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 фахівців цієї галузі. Ймовірно, 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, *автори*

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

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

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.

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 у сучасному розумінні фактично почали розвиватися з появою електронних обчислювальних машин.

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. Всі мови програмування можна умовно класифікувати за деякими *основними ознаками* (рис. 1).

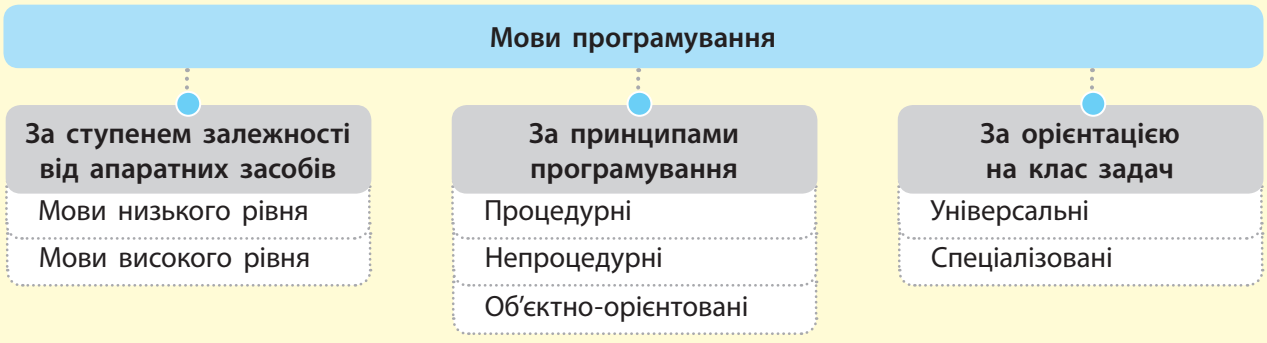

Рис. 1. Класифікація мов програмування

- 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 і відсутністю світлового променю 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 і означати, наприклад, операцію Додати, а B7 і C5 — адресною частиною, яка визначає місце збереження даних, над якими слід виконати операцію.

Вже 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 декількох машинних команд. До різновидів мов символьного кодування належать мови асемблера та 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.

Першою мовою високого рівня, яка набула широкого визнання серед програмістів світу, була **Fortran**. Її було розроблено корпорацією IBM (США) у 1954 році. Мова Фортран 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.

- 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 та 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 та ін.

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. Сьогодні мови ООП практично витіснили з ринку професійного програмування процедурні мови.

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

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

Спеціалізовані мови враховують специфіку предметної галузі. На цей час існують десятки спеціалізованих мов програмування, наприклад, мови веб-програмування, мови скриптів та ін. *Мова скриптів* використовується для створення невеликих допоміжних програм, мова Javascript — для створення динамічних об'єктів на веб-сторінках. *Мови розмітки* містять шаблони і засоби опису вмісту, структури і 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 перерахованих мов у класичному розумінні є мовами програмування. Так, мова HTML є мовою розмітки гіпертексту, але її також часто називають мовою програмування.

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 в 1961 році та 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, з яких утворюються команди програми та інші конструкції мови.

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

Синтаксис

Сукупність правил запису команд та інших конструкцій мови.

Порушення правил синтаксису виявляється автоматично, про що програміст отримує повідомлення.

Семантика

Сукупність правил тлумачення і 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 і 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).

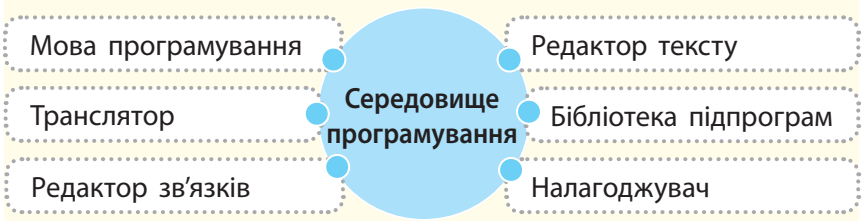

Рис. 1. Складові середовища програмування

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 введення, здійснюється перетворення програми з мови програмування високого рівня у двійковий код. Таке перетворення здійснюється за допомогою транслятора програм. Розрізняють два типи трансляторів: *компілятори* та *інтерпретатори*. У процесі *інтерпретації* з початкового програмного коду послідовно кожна команда (інструкція) перетворюється у двійковий код і віддразу виконується, — на екрані висвітлюється результат її виконання. Після завершення виконання однієї команди виконується наступна і так далі 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, бо 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).

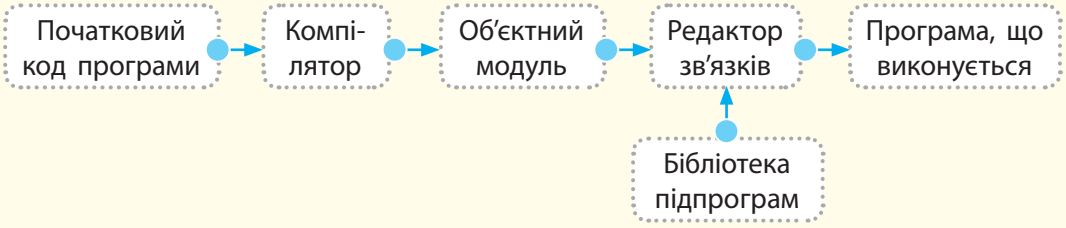

Рис. 2. Процес опрацювання початкового коду програми

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 і 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 і 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 у 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 обумовлений тим, що в Інтернеті вільно переміщуються дані і програми (*аплети* — невеликі програми, призначені для передавання через Інтернет і виконання

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

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

Спочатку виникла потреба у 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 у мові, яку можна було б використовувати на будь-якому типі процесора. Це було обумовлено 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 і виконання аплета на комп'ютерах із будь-яким типом процесора, будь-якою операційною системою і браузером, що під'єднано 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. Якщо б 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 у 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.

Сьогодні все частіше використовуються інтегровані середовища програмування, які забезпечують роботу з декількома 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 у 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 і недоліки мають компілятори й інтерпретатори програм?
6. Для чого призначено редактор зв'язку?
7. Які основні дії виконуються у 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 і бібліотеками інших користувачів. Розширення NumPy містить реалізацію різноманітних математичних обчислень, модуль tkinter дозволяє 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 ході 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 існуючі у 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).

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 і 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 і визначає їх тип на основі їх зовнішнього вигляду. Автоматично звільнюється пам'ять для тих даних, які стають непотрібними.

* Мова програмування Python розповсюджується вільно за посиланням: http://python.org

На рис. 1 подано варіант структури проекту мовою Python.

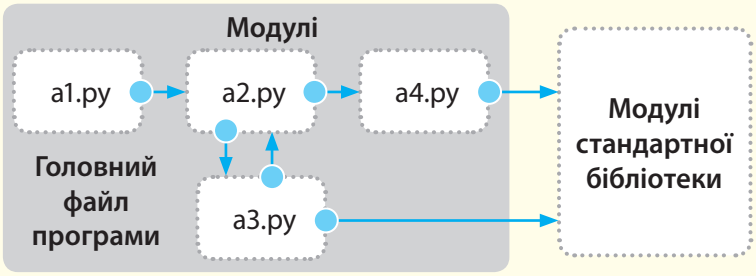

Рис. 1. Варіант структури проекту мовою Python

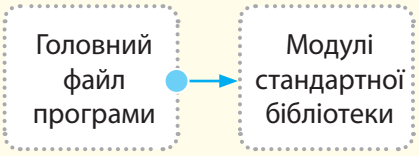

Рис. 2. Найпростіша структура програми мовою Python

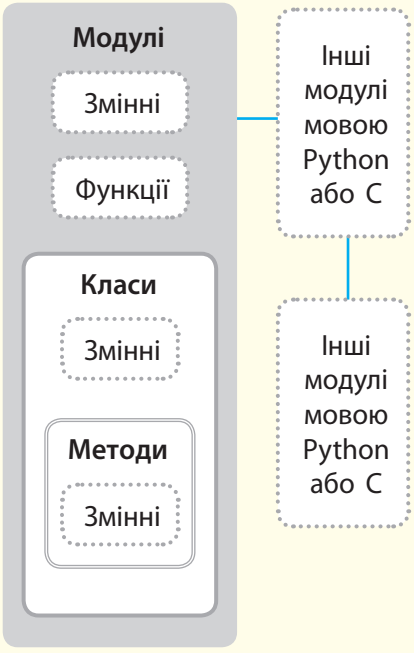

Рис. 3. Структура проекту модуля мовою Python

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, з 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 у 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 у 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 у головний файл програми. Реально програмний модуль може складатися не тільки з інструкцій мови Python, але й із 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 та 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 \Pyhhon34\python.

  Відкриється вікно, яке зображено на рис. 1.

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

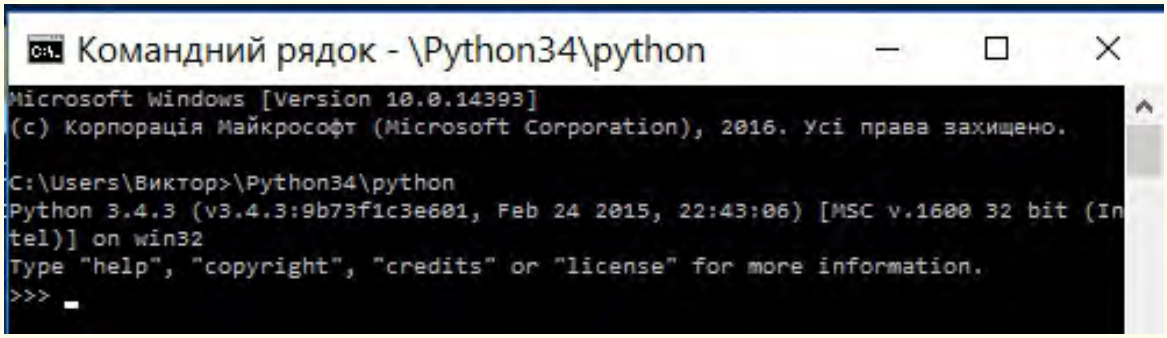

Рис. 1. Вікно інтерпретатора, яке відкрито за допомогою командного рядка

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

  Пуск → Усі програми → Python3.4 → IDLE (Python3.4GUI – 32 біт).

  Відкриється вікно, яке зображено на рис. 2.

  Саме цей спосіб запуску застосовуватиметься нами далі.

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

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

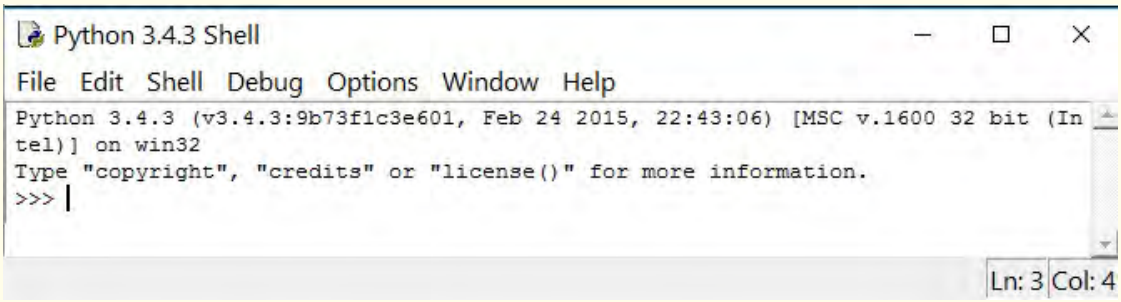

Рис. 2. Вікно інтерпретатора, яке відкрито у середовищі IDLE

Приклад. Припустимо, що програма повинна містити повідомлення про стилі програмування та побажання успіхів. Цю програму можна подати у 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 у круглих дужках у подвійних лапках.

Хід виконання

1. Для введення програми запускаємо інтерпретатор мови Python середовища IDLE.
2. Одразу після знаку запрошення (>>>) уводимо перший рядок програми (#Починаємо вивчати мову Python) і натискаємо клавішу Enter.
3. Знак запрошення переміститься на наступний рядок, 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 і виконання програми в інтерактивному режимі зображено на рис. 3.

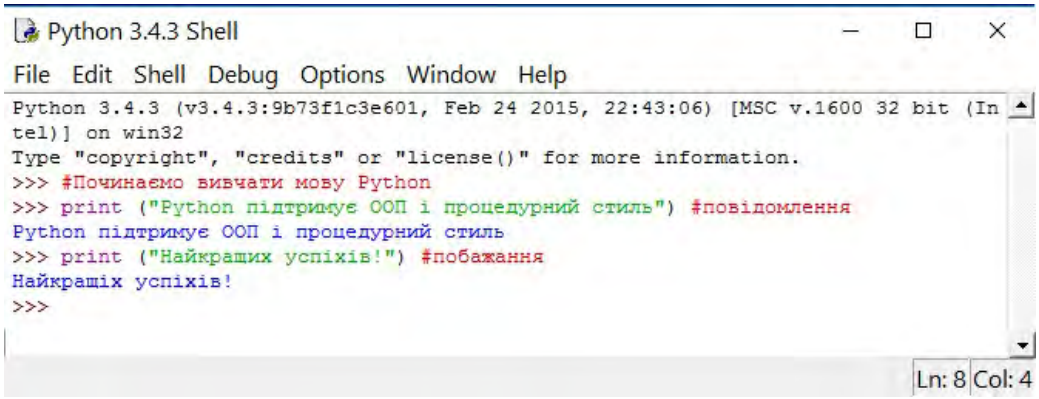

Рис. 3. Перша програма мовою Python (результат роботи)

Після знаку запрошення (>>>) не повинно бути пробілів, 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 декількох 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 блоку мовою Python відокремлюється від початку рядка 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 2 чи 2,5 роки з'являється нова версія мови.

python

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 декілька 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

- З 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 відзначалося, 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 повинно містити розширення ру.

Гарний токар працює у декілька разів краще від середнього, але гарний програміст коштує у 1000 разів більше від звичайного.

Білл Гейтс

Файл програмного коду інтерпретатор може виконувати необмежену кількість разів.

Після запуску коду на виконання інтерпретатор виконує послідовно одну інструкцію за іншою у порядку їх розташування і видає результат на екран монітора. Якщо у 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, що виконує складання чисел 65 і 279 та множення числа 21 на число 15.

1. Відкриваємо вікно Python 3.4.3 Shell і виконуємо команду File → New File.

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

Останньою у 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.

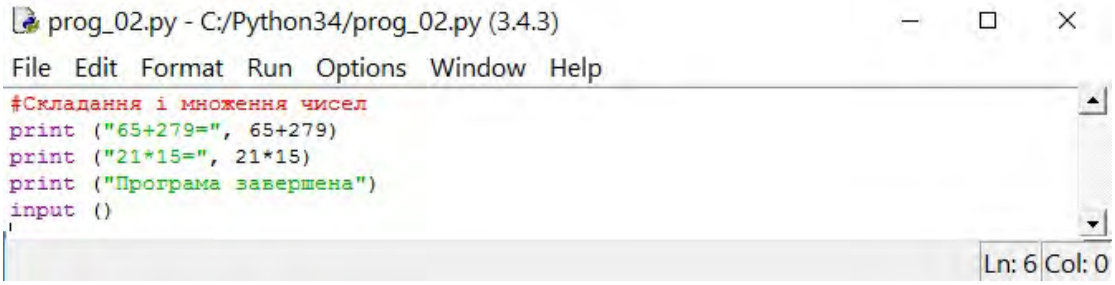

Рис. 1. Програма додавання і множення двох чисел

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 Python34 диску C:. Далі усі файли програм зберігатимемо саме в ній. Інакше необхідно змінити ім'я папки і місце її розташування.

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

Файл буде збережено.

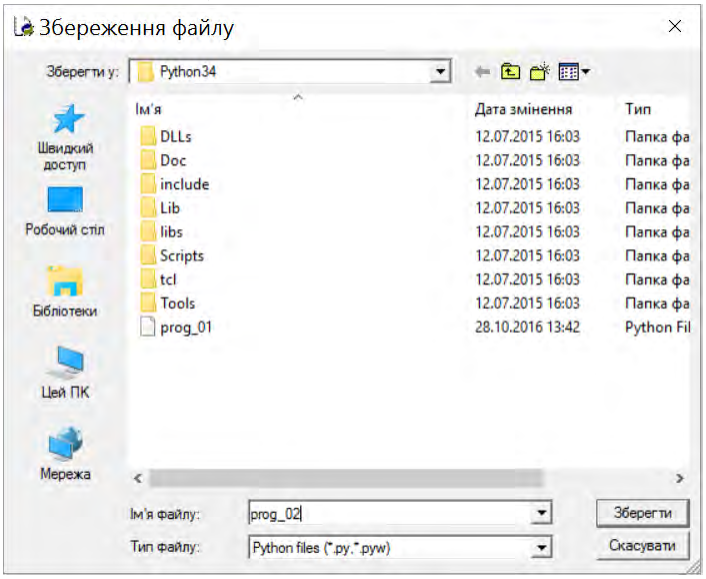

Рис. 2. Вікно **Збереження файлу**

Виконати програму можна у різний спосіб.

- Якщо програму відкрито у вікні Python 3.4.3 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.

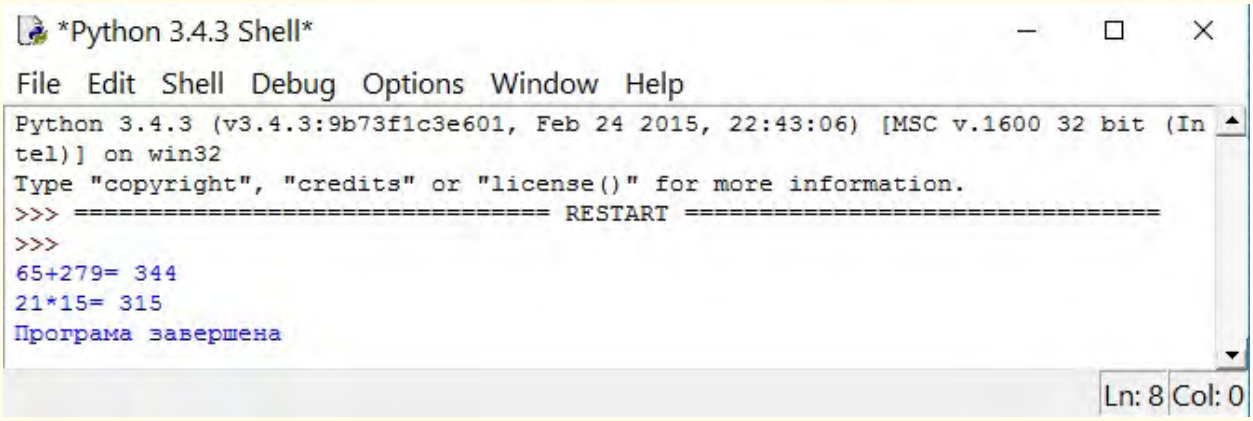

Рис. 3. Результат виконання програми додавання і множення чисел

- Якщо програму створено раніше, її можна викликати у вікно Python 3.4.3 Shell, виконавши команду File → Open... Потім у папці C:\Python34 слід знайти ім'я необхідного файла, перенести його у рядок Ім'я файлу і натиснути кнопку Відкрити.

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

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

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 у 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 недоліки має інтерактивний режим інтерпретатора?
6. Чому доцільно зберігати програмний код у файлі?
7. Для чого у 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 переконатеся, що отриманий результат є правильним.
2. Завантажте середовище IDLE, викличте створену у пункті 1 програму з файла f_01, замініть у програмі операцію множення числа 106 на 4 на операцію ділення. Збережіть програму у тому самому файлі. Виконайте програму і 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 і 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 у 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 умовчанням 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 декілька десятків ключових (зарезервованих) слів (наприклад, int, list, input, print, float та ін.). 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 знаку 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 у 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, IBM, Red Hat, Dropbox, Pinterest.

У мові Python існують базові об'єктні типи (вбудовані у 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 у розділі 5.

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'}

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»
```

Як уже зазначалося, у мові Python застосовується динамічна типізація змінних. Це означає, що не потрібно оголошувати типи змінних, як це робиться у багатьох мовах програмування, оскільки їхній тип визначається автоматично у 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 декільком змінним, наприклад:

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

Ще раз відзначимо, що після виконання оператора присвоювання у 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).

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
```

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).

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
```

Із прикладу 3 видно, що ми змінили тільки значення y[0], а фактично об'єкти мають однакові значення, тому що 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]
```

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 декільком змінним.

У такому разі змінні і 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 повинна 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 із 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 у змінних?
6. Поясніть сутність динамічної типізації змінних.
7. Для чого слугує оператор **is**?
8. Поясніть сутність поняття «посилання на об'єкт».
9. Як можна одним оператором присвоїти різні значення декільком об'єктам?
10. Що називають лексемою?
11. Які існують види лексем?

Завдання для самостійного виконання

1. Виявіть помилки у 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 у перетворенні типів даних? Чи доводилося вам стикатися з цією проблемою під час розв'язування математичних задач?

Одна й та сама змінна у 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, слугує 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 і 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.

prog_03.py - C:\Python34\prog_03.py (3.4.3)

File Edit Format Run Options Window Help

```
# Програма складання чисел з перетворенням типу str в 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 ж запрошення для уведення значення змінної y. Уведемо, наприклад, число 9 і натиснемо Enter. У результаті отримаємо результат 35.

Після того як змінні у 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, тому що змінну x видалено.

Приклад 7.

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

Запитання для перевірки знань

1. Яка функція слугує 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 у 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 у 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* та ін.

Арифметичні оператори призначено для виконання операцій над числами. Якщо операція виконується над цілим і дійсним числами, то ціле число буде спочатку перетворено 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.

До арифметичних операторів належать такі загальновідомі оператори:

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

Крім 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
```

Операторы порівняння порівнюють значення об'єкта, який розташовано ліворуч від оператора, із 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.

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

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

Таблиця 3. Операторы порівняння

Позначення	Назва	Пояснення	Приклад
==	Дорівнює	Якщо значення операндів однакові, повертається значення True, інакше — False	>>>23==40 False
!= або <>	Не дорівнює	Якщо значення операндів не однакові, повертається значення True, інакше — False	>>>2!=13 True
>	Більше	Якщо значення операнда зліва більше за значення справа, повертається значення True, інакше — False	>>>5>204 False
<	Менше	Якщо значення операнда зліва менше за значення справа, повертається значення True, інакше — False	>>>5<204 True
>=	Більше або дорівнює	Якщо значення операнда зліва більше або дорівнює правому, повертається значення True, інакше — False	>>>17>=17 True
<=	Менше або дорівнює	Якщо значення операнда зліва менше або дорівнює правому, повертається значення True, інакше — False	>>>43<=17 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 у 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 у тому випадку,

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

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 і рядками.

Далі роз'яснено сутність і наведено приклади виконання операцій над рядками (табл. 5) (операції над іншими типами послідовностей описано у розділі 5):

Таблиця 5. Операції над рядками

Операція	Позначення	Приклад
Об'єднання (слугує для з'єднання послідовностей)	+	У результаті виконання операції над рядками ''послі''+''довність'' отримаємо результат 'послідовність'
Повторення (слугує для повторення послідовності задану кількість разів)	*	>>>''ав''*5 'авававававав'
Перевірка на входження послідовностей одна до іншої	in	Якщо одна послідовність входить до іншої, видається значення True, інакше — False. Приклад: >>>''виконання'' in ''Результатом виконання операції'' True >>>''оператор'' in ''Результатом виконання операції'' False
Перевірка на невходження однієї послідовності до іншої	not in	Якщо одна послідовність входить до іншої, видається повідомлення False, інакше — True. Приклад: >>>''байт'' not in ''біт'' True >>>''байт'' not in ''байт і біт'' 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) арифметичні
Повторення задану кількість разів	*=	>>>s= ''пр''; s*=6; s '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 із 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) і *рядкові* (результат рядкового типу).

Для кожного типу операцій існують чіткі правила їх запису і виконання, з якими ми будемо знайомитися поступово у 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 повинні бути узгоджені. Якщо автоматично вони не узгоджуються, слід використати засоби перетворення типів, що були описані вище.

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 і ділення за модулем.
2. Дано два числа. Виконайте їх додавання і 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 рядку "Чемпіонат України" у 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 в 8 і 9 класах. Чи доводилося вам самостійно розробляти функції опрацювання чисел?

Числа можуть подаватися у 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 у 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") (8, 57)
float([<число або рядок>])	Перетворює ціле число або рядок на дійсне число	>>>float(23), float("11.7") (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) (0, 45.3)
abs(число)	Повертає абсолютне значення	>>>abs(22), abs (–66) (22,66)
pow(<число>,<степінь>)	Повертає число у степені	>>>pow (10, 2), pow (2, 4) (100, 16)
max(<числа через кому>)	Повертає максимальне значення	>>>max(9,4,7), max(5,9.7,3) (9, 9.7)
min(<числа через кому>)	Повертає мінімальне значення	>>>min(21,5,19), min(7,2.5,6) (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) (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 і функції, використовують для роботи з числами. Для роботи з константами і функціями необхідно імпортувати його у програму за допомогою інструкції import math.

Найвживаніші функції модуля math наведено в табл. 2.

Таблиця 2. Найчастіше вживані функції модуля math

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

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

У табл. 3 наведено найчастіше вживані 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	>>>import random >>>random.random () 0.4632200164843052
uniform (початок,кінець)	Генерує дійсне випадкове число у діапазоні від «початок» до «кінець»	>>>random.uniform (3, 9) 4.951275580428769
randint (початок,кінець)	Генерує ціле випадкове число у діапазоні від «початок» до «кінець»	>>>random.randint (3, 8) 7
choice (послідовність)	Вибирає з послідовності (рядка, списку або кортежу) випадковий елемент	>>>random.choice («Python») '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 і мінімальне число у 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 у послідовності: 23, 40, 29, 33, 17.
4. Обчисліть суму факторіалів чисел 5 і 7.

3. Реалізація базових алгоритмічних конструкцій
3.1. Реалізація алгоритмів з розгалуженням

Відомо, що існує три базові структури алгоритмів: слідування, розгалуження і повторення. Пригадайте, якими операторами мови ви реалізували 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).

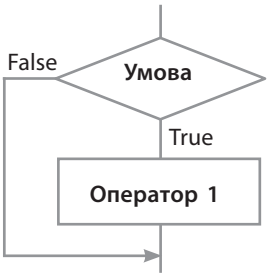

Рис. 1. Графічна схема одноальтернативного розгалуження

- 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>
```

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 у наведеному прикладі в операторі умовного переходу міститься блок із двох операторів, який починається після чотирьох пробілів.

Приклад 1.

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

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.

prog_3.2.py - C:\Python34\prog_3.2.py (3.4.3)

File Edit Format Run Options Window Help

```
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>
```

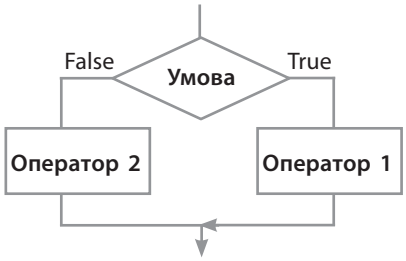

Рис. 3. Графічна схема двоальтернативного розгалуження

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 вписати 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 в 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 і рухався 5,3 год. Потім о 9.00 виїхав Микола і рухався за тим самим маршрутом 4,2 год. Хто з хлопців 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 галуження з багатьма варіантами вибору. Графічну схему розгалуження з трьома можливими варіантами зображено на рис. 6.

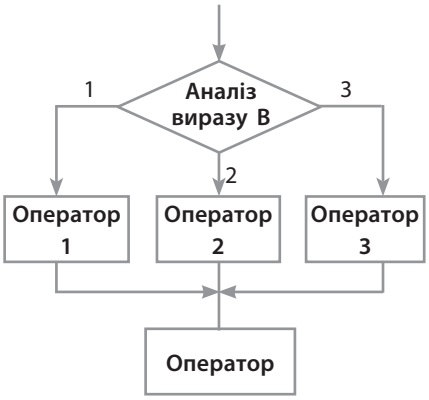

Рис. 6. Графічна схема розгалуження з трьома варіантами вибору

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 повинно бути унікальним константним виразом. Значення виразу порівнюється з кожним значенням в операторах elif у порядку їх розташування. Якщо певне значення вираз співпадає зі значенням оператора elif, то виконується той блок операторів, що розташований безпосередньо за цим оператором. Якщо значення вираз не співпадає 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 у 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 дорівнює s грн. За якісне й дострокове виконання завдання йому нараховується премія у розмірі 50 % окладу. Розробіть програму визначення реальної заробітної платні працівника.
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. Тренер формує команду для гри у баскетбол з учнів, які на зріст не нижчі ніж 180 см. Розробіть програму визначення, чи потрапить у цю команду учень зростом h см?
5. Дано рівносторонній трикутник зі стороною b. Розробіть програму визначення, чи можна у трикутник вписати коло з радіусом r.
6. Відомі три найкращі результати учнів школи у стрибках у довжину. Розробіть програму, за допомогою якої за результатами стрибків визначаються прізвища учнів.
7. У п'ятницю у 9 класі такі уроки: 1 — історія, 2 — математика, 3 — географія, 4 — інформатика, 5 — фізкультура. Розробіть програму, за допомогою якої визначається назва предмету 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 зустрічаються випадки, коли виконання або невиконання події залежить не від однієї умови, а від декількох. 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).

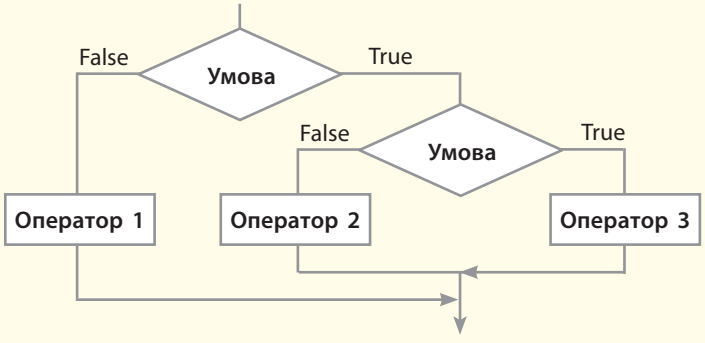

Рис. 1. Графічна схема алгоритму з вкладеним умовним переходом

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, опирайтесь 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.

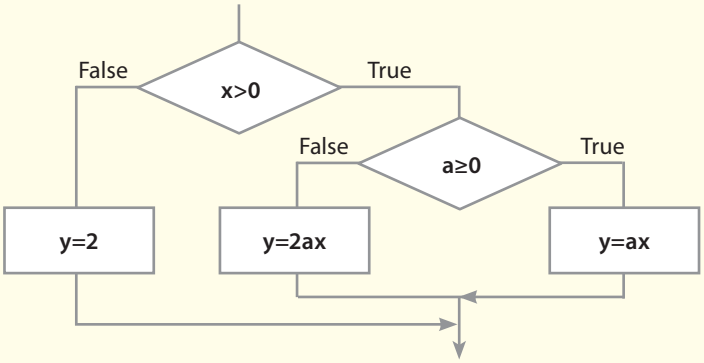

Рис. 2. Графічна схема алгоритму обчислення значення виразу

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         # Так для другого оператора if
    else :
        y = 2*a*x       # Ні для другого оператора if
else :
    y =2                # Ні для 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.

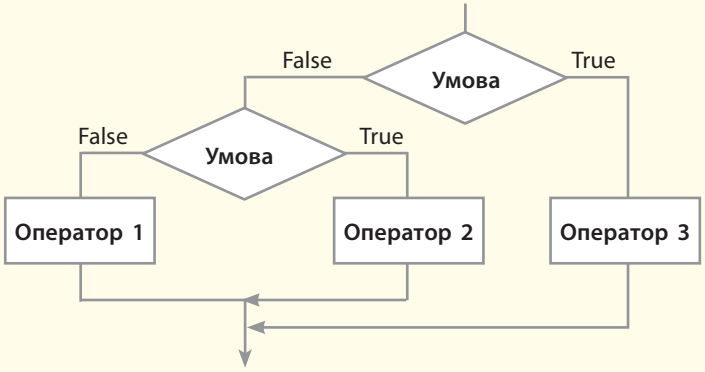

Рис. 4. Графічна схема алгоритму з вкладеним умовним переходом

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

У науковій сфері **Python** широко використовують західні вчені-непрограмісти (математики, фізики, біологи) завдяки простоті вивчення.

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

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 з 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 удвічі, 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 з Києва у 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 усі 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 у команди «Ворскла», і щоб «Шахтар» при цьому програв «Дніпру». Розробіть алгоритм із 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.

Змінна і — 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.

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. Інструкція i: = 1 виконує підготовку до реалізації інструкцій тіла циклу, а за допомогою інструкції i ≤ 10 здійснюється перевірка завершення їх виконання.

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
]
```

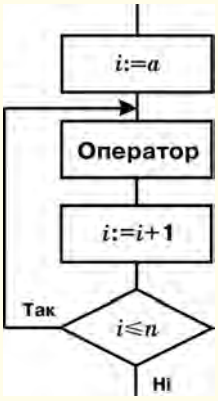

Рис. 1. Фрагмент графічної схеми циклу з параметром

Рис. 2. Фрагмент графічної схеми циклічного алгоритму

Інструкції у квадратних дужках є необов'язковими.
Якщо всередині циклу не використовується оператор **break** (із ним ми познайомимося пізніше), то після завершення виконання циклу буде виконуватися 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 може також створити програміст 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.

Етап 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.
Етап 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.
Етап 2	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 і 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
    p = 8*i
    print ("8*",i,"= ",p)
input ()
```

Рис. 3. Код множення чисел на 8

Приклад 2.

У банк покладено 5000 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
    s = 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.

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

Рис. 5. Код обчислення кількості букв у рядку

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. 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 терміну й вивести його стільки разів, скільки він повторюється. На рис. 6 неведено код, за допомогою якого у заданої 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 =k+1             # обчислення кількості слів
print ()                   # перехід на новий рядок
print ("Таких слів: ",k)
```

Рис. 6. Код обчислення кількості заданих слів

Результат виконання коду є таким:

```
біт біт біт
Таких слів:  3
```

3.3.2. Цикли з умовою

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

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

- Цикли з передумовою

Графічну схему циклу з передумовою зображено на рис. 7.

Рис. 7. Графічна схема циклу з передумовою

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.

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 1.

Автомобіль, рухаючись зі швидкістю v км/год, раптом здійснює термінове гальмування. За першу секунду швидкість його руху падає на 10 км/год, а за кожну наступну секунду швидкість зменшується на 10 км/год від значення за попередню секунду. Через скільки секунд автомобіль зупиниться?

На рис. 8 наведено код програми моделювання процесу гальмування автомобіля.

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

Рис. 8. Код обчислення часу гальмування автомобіля

Змінна k — це кількість секунд гальмування, змінна y — значення швидкості гальмування автомобіля за поточну секунду.

Алгоритм стиснення даних MP3 був розроблений німецькими дослідниками у 1987 році. Алгоритм стискає деякі аудіофайли в 10 разів. Він фактично спричинив революцію в музичної індустрії.

Приклад 2.

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 к ="))
n =0  # n - це кількість розрядів
p =1  # p - поточне значення кількості байтів
while (k > p) :
    n = n + 1
    p = 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 3.

Батискаф заглиблюється в океан. За першу хвилину він заглиблюється на 10 м, а за кожну наступну хвилину 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.

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

Рис. 11. Код моделювання процесу заглиблення батискафа

У програмі використані такі змінні: k — кількість хвилин заглиблення, h — поглиблення за поточну хвилину, H — поточне значення глибини заглиблення. Оператор break здійснює переривання виконання циклу.

Приклад 4.

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 цього завдання зображено на рис. 12.

```
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

У програмі використані змінні: s — поточна сума вкладу, k — кількість років вкладу. Після виконання програми на екран буде виведено: Через 5 років.

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 14 до 24.

```
for i in range (6, 31, 2) :
    if 12 < i < 26 :  # якщо числа в діапазоні 12<i26
        continue      # перехід на нову ітерацію 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 зображений ще один приклад програми з використанням оператора **break**. У цьому прикладі 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 рядку у число
    s = s + a    # обчислення суми чисел
print ("сума чисел = ", s)
input ()
```

Рис. 14. Код з використанням оператора continue

3.3.4. Вкладені цикли

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

Вкладені цикли — це такі 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

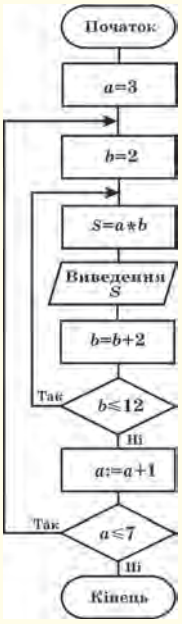

Рис. 15. Графічна схема алгоритму зі вкладеним циклом

Запитання для перевірки знань

1. Які існують види циклічних алгоритмів?
2. Поясніть на прикладі сутність параметру і тіла циклу.
3. Для чого призначений оператор **break**?
4. Які цикли називають вкладеними?
5. Накресліть графічну схему циклу з параметром.
6. Яку структуру має оператор циклу **for**?
7. Яку структуру має оператор циклу з передумовою?
8. Наведіть приклад вкладеного циклу.
9. Як у мові Python можна реалізувати цикл з післяумовою?
10. Поясніть на прикладі сутність оператора **continue**.

Завдання для самостійного виконання

1. Розробіть код обчислення суми для чисел 2, 7, 21, 9, 33, 13.
2. Розробіть код, який обчислює суму 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 дорівнює 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 кожного з них.
6. Дано рівносторонній трикутник зі стороною b. Розробіть програму визначення, чи можна в трикутник уписати коло з радіусом r.
7. Радіус першої кулі дорівнює 2 см, а радіус кожної наступної збільшується на 0.5 см. Розробіть код для визначення бокових поверхонь перших шести куль.
8. Відомо результати плавання вільним стилем на дистанції 50 м п'яти учнів кожного з трьох 10-х класів школи. Розробіть код, за допомогою якого визначається середній час запливу учнями кожного класу.
9. У банк покладено S грн під n % 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 мові програмування Python лінійні й табличні структури вбудовані в саму мову, що суттєво полегшує та спрощує користувачу роботу з ними.

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

Структуровані дані поділяють на лінійні, табличні та ієрархічні.

У *лінійній структурі* адреса елемента даних однозначно визначається його номером (індексом) у цієї 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 даного елемента.

Далі розглянемо ті структури даних, які передбачені навчальною програмою для 10 класу, решта вивчаються 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 програмування Python — це певна сукупність об'єктів будь-якого типу в 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. Самі списки можуть бути вкладені в інші типи об'єктів.

Списки деякою мірою нагадують масиви, 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, який 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.

Списки можуть бути *одновимірними* і *багатовимірними*.

Розглянемо спочатку одномірні списки. Позиція елемента у списку задається індексом, який починається з нуля.

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

Приклад 2.
```
>>>al_1 = ["файл", 21, 13, "5"]
>>>al_1
['файл', 21, 13, '5']
```

Розглянемо операції, які можна виконувати над списками, на прикладах.

Щоб звернутися до елемента списку, необхідно у квадратних дужках зазначити індекс елемента	>>>a_1 = [7, 20, "миша"] >>>a_1 [2] **'миша'**	 # виведення другого елемента
Значення елементів списку можна змінювати шляхом присвоювання їм нових значень	>>>b_1 = [21, 11, 17, 24] >>>b_1 [2] = 15 >>>b_1 **[21, 11, 15, 24]**	 # зміна значення другого елемента # виведення списку
Елементи списку можна присвоїти декільком змінним	>>>x1, x2, x3 = [3, 7, 5] >>>x1, x2, x3 **(3, 7, 5)**	# елементи списку присвоюються # 3 змінним # виведення значень змінних
Якщо елементів списку більше кількості змінних, зайві елементи можна присвоїти одній змінній, поміченій зірочкою	>>>x1, *x2, x3 = [1, 3, 7, 5] >>>x1, x2, x3 **(1, [3, 7], 5)**	# змінній x2 присвоюються два # значення: 3 і 7 # 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] >>>a [1:4] **[2, 3, 4]**	 # виділення елементів з 1-го до 3-го # включно
Списки можна об'єднувати	>>>a1 = [1, 2, 3, 4] >>>a2 = [5, 6, 7] >>>a1 + a2 **[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") **['п', 'р', 'о', 'ц', 'е', 'с', 'о', 'р']**	# перетворення рядка у список
Отримати кількість елементів у списку можна за допомогою функції len ()	>>>a = [1,4, 6,8] >>>len (a) **4**	 # довжина списку

Отримати список із випадковими числами з іншого списку можна за допомогою функції sample (послідовність, кількість елементів) з модуля random	>>>import random >>>random.sample (range (51), 6) **[48, 11, 46, 3, 21, 0]**	# імпортування модуля random # шість випадкових чисел # у діапазоні 0–51
Перетворити список у рядок можна різними засобами. Найпростіший з них — використання функції str	>>>a1 = ["file1", 25, "file2", 21] >>>str (a1) **"['file1', 25, 'file2', 21]"**	# список # 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] >>>max (a1), min (a1) **(44, 3)**	 # виведення максимального # та мінімального елементів
Для отримання випадкового елемента із списку слід імпортувати модуль random і скористатися функцією choice (послідовність) або функцією sample (послідовність, кількість елементів)	>>>import random >>>random.choice (["a", "ab", "c", "ca", "f"]) **'ab'**	# імпортування модуля random # 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] >>>a1.append ("bmv") >>>a1 **[1, 'as', 2, 'bmv']**	# список # додавання елемента bmv # у кінець списку # виведення списку
Метод extend ("послідовність") — додає елементи послідовності в кінець списку	>>>a1 = [1, 2, "web"] >>>a1.extend ("file, 7") >>>a1 **[1, 2, 'web', 'f', 'i', 'l', 'e', ',', ' ', '7']**	# список # додавання елементів у список # виведення списку
Метод insert (індекс, об'єкт) — вставляє один об'єкт у вказану позицію списку	>>>a1 = [3, "vin", 5] >>>a1.insert (1, "file") >>>a1 **[3, 'file', 'vin', 5]**	# список # додавання у 1 позицію списку # елемента # виведення списку
Метод pop (індекс) — видаляє елемент зі списку за вказаним індексом. Видалити елемент зі списку можна також за допомогою оператора del [індекс]	>>>b1 = [7, "com", 5] >>>b1.pop (2) **5** >>> b1 **[7, 'com']**	# список # виведення видаленого елемента # виведення списку після видалення # елемента

Метод remove (значення) — видаляє із списку перший елемент, який містить вказане значення	>>>a1 = [3, 5, "lad", 7, "lad"] >>>a1.remove ("lad"); a1 # видалення елемента lad із другої позиції **[3, 5, 7, 'lad']**
Метод clear () — видаляє зі списку всі елементи	>>>b1 = [5, 11, 8, 7, 20] >>>b1.clear (); b1 **[]**
Метод index (значення [' початок ['кінець]]) — повертає індекс елемента, який має вказане значення. За замовчуванням параметрів **початок** і **кінець** пошук елементу буде виконуватися від початку до кінця списку	>>>b1 = [10, 7, 55, 16, 8, 20] >>>b1.index (16) # виведення індексу елемента зі значенням 16 **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] >>>c1.count (7) # виведення кількості елементів зі значенням 7 **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: Відзначимо, що **метод sort ()** перетворює старий список у новий	>>>a1 = [17, 100, 20, 3, 8, 16, 25] # список >>>a1.sort (reverse = True) # сортування в порядку зменшення значень >>> a1 # виведення впорядкованого списку **[100, 25, 20, 17, 16, 8, 3]**
Метод sorted (послідовність ['key=None]['reverse = False]) — призначений для сортування списку і збереження старого	>>>b1 = [39, 16, 21, 7, 23, 15] # початковий список >>>sorted (b1) # повернення впорядкованого списку **[7, 15, 16, 21, 23, 39]** >>>b1 # повернення початкового списку **[39, 16, 21, 7, 23, 15]**

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 стека. 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 стека 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. Враховуючи те, що звернення до підпрограм виконується досить часто, стек здебільшого реалізується на апаратному рівні, а не програмному.

У мові Python програмно стек реалізується в основному на основі списку.

Програмно стек реалізується на основі списку або масиву. Якщо використовується масив, то потрібно визначати його розмір, комірки якого використовуються за необхідністю. Неправильне визначення розміру масиву може призвести до помилок у роботі програми або до неефективного використання пам'яті.

Для 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:

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

Реалізація стеку на основі масиву простіше і потрібний менший обсяг 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». Її можна порівняти, наприклад, із чергою у залізничну касу: перший клієнт біля каси обслуговується першим. Елемент, що додається до черги, опиняється в її кінці, а елемент, який видаляється із черги, знаходиться в її початку.

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

Приклад 4.

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

4.1.4. Багатовимірні списки

У багатовимірних (або вкладених) списках кожна група елементів списку береться у квадратні дужки.

Наприклад, двовимірний список можна створити 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 кожний 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-ій позиції
print (sorted (a))   # сортувати список із збереженням старого
print (a)            # виведення старого списку
a.insert (4, 5)      # вставити число 5 на 4-ту позицію
print (a)            # виведення списку із змінами
a[1] = 10            # замінити перший елемент на число 10
print (a)            # виведення списку із змінами
```

Рис. 1. Програма опрацювання списку

Результат виконання програми є таким:

```
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-го рядку
b.append ("Россум")             # зміна другого списку
c = a+b                         # об'єднання списків
print (c)                       # виведення нового списку
print (" ".join (c))            # перетворення списку у рядок
```

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

Нижче 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

Результат виконання програми такий:

```
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 можна отримати випадкові елементи з іншого списку?
7. Як перетворюється список у рядок?
8. Для чого призначений метод **append ()**?
9. Пояснити сутність методу **insert ()**.
10. Для чого призначений метод **remove ()**?
11. 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 знайти 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 із 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'єднайте списки в один і перетворіть його на рядок.
6. Дано список [5, 7, 8, 12, 4]. Обчислити 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?

4.2.1. Кортежі

Кортеж — це сукупність незмінних 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.

Найпростіший спосіб створення кортежу — перелічення його елементів через кому в круглих дужках або без них (приклад 8):

Звернемо особливу увагу на другий рядок прикладу 8, а саме на кому. Фактично кортеж формує кома, а не круглі дужки. Якщо в другому і третьому рядках не було б коми, то був би створений не кортеж, а об'єкт іншого типу. Тому в другому рядку, незважаючи на те, що в ньому тільки один елемент, поставлена кома.

Приклад 8.

```
>>>a1 = ()                    # пустий кортеж
>>>a2 = ("ab, ")              # кортеж з одного елемента
>>>a3 = 21, "abc", (3, 4)     # кортеж із трьох елементів
>>>a1, a2, a3                 # виведення значень елементів
                               # кортежу
((), 'ab, ', (21, 'abc', (3,4)))
```

Позиція елемента в кортежі визначається його індексом, нумерація починається з нуля. У кортежі, як і в інших послідовностях, можна отримати елемент за його індексом, отримати зріз, конкатенацію, повторення, перевірку на входження (оператор in) і невходження (приклад 9).

Приклад 9.

```
>>>a = 3, 4, 5, 7, 9, 11      # кортеж
>>>a[3]                       # виведення елемента, розташованого на 3 позиції
7
>>>a[2:5]                     # зріз елементів з 2 до 4 позиції включно
(5, 7, 9)
>>>2 in a                     # перевірка входження числа 3 у кортеж a
False
>>>(1,2)*3                    # повторення елементів кортежу 3 рази
(1, 2, 1, 2, 1, 2)
>>>(13, 14) + (15, 16, 17)    # об'єднання 3 кортежів
(13, 14, 15, 16, 17)
```

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

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

Нижче наведені деякі приклади використання кортежів.

Кортеж у математиці — впорядкована 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.

Приклад 10.

```
>>>b = 21, "ab", 17, "abc", (1, 2), 17      # кортеж із різними
                                            # типами елементів
>>>len (b)                                  # визначення довжини
                                            # кортежу
6
>>>b.index ("abc")                          # визначення індексу
                                            # елемента "abc"
3
>>>b.count (17)                             # визначення кількості
                                            # елементів, рівних 17
2
```

Приклад 11.

Створити кортеж з елементами: 14, 107, Київ, Театральна, 21. Визначити елемент, розташований на другій позиції, елементи з нульової до другої позиції включно, кількість елементів, дорівняних 107, і номер позиції, із якої починається елемент Театральна. Варіант програми реалізації цього завдання зображено на рис. 1.

```
a = 14, 107, "Київ", "Театральна", 21
print (a[2])                    # елемент на другій позиці
print (a[0:3])                  # виріз ()-го, 1-го і 2-го елементів
print (len (a))                 # довжина кортежу
print (a.count (107))           # кількість елементів 107
print (a.index ("Театральна"))  # позиція елемента "Театральна"
```

Рис. 1. Програма опрацювання кортежу

Результат виконання програми є таким:

```
Київ
(14, 107, 'Київ')
5
1
3
```

4.2.2. Діапазони

Діапазон — це незмінна послідовність цілих чисел з 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 умовчанням значення параметра **початок** дорівнює нулю, а крок — одиниці.

Приклад 12.

```
>>>a = range (7)                  # діапазон цілих чисел від 0 до 6
>>>for i in a: print (i, end=" ")  # циклічне виведення значень елементів
0 1 2 3 4 5 6
>>>b = range (4, 20, 3)            # діапазон цілих чисел від 4 до 20 з кроком 3
>>>for i in b: print (i, end=" ")  # циклічне виведення значень елементів
4 7 10 13 16 19
```

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

Перетворити діапазон у список і кортеж можна за допомогою відповідно функцій list () і tuple ():

```
>>>list (range (1, 13, 2))      # перетворення діапазону від 1 до 13 у список
[1, 3, 5, 7. 9, 11]
>>>tuple (range (1, 13, 2))     # перетворення діапазону від 1 до 13 у кортеж
(1, 3. 5, 7, 9, 11)
```

Діапазони можна порівнювати на рівність. Якщо вони рівні, генерується значення True, інакше — False.

Приклад 13.

```
>>>a = range (2, 9)              # діапазон цілих чисел від 2 до 8 з кроком 1
>>>a[3], a[6]                    # звернення до елементів на 3-ій і 6-ій позиціях
(5, 8)
>>>a [2:6]                       # зріз діапазону з 2-ої до 6-ої позиції
range (4, 8)
>>>1 in a, 7 in a                # перевірка входження чисел 1 і 7 у діапазон
(False, True)
>>>len (a)                       # визначення довжини діапазону
7
>>>max (a), min (a)              # визначення максимального і мінімального чисел
(8, 2)
>>>range (6) == range (6)        #порівняння значень елементів діапазону
True
```

Приклад 14.

Створити діапазон чисел від 3 до 17 з кроком 2. Перетворити діапазон у список і кортеж, викликати значення елемента на п'ятій позиції, визначити довжину діапазону та суму всіх елементів.

Варіант програми реалізації цього завдання зображено на рис. 1.

```
b = range (3, 18, 2)          # створення діапазону
print (b)                     # виведення діапазону
s = 0                         # початкове значення суми
print (list (b))              # перетворення діапазону у список
print (tuple (b))             # перетворення діапазону у кортеж
print (b [5])                 # виведення елементу 5-ої позиції
print (len (b))               # довжина діапазону
for i in range (len (b)):     # циклічне обчислення суми
    s = s +b[i]               # обчислення суми елементів діапазону
print ("сума = ",s)           # виведення суми
input ()
```

Рис. 1. Програма опрацювання діапазону

Нижче наведено результат виконання програми:

```
range(3, 18, 2)
[3, 5, 7, 9, 11, 13, 15, 17]
(3, 5, 7, 9, 11, 13, 15, 17)
13
8
80
```

4.2.3. Множини

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

Існує два типи множин: змінна (set) і незмінна (frozenset ()). Далі розглядається перший із них. 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}
```

Для виконання операцій над множинами у мові Python існують різні оператори, функції і методи. Далі розглянемо деякі з них на прикладах.

Множини в мові Python створюються абсолютно випадковим чином. Елементи можна розміщувати як завгодно, згодом вони все одно розташуються у випадковому порядку. Крім того, множини не можуть мати елементів, які повторюються, тому всі елементи, що будуть однаковими, НЕ будуть виведені повторно.

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

Frozenset це множина, яку не можна надалі змінювати. Тобто не можливо ані додати нові елементи, ані видалити з нього вже наявні.

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

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

Приклад 15.

Створити дві множини: 2, 4, 6, 8 і 10, 12, 14, 16. Перевірити, чи рівні ці множини. Об'єднати створені множини в одну й додати до неї число 18. Визначити довжину нової множини, перевірити, чи є в ній число 12 і знайти суму чисел другої початкової довжини. Варіант програми реалізації цього завдання зображено на рис. 2.

```
a = set ([2, 4, 6, 8])      # перша множина
b = set ([10, 12, 14, 16]) # друга множина
print (a == b)              # множини рівні?
c = a.union (b)             # об'єднання множин
print (c)                   # виведення об'єднанної множини
c.add (18)                  # дабавлення нового елемента
print (c)                   # виведення нової множини
print (len (c))             # довжина нової множини
print (12 in c)             # чи є число 12 у множині
s = 0                       # початкове значення суми
for i in set ([10, 12, 14, 16]):
    s = s + i               # обчислення суми другої множини
print ("сума =", s)         # виведення суми
```

Рис. 2. Програма опрацювання множини

Нижче наведено результат виконання програми:

```
False
{2, 4, 6, 8, 10, 12, 14, 16}
{2, 4, 6, 8, 10, 12, 14, 16, 18}
9
True
52
```

Запитання для перевірки знань

1. Що називають діапазоном?
2. Що називають множиною?
3. Як створюються кортежі?
4. Як створюється діапазон?
5. Для чого використовуються методи **union()** і **add()**?
6. Як перетворюється список у множину?
7. Як перетворюється діапазон у кортеж?
8. Навести приклад використання функцій **len()** і **max()** для кортежу.
9. Навести приклади використання методів **index()** і **count()** для діапазону.
10. Навести приклад використання методу **update()**.

Завдання для самостійного виконання

1. Створити й вивести діапазон чисел від 5 до 15 із кроком 1. Перетворити діапазон у список і визначити його довжину.
2. Створіть кортеж з елементами: 21, 16, "Прут", 16, "Псел", 16. Визначте кількість елементів, що дорівнюють 16, позицію, з якої починається елемент **Псел**, і зріз елементів із першої по третю позицію включно.
3. Створіть кортеж, елементи якого мають такі значення: "Потяг", "слідує", "у", 5, "Запоріжжя о", 12.0. Визначте кількість елементів у кортежі й елемент на 3 позиції.
4. Створіть дві множини. Елементи першої: 1, 2, 3; елементи другої: 2, 4, 6. Об'єднайте множини. Додайте до неї елемент 7.
5. Створіть діапазон чисел від 12 до 28 із кроком 2. Перетворіть діапазон у кортеж. Обчисліть суму значень його елементів. .
6. Створіть дві множини з елементами: 5, 7, 8 і 12, 14, 16, 18. Визначте довжину кожної з них. Об'єднайте множини та обчисліть суму значень її елементів.

4.3. Словники

Словником у класичному розумінні ви неодноразово користувалися. У ньому за конкретним терміном (словом) знаходили відповідний пояснювальний текст або його переклад.

4.3.1. Функції, операції і методи опрацювання словників

Ключем може бути число, рядок або кортеж. Значення елементів словника можна змінювати. Словники не є послідовностями, тому такі загальні операції, як отримання зрізу, конкатенації та інші вони не підтримують.

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

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

Словники можна створювати різними способами. Розглянемо основні з них.

- За **допомогою функції 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}
```

Останній спосіб створення словників застосовується найчастіше.

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