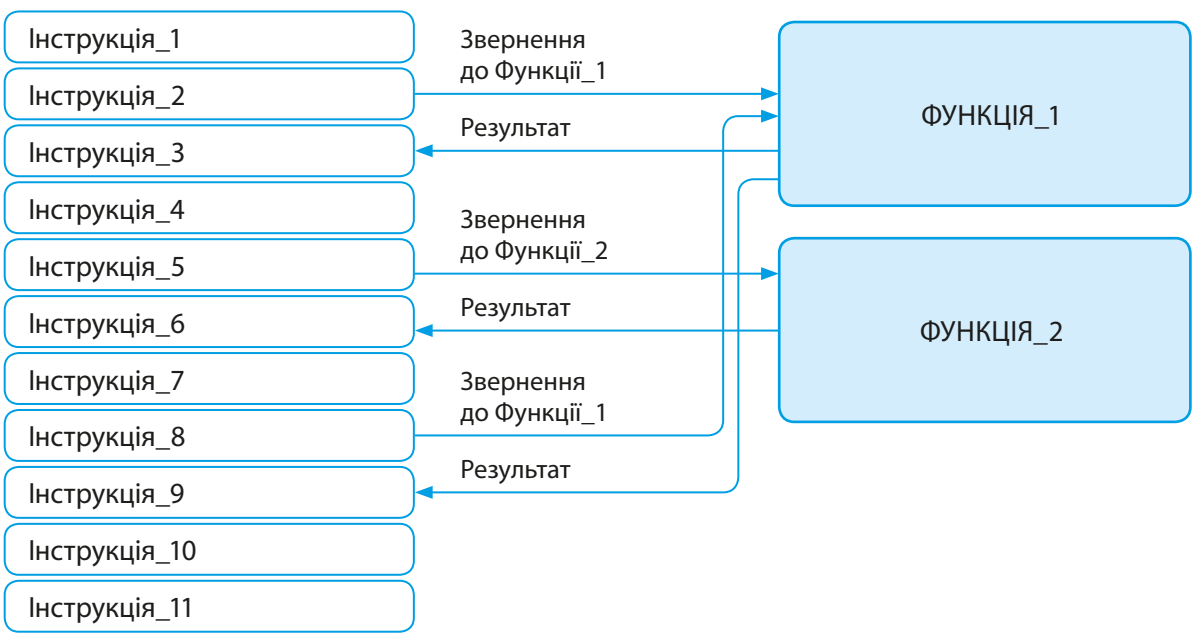


Рис. 1. Приклад структури програми з двома функціями

Як бачимо, основна програма містить 11 інструкцій.

Звернення до першої функції виконується двічі: перший раз з інструкції_2, другий — з інструкції_8. До другої функції звернення виконується один раз — з інструкції_5. Після того

як завершується виконання кожної функції, результат повертається до основної програми в інструкції, що розташовані за інструкцією виклику.

**Функції** — це універсальний засіб структурування програм, вони дають змогу розбити складну програму на окремі частини, можуть викликатися у виразах і застосовуватися в умовних інструкціях **if** та інструкціях циклу **while**.

Використання функцій позбавляє необхідності вставляти до основної програми копії блоків одного й того самого програмного коду. За рахунок цього зменшується загальний обсяг програми і, відповідно, зменшується обсяг пам'яті, потрібної для її збереження. Окрім цього, зменшуються трудовитрати програміста. Наприклад, якщо необхідно змінити інструкцію, то вона змінюється лише один раз у самій функції, а не в багатьох місцях основної програми.

Користувацькі функції оголошуються (визначаються) за допомогою інструкції def, яка має таку структуру:

```
def  <ім'я функції> ([параметри]):
     <тіло функції>
     [return<результат>]
```

Як бачимо, існують функції *з параметрами* і *без параметрів*.

**Ім'я функції** — це звичайний унікальний ідентифікатор латинськими буквами. Ім'я функції складається з малих символів.

**Тіло функції** — це сукупність інструкцій, які реалізують певне завдання, наприклад, інструкції обчислення середнього значення списку чисел.

**Параметри** — це імена об'єктів (змінних, списків тощо), які отримують конкретні значення (атрибути) під час звернення до функцій. Параметри відокремлюються один від одного комою. Якщо функція не має параметрів, зазначаються лише порожні круглі дужки.

Якщо тіло функції не містить інструкцій, слід розмістити оператор pass, який жодних дій не виконує. Наприклад:

```
def  funkt_1:
     pass
```

Оголошення функцій визначається зазвичай на початку програми, одразу після імпортування необхідних модулів.

Усі інструкції тіла функції відокремлюються зліва однаковою кількістю пробілів (бажано дотримуватися чотирьох пробілів). Кінцем функції є перша інструкція, яка має меншу кількість пробілів.

Інструкція **return** повертає результат виконання функції в основну програму. Інструкції, що розташовані в тілі функції після **return**, не виконуються. Якщо інструкцію **return** не зазначено, повертається значення **None**.

**Приклад 1.**

На рис. 2 наведено програмний код, у якому оголошено дві функції: функція без параметрів — funct_03 і функція з двома параметрами x і y — funct_04. Звернення до функції funct_03 здійснюється один раз, а до функції funct_04 — двічі. Після звернення до функції funct_03 виконується множення числа 2.3 на число 3.5 і виведення отриманого результату на екран. Далі управління передаватиме в основну програму інструкції a, b = 2, 3, у результаті чого змінна a набуде значення 2, змінна b — значення 3.

```
def funct_03():          # оголошення функції без параметрів
    y = 2.3 * 3.5        # тіло функції, обчислення виразу
    print(y)             # тіло функції, виведення результату
def funct_04(x, y):      # оголошення функції з параметрами
    z = x * x + y * y    # тіло функції, обчислення виразу return z
    return z             # тіло функції, повернення результату
funct_03()               # звернення до функції funct_03
a, b = 2, 3              # значення аргументів функції funct_04
print(funct_04(a, b))    # перше звернення до функції funct_04
a, b = 3, 4              # нові значення аргументів функції funct_04
print(funct_04(a, b))    # друге звернення до функції funct_04
```

Рис. 2. Програма з двома функціями

У результаті першого звернення до функції funct_04 параметр x набуде значення 2, а параметр y — значення 3. У функції funct_04 є оператор return, тому отриманий результат повертається в програму, що її викликала, і виводиться на екран. Потім змінна x набуває значення 3, а змінна y — 4.

У процесі другого звернення до функції funct_04 змінній x передається значення 3, а змінній y — 4. Отриманий результат повертається в основну програму й виводиться на екран.

Результат виконання програми наведено на рис. 3.

```
8.049999999999999
12
25
```

Рис. 3. Виконання програми з двома функціями

Одна й та сама функція може виконувати різні дії над об'єктами. Яка конкретно дія буде виконуватися, залежить від типу об'єктів, над якими виконуються відповідні операції.

Поліморфізм є одним із фундаментальних понять об'єктно-орієнтованого програмування.

Змінні поділяються на локальні та глобальні.

**Локальні змінні** — це змінні, які оголошені всередині функції. У вже розглянутому прикладі у функції funct_03 локальною змінною є y, а у функції funtc_04 — змінні x, y і z. Локальні змінні доступні тільки програмному коду всередині функції. Вони існують тільки під час виконання функції.

**Глобальні змінні** — це змінні, які оголошені в основній програмі, тобто за межами функції. Вони доступні в будь-якій частині програми, у тому числі всередині функції.

Локальні та глобальні змінні можуть мати однакові імена. Але операції над локальними змінними всередині функцій не впливають на значення однойменних глобальних змінних. Однак для запобігання плутанині краще не користуватися однаковими іменами локальних і глобальних змінних.

Якщо у функції обчислюється значення виразу x + y та ці об'єкти є числами, то буде виконано їх додавання, а якщо об'єктами є рядки — їх конкатенація. Якщо у функції обчислюється значення виразу x * y і ці об'єкти є числами, то буде виконано їх множення, а якщо один із них число, а другий — рядок, то повторення рядка.

Отже, у мові Python саме об'єкти визначають синтаксичну сутність операцій, які будуть виконуватися над ними.

Явище, коли синтаксична сутність операцій залежить від типу об'єктів, які опрацьовуються, називають **поліморфізмом**.

Поліморфними є фактично всі операції у мові Python, а також відповідні функції. Інтерпретатор мови сам перевіряє сумісність типів об'єктів. Якщо виявиться, що вони несумісні, видає повідомлення про синтаксичну помилку.

**Приклад 2.**

На рис. 4 зображено програму, у якій параметрам функції передаються значення кортежів і списків.

У результаті виконання програми отримаємо:

**3.0**
**6.0**

```
def funct_05(a, b, c):          # функція має три параметри
    return (a + b + c) / y      # обчислення і повернення результату
x1 = (2, 3, 4)                  # кортеж
x2 = [5, 6, 7]                  # список
y = len(x1)                     # довжина кортежу
print(funct_05(*x1))            # перше звернення до функції
y = len(x2)                     # довжина списку
print(funct_05(*x2))            # друге звернення до функції
```

Рис. 4. Програма з передаванням функції значень кортежів і списків

Параметрам функцій можна передавати не лише значення літералів, змінних, а й значення кортежів, списків, словників. Якщо параметрам функції передаються значення словника, то в інструкції звернення до функції перед її іменем ставлять два символи зірочка (**). А якщо списку або кортежу, то одну зірочку. Одночасно можна передавати параметрам функції значення різних типів.

**Приклад 3.**

На рис. 5 зображено програму, у якій під час першого звернення до функції параметрам передаються значення словника, а під час другого звернення — значення кортежу та словника.

```
import math                     # імпортування модуля math
def func_06(a, b, c):           # функція з 3 параметрами
    s = a + b + c               # обчислення суми чисел
return math.sqrt(s)             # обчислення кореня квадратного
d1 ={"a":1, "b":2, "c":3}       # словник
# параметри функції набудуть значень словника
print(func_06(**d1))            # перше звернення до функції
# параметри функції набудуть значень кортежу і словника
t, d2 = (5, 6), {"c": 7}        # кортеж і словник
print(func_06(*t, **d2))        # друге звернення до функції
```

Рис. 5. Програма з передаванням функції значень словника й кортежу

Результат виконання програми наведено на рис. 6.

```
2.449489742783178
4.242640687119285
```

Рис. 6. Виконання програми з передаванням функції значень словника й кортежу

Інструкція **def** у процесі оголошення функції створює об'єкт, який має тип **function**, і зберігає посилання на цей об'єкт в ідентифікаторі, яким є ім'я функції. Це посилання на функцію можна зберегти в іншій змінній.

**Приклад 4.**

На рис. 7 зображено програму зі зверненням до функції за допомогою змінної.

```
import math                          # імпортування модуля math
import random                        # імпортування модуля random
def func_07(a):                      # функція з одним параметром
    return math.factorial(a)         # обчислення факторіалу
z = func_07                          # посилання на об'єкт у змінній z
x = random.randint(3, 9)             # x - випадкове число від 3 до 9
print(x)                             # виведення випадкового числа
print(z(x))                          # перше звернення до функції
x = random.randint(2, 7)             # x - випадкове число від 2 до 7
print(x)                             # виведення випадкового числа
print(z(x))                          # друге звернення до функції
```

Рис. 7. Програма з посиланням на функцію за допомогою змінної

Структуру виведення та можливі значення виконання програми наведено на рис 8.

```
9
362880
5
120
```

Рис. 8. Виконання програми з посиланням на функцію за допомогою змінної

## Запитання для перевірки знань

1. Які функції називають користувацькими?
2. Що називають тілом функції?
3. Яка різниця між аргументами й параметрами?
4. Чи завжди функції мови Python мають тіло функції?
5. Для чого застосовується оператор pass?
6. Поясніть сутність звернення до функції із основної програми.
7. Які переваги надає використання функції у програмуванні?
8. Яку роль виконує інструкція return у тілі функції?
9. Поясніть різницю між локальними та глобальними змінними.
10. Поясніть сутність поліморфізму.

## Завдання для самостійного виконання

1. Складіть програму з використанням функції без параметрів для обчислення об'єму конуса.
2. Складіть програму з використанням функцій із параметрами для обчислення двох різних за розміром прямокутних трикутників із відомими значеннями їх катетів і обчисліть площі кола з відомим радіусом.
3. Кубик із цифрами від 1 до 6 підкидають п'ять разів. Складіть програму обчислення середнього значення цих чисел із використанням функції з параметрами.
4. Складіть програму обчислення значення виразу a*b-c/d із використанням функції з параметрами для значень списку [3, 5, 2, 7] і кортежу (4, 1, 5, 6).
5. Генеруються три цілі випадкові числа в діапазоні від 3 до 7. Складіть програму з використанням функції з параметрами для обчислення суми квадратів цих чисел.
6. З'ясуйте в Інтернеті, через які області України протікає Дніпро, та знайдіть їх площі. Складіть програму визначення областей із максимальною та мінімальною площами. Обчисліть середню площу. Використайте функції з параметрами.

## 5.1.2. Розширені можливості функцій

*Який, на вашу думку, недолік мають уже описані користувацькі функції? Чи відомо вам про випадок, коли кількість параметрів і аргументів була неоднаковою?*

Пригадаємо, що в розглянутих прикладах кількість параметрів в оголошенні функції і кількість аргументів у інструкції звернення була однаковою. Водночас значення першого аргументу передається першому параметру функції, значення другого аргументу — другому параметру.

Наприклад, Якщо оголошено функцію def sum(x, y):

```
def sum(x, y):
return x + y,
```

і виконується до неї звернення: sum(a, b), то значення аргументу *a* передається параметру *x*, а значення *b* — параметру *y*.

Такий варіант є основним (класичним), і застосовується в багатьох мовах програмування. Разом із тим, можливості функцій мови Python значно ширші за можливості класичних функцій (приклад 6).

У випадку 1) функція може бути реалізована за умови, що значення деяким параметрам присвоєне в процесі оголошення функції.

Якщо оголошено функцію def sum(x, y=5):

```
def sum(x, y=5):
return x + y,
```

то до неї можна звернутися так: sum(a). До того ж параметр *x* набуде значення *a*, параметр *y* — значення 5.

Параметри, яким присвоюються значення в процесі оголошення функції, називають необов'язковими. Необов'язкові параметри повинні розміщуватися після обов'язкових.

Зокрема мова Python дає змогу:

- в інструкції звернення до функції мати меншу кількість аргументів, аніж кількість параметрів, зазначених у процесі оголошення функції;
- передавати параметрам значення аргументів не послідовно, а в довільному порядку;
- використовувати змінну кількість параметрів у функції;
- під час оголошення не вказувати ідентифікатор функції.

**Приклад 7.**

На рис. 9 зображено програму з функцією обов'язкових і необов'язкових параметрів.

Під час першого звернення до функції число 8 передається тільки параметру x (параметр y матиме значення 3). Під час другого звернення параметр x набуде значення 27, а параметр y — значення 4, незважаючи на те, що в оголошенні функції йому присвоєно значення 3. У результаті <виконання програми> отримаємо числа: 2.0 і 3.0.

```
import math                    # імпортування модуля math
def func_08(x, y=3):           # функція з необов'язковим параметром
    return math.fmod(x, y)     # остача від ділення x на y
a = func_08(8)                 # число 8 передається параметру x
print(a)                       # виведення результату 1 звернення
# параметру x передається число 27, параметру y - число 4
print(func_08(27,4))           # виведення результату 2 звернення
```

Рис. 9. Програма з обов'язковими й необов'язковими параметрами функції

У другому випадку в інструкції звернення до функції вказуються ті самі ідентифікатори аргументів, що й ідентифікатори параметрів, а також присвоюються відповідні їм значення.

Наприклад, у фрагменті програми:

```
def ant(x, y):
    return x * y
print(ant(y=7, x=5))
```

параметр при x набуде значення 5, а параметр при y — значення 7. Такий варіант доцільно використовувати, якщо функція має кілька необов'язкових параметрів.

Розглянемо функції, у яких параметрам можна передавати довільну кількість значень аргументів. Для того щоб параметри могли набувати довільної кількості аргументів, їх у оголошенні функції необхідно вказувати з символом зірочка (*).

**Приклад 8.**

На рис. 10 зображено програму з функцією, параметр якої може набувати довільної кількості аргументів. У цій програмі в процесі першого звернення до функції параметр x набуває двох значень: 9 і 6, а в процесі другого звернення — чотирьох значень: 3, 4, 7, 10. У результаті виконання програми після першого звернення до функції отримаємо значення 20, а після другого — 29.

```
def func_09(*x):              # параметр зі змінною кількістю аргументів
    s = 5                     # початкове значення змінної
    for i in x:               # цикл
        s = s + i             # обчислення суми s+=i
        return s              # повернення результату
print(func_09(9, 6))          # параметру передаються 2 оргументи
print(func_09(3, 4, 7, 10)) # параметру передаються 4 аргументи
```

Рис. 10. Програма з функцією зі змінною кількістю аргументів

В одній функції можна використовувати комбінації різних параметрів: звичайні (обов'язкові) параметри, необов'язкові (параметри за замовчуванням) і параметри зі змінною кількістю аргументів.

**Приклад 9.**

Програму з такою функцією зображено на рис. 11.

У процесі першого звернення до функції параметр при x набуває значення 7, параметр при y за замовчуванням має значення 4, а параметр при z — значення 0. Тому інструкція s=s+i в операторі циклу не виконується.

У процесі другого звернення до функції параметр при x набуває значення 5, параметр при y — значення 8, а параметр при z — двох значень: 9 і 12. Таким чином, вираз s=s+i виконується двічі: для значення змінної при i, яка дорівнює 9 і 12. У результаті виконання програми отримаємо значення 15 і 42.

```
def func_10(x, y=4, *z):          #комбінація різних параметрів
    s = 2 * y + x                 #обчислення виразу для значень x і y
    for i in z:                   #цикл for
#вираз s=s+i не виконується під час першого звернення до функції
#вираз s=s+i виконується двічі під час другого звернення до функції
        s = s + i                 #обчислення суми для значень z
        return s                  #повернення результату
print(func_10 (7))                #перше звернення до функції
print(func_10 (5, 8, 9, 12))      #друге звернення до функції
```

Рис. 11. Програма, функція якої має комбінацію параметрів

**Значення, що повертається,** — це вираз, результат виконання якого повертається функцією. Це значення можна зберегти в змінній або передати як параметр у іншу функцію.

Мова Python реалізує анонімні функції, які ще називають лямбда-функціями. Ці функції не мають імені. Вони оголошуються за допомогою ключового слова lambda за такою структурою:

lambda [<параметр1> [,…,<параметрN>]]:
<значення, що повертається>

**Приклад 10.**

На рис. 12 зображено програму з анонімними функціями.

```
a1 = lambda: 5 * 7 + 9                  #функція без параметрів
a2 = lambda: x, y: x * x + y / 2        #функція з 2 араметрами
a3 = lambda: x, y, z = 3: x / y + 2 * z # функція з необов'язковим параметром
print(a1())                             #звернення до першої функції
print(a2(7, 8))                         #звернення до другої функції
print(a3(5, 6))                         #звернення до третьої функції
```

Рис. 12. Програма з анонімними функціями

У результаті виконання програми отримаємо такі результати (рис. 13):

44,
53.9,
6.83.

Рис. 13. Виконання програми з анонімними функціями

Анонімні функції викликаються так само, як і звичайні функції. Вони можуть бути з параметрами та без параметрів. Як і у звичайних функціях, деякі параметри анонімних функцій можуть бути необов'язковими.

## Запитання для перевірки знань

1. Як передаються значення аргументів необов'язковим параметрам?
2. Як у функції розміщуються необов'язкові та обов'язкові параметри?
3. Які функції називають анонімними?
4. Як оголошують анонімні функції?
5. Як реалізується передавання параметрам у довільному порядку значення аргументів?
6. Як можна реалізувати передавання параметрам довільної кількості значень аргументів?
7. Поясніть порядок використання у функції комбінації різних параметрів.

## Завдання для самостійного виконання

1. Складіть програму обчислення значення виразу a * b2 – 2 * a, якщо a > b і виразу a / b, якщо a <= b із використанням функції з одним обов'язковим та одним необов'язковим параметрами.
2. Складіть програму з використанням однієї функції зі змінною кількістю аргументів для обчислення сум чисел 1, 3, 5, 7, 9 і 6, 8, 10.
3. Складіть програму з використанням анонімних функцій без параметрів і з параметрами для обчислення значень виразів 3 / 7 + 5 * 8 / 3, 2 * x + x / y і (3 * x + 5) / (y – 2) для відомих значень x і y.
4. Функція має три параметри: обов'язковий, необов'язковий, зі змінною кількістю аргументів. Складіть програму обчислення добутку чисел 3 на 7 і добутку чисел 4, 6, 7, 9, 10.
5. У змаганнях із підняття гирі від 10-А класу беруть участь 5 учнів, а від 10-Б класу — 4 учні. Відомо скільки разів підняв гирю кожен учень. Розробіть програму визначення різниці піднять гирі учнями обох класів.
6. Прибуток від двох фірм Кагарлицького району Київської області постійний і складає, відповідно, 10 і 15 млн грн на рік. Ще від двох фірм цього району прибуток визначається наприкінці року. Постійний прибуток також мають дві фірми Обухівського району Київської області й становить, відповідно, 9 і 17 млн грн на рік, а від однієї фірми прибуток визначається наприкінці року. Розробіть програму обчислення загального прибутку фірмами з кожного району. Виберіть і обґрунтуйте раціональний варіант програмного коду.

# 5.2. Рекурсивні функції

*У 8 і 9 класах ви вже ознайомилися з терміном "рекурсія". Спробуйте сформолювати означення рекурсії.*

**Приклад 1.**
Речення «Тип об'єктів визначає синтаксичний смисл оператора» може бути розширено до такого: «Сашко зрозумів, що тип об'єктів визначає синтаксичний смисл оператора», а останнє речення розширено до такого: «Тепер Катря знає, що Сашко зрозумів, що тип об'єктів визначає синтаксичний смисл оператора».

Рекурсія (від латин. *recursio* — повернення) у широкому розумінні — це опис або зображення об'єкта (процесу, явища) через самого себе. Рекурсія застосовується в різних галузях людської діяльності, найчастіше — в математиці й інформатиці.

У лінгвістиці рекурсією називають можливість мови породжувати нові мовні конструкції на основі попередньої (приклад 1).

Класичним прикладом рекурсії в математиці є визначення чисел Фібоначчі, кожне з яких обчислюється як сума двох найближчих попередніх чисел, починаючи з числа, розташованого на третій позиції (n>2), тобто таких, які обчислюються за допомогою формули f(n) = f(n–2) + f(n–1), де n>2. Ці числа мають такі значення: 1, 1, 2, 3, 5, 8, 13, 21...

Як бачимо з прикладу 2, на кроці 4 знайдено опорне значення функції f(3), яке дорівнює 2, а потім у зворотному порядку слід виконати відповідні дії, тобто:

2 + 1 = 3;
3 + 2 = 5;
5 + 3 = 8.

**Рекурсія** в інформатиці — це спосіб організації обчислювального процесу, за яким програма в процесі виконання звертається сама до себе з різними значеннями вхідних параметрів. Цей процес може бути нескінченним, тому для переривання цього процесу в програмі повинна бути умова його переривання.

У стеку спочатку запам'ятовуються всі дії, що виконуються до моменту обчислення опорного значення, а потім у зворотному порядку вони виконуються.

Наприклад, якщо рекурсивна функція обчислює числа Фібоначчі, то запам'ятовуються всі дії f(n) = f(n–2) + f(n–1) до тих пір, поки не буде знайдене опорне значення, а потім у зворотному порядку виконується обчислення.

Отже, для обчислення рекурсивних функцій потрібен стек і виконання щонайменше удвічі більше операцій, ніж, наприклад, у процесі використання рекурентних обчислень.

Пригадаємо, що рекурентне обчислення значення функції на кожному кроці здійснюється через її значення на попередньому кроці.

Уже відомий вам алгоритм *рекурентного обчислення виразу* $y = x^n$ можна подати так:

| | |
|---|---|
| Крок 1 | Увести n, x |
| Крок 2 | y = 1 |
| Крок 3 | i = 1 |
| Крок 4 | y *= x |
| Крок 5 | i += i |
| Крок 6 | Якщо i <= n, то п. 4, інакше п. 7 |
| Крок 7 | Виведення y |
| Крок 8 | Кінець |

**Приклад 2.**
Щоб знайти значення числа на шостій позиції (число 8), потрібно спочатку запам'ятати виконання таких дій:
1) f(6) = f(5) + f(4);
2) f(5) = f(4) + f(3);
3) f(4) = f(3) + f(2);
4) f(3) = f(2) + f(1).

Програми обчислення рекурсій оформлюються у вигляді підпрограм (у мові **Python** — функцій). Реалізація таких функцій заснована на структурі даних, що називають стеком.

У наведенному алгоритмі після того, як вираз **i<=n** на кроці 6 набуде значення **False**, виконання алгоритму завершується і змінна **y** міститиме результат обчислення.

Для рекурсивного обчислення значення $x^n$ (n — додатне ціле, x — може бути дійсне) необхідно обчислити значення $x^{n-1}$, оскільки $x^n = x^{n-1} \cdot x$. У свою чергу, для обчислення $x^{n-1}$ потрібно обчислити $x^{n-2}$, оскільки $x^{n-1}=x^{n-2} \cdot x$ і т. д. Процес обчислення припиняється після обчислення $x^0$, оскільки воно має значення 1.

Усі ці дії запам'ятовуються в стеку. Але отримано не результат, а лише опорне значення функції. Для отримання кінцевого значення функції потрібно реалізувати зворотний процес обчислення, що міститься в стеку.

**Приклад 3.**

Розглянемо сутність рекурсії на прикладі обчислення значення виразу $x^n$. Програму його обчислення зображено на рис. 1.

```
#рекурсивне обчислення числа x у степені n
def st(x, n):                       # заголовок функції
    if n == 0:                      # чи дорівнює число нулю
        return 1                    # повернення одиниці
    else:                           # початок обчислення числа в степені
        return st(x, n-1)*x         # обчислення і повернення результату
print("уведіть число x")            # повідомлення про уведення числа x
x = int(input())                    # уведення числа x
print("уведіть число n")            # повідомлення про введення числа n
n = int(input())                    # уведення числа n
print("результат =", st(x, n))      # звернення до рекурсивної функції
```

Рис. 1. Програма обчислення значення виразу $x^n$

Опишемо **порядок виконання програми.**

1. В основній програмі уводяться числа x і n. Припустимо, що уведені значення x=2 і n=3. Після цього здійснюється звернення до функції st із параметрами st(2, 3). У результаті в стеку виділяється перший блок пам'яті (вершина стеку) й управління передається умовному оператору. Оскільки на цьому етапі вираз n -= 0 набуває значення False, то виконується гілка else. Тут у виразі st(x, n–1)*x робиться спроба обчислити значення st(2, 2), тобто виконується звернення до самої функції st із новими значеннями параметрів. Тому обчислення переривається.

2. Другий раз звернення до функції st виконується з параметрами st(2, 2). У стеку також виділяється наступний блок пам'яті, а в умовному операторі також виконується гілка else і робиться спроба обчислити значення функції st із параметрами st(2, 1).

3. Третій раз звернення до функції st виконується з параметрами st(2, 1). У стеку також виділяється блок пам'яті й знову виконується гілка else з параметрами st(2, 0).

4. Четвертий раз виклик функції виконується з параметрами st(2, 0). У стеку виділяється четвертий блок пам'яті, але виконується гілка then, а функція st набуває значення 1. У результаті функція st більше не викликається і здійснюється повернення в основну програму, яка викликала цю функцію.

5. Далі зі стеку вибирають опорне значення st(2, 0), яке дорівнює одиниці, оскільки в стеку реалізується принцип «останній прийшов — перший вийшов».

6. На наступному кроці обчислюється значення st(2, 1), яке дорівнює 2, потім значення st(2, 2), яке дорівнює 4, нарешті — st(2, 3), яке дорівнює 8. Це значення повертається в основну програму.

Динаміку описаного процесу «заглиблення» й виходу зі стеку для описаного прикладу наведено в таблиці (с. 73).

| Рівень рекурсії | Рекурсивне «заглиблення» | Рекурсивне повернення |
|---|---|---|
| 0 | Виклик st(2, 3) ↓ | →→Вихід st = 8 |
| 1 | st(2, 3) = st(2, 2)*2 ↓ | ↑ st(2, 3) = 2*4 ( 8) |
| 2 | st(2, 2) = st(2, 1)*2 ↓ | ↑ st(2, 2) = 2*2 (4) |
| 3 | st(2, 1) = st(2, 0)*2 ↓ | ↑ st(2, 1) = 1*2 (2) |
| 4 | st(2, 0) = 1 →→ | → |

Класичним варіантом рекурсивної функції є обчислення факторіалу заданого числа n, яке виконується за формулою n! = n * (n–1)!, якщо n>0, і n! = 1, якщо n=0. Програму обчислення факторіалу наведено на рис. 2.

```
# програма обчислення факторіалу уведеного числа
def fact(n):                    # заголовок функції
  if n == 0:                    # пошук опорного значення
      return 1                  # повернення опорного значення
  else:                         # починається обчислення факторіалу
      return n * fact(n-1)      # обчислення і повернення факторіалу
print("уведіть число")          # повідомлення про уведення числа
n = int(input())                # уведення числа
print(fact(n))                  # звернення до рекурсивної функції
```

Рис. 2. Програма обчислення факторіалу з використанням рекурсивної функції

Рекурсія є складним процесом, який вимагає додаткової пам'яті та часу її реалізації. Тому частіше використовується рекурентне обчислення. Однак без рекурсії інколи обійтися досить складно, оскільки без неї алгоритм має заплутану логіку.

**Приклад 4.**

Дано два цілі додатні числа. Розробити програму обчислення найбільшого спільного дільника з використанням рекурсивної функції.

Програма знаходження найбільшого спільного дільника двох цілих чисел без використання рекурсивних підпрограм уже розглядалася раніше. Розв'язання задачі засноване на використанні алгоритму Евкліда. Програму реалізації цього алгоритму з використанням рекурсивної функції зображено на рис. 3. Функція має ім'я nod, а числа, для яких обчислюється найбільший спільний дільник, позначені змінними a і b.

```
# рекурсивне обчислення найбільшого спільного дільника
def nod(a, b):                  # заголовок функції
    if b == 0:                  # перевірка чисел на рівність
        return a                # повернення числа a
    else:                       # починається обчислення nod
        return nod(b, a % b)    # обчислення і повернення числа nod
print("уведіть число a")        # повідомлення про уведення числа a
a = int(input())                # уведення числа a
print("уведіть число b")        # повідомлення про уведення числа b
b = int(input())                # уведення числа b
print("nod = ", nod(a, b))      # звернення до рекурсивної функції
```

Рис. 3. Програма обчислення найбільшого спільного дільника

## Запитання для перевірки знань

1. Що називають рекурсією в програмуванні?
2. Із якою метою в тілі рекурсії передбачається умовний оператор?
3. На якій структурі даних засновано реалізацію рекурсивних функцій?
4. Які дефекти має рекурсивне обчислення?
5. Поясніть сутність «заглиблення» та виходу зі стеку в процесі реалізації рекурсії.

## Завдання для самостійного виконання

1. Розробіть програму з рекурсивною функцією обчислення n перших членів геометричної прогресії, перший член якої дорівнює a і знаменник q.
2. Розробіть програму з використанням рекурсивної функції обчислення кількості цифр у десятковому натуральному числі a.
3. Для одновимірного числового масиву довжиною n розробіть програму з використанням рекурсивної функції обчислення добутку значень його елементів.

# 5.3. Модулі

*Які переваги, на вашу думку, надає модульний принцип будови програмного коду? Із якими модулями стандартної бібліотеки мови Python ви вже стикалися?*

Ми вже розглядали загальну структуру програми мовою Python, яка складалася з кількох файлів. Кожен файл є модулем. Використовувалися окремі модулі зі стандартної бібліотеки, але сутність процесів імпортування модулів не описувалася.

Програма мовою Python складається зі множини текстових файлів. Один із файлів програми є головним, до нього підключаються (імпортуються) додаткові файли, тобто модулі.

Програма запускається з головного файла, який і визначає порядок виконання інших модулів. Файли модулів містять компоненти, які використовує головний файл, а модулі використовують компоненти, що визначені в інших модулях. Але для того щоб їх використати, ці модулі слід імпортувати.

Розглянемо сутність імпортування модулів і використання їх компонентів на прикладі структури найпростішої програми (рис. 1).

У модулі визначаються імена (функції, змінні тощо), які називають **атрибутами модулів**. Вони стають доступними іншим модулям після їх імпортування. Навіть у тому випадку, коли вдається програму логічно оформити в одному модулі, часто доводиться використовувати в ньому вже наявні модулі, зокрема модулі стандартної бібліотеки.


Рис. 1. Структура найпростішої програми

Із рис. 1 видно, що програма складається з модуля головного файла (a.py), додаткового модуля (b.py) і модулів стандартної

бібліотеки. Припустимо, що в модулі b.py визначена функція без параметрів funct_1 (у цьому випадку ця функція є атрибутом модуля b.py):

```
def funct_1():
    y = 7.7 / 3.2
    print(y)
```

Для того щоб функцію funct_1 можна було використовувати в модулі a.py, спочатку необхідно імпортувати модуль b.py до модуля a.py, а потім можна її використовувати за допомогою інструкції b.funct_1().

Імпортування модуля реалізується інструкцією import <ім'я модуля>. За допомогою однієї інструкції import можна імпортувати кілька модулів, наприклад, імпортувати модулі math і random можна за допомогою інструкції:

```
import math, random
```

Після імпортування модуля можна отримати доступ до його атрибутів, які відокремлюються від назви модуля крапкою, наприклад, math.pi:

```
>>> import math
>>> math.pi
3.141592653589793
```

Значення атрибута модуля можна отримати також за допомогою функції getattr () за його назвою.
Функція має такий формат:
getattr(модуль, назва атрибута [,значення за замовчуванням]):

```
>>> import math
>>> print(getattr(math, "pi"))
3.141592653589793
```

Якщо атрибут не знайдено, генерується виняток AttributeError. Для того щоб цей виняток не генерувався, можна у функції getattr() указати значення за замовчуванням, наприклад:

```
>>> import math
>>> print(getattr (math, "ant", "Відсутній"))
Відсутній
```

Перевірити наявність атрибута в модулі можна за допомогою функції hasattr(модуль, назва атрибута). Якщо атрибут існує, повертається значення True.

```
>>> import math
>>> hasattr(math, «pi»)
True
```

Інколи доцільно користуватися не іменем модуля, а його псевдонімом. Наприклад, псевдоніми доцільно застосовувати, якщо ім'я модуля довге. Але якщо оголошено псевдонім модуля, то після цього іменем модуля користуватися не можна. Псевдонім указується в інструкції імпортування модуля за такою структурою: as <ім'я модуля>:

```
>>> import math as p
>>> print(p.pi)
3.141592653589793
```

Стандартна бібліотека Python містить колекцію майже з 200 різноманітних модулів, які забезпечують незалежну платформу підтримки багатьох задач програмування (пошук за шаблоном, створення графічного інтерфейсу й ін.). На цьому етапі нам потрібні лише модулі math і random, модуль sys, що містить інформацію про середовище виконання програм інтерфейсу Python.

Якщо в програму необхідно імпортувати не весь модуль, а лише окремі його ідентифікатори, то можна скористатися інструкцією from. Один із можливих її форматів є таким:

```
from <ім'я модуля>  import <ідентифікатор1> [as <псевдонім1>]
    [,…,<ідентифікаторN> [as <псевдонімN>]]
```

**Приклад.**

```
>>> from random import randint, uniform as un      # імпортування двох функцій
>>> print(randint(2, 10))                          # звернення до функції randint
3
>>> print(un(1, 9))                                # звернення до функції uniform через псевдонім
1.207729134588777
```

У процесі збереження розроблених файлів програмного коду розширення .py не застосовувалося. Це пояснюється лише тим, що програми містили лише головні файли програм, які не передбачалося імпортувати до інших модулів. Але для файлів, які планується імпортувати в інші, розширення .py слід обов'язково вказувати.

Із наведеного прикладу видно, що різниця між інструкціями import і from полягає також і в порядку звернення до атрибутів імпортованих модулів. Якщо імпортування модуля виконується за допомогою інструкції import, для звернення до його атрибутів необхідно вказувати через крапку ім'я цього модуля, наприклад, math.pi. Якщо ж імпортування здійснюється за допомогою інструкції from, указувати ім'я модуля не потрібно.

Відзначимо також, що в процесі першого запуску програми під час виконання інструкції імпортування модуля спочатку здійснюється його пошук, потім трансляція в байт-код, після чого запускається програмний код модуля. Під час наступних запусків ці дії не виконуються, а програмний код модуля просто вибирається з пам'яті.

Такий варіант застосовується для невеликих програм, зокрема для програм навчального призначення, які зберігаються в одній папці. Система в такому разі сама автоматично здійснює їх пошук. Цей варіант вважається стандартним шляхом пошуку файлів. Файли реальних програм можуть зберігатися в різних папках і тому в процесі імпортування необхідно вказувати шлях до кожного з них. Ці питання тут не розглядаються.

Зазначимо, що в розглянутих прикладах програм використовувалося імпортування окремих модулів і їх атрибутів. Але імпортувати можна також і каталоги, які в мові Python називають пакетами. Найчастіше пакети застосовуються для групування модулів за їх функціональним призначенням.

Пакети мають деякі специфічні особливості, характерні для мови Python. Вони застосовуються переважно в професійному програмуванні й тому тут не розглядаються.

## Запитання для перевірки знань

1. Для чого в мові Python застосовується імпортування модулів?
2. Який файл називають головним файлом програми?
3. За допомогою яких інструкцій імпортуються модулі?
4. Назвіть складові програми мовою Python.
5. Поясніть на прикладі сутність імпортування модулів.
6. Із якою метою застосовуються псевдоніми модулів?
7. Поясніть різницю між інструкціями import і from.
8. За допомогою якої інструкції перевіряється наявність атрибута в модулі?
9. У яких випадках у файлі програмного коду не застосовується розширення .py?

# 6. Класи, об'єкти, наслідування

Основний дефект процедурного стилю програмування полягає в тому, що він слабко пристосований для розроблення великих програм. Із часом програми стають настільки складними, що розібратися в них непросто. Одним із напрямків розв'язування цієї проблеми є об'єктно-орієнтоване програмування (ООП).

## 6.1. Елементи теорії об'єктно-орієнтованого програмування

*Із якими об'єктами ОПП ви вже ознайомилися?*

ООП базується на трьох **основних принципах**:

- наслідування,
- поліморфізм,
- інкапсуляція.

Ці принципи є досить абстрактними та складними для розуміння на початкових етапах ознайомлення з мовою. В ООП загалом використовується значна кількість нових термінів. Щоб свідомо опанувати їх, розглядатимемо їх поступово на прикладах.

До основних понять об'єктно-орієнтованого програмування належать **об'єкт**, **клас**, **метод**.

Для загального ознайомлення наведемо основні терміни.

**Клас (Class)** — об'єкт, що складається із сукупності методів і змінних (атрибутів), які описують цей об'єкт.

**Метод (Method)** — сукупність інструкцій мови опрацювання даних.

**Змінна класу (Class variable)** — звичайна змінна, що визначається всередині класу (а не всередині методів класу), яка є доступною для всіх екземплярів цього класу.

**Екземпляр класу (Instance)** — окремий об'єкт класу.

**Змінна екземпляра класу (Instance variable)** — змінна, що визначена всередині методу класу і яка належить тільки цьому класу.

Сутність ООП у різних джерелах тлумачиться по-різному. Здається, що найбільш удало та просто описав її Алан Кей. Він зазначив, що мову можна назвати ООП, якщо вона відповідає певним вимогам.

Мова ООП має відповідати таким вимогам:

- усі дані подаються об'єктами;
- програма є набором об'єктів, які взаємодіють між собою та надсилають один одному необхідні повідомлення;
- кожен об'єкт може мати у своєму складі інші об'єкти і для всіх них виділяється власна частина пам'яті;
- кожен об'єкт належить одному типу (класу), який задає поведінку об'єктів, створених на їх основі;
- об'єкти одного типу можуть виконувати одні й ті самі дії.

Алан Кертіс Кей — американський інформатик, відомий своїми працями в галузі ООП, президент дослідного інституту В'юпоїнта, ад'юнкт-професор інформатики в Каліфорнійському університеті (США). Кей сказав, що найкращий спосіб спрогнозувати майбутнє — винайти його.

Використання принципів ООП у мові Python не є обов'язковим. Залежно від особливостей задачі, програму можна розробляти на основі принципів як процедурного програмування, так і ООП. Така **унікальність мови** дає змогу розробляти досить компактні програми, адже не для кожної задачі ОПП є оптимальним.

**Гібридна мова** поєднує властивості як процедурного, так і об'єктно-орієнтованого програмування. Саме до них належить мова Python.

Під час кожного звернення до класу створюється новий об'єкт — екземпляр класу, тому можна отримати необмежену кількість екземплярів на основі цього класу.

Вимоги до мов ООП у різних мовах програмування реалізуються по-різному: найскладніше в мовах Java і C++, значно простіше — у мові Python. Уже розроблені нами програми складалися з окремих частин (фрагментів) на основі функцій і модулів, до яких можна було звертатися багаторазово. Зважаючи на те, що в цих програмах застосовувалися об'єкти та методи (що є базовими поняттями ООП), такі програми не можна в повному розумінні віднести до ООП. Досі ми не використовували такі фундаментальні поняття, як клас, екземпляр класу, наслідування й ін. А без них про реалізацію ООП не може навіть ітися.

Переваги ООП стають відчутними лише в процесі розв'язування великих і складних завдань. Для багатьох задач середньої складності, особливо навчального призначення, не тільки можна, а й доцільно застосовувати гібридні мови.

Така особливість мови Python, як гібридність, дала змогу на початкових етапах її вивчення основну увагу приділити синтаксичним конструкціям, функціям і модулям, без яких не може існувати не лише процедурне, а й об'єктно-орієнтоване програмування. Розуміння цих питань допоможе тепер якісно оволодіти й методами ООП.

**Характерними особливостями** ООП мовою Python можна вважати такі:

- основною складовою мови ООП є клас, головне призначення якого — створення та маніпулювання об'єктами, а також підтримка механізму наслідування як способу адаптації програмного коду для розв'язування однотипних задач і неодноразового його використання;
- об'єкти мови Python нагадують убудовані типи, які було описано раніше;
- програма складається з окремих, достатньо незалежних частин, що суттєво полегшує її розуміння й читання;
- нова програма може розроблятися не «з нуля», а шляхом адаптування вже наявного програмного коду, без його зміни.

Отже, основою ООП є клас — це складний тип даних із певним набором змінних (атрибутів) і функцій (методів) опрацювання значень, що зберігаються в цих змінних.

## Запитання для перевірки знань

1. На яких основних принципах базується ООП?
2. Для яких задач відчутні переваги ООП?
3. Яким основним вимогам має відповідати ООП?
4. Які мови називають гібридними?
5. Які фундаментальні поняття застосовуються в ООП?
6. Назвіть основні особливості ООП мовою Python.

# 6.2. Створення класів і об'єктів

*Пригадайте, які елементи ООП ви вже використовували.*

У мові Python реалізація принципів ООП практично зводиться до виконання виразу: <об'єкт>.<атрибут>. Цей вираз уже неодноразово використовувався нами для доступу до атрибутів модуля, звернення до методів об'єктів тощо. Але реалізація таких виразів відбувалася за відсутності в програмі класів.

Якщо програма містить класи, то створюється дерево пов'язаних об'єктів. У такому разі в процесі реалізації виразу <об'єкт>.<атрибут> інтерпретатор починає пошук зазначеного атрибута в дереві пов'язаних об'єктів у такому порядку: спочатку атрибут відшукується в цьому самому об'єкті, потім — у дереві об'єктів знизу догори і зліва направо. Пошук завершується, щойно буде знайдено першу появу атрибута.

Тіло класу містить змінні та функції класу. Функції в ООП називають **методами**, а змінні класу — **атрибутами**. Атрибути створюються як звичайні змінні, а методи — як звичайні функції за допомогою інструкції def.

Об'єкти в програмі можна створювати лише на основі класів, тому програма починається зі створення класу. Класи можна розробляти самостійно або імпортувати з інших модулів. Вони розміщуються на початку програмного коду.

Щоб визначити, який саме екземпляр класу слід опрацьовувати, у методі обов'язково слід указати перший параметр з іменем self (інші параметри можуть бути відсутні). Цьому параметру автоматично передається посилання на відповідний екземпляр класу.

За допомогою параметру self здійснюється доступ до змінних екземпляра класу та методів класу всередині визначеного методу. Параметр self відокремлюється від змінної або методу крапкою. Наприклад, до змінної ask із методу класу можна звернутися так: self.ask.

Класи створюються за допомогою інструкції class і мають таку загальну структуру:
**class <ім'я класу> [(класи, що є нащадками)]:**
**<тіло класу>**

**Структуру класу** можна подати детальніше:

```
class <ім'я класу> ([суперклас_1, суперклас_2,...]):
        змінна = значення
        def ім'я методу (self,...):
            self.змінна = значення
...
```

Методи можна розглядати як невеликі програми, призначені для опрацювання об'єктів, тобто вони створюють нові властивості об'єктів, змінюють наявні тощо. Методи фактично є аналогами підпрограм у мовах процедурного програмування.

Необхідно чітко розуміти, що в результаті виконання інструкції class буде створено новий екземпляр класу, який також є об'єктом, посилання на який присвоюється імені змінної. Якщо цей клас має батьківські класи (суперкласи), від якого він є нащадком, то вони вказуються в круглих дужках через кому. Якщо він не є нащадком, то круглі дужки не вказуються.

Для того щоб можна було використовувати атрибути та методи класу, необхідно створити екземпляр класу за такою структурою:

<екземпляр класу> = <ім'я класу> ([параметри])

Метод викликається з основного блоку програми за допомогою інструкції:
**<екземпляр класу>.<ім'я методу> ([параметри])**

Тут <екземпляр класу> є звичайним ідентифікатором. У результаті кожного виконання цієї інструкції у пам'яті комп'ютера виділяється ділянка пам'яті, доступ до якої здійснюється за допомогою пов'язаного з ним ідентифікатора, тобто в пам'яті створюється об'єкт. Цей об'єкт має характеристики того класу, у якому його створено, тобто він отримує атрибути свого класу.

Під час кожного звернення до класу створюється новий екземпляр класу з різними значеннями атрибутів. Наприклад, якщо в класі обчислюється об'єм піраміди з висотою h, основою якої є квадрат зі стороною a, то під час кожного звернення до класу буде створюватися такий самий об'єкт з атрибутами h і a, але значення їх будуть різними.

Зазвичай класи без методів не використовуються.

**Приклад 1.**

Для усвідомлення сутності ООП спочатку розробимо найпростішу програму без методу (рис. 1).

```
class Kl7_01:          # клас з іменем Kl7_01
    a = "гарно"        # змінна a класу kl7_01
ob1 = Kl7_01()         # екземпляр ob1 класу Kl7_01
ob2 = Kl7_01()         # екземпляр ob2 класу Kl7_01
print(ob1.a)           # виведення значення змінної a екземпляра ob1
print(ob2.a)           # виведення значення змінної a екземпляра ob1
```

Рис. 1. Найпростіша програма без методу

У цій програмі оголошено змінну (атрибут) з іменем a і два екземпляри ob1 і ob2 класу Kl7_01. Імена екземплярів різні, тому для них буде відведено дві ділянки пам'яті, але їх зміст однаковий і має значення змінної a, тобто гарно. Виведення атрибута цих об'єктів здійснюється двічі за допомогою вбудованої функції print. У результаті буде двічі виведено слово гарно.

Додамо в цю програму метод з іменем func і параметром self (рис. 2). В інструкції цього методу виводиться значення змінної a, до якого додається слово і правильно.

```
class Kl7_02:                          # клас з іменем Kl7_02
    a = "гарно"                        # змінна a класу kl7_02
    def func(self):                    # метод func із параметром self
        print(self.a + "і правильно")  # інструкція методу
ob1 = Kl7_02()                         # екземпляр ob1 класу Kl7_02
ob2 = Kl7_02()                         # екземпляр ob2 класу Kl7_02
# виведення значення змінної a екземпляра ob1 за допомогою вбудованої функції
print(ob1.a)
# виведення значення змінної a екземпляра ob2 за допомогою вбудованої функції
print(ob2.a)
ob1.func() # виклик методу func екземпляра ob1 для виведення нового значення
ob2.func() # виклик методу func екземпляра ob2 для виведення нового значення
```

Рис. 2. Найпростіша програма з одним методом

Метод func викликається в програмі двічі: з екземплярів класу ob1 і ob2. Результат виведеннння зображено на рис. 3.

```
гарно
гарно
гарно і правильно
гарно і правильно
```

Рис. 3. Виконання програми з одним методом

У програмі (див. рис. 3) використовувався один метод лише зі стандартним параметром self.

**Приклад 2.**

Розглянемо програму з двома методами, у кожному з яких, окрім параметра self, використовується ще один параметр (рис. 4.).

```
class Kl7_03:                          # клас Kl7_03
    predm = "інформатика"              # змінна predm за межами методів
    klas = 8                           # змінна klas за межами методів
    def func_1(self, a1):              # метод func_1 з параметром a1
        self.predm = a1                # зміна значення змінної predm
    def func_2(self, a2):              # метод func_2 з параметром a2
        self.klas = a2                 # зміна значення змінної klas
ob1 = Kl7_03()                         # екземпляр ob1 класу Kl7_03
ob2 = Kl7_03()                         # екземпляр ob2 класу Kl7_03
# виведення значень змінних predm і klas екземпляра ob1
print(ob1.predm, ob1.klas)
# виведення значень змінних predm і klas екземпляра ob2
print(ob2.predm, ob2.klas)
# звернення до методу func_1 екземпляра ob1 зі значенням аргументу "математика"
ob1.func_1("математика")
# звернення до методу func_1 екземпляра ob2 зі значенням аргументу "історія"
ob2.func_1("історія")
# звернення до методу func_2 екземпляра ob2 зі значенням аргументу 9
ob2.func_2(9)
# виведення значень змінних predm і klas після їх зміни в методі func_1
print(ob1.predm, ob1.klas)
# виведення значень змінних predm і klas після їх зміни в методі func_2
print(ob2.predm,  ob2.klas)
```

Рис. 4. Програма з двома методами, у кожному з яких два параметри

Основна ідея цієї програми полягає в демонстрації порядку зміни властивостей створених об'єктів за допомогою методів. Дві змінні predm і klas оголошені поза межами методів. Тому обидва об'єкти (ob1 і ob2) набувають властивості, які мають ці змінні, тобто інформатика і 8. А потім за допомогою методів ці властивості змінюються.

У програмі метод func_1 має параметри self і a1, а метод func_2 — параметри self і a2. Ці методи виконують тільки одну дію, а саме: метод func_1 змінює значення змінної predm, а метод func_2 — значення змінної klas.

У програмі створено два екземпляри класів: ob1 і ob2. Поза межами методів змінна predm набуває значення «інформатика», а змінна klas — значення 8. За допомогою інструкцій print (ob1.predm, ob1.klas) і print (ob2.predm, ob2.klas) здійснюється виведення значень цих змінних.

Зважаючи на те, що в першій інструкції змінні викликаються з екземпляра ob1, а в другій інструкції вони викликаються з екземпляра ob2, значення змінних однакове, тому двічі буде виведено інформатика 8.

Далі за допомогою інструкції ob1.func_1 («математика») з екземпляра ob1 викликається метод func_1 з аргументом математика. Цей аргумент передається параметру a1, а за допомогою інструкції self.predm = a1 змінна predm набуває нового значення, а саме — математика.

Звернемо увагу, що значення змінної klas екземпляра ob1 не змінювалося, тому за допомогою інструкції print (ob1.predm, ob1.klas) буде виведено математика 8.

Після цього за допомогою інструкцій ob2.func_1 («історія») і ob2.func_2 (9) викликаються методи func_1 і func_2 з екземпляра ob2. У результаті обидві змінні змінюють своє значення і буде надруковано історія 9. Результат виконання програми наведено на рис. 5.

```
інформатика 8
інформатика 8
математика 8
історія 9
```

Рис. 5. Виконання програми з двома методами

Розглянемо ще один приклад для розв'язування такої задачі.

**Приклад 4.**

Дано два конуси різних розмірів із відомими радіусом основи й висоти. Потрібно обчислити об'єм одного з них і загальну площу другого. Програму розв'язування цього завдання зображено на рис. 6.

```
class Kl7_04:                                    # клас Kl_04
  def func_1 (self, a1, a2):                     # метод func_1 з параметрами self, a1 і a2
      volume = 1/3 * (3.14 * a1 * a1 * a2)       # обчислення об'єму конуса
      return volume                              # повернення результату
  def func_2 (self, a3, a4):                     # метод func_2 з параметрами self, a3 і a4
      v = a3 * a3 + a4 * a4                      # сума квадратів радіуса і висоти
      surface =  3.14*a3*(a3+math.sqrt(v))       # обчислення повної площі конуса
      print("площа = ", surface)                 # виведення площі конуса
import math                                      # імпортування модуля math
ob1 = Kl7_04()                                   # екземпляр ob1 класу Kl7_04
ob2 = Kl7_04()                                   # екземпляр ob2 класу Kl7_04
# звернення до методу func_1 і виведення результату
print ("об'єм = ", ob1.func_1(3, 4))
ob2.func_2(4, 5)                                 # звернення до методу func_2
```

Рис. 6. Програма з двома методами, кожен із яких має три параметри

Для обчислення об'єму конуса використовують метод func_1 із параметрами self, a1 і a2, а для обчислення повної поверхні конуса — метод func_2 із параметрами self, a3 і a4.

Звернемо увагу, що в програмі відсутні змінні класу, тобто змінні за межами методів. У процесі звернення до методу func_1 параметру a1 передається число 3, а параметру a2 — число 4. Результат обчислення об'єму конуса повертається в інструкцію, з якої відбулося звернення до методу, тобто до інструкції print ("об'єм = ", ob1.func (3, 4)).

Отже, результат обчислення буде виведено на екран. Після звернення до методу func_2 параметр a3 набуде значення 4, а параметр a4 — значення 5. Результат обчислення не повертається в інструкцію, що його викликала, а виводиться за допомогою інструкції print ("площа = ", surface) у цьому самому методі.

Результат виконання програми наведено на рис. 7.

```
об'єм = 37.67999999999999
площа = 130.66324042215658
```

Рис. 7. Виконання програми з двома методами, кожен із яких має три параметри

## Запитання для перевірки знань

1. За допомогою якої інструкції створюється клас?
2. Які основні складові містить тіло класу?
3. Що називають методом класу?
4. Із якою метою в методі класу використовується параметр self?
5. За допомогою якої інструкції створюється метод класу?
6. Як створюється об'єкт класу?
7. Яка інструкція використовується для звернення до методу класу?

## Завдання для самостійного виконання

1. Змінна *a1* набуває значення «інформатика», а змінна *a2* — «фізика». Створіть програму з двома об'єктами, але без методу, за допомогою якої з використанням першого об'єкта виводиться слово інформатика, а з використанням другого — фізика.
2. Змінна *a1* набуває значення «байт». Створіть програму з двома об'єктами і функцією. Метод викликається спочатку з першого, потім — із другого об'єкта. Кожного разу виводиться значення змінної *a1*.
3. Змінна *a* набуває значення «Україна», а змінна *b* — «Франція». За допомогою інструкції input() уводиться довільне ціле число. Розробіть програму без методів, у якій створюються два об'єкти. Якщо уведене число більше 5, виводиться значення з використанням першого об'єкта, інакше — з використанням другого об'єкта.
4. Створіть програму, у якій одна змінна набуває значення «файл», а друга — «папка». Метод, який викликається з обох об'єктів, виводить ці слова.
5. Розробіть програму з двома методами. У першому з них обчислюється площа (у га), що припадає на одного мешканця чи мешканку Дніпропетровської області, а в другому — Хмельницької області.
6. Розробіть програму з використанням двох методів. В одному з них обчислюється об'єм кулі радіусом *r*, *a* в другому — площа трикутника зі сторонами *a*, *b*, *c*.

# 6.3. Конструктор класу

*Пригадаємо, що ініціалізація змінних класу здійснювалася під час кожного створення його екземплярів, що є суттєвим дефектом. Для його подолання в мові Python застосовується конструктор класу.*

У мові Python є спеціальний метод __init__ (зліва і справа по два знаки підкреслення), який називають **конструктором класу**.

Цей метод викликається автоматично під час кожного створення екземпляра класу. Якщо метод __init__ містить параметри, то в інструкції створення екземпляра класу в круглих дужках указуються аргументи, які, відповідно, передаються параметрам.

У нашому випадку екземпляр класу створюється за такою структурою:

<екземпляр класу>=<ім'я класу>(аргумент1,…,аргументN)

Метод __init__ викликається автоматично. Наразі виконуються такі дії. Посилання на екземпляр класу передається першому параметру методу __init__ (таким параметром є self). Це означає,

**Загальна форма методу конструктором класу** така:

```
def __init__(self
[,<параметр1>[,…,<параметрN>]]):
    <тіло методу>
```

що параметр self указує на об'єкт, із якого викликається метод, а всі інші значення аргументів, перелічені в круглих дужках, передаються іншим параметрам методу.

Під час створення екземпляра класу щоразу викликається метод __init__. Новий екземпляр класу передається як аргумент параметру self, а значення аргументів — відповідним параметрам методу.

Значення параметрів у методі __init__ можна присвоювати не тільки в процесі створення об'єкта, а й безпосередньо в самому методі. Наприклад:

```
def __init__(self, a1="файл", a2=25):
```

**Приклад 1.**

На рис. 1 зображено найпростішу програму, у якій демонструється сутність описаних процесів:

```
class Kl7_05:                          # клас Kl7_05
    def __init__(self, a1, a2):        # конструктор класу
        self.p1 = a1  # змінна екз. класу p1 набуває значення параметра a1
        self.p2 = a2  # змінна екз. класу p2 набуває значення параметра a2
# аргумент Вивчаємо передається параметру a1, аргумент інформатику - a2
ob1 = Kl7_05("Вивчаємо", "інформатику")    # екземпляр класу з аргументами
print(ob1.p1, ob1.p2)                  # Виведення значень змінних екз. класу
```

Рис. 1. Найпростіша програма з конструктором

Зробимо пояснення до цієї програми. Перш за все відзначимо, що змінні p1 і p2 є змінними екземпляра класу, в цьому випадку — екземпляра ob1. Тепер чітко зрозуміла різниця між змінними класу та змінними екземпляра класу. За допомогою інструкції ob1 = Kl7_05 («Вивчаємо»,«інформатику») створюється екземпляр класу, аргументами якого є слова Вивчаємо й інформатику.

Під час створення екземпляра класу автоматично викликається метод __init__, у результаті чого посилання на екземпляр ob1 буде присвоєно параметру self, слово Вивчаємо передається параметру a1, а слово інформатику — параметру a2. Наступними інструкціями (self.p1=a1 і self.p2.a2) значення цих слів присвоюються, відповідно, змінним p1 і p2 екземпляра ob1. Тому в процесі виконання інструкції print (ob1.p1, ob1.p2) на екран буде виведено Вивчаємо інформатику. Отже, атрибути (змінні) p1 і p2 екземпляра класу ініціалізуються в процесі створення саме екземпляра класу.

**Приклад 2.**

Для свідомішого розуміння сутності конструктора й доцільності його використання складемо програму, яка виконує ті самі дії, що й попередня програма, але без конструктора (рис. 2).

```
class Kl7_06:                      # клас Kl7_06
    def stm(self, a1, a2):         # метод stm з параметрами
        self.p1 = a1               # змінна p1 екз. класу набуває значення параметра a1
        self.p2 = a2               # змінна p2 екз. класу набуває значення параметра a2
ob1 = Kl7_06()                     # екземпляр класу ob1 класу Kl7_06 без аргументів
ob1.stm("Вивчаємо","інформатику")  # звернення до методу stm
print(ob1.p1, ob1.p2)              # виведення значень змінних p1  p2
```

Рис. 2. Програма без конструктора

Як бачимо, ця програма складніша. У ній екземпляр класу створюється без аргументів і необхідно викликати метод stm, у якому значення його аргументів присвоюються змінним екземпляра, які потім виводяться на екран.

Якщо метод викликається з екземпляра класу без аргументів, то значення, що вказані в методі, зберігаються. Якщо ж у екземплярі класу вказані аргументи, то вони передаються в метод, а значення, що вказані в методі, не враховуються.

**Приклад 3.**

На рис. 3 зображено програму, у якій демонструються сформульовані положення.

```
class Kl7_07:                                        # клас Kl7_07
    def __init__(self, a1="Харків", a2="Одеса"): # конструктор
        self.p1 = a1  # змінна p1 екз. класу набуває значення параметра a1
        self.p2 = a2  # змінна p2 екз. класу набуває значення параметра a2
ob1 = Kl7_07("Київ", "Полтава")  # екз. класу ob1 з двома аргументами
ob2 = Kl7_07("Львів")            # екз. класу ob2 з одним аргументом
ob3 = Kl7_07()                   # екз. класу ob3 без аргументів
print(ob1.p1, ob1.p2)            # виведення Київ Полтава
print(ob2.p1, ob2.p2)            # виведення Львів Одеса
print(ob3.p1, ob3.p2)            # виведення Харків Одеса
```

Рис. 3. Програма з різними варіантами конструктора

Звернемо увагу, що екземпляр класу ob3 не має аргументів, тому вказані в методі значення параметрів (Харків і Одеса) зберігаються. В екземплярі ob2 є тільки один аргумент (Львів). Тому змінюється лише значення параметра a1, а значення параметра a2 залишається незмінним. На рис. 4 наведено результат виконання програми.

```
Київ Полтава
Львів Одеса
Харків Одеса
```

Рис. 4. Виконання програми з різними варіантами конструктора

Конструктор може одночасно містити як параметри, яким не присвоюються значення, так і параметри, яким присвоюються відповідні значення. У такому разі параметри, яким не присвоюються значення, вказуються першими, а параметри, яким присвоюються значення — після них.

**Приклад 4.**

На рис. 5 зображено програму з такими параметрами.

```
class Kl7_08:                                # клас Kl7_08
    def __init__(self, a1, a2=10):      # конструктор
        self.p1 = a1      # змінна p1 екз. класу набуває значення параметра a1
        self.p2 = a2      # змінна p2 екз. класу набуває значення параметра a2
ob1 = Kl7_08("принтер", "P1006")       # екз. класу ob1 з двома аргументами
ob2 = Kl7_08("Windows")  # екз. класу ob2 з одним аргументом
print(ob1.p1, ob1.p2)    # виведення значень змінних екз. класу ob1
print(ob2.p1, ob2.p2)    # виведення значень змінних екз. класу ob2
```

Рис. 5. Програма з різними параметрами

Результат виконання програми наведено на рис. 6.

```
принтер P1006
Windows
```

Рис. 6. Виконання програми з різними параметрами

### Приклад 5

Проводиться жеребкування ліги чемпіонату Європи з футболу серед 16 команд, які залежно від отриманого номера потрапляють в одну з чотирьох груп. Якщо команда отримує номер менший 5, вона потрапляє в першу групу, якщо номер більший 4, але менший 9, то в другу групу, якщо номер більший 9, але менший 13, то в третю групу, інакше — в четверту групу. На рис. 7 зображено програму реалізації результатів жеребкування для чотирьох команд.

```
class K17_09:                                  # клас K17-09
    def __init__(self, a1, a2, a3):           # конструктор
        self.p1 = a1        # змінна p1 екз. класу набуває значення параметра a1
        self.p2 = a2        # змінна p2 екз. класу набуває значення параметра a2
        self.func1(a3)      # звернення до функції func1 з певного екз. класу
    def func1(self, a3):    # функція func1 з двома параметрами
        if a3 <= 4:         # якщо значення a3 менше 4
            self.p3 = "-перша група"     # значення змінної p3 певного екз. кл.
        elif 4 < a3 <= 8:                # якщо 4<a3<=8
            self.p3 = "-друга група"     # значення змінної p3 певного екз. кл.
        elif 8 < a3 <= 12:               # якщо 8<a3<=12
            self.p3 = "-третя група"    # значення змінної p3 певного екз. кл.
        else:                            # якщо значення параметра a3 більше 12
            self.p3 = "-четверта група" # значення змінної p3 певного екз. кл.
ob1 = K17_09("Динамо", "Київ", 7)        # екземпляр класу ob1 з 3 аргумент.
ob2 = K17_09("Реал", "Мадрид", 3)        # екземпляр класу ob2 з 3 аргумент.
ob3 = K17_09("Баварія", "Мюнхен", 9)     # екземпляр класу ob3 з 3 аргумент.
ob4 = K17_09("Інтер","Мілан", 5)         # екземпляр класу ob4 з 3 аргумент.
print(ob1.p1, ob1.p2, ob1.p3)            # виведення даних про Динамо
print(ob2.p1, ob2.p2, ob2.p3)            # виведення даних про Реал
print(ob3.p1, ob3.p2, ob3.p3)            # виведення даних про Баварію
print(ob4.p1, ob4.p2, ob4.p3)            # виведення даних про Інтер
```

Рис. 7. Програма жеребкування команд ліги чемпіонату Європи з футболу

Після запуску програми в процесі створення екземпляра ob1 класу K17_09 викликається конструктор, у результаті чого параметр self набуває значення екземпляра класу ob1, параметр a1 — значення Динамо, параметр a2 — значення Київ, параметр a3 — значення 7. Потім викликається функція func1 із параметром a3. У ній визначається, до якої групи потрапляє команда «Динамо» і назва цієї групи запам'ятовується в змінній p3 екземпляра класу ob1. Далі виконуються аналогічні дії для екземплярів класу ob2, ob3 і ob4. У результаті для кожного об'єкта буде виділено ділянки пам'яті, у яких будуть зберігатися значення змінних p1, p2 і p3. Наступними чотирма інструкціями значення змінних кожного екземпляра класу виводяться на екран. Результат виконання програми наведено на рис. 8.

```
Динамо Київ - друга група
Реал Мадрид - перша група
Баварія Мюнхен - третя група
Інтер Мілан - четверта група
```

Рис. 8. Виконання програми жеребкування команд ліги чемпіонату Європи з футболу

## Запитання для перевірки знань

1. Що називають конструктором класу в мові Python?
2. Яку загальну структуру має метод __init__?
3. У яких випадках викликається метод __init__?
4. Які дії виконуються в процесі створення об'єкта і наявності методу __init__?
5. Що передається параметру self у процесі створення об'єкта?
6. Наведіть загальну структуру інструкції створення об'єкта класу.
7. Як можна присвоїти значення параметрам методу __init__ у самому методі?

## Завдання для самостійного виконання

1. Створіть програму з конструктором, за допомогою якої із набору «Київ», « — столиця», «України» створюється повідомлення Київ — столиця України.
2. Створіть програму з конструктором і методом обчислення значення виразу (a2 + b2) / 2, якщо значення a і b уводяться з клавіатури.
3. Конструктор має такі значення параметрів: self, a1 = 3, a2 = 4. Створіть програму множення значень параметрів конструктора, за умови, що він викликається — з першого об'єкта без аргументів, з другого об'єкта — з аргументом 5 і 6, а з третього — з аргументом 7.
4. Розробіть програму, у якій конструктор має параметри self, a1 і a2. Він викликається з двох об'єктів, аргументами яких є два цілих числа. Якщо значення першого аргумента більше другого, ці числа множаться, інакше — значення першого ділиться на друге.
5. Конструктор має такі значення параметрів: self, a1 = "Дністер" a2 = "Десна". Розробіть програму виведення всіх можливих результатів на екран, у якій конструктор викликається з першого об'єкта з аргументами Дніпро та Ворскла, з другого об'єкта — з аргументом Дніпро, та з третього об'єкта без аргументів.
6. Конструктор має параметри зі значеннями: self, a1 = "Операційна" a2 = "система". За допомогою інструкції input() уводиться слово. Якщо це слово Linux, виводиться повідомлення Операційна система Linux, інакше — повідомлення Операційна система Windows.

# 6.4. Наслідування

*У різних частинах програмного коду часто використовуються одні й ті самі атрибути, що суттєво впливає на загальний обсяг програми. Як, на вашу думку, можна цього позбавитися?*

**Наслідування** в мові Python — це здатність об'єктів класу застосовувати атрибути цього самого класу, а також здатність одними класами застосовувати атрибути інших класів.

Розглянемо спочатку здатність об'єкта класу наслідувати атрибути цього самого класу.

**Приклад 1.**

На рис. 1 подано програму обчислення площ двох прямокутних трикутників із відомими катетами.

```
class Kl7_10:                          # клас Kl7_10
    def __init__(self , a1, a2,a3):    # конструктор
        self.p1 = a1                   # змінна p1 набуває значення параметра a1
        self.p2 = a2                   # змінна p2 набуває значення параметра a2
        self.p3 = a3                   # змінна p3 набуває значення параметра a3
        self.func1 ()                  # звернення до функції func1
    def func1(self):                   # функція func1 з параметром self
        s = (self.p2*self.p3)/2        # обчислення площі трикутника
        print(self.p1, s)              # виведення площі трикутника
ob1 = Kl7_10("площа першого =", 3, 4) # екземпляр класу ob1 з 3 аргументами
ob2 = Kl7_10("площа другого =", 5, 6) # екземпляр класу ob2 з 3 аргументами
ob1.form = "перший зафарбований"      # властивість екземпляра класу ob1
print(ob1.form)                        # виведення властивості екз. класу ob1
```

Рис. 1. Програма з наслідуванням об'єктами атрибутів одного класу

У програмі обидва екземпляри класу ob1 і ob2 наслідують властивості p1, p2 і p3, а екземпляр класу ob1 наслідує ще й властивість form. Екземпляр класу ob2 цієї властивості не має, оскільки атрибути класу наслідуються об'єктами, створеними лише на його основі. Атрибути конкретного об'єкта не залежать від атрибутів інших об'єктів і мають власний простір імен об'єкта. Це означає, що об'єкти одного класу можуть мати різні значення атрибутів. Результат виконання програми наведено на рис. 2.

```
площа першого = 6.0
площа другого = 15.0
перший зафарбований
```

Рис. 2. Виконання програми з наслідуванням об'єктами атрибутів одного класу

Розглянемо тепер приклад програми з наслідуванням атрибутів одного класу іншим класом. Клас, від якого наслідується інший клас, називають суперкласом, а клас, що унаслідується, — підкласом. В оголошенні підкласу в круглих дужках указуються ім'я суперкласу, від якого він наслідується.

**Приклад 2.**

На рис. 3 зображено програму, у якій клас Kl7_11 є суперкласом, а клас Kl7_11a — підкласом. Клас Kl7_11a має доступ до всіх атрибутів класу Kl7_11.

```
class Kl7_11:                          # суперклас Kl7_11
    def func1(self, a1, a2):           # метод func1 суперкласу
        s = a1 + a2                    # сума чисел
        print("сума =",s)              # виведення суми
    def func2(self, a1, a2):           # метод func2 суперкласу
        s = a1 * a2                    # добуток чисел
        print("добуток =", s)          # виведення добутку
class Kl7_11a(Kl7_11):                 # клас Kl7_11a наслідує клас l7_11
        def func3(self, a1, a2):       # метод func3 підкласу
            s = a1 / a2                # ділення чисел
            print ("ділення =", s)     # виведення результату ділення
ob1 = Kl7_11a()                        # екземпляр підкласу
ob1.func1(2, 3)                        # звернення до методу func1 суперкласу
ob1.func2(4, 5)                        # звернення до методу func2 суперкласу
ob1.func3(6, 7)                        # звернення до методу func3 підкласу
```

Рис. 3. Програма з наслідуванням методів одного класу іншим класом

У цій програмі клас Kl7_11a має доступ до методів func1 і func2 класу Kl7_11, а також до власного методу func3. Результат виконання програми подано на рис. 4.

```
сума =5
добуток = 20
ділення = 0.8571428571428571
```

Рис. 4. Виконання програми з наслідуванням методів одного класу іншим класом

Якщо ім'я методу в суперкласі збігається з іменем методу підкласу, то в процесі звернення до нього буде використовуватися метод із підкласу. Щоб викликати однойменний метод із суперкласу, слід указати також перед цим методом назву суперкласу. У першому параметрі методу необхідно вказати посилання на екземпляр класу.

**Приклад 3.**

Відомі всі три сторони двох прямих контейнерів (паралелепіпедів), а також їх реальна вага. Обчисліть об'єм кожного контейнера, а також можливість їх використання. Контейнери можна використовувати за умови, що їх реальна вага не перевищує контрольну, значення якої уводиться з клавіатури. Програму реалізації завдання зображено на рис. 5.

```
class Kl7_12:                  # суперклас Kl7_12
    contr = int(input("увести контрольну вагу : "))
    def __init__(self, a1, a2, a3):          # конструктор
        self.p1 = a1           # змінна p1 набуває значення параметра a1
        self.p2 = a2           # змінна p2 набуває значення параметра a2
        self.p3 = a3           # змінна p3 набуває значення параметра a3
    def func1(self):           # функція func1 класу Kl7_12
        v = self.p1 * self.p2 * self.p3       # обчислення об'єму
        print("Об'єм", m, " =", v)            # виведення об'єму

class Kl7_12a(Kl7_12):         # підклас kl7_12a суперкласу Kl7_12
    def func2(self, vaga):# функція func2 класу Kl7_12a
        if vaga < Kl7_12.contr:               # реальна вага менше контрольної?
            self.p4 = "можна використовувати" #
        else:                  # вага більше контрольної
            self.p4 = "використовувати не можна"   #
    def func3(self):           # функція func3 класу Kl7_12a
        print(self.p4)         # виведення результату

ob1 = Kl7_12a(2, 3, 4)         # екземпляр ob1 класу Kl7_12a
m = "першого"                  # ознака номера паралелепіпеда
ob1.func1()                    # звернення до функції func1 із екз. кл. ob1
ob2 = Kl7_12(3, 4, 5)          # екземпляр ob2 класу Kl7_12
m = "другого"                  # ознака номеру паралелепіпеда
ob2.func1()                    # звернення до функції func1 із екз. кл. ob2
ob1.func2(5)                   # звернення до функції func2 із екз. кл. ob1
ob1.func3()                    # звернення до функції func3 із екз. кл. ob1
```

Рис. 5. Програма з наслідуванням атрибутів одного класу іншим класом

У програмі використано такі змінні: vaga — реальна вага контейнера, contr — контрольна їх вага, p1, p2, p3 — розміри сторін контейнерів, m — змінна ознаки номера контейнерів, v — об'єм поточного контейнера, p4 — результат можливості використання контейнерів. Програма складається з двох класів: суперкласу Kl7_12 і підкласу Kl7_12a. Підклас наслідує обидва методи суперкласу (__init__ і func1), а також змінну contr. Екземпляр класу ob1 належить підкласу Kl7_12a і наслідує всі методи цього класу та суперкласу. Екземпляр класу ob2 належить суперкласу. До підкласу Kl7_12a він ніякого стосунку не має й тому не може звертатися до методів func2 і func3.

Опишемо порядок виконання програми на прикладі.

**Приклад 4.**

1. Спочатку в суперкласі виконується інструкція contr = int (...). Після уведення цілого числа виконання програми продовжується. Під час створення екземпляра класу ob1 автоматично викликається метод __init__, у результаті чого аргумент 2 передається параметру a1 цього методу, аргумент 3 — параметру a2, аргумент 4 — параметру a3. Наступними трьома інструкціями значення цих параметрів присвоюються, відповідно змінним p1, p2 і p3 екземпляра класу ob1.

2. Далі управління передається інструкції m = «першого» і далі здійснюється звернення до функції func1. У результаті обчислюється й виводиться на екран об'єм першого паралелепіпеда. Після цього створюється екземпляр класу ob2 і виконуються дії, аналогічні діям, що виконувалися в процесі створення екземпляра класу ob1. У результаті буде обчислено й виведено значення об'єму другого контейнера.

3. Далі управління передається інструкції ob1.func2. За допомогою методу func2 перевіряється можливість використання контейнерів і відповідне значення присвоюється змінній p4. Після цього викликається метод func3, за допомогою якого відповідний результат виводиться на екран.

Один із варіантів виконання програми може бути таким (рис. 6).

```
увести контрольну вагу : 8
об'єм першого = 24
об'єм другого = 60
можна використовувати
```

Рис. 6. Виконання програми

## Запитання для перевірки знань

1. Які можливості надає наслідування в мові Python?
2. Які класи називають суперкласами й підкласами?
3. Як оголошується підклас, який унаслідується від суперкласу?
4. Наведіть приклад наслідування атрибутів в одному класі.
5. Як здійснити звернення до однойменних методів у підкласі й суперкласі?
6. Наведіть приклад наслідування одного класу іншим.

## Завдання для самостійного виконання

1. Розробіть програму без конструктора, у якій один клас наслідує атрибути іншого. У суперкласі два числа множаться, а в підкласі виводиться результат множення.
2. Обчисліть площі двох прямокутних трикутників із відомими значеннями катетів із використанням суперкласу, підкласу та конструктора. У суперкласі обчислюються площі трикутників, а в підкласі — виведення значень обчислених їх площ.
3. Конструктор має параметри self, a1, a2, який викликається з двох об'єктів. Перший об'єкт має аргументи «Київ» і «Вінниця», другий — «Львів» і «Харків». Розробіть програму виведення значень параметрів конструктора для обох об'єктів, а також виведення значення першого об'єкта: міста України.
4. Розробіть програму з суперкласом і підкласом. У суперкласі за допомогою одного методу обчислюється сума значень трьох параметрів методу, а за допомогою другого — їх добуток. У підкласі від суми значень перших двох параметрів віднімається значення третього параметра. Звернення до всіх методів виконується з одного об'єкта з трьома аргументами.
5. Червона куля має радіус r1, а жовта — r2. Розробіть програму з використанням принципу наслідування, за допомогою якої визначається різниця об'ємів між червоною і жовтою кулею.
6. Знайдіть в Інтернеті обласні центри України, розташовані на широтах між 49° і 48°. Розробіть програму для визначення різниці в широтах між Києвом і цими центрами. Для обчислення й виведення різниці використайте суперклас, підклас і конструктор.

# 7. Поліформізм, перевизначення методів, модулі користувача

## 7.1. Поліморфізм

*Пригадайте, із якими принципами об'єктно-орієнтованого програмування ви вже знайомі. Поясніть їх сутність.*

Поліморфізм (від грец. πολύς — багато, μορφή — форма) означає "багато форм". Кожна мова ООП має свої особливості поліморфізму.

Розглянемо основні прояви поліморфізму в мові Python.

- **Тип об'єктів визначає синтаксичний смисл оператора, який виконується над об'єктами.**

Оператор + є лише вказівкою для об'єктів, а яка буде виконуватися над об'єктами операція, залежить від типу самих об'єктів (приклад 1).

**Приклад 1.**
Якщо операція додавання виконується над двома об'єктами типу int, то буде виконуватися операція їх додавання, а якщо об'єкти мають тип str — то операція об'єднання рядків.

**Приклад 2.**
На рис. 1 зображено програму, у якій демонструється виконання операторів додавання та множення над об'єктами типу int і str.

```
class Kl8_01:                          # клас Kl8_01
    a = 4                              # змінна a набуває ціле число
    b = 5                              # змінна b набуває ціле число
    print("результат = ", a + b)       # результат = 9
    a = "4"                            # змінна a набуває значення рядка
    b = "5"                            # змінна b набуває значення рядка
    print("результат = ", a + b)       # результат = 45
    a = 7                              # змінна a набуває ціле число
    b = 8                              # змінна b набуває ціле число
    print("результат = ", a * b)       # результат = 56
    b = "8"                            # змінна b набуває значення рядка
    print("результат = ", a * b)       # результат = 8888888
```

Рис. 1. Програма з різними діями над об'єктами операції додавання та множення

Як бачимо, в останньому рядку програми об'єкт b, який має тип str, повторюється 7 разів, тобто стільки разів, які має об'єкт a типу int.

- **Методи з однаковими іменами в різних класах можуть виконувати різні дії.**

**Приклад 3.**
На рис. 2 зображено програму, у якій метод func класу Kl8_02a обчислює корінь квадратний із суми квадратів двох чисел, а метод із таким самим іменем класу Kl8_02b обчислює середнє значення суми трьох чисел.

```
class Kl8_02a:                          # клас Kl_02a
    def func(self, a1, a2):             # функція func класу Kl8_02a
# метод func обчислення кореня квадратного суми квадратів двох чисел
        self.func = int(math.sqrt(a1 * a1 + a2 * a2))
class Kl8_02b:                          # клас Kl8_12b
    def func(self, a1, a2, a3):         # функція func класу Kl8_02b
# метод func обчислення середнього значення трьох чисел
        self.func = int((a1 + a2 + a3) / 3)
import math          # імпортування модуля math
ob1 = Kl8_02a()      # екземпляр класу Kl8_02a
ob2 = Kl8_02b()      # екземпляр класу Kl8_02b
ob1.func(3, 4)       # звернення до методу func об'єкта ob1
ob2.func(6, 8, 13)   # звернення до методу func об'єкта ob2
print(ob1.func)      # виведення результату викон. методу func об'єкту ob1
print(ob2.func)      # виведення результату викон. методу func об'єкту ob2
```

Рис. 2. Програма з однойменними методами у двох класах

У програмі створюється екземпляр ob1 класу Kl8_02a і екземпляр ob2 класу Kl8_02b.

Метод func класу Kl8_02a викликається з об'єкта ob1 і аргумент 3 передається параметру a1, а аргумент 4 — параметру a2 цього класу. Далі в цьому методі обчислюється корінь квадратний суми квадратів цих чисел.

Метод func класу Kl8_02b викликається з об'єкта ob2 і аргумент 6 передається параметру a1, аргумент 8 — параметру a2, а аргумент 13 — параметру a3. Далі в цьому методі обчислюється середнє значення суми трьох чисел.

- **За допомогою оператора if можна створити один і той самий об'єкт одного або іншого класу, у яких використовуються методи з однаковими іменами, але які виконують різні дії.**

**Приклад 4.**

На рис. 3 зображено програму з двома класами, у кожного з яких є метод __init__. У методі класу Kl8_03a обчислюється сума введених чисел, а в методі класу Kl8_03b — їх добуток.

```
class Kl8_03a:                          # клас Kl8_03a
    def __init__(self, a1, a2):         # конструктор класу Kl8_03a
        self.s = a1 + a2                # обчислення суми чисел
class Kl8_03b:                          # клас Kl8_03b
    def __init__(self, a1, a2):         # конструктор класу Kl8_03b
        self.s = a1 * a2                # обчислення добутку чисел
b1 = int(input("увести число b1 "))     # введення числа b1
b2 = int(input("увести число b2 "))     # введення числа b
if b1 < b2:                             # якщо b1 менше b2
    ob1 = Kl8_03b(b1, b2)               # екземпляр ob1 класу Kl8_03b
else:                                   # b1 не менше b2
    ob1 = Kl8_03a(b1, b2)               # екземпляр ob1 класу Kl8_03a
print(ob1.s)                            # виведення результату
```

Рис. 3. Програма з конструкторами в різних класах

Якщо перше уведене число (b1) менше другого уведеного числа (b2), то створюється екземпляр ob1 класу Kl8_03b, у результаті чого здійснюється звернення до методу цього самого класу та обчислюється добуток уведених чисел, інакше створюється екземпляр ob1 класу Kl8_03, а потім обчислюється й виводиться сума введених чисел. Зауважимо, що в процесі звернення до методів __init__ обох класів, аргумент b1 передається параметру a1, а аргумент b2 — параметру a2.

Отже, у наведеному прикладі поліморфізм проявляється в тому, що за допомогою оператора if створюється об'єкт ob1 класу Kl8_03b або класу Kl8_03, а один і той самий метод __init__ у двох різних класах виконує різні дії над уведеними числами.

- **Перезавантаження операторів**

Сутність перезавантаження операторів полягає в тому, що за допомогою спеціальних методів, які інколи називають магічними, одні й ті самі оператори виконують різні дії над об'єктами. Ці методи, так само, як і метод __init__, на початку й наприкінці мають два знаки підкреслення. За аналогією з методом __init__, вони викликаються автоматично, коли екземпляр об'єкта бере участь у відповідній операції, наприклад, у операції додавання, множення й ін.

У мові Python існує величезна кількість магічних методів, наведемо лише деякі з них:

| Метод | Опис |
|---|---|
| __add__ | Викликається автоматично, коли екземпляр класу бере участь в операції додавання (+) |
| __sub__ | Викликається автоматично, коли екземпляр класу бере участь в операції віднімання (–) |
| __mul__ | Викликається автоматично, коли екземпляр класу бере участь в операції множення (*) |
| __truediv__ | Викликається автоматично, коли екземпляр класу бере участь в операції ділення (/) |
| __str__ | Викликається, коли об'єкт перетворюється в рядок для виведення викликом убудованої функції str |

Перш ніж розглянути приклад програми з перезавантаженням операторів, опишемо призначення операторів % і %s.

Оператор % (а також метод format) дає змогу вставити в рядок дані, отримані в процесі виконання програми.

Якщо необхідно вставити дані, які мають сприйматися звичайним рядком, використовується комбінація символів %s.

Сутність цих операторів пояснюється таким прикладом:

```
>>>'Наталя %s Олега' % 'подруга'
Наталя подруга Олега
```

Із прикладу видно, що після оператора % вказується, що потрібно вставити (це може бути результат, отриманий у процесі виконання програми), а комбінація символів %s — куди необхідно вставити, у цьому випадку символ s свідчить про те, що результат, який вставляється, сприймається як рядок.

Френсіс Елізабет Аллен — американська вчена, піонерка в галузі оптимізації компіляторів, стала перша жінка, яка отримала премію Тюрінга.

**Приклад 5.**
На рис. 4 наведено приклад програми з використанням методів __add__ і __str__ .

```
class Kl8_04c:                  # клас Kl8_04c
  def __init__(self, a1):       # конструктор класу Kl8_04c
      self.p1 = a1              # змінна p1 набуває значення параметра a1
  def __add__(self, a2):        # перезавантаження оператора +
      self.p1 = self.p1 + a2    # додавання або з'єднання значень
  def __str__ (self):           # перетворення об'єкта в рядок
      return '%s' % self.p1     # повернення звичайного рядка
ob1 = Kl8_04c(21)               # екземпляр ob1 класу Kl8_04c
ob1 + 15                        # до значення екземпляра ob1 додається 15
print(ob1)                      # виведення значення екземпляра ob1
ob2 = Kl8_04c("Пере")           # екземпляр ob2 класу Kl8_04c
ob2 + "завантаження"            # до значення екз. ob2 приєднується завантаження
print(ob2)                      # виведення значення екземпляра ob2
ob3 = Kl8_04c([19, 7, 12])      # екземпляр ob3 класу Kl8_04c
ob3 + [5, 9]                    # до значення екземпляра ob3 приєднується [5, 9]
print(ob3)                      # виведення значення екземпляра ob3
```

Рис. 4. Код програми з методами __add__ і __str__

Опишемо порядок виконання програми.

1. Після запуску програми створюється екземпляр класу ob1. Викликається конструктор, і параметр a1 набуває значення 21; це значення присвоюється змінній p1 об'єкта ob1.

Далі виконується інструкція ob1+15. Операція додавання (+) перехоплюється методом __add__ і за допомогою наступної операції виконується звичайна операція додавання двох чисел (числа 21 до числа 15).

2. Далі управління передається інструкції print (ob1), здійснюється перехоплення методом __str__, число 36 перетворюється у звичайний рядок і за допомогою інструкції print (ob1) виводиться на екран.

3. Створюється екземпляр класу ob2. Виконуються дії у тому самому порядку, як описано в пункті 1, але для значень 'Пере' і 'завантаження'.

4. Створюється екземпляр класу ob3. Виконуються такі самі дії над списками [19, 7, 12] і [5, 9].

Результат виконання програми наведено на рис. 5.

```
36
Перезавантаження
[19, 7, 12, 5, 9]
```

Рис. 5. Виконання програми з методами __add__ і __str__

Перезавантаження операторів є потужним засобом програмування, але ним не слід зловживати. Якщо є можливість обійтися без перезавантаження операторів, то краще його не використовувати.

## Запитання для перевірки знань

1. Як проявляється поліморфізм у мові Python?
2. Від чого залежить сутність виконання оператора множення?
3. Чи можуть в одній програмі використовуватися методи з однаковими іменами?
4. Як за допомогою оператора if можуть створюватися об'єкти з одним іменем?
5. Яке призначення має метод __str__?
6. Поясніть сутність перезавантаження методів.
7. Які дії виконують оператори % і %s?

## Завдання для самостійного виконання

1. Розробіть програму, у якій виконується оператор (+) над даними: 21.5, 4, 37 ; «Київ», «–21», а також оператор (*) над даними: 48, 5; «7», «Україна».
2. Розробіть програму з використанням методу __add__ додавання чисел 143.5 і 32.4, а також об'єднання рядків «ай» і «фон».
3. Дано масив чисел a[1], a[2], a[3], a[4], a[5]. Якщо перше число більше останнього, то в першому класі обчислюється добуток чисел масиву, інакше — в другому класі — їх сума.
4. Розробіть програму з двома класами, у першому з яких за допомогою методу func1 обчислюється сума чисел одновимірного масиву, а в другому класі за допомогою методу func1 — їх добуток.
5. Формується масив чотирма випадковими цілими числами в діапазоні від 7 до 14. Якщо друге число більше третього, то обчислюється сума всіх чисел масиву, інакше — в другому класі —їх добуток.

# 7.2. Перевизначення та розширення можливостей методів

*Значну кількість об'єктів побудовано за ієрархічною структурою. В ООП методи теж можуть мати ієрархічну структуру. Для звернення до конкретного методу в цій структурі необхідно мати відповідні правила. Як, на вашу думку, можна організувати звернення до методу в такій ієрархії?*

Поліморфізм і наслідування класів дають змогу реалізувати перевизначення методів.

Сутність перевизначення методів полягає в тому, що в ієрархічній структурі класів, якщо підклас не містить методу, до якого виконується звернення із певного об'єкта, то здійснюється пошук такого методу в найближчому зверху батьківському класі. Якщо й у ньому відсутній такий метод, то він відшукується також у найближчому до нього зверху батьківському класі.

Принцип перевизначення пояснюється схемою (рис. 1). Як бачимо, в процесі звернення до методу_3 з підкласу_2 цей метод виконується одразу. Аналогічно виконується й метод_2 у процесі звернення до нього з підкласу_1. У процесі звернення до методу_1 із підкласу_1 цей метод відшукується в самому підкласі, а потім виконується звернення до суперкласу, де й виконується цей метод.

Нарешті, у процесі звернення до методу_4 з підкласу_2 метод відшукується спочатку в цьому підкласі, потім у підкласі_1 і виконується в суперкласі.

Метод виконується в тому класі, у першому з яких буде знайдено цей метод. Якщо в жодному батьківському класі такого методу не буде знайдено, то буде видано повідомлення про синтаксичну помилку.

Пошук методу виконується за принципом «знизу догори».

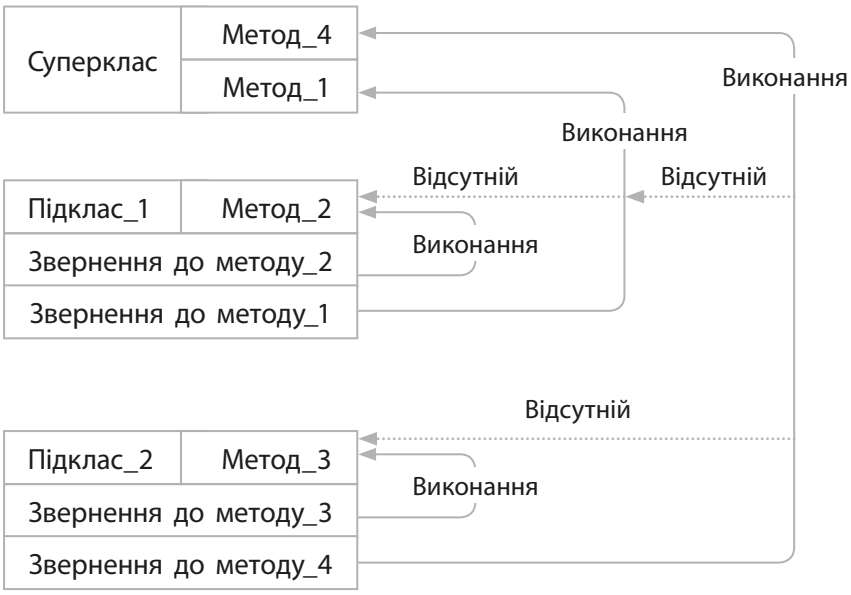


Рис. 1. Варіант перевизначення методу

**Приклад 1.**

На рис. 2 зображено програму з перевизначенням методу. У програмі є один суперклас і два підкласи. У суперкласі використовуються методи __init__ і func1, у підкласі Kl8_04a один метод func2 і в підкласі Kl8_04b методи func3 і func1.

```
class Kl8_04:                       # супеклас Kl8_04
    def __init__(self, a1):         # конструктор суперкласу Kl8_04
        self.p = a1                 # змінна p набуває значення параметра ф1
    def func1(self):                # метод func1 суперкласу Kl8_04
        print("у класі Kl8_04 =", self.p - 13 )      # виведення значення зміної p

class Kl8_04a(Kl8_04):              # підклас Kl8_04a суперкласу Kl8_04
    def func2(self, a2):            # метод func2 підкласу Kl8_04a
        self.p *= a2                # еквівалентно self.p=self.p*a2
        print("у підкласі Kl8_04a =",self.p)         # виведення значення змінної p

class Kl8_04b(Kl8_04):              # підклас Kl8_04b суперкласу Kl8_04
    def func3(self):                # метод func3 підкласу Kl8_04b
        self.func3 = list(str(self.p))              # перетворення рядка у список
        print("список: =", self.func3)               # виведення списку рядку 'байт'
    def func1(self):                # метод func1 підкласу Kl8_04b
        i = 0                       # початкове значення змінної
        while i < len(self.func3):                   # доти, доки і менше довжини списку
            print(i, "елемент: ", self.func3[i])     # виведення букв у стовпець
            i += 1                                   # еквівалентно i = i+1

ob1 = Kl8_04a(37)                   # екземпляр ob1 підкласу Kl8_04a
ob2 = Kl8_04b('байт')               # екземпляр ob2 підкласу Kl8_04b
ob1.func2(3)                        # звернення до методу func2 із об'єкта ob1
ob1.func1()                         # звернення до методу func1 із об'єкта ob1
ob2.func3()                         # звернення до методу func3 із об'єкта ob2
ob2.func1()                         # звернення до методу func1 із об'єкта ob2
```

Рис. 2. Програма з перевизначенням методу

Опишемо порядок виконання програми.

1. У процесі створення об'єкта ob1 класу Kl8_04a автоматично викликається метод __init__ і аргумент 37 передається параметру a1 цього методу. За допомогою наступної інструкції self.p = a1 змінна p об'єкта ob1 набуває значення 37. А в процесі створення об'єкта ob2 підкласу Kl8_04b змінна p об'єкта ob2 набуває значення рядка 'байт'.

2. Далі здійснюється звернення до методу func2 з об'єкта ob1 підкласу Kl8_04a. Оскільки цей метод є в цьому підкласі, він одразу виконується. У результаті чого параметру a2 передається значення аргумента, тобто число 3. Наступною командою в цьому підкласі значення змінної p об'єкта ob1 (це значення дорівнює 37) множиться на 3 і на екран виводиться результат 111.

3. Після цього здійснюється звернення до методу func1 з об'єкта ob1 підкласу Kl8_04a. Але такого методу в цьому підкласі немає. Тому здійснюється пошук такого методу за принципом «знизу догори». Такий метод є в суперкласі Kl8_04, який є батьківським підкласу Kl8_08a. Цей метод і буде виконуватися. У результаті від значення p (тобто від 111) віднімається число 13 і виводиться на екран. Отже, виконалося перевизначення методу func1.

4. Далі з об'єкта ob2 підкласу Kl8_04b викликається метод func3. Він є в цьому підкласі, тому одразу й виконується. У результаті змінна p об'єкта ob2, яка має значення рядка 'байт', перетворюється на список і виводиться на екран.

5. Останньою командою програми викликається метод func1 з об'єкта ob2 підкласу Kl8_04b. Цей метод також є в підкласі Kl8_04b і він також одразу виконується. У результаті в стовпець виводяться елементи списку ['б','а','й','т'].

Результат виконання програми наведено на рис. 3. Таким чином, у програмі здійснилося перевизначення методу func1.

```
у підкласі Kl8_04a = 104
у класі Kl8_04 = 98
список: = ['б', 'а', 'й', 'т'
0 елемент: б
1 елемент: а
2 елемент: й
3 елемент: т
```

Рис. 3. Виконання програми з перевизначенням методу func1

Розглянемо механізм розширення можливостей методів. До цього вдаються, коли виникає потреба звернутися від методу одного класу до методу іншого. Наприклад, із методу підкласу потрібно звернутися до методу суперкласу для розширення можливостей підкласу.

**Приклад 2.**

У першому риболовецькому господарстві виловили 10 т риби, у другому — 7 т. До кожного з них може надійти замовлення на постачання риби від 1 до 6 магазинів. Риба між замовниками розподіляється порівну. Визначте, скільки тонн риби дістанеться кожному замовнику для таких варіантів: до першого господарства може надійти замовлення від 1 або 2 магазинів; до другого господарства — від 3 або 4 магазинів; до першого господарства — від 5 або 6.

Можна виконати операцію ділення в такому порядку: поділити число 10 на 1 і на 2, потім число 7 на 3 і на 4, потім число 10 на 5 і на 6. Програму реалізації завдання подано на рис. 4.

```
class Kl8_06:                          # суперклас Kl8_06
  def __init__(self, a1, a2):          # конструктор суперкласу Kl8_06
# змінні p1 і p2 об'єкта ob1 набувають значень параметрів a1 і a2
      self.p1 = a1
      self.p2 = a2
  def func1(self):                     # метод func1 суперкласу Kl8_06
      self.p1 =self.p1 / self.p2       # ділення значення p1 на p2
      print(self.p1)                   # виведення значення змінної p1

class Kl8_06a(Kl8_06):                 # підклас Kl8_06a суперкласу Kl8_06
  def func1(self):                     # метод func1 підкласу Kl8_06a
      print("перехід 1")               # виведення повідомлення
      Kl8_06.func1(self)               # звернення до методу func1 суперкласу
      print("перехід 2")               # виведення повідомлення

i = 1                                  # початкове значення змінної
while i < 7:                           # доти, доки і менше 7
  if  2 < i and i < 5:                 # умова
      ob1 = Kl8_06a(7 , i)             # об'єкт ob1 підкласу Kl8_06a
  else:                                # інакше
      ob1 = Kl8_06(10, i)              # об'єкт ob1 суперкласу Kl8_06
  i += 1                               # еквівалентно i = i + 1
  ob1.func1()                          # звернення до методу func1 із об'єкта ob1
```

Рис. 4. Код програми з розширенням можливостей методу

Опишемо порядок виконання програми. Зазначимо, що, запустившись, програма починає виконуватися з інструкції i=1.

**Приклад 3.**

**I цикл.** Оскільки результатом оператора if є значення False, здійснюється перехід на гілку else. У результаті створюється екземпляр класу ob1 суперкласу Kl8_06. Автоматично викликається метод __init__ , параметр a1 набуває значення 10, а параметр a2 — значення 1. Значення цих параметрів за допомогою наступних двох інструкцій цього методу присвоюються, відповідно, змінним p1 і p2.

Далі управління передається інструкції i += 1 основної програми. Змінна i набуває значення 2, а за допомогою інструкції ob1.func1() викликається з об'єкта ob1 метод func1 суперкласу і значення змінної p1 ділиться на значення змінної p2 (число 10 ділиться на 1) і результат (10) виводиться на екран. Після цього управління передається оператору циклу.

**II цикл.** Починається з перевірки умови (2<i) and (i<5), яке також має значення False. Так само реалізується гілка else і так само, як і в попередньому випадку, викликається метод __init__, у результаті чого параметр a1 набуде значення 10, а параметр a2 — значення 2. Відповідно, змінні p1 і p2 набудуть значень 10 і 2. Знову виконується інструкція i += 1 і змінна i набуде значення 3, потім здійснюється звернення до методу func1 суперкласу, у результаті чого буде виведено число 5 (у результаті ділення: 10/2=5). Як і в попередньому випадку, далі управління передається оператору циклу.

**III цикл.** Нагадаємо, що на цей момент змінна i має значення 3, тому результат оператора if має значення True і створюється екземпляр ob1 підкласу Kl8_06a. Так само автоматично викликається метод __init__, у результаті чого змінна p1 набуде значення 7, а змінна p2 — значення 3. Потім виконується інструкція i =+ 1 і змінна i набуде значення 4. Далі викликається метод func1, але цього разу з підкласу. Тому виводиться повідомлення 'перехід 1' і за допомогою інструкції Kl8_06.func1 (self) цього методу викликається метод func1 із суперкласу, де значення змінної p1 (нагадаємо, воно дорівнює 7) ділиться на значення змінної p2 (тобто на 3) і результат (2.33) виводиться на екран. Після цього управління передається інструкції print ("перехід 2") і буде виведено перехід 2. Після цього управління знову передається оператору циклу.

**IV цикл.** Змінна i на цей момент має значення 4, тому виробляється значення True і створюється екземпляр ob1 підкласу Kl8_06a. Викликається метод __init__, у результаті чого змінна p1 набуде значення 7, а змінна p2 — значення 4. Потім виконується інструкція i += 1, змінна i набуває значення 5 і викликається метод func1 підкласу. У результаті друкується перехід 1 і за допомогою наступної команди викликається метод func1 суперкласу, у якому число 7 ділиться на число 4 і результат 1.75 виводиться на екран. Потім управління передається інструкції print ("перехід 2") і перехід 2 виводиться на екран. Далі управління знову передається оператору циклу.

**V цикл.** На цей момент змінна i має значення 5. У результаті перевірки умови if 2<i<5, як і в перших двох циклах, виробляється значення False і створюється екземпляр ob1 об'єкта суперкласу. Викликається метод __init__, у результаті чого змінна p1 буде мати значення 10, а змінна p2 — значення 5. Далі виконується оператор i += 1, змінна i набуває значення 6, викликається метод func1 суперкласу і значення змінної p1 ділиться на p2. Результат 2.0 виводиться на екран. Далі управління передається оператору циклу.

**VI цикл.** Змінна i має значення 6. Цикл виконується аналогічно до попереднього. Буде виведено значення 1.66. Під час виконання циклу змінна i матиме значення 7, тому оператор while завершує своє виконання.

Результат виконання програми подано на рис. 5.

```
10.0
5.0
перехід 1
2.3333333333333335
перехід 2
перехід 1
1.75
перехід 1
2.0
1.6666666666666667
```

Рис. 5. Виконання програми з розширенням можливостей методу

## Запитання для перевірки знань

1. Поясніть, у чому полягає сутність перевизначення методів.
2. У чому полягає сутність розширення можливостей методу?
3. Накресліть схему класів із прикладом перевизначення методів.
4. Наведіть приклад реалізації розширення можливостей методу.

## Завдання для самостійного виконання

1. Дано числа 16 і 7. Розробіть програму визначення їх суми, добутку, різниці та ділення першого на друге з використанням перевизначення методу. У програмі передбачте створення суперкласу і двох його підкласів, а також чотирьох методів у різних класах.
2. Дано два рядки «екземпляр» і «підкласу». Розробіть програму об'єднання цих рядків, перетворення першого рядка в список і вилучення з другого рядка останньої букви. У програмі передбачте перевизначення методу з використанням суперкласу і його підкласу.
3. У суперкласі та його підкласі використовуються два методи з однаковими іменами. У суперкласі за допомогою методу обчислюється середнє значення суми трьох чисел, а в підкласі здійснюється звернення до методу суперкласу. Розробіть програму реалізації цієї ситуації.
4. Генеруються два цілі випадкові числа в діапазоні від 7 до 15. Якщо їх сума більше 17, числа складаються, інакше множаться. Розробіть програму реалізації завдання з використання суперкласу та двох підкласів.
5. Модифікуйте програму так, щоб у ній використовувався її суперклас і підклас Kl8_06b, зберігся принцип перевизначення методу і програма повинна видавати той самий результат, що й початкова програма.

# 7.3. Композиційний підхід в ООП мовою Python

*Раніше нам уже багаторазово доводилося створювати програмний код з окремих функцій, методів і модулів. Ще один варіант створення цілісного програмного коду реалізується на основі композиційного підходу.*

Сутність композиційного підходу в програмуванні мовою Python полягає в тому, що в класі, який називається **класом-контейнером**, створюються інструкції виклику інших класів. Такий підхід дає змогу на основі створених об'єктів класу-контейнера отримати і об'єкти класів, що викликаються з класу-контейнера.

Композицію інколи ще називають **агрегуванням**.

**Композиція в програмуванні** — це вбудовування інших об'єктів у об'єкт-контейнер і використання їх для реалізації методів класу-контейнера.
На практиці композиція забезпечує формування цілого фрагмента коду з окремих частин.

Розглянемо сутність композиції на прикладі.

**Приклад.**

На аркуші паперу з відомими розмірами накреслено два кола радіусом r1, два кола радіусом r2 та один рівносторонній трикутник зі стороною a. Усі ці фігури не перетинаються одна з одною. Необхідно обчислити загальну площу аркуша паперу й залишок площі аркуша. Зрозуміло, що для обчислення залишку площі аркуша потрібно обчислити площу кожного кола та площу трикутника. Обчислення цих площ фактично і є тими частинами, на основі яких можна обчислити залишок площі.

Програму для розв'язування цієї задачі можна розробити різними способами. Можна обійтися і без композиційного підходу, але ми розробимо програму саме на його основі.

У нашій задачі фактично є три об'єкти: об'єкт-аркуш, об'єкт-коло і об'єкт-трикутник. Відповідно, в одному класі (класі-контейнері) будемо оперувати з площею аркуша, у другому — із площею кола й у третьому — із площею трикутника.

Площу кожного кола обчислимо в класі Kl8_07a, а площу трикутника — у класі Kl8_07b.

Звернення до класу Kl8_07a для обчислення площі першого кола виконаємо з об'єкта p1 класу Kl8_07a, який є об'єктом класу Kl7_07 (має позначення self), а звернення до цього самого класу для обчислення площі другого кола — з об'єкта p2.

Звернення до класу Kl8_07b для обчислення площі трикутника виконаємо з об'єкта p3 класу Kl8_07b, який також є об'єктом класу Kl8_07 і також має позначення self.

Обчислення залишку площі виконаємо за допомогою методу func2, а виведення результатів — за допомогою методу func3.

З урахуванням викладеного програму реалізації завдання зображено на рис 1.

Звернемо увагу на те, що в методі func1 створюються три об'єкти: p1, p2 і p3, а також атрибути k1 і k2, у яких міститься кількість кіл, відповідно, з радіусами r1 і r2.

Також звернемо увагу на те, як виконується в методі func2 звернення до обчислених площ кіл і трикутника: указується об'єкт класу Kl8_07, який у класі його замінює self, далі — об'єкт класу Kl8_07a або класу Kl8_07b (мають позначення p1, p2, p3) і потім сам атрибут (plcolo або pltrik).