

Python у шкільному курсі інформатики

Основи програмування



Чернігів
06.01.2020

УДК 004.42: 004.432.2(076.5)

ББК 32.973.26- 018.2

Р 463

Укладач: Ракута Валерій, старший викладач кафедри природничо-математичних дисциплін та інформаційно-комунікаційних технологій в освіті Чернігівського обласного інституту післядипломної педагогічної освіти імені К. Д. Ушинського.

Р463 Python у шкільному курсі інформатики. Основи програмування:
навчальний посібник / В.М. Ракута – Чернігів, 2020. – 160 с.

Право на некомерційне використання даної копії посібника має виключно Гузь Валентина.

Посібник присвячено використанню мови програмування Python у процесі вивчення шкільного курсу інформатики. Він є частиною дидактичного забезпечення авторського спец-курсу Валерія Ракути [«Python у шкільному курсі інформатики. Основи програмування»](#), але разом з тим, містить матеріали, що виходять за рамки програми даного курсу.

Книга призначена у першу чергу для вчителів інформатики закладів загальної середньої освіти, але також може бути корисна учням середніх шкіл та студентам ВНЗ у процесі вивчення ними основ програмування на мові Python.

Усі права захищено. Жодна частина, елемент, ідея, композиційний підхід цього видання не можуть бути скопійованими чи відтвореними в будь-якій формі та будь-якими засобами – ні електронними, ні фотомеханічними, зокрема ксерокопіюванням, записом чи комп'ютерним архівуванням – без письмового дозволу автора.

© Ракута В. М., 2020

Зміст

Вступ.....	6
Основи мови програмування Python	8
1. Про мову програмування Python. Чому саме Python?.....	8
Дзен Python.....	9
Питання для самоконтролю.....	10
2. Інсталяція Python у системі Microsoft Windows.....	10
Питання для самоконтролю.....	11
3. Інтегроване середовище IDLE. Перша програма.....	12
3.1. Практикум. Завдання 3.1	13
3.2. Практикум. Завдання 3.2	13
3.3. Практикум. Завдання для самостійного виконання	13
Питання для самоконтролю.....	14
4. Додаткові можливості IDLE	14
4.1. Практикум. Завдання 4.1	15
1.1. Практикум. Завдання для самостійного виконання	15
Питання для самоконтролю.....	16
5. Початкові відомості про синтаксис мови Python	16
Питання для самоконтролю.....	16
6. Коментарі.....	16
Питання для самоконтролю.....	17
7. Модуль turtle (Черепашка)	17
7.1. Практикум. Завдання 7.1 (Черепашка).....	21
7.2. Практикум. Завдання 7.2 (Черепашка).....	22
7.3. Практикум. Завдання 7.3 (Черепашка).....	22
7.4. Практикум. Завдання 7.4 (Черепашка).....	23
7.5. Практикум. Завдання 7.5 (Черепашка).....	23
7.6. Практикум. Завдання для самостійного виконання	23
Питання для самоконтролю.....	25
8. Змінні	25
Питання для самоконтролю.....	27
9. Типи даних у мові Python	27
Питання для самоконтролю.....	30
10. Виведення результатів роботи програми.....	30
Питання для самоконтролю.....	32
11. Введення даних.....	32
Питання для самоконтролю.....	33
12. Числа у мові Python.....	33

12.1.	Математичні оператори	34
12.2.	Оператори присвоювання	36
12.3.	Пріоритет виконання операторів	37
12.4.	Вбудовані функції	37
12.5.	Модуль math. Математичні функції	39
12.6.	Модуль random. Генерація випадкових чисел	40
12.7.	Практикум. Завдання 12.7	42
12.8.	Практикум. Завдання 12.8	43
12.9.	Практикум. Завдання 12.9*	43
12.10.	Практикум. Завдання 12.10*	44
12.11.	Практикум. Завдання 12.11	44
12.12.	Практикум. Завдання 12.12 (Черепашка)	45
12.13.	Практикум. Завдання 12.13	45
12.14.	Практикум. Завдання 12.14 (Черепашка)	46
12.15.	Практикум. Завдання для самостійного виконання	46
	Питання для самоконтролю	51
13.	Умовні оператори та цикли	52
13.1.	Операції порівняння	52
13.2.	Оператор розгалуження <i>if ... else</i>	53
13.3.	Практикум. Завдання 13.3	54
13.4.	Практикум. Завдання 13.4	55
13.5.	Практикум. Завдання 13.5	56
13.6.	Практикум. Завдання 13.6	56
13.7.	Практикум. Завдання для самостійного виконання	57
13.8.	Цикл <i>for</i>	58
	Функція <i>range()</i>	58
	Цикл <i>for</i>	59
	Приклади використання циклу <i>for</i>	59
13.9.	Практикум. Завдання 13.9	60
13.10.	Практикум. Завдання 13.10	61
13.11.	Практикум. Завдання 13.11	62
13.12.	Практикум. Завдання 13.12	62
13.13.	Практикум. Завдання 13.13*	63
13.14.	Практикум. Завдання 13.14 (Черепашка)	65
13.15.	Практикум. Завдання для самостійного виконання	66
13.16.	Цикл <i>while</i>	67
	Перехід на наступну ітерацію циклу. Оператор <i>continue</i>	68
	Переривання циклу. Оператор <i>break</i>	68
13.17.	Практикум. Завдання 13.17	69

13.18.	Практикум. Завдання 13.18.....	70
13.19.	Практикум. Завдання для самостійного виконання	70
	Питання для самоконтролю.....	72
14.	Функції користувача	73
14.1.	Створення функції та її виклик	73
	Розташування визначень функцій	74
14.2.	Практикум. Завдання 14.2*.....	75
14.3.	Практикум. Завдання 14.3 (Черепашка).....	75
14.4.	Практикум. Завдання для самостійного виконання	76
	Питання для самоконтролю.....	76
15.	Списки та кортежі.....	77
15.1.	Створення списку	77
15.2.	Основні операції над списками	78
15.3.	Багатовимірні списки	80
15.4.	Перебір елементів списку	81
15.5.	Додавання і видалення елементів списку	81
15.6.	Пошук елемента у списку	83
15.7.	Перевертання та перемішування списку	84
15.8.	Вибір елементів випадковим чином.....	85
15.9.	Сортування списку	85
15.10.	Заповнення списку числами	86
15.11.	Кортежі.....	87
15.12.	Практикум. Завдання 15.12.....	89
15.13.	Практикум. Завдання 15.13	89
15.14.	Практикум. Завдання 15.14	90
15.15.	Практикум. Завдання 15.15	90
15.16.	Практикум. Завдання 15.16	91
15.17.	Практикум. Завдання 15.17	91
15.18.	Практикум. Завдання 15.18.....	92
15.19.	Практикум. Завдання 15.19	92
15.20.	Практикум. Завдання 15.20	93
15.21.	Практикум. Завдання 15.21	93
15.22.	Практикум. Завдання 15.22	93
15.23.	Практикум. Завдання 15.23	94
15.24.	Практикум. Завдання 15.24	94
15.25.	Практикум. Завдання 15.25	94
15.26.	Практикум. Завдання 15.26	94
15.27.	Практикум. Завдання для самостійного виконання	95
	Питання для самоконтролю.....	96

16.	Масиви у Python. Модуль array	98
16.1.	Коли доцільно використовувати модуль array?	98
16.2.	Тип елементів. Створення масиву	98
16.3.	Методи для роботи з масивами (array) у Python	100
16.4.	Практикум. Завдання 16.4	100
16.5.	Практикум. Завдання 16.5	101
16.6.	Практикум. Завдання 16.6	102
16.7.	Практикум. Завдання 16.7	102
16.8.	Практикум. Завдання для самостійного виконання	102
	Питання для самоконтролю	103
17.	Рядки	104
17.1.	Створення рядків	104
17.2.	Зміна регістру символів у рядках	105
17.3.	Конкатенація	105
17.4.	Табуляції та розриви рядків	106
17.5.	Видалення пропусків	107
17.6.	Операції над рядками	108
17.7.	Методи для роботи з рядками	110
17.8.	Функції для роботи з символами	113
17.9.	Пошук та заміна у рядку	113
17.10.	Практикум. Завдання 17.10	116
17.11.	Практикум. Завдання 17.11	116
17.12.	Практикум. Завдання 17.12	117
17.13.	Практикум. Завдання 17.13	117
17.14.	Практикум. Завдання 17.14	118
17.15.	Практикум. Завдання 17.15	118
17.16.	Практикум. Завдання 17.16	119
17.17.	Практикум. Завдання для самостійного виконання	120
	Питання для самоконтролю	120
18.	Робота з файлами	121
18.1.	Загальні питання	121
18.2.	Зчитування всього файлу	123
18.3.	Шляхи до файлів	124
18.4.	Читання по рядках	125
18.5.	Створення списку рядків на основі вмісту файлів	126
18.6.	Робота із вмістом файла	127
18.7.	Великі файли: мільйон цифр	128
18.8.	Запис у файл	129
	Запис у порожній файл	129

Багаторядковий запис	129
Приєднання даних до файлу.....	130
18.9. Практикум. Завдання 18.9	130
18.10. Практикум. Завдання 18.10	132
18.11. Практикум. Завдання 18.11	132
18.12. Практикум. Завдання 18.12	133
18.13. Практикум. Завдання 18.13	133
18.14. Практикум. Завдання 18.14	134
18.15. Практикум. Завдання 18.15	134
18.16. Практикум. Завдання 18.16	134
18.17. Практикум. Завдання 18.17	135
18.18. Практикум. Завдання 18.18	135
18.19. Практикум. Завдання 18.19	136
18.20. Практикум. Завдання 18.20	136
18.21. Практикум. Завдання 18.21	137
18.22. Практикум. Завдання 18.22	138
18.23. Практикум. Завдання 18.23	138
18.24. Практикум. Завдання 18.24	139
18.25. Практикум. Завдання 18.25	140
18.26. Практикум. Завдання 18.26	140
18.27. Практикум. Завдання для самостійного виконання	141
Питання для самоконтролю	142
Додатки	143
Додаток 1. Елементи функціонального програмування у Python	143
Анонімні функції.....	143
Функція map	144
Функція zip	145
Функція filter	146
Ще дві корисні функції.....	146
Додаток 2. Список ключових слів та вбудованих ідентифікаторів	147
Додаток 3. Модулі у Python	148
Додаток 4. Інформація про олімпіадні завдання	149
Додаток 5. Перелік кольорів	150
Додаток 6. Проблема української мови в консолі	150
Додаток 7. Генератори списків у Python	151
Термінологічний словник.....	153
Література та використані джерела	155

Вступ

Один з найкращих способів чомусь навчитись – це написати на цю тему навчальний посібник. І коли у мене з'явилась ідея познайомитися з мовою програмування Python, то я і вирішив створити відповідну книгу.

Перш ніж ми почнемо, необхідно зробити два зауваження.

По-перше, не забувайте, що книги з програмування потрібно не тільки читати, а й виконувати всі приклади та завдання, а також експериментувати, змінюючи щонебудь у прикладах щоб глибше зрозуміти та краще засвоїти текст книги. Намагайтеся вдосконалити запропоновані у книзі алгоритми та оптимізувати наведений код. Це запорука успіху. Тому, якщо ви зручно влаштувалися на дивані і налаштувалися просто читати, у вас практично немає шансів вивчити мову.

По-друге, пам'ятайте, що прочитати цю книгу один раз недостатньо. Більшу частину книги ви повинні знати та розуміти! Скільки на це піде часу, залежить від ваших здібностей і бажання. Чим більше ви будете самостійно програмувати, тим більших успіхів досягнете. Щоб навчитися програмувати – потрібно ..., правильно – програмувати ☺!

Програмування, як відомо, – це процес створення комп'ютерних програм. Фундаментальні складові, що входять до поняття програмування, наведені на схемі 1. У посібнику основна увага буде приділена саме основам програмування.



Схема 1

У зв'язку з неузгодженістю використання деяких термінів у різноманітній літературі з програмування (особливо в книгах, перекладених з інших мов)

вважаємо за доцільне навести тлумачення деяких понять і термінів з програмування та алгоритмізації, що ми використовували у цьому посібнику. З цією метою був створений термінологічний словник.

У процесі роботи над книгою було опрацьовано та використано значну кількість джерел з програмування та алгоритмізації, зокрема присвячених мові Python та її функціональним можливостям, що наведені у списку літератури та використаних джерел. Це значною мірою сприяло підвищенню якості книги та пришвидшило роботу над нею.

Висловлюю вдячність рецензентам. Окрема подяка Андрію Костюченко. Його слушні поради та критичні зауваження значно сприяли покращенню якості книги.

Посібник розраховано на читачів з різним рівнем підготовки, як на тих хто вже знайомий з певною мовою програмування (що значно полегшить оволодіння мовою Python) так і на тих хто тільки починає знайомитися з алгоритмізацією та програмуванням.

Запрошую, також, всіх бажаючих на свій авторський курс [«Python у шкільному курсі інформатики. Основи програмування»](#).

Посібник є відкритим для подальшого вдосконалення. Автор буде вдячний за побажання та зауваження читачів, що сприятимуть підвищенню якості книги.

(E-mail: valerarakuta@gmail.com).

Основи мови програмування Python

1. Про мову програмування Python. Чому саме Python?

Отже, чому саме Python? Python – це об'єктно-орієнтована мова програмування високого рівня, призначена для самого широкого кола завдань. За її допомогою можна обробляти різні дані, створювати зображення, працювати з базами даних, розробляти Web-сайти та додатки з графічним інтерфейсом. Python є кросплатформенною мовою, що дозволяє створювати програми, які будуть працювати у всіх операційних системах. У цій книзі ми розглянемо базові можливості Python 3.7 стосовно операційної системи Windows.

Згідно з офіційною версією назва мови пішла не від змії. Творець мови Гвідо ван Россум (Guido van Rossum) назвав своє творіння на честь британського комедійного телешоу BBC "Monty Python's Flying Circus". Тому більш правильною назвою буде "Пайтон". Проте багато хто вважає, що більш звично називати мову "Пітон". Так чи інакше, а у цій книзі ми будемо дотримуватися традиційного написання слова Python англійською мовою.

Програма на мові Python являє собою звичайний текстовий файл з розширенням *py* (консольна програма) або *pyw* (програма з графічним інтерфейсом). Всі інструкції з цього файлу виконуються інтерпретатором порядково.

Python є об'єктно-орієнтованою мовою. Це означає, що практично всі дані є об'єктами, навіть самі типи даних. У змінній завжди зберігається тільки посилання на об'єкт, а не сам об'єкт. Наприклад, можна створити функцію, зберегти посилання на неї у змінній, а потім викликати функцію через цю змінну. Ця обставина робить мову Python ідеальним інструментом для створення програм, що використовують функції зворотного виклику, наприклад, при розробці графічного інтерфейсу. Той факт, що мова є об'єктно-орієнтованою, аж ніяк не означає, що ООП-стиль програмування є обов'язковим. Мовою Python можна писати програми як в ООП-стилі, так і в процедурному стилі.

Python не допускає двоякого написання коду. Наприклад, в мові Perl існує залежність від контексту і множинність синтаксису. Часто два програміста, які пишуть на Perl, просто не розуміють код один одного. В Python код можна написати тільки одним способом, бо у ньому відсутні зайві конструкції. Всі програмісти повинні дотримуватися стандарту, описаного у документі <http://python.org/dev/peps/pep-0008/>. Більш читабельного коду немає у жодній іншій мові програмування.

Синтаксис мови Python викликає багато нарікань у програмістів, знайомих з іншими мовами. На перший погляд може здатися, що відсутність обмежувальних знаків (фігурних дужок або конструкції `begin ... end`) для виділення блоків і обов'язкова вставка пробілів попереду інструкцій може приводити до помилок. Однак це тільки перше і неправильне враження. Хороший стиль програмування в будь-якій мові зобов'язує виділяти інструкції всередині блоку однаковою кількістю пробілів. У цій ситуації обмежувальні символи просто є зайвими. Існує думка, що програма буде по-різному виглядати у різних редакторах. Це невірно. Відповідно до стандарту для виділення блоків необхідно використовувати чотири пробіли. Чотири

пробіли в будь-якому редакторі виглядатимуть однаково. Якщо в іншій мові вас не привчили до хорошого стилю програмування, то мова Python швидко це виправить. Якщо кількість пробілів всередині блоку буде різним, то інтерпретатор виведе повідомлення про фатальну помилку, і програма буде зупинена. Таким чином, мова Python привчає програмістів писати красивий і зрозумілий код.

Оскільки програма на мові Python являє собою звичайний текстовий файл, його можна редагувати за допомогою будь-якого текстового редактора, наприклад за допомогою Notepad ++. Однак краще скористатися спеціалізованими редакторами, які не тільки підсвічують код, але і виводять різні підказки і дозволяють налагодити програму. Таких редакторів дуже багато, наприклад PyScripter, PythonWin, UliPad, Eclipse, + PyDev, Netbeans та ін. Повний список редакторів розташований на сторінці <http://wiki.python.org/moin/PythonEditors>. У цій книзі ми будемо використовувати редактор IDLE, який входить до складу стандартної бібліотеки Python у Windows. [4]

І як підсумок – кілька вагомих причин вивчення Python. [3, 7]

- Python – проста і зручна мова програмування. У порівнянні з багатьма іншими мовами читати і складати програми на Python зовсім не складно.
- У Python є бібліотеки готових процедур для використання у своїх програмах. Це дозволяє створювати складні програми швидко.
- Python підходить для створення серйозних програм. Наприклад, його використовують у Google, NASA та на студії Pixar.
- Python працює на різних платформах (Windows, Mac, Linux, Raspberry Pi тощо).
- Підтримка різних [парадигм програмування](#).
- Python має простий синтаксис, який дозволяє розробникам писати програми з меншою кількістю рядків, ніж деякі інші мови програмування.
- Python виконує програми за допомогою інтерпретатора, що значно спрощує процес написання та налагодження програм.

Можна також переглянути відео на YouTube https://youtu.be/BWiLeud_yh4.

Дзен Python

Розробники мови Python є прихильниками певної філософії програмування, яку називають «The Zen of Python» («Дзен Пайтона»). Її текст можна отримати у інтерпретаторі Python за допомогою команди `import this` (лише один раз за сесію). Автором цієї філософії вважається Тім Пейтерс. Текст філософії [12]:

- Гарне краще за потворне. (Beautiful is better than ugly.)
- Явне краще за неявне. (Explicit is better than implicit.)
- Просте краще за складне. (Simple is better than complex.)
- Складне краще за заплутане. (Complex is better than complicated.)
- Плоске краще за вкладене. (Flat is better than nested.)
- Розріджене краще за щільне. (Sparse is better than dense.)
- Легкість читання має значення. (Readability counts.)

- Особливі випадки не є настільки особливими, щоб порушувати правила. (Special cases aren't special enough to break the rules.)
- Хоча практичність є важливішою за бездоганність. (Although practicality beats purity.)
- Помилки ніколи не повинні проходити непомітно. (Errors should never pass silently.)
- Якщо їх приховування не прописано явно. (Unless explicitly silenced.)
- Зустрівши неоднозначність, опирайтесь спокусі вгадати. (In the face of ambiguity, refuse the temptation to guess.)
- Має бути один – і, бажано, тільки один – очевидний спосіб зробити це. (There should be one – and preferably only one – obvious way to do it.)
- Хоча спочатку він може бути й не очевидним, якщо ви не голландець. (Although that way may not be obvious at first unless you're Dutch.)
- Зараз – краще, ніж ніколи. (Now is better than never.)
- Хоча ніколи, найчастіше, – краще, ніж просто зараз. (Although never is often better than *right* now.)
- Якщо реалізацію важко пояснити – задум поганий. (If the implementation is hard to explain, it's a bad idea.)
- Якщо реалізацію легко пояснити – можливо, задум добрий. (If the implementation is easy to explain, it may be a good idea.)
- Простори імен – чудова річ, тож робімо їх більше! (Namespaces are one honking great idea -- let's do more of those!)

Питання для самоконтролю

1. Перерахуйте основні переваги та особливості мови програмування Python.
2. Які ви можете навести вагомі причини вивчення Python?
3. Що у філософії програмування «The Zen of Python» вам сподобалось найбільше?

2. Інсталяція Python у системі Microsoft Windows

Щоб встановити Python в системі Microsoft Windows виконайте такі дії:

- ⇒ у пошуковій системі Google наберіть запит «Python»;
- ⇒ серед результатів пошуку оберіть офіційний сайт проекту (<https://www.python.org>);
- ⇒ на сайті, що відкриється (рис. 2.1), виберіть пункт **Downloads**;
- ⇒ на сторінці, що відкриється, під написом «Download the latest version for Windows» натисніть кнопку **Download Python 3.7.0**. (Версія може бути новішою, ви також маєте можливість обрати попередні версії, наприклад Python 3.6.6, або Python 3.7.5 тощо. Ми рекомендуємо використовувати останню.);

⇒ далі, у залежності від налаштувань вашого браузера, відбудеться завантаження інсталяційного пакету або вам перед початком скачування потрібно буде обрати папку для завантаження;

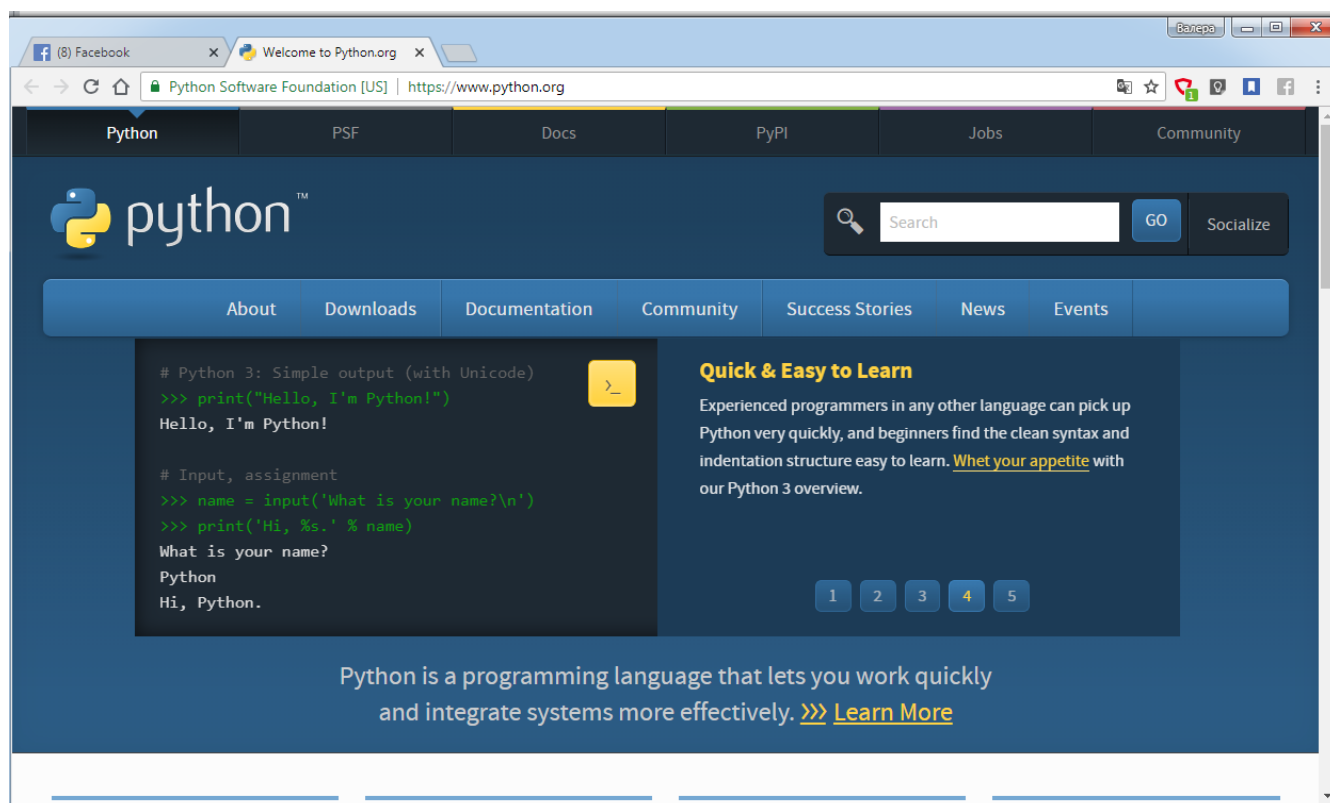
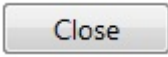


Рис. 2.1

⇒ після завершення завантаження відкриваємо папку та запускаємо файл інсталяції;

⇒ ставимо прапорець біля пункту «Add Python 3.7 to PATH» та обираємо «Install Now»;

⇒ у вікні, що з'явиться натискаєте кнопку <Так>, дозволяючи зробити зміни на комп'ютері;

⇒ чекаєте доки завершиться процес інсталяції та натискаєте кнопку ;

Можете приступати до використання Python.

Питання для самоконтролю

1. Де можна завантажити інсталяційний пакет Python?
2. Які особливості інсталяції Python в операційній системі Microsoft Windows?

3. Інтегроване середовище IDLE. Перша програма

При інсталяції Python також встановлюється інтегроване середовище для розробки та навчання IDLE (Python's Integrated Development and Learning Environment). Щоб його запустити у операційній системі Microsoft Windows потрібно обрати пункт меню Пуск / Все программы / Python 3.7 / IDLE (Python 3.7 32-bit). У результаті відкриється вікно командної оболонки Python (рис. 3.1), яка входить до IDLE, а три знаки «більше» (>>>) називаються запрошенням. Після запрошення можна вводити різні команди.

У IDLE є два види вікон: вікно програми (рис. 3.2) та вікно консолі (рис. 3.1). У вікні програми можна писати, редагувати, зберігати та запускати програмний код,

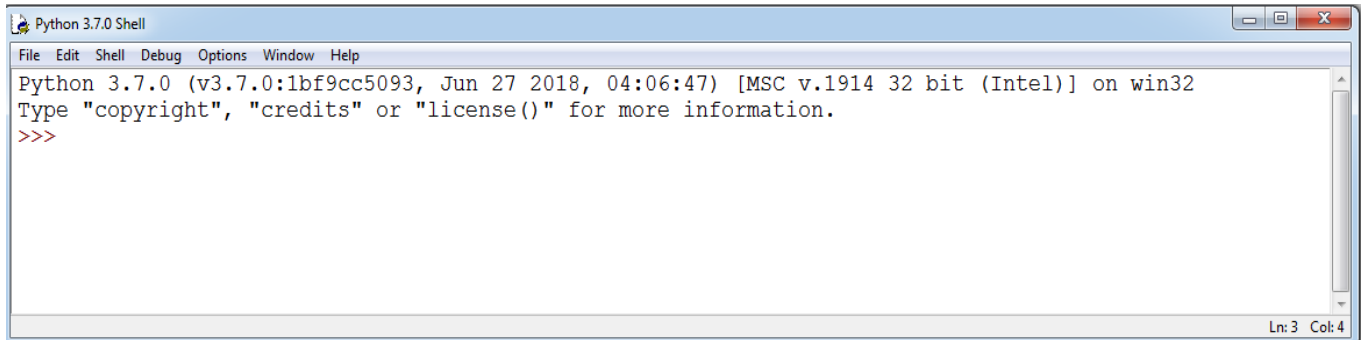


Рис. 3.1

а результат виконання можна переглянути у вікні консолі. У вікні консолі також можна відразу виконувати команди Python.

Вікно програми добре підходить для великих програм, які потрібно зберігати та редагувати. Це простіше, ніж друкувати команди кожного разу, коли вони знадобляться. Вікно консолі ідеально для експериментів – наприклад, якщо потрібно розібратися, що робить команда. Крім того, консоль можна використовувати як калькулятор. Однак, якщо якісь команди потрібно повторювати, краще використовувати вікно програми.

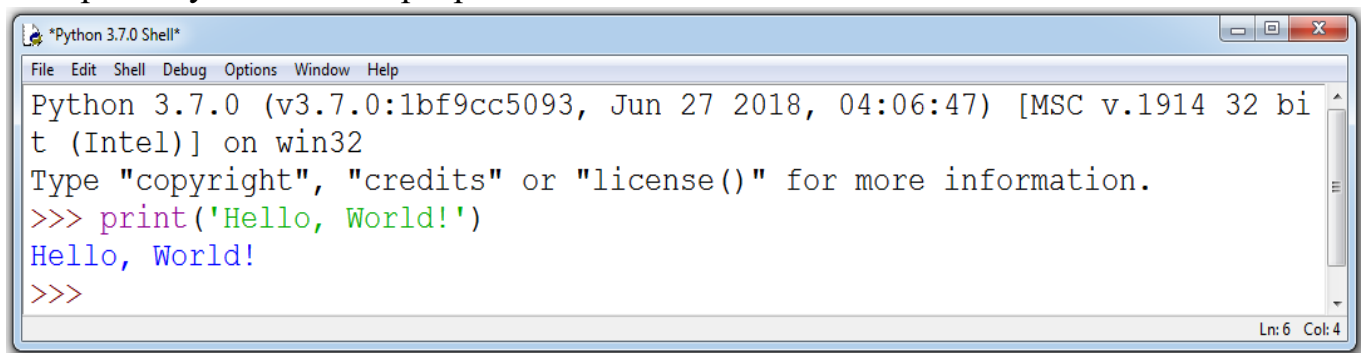


Рис. 3.2

Традиційно при вивченні нової мови програмування першою програмою є програма-вітання «Hello, World!». Ось з неї і почнемо. Для цього:

- ⇒ відкрийте IDLE;
- ⇒ після >>> введіть `print('Hello, World!');`;
- ⇒ натисніть клавішу <Enter>;

⇒ після виконання у новому рядку повинен з'явитися напис **Hello, World!** (рис. 3.2).

У IDLE є два види вікон. У вікні програми можна писати і зберігати програмний код, а у вікні консолі – відразу виконувати команди Python.

А тепер розглянемо як створений код зберегти у окремому файлі на диску та виконати його.

3.1. Практикум. Завдання 3.1

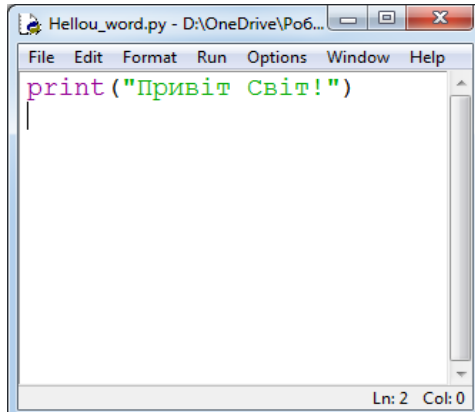


Рис. 3.1.1

Створіть та збережіть на диску програму після виконання якої на екран буде виведено текст «Привіт Світ!». Файл для збереження «Завдання 3-1».

Технологія виконання

1. Запустіть IDLE.
2. Для створення та збереження програми виконайте такі дії:

⇒ оберіть **File** / **New File** **Ctrl+N**;

⇒ відкриється вікно програми (рис. 3.1.1);

⇒ введіть у вікні програми текст `print("Привіт Світ!");`

⇒ щоб зберегти файл оберіть **File** / **Save**;

⇒ у вікні, що відкриється оберіть папку для збереження, введіть ім'я файлу «Завдання 3-1» та натисніть кнопку **Сохранить**;

⇒ для виконання програми оберіть пункт меню **Run** / **Run Module** **F5** або просто натисніть клавішу <F5>;

⇒ у вікні консолі відобразиться результат виконання програми **Привіт Світ!**.

3. Закрийте вікна програми та консолі.

3.2. Практикум. Завдання 3.2

Створіть та збережіть на диску програму після виконання якої на екран буде виведено текст «Мені подобається Python!». Файл для збереження «Завдання 3-2».

Технологія виконання

1. Запустіть IDLE.
2. Скористайтеся досвідом, набутим під час виконання [Завдання 3.1](#).

3.3. Практикум. Завдання для самостійного виконання

Створіть та збережіть на диску програму (ім'я файлу «Завдання 3-3-с») після виконання якої на екран буде виведено текст «Ура! Я почав вивчати Python!».

Питання для самоконтролю

1. Що таке IDLE?
2. Які є два види вікон у IDLE? Розкажіть про їх призначення.

4. Додаткові можливості IDLE

Як ви вже знаєте, у вікні командної оболонки символи `>>>` означають запрошення ввести команду. Якщо після введення команди натиснути клавішу `<Enter>`, то на наступному рядку відразу відобразиться результат (за умови, що інструкція повертає значення), а далі – запрошення для введення нової команди. При введенні багаторядкової команди після натискання клавіші `<Enter>` редактор автоматично вставитиме відступ і буде очікувати подальшого введення. Щоб повідомити редактору про кінець введення команди, необхідно двічі натиснути клавішу `<Enter>`.

Ви можете самостійно розглянути приклад:

```
>>> for n in range(1, 3):  
    print(n)
```

```
1
```

```
2
```

У прикладі використовуються цикл `for` (про нього ви дізнаєтеся пізніше) та функції `range()` (див. Додаток 1. Функції).

Раніше ми виводили рядок "Привіт, Світ!" за допомогою функції `print()`. У вікні **Python Shell** це робити не обов'язково. Наприклад, ми можемо просто ввести рядок і натиснути клавішу `<Enter>` для отримання результату:

```
>>> "Привіт, Світ!"  
'Привіт, Світ!'  
>>>
```

Зверніть увагу на те, що рядки виводяться з апострофами. Цього не станеться, якщо виводити рядок за допомогою функції `print()`.

З огляду на можливість отримати результат відразу після введення команди, вікно **Python Shell** можна використовувати для вивчення команд, а також в якості багатофункціонального калькулятора. Наприклад:

```
>>> 7*(11+8)+17  
150  
>>> 7+50/5  
17.0  
>>>
```

Зверніть увагу, що при використанні ділення у числовому виразі результатом буде дійсне число у якому замість коми (до якої ми звикли при вивченні математики) використовується крапка.

Результат обчислення останньої інструкції зберігається у змінній `_` (одне підкреслення). Це дозволяє проводити подальші розрахунки без введення попереднього результату. Замість нього досить ввести символ підкреслення. Наприклад:


```
>>> 35*4-20
120
>>> _*7
840
>>> _/10
84.0
>>>
```

При введенні команди можна скористатися комбінацією клавіш <Ctrl> + <Пробіл>. В результаті буде відображено список, з якого можна вибрати потрібний ідентифікатор. Якщо при відкритому списку вводити літери, то будуть з'являтися ідентифікатори, що починаються з цих букв. Вибирати ідентифікатор необхідно за допомогою клавіш <↑> і <↓>. Після вибору не слід натискати клавішу <Enter>, інакше це призведе до виконання інструкції. Просто вводьте інструкцію далі, а список закриється.

Такий же список буде автоматично з'являтися (з деякою затримкою) при зверненні до атрибутів об'єкта або модуля після введення точки. Для автоматичного завершення ідентифікатора після введення перших букв можна скористатися комбінацією клавіш <Alt> + </>.

Для пришвидшення процесу введення коду можна використовувати комбінації клавіш швидкого доступу – <Ctrl> + <C> (копіювати) і <Ctrl> + <V> (вставити). Комбінації стандартні для Windows, і ви напевно їх вже використовували раніше. Але знов-таки, перш ніж скопіювати інструкцію, її попередньо необхідно виділити. Редактор IDLE позбавляє нас від зайвих дій і надає комбінацію клавіш <Alt> + <N> для вставки першої введеної інструкції, а також комбінацію <Alt> + <P>, щоб додати останню інструкцію. Кожне наступне натискання цих клавіш буде вставляти наступну (або попередню) інструкцію. Для ще більш швидкого повторного введення інструкції слід попередньо ввести її перші літери. В цьому випадку перебиратися будуть тільки інструкції, що починаються з цих букв.

Зауваження. При використанні у середовищі *Python Shell* комбінацій клавіш мова повинна бути виставлена англійська.

4.1. Практикум. Завдання 4.1

За допомогою інтегрованого середовища IDLE обчисліть значення виразів:

- a) $\frac{7+9-4}{5}$;
- b) $7,77 \cdot 27,8 - 348:34 + 10$;
- c) $(45 - 67) \cdot 34 + 100$;
- d) $(7,2 + 2,4)^7:5,6$

Технологія виконання

1. Запустіть IDLE.
2. Скористайтесь знаннями та навичками набутими під час вивчення [пункту 4](#).

1.1. Практикум. Завдання для самостійного виконання

За допомогою інтегрованого середовища IDLE обчисліть значення виразів:

- a) $\frac{17-9 \cdot 4}{10} + 9$;
b) $7,77 \cdot (27,8 + 3,4) : 20 - 10$;
c) $9,8 \cdot (6,3 - 3,5^4)$.

Питання для самоконтролю

1. Як можна використовувати вікно Python Shell?
2. З якою метою в IDLE використовується комбінація клавіш <Ctrl> + <Пробіл>?
3. Якою повинна бути виставлена мова при використанні у середовищі Python Shell комбінацій клавіш?

5. Початкові відомості про синтаксис мови Python

Синтаксис мови Python, як і сама мова, доволі простий. [6]

- Кінець рядка є кінцем інструкції (крапка з комою не потрібна).
- Вкладені інструкції об'єднуються в блоки за величиною відступів. Відступ може бути будь-яким, головне, щоб в межах одного вкладеного блоку відступ був однаковий. І про читабельність коду не забувайте. Відступ в 1 пробіл, наприклад, не найкраще рішення. Використовуйте 4 пробіли.
- Вкладені інструкції в Python записуються за одним і тим же шаблоном, а саме – основна інструкція завершується двокрапкою, слідом за якою розташовується вкладений блок коду, зазвичай з відступом під рядком основної інструкції.
- Іноді декілька інструкцій можна записувати в одному рядку, розділяючи їх крапкою з комою але краще цього не робити.
- Є можливим запис однієї інструкції у кількох рядках. Для цього її достатньо взяти у круглі дужки.

З іншими особливостями синтаксису мови Python будемо знайомитися пізніше під час її вивчення.

Питання для самоконтролю

1. Чи потрібна крапка з комою у мові Python в кінці інструкції?
2. Яким чином у мові програмування Python вкладені інструкції об'єднуються в блоки?
3. Чи є можливим у Python запис однієї інструкції у кількох рядках?

6. Коментарі

Коментарі призначені для вставки пояснень в текст програми. Інтерпретатор їх повністю ігнорує. В середині коментаря може розташовуватися будь-який текст, включаючи інструкції, які виконувати не слід. Пам'ятайте, коментарі потрібні програмісту, а не інтерпретатору Python. Вставка коментарів в код дозволить через деякий час швидко згадати призначення фрагмента коду.

У мові Python присутній тільки однорядковий коментар. Він починається з символу #. Однорядковий коментар може починатися не тільки з початку рядка, а й розташовуватися після інструкції. Якщо символ коментаря розмістити перед

інструкцією, то вона також не буде виконана. Щоб продемонструвати це скористаємося файлом, *Hello_world*. Відкриємо його за допомогою інтегрованого середовища IDLE та збережемо під ім'ям *Komentar*, а потім додамо рядок коду, який наведено нижче:

```
print("Привіт, Світ!")  
# Це звичайний коментар  
або  
print("Привіт, Світ!") # Це звичайний коментар
```

Якщо ми виконаємо змінену програму, то побачимо, що додані коментарі на результат не вплинули.

Питання для самоконтролю

1. Яке призначення коментарів у мові Python?
2. Які правила додавання коментарів?

7. Модуль turtle (Черепашка)

Графічний модуль `turtle` (Черепашка) створений для тих хто починає вивчати програмування. За його допомогою можна малювати різні фігури. При роботі з графікою `turtle` можна написати інструкції для віртуальної Черепашки, що повідомлятимуть їй про те, як переміщатися по екрану. Черепашка має ручку (перо), а користувач може проінструктувати черепашку (скласти відповідну програму) яким чином скористатися цією ручкою для малювання ліній під час переміщення по екрану.

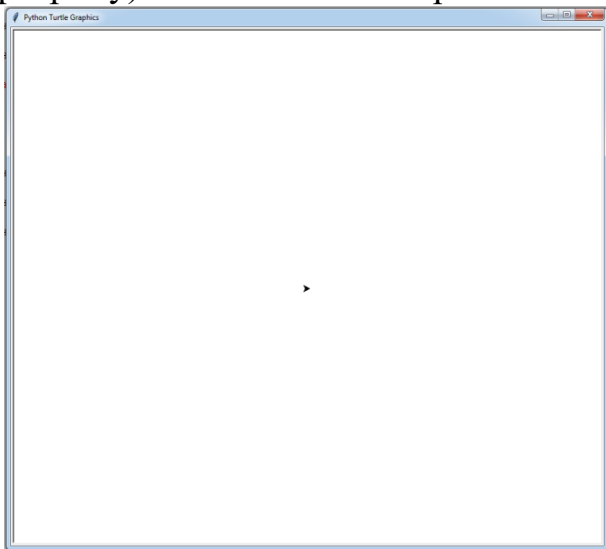


Рис. 7.1

Використання графіки `turtle` дає можливість не тільки створювати малюнки та орнаменти за допомогою всього лише кількох рядків коду, але також дозволяє відстежувати рух черепашки, щоб зрозуміти, яким чином кожний рядок коду впливає на траєкторію руху черепашки. Це допомагає зрозуміти логіку програмного коду.

Виконавець Черепашка керується командами відносних («вперед-назад» і «вправо-вліво») і абсолютних («перейти у точку з координатами ...») переміщень.

Виконавець є «перо», що залишає слід на площині малювання (полотні). Перо можна підняти, тоді при переміщенні слід залишатися не буде. Крім того, для пера можна встановити товщину і колір. Всі ці функції виконавця забезпечуються модулем `turtle` ([документація з модуля turtle \(Черепашка\)](#)).

Наведений нижче код створює графічне вікно (рис. 7.1) і поміщає перо («Черепашку») у початкове положення.

```
from turtle import*  
reset()  
exitonclick()
```

Команда `from turtle import*` використовується для підключення модуля Черепашки (див. «Додаток 3. Модулі у Python»). Команда `reset()` повертає Черепашку (стрілочка у центрі) до початкового стану: очищається полотно, скидаються всі параметри, а Черепашка встановлюється в початок координат, дивлячись вправо. Виконання інструкції `exitonclick()` призводить до закриття графічного вікна при натисканні лівої кнопки мишки.

Ідея малювання полягає у переміщенні пера (Черепашки) у точки полотна з вказаними координатами або у зазначених напрямках на задані відстані, а також у проведенні відрізків прямих, дуг та кіл.

Поточний напрямок переміщення пера (що відповідає напрямку «вперед») вказується вістрям стрілки зображення Черепашки.

Список команд управління «черепашкою» (і, відповідно, малювання), а також функцій, які забезпечуються модулем, можна отримати, набравши команду `help('turtle')`. Список цей досить довгий, а серед функцій є також математичні, оскільки вони можуть бути затребувані при обчисленні параметрів відрізків, дуг та кіл. Наведемо, команди Черепашки, які ми будемо використовувати у посібнику (таблиця 7.1).

Таблиця 7.1

<i>Синтаксис</i>	<i>Призначення</i>	<i>Приклади використання</i>
Команди переміщення		
<code>forward(n)</code> <code>fd(n)</code> (<code>forward</code> – вперед)	Переміщає Черепашку вперед на n кроків (пікселів).	<code>forward(10)</code> <code>fd(50)</code>
<code>back(n)</code> <code>bk(n)</code> (<code>back</code> – назад)	Переміщає Черепашку назад на n кроків (пікселів).	<code>back(60)</code> <code>bk(70)</code>
<code>right(k)</code> <code>rt(k)</code>	Поворот направо (за годинниковою стрілкою) на k одиниць.	<code>right(90)</code> <code>fd(50)</code> <code>rt(90)</code>
<code>left(k)</code> <code>lt(k)</code>	Поворот наліво (проти годинникової стрілки) на k одиниць.	<code>fd(50)</code> <code>left(45)</code> <code>forward(100)</code> <code>lt(45)</code>
<code>circle(r)</code>	Малювання кола радіусом $ r $ точок з поточної позиції пера. Якщо r додатне, коло малюється проти годинникової стрілки, якщо від'ємне – за годинниковою стрілкою.	<code>circle(20)</code> <code>circle(-40)</code>
<code>circle(r, k)</code>	Малювання дуги радіусом $ r $ точок і кутом k одиниць. Варіант команди <code>circle()</code> .	<code>circle(45, 60)</code> <code>circle(-70, 180)</code>
<code>goto(x, y)</code> <code>setpos(x, y)</code>	Переміщення пера (Черепашки) у точку з координатами (x, y) в	<code>goto(5, -45)</code> <code>setpos(-100, 100)</code>

	системі координат полотна.	
<code>setx(x)</code>	Встановити x-координату Черепашки.	<code>setx(50)</code>
<code>sety(y)</code>	Встановити y-координату Черепашки.	<code>sety(100)</code>
<code>home()</code>	Повернути черепашку додому – у точку, з координатами (0, 0).	<code>home()</code>
<code>setheading(angle)</code>	Повернути Черепашку під кутом <code>angle</code> до вертикалі (0 – наверх, 90 – направо).	<code>setheading(30)</code>
<code>speed(v)</code> (<code>speed</code> – швидкість)	Встановити швидкість черепашки. <code>v</code> має бути від 1 (повільно) до 10 (швидко), або 0 (миттєво).	<code>speed(1)</code>
<code>delay(z)</code>	Встановлює затримку малювання в мілісекундах. Чим більше <code>z</code> , тим повільніше рухається Черепашка. ($z \geq 0$)	<code>delay(25)</code>
Команди малювання		
<code>down()</code> (<code>down</code> – вниз)	Опускає перо (по замовчанню – перо опущене). Після цієї команди Черепашка почне залишати слід при будь-якому своєму пересуванні.	<code>home()</code> <code>down()</code>
<code>up()</code> (<code>up</code> – вгору)	Піднімає перо.	<code>up()</code>
<code>dot(d[, 'color'])</code>	Малює точку діаметра <code>d</code> кольору <code>color</code> . Параметр <code>color</code> необов'язковий.	<code>dot(30, 'red')</code>
<code>width(n)</code> (<code>width</code> – ширина)	Встановлює ширину пера Черепашки у <code>n</code> пікселів.	<code>fd(50)</code> <code>width(10)</code> <code>fd(50)</code>
<code>color('color')</code> (<code>color</code> – колір)	Встановлює колір Черепашки. Колір вказується у дужках англійською мовою у верхніх лапках – 'red'. (Див. Додаток 5. Перелік кольорів .)	<code>color('green')</code> <code>fd(50)</code>
<code>color('color', 'fillstring')</code>	Дозволяє змінити кольори ліній та заливки.	
<code>fillcolor('color')</code>	Встановлює колір зафарбовуваної області.	<code>fillcolor('red')</code>
<code>begin_fill()</code> та <code>end_fill()</code>	Використовуються для малювання зафарбованих областей. Перед початком малювання вказується команда <code>begin_fill()</code> , а у кінці – <code>end_fill()</code> .	<code>begin_fill()</code> <code>fd(100); lt(90)</code> <code>fd(100); lt(90)</code> <code>fd(100); lt(90)</code> <code>fd(100); lt(90)</code> <code>end_fill()</code>

<code>hideturtle()</code>	Сховати Черепашку.	<code>hideturtle()</code>
<code>showturtle()</code>	Показати Черепашку.	<code>showturtle()</code>
Інформація про Черепашку		
<code>position()</code> <code>pos()</code>	Отримати поточні координати Черепашки.	<code>x,y=pos()</code> <code>print(x, y)</code>
<code>xcor()</code>	Отримати x координату Черепашки.	<code>b=xcor()</code>
<code>ycor()</code>	Отримати y координату Черепашки.	<code>y=ycor()</code> <code>print(y)</code>
Інші команди		
<code>radians()</code>	Встановлює одиниці вимірювання кутів у радіанах.	<code>radians()</code> <code>lt(1.7)</code>
<code>degrees()</code>	Встановлює одиниці вимірювання кутів у градусах (увімкнений за замовчуванням).	<code>degrees()</code> <code>lt(90)</code>
<code>undo()</code>	Відмінняє попередню дію Черепашки.	<code>undo()</code>
<code>reset()</code> (reset – скинути)	Повертає Черепашку (стрілочка по центру) до початкового стану: очищається полотно, скидаються всі параметри, Черепашка встановлюється у початок координат, дивлячись вправо.	<code>reset()</code>
<code>clear()</code>	Очищає полотно. Черепашка залишається на попередньому місці.	<code>fd(50)</code> <code>clear()</code>
<code>write(s, move, align, font)</code>	Виводить текстовий рядок s у точці знаходження Черепашки. Параметр move може приймати значення True або False (по замовченню move=False). move=False – Черепашка залишається на місці. move=True – Черепашці переміщається у кінець тексту. Параметр align вирівнює текст відносно початкового положення Черепашки (left, center або right). Font – визначає характеристики шрифту (Назва, Розмір, Тип).	<code>write('Всім привіт!!!', move=True, align="center", font=("Arial", 45, "normal"))</code>

<code>exitonclick()</code>	Призводить до закриття графічного вікна при натисканні лівої кнопки мишки.	<code>from turtle import* reset() shape("triangle") exitonclick()</code>
<code>done()</code> (done – зроблено)	Остання команда у програмі для Черепашки. Не слід використовувати, якщо сценарій запускається з IDLE.	<code>done()</code>
<code>shape()</code> (shape – форма)	Дозволяє змінити форму Черепашки. Існують такі варіанти: "arrow" (стрілка), "turtle" (черепашка), "circle" (коло), "square" (квадрат), "triangle" (трикутник), "classic" (класичний).	<code>from turtle import* reset() shape("triangle") exitonclick()</code>
<code>x, y = pos()</code>	Поточні координати присвоюються змінним x та y.	<code>setpos(-100,100) x, y = pos() goto(x+30, y-70)</code>
<code>setup(n,m)</code> (setup – установка, налагодження)	Встановлює розміри екрану довжиною n пікселів та шириною m.	<code>setup(1000,500)</code>

А тепер перейдемо до створення програм.

7.1. Практикум. Завдання 7.1 (Черепашка)

Створіть та збережіть на диску програму, після виконання якої, на полотні Python Turtle Graphics за допомогою модуля Черепашка буде побудовано прямокутник (розміри сторін оберіть довільні, наприклад, 150 і 80, рис. 7.1.1).

```
from turtle import*
reset()
fd(150)
rt(90)
fd(80)
rt(90)
fd(150)
rt(90)
fd(80)
rt(90)
exitonclick()
```

Рис. 7.1.2


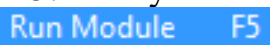
- Технологія виконання
1. Запустіть IDLE.
 2. Створіть новий файл та збережіть його під іменем «Завдання 7-1».
 3. Введіть текст програми (рис. 7.1.2).
 4. Збережіть зміни.
 5. Запустіть програму на виконання обравши  /  F5 або просто натиснувши клавішу <F5>.

Рис. 7.1.1



6. У вікні Python Turtle Graphics перегляньте результати виконання.
7. При потребі зробіть корективи у програмі.
8. Досягши шуканого результату збережіть зміни та закрийте IDLE.

7.2. Практикум. Завдання 7.2 (Черепашка)

Створіть та збережіть на диску програму, після виконання якої, на полотні Python Turtle Graphics за допомогою модуля Черепашка буде побудовано прямокутник згідно зразка, наведеного на рисунку 7.2.1 (розміри оберіть довільними).

Технологія виконання

1. Запустіть IDLE. Для спрощення виконання відкрийте файл, що є результатом виконання завдання 7.1 та збережіть його під ім'ям «Завдання 7-2».

2. Для збільшення товщини лінії та кольору потрібно після команди `reset()` додати команди `width(7)` та `color('red')` відповідно. Щоб здійснити заливку прямокутника зеленим кольором після команди `color('red')` необхідно вставити інструкції `fillcolor('green')` та `begin_fill()`, а перед командою `exitonclick()` інструкцію `end_fill()`.



Рис. 7.2.1

3. Збережіть внесені до файлу програми зміни.

4. Запустіть програму на виконання.

5. Перегляньте результати виконання та при потребі зробіть потрібні зміни у програмі.

6. Збережіть зміни та закрийте IDLE.

7.3. Практикум. Завдання 7.3 (Черепашка)

Створіть та збережіть на диску програму, після виконання якої, на полотні Python Turtle Graphics за допомогою модуля Черепашка буде побудовано правильний (рівносторонній) п'ятикутник згідно зразка, наведеного на рисунку 7.3.1 (розмір сторони оберіть на власний розсуд, наприклад, 150).

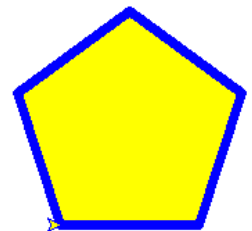


Рис. 7.3.1

Технологія виконання

1. Для спрощення виконання зробіть копію файлу «Завдання 7-2», що є результатом виконання відповідного завдання та змініть його ім'я на «Завдання 7-3».

2. Відкрийте файл «Завдання 7-3» за допомогою середовища IDLE (наприклад, так – оберіть пункт контекстне меню файлу `Edit with IDLE` / `Edit with IDLE 3.7`, або іншим способом).

3. Скориставшись досвідом, набутим під час виконання [завдання 7.2](#) та знаннями зі шкільного курсу математики (градусна міра кутів правильного п'ятикутника дорівнює 108°) зробіть у програмі потрібні зміни та доповнення.

4. Збережіть внесені зміни.

5. Запустіть програму на виконання.

6. Перегляньте результати та при потребі зробіть потрібні корективи. Перевірте, виконавши програму.

7. Збережіть зміни та закрийте IDLE.

7.4. Практикум. Завдання 7.4 (Черепашка)

Створіть програму, після виконання якої, на полотні Python Turtle Graphics за допомогою модуля Черепашка буде побудовано державний прапор України (співвідношення довжини та ширини як 3:2, розміри довільні). Збережіть програму у файлі з ім'ям «Завдання 7-4».

Технологія виконання

1. Відкрийте інтегроване середовище IDLE.
2. Створіть новий файл та збережіть його під ім'ям «Завдання 7-4».
3. Скориставшись досвідом, набутим під час виконання завдань [7.1](#), [7.2](#) та [7.3](#) розробіть алгоритм малювання прапору та введіть відповідний текст програми.
4. Збережіть внесені зміни.
5. Запустіть програму на виконання.
6. Перегляньте результати виконання та при потребі зробіть необхідні корективи. Перевірте, виконавши програму.
7. У випадку виникнення труднощів зверніться до [лістингу програми](#).
8. Збережіть зміни та закрийте IDLE.

Зауваження

Програма буде виконуватися і якщо просто відкрити її файл, наприклад подвійним клацанням мишки.

7.5. Практикум. Завдання 7.5 (Черепашка)

Створіть програму, під час виконання якої, на полотні Python Turtle Graphics за допомогою модуля Черепашка буде побудовано фігуру згідно зразка на відео <https://youtu.be/63YwvzDb0E8>. Екран демонстрації повинен мати розміри 800×600 пікселів. Підчас побудови виконавець повинен змінювати зовнішній вигляд відповідно до наведеного зразка. Збережіть програму у файлі з ім'ям «Завдання 7-5».

Технологія виконання

1. Відкрийте інтегроване середовище IDLE.
2. Створіть новий файл та збережіть його під ім'ям «Завдання 7-5».
3. Скориставшись досвідом, набутим під час виконання завдань 7.1-7.4 розробіть алгоритм та введіть відповідний текст програми. Для зміни зовнішнього вигляду Черепашки використовуйте команду `shape ('форма')`
4. Збережіть внесені зміни.
5. Запустіть програму на виконання.
6. Перегляньте результати виконання та при потребі зробіть необхідні корективи. Перевірте, виконавши програму.
7. У випадку виникнення труднощів зверніться до [лістингу програми](#).
8. Збережіть зміни та закрийте IDLE.

7.6. Практикум. Завдання для самостійного виконання

1. Створіть та збережіть (ім'я файлу «Завдання 7-6-1-с») програму, після виконання якої, на полотні Python Turtle Graphics за допомогою модуля Черепашка буде побудовано квадрат (розміри сторін оберіть довільні, рис. 7.6.1).

- Створіть та збережіть на диску програму (ім'я файлу «Завдання 7-6-2-с»), після виконання якої, на полотні Python Turtle Graphics за допомогою модуля Черепашка буде побудовано квадрат згідно зразка, наведеного на рисунку 7.6.2 (розмір сторони оберіть довільний).
- Створіть та збережіть (ім'я файлу «Завдання 7-6-3-с») програму, після

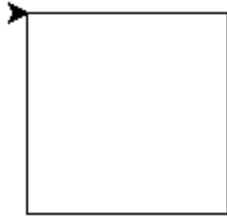


Рис. 7.6.1

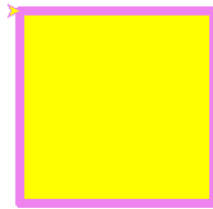


Рис. 7.6.2



Рис. 7.6.3

виконання якої, на полотні Python Turtle Graphics за допомогою модуля Черепашка буде побудовано правильний (рівносторонній) трикутник згідно зразка, наведеного на рисунку 7.6.3 (розмір сторони оберіть довільний).

- Створіть програму, для виконавця Черепашка, після виконання якої, на полотні Python Turtle Graphics буде побудовано фігуру, що складається з 4 квадратів червоного та зеленого кольорів згідно зразка, наведеного на рисунку 7.6.4. Розміри квадратів та товщину ліній підберіть довільним чином. Збережіть її (ім'я файлу «Завдання 7-6-4-с»).

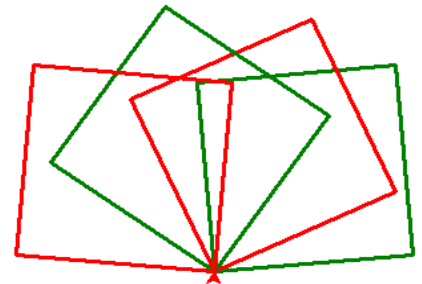


Рис. 7.6.4

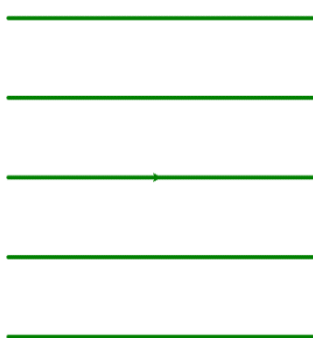


Рис. 7.6.6

5. Створіть та збережіть (ім'я файлу «Завдання 7-6-5-с») програму, після виконання якої, на полотні за допомогою модуля Черепашка буде побудовано горизонтальний відрізок прямої, червоного кольору, довжиною 400 пікселів, не повинно бути подвійних ліній (економія фарби). Після виконання – Черепашка у початковому положенні.

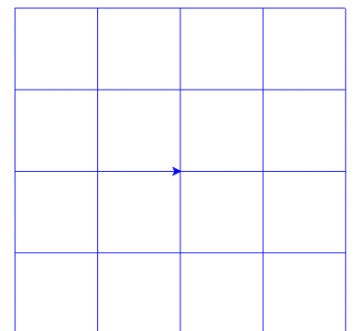


Рис. 7.6.7

6. Створіть та збережіть програму, після виконання якої, на полотні за допомогою модуля Черепашка буде побудовано 5 горизонтальних відрізків прямої зеленого кольору, довжиною 400 пікселів, товщиною 5 пікселів (рисунок 7.6.6). Повинні виконуватися умови економії «фарби» (довжина шляху який Черепашка проходить з опущеним пером повинна бути мінімальною) та «пального» (довжина шляху який черепашка проходить повинна бути як можна меншою). Після виконання –

Черепашка у початковому положенні. (Ім'я файлу для збереження «Завдання 7-6-6-с».)

7. Створіть та збережіть програму, після виконання якої, на полотні за допомогою модуля Черепашка буде побудовано у центрі полотна квадрат згідно зразка, поданого на рисунку 7.6.7, розміром 400×400 пікселів, товщиною лінії 1 піксель. Повинні виконуватися умови економії «фарби» (довжина шляху який черепашка проходить з опущеним пером повинна бути мінімально) та «пального» (довжина шляху який черепашка проходить повинна бути як можна меншою). Після виконання – Черепашка у початковому положенні. (Ім'я файлу для збереження «Завдання 7-6-7-с».)

Питання для самоконтролю

1. Для чого призначений модуль turtle (Черепашка)?
2. Яка інструкція використовується для підключення модуля Черепашки?
3. Яким чином можна отримати весь список команд управління «черепашкою»?
4. Які команди з переміщення Черепашки вам відомі?
5. Перерахуйте відомі вам команди Черепашки для малювання?

8. Змінні

Всі дані в мові Python представлені об'єктами. Кожен об'єкт має тип даних і значення. Для доступу до об'єкта призначені змінні (ідентифікатори). При ініціалізації у змінній зберігається посилання (адреса об'єкта в пам'яті комп'ютера) на об'єкт. Завдяки цим посиланням можна надалі працювати та змінювати об'єкт з програми.

Кожна змінна повинна мати унікальне ім'я, що складається з латинських букв, цифр і знаків підкреслення, причому ім'я змінної не може починатися з цифри. Крім того, слід уникати використання символу підкреслення на початку імені тому, що ідентифікатори з таким символом мають спеціальне значення.

Як ім'я змінної не можна використовувати ключові слова. Отримати список всіх ключових слів (див. також додаток 2) дозволяє код:

```
>>> import keyword
>>> keyword.kwlist
['False', 'None', 'True', 'and', 'as', 'assert', 'async', 'await',
 'break', 'class', 'continue', 'def', 'del', 'elif', 'else', 'except',
 'finally', 'for', 'from', 'global', 'if', 'import', 'in', 'is', 'lambda',
 'nonlocal', 'not', 'or', 'pass', 'raise', 'return', 'try', 'while', 'with', 'yield']
```

Окрім ключових слів слід уникати збігів з вбудованими ідентифікаторами. На відміну від ключових слів, вбудовані ідентифікатори можна перевизначати, але краще цього не робити.

У редакторі IDLE вбудовані ідентифікатори підсвічуються фіолетовим кольором. Звертайте увагу на колір змінної, він повинен бути чорним. Якщо ви помітили, що змінна підсвічена, то назву змінної слід обов'язково змінити. Отримати повний список (див. додаток 2) вбудованих ідентифікаторів дозволяє код:

```
>>> import builtins
>>> dir(builtins)
```

В імені змінної важливо враховувати регістр букв: `x` і `X` – різні змінні, але верхній регістр краще не використовувати.

У мові Python зв'язок між даними і змінними встановлюється за допомогою знаку “=”.

Така операція називається «**присвоюванням**». Наприклад, вираз `a=4` означає, що на об'єкт (дані) у певній області пам'яті посилається ім'я **a** і звертатися до нього тепер слід за цим іменем.

Щоб дізнатися значення, на яке посилається змінна, перебуваючи в режимі інтерпретатора, достатньо її викликати (написати ім'я і натиснути клавішу <Enter>). Розглянемо приклад роботи зі змінними в інтерактивному режимі:

```
>>> a=7
>>> b=5
>>> c=a+b
>>> c
12
>>> b=b+4
>>> b
9
```

У мові Python використовується динамічна типізація. Це означає, що при присвоєнні змінній значення інтерпретатор автоматично відносить змінну до одного з типів даних.

В одному рядку можна присвоїти значення зразу декільком змінним:

```
>>> a=b=c=7
>>> a
7
>>> b
7
>>> c
7
```

Після надання значення у змінній зберігається посилання на об'єкт, а не сам об'єкт. Це обов'язково слід враховувати при груповому присвоєнні. Групове присвоювання можна використовувати для чисел, рядків і кортежів, але для змінюваних об'єктів цього робити не можна.

Крім групового присвоювання мова Python підтримує позиційне присвоювання. У цьому випадку змінні вказуються через кому зліва від оператора «=», а значення – через кому справа. Приклад позиційного присвоювання:

```
>>> a, b, c=3, 7, 10
>>> a
3
>>> c
10
>>> b
7
```

Питання для самоконтролю

1. З чого може складатися ім'я змінної у мові Python?
2. Що означає динамічна типізація?
3. Яким чином у Python здійснюється позиційне присвоювання?
4. Чи важливо враховувати регістр в іменах змінних у мові програмування Python?
5. Яке призначення операції присвоювання?

9. Типи даних у мові Python

Якщо вам доводилося використовувати мови програмування, такі як C або C++, то ви вже знаєте, що значна частка роботи перепadaє на реалізацію об'єктів, відомих також як структури даних, які призначені для подання складових предметної області. У таких мовах програмування необхідно займатися опрацюванням структур даних, управляти виділенням пам'яті, реалізовувати функції пошуку і доступу до елементів структур і так далі. Це досить складно (і сприяє появі помилок) і, як правило, відволікає від досягнення істинних цілей.

У типових програмах на мові Python в цьому немає необхідності. Python надає потужну колекцію об'єктних типів, вбудованих безпосередньо у мову, тому зазвичай немає ніякої необхідності створювати власні реалізації об'єктів, призначених для вирішення поставлених завдань. Фактично якщо у вас немає потреби у спеціальних видах обробки, які не забезпечуються вбудованими типами об'єктів, вам завжди краще використовувати вбудовані об'єкти замість реалізації своїх власних.

Іншими словами, вбудовані типи об'єктів не тільки спрощують процес програмування, але вони мають більшу ефективність і продуктивність, ніж більшість типів, створених вручну. Навіть якщо ви створюєте власні типи об'єктів, вбудовані об'єкти будуть ядром будь-якої програми на Python.

Наведемо типи даних які можуть мати об'єкти у мові Python 3.

- `int` – цілі числа. Розмір числа обмежений лише обсягом оперативної пам'яті:

```
>>> type(3895640)
<class 'int'>
>>>
>>> type(-39777259799)
<class 'int'>
```

- `float` – дійсні числа:

```
>>> type(7.41)
<class 'float'>
>>> type(2.1775e-5)
<class 'float'>
```

- `complex` – комплексні числа:

```
>>> type(3+5j)
<class 'complex'>
```

- `str` – Unicode-рядки:

```
>>> type("Це приклад рядка.")
<class 'str'>
```

- bool - логічний тип даних. Може містити значення True або False, які ведуть себе як числа 1 і 0 відповідно:

```
>>> type(True)
<class 'bool'>
>>> type(False)
<class 'bool'>
>>> int(True)
1
>>> int(False)
0
```

- list - списки. Тип даних list схожий на масиви в інших мовах програмування:

```
>>> type([7, 7, 3, -4])
<class 'list'>
```

tuple - кортежі:

```
>>> type((7, 2, 5))
<class 'tuple'>
```

- dict - словники. Тип даних dict аналогічний асоціативним масивам в інших мовах програмування:

```
>>> type({"x": 5, "y": 20})
<class 'dict'>
```

- set - множини (колекції унікальних об'єктів):

```
>>> type({"p", "в", "м"})
<class 'set'>
```

- frozenset - незмінні множини:

```
>>> type(frozenset(["m", "b", "c"]))
<class 'frozenset'>
```

- bytes - незмінювана послідовність байтів:

```
>>> type(bytes("Це рядок", "utf-8"))
<class 'bytes'>
```

- function - функції:

```
>>> def func(): pass

>>> type(func)
<class 'function'>
```

- module - модулі:

```
>>> import sys
>>> type(sys)
<class 'module'>
```


- `type` – класи та типи даних. Не дивуйтесь! Всі дані в мові Python є об'єктами, навіть самі типи даних!

```
>>> class C: pass
```

```
>>> type(C)
<class 'type'>
```

- `NoneType` – об'єкт зі значенням `None` (позначає відсутність значення):

```
>>> type(None)
<class 'NoneType'>
```

Основні типи даних діляться на змінювані і незмінювані. До змінюваних типів належать списки, словники і тип `bytearray`. Приклад зміни елемента списку:

```
>>> arr = [7, 2, 3, 10]
>>> arr[2]=9
>>> arr
[7, 2, 9, 10]
```

До незмінюваних типів належать числа, рядки, кортежі і тип `bytes`.

У мові Python використовується динамічна типізація. Після присвоєння значення у змінній зберігається посилання на об'єкт певного типу, а не сам об'єкт. Якщо потім змінній присвоїти значення іншого типу, то змінна буде посилатися на інший об'єкт, і тип даних відповідно зміниться. Таким чином, тип даних в мові Python – це характеристика об'єкта, а не змінної. Змінна завжди містить тільки посилання на об'єкт.

Після присвоєння змінній значення над об'єктом можна виконувати операції, призначені для цього типу даних. Наприклад, рядок не можна скласти з числом, тому, що це призведе до виводу повідомлення про помилку.

```
>>> 7+"7"
Traceback (most recent call last):
  File "<pyshell#0>", line 1, in <module>
    7+"7"
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

Python у будь-який момент часу змінює тип змінної відповідно до даних, яким вона відповідає. Визначити, на який тип даних посилається змінна, дозволяє функція `type(<Ім'я змінної>)`:

```
>>> a=77
>>> type(a)
<class 'int'>
```

Типи даних можна змінювати за допомогою відповідних функцій. Наприклад, потрібно ціле число записати, як рядок. або, навпаки, якщо рядок складається з цифр, то корисно цей рядок представити у вигляді числа, щоб далі можна було виконувати арифметичні операції з ним. Для цього використовуються функції, однойменні з ім'ям типу, тобто `int`, `float`, `str`. Наприклад, `int('757')` поверне ціле число 757, а `str(777)` поверне рядок '777'.

```
>>> str(777)
'777'
>>> int('757')+10
767
```

Питання для самоконтролю

1. Які переваги вбудованих у мову Python типів?
2. Перерахуйте відомі вам типи даних, що можуть мати об'єкти у мові програмування Python 3? Наведіть приклади.
3. Яким чином можна змінювати типи даних у мові Python? Наведіть приклади.

10. Виведення результатів роботи програми

Як ви вже знаєте, вивести результати роботи програми можна за допомогою функції `print()`.

Функція має наступний формат:

```
print([<Об'єкти>][, sep=' '][, end='\n'][, file=sys.stdout])
```

Функція `print()` перетворює об'єкт в рядок і посилає її в стандартний вивід `stdout`. За допомогою параметра `file` можна перенаправити вивід в інше місце, наприклад в файл. Перенаправлення виводу ми детальніше розглянемо при вивченні файлів.

Після виведення рядка автоматично додається символ переходу на новий рядок (`'\n'`):

```
print("Рядок 1")
print("Рядок 2")
```

Результат:

```
Рядок 1
Рядок 2
```

Якщо необхідно вивести результат у тому ж рядку, то у функції `print()` дані вказуються через кому у першому параметрі:

```
print("Рядок 1", "Рядок 2")
```

Результат:

```
Рядок 1 Рядок 2
```

Як видно з прикладу, між виведеними словами автоматично вставляється пробіл. За допомогою параметра `sep` можна вказати інший символ. Виведемо рядки із зірочкою «*» замість пробілу:

```
print("Рядок 1", "Рядок 2", sep="*")
```

Результат:

```
Рядок 1*Рядок 1
```

Після виведення об'єктів в кінці додається символ переходу на новий рядок. Якщо необхідно провести подальше виведення у тому ж рядку, то в параметрі `end` слід вказати інший символ:

```
print("Рядок 1", "Рядок 2", end=" ")
print("Рядок 3")
```

Результат виконання:

Рядок 1 Рядок 2 Рядок 3

Якщо навпаки потрібно вставити символ переходу на новий рядок, то функція `print()` вказується без параметрів. Приклад:

```
for n in range(1, 5):  
    print(n, end=" ")  
print()  
print("Цей текст на новому рядку")
```

Результат отримаємо такий:

```
1 2 3 4  
Цей текст на новому рядку
```

У цьому прикладі ми використовували цикл `for`, який дозволяє послідовно перебирати елементи. На кожній ітерації циклу змінній `n` присвоюється нове число, яке ми виводимо за допомогою функції `print()`, розташованої на наступному рядку. Зверніть увагу, що перед функцією ми додали чотири пробіли. Таким чином у мові Python виділяються блоки. Інструкції, перед якими розташовано однакову кількість пробілів, є тілом циклу. Всі ці інструкції виконуються певну кількість разів. Кінцем блоку є інструкція, перед якою розташовано меншу кількість пробілів. У нашому випадку це функція `print()` без параметрів, яка вставляє символ переходу на новий рядок.

Якщо необхідно вивести великий блок тексту, то його слід розмістити між потроєними лапками або потроєними апострофами. В цьому випадку текст зберігає своє форматування. Приклад:

```
print("""Рядок 1  
Рядок 2  
Рядок 3""")
```

У результаті ми отримаємо 3 рядки тексту:

```
Рядок 1  
Рядок 2  
Рядок 3
```

Для виведення результатів роботи програми замість функції `print()` можна використовувати метод `write()` об'єкта `sys.stdout`:

```
import sys # Під'єднуємо модуль sys  
sys.stdout.write("Рядок") # Виводимо рядок
```

Результат:

```
Рядок
```

У першому рядку, за допомогою оператора `import`, ми підключаємо модуль `sys`, у якому оголошено об'єкт. Далі за допомогою методу `write()` виводимо рядок. Слід зауважити, що метод не вставляє символ переходу на новий рядок. Тому при необхідності слід додати його самим за допомогою символу `\n`:

```
import sys  
sys.stdout.write("Рядок 1\n")  
sys.stdout.write("Рядок 2")
```

Питання для самоконтролю

1. За допомогою якої функції можна вивести результати роботи програми? Який її формат?
2. Яким чином можна вивести великий блок тексту?
3. Що можна використати для виведення результатів роботи програми замість функції `print()`?

11. Введення даних

Для введення даних у Python 3 призначена функція `input()`. Вона призупиняє виконання програми і чекає, доки користувач введе деякий текст. Отримавши дані, Python зберігає їх у змінній, щоб вам було зручніше працювати з ними. Функція має наступний формат (взяті у квадратні дужки частини є необов'язковими):

```
[<Змінна> = ] input([<Повідомлення>]) .
```

Кожен раз, коли у вашій програмі використовується функція `input()`, доцільно включати чітку, зрозумілу підказку, яка точно повідомить користувачеві, яку інформацію ви від нього хочете отримати.

Розглянемо приклад:

```
name=input("Введіть ваше ім'я та натисніть клавішу <Enter> ")
print("Привіт, ", name, "!", sep="")
```

В результаті ми отримаємо наступне :

```
Введіть ваше ім'я та натисніть клавішу <Enter> Валерій
Привіт, Валерій!
>>>
```

При використанні функції `input()` Python інтерпретує всі дані, введені користувачем, як рядок. Якщо нам потрібно щоб змінні мали значення іншого типу, то після введення ми можемо змінити тип, наприклад на `int` або `float`.

```
>>> n=input()
55
>>> print(n)
55
>>> type(n)
<class 'str'>
>>> n=int(n)
>>> type(n)
<class 'int'>
```

Можна об'єднати зчитування рядків і перетворення типів, якщо викликати функцію `int` (або, наприклад, `float`) для того значення, яке поверне функція `input`:

```
>>> b=int(input('Введіть значення змінної b: '))
Введіть значення змінної b: 77
>>> print('b =', b)
b = 77
>>> b=float(input('Введіть значення змінної b: '))
Введіть значення змінної b: 77
>>> print('b =', b)
У цьому випадку результат буде:
b= 77.0
```

Питання для самоконтролю

1. За допомогою якої функції можна ввести дані у Python 3? Який її формат?
2. Яким чином Python інтерпретує всі дані при використанні функції `input()`?
3. Як можна об'єднати зчитування рядків і перетворення типів?

12. Числа у мові Python

Мова Python 3 підтримує наступні числові типи:

- `int` – цілі числа. Розмір числа обмежений лише обсягом оперативної пам'яті;
- `float` – дійсні числа;
- `complex` – комплексні числа (наразі у посібнику розглядати не планується).

Операції над числами різних типів повертають число, що має більш складний тип з типів, що беруть участь в операції. Цілі числа мають найпростіший тип, далі йдуть дійсні числа і найскладніший тип – комплексні числа. Таким чином, якщо в операції беруть участь ціле і дійсне числа, то ціле число буде автоматично перетворено в дійсне число, а потім проведена операція над числами. Результатом цієї операції буде дійсне число.

Створити об'єкт цілочисельного типу можна звичайним способом:

```
>>> a=10
>>> b=-7
>>> print(a, b)
10 -7
```

Крім того, можна вказати число в двійковій, вісімковій або шістнадцятковій формі. Такі числа будуть автоматично перетворені в десяткові цілі числа. Двійкові числа починаються з комбінації символів `0b` (або `0B`) і містять цифри 0 та 1:

```
>>> x=0b111
>>> print(x)
7
```

Вісімкові числа починаються з нуля і наступної за ним латинської літери `o` (реєстр не має значення) і містять цифри від 0 до 7:

```
>>> 0o25
21
>>> 0o77
63
```

Шістнадцяткові числа починаються з комбінації символів 0x (або 0X) і можуть містити цифри від 0 до 9 і букви від A до F (реєстр букв не має значення):

```
>>> 0x9, 0xB, 0x10, 0xCCC, 0xcscs
(9, 11, 16, 3276, 3276)
```

Дійсне число може містити точку і (або) бути представлено в експоненційній формі з літерою E (реєстр не має значення):

```
>>> 17., .44, 5.14, 17E15, 2.5e-14
(17.0, 0.44, 5.14, 1.7e+16, 2.5e-14)
```

12.1. Математичні оператори

Для виконання операцій над числами використовуються математичні оператори. Почнемо з операторів для роботи з цілими (int) числами (таблиця 12.1.1).

Таблиця 12.1.1

Синтаксис	Назва та призначення	Приклади використання
$a + b$	Додавання	<pre>>>> 5+10 15 >>> -5+10 5 >>> 7+(-7) 0</pre>
$a - b$	Віднімання	<pre>>>> 15-8 7 >>> 4-10 -6</pre>
$a * b$	Множення	<pre>>>> -5*7 -35 >>> 4*7 28</pre>
a / b	Ділення	<pre>>>> 16/2 8.0 >>> -5/3 -1.6666666666666667</pre>
$a // b$	Ціла частина від ділення	<pre>>>> 25//2 12 >>> -4//2 -2 >>> -5//2 -3</pre>
$a \% b$	Остача від ділення	<pre>>>> 8%4 0 >>> 9%4 1</pre>
$a ** b$	Піднесення до степені	<pre>>>> 2**4 16 >>> -4**3 -64 >>> -2**2 -4 >>> (-2)**2 4</pre>

-a	Унарний мінус	>>> a=-5 >>> a=-a >>> a 5
+a	Унарний плюс	>>> a=-5 >>> a=+a >>> a -5

А тепер перейдемо до операторів для роботи з дійсними числами (таблиця 12.1.2).

Таблиця 12.1.2

<i>Синтаксис</i>	<i>Назва та призначення</i>	<i>Приклади використання</i>
a + b	Додавання	>>> 15.6+17.9 33.5 >>> 27+7.5 34.5
a - b	Віднімання	>>> 17.2-25.5 -8.3 >>> 7-3.4 3.6
a * b	Множення	963.9000000000001 >>> 7.45*5.4 40.230000000000004 >>> 22*7.2 158.4
a / b	Ділення	>>> 10/2.5 4.0 >>> 0.75/3.5 0.21428571428571427 >>> 2.5/5 0.5
a // b	Ціла частина від ділення	>>> 7.0//2 3.0 >>> 7.1//2 3.0 >>> 7.5//2 3.0 >>> 7.7//2 3.0
a % b	Остача від ділення	>>> 10.0%2 0.0 >>> 10.1%2 0.09999999999999964 >>> 10.7%2 0.6999999999999993

$a**b$	Піднесення до степені	<pre>>>> 10.5**1.5 34.02388866664127 >>> 25**0.5 5.0 >>> 25**(1/2) 5.0 >>> 0.5**3 0.125 >>> -0.5**2 -0.25 >>> (-0.5)**2 0.25 >>> 4.0**(-1/2) 0.5</pre>
$-a$	Унарний мінус	<pre>>>> a=7.5 >>> a=-a >>> a -7.5</pre>
$+a$	Унарний плюс	<pre>>>> a=-7.5 >>> a+=a >>> a -7.5</pre>

При виконанні операцій над числами слід враховувати обмеження точності обчислень. Наприклад, результат наступної операції може здатися дивним:

```
>>> 0.5-0.1-0.1-0.1-0.1-0.1
2.7755575615628914e-17
>>> 0.4-0.1-0.1-0.1-0.1
2.7755575615628914e-17
```

Очікуваним був би результат 0.0, але, як видно з прикладу, ми отримали зовсім інший результат. Якщо необхідно проводити операції з фіксованою точністю, то слід використовувати [модуль](#) decimal:

```
>>> from decimal import *
>>> Decimal('0.5')-Decimal('0.1')-Decimal('0.1')-
Decimal('0.1')-Decimal('0.1')-Decimal('0.1')
Decimal('0.0')
```

12.2. Оператори присвоювання

Розглянемо оператори присвоювання (таблиця 12.2.1).

Таблиця 12.2.1

Синтаксис	Назва та призначення	Приклади використання
$=$	Присвоювання – присвоює змінній значення	<pre>>>> v=10 >>> v 10</pre>
$+=$	Збільшує значення змінної на вказану величину. Для послідовностей оператор $+=$ здійснює конкатенацію	<pre>>>> a=5; a+=7 >>> a 12 >>> b="Vale"; b+="ra" >>> b 'Valera'</pre>
$-=$	Зменшує значення змінної на вказану величину	<pre>>>> c=10; c-=5 >>> c 5</pre>

<code>*=</code>	Множить значення змінної на вказану величину. Для послідовностей оператор <code>*=</code> здійснює повторення	<pre>>>> x=7; x*=5 >>> x 35 >>> y='*'; y*=10 >>> y '*****'</pre>
<code>/=</code>	Ділить значення змінної на вказану величину	<pre>>>> a=7; a/=3 >>> a 2.3333333333333335</pre>
<code>//=</code>	Цілочисельне ділення з присвоюванням	<pre>>>> k=5; k//=3 >>> k 1</pre>
<code>%=</code>	Остача від ділення і присвоювання	<pre>>>> k=17; k%=5 >>> k 2</pre>
<code>**=</code>	Піднесення до степені та присвоювання	<pre>>>> b=7; b**=2 >>> b 49</pre>

12.3. Пріоритет виконання операторів

Черговість виконання операторів у виразі визначається їх пріоритетом та дужками. Спочатку, як і в математиці, виконуються оператори в дужках.

Перерахуємо оператори в порядку убуття пріоритету:

- 1) `-x`, `+x`, `**` – унарний мінус, унарний плюс, піднесення до степені. Якщо унарні оператори розташовані зліва від оператора `**`, то піднесення до степені має більший пріоритет, а якщо справа – то менший. Наприклад, вираз `-10 ** -2` еквівалентний наступній розстановці дужок: `-(10 ** (-2))`.
- 2) `*`, `%`, `/`, `//` – множення (повторення), залишок від ділення, ділення, ціла частина від ділення.
- 3) `-`, `+` – віднімання, додавання (конкатенація).
- 4) `=`, `+=`, `-=`, `*=`, `/=`, `//=`, `%=`, `**=` – присвоювання.

12.4. Вбудовані функції

Для роботи з числами призначені вбудовані функції (таблиця 12.4.1).

Таблиця 12.4.1

<i>Синтаксис</i>	<i>Призначення</i>	<i>Приклади використання</i>
<code>int([<Об'єкт>[, <Система числення>]])</code>	Перетворює об'єкт у ціле число. У другому параметрі можна вказати систему числення (значення за замовчуванням 10).	<pre>>>> int(10.5) 10 >>> int("0b111", 2) 7 >>> int("37", 10) 37</pre>
<code>float([<Число чи рядок>])</code>	Перетворює ціле число або рядок у дійсне число	<pre>>>> float(7) 7.0 >>> float("7.5") 7.5 >>> float(".7") 0.7</pre>

<code>bin(<Число>)</code>	Перетворює десяткове число у двійкове. Повертає рядкове представлення числа.	<pre>>>> bin(8) '0b1000' >>> bin(-11) '-0b1011'</pre>
<code>oct(<Число>)</code>	Перетворює десяткове число у вісімкове. Повертає рядкове представлення числа.	<pre>>>> oct(5) '0o5' >>> oct(67) '0o103'</pre>
<code>hex(<Число>)</code>	Перетворює десяткове число у шістнадцяткове. Повертає рядкове представлення числа.	<pre>>>> hex(10) '0xa' >>> hex(37) '0x25'</pre>
<code>round(<Число>[, <Кількість знаків після коми>])</code>	Повертає число, округлене до найближчого меншого цілого для чисел з дробовою частиною менше 0.5, або значення, округлене до найближчого більшого цілого для чисел з дробовою частиною більше 0.5. Якщо дробова частина дорівнює 0.5, то округлення проводиться до найближчого парного числа. У другому параметрі можна вказати кількість знаків у дробовій частині. Якщо параметр не вказано, то використовується значення 0.	<pre>>>> round(0.49) 0 >>> round(0.50) 0 >>> round(0.51) 1 >>> round(1.49) 1 >>> round(1.50) 2 >>> round(1.51) 2 >>> round(2.49) 2 >>> round(2.50) 2 >>> round(2.51) 3 >>> round(3.49) 3 >>> round(3.50) 4 >>> round(3.51) 4 >>> round(7.824, 2) 7.82 >>> round(7.825, 2) 7.83 >>> round(7.8555, 3) 7.856</pre>
<code>abs(<Число>)</code>	Повертає абсолютне значення.	<pre>>>> abs(-70) 70 >>> abs(77) 77 >>> abs(-77.5) 77.5</pre>
<code>pow(<Число>, <Степінь>[, <Остача від ділення>])</code>	Підносить <Число> до <Степені>. Якщо вказано третій параметр, то повертається остача від ділення.	<pre>>>> pow(7, 2) 49 >>> pow(0.7, 2) 0.48999999999999994 >>> pow(5, 2, 4) 1</pre>

<code>max(<Список чисел через кому>)</code>	Максимальне значення зі списку.	<pre>>>> max(2, 1.7, 9, 7) 9 >>> max(2, 1.7, 9.1, 7) 9.1</pre>
<code>min(<Список чисел через кому>)</code>	Мінімальне значення зі списку.	<pre>>>> min(2, 1.7, 9.1, 7) 1.7 >>> min(2, 1, 9.1, 7) 1</pre>
<code>sum</code> (<code><Послідовність></code> <code>[, <Початкове значення>]</code>)	Повертає суму значень елементів послідовності (наприклад, списку, кортежу) плюс <code><Початкове значення></code> . Якщо другий параметр не вказано, то значення параметра дорівнює 0. Якщо послідовність порожня, то повертається значення другого параметра.	<pre>>>> sum((70, 10, 30, 40)) 150 >>> sum([70, 10, 30, 40]) 150 >>> sum((70, 10, 30, 40), 5) 155 >>> sum([], 7) 7</pre>
<code>divmod(x, y)</code>	Повертає кортеж з двох значень (<code>x // y</code> , <code>x % y</code>).	<pre>>>> divmod(15, 2) (7, 1) >>> 7*2+1 15 >>> divmod(17, 3) (5, 2) >>> 17//3 5 >>> 17%3 2</pre>

12.5. Модуль math. Математичні функції

Модуль `math` надає додаткові функції для роботи з числами, а також стандартні константи. Перш ніж використовувати модуль, необхідно підключити його за допомогою інструкції:

```
import math
```

Розглянемо стандартні константи та основні функції, що надає модуль `math` (таблиця 12.5.1).

Таблиця 12.5.1

Синтаксис	Призначення	Приклади використання
<code>pi</code>	Повертає число π .	<pre>>>> import math >>> math.pi 3.141592653589793</pre>
<code>e</code>	Повертає значення константи e .	<pre>>>> math.e 2.718281828459045</pre>
<code>sin()</code> <code>cos()</code> <code>tan()</code>	Стандартні тригонометричні функції (синус, косинус, тангенс). Значення аргументу вказується в радіанах.	<pre>>>> a=math.pi/6 >>> math.sin(a) 0.49999999999999994 >>> math.sin(1.7) 0.9916648104524686 >>> math.cos(a) 0.8660254037844387</pre>

<code>asin()</code> <code>acos()</code> <code>atan()</code>	Обернені тригонометричні функції (арксинус, арккосинус, арктангенс). Значення повертається в радіанах.	<pre>>>> math.asin(0) 0.0 >>> math.asin(0.999) 1.526071239626163 >>> math.acos(-0.999) 3.09686756642106</pre>
<code>degrees()</code>	Перетворює радіани у градуси.	<pre>>>> math.degrees(math.pi/6) 29.999999999999996</pre>
<code>radians()</code>	Перетворює градуси у радіани.	<pre>>>> math.radians(180.0) 3.141592653589793</pre>
<code>exp()</code>	Експонента.	<pre>>>> import math >>> math.exp(0) 1.0 >>> math.exp(2) 7.38905609893065</pre>
<code>log()</code>	Логарифм.	<pre>>>> math.log(8, 2) 3.0 >>> math.log(100, 10) 2.0</pre>
<code>sqrt()</code>	Квадратний корінь	<pre>>>> math.sqrt(49) 7.0 >>> math.sqrt(100) 10.0</pre>
<code>ceil()</code>	Значення, округлене до найближчого більшого цілого.	<pre>>>> math.ceil(6.49) 7</pre>
<code>floor()</code>	Значення, округлене до найближчого меншого цілого.	<pre>>>> math.floor(7.77) 7</pre>
<code>pow(<Число>, <Степінь>)</code>	Підносить <Число> до <Степені>.	<pre>>>> math.pow(3, 4) 81.0</pre>
<code>fabs()</code>	Абсолютне значення.	<pre>>>> math.fabs(-14.5) 14.5</pre>
<code>fmod()</code>	Остача від ділення.	<pre>>>> math.fmod(25, 5) 0.0 >>> math.fmod(25, 7) 4.0</pre>
<code>factorial()</code>	Факторіал числа.	<pre>>>> math.factorial(3) 6</pre>
<code>modf(x)</code>	Повертає дробову і цілу частину x. Обидва числа мають той же знак, що й x.	<pre>>>> math.modf(4.07) (0.070000000000000028, 4.0)</pre>

Ми розглянули основні функції, щоб отримати повний список можна звернутися до [документації з модуля math](#).

12.6. Модуль random. Генерація випадкових чисел

Модуль random дозволяє генерувати випадкові числа. Перш ніж використовувати модуль, необхідно підключити його за допомогою інструкції:

```
import random
```

Розглянемо основні функції:

- `random()` – повертає псевдовипадкове число від 0.0 до 1.0:

```
>>> import random
>>> random.random()
0.5496035697985201
>>> random.random()
0.03130983927165354
```
- `seed ([<Параметр>] [, version=2])` – налаштовує генератор випадкових чисел на нову послідовність. За замовчуванням використовується системний час. Якщо значення параметра буде однаковим, то генерується однакове число:

```
>>> random.seed(7)
>>> random.random()
0.32383276483316237
>>> random.random()
0.15084917392450192
>>> random.seed(7)
>>> random.random()
0.32383276483316237
```
- `uniform(<Початкове значення>, <Кінцеве значення>)` – повертає псевдовипадкове дійсне число в діапазоні від <Початкове значення> до <Кінцеве значення>:

```
>>> import random
>>> random.uniform(0, 7)
2.998147608364766
>>> random.uniform(0, 7)
1.264932072954918
```
- `randint(<Початкове значення>, <Кінцеве значення>)` – повертає псевдовипадкове ціле число в діапазоні від <Початкове значення>, до <Кінцеве значення>:

```
>>> random.randint(0, 7)
7
>>> random.randint(0, 7)
4
>>> random.randint(0, 100)
50
```
- `randrange([<Початкове значення>,] <Кінцеве значення> [, <Крок>])` – повертає випадковий елемент з числової послідовності. Параметри аналогічні параметрам функції `range()`. Саме зі списку, що повертається функцією `range()`, і вибирається випадковий елемент:

```
>>> random.randrange(10)
9
>>> random.randrange(0, 7)
2
>>> random.randrange(0, 15, 3)
12
```

- `choice(<Послідовність>)` – повертає випадковий елемент з будь-якої послідовності (рядка, списку, кортежу):

```
>>> random.choice("рядок")
'р'
>>> random.choice("рядок")
'я'
```

- `shuffle(<Список>[, <Число від 0.0 до 1.0>])` – перемішує елементи списку випадковим чином. Функція перемішує сам список і нічого не повертає. Якщо другий параметр не вказано, то використовується значення, яке повертається функцією `random()`.

Приклад:

```
>>> arr = [1, 2, 3, 4, 5, 6, 7]
>>> random.shuffle(arr)
>>> arr
[5, 1, 6, 4, 2, 3, 7]
```

- `sample(<Послідовність>, <Кількість елементів>)` – повертає список із зазначеної кількості елементів. У цей список потраплять елементи з послідовності, вибрані випадковим чином. Як послідовності можна вказати будь-які об'єкти, що підтримують ітерації. Приклади:

```
>>> random.sample("рядок", 2)
['д', 'р']
>>> random.sample(range(100), 7)
[62, 53, 23, 32, 51, 82, 41]
```

12.7. Практикум. Завдання 12.7

Складіть програму для обчислення площі трапеції ($S = \frac{a+b}{2}h$) за відомими основами (a, b) та висотою (h) та збережіть її у файлі «Завдання 12-7». Виконайте її за допомогою інтегрованого середовища IDLE для 2-3 наборів даних, а потім за допомогою відкриття файлу «Завдання 12-7».

Технологія виконання

1. Відкрийте інтегроване середовище IDLE.
2. Створіть новий файл та збережіть його під ім'ям «Завдання 12-7».
3. Введіть текст програми:

```
print("Введіть основи (a і b) та висоту (h) трапеції.")
a=float(input('a= '))
b=float(input('b= '))
h=float(input('h= '))
print('S= ', (a+b)/2*h)
input('Для завершення роботи натисніть клавішу <Enter>.'
```

4. Збережіть внесені зміни.
 5. Запустіть програму на виконання.
 6. Перегляньте результати виконання та при потребі зробіть необхідні корективи.
- Перевірте, виконавши програму.
7. Збережіть зміни та закрийте IDLE.
 8. Виконайте програму відкривши файл «Завдання 12-7» (виконавши на ньому подвійне клацання миші або скориставшись контекстним меню).

12.8. Практикум. Завдання 12.8

Складіть програму для обчислення значень функції $f(x) = 5x^4 - 3x^2 + 7x - 15$ та збережіть її у файлі «Завдання 12-8». Виконайте програму за допомогою інтегрованого середовища IDLE для 2-3 наборів даних, а потім за допомогою відкриття файлу «Завдання 12-8».

Технологія виконання

1. Відкрийте інтегроване середовище IDLE.
 2. Створіть новий файл та збережіть його під ім'ям «Завдання 12-8».
 3. Текст програми, наприклад, може бути таким:


```
print('Введіть значення аргумента функції.')
x=float(input('x = '))
print('f(', x, ') = ', 5*x**4-3*x**2+7*x-15)
input("Для завершення натисніть <Enter>.")
```
 4. Збережіть внесені зміни.
 5. Запустіть програму на виконання.
 6. Перегляньте результати виконання та при потребі зробіть необхідні корективи.
- Перевірте, виконавши програму.
7. Збережіть зміни та закрийте IDLE.
 8. Виконайте програму відкривши файл «Завдання 12-8» (виконавши на ньому подвійне клацання миші або скориставшись контекстним меню).

12.9. Практикум. Завдання 12.9*

Із пунктів А і В, розташованих одне від одного на віддалі d км, назустріч один одному одночасно відправляються два поїзди; швидкість першого V_1 км/год, швидкість другого V_2 км/год. В цей же час із пункту А вилітає зверхшвидкісна муха із швидкістю V км/год і летить назустріч поїзду з пункту В. Зустрівшись з ним, вона летить до поїзда із пункту А, і т. д. до тих пір, поки поїзди не зустрінуться. Визначити загальну відстань, яку пролетить муха. (Ім'я файлу для збереження «Завдання 12-9».)

Технологія виконання

1. Відкрийте інтегроване середовище IDLE.
2. Створіть новий файл та збережіть його під ім'ям «Завдання 12-9».
3. Якщо у вас виникли труднощі при складанні алгоритму для розв'язування задачі ви можете скористатися підказкою.

Підказка

Загальний шлях мухи залежить від часу її польоту, який дорівнює часу руху поїздів до зустрічі. А цей час дорівнює відношенню відстані між містами до сумарної швидкості поїздів (швидкості зближення).

4. Текст програми, наприклад, може бути таким:

```
d=float(input('d = '))
V1=float(input('V1 = '))
V2=float(input('V2 = '))
Vm=float(input('Vm = '))
print('S = ', d/(V1+V2)*Vm)
input("Для завершення натисніть <Enter>.")
```

5. Збережіть внесені зміни.

6. Запустіть програму на виконання.

7. Перегляньте результати виконання та при потребі зробіть необхідні корективи.

Перевірте, виконавши програму.

8. Збережіть зміни та закрийте IDLE.

12.10. Практикум. Завдання 12.10*

Складіть програму для обчислення суми n перших натуральних чисел $S = 1 + 2 + 3 + 4 + 5 + \dots + n$. Збережіть її у файлі «Завдання 12-10».

Технологія виконання

1. Відкрийте інтегроване середовище IDLE.

2. Створіть новий файл та збережіть його під ім'ям «Завдання 12-10».

3. Якщо у вас виникли труднощі при складанні алгоритму для розв'язування задачі ви можете скористатися підказкою.

Підказка

$$1 + 2 + 3 + 4 + 5 + \dots + n = \frac{n(n + 1)}{2}$$

4. Текст програми, наприклад, може бути таким:

```
n=int(input('n = '))
print('1+2+...+', n, '=', n*(n+1)/2)
input("Для завершення натисніть <Enter>.")
```

5. Збережіть внесені зміни.

6. Запустіть програму на виконання.

7. Перегляньте результати виконання та при потребі зробіть необхідні корективи.

Перевірте, виконавши програму.

8. Збережіть зміни та закрийте IDLE.

12.11. Практикум. Завдання 12.11

Складіть програму для обчислення значень функції $f(x) = \sin(x) + \cos^2(x)$ збережіть її у файлі «Завдання 12-11».

Технологія виконання

1. Відкрийте інтегроване середовище IDLE.

2. Створіть новий файл та збережіть його під ім'ям «Завдання 12-11».

Підказка

Для використання тригонометричних функцій потрібно приєднати модуль math.

3. Створіть алгоритм та введіть текст програми. Наприклад, він може бути таким:

```
import math
print('Введіть значення аргумента функції.')
x=float(input('x = '))
print('f(', x, ') = ', math.sin(x)+(math.cos(x))**2)
input("Для завершення натисніть <Enter>.")
```

4. Збережіть внесені зміни.
 5. Запустіть програму на виконання.
 6. Перегляньте результати виконання та при потребі зробіть необхідні корективи.
- Перевірте, виконавши програму.
7. Збережіть зміни та закрийте IDLE.

12.12. Практикум. Завдання 12.12 (Черепашка)

Створіть та збережіть програму, після виконання якої, на полотні за допомогою модуля Черепашка буде побудовано горизонтальний відрізок прямої, зеленого кольору, довжиною **a**, значення якої повинен ввести користувач під час виконання програми. Повинна виконуватися умова економії «фарби» (довжина шляху який черепашка проходить з опущеним пером повинна бути мінімально). Після виконання – Черепашка у початковому положенні, перо підняте. (Ім'я файлу для збереження «Завдання 12-12»)

Технологія виконання

1. Відкрийте інтегроване середовище IDLE.
 2. Створіть новий файл та збережіть його під ім'ям «Завдання 12-12».
 3. Скориставшись досвідом, набутим пі час виконання завдання №5 з пункту 7.6 та завдань 12.7-12.11, складіть алгоритм (план розв'язання) та реалізуйте його у вигляді програми.
 4. Якщо у вас виникли труднощі при складанні алгоритму, то ви можете скористатися [лістингом програми](#).
 5. Запустіть програму на виконання. Зверніть увагу, що для введення значення довжини використовується інше вікно.
 6. Перегляньте результати виконання та при потребі зробіть необхідні корективи.
- Перевірте, виконавши програму.
7. Збережіть зміни та закрийте IDLE.

12.13. Практикум. Завдання 12.13

Створіть програму, для визначення цифр тризначного натурального числа. (Ім'я файлу для збереження «Завдання 12-13».)

Технологія виконання

1. Створіть у середовищі IDLE новий файл та збережіть його під ім'ям «Завдання 12-13».

2. Для визначення цифри одиниць цифри можна скористатися операцією Остача від ділення (%), а для визначення цифри десятків та сотень операціями Остача від ділення (%) та Ціла частина від ділення (/).

3. Тоді програма може бути, наприклад такою:

```
n=int(input('Введіть число: '))
print('Цифра одиниць: ', n%10)
print('Цифра десятків: ', (n//10)%10)
print('Цифра сотень: ', (n//100)%10)
input()
```

4. Запустіть програму на виконання.

5. Перегляньте результати виконання та при потребі зробіть необхідні корективи. Перевірте, виконавши програму.

6. Збережіть зміни та закрийте IDLE.

12.14. Практикум. Завдання 12.14 (Черепашка)

Створіть та збережіть програму, після виконання якої, за допомогою Черепашки буде побудовано правильний (рівносторонній) шестикутник з розміром сторони та товщиною лінії, які визначає користувач у процесі виконання програми. (Ім'я файлу для збереження «Завдання 12-14».)

Технологія виконання

1. Запустіть IDLE. Створіть новий файл та збережіть його (для прискорення процесу введення програми ви можете скопіювати файл вже виконаного завдання 12.12 та перейменувати його).

2. Для проектування алгоритму скористайтеся досвідом, набутим під час виконання завдань [7.3 \(Черепашка\)](#) й [12.12 \(Черепашка\)](#), та знаннями зі шкільного курсу математики (градусна міра кутів правильного шестикутника дорівнює 120°).

3. Виконайте програму. У випадку виявлення помилок, виправіть їх. Збережіть внесені зміни та закрийте IDLE.

4. Якщо у вас виникли труднощі, то ви можете скористатися [лістингом програми](#).

12.15. Практикум. Завдання для самостійного виконання

1. Складіть програму для обчислення площі паралелограма ($S = ah$) за відомою основою (a) та висотою (h) та збережіть її у файлі «Завдання 12-15-1-с». Виконайте її за допомогою інтегрованого середовища IDLE для 2-3 наборів даних, а потім за допомогою відкриття файлу «Завдання 12-15-1-с».

2. * Скласти програму обміну значеннями між змінними A і B, не застосовуючи третю змінну та оператор обміну. Значення змінних – дійсні числа. (Ім'я файлу для збереження «Завдання 12-15-2-с» та «Завдання 12-15-2-1-с».)

3. * Як правило, комплект доміно має 28 пластинок. Якби кількість крапок на пластинках змінювалась не від 0 до 6, а від 0 до деякого k, то вочевидь, кількість пластинок буде іншою. Складіть програму, яка запитує у користувача кількість вічок ($0 < k < 21$), а виводить кількість пластинок (n), необхідну для відповідного комплекту доміно. Наприклад:

k	n
6	28

2	6
---	---

(Ім'я файлу для збереження «Завдання 12-15-3-с».)

4. Створіть та збережіть (ім'я файлу «Завдання 12-15-4-с») програму, після виконання якої, на полотні за допомогою модуля Черепашка буде побудовано вертикальний відрізок прямої, червоного кольору, довжиною **b**, значення якої повинен ввести користувач під час виконання програми. Повинна виконуватися умова економії «фарби» (довжина шляху який Черепашка проходить з опущеним пером повинна бути мінімальною). Після виконання – Черепашка у початковому положенні, перо підняте. (Черепашка)
5. Створіть програму, для визначення цифр двозначного натурального числа. (Ім'я файлу для збереження «Завдання 12-15-5-с».)
6. Створіть програму, для визначення суми та добутку цифр двозначного натурального числа. (Ім'я файлу для збереження «Завдання 12-15-6-с».)
7. Створіть програму, для визначення цифр чотиризначного натурального числа. (Ім'я файлу для збереження «Завдання 12-15-7-с».)
8. Створіть програму, для визначення суми та добутку цифр тризначного натурального числа. (Ім'я файлу для збереження «Завдання 12-15-8-с».)
9. Створіть програму, для визначення числа, що утвориться у результаті перестановки одиниць та сотень даного чотиризначного натурального числа. (Ім'я файлу для збереження «Завдання 12-15-9-с».)
- 10.* [14, стор. 4] Ім'я файлу для збереження «Завдання 12-15-10-с».

Загальний бал:	100
Обмеження часу:	100 мс
Обмеження реального часу:	5 с
Обмеження пам'яті	64 М

Степан вирішив пригостити однокласників шоколадками. Шоколадка коштувала **N** грн. З першого листопада вартість шоколадки збільшилась рівно на **P** відсотків. Визначте, скільки шоколадок зможе купити Степан на **S** грн після подорожчання.

Формат вхідних даних:

У першому рядку задано число **N** ($1 \leq N \leq 107$) - вартість шоколадки до подорожчання. У другому рядку - **P** ($0 \leq P \leq 100$) - величина подорожчання шоколадки у відсотках. В третьому рядку - **S** ($1 \leq S \leq 107$) - сума грошей, яка є у Степана.

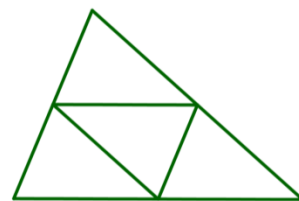
Формат вихідних даних:

Виведіть одне число - кількість шоколадок, які може купити Степан.

Приклади

Вхідні дані	Результат роботи
25	3
5	

11. Михайлик любив малювати трикутники, але він це робив у незвичний спосіб. Спочатку малював довільний трикутник, потім кожну сторону ділив на n рівних частин і проводив через точки поділу прямі, паралельні сторонам трикутника. У результаті виходить декілька рівних між собою трикутників. Допоможіть Михайлику знайти найбільшу кількість рівних трикутників у його фінальному рисунку.



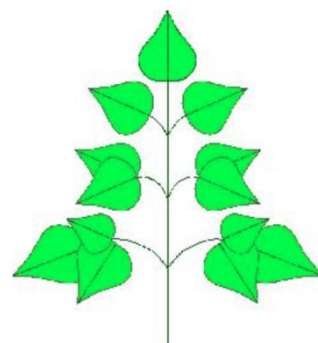
Вхідні дані

Ціле число n ($0 < n < 2 * 10^9$).

Вихідні дані

Вивести найбільшу кількість рівних між собою трикутників. Ім'я файлу для збереження «Завдання 12-15-11-с». [17, стор. 15; 19, №2400]

12. Мама попросила Васю полити всі молоді деревця у саду. Вася знає, що поки дерева маленькі, їх потрібно дуже добре поливати. А ось скільки поливати – невідомо. Але Вася – дуже розумний хлопчик. Він уважно прочитав весь підручник ботаніки для середньої школи і вияснив, що полив прямо пропорційний кількості листочків на дереві. Для гарного росту дерев достатньо виливати під дерево щоденно по одному літру води для кожного листка. На щастя Васі виявилось, що листки на деревах ростуть ярусами, причому на верхньому ярусі два листка, на другому – чотири, на наступному – шість, і так далі, на кожному наступному ярусі на два листки більше у порівнянні з попереднім. А на самій верхівці росте ще один листочок. Хитрий Вася послав молодшу сестричку Машеньку підрахувати кількість ярусів на кожному дереві, а Вас просить написати програму, яка для кожного дерева обрахує кількість літрів води для його поливу.



Вхідні дані

Кількість ярусів n ($0 \leq n \leq 1000$) на дереві.

Вихідні дані

Вивести кількість літрів води потрібних для поливу цього дерева. Ім'я файлу для збереження «Завдання 12-15-12-с». [17, стор. 16; 19, №248]

13. Вартість пляшки води, враховуючи вартість порожньої пляшки, становить 1 грн 20 коп., а вартість порожньої пляшки 20 коп.

Скільки пляшок води можна випити на n грн, враховуючи, що порожні пляшки можна здавати, і на одержані гроші купувати нові пляшки води.

Вхідні дані

Натуральне значення n ($1 \leq n \leq 1000$).

Вихідні дані

Кількість пляшок води, яку можна випити на n грн. Ім'я файлу для збереження «Завдання 12-15-13-с». [17, стор. 16; 19, №6277]

14. «Гарним» будемо вважати число, що складається лише з непарних цифр. Наприклад число 157953 гарне, а число 2452117 ні. Необхідно з'ясувати, скільки існує n – значних гарних чисел.

Вхідні дані

Одне ціле невід'ємне число n ($1 \leq n \leq 20$).

Вихідні дані

Вивести кількість гарних чисел. Ім'я файлу для збереження «Завдання 12-15-14-с». [19, №7817]

15. Знайти K – кількість N -значних натуральних чисел, що мають у своєму запису хоча б одну цифру 7.

Вхідні дані

Одне натуральне число N ($1 \leq N \leq 10$).

Вихідні дані

Шукане число K .

Вхідні дані (приклад)

2

Вихідні дані (приклад)

18

Ім'я файлу для збереження «Завдання 12-15-15-с». [19, №1355]

16. Пиріжок у шкільній їдальні коштує a гривень та b копійок. Знайдіть скільки гривень та копійок заплатить Петрик за n пиріжків.

Вхідні дані

Три натуральних числа a, b, n ($0 \leq a, b, n \leq 100$).

Вихідні дані

Через пропуск два числа: вартість покупки у гривнях та копійках.

Вхідні дані (приклад)

1 25 2

Вихідні дані (приклад)

2 50

Ім'я файлу для збереження «Завдання 12-15-16-с». [19, №7336]

17. Учні 10-Б класу, на осінні канікули, вирішили поїхати на екскурсію до столиці. Знаючи кількість хлопчиків n та дівчаток m , визначити скільки потрібно замовити кімнат в готелі, в якому є кімнати на k місць кожна, за умови, що хлопчиків та дівчаток поселяти разом заборонено.

Вхідні дані

В одному рядку записано три числа n, m, k ($n, m, k \leq 100$).

Вихідні дані

Вивести одне число - кількість кімнат, які потрібно забронювати в готелі.

Вхідні дані (приклад)

6 12 3

Вихідні дані (приклад)

6

Ім'я файлу для збереження «Завдання 12-15-17-с». [19, №7460]

18. Знайти суму цифр даного двоцифрового числа.

Вхідні дані

У єдиному рядку задане двоцифрове ціле число.

Вихідні дані

У єдиному рядку сума його цифр.

Вхідні дані (приклад)

25

Вихідні дані (приклад)

7

Ім'я файлу для збереження «Завдання 12-15-18-с». [19, №933]

19. У заданому трицифровому натуральному числі поміняти першу та останню цифри місцями.

Вхідні дані

Одне натуральне трицифрове число n ($100 \leq n \leq 999$).

Вихідні дані

Вивести число, отримане в результаті вказаного обміну.

Вхідні дані (приклад)

125

Вихідні дані (приклад)

521

Ім'я файлу для збереження «Завдання 12-15-19-с». [19, №943]

20. Знайдіть квадрат суми цифр чотирицифрового натурального числа.

Вхідні дані

Одне натуральне чотирицифрове число.

Вихідні дані

Виведіть квадрат суми цифр заданого числа.

Вхідні дані (приклад)

4765

Вихідні дані (приклад)

484

Ім'я файлу для збереження «Завдання 12-15-20-с». [19, №955]

21. Знайти добуток цифр п'ятицифрового числа n , які стоять на непарних розрядах.

Вхідні дані

Ціле п'ятицифрове число n .

Вихідні дані

Вивести добуток цифр на непарних розрядах.

Вхідні дані (приклад)

12345

Вихідні дані (приклад)

15

Ім'я файлу для збереження «Завдання 12-15-21-с». [19, №7459]

22. Знайти цілочисельну довжину сторони квадрата, який можна отримати з двох прямокутників axb та cxd , розрізавши їх на прямокутники, а потім склавши так, щоб утворився квадрат найбільш можливої площі.

Вхідні дані

У одному рядку знаходиться чотири дійсних числа a, b, c, d . Площа кожного прямокутника не перевищує $2 \cdot 10^9$.

Вихідні дані

Вивести одне число – сторону утвореного квадрата.

Вхідні дані (приклад)

1 2 3 4

Вихідні дані (приклад)

3

Ім'я файлу для збереження «Завдання 12-15-22-с». [19, №1359]

23. Задано натуральне число N . Напишіть програму, яка знаходить кількість натуральних чисел, що не перевищують N і не діляться на жодне з чисел 2, 3, 5.

Вхідні дані

Один рядок, у якому міститься число $N (1 \leq N \leq 1000000000)$.

Вихідні дані

Вивести знайдене число.

Вхідні дані (приклад)

10

Вихідні дані (приклад)

2

Ім'я файлу для збереження «Завдання 12-15-23-с». [19, №2806]

Питання для самоконтролю

1. Які числові типи підтримує Python 3?
2. В мові Python якого типу буде результат операції у якій беруть участь ціле і дійсне числа?
3. Як на мові Python записується математичний оператор «Ціла частина від ділення»?
4. Який оператор на мові Python записується наступним чином: $a \% b$?
5. Як на мові Python записується математичний оператор «Піднесення до степені»?
6. Перерахуйте відомі вам оператори присвоювання у мові програмування Python? Як вони позначаються та виконуються?
7. Який пріоритет математичних операторів у Python? За допомогою чого його можна змінити?
8. Назвіть відомі вам вбудовані функції для роботи з числами.
9. Яке призначення модуля `math`?
10. Що дозволяє робити модуль `random`?

13. Умовні оператори та цикли

Умовні оператори дозволяють виконувати окрему ділянку програми залежно від значення логічного виразу або, навпаки, не виконувати його. [4]

13.1. Операції порівняння

Операції порівняння використовуються в логічних виразах. Розглянемо їх за допомогою таблиці 13.1.1.

Таблиця 13.1.1

Синтаксис	Значення	Приклади використання
==	Дорівнює.	<pre>>>> 7==7; 7==10 True False</pre>
!=	Не дорівнює.	<pre>>>> 7!=7, 7!=10 (False, True)</pre>
<	Менше.	<pre>>>> 10<10, 10<7; 5<10 (False, False) True</pre>
>	Більше.	<pre>>>> 10>10, 10>7, 5>10 (False, True, False)</pre>
<=	Менше або дорівнює.	<pre>>>> 5<=4, 5<=5, 5<=7 (False, True, True)</pre>
>=	Більше або дорівнює.	<pre>>>> 5>=4, 5>=5, 5>=7 (True, True, False)</pre>
in	Перевірка на входження до послідовності.	<pre>>>> 5 in [3,7,1] False >>> "Рядок" in "Рядок символів" True >>> 7 in (3,7,1) True</pre>
is	Перевіряє, чи посилаються дві змінні на один і той же об'єкт.	<pre>>>> m=n=[3, 4] >>> m is n True >>> x=[3,4]; y=[3,4] >>> x is y</pre>

Декілька логічних виразів можна об'єднати в один за допомогою логічних операцій (таблиця 13.1.2).

Таблиця 13.1.2

Вираз 1	Вираз 2	not	and	or
False	-	True	-	-
True	-	False	-	-
False	False	-	False	False
False	True	-	False	True

True	False	-	False	True
True	True	-	True	True

Перерахуємо операції згідно порядку убунання пріоритету:

- 1) <, >, <=, >=, ==, !=, <>, is, is not, in, not in;
- 2) not – логічне заперечення;
- 3) and – логічне І;
- 4) or – логічне АБО.

У логічних виразах можна використовувати дужки, тоді спочатку виконуватимуться операції у дужках.

Приклади використання:

```
>>> 1>5 or 3>1
True
>>> 1>5 and 3>1
False
>>> not (1>5 or 3>1)
False
>>> not 1>5 or 3>1
True
```

13.2. Оператор розгалуження *if... else*

Оператор розгалуження *if...else* дозволяє в залежності від значення логічного виразу виконати окрему ділянку програми або, навпаки, не виконувати її. Оператор має такий вигляд:

```
if <Логічний вираз>:
    <Блок, що виконується, якщо умова істинна>
elif <Логічний вираз>:
    <Блок, що виконується, якщо умова істинна>
else:
    <Блок, що виконується, якщо всі умови помилкові>
```

Блоки всередині складеної інструкції виділяються однаковою кількістю пробілів (зазвичай чотири пробіли, можна також використовувати клавішу <Tab>). Кінцем блоку є інструкція, перед якою розташовано менша кількість пробілів. У деяких мовах програмування логічний вираз беруть у круглі дужки. У мові Python це робити необов'язково, але можна, бо будь-який вираз може бути розташований всередині круглих дужок.

Розглянемо роботу оператора розгалуження на прикладі простої програми:

```
a=int(input('a = '))
if a>7:
    print('7')
elif a>3:
    print('6')
elif a>1:
    print('5')
else:
    print('0')
input('Для завершення натисніть <Enter>.'
```

Результати роботи програми для різних значень **a** будуть такими:

```

RESTART: D:\OneDrive\Робоча документація\rik 2017-2018\
Наукова робота\Посібники\Python для інф\Матеріали\Програ
ми\Приклади\Пункт 1-13-2\Ком-розгал.ру
a = 8
7
Для завершення натисніть <Enter>.
>>>
RESTART: D:\OneDrive\Робоча документація\rik 2017-2018\
Наукова робота\Посібники\Python для інф\Матеріали\Програ
ми\Приклади\Пункт 1-13-2\Ком-розгал.ру
a = 4
6
Для завершення натисніть <Enter>.
>>>
RESTART: D:\OneDrive\Робоча документація\rik 2017-2018\
Наукова робота\Посібники\Python для інф\Матеріали\Програ
ми\Приклади\Пункт 1-13-2\Ком-розгал.ру
a = 2
5
Для завершення натисніть <Enter>.
>>>
RESTART: D:\OneDrive\Робоча документація\rik 2017-2018\
Наукова робота\Посібники\Python для інф\Матеріали\Програ
ми\Приклади\Пункт 1-13-2\Ком-розгал.ру
a = 1
0

```

Один умовний оператор можна вкласти в інший. У цьому випадку відступ вкладеної інструкції повинен бути у два рази більшим.

Оператор **if ... else** має ще один формат:

<Змінна> = <Якщо істина> if <Умова> else <Якщо хиба>

Наприклад:

```

>>> print("Так" if 10 % 2 == 0 else "Hi")
Так
>>> s = "Так" if 10 % 2 == 0 else "Hi"
>>> s
'Так'
>>> s = "Так" if 7 % 2 == 0 else "Hi"
>>> s
'Hi'

```

13.3. Практикум. Завдання 13.3

Створіть програму, яка перевіряє, чи є введене користувачем натуральне число парним чи ні. Після перевірки повинно вивестися відповідне повідомлення. Збережіть її у файлі «Завдання 13-3». Виконайте її за допомогою інтегрованого середовища IDLE для 2-3 наборів даних.

Технологія виконання

1. Відкрийте інтегроване середовище IDLE.
2. Створіть новий файл та збережіть його під ім'ям «Завдання 13-3».
3. При створенні програми використаємо команди введення, виведення та розгалуження:


```
x=int(input("Введіть натуральне число: "))
if x%2==0:
    print(x, '- парне число')
else:
    print(x, '- непарне число')
input()
```

4. Збережіть програму та виконайте її для різних наборів даних.
 5. Перегляньте результати виконання та при потребі зробіть необхідні корективи.
- Перевірте, виконавши програму.
6. Збережіть зміни та закрийте IDLE.

Зауваження

Якщо блок складається з однієї інструкції, то цю інструкцію можна розмістити на одному рядку із ключовим словом if та логічним виразом:

```
x=int(input("Введіть натуральне число: "))
if x%2==0: print(x, '- парне число.')
else: print(x, '- непарне число.')
input()
```

13.4. Практикум. Завдання 13.4

Створіть програму для розв'язування квадратного рівняння $ax^2 + bx + c = 0$. Збережіть її у файлі «Завдання 13-4». Протестуйте її за допомогою системи тестів наведеної у таблиці 13.4.1.

Таблиця 13.4.1

Номер тесту	Випадок, що перевіряється	Коефіцієнти			Результати
		a	b	c	
1	$d > 0$	1	1	-2	$x_1 = 1, x_2 = -2$
2	$d = 0$	1	2	1	Корені дорівнюють: $x_1 = -1, x_2 = -1$
3	$d < 0$	2	1	2	Не має дійсних коренів.
4	$a = 0, b = 0, c = 0$	0	0	0	Всі коефіцієнти дорівнюють нулю. x – довільне число.
5	$a = 0, b = 0, c \neq 0$	0	0	2	Неправильне рівняння.
6	$a = 0, b \neq 0$	0	2	1	Лінійне рівняння. Один корінь: $x = -0,5$
7	$a \neq 0, b \neq 0, c = 0$	2	1	0	$x_1 = 0, x_2 = -0,5$

Технологія виконання

1. За допомогою інтегрованого середовища IDLE створіть новий файл та збережіть його під ім'ям «Завдання 13-4».
2. Перед створенням програми для розв'язування квадратного рівняння потрібно спочатку проаналізувати різні види рівнянь, що ми отримаємо у залежності від значень коефіцієнтів a , b та c . Це завдання значно спрощується завдяки наведеній системі тестів. Результати аналізу відповідно необхідно врахувати при створенні алгоритму та написанні програми.

3. Введіть програму, збережіть її та виконайте для різних наборів даних, наведених у системі тестів.
4. Виконайте програму для кількох власних наборів даних.
5. Перегляньте результати виконання та при потребі зробіть необхідні корективи. Перевірте, виконавши програму.
6. У випадку виникнення труднощів можете скористатися [лістингом програми](#).
7. Закрийте IDLE.

Підказка

Для обчислення \sqrt{D} потрібно приєднати модуль math.

13.5. Практикум. Завдання 13.5

Створіть програму для виведення на екран більшого з трьох введених користувачем чисел. Функції (такі, наприклад, як `max()`, `min()` тощо) використовувати забороняється. (Ім'я файлу для збереження «Завдання 13-5».)

Технологія виконання

1. Відкрийте інтегроване середовище IDLE.
2. Створіть новий файл та збережіть його під ім'ям «Завдання 13-5».
3. Після розбору матеріалу і прикладу, наведеного у пункті 13.2, розбору пункту 13.1 та виконання практичних завдань з пунктів 13.3 й 13.4, єдина незначна складність при створенні програми може полягати у складанні відповідних логічних виразів у операторі розгалуження. У такому випадку можете скористатися [лістингом програми](#).
4. Введіть і збережіть програму. Виконайте її для різних наборів даних.
5. Перегляньте результати виконання та при потребі змініть код. Перевірте, виконавши програму.
6. Збережіть зміни та закрийте IDLE.

13.6. Практикум. Завдання 13.6

Створіть програму для обчислення значень функції $y = \begin{cases} x^2 + 1, & \text{якщо } x > 3; \\ x - 4, & \text{якщо } x \leq 3. \end{cases}$
(Ім'я файлу для збереження «Завдання 13-6».)

Технологія виконання

1. За допомогою IDLE створіть та збережіть новий файл під ім'ям «Завдання 13-6».
2. Досвід, набутий під час виконання завдань 13-3, 13-4, 13-5 дозволить вам з легкістю виконати це завдання. Про всяк випадок наводимо один з можливих варіантів коду у [лістингу програм](#).
3. Виконайте програму для кількох значень аргументу.
4. Перегляньте результати виконання та при потребі змініть код. Перевірте, виконавши програму.
5. Збережіть зміни та закрийте IDLE.

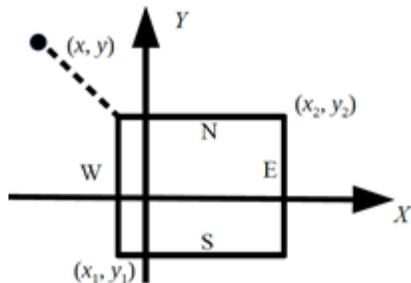
13.7. Практикум. Завдання для самостійного виконання

1. Складіть програму, яка перевіряє, чи ділиться введене користувачем число на 7 чи ні. Після перевірки повинно вивестися відповідне повідомлення. Збережіть її у файлі «Завдання 13-7-1-с». Виконайте її за допомогою інтегрованого середовища IDLE для 2-3 наборів даних.
2. Створіть програму для розв'язування лінійного рівняння $ax + b = 0$. Збережіть її у файлі «Завдання 13-7-2-с». Розробіть систему тестів (див. [завдання 13.2](#)) та виконайте програму для різних наборів даних.
3. Створіть програму для виведення на екран більшого з двох введених користувачем чисел. (Ім'я файлу для збереження «Завдання 13-7-3-с».)
4. Вивести на екран номер чверті, якій належить точка з координатами (x, y) при умові, що x та y відмінні від 0. (Ім'я файлу для збереження «Завдання 13-7-4-с».)
5. Дано натуральне двозначне число. Написати програму для визначення, чи є сума його цифр двозначним числом. (Ім'я файлу для збереження «Завдання 13-7-5-с».)
6. Дано натуральне двозначне число. Написати програму для визначення, чи є більшою цифра десятків за цифру одиниць, чи вони рівні. (Ім'я файлу для збереження «Завдання 13-7-6-с».)
7. Дано натуральне тризначне число. Написати програму для визначення чи є воно паліндромом («перевертнем»), іншими словами числом, десятковий запис якого читається однаково зліва направо і справа наліво. (Ім'я файлу для збереження «Завдання 13-7-7-с».)
8. Створіть програму для обчислення значень функції $y = \begin{cases} x - 7, & \text{якщо } x < 0, \\ 5, & \text{якщо } 0 \leq x \leq 5, \\ x^2 + 5, & \text{якщо } x > 5. \end{cases}$
(Ім'я файлу для збереження «Завдання 13-7-8-с».)
9. Дано три різних цілих числа, знайти середнє з них. Середнім назовемо число, яке більше найменшого з даних чисел, але менше максимального. (Ім'я файлу для збереження «Завдання 13-7-9-с».)
10. Складіть програму знаходження добутку двох найбільших з трьох введених з клавіатури чисел. (Ім'я файлу для збереження «Завдання 13-7-10-с».)
11. Дано тризначне натуральне число. Написати програму для визначення чи входить до нього цифра 4. (Ім'я файлу для збереження «Завдання 13-7-11-с».)
12. Дано двозначне натуральне число. Написати програму для визначення чи входять до нього цифри 5 і 7. (Ім'я файлу для збереження «Завдання 13-7-12-с».)
13. Складіть програму, яка з трьох введених з клавіатури чисел підносить до квадрату додатні, а від'ємні залишає без зміни. (Ім'я файлу для збереження «Завдання 13-7-13-с».)
- 14.* [14, стор. 6] Ім'я файлу для збереження «Завдання 13-7-14-с».

Загальний бал:	100
Обмеження часу:	100 мс
Обмеження реального часу:	5 с

Обмеження пам'яті	64 М
-------------------	------

Степан влітку відпочиває у бабусі в селі. Особливо йому подобається купатись на сільському озері. Посередині озера плаває пліт, який має форму прямокутника. Сторони плоту спрямовані уздовж паралелей і меридіанів. Введемо систему координат, в якій вісь **OX** направлена на схід, а вісь **OY** - на північ. Нехай південно-західний кут плоту має координати **(x1, y1)**,



північно-східний кут - координати **(x2, y2)**.

Степан знаходиться в точці з координатами **(x, y)**. Визначте, до якої сторони плоту (північної, південної, західної чи східної) або до будь-якого кута плоту (північно-західному, північно-східному, південно-західному, південно-східному) Степану потрібно плисти, щоб якомога

швидше дістатися до плоту.

Формат вхідних даних:

Дано шість чисел в наступному порядку: **x1, y1** (координати південно-західного кута плоту), **x2, y2** (координати північно-східного кута плоту), **x, y** (координати Степана). Всі числа цілі і по модулю не перевершують 100. Гарантується, що **x1 < x2, y1 < y2, x ≠ x1, x ≠ x2, y ≠ y1, y ≠ y2**, координати Степана знаходяться поза плотом.

Формат вихідних даних:

Якщо Степану слід плисти до північної сторони плоту, програма повинна вивести символ «N», до південної - символ «S», до західної - символ «W», до східної - символ «E». Якщо Степану слід плисти до кута плоту, потрібно вивести один з наступних рядків: «NW», «NE», «SW», «SE».

Приклади

Вхідні дані	Результат роботи
-1 -2 5 3 -4 6	NW

13.8. Цикл *for*

Функція *range()*

Функція *range()* має такий вигляд:

range ([<Початкове значення>,] <Кінцеве значення>[, <Крок>])

Якщо <Початкове значення> не вказано, то за замовчуванням використовується значення 0. Зауважимо, що кінцеве значення не входить до значень які повертаються. Якщо параметр <Крок> не вказано, то використовується значення 1.

Розглянемо окремі випадки:

- `range(n)` – шкала чисел від 0 до $n-1$;
- `range(k, n)` – шкала чисел від k до $n-1$;
- `range(k, n, m)` – шкала чисел від k до $n-1$ з кроком m (причому m може бути від'ємним).

Цикл *for*

Припустимо, потрібно вивести всі числа від 1 до 70 по одному у рядку. Звичайним способом довелося б писати 70 рядків коду:

```
print(1)
print(2)
...
print(70)
```

За допомогою циклів ту саму дію можна виконати одним рядком коду:

```
>>> for x in range(1, 71): print(x)
```

Іншими словами, цикли дозволяють виконати одні й ті ж інструкції багаторазово.

У мові Python використовуються два цикли: `for` і `while`. Цикл `for` найчастіше застосовується для перебору елементів послідовності. Він має такий формат:

```
for <Поточний елемент> in <Послідовність>:
    <Інструкції всередині циклу>
[else:
    <Блок, що виконується, якщо не використовувався оператор
break>]
```

Тут є такі конструкції:

- `<Послідовність>` – об'єкт, що підтримує механізм ітерації. Наприклад, рядок, список, кортеж, словник та ін.;
- `<Поточний елемент>` – на кожній ітерації через цей параметр є доступним поточний елемент послідовності або ключ словника;
- `<Інструкції всередині циклу>` – блок, який буде багаторазово виконуватися;
- якщо всередині циклу не використовувався оператор `break`, то після завершення виконання циклу буде виконано блок в інструкції `else`. Даний блок не є обов'язковим.

Приклади використання циклу *for*

Створимо програму для обчислення суми квадратів n перших натуральних чисел ($s = 1 + 2^2 + 3^2 + \dots + n^2$).

Алгоритми знаходження сум подібного типу мають схожу структуру:

- вводимо значення n ;
- присвоюємо змінній, у якій буде зберігатися сума, початкове значення;
- використовуємо інструкцію циклу для обчислення суми;
- виводимо результат.

Реалізація алгоритму цієї задачі на мові Python може бути, наприклад, такою:

```
n=int(input("n="))
s=0
for i in range(1, n+1):
    s=s+i**2
print("S=", s, sep=" ")
input()
```

Для розуміння виконання циклу розглянемо як будуть змінюватися значення змінних при виконанні програми для $n = 4$.

n	4	4	4	4	4
i		1	2	3	4
s	0	1	5	14	30

А тепер створимо програму для обчислення суми квадратів перших парних натуральних чисел, що не перевершують задане натуральне число n ($s = 2^2 + 4^2 + 6^2 + 8^2 + \dots + k^2$, де $k \leq n$).

Алгоритм для розв'язування цієї задачі буде майже аналогічним, єдине, що потрібно змінити та додати це значення параметрів функції `range()`.

```
n=int(input("n="))
s=0
for i in range(2, n+1, 2):
    s=s+i**2
print("S=", s, sep=" ")
input()
```

13.9. Практикум. Завдання 13.9

Складіть програму для обчислення факторіала. (Факторіал натурального числа n – добуток натуральних чисел від одиниці до n включно, позначається $n!$). Збережіть її у файлі «Завдання 13-9». Обчисліть за допомогою програми факторіали кількох натуральних чисел.

Технологія виконання

1. За допомогою інтегрованого середовища IDLE створіть новий файл та збережіть його під ім'ям «Завдання 13-5».

2. Спочатку розглянемо алгоритм розв'язування задачі:

⇒ ввести значення n ;

⇒ обчислити $n!$;

⇒ вивести результат.

3. Щоб ввести n використовуємо команду `n=int(input("n="))`.

4. Для збереження значення $n!$ застосуємо змінну f , а для його обчислення використаємо цикл `for`.

5. Щоб вивести результат використовуємо команду `print()`.

6. У результаті отримаємо:

```

n=int(input("n="))
f=1
for i in range(1, n+1):
    f=f*i
print(n, "!=" , f, sep=" ")
input()

```

7. Збережіть та виконайте програму для кількох значень n.
8. Перегляньте результати виконання та при потребі зробіть у програмі необхідні корективи. Перевірте, виконавши програму.
9. Закрийте IDLE.

13.10. Практикум. Завдання 13.10

Складіть програму для обчислення n-го члена послідовності Фібоначчі. (1, 1, 2, 3, 5, 8, ... – перші два члени дорівнюють 1, а всі інші – це сума двох попередніх.

Формат вхідних даних: одне число $0 < n < 100$.

Формат вихідних даних: одне число – n-й член послідовності.

Збережіть її у файлі «Завдання 13-10».

Технологія виконання

1. Створіть новий файл та збережіть його під ім'ям «Завдання 13-10».
2. Ідея розв'язання – обчислення й збереження тільки двох елементів послідовності:

$$y = x + y$$

$$x = y - x$$

3. Для розуміння покажемо, як змінюються значення x та y для n=5.

x	y
1	1
1	2
2	3
3	5
5	8

4. Використавши цю ідею, складіть відповідну програму.
5. Введіть програму, збережіть її та виконайте для різних наборів даних, наведених у системі тестів.
6. Виконайте програму для кількох власних наборів даних.
7. Перегляньте результати виконання та при потребі зробіть необхідні корективи. Перевірте, виконавши програму.
8. У випадку виникнення труднощів можете скористатися [лістингом програми](#).
9. Закрийте IDLE.

13.11. Практикум. Завдання 13.11

*Дано натуральне число n ($n < 9999$). Визначте, чи є воно паліндромом («перевертнем»), з урахуванням чотирьох цифр. Наприклад, паліндромами є числа: 2222, 6116, 0440. (Файл для збереження «Завдання 13-11»).

Технологія виконання

Почнемо не з програми, а з ручного трасування логіки рішення. Ми з Вами за допомогою цього прийому повинні досягти того, щоб при написанні програми у Вас одночасно складався «зоровий образ» її роботи, Ви бачили її роботу, причому це має бути не статична «картинка», а динамічна. Трасування зазвичай виконується для конкретних значень вхідних параметрів задачі.

Отже, у нас чотиризначне число, тому змінна оператора `for` змінюється від 1 до 4. У змінній з ім'ям **m** зберігається «залишок» числа, в початковий момент часу він дорівнює введеному числу. В змінній з ім'ям **r** формуємо значення числа «перевертня». Основними операціями є: $r = r * 10 + m \% 10$ (додавання чергової цифри до числа «перевертня») і $m = m // 10$ (зміна числа, що перевіряється). Процедура трасування наведена в таблиці 13.11.1. Після її виконання написання програми не являє складнощів.

Таблиця 13.11.1

i	m	r
-	2773	0
1	277	$0 * 10 + 2773 \% 10 = 0 + 3 = 3$
2	27	$3 * 10 + 277 \% 10 = 30 + 7 = 37$
3	2	$37 * 10 + 27 \% 10 = 370 + 7 = 377$
4	0	$377 * 10 + 2 \% 10 = 3770 + 2 = 3772$

1. Створіть новий файл та збережіть його під ім'ям «Завдання 1-13-11».
2. Спочатку розглянемо алгоритм розв'язування задачі:
 - ⇒ ввести значення n та m присвоїти значення n ;
 - ⇒ за допомогою циклу обчислити r ;
 - ⇒ використавши команду розгалуження вивести результат.
3. Введіть програму, збережіть та виконайте її для кількох значень n .
4. Перегляньте результати виконання та при потребі зробіть необхідні корективи. Перевірте, виконавши програму. У якості підказки можна скористатися [лістингом програми](#).
5. Закрийте IDLE.

13.12. Практикум. Завдання 13.12

*Дано натуральні числа n , k ($n < k$, $k < 9999$). З чисел від n до k вибрати ті, запис яких містить рівно три однакових цифри. Наприклад, числа 5855, 5777, 0004, 0200 містять рівно три однакових цифри. (Файл для збереження «Завдання 13-12»).

Технологія виконання

Якщо дане число містить рівно три однакових цифри, то тільки одна з цифр відрізняється від інших, тобто можливі чотири випадки, наведених у таблиці 13.12.1.

Таблиця 13.12.1

	1	2	3	4	
1	x	x	x		перша умова
2	x	x		x	друга умова
3	x		x	x	третя умова
4		x	x	x	четверта умова

Нехай у якості n і k введені числа 3532 і 3540. У змінних a_1, a_2, a_3, a_4 зберігаємо значення цифр поточного числа i (таблиця 13.12.2).

Таблиця 13.12.2

i	a_1	a_2	a_3	a_4	
3532	3	5	3	2	False
3533	3	5	3	3	True
3534	3	5	3	4	False
3535	3	5	3	5	False
3536	3	5	3	6	False
3537	3	5	3	7	False
3538	3	5	3	8	False
3539	3	5	3	9	False
3540	3	5	4	0	False

Переходимо до створення програми.

1. Створіть новий файл та збережіть його під ім'ям «Завдання 13-12».
2. Зважаючи на проведений аналіз задачі алгоритм її розв'язання може містити такі частини:

⇒ введення значень n та k ;

⇒ за допомогою циклу `for` здійснюється перебір чисел на відповідність умові задачі (використовується команда розгалуження) і якщо число задовольняє умові то здійснюється його виведення.

3. Введіть програму, збережіть та виконайте її для кількох значень n .
4. Перегляньте результати виконання та при потребі зробіть необхідні корективи. Перевірте, виконавши програму. У якості підказки можна скористатися [лістингом програми](#).
5. Закрийте IDLE.

13.13. Практикум. Завдання 13.13*

Ім'я файлу для збереження «Завдання 13-13».

Загальний бал:	100
Обмеження часу:	200 мс

Обмеження реального часу:	5 с
Обмеження пам'яті	64 М

Степан збирає речі у відпустку. З собою в літак він може взяти ручну поклажу і багаж. Для ручної поклажі у нього є рюкзак, а для багажу – здоровенна валіза.

За правилами перевезення маса ручної поклажі не повинна перевищувати S кг, а багаж може бути будь-якої маси (за наднормативний багаж Степан готовий доплатити). Зрозуміло, найбільш цінні речі – ноутбук, фотоапарат, документи і т. д. – Степан хоче покласти в ручну поклажу.

Степан розклав усі свої речі в порядку зменшення їх цінності і починає складати найбільш цінні речі в рюкзак. Він діє в такий спосіб – бере найцінніший предмет, і якщо його маса не перевищує S , то кладе його в рюкзак, інакше кладе його до валізи. Потім він бере наступний за цінністю предмет, якщо його можна покласти в рюкзак, тобто якщо його маса разом з масою вже покладених в рюкзак речей не перевищує S , то кладе його в рюкзак, інакше до валізи, і таким же чином процес триває для всіх предметів в порядку спадання їх цінності.

Визначте вагу рюкзака і валізи після того, як Степан складе всі речі.

Формат вхідних даних:

Перший рядок вхідних даних містить число S ($1 \leq S \leq 2 \times 10^9$) – максимально дозволена вага рюкзака. У другому рядку вхідних даних записано число N ($1 \leq N \leq 105$) – кількість предметів.

У наступних N рядках дано маси предметів, самі предмети перераховані в порядку спадання цінності (спочатку вказана маса найціннішого предмета, потім маса другого по цінності предмета і т. д.). Всі числа натуральні, сума ваги всіх предметів не перевищує 2×10^9 .

Формат вихідних даних:

Виведіть два числа – вагу рюкзака і вагу валізи (вага порожнього рюкзака і валізи не враховується).

Приклади

Вхідні дані	Результат роботи
10 4 6 3 2 1	102

[15, стор. 10]

Технологія виконання

В даній задачі необхідно реалізувати принцип, який описаний в умові, а саме: беремо предмет, і якщо його маса не перевищує максимально дозволenu масу, то кладемо його в рюкзак, інакше кладемо його до валізи.

1. Створіть новий файл та збережіть його під ім'ям «Завдання 13-13».
2. Нехай **vr** – маса рюкзака і **vv** – маса валізи. Беремо перший предмет з масою **r** і перевіряємо, якщо **vr+r** не перевищує **s**, то збільшуємо **vr** на **r**, інакше **vv** збільшуємо на **r**.

3. Розглянемо алгоритм розв'язування задачі:

⇒ ввести значення **s** та **n**, присвоїти значення 0 змінним **vr** і **vv**;

⇒ за допомогою циклу та команди розгалуження обчислити **vr** і **vv**;

⇒ вивести результат.

4. Отримуємо програму:

```
s=int(input())
n=int(input())
vr=int(0)
vv=int(0)
for i in range(1, n+1):
    r=int(input())
    if (vr+r)<=s:
        vr=vr+r
    else:
        vv=vv+r
print(vr, vv)
```

5. Збережіть та виконайте її для значень наведених в умові та ще кількох наборів даних.

6. Перегляньте результати виконання та при потребі зробіть необхідні корективи. Перевірте, виконавши програму.

7. Закрийте IDLE.

13.14. Практикум. Завдання 13.14 (Черепашка)

Створіть та збережіть (ім'я файлу «Завдання 13-14») програму, після виконання якої, на полотні Python Turtle Graphics за допомогою модуля Черепашка буде побудовано квадрат розміри сторони якого визначає користувач (рис. 13.14) використавши цикл `for`.

Технологія виконання

1. Створіть новий файл та збережіть його під ім'ям «Завдання 13-14».

2. Зважаючи на отриманий при виконанні попередніх завдань досвід, створення програми не повинно викликати труднощів.

3. Збережіть та виконайте програму для кількох значень **a** (довжина сторони квадрату).

4. Перегляньте результати виконання та при потребі зробіть необхідні корективи. Перевірте, виконавши програму.

5. Закрийте IDLE.

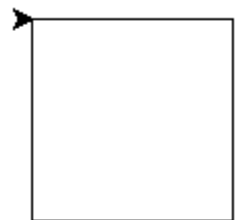


Рис. 13.14

6. Познайомитись одним з можливих варіантів розв'язання можна, скориставшись [лістингом програми](#).

13.15. Практикум. Завдання для самостійного виконання

1. Створіть програму для обчислення суми кубів n перших натуральних чисел ($s = 1 + 2^3 + 3^3 + \dots + n^3$). Збережіть її у файлі «Завдання 13-15-1-с».
2. Створіть програму для побудов орнаментів (приклад наведено на рис. 13.15.2), що складаються з квадратів зі спільною вершиною повернутих на певний кут. Кількість квадратів, довжину сторони та кут повороту визначає користувач у процесі діалогу під час виконання програми. Файл для збереження «Завдання 13-15-2-с». (Черепашка.)

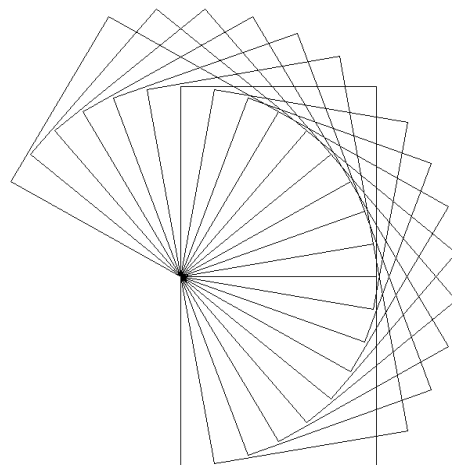


Рис. 13.15.2

3. Створіть програму для побудов штрихової лінії (приклад наведено на рис. 13.15.3). Кількість штрихів та їх довжину визначає користувач у процесі діалогу під час виконання програми. Файл для збереження «Завдання 13-15-3-с». (Черепашка.)

Рис. 13.15.3

4. Знайти всі двозначні числа, в яких є цифра N або саме число ділиться на N . Файл для збереження «Завдання 13-15-4-с». [16]
5. Визначити кількість тризначних натуральних чисел, сума цифр яких дорівнює заданому числу N та вивести їх на екран. Файл для збереження «Завдання 13-15-5-с».
6. Скласти програму обчислення суми кубів чисел від 25 до 55. Файл для збереження «Завдання 13-15-6-с». [16]
7. Серед двозначних чисел знайти ті, сума квадратів цифр яких ділиться на 13. Файл для збереження «Завдання 13-15-7-с». [16]
8. Написати програму пошуку двозначних чисел, таких, що якщо до суми цифр цього числа додати квадрат цієї суми, то вийде це число. Файл для збереження «Завдання 13-15-8-с». [16]
9. Квадрат тризначного числа закінчується трьома цифрами, які якраз і складають це число. Написати програму пошуку таких чисел. Файл для збереження «Завдання 13-15-9-с». [16]
10. Написати програму пошуку чотиризначного числа, яке при діленні на 133 дає в залишку 125, а при діленні на 134 дає в залишку 111. Файл для збереження «Завдання 13-15-10-с». [16]
11. Знайти суму натуральних чисел з проміжку від A до B ($B \leq 100000$), кратних 4 (значення змінних A і B вводяться з клавіатури). Файл для збереження «Завдання 13-15-11-с». [16]

12. Знайти суму натуральних чисел, більших 20, менших 100, кратних 3 і які закінчуються на 2, 4 або 8. Файл для збереження «Завдання 13-15-12-с». [16]
13. У тризначному натуральному числі закреслили першу цифру зліва, коли отримане двозначне число помножили на 7, то отримали дане число. Знайти це число. Файл для збереження «Завдання 13-15-13-с». [16]
14. Сума цифр тризначного натурального числа кратна 7, саме число також ділиться на 7. Знайти всі такі числа. Файл для збереження «Завдання 13-15-14-с». [16]
15. Серед чотиризначних натуральних чисел вибрати ті, у яких всі чотири цифри різні. Файл для збереження «Завдання 13-15-15-с». [16]
16. Серед двозначних натуральних чисел знайти ті, сума цифр яких дорівнює n ($0 < n < 18$) і число ділиться без залишку на числа a . Файл для збереження «Завдання 13-15-16-с». [16]
17. Дано чотиризначне натуральне число n . Викинути із запису числа n цифри 0 і 5, залишивши колишнім порядок інших цифр. Наприклад, з числа 1509 повинно вийти 19. Файл для збереження «Завдання 13-15-17-с». [16]
18. Натуральне число з n цифр є числом Армстронга, якщо сума його цифр, піднесених до n -го степеня, дорівнює самому числу (наприклад, $153 = 1^3 + 5^3 + 3^3$). Отримати всі числа Армстронга, що складаються з трьох і чотирьох цифр. Файл для збереження «Завдання 13-15-18-с». [15]
19. Дано натуральне число n ($n \leq 1000000$). Знайти всі його дільники та їх суму. Файл для збереження «Завдання 13-15-19-с». [15]
20. Створіть програму, після виконання якої, на полотні Python Turtle Graphics за допомогою модуля Черепашка буде побудовано правильний (рівносторонній) семикутник (розмір та товщину сторони, колір заливки оберіть довільним чином). Файл для збереження «Завдання 13-15-20-с». (Черепашка.)
21. Створіть програму, після виконання якої, на полотні Python Turtle Graphics за допомогою модуля Черепашка буде побудовано правильний (рівносторонній) n -кутник довжину сторони та їх кількість якого визначає користувач у процесі виконання програми (розмір та товщину сторони, колір заливки оберіть довільним чином). Файл для збереження «Завдання 13-15-21-с». (Черепашка.)

13.16. Цикл while

Цикл while має такий вигляд:

<Початкове значення>

```
while <Умова>:
```

```
    <Інструкції>
```

```
    <Приріст>
```

```
[else:
```

```
    <Блок, що виконується, якщо не використовувався оператор  
break>]
```

Виконання інструкцій в циклі `while` продовжується до тих пір, доки умова виконується (логічний вираз після `while` істинний).

Послідовність роботи циклу `while`:

1. Змінній-лічильнику присвоюється початкове значення.
2. Перевіряється умова, якщо вона істинна, виконуються інструкції всередині циклу, інакше виконання циклу завершується.
3. Змінна-лічильник змінюється на величину, зазначену в параметрі `<Приріст>`.
4. Перехід до пункту 2.
5. Якщо всередині циклу не використовувався оператор `break`, то після завершення виконання циклу буде виконано блок в інструкції `else`.
Даний блок не є обов'язковим.

У якості прикладу розглянемо програму за допомогою якої виводяться числа від 1 до 27, використовуючи цикл `while`.

```
i = 1          # <Початкове значення>
while i < 28:   # <Умова>
    print(i)    # <Інструкції>
    i=i+1       # <Приріст>
input()
```

Зауваження

Якщо `<Приріст>` не вказано, то цикл буде нескінченним. Щоб перервати нескінченний цикл, слід натиснути комбінацію клавіш `<Ctrl> + <C>`. В результаті генерується виключення `KeyboardInterrupt`, і виконання програми буде зупинено. Слід враховувати, що перервати таким чином можна тільки цикл, який виводить дані.

За допомогою циклу `while` можна перебирати і елементи різних структур. Але в цьому випадку слід пам'ятати, що цикл `while` працює повільніше циклу `for`.

Перехід на наступну ітерацію циклу. Оператор `continue`

Оператор `continue` дозволяє перейти до наступної ітерації (повторювання) циклу до завершення виконання всіх інструкцій всередині циклу. Як приклад виведемо всі числа від 1 до 50, крім чисел від 17 до 27 включно.

```
for i in range(1, 51):
    if 16 < i < 28:
        continue    # Переходимо до наступної ітерації
    print(i)
```

Переривання циклу. Оператор `break`

Оператор `break` дозволяє перервати виконання циклу достроково. Для прикладу виведемо всі числа від 1 до 27 ще одним способом.

```
i = 1
while True:
    if i > 27: break # Перериваємо цикл
    print(i)
    i = i + 1
input()
```

Тут ми в умові вказали значення True. У цьому разі висловлення всередині циклу будуть виконуватися нескінченно. Однак використання оператора break перериває його виконання, як тільки надруковано 27 рядків.

УВАГА!

Оператор break перериває виконання циклу, а не програми, іншими словами, далі буде виконана інструкція, наступна за циклом.

Цикл while спільно з оператором break зручно використовувати для отримання невизначеної заздалегідь кількості даних від користувача. Як приклад розглянемо програму для знаходження суми невизначеної кількості чисел.

```
print("Введіть слово 'стоп' для отримання результату")
summa = 0
while True:
    x = input("Введіть число: ")
    if x == "стоп": break          # Вихід з циклу
    x = float(x)                  # Перетворюємо рядок у число
    summa = summa+x
print("Сума чисел дорівнює:", summa)
input()
```

Процес введення 4 чисел і отримання суми виглядає так:

```
Введіть слово 'стоп' для отримання результату
Введіть число: 15
Введіть число: 25
Введіть число: 33
Введіть число: 17
Введіть число: стоп
Сума чисел дорівнює: 90.0
```

13.17. Практикум. Завдання 13.17

Складіть програму для обчислення найбільшого спільного дільника (НСД) двох натуральних чисел. Збережіть її у файлі «Завдання 13-17».

Технологія виконання

1. Створіть новий файл та збережіть його під ім'ям «Завдання 13-17».
2. Для створення програми використаємо алгоритм знаходження НСД, який полягає у наступному:

⇒ від більшого числа віднімаємо менше і різницю ставимо на місце більшого числа;

⇒ повторюємо це процес доти, доки числа не стануть рівними;

⇒ це і буде НСД двох цих чисел.

3. Результат реалізації даного алгоритму на мові Python може бути, наприклад, таким:


```

a=int(input("a="))
b=int(input("b="))
m=a
n=b
while a!=b:
    if a>b:
        a=a-b
    else:
        b=b-a
print("НСД(", m, ", ", n, ")=", a)
input()

```

4. Виконайте програму для кількох наборів даних.
 5. Перегляньте результати виконання та при потребі зробіть необхідні корективи.
- Перевірте, виконавши програму.
6. Закрийте IDLE.

13.18. Практикум. Завдання 13.18

Дано натуральне число n . Складіть програму для підрахунку кількості цифр даного числа. Збережіть її у файлі «Завдання 13-18».

Технологія виконання

Кількість цифр у числі n невідомо, тому необхідно використовувати оператор `while`. Використання `for` вимагає або введення додаткових змін, або штучного

m	k
75490	0
7549	1
754	2
75	3
7	4
0	5

виходу з циклу. Присвоїмо змінній m значення змінної n . Підрахунок кількості цифр почнемо з останніх цифр числа. Збільшимо лічильник цифр на одиницю (k). Число (m) зменшимо в 10 разів, тим самим прибираючи з нього останню цифру (вже підраховану). Далі з отриманим числом виконаємо туж послідовність дій і т. д., доки число не стане рівним нулю. Розглянемо ручне «трасування». Нехай введено число 75490. Результат трасування наведено у таблиці. Отже кінцеве значення

змінної k дорівнює 5. Кількість цифр – 5.

1. Створіть новий файл та збережіть його під ім'ям «Завдання 13-18».
 2. Після проведеного аналізу створення відповідної програми не повинно викликати труднощів.
 3. Один з можливих варіантів розв'язання наведено у лістингу [програми](#).
 4. Виконайте програму для кількох наборів даних.
 5. Перегляньте результати виконання та при потребі зробіть необхідні корективи.
- Перевірте, виконавши програму.
6. Закрийте IDLE.

13.19. Практикум. Завдання для самостійного виконання

1. Складіть програму для знаходження суми цифр натурального числа n . Ім'я файлу для збереження «Завдання 13-19-1-с».
2. Складіть програму для визначення того, скільки раз задана цифра зустрічається у натуральному числі n . Ім'я файлу для збереження «Завдання 13-19-2-с».

3. Складіть програму для знаходження найбільшої цифри заданого натурального числа n . Ім'я файлу для збереження «Завдання 13-19-3-с».
4. У багажник автомобіля вантажать овочі і фрукти з дачі: картоплю, капусту, моркву, яблука, груші та ін. Обсяг багажника дорівнює V л. Продукти кладуть послідовно, обсяг кожного вантажу відомий у літрах. Потрібно сказати в який момент (визначити номер вантажу) багажник переповниться. Ім'я файлу для збереження «Завдання 13-19-4-с».
5. Напишіть програму, яка буде підсумовувати числа, що вводяться з клавіатури до тих пір, доки вони від'ємні. Ім'я файлу для збереження «Завдання 13-19-5-с».
6. Створіть програму, яка буде підсумовувати числа, які вводяться з клавіатури до тих пір, доки вони парні. Ім'я файлу для збереження «Завдання 13-19-6-с».
7. Дано число натуральне число n . Надрукувати ті натуральні числа, квадрат яких не перевищує n . Ім'я файлу для збереження «Завдання 13-19-7-с».
8. Дано число a ($1 \leq a \leq 1,5$). Серед чисел $1 + \frac{1}{2}, 1 + \frac{1}{3}, 1 + \frac{1}{4}, \dots$ (зауважимо, що кожне наступне число в послідовності менше попереднього) знайдіть найперше, менше a . Ім'я файлу для збереження «Завдання 13-19-8-с».
9. Напишіть програму, яка запитує у користувача числа до тих пір, доки кожне наступне число більше попереднього. В кінці програма повідомляє, скільки чисел було введено. Ім'я файлу для збереження «Завдання 13-19-9-с».
10. Дано натуральне число, у якому всі цифри різні. Визначити порядковий номер його максимальної цифри, рахуючи номери від кінця числа. Ім'я файлу для збереження «Завдання 13-19-10-с».
11. *Написати програму перевірки, чи є число X степенем числа 2. Ім'я файлу для збереження «Завдання 13-19-11-с». [15, стор. 2]
12. *Ім'я файлу для збереження «Завдання 13-19-12-с».

Загальний бал:	100
Обмеження часу:	100 мс
Обмеження реального часу:	5 с
Обмеження пам'яті	64 М

Степан виписує на листочку усі цілі числа від 1 до N в кілька груп, при цьому якщо одне число ділиться на інше, то вони обов'язково будуть у різних групах.

Наприклад, якщо $N = 9$, то отримаємо 4 групи:

Перша група: 1.

Друга група: 2 3 7.

Третя група: 4 5 6.

Четверта група: 8 9.

Очевидно, що оскільки, будь-яке число ділиться на 1, то одна група завжди буде складатись тільки з числа 1, а от інші групи можуть бути створені різними способами.

Допоможіть Степану написати програму, яка визначає мінімальне число груп, на яке можна розбити усі числа від 1 до N у відповідності до наведеної вище умови.

Формат вхідних даних:

Перший рядок вхідних даних містить єдине число N ($1 \leq N \leq 10^9$).

Формат вихідних даних:

Виведіть одне число – знайдену мінімальну кількість груп.

Приклади

Вхідні дані	Результат роботи
9	4

[14, стор. 13]

Питання для самоконтролю

1. Які операції порівняння використовуються у мові Python 3? Який їх синтаксис?
2. Навіщо використовуються логічні операції? Перерахуйте їх.
3. Який синтаксис та правила виконання оператора розгалуження `if...else`?
4. Коли доцільно використовувати цикл `for`? Який його синтаксис та правила виконання?
5. Який синтаксис та правила виконання циклу `while`? Коли доцільно його використовувати?
6. Яке призначення оператора `continue`?
7. За допомогою якого оператора можна перервати виконання циклу `while` достроково?

14. Функції користувача

Якщо певна послідовність команд при створенні програми повторюється, то її (послідовність) можна подати у вигляді функції.

Функція – це фрагмент коду, який можна викликати з будь-якого місця програми. У попередніх розділах ми вже використовували вбудовані функції мови Python. У цьому розділі ми розглянемо створення призначених для користувача функцій, які дозволять зменшити надмірність програмного коду та підвищити його структурованість.

14.1. Створення функції та її виклик

Функція описується за допомогою ключового слова `def` наступним чином:

```
def <Ім'я функції> ([<Параметри>]) :  
    ["Рядок документування"]  
    <Тіло функції(послідовність команд)>  
    [return <Значення>]
```

Ім'я функції має бути унікальним ідентифікатором, що складається з букв (краще латинських, але можна використовувати і кирилицю), цифр і знаків підкреслення, причому ім'я функції не може починатися з цифри. Як ім'я не можна використовувати ключові слова, крім того, слід уникати збігів з назвами вбудованих ідентифікаторів. Регістр символів у назві функції має значення.

Після імені функції в круглих дужках можна вказати один або декілька параметрів через кому. Якщо функція не приймає параметри, то просто вказуються круглі дужки. Після круглих дужок ставиться двокрапка.

Тіло функції складається з інструкцій, які виділяються однаковою кількістю пробілів зліва. Функції закінчується інструкцією, перед якою менша кількість пробілів (ця інструкція не входить до тіла функції). Якщо тіло функції не містить інструкцій, то всередині необхідно розмістити оператор `pass`. Цей оператор зручно використовувати на етапі налагодження програми, коли ми визначили функцію, а тіло плануємо дописувати пізніше.

Необов'язкова інструкція `return` дозволяє повернути значення з функції. Після виконання цієї інструкції виконання функції буде зупинено. Це означає, що інструкції після оператора `return` ніколи не будуть виконані.

Інструкції `return` може не бути взагалі. У цьому випадку виконуються всі інструкції всередині функції і повертається значення `None`.

Розглянемо використання функції при складанні програми для знаходження найбільшого з 5 чисел (за умови, що функції `max()`, `min()`, тощо використовувати забороняється).

На початку визначимо функцію для знаходження більшого з двох чисел, тоді програма набуде такого вигляду:

```

def maxx2(x, y):
    if x>y:
        return x
    else:
        return y
a1=float(input('Введіть перше число: '))
a2=float(input('Введіть друге число: '))
a3=float(input('Введіть третє число: '))
a4=float(input('Введіть четверте число: '))
a5=float(input('Введіть п'яте число: '))
max=maxx2(a1, a2)
max=maxx2(a3, max)
max=maxx2(a4, max)
max=maxx2(a5, max)
print('Максимальне число дорівнює', max)
input()

```

Для даних 10, -7, 10, 25.4, 100 отримаємо результат виконання у середовищі IDLE:

```

Введіть перше число: 10
Введіть друге число: -7
Введіть третє число: 10
Введіть четверте число: 25.4
Введіть п'яте число: 100
Максимальне число дорівнює 100.0

```

При виклику функції значення передаються всередині круглих дужок через кому. Якщо функція не приймає параметрів, то вказуються тільки круглі дужки. Необхідно також зауважити, що кількість параметрів у визначенні функції має збігатися з кількістю параметрів при виклику, інакше буде виведено повідомлення про помилку.

Ім'я змінної у виклику функції може не збігатися з ім'ям змінної у визначенні функції. Змінні, зазначені у визначенні функції, є локальними і доступні тільки всередині функції.

Як ви вже знаєте, все в мові Python є об'єктом, наприклад рядки, списки і навіть самі типи даних. Функції не є винятком. Інструкція `def` створює об'єкт, що має тип `function`, і зберігає посилання на нього в ідентифікаторі, зазначеному після інструкції `def`. Таким чином, ми можемо зберегти посилання на функцію в іншій змінній. Для цього назва функції вказується без круглих дужок. Наприклад:

```
m2 = maxx2.
```

Розташування визначень функцій

Всі інструкції в програмі виконуються послідовно зверху вниз. Це означає, що перш ніж використовувати ідентифікатор в програмі, його необхідно попередньо визначити, присвоївши йому значення. Тому визначення функції має бути розташоване перед викликом функції.

Щоб уникнути помилки, визначення функції розміщують на самому початку програми після підключення модулів або в окремому файлі, який називається *модулем*.

14.2. Практикум. Завдання 14.2*

Складіть програму для знаходження найбільшого спільного дільника п'ятих натуральних чисел. Збережіть її у файлі «Завдання 14-2».

Технологія виконання

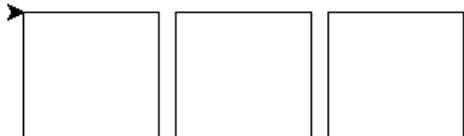
1. Створіть новий файл та збережіть його під ім'ям «Завдання 14-2».
2. Створимо функцію для знаходження НСД двох натуральних чисел, скориставшись результатом виконання «Завдання 13.12». Вона може бути, наприклад, такою:

```
def ncd(a, b):  
    while a!=b:  
        if a>b:  
            a=a-b  
        else:  
            b=b-a  
    return a
```

3. Подальша розробка програми не являє особливих складнощів. Введіть програму, збережіть її та виконайте для різних наборів даних.
4. Перегляньте результати виконання та при потребі зробіть необхідні корективи. Перевірте, виконавши програму.
5. У випадку виникнення труднощів можете скористатися [лістингом програми](#).
6. Закрийте IDLE.

14.3. Практикум. Завдання 14.3 (Черепашка)

Створіть програму, після виконання якої, на полотні Python Turtle



Graphics за допомогою модуля Черепашка буде побудовано **n** квадратів, довжиною сторони **b** пікселів та на відстані **c** один від одного (значення **n**, **b** та **c** визначає користувач у процесі діалогу користувача з комп'ютером під час виконання програми). Після

виконання побудов виконавець Черепашка повертається у початкове положення (рис. 14.3.1). Збережіть програму у файлі з ім'ям «Завдання 14-3(Черепашка)».

Рис. 14.3.1

Технологія виконання

1. Відкрийте інтегроване середовище IDLE.
2. Створіть новий файл та збережіть його під ім'ям «Завдання 14-3(Черепашка)».
3. Для побудови квадрату з певною довжиною сторони використаємо функцією з параметром:

```
def sq(a):  
    for i in range(4):  
        down()  
        forward(a)  
        right(90)  
        up()
```

4. Щоб побудувати потрібну кількість квадратів застосуємо цикл for:

```
for i in range(n):  
    sq(b)  
    fd(b+c)
```

5. Запустіть створену програму на виконання.
6. Перегляньте результати та при потребі зробіть необхідні корективи. Перевірте, виконавши програму.
7. Збережіть зміни та закрийте IDLE.

14.4. Практикум. Завдання для самостійного виконання

1. Складіть програму для знаходження НСК 5-ти натуральних чисел. Збережіть її у файлі «Завдання 14-4-1-с».
2. Створіть програму, після виконання якої, на полотні Python Turtle Graphics за допомогою модуля Черепашка буде побудовано **n**

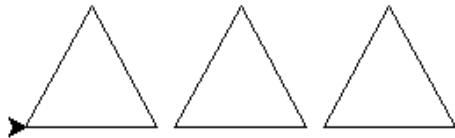


Рис. 14.4.2

правильних трикутників, довжиною сторони **b** пікселів та на відстані **c** один від одного (значення **n**, **b** та **c** визначає користувач у процесі діалогу користувача з комп'ютером під час виконання програми). Після виконання побудов виконавець Черепашка

повертається у початкове положення (рис. 14.4.2). Збережіть програму у файлі з ім'ям «Завдання 14-4-2-с (Черепашка)».

3. Створіть програму, після виконання якої, на полотні Python Turtle Graphics за допомогою модуля Черепашка буде побудовано соти (складаються з правильних шестикутників) згідно разка, наведеного на рисунку 1.14.4.3. Розміри шестикутників оберіть на ваш розсуд. Збережіть програму у файлі з ім'ям «Завдання 1-14-4-3-с(Черепашка)».

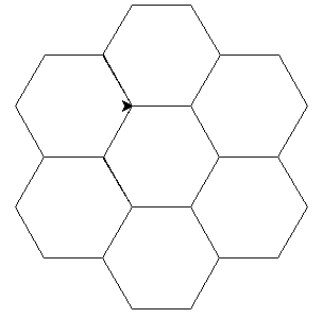


Рис. 14.4.3

4. Створіть програму, після виконання якої, на полотні Python Turtle Graphics за допомогою модуля Черепашка буде побудовано соти (складаються з правильних шестикутників) згідно разка, наведеного на рисунку 14.4.3. Розміри шестикутників визначає користувач у процесі діалогу користувача з комп'ютером під час виконання програми. Збережіть програму у файлі з ім'ям «Завдання 14-4-4-с(Черепашка)».

Питання для самоконтролю

1. Навіщо використовуються функції користувача?
2. Як описується функція та яким має бути її ім'я?
3. Коли та навіщо використовується інструкція `return`?
4. Які правила розташування визначень функцій користувача?

15. Списки та кортежі

Списки та кортежі – це набори об'єктів. Кожен елемент набору містить лише посилання на об'єкт. З цієї причини списки і кортежі можуть містити об'єкти довільного типу даних і мати необмежену ступінь вкладеності. Позиція елемента в наборі задається індексом. Зверніть увагу на те, що нумерація елементів починається з 0, а не з 1.

У мові Python списки є аналогом масивів у інших мовах програмування, але вони мають більш широкі можливості. З одного боку, списки не обмежені одним типом елементів, з іншого, розмір списків не обмежений, завдяки чому кількість елементів списку можна збільшувати та зменшувати по мірі необхідності.

Списки та кортежі є впорядкованими послідовностями елементів. Як і всі послідовності, вони підтримують звернення до елемента за індексом, отримання зрізу, конкатенацію (оператор +), повторення (оператор *), перевірку на входження (оператор in).

Списки відносяться до змінюваних типів даних. Це означає, що ми можемо не тільки отримати елемент за індексом, а й змінити його. Кортежі відносяться до незмінних типів даних. Іншими словами, можна отримати елемент за індексом, але змінити його не можна.

15.1. Створення списку

Створити список можна наступними способами:

- за допомогою функції `list(<Послідовність>)`. Функція дозволяє перетворити будь-яку послідовність в список. Якщо параметр не вказано, то створюється порожній список.

Приклад:

```
>>> list() # Створюємо порожній список.
[]
>>> list('Рядок') # Перетворюємо рядок у список.
['Р', 'я', 'д', 'о', 'к']
>>> list((7, 2, 5, 5, 9)) # Перетворюємо кортеж у список.
[7, 2, 5, 5, 9]
```

- вказавши всі елементи списку всередині квадратних дужок:

```
>>> arr=[5, 1, 'py', 8, 't']
>>> arr
[5, 1, 'py', 8, 't']
```

- заповнивши список поелементно за допомогою методу `append()`:

```
>>> arr=[] # Створюємо порожній список.
>>> arr.append(7) # Додаємо елемент 7(індекс 0).
>>> arr.append(-10) # Додаємо елемент -10(індекс 1).
>>> arr
[7, -10]
```

Наведемо фрагменти коду з прикладами введення значень елементів списку з клавіатури.

Фрагмент програми, при виконанні якого буде введено значення 4-х елементів створеного списку.


```
A=[]
for i in range(1, 4):
    print("A[" ,i, "]=")
    A.append(input())
print(A)
```

Фрагмент програми, при виконанні якого буде введено значення n (також визначається користувачем) елементів створеного списку A.

```
A=[]
print("Введіть кількість елементів масиву.")
n=int(input("n="))
for i in range(1, n+1):
    print("A[" ,i, "]=")
    A.append(input())
print(A)
```

Фрагмент програми, при виконанні якого буде введено значення n (визначається користувачем) елементів створеного списку A. Значеннями елементів списку будуть цілі числа.

```
A=[]
print("Введіть кількість елементів масиву.")
n=int(input("n="))
for i in range(1, n+1):
    print("A[" ,i, "]=")
    A.append(int(input()))
print(A)
```

Ввести масив цілих чисел, елементи якого записані у стрічку через проміжок можна ще за допомогою таких інструкцій:

- `A = list(map(int, input().split()))`, в інструкції використовується функція [map](#) та метод [split](#) для роботи з рядками;
- `A = [int(i) for i in input().split()]`, тут застосовується [list comprehension](#) (генерування списку).

15.2. Основні операції над списками

Звернення до елементів списку здійснюється за допомогою квадратних дужок, в яких вказується індекс елемента. Нумерація елементів списку починається з нуля.

Приклад:

```
>>> arr=[7, 8, 'm']
>>> arr[0]
7
>>> arr[2]
'm'
```

За допомогою позиційного присвоювання можна присвоїти значення елементів списку будь-яким змінним. Кількість елементів праворуч і ліворуч від оператора = має збігатися, інакше буде виведено повідомлення про помилку:

```
>>> x, y = [7, 2, 10]
Traceback (most recent call last):
  File "<pyshell#6>", line 1, in <module>
    x, y = [7, 2, 10]
ValueError: too many values to unpack (expected 2)
```


В Python 3 при позиційному присвоюванні перед однією із змінних зліва від оператора = ви можете вказати зірочку. У цієї змінної буде зберігатися список, що складається з "зайвих" елементів. Якщо таких елементів немає, то список буде порожнім:

```
>>> x, y, *z = [7, 21, 3, 14, 5]
>>> x
7
>>> z
[3, 14, 5]
```

Оскільки нумерація елементів списку починається з 0, індекс останнього елемента буде на одиницю менше кількості елементів. Отримати кількість елементів списку дозволяє функція `len()`:

```
>>> arr = [7, 2, 8, 7, 5]
>>> len(arr)
5
```

Якщо елемент, що відповідає вказаному індексу, відсутній у списку, то збуджується виключення `IndexError`:

```
>>> arr = [7, 12, 8, 7, 15]
>>> arr[5]
Traceback (most recent call last):
  File "<pyshell#3>", line 1, in <module>
    arr[5]
IndexError: list index out of range
```

Оскільки списки відносяться до змінюваних типів даних, то ми можемо змінити елемент за індексом:

```
>>> arr = [7, 12, 8, 17, 1]
>>> arr[0] = 100
>>> arr
[100, 12, 8, 17, 1]
```

Крім того, списки підтримують операцію вилучення зрізу, яка повертає зазначений фрагмент списку. Формат операції:

```
[<Початок>: <Кінець>: <Крок>]
```

Всі параметри є необов'язковими. Якщо параметр <Початок> не вказано, то використовується значення 0. Якщо параметр <Кінець> не вказано, то повертається фрагмент до кінця списку. Слід також зауважити, що елемент з індексом, зазначеним у цьому параметрі, не входить до фрагменту, що повертається. Якщо параметр <Крок> не вказано, то використовується значення 1. В якості значення параметрів можна вказати від'ємні значення.

Тепер розглянемо кілька прикладів. Спочатку отримаємо поверхневу копію списку:

```
>>> arr = [11, 72, 35, 4, 5]
>>> k=arr[:]      #Створюємо поверхневу копію.
>>> k
[11, 72, 35, 4, 5]
```

Створимо копію списку у якій елементи розташовані у зворотному порядку:

```
>>> m=arr[::-1]
>>> m
[5, 4, 35, 72, 11]
>>> arr
[11, 72, 35, 4, 5]
```

Виведемо список без першого і останнього елементів:

```
>>> arr[:-1] #Без останнього елемента.
[11, 72, 35, 4]
>>> arr[1:] #Без першого елемента.
[72, 35, 4, 5]
```

Отримаємо перші два елементи списку:

```
>>> arr[0: 2] #Символ з індексом 2 не входить до діапазону.
[11, 72]
```

А тепер отримаємо останній елемент:

```
>>> arr[-1:] # Останній елемент списку.
[5]
```

І, нарешті, виведемо фрагмент від другого елементу до четвертого включно:

```
>>> arr[1:4]
[72, 35, 4]
```

З'єднати два списки у один дозволяє оператор +. Результатом об'єднання буде новий список:

```
>>> arr1=[7,6,15]
>>> arr2=[17,8,15,25]
>>> arr3=arr2+arr1
>>> arr3
[17, 8, 15, 25, 7, 6, 15]
```

Замість оператора + можна використовувати оператор +=. Слід враховувати, що у цьому випадку елементи додаються до поточного списку:

```
>>> arr = [1, 2, 3, 4, 5]
>>> arr += [6, 7, 8, 9]
>>> arr
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Крім розглянутих операцій, списки підтримують операцію повторення і перевірку на входження. Повторити список вказану кількість разів можна за допомогою оператора *, а виконати перевірку на входження елемента у список дозволяє оператор in:

```
>>> [7, 5]*3
[7, 5, 7, 5, 7, 5]
>>> 7 in [8, 3, 7, 5]
True
```

15.3. Багатовимірні списки

Будь-який елемент списку може містити об'єкт довільного типу. Наприклад, елемент списку може бути числом, рядком, списком, кортежем, тощо. Створити вкладений список можна, наприклад, так:

```
>>> arr = [ [7, 2, 4], [14, 5, 9], [7, 10, 19] ]
```

Як ви вже знаєте, вираз всередині дужок може розташовуватися на декількох рядках. Отже, попередній приклад можна записати інакше:

```
>>> arr = [  
    [7, 2, 4],  
    [14, 5, 9],  
    [7, 10, 19]  
]
```

Щоб отримати значення елемента у вкладеному списку, слід вказати два індекси:

```
>>> arr[2][1]  
10
```

Елементи вкладеного списку також можуть мати елементи довільного типу. Кількість вкладень необмежена. Таким чином, ми може створити об'єкт будь-якого ступеня складності. В цьому випадку для доступу до елементів вказується кілька індексів поспіль.

Приклад:

```
>>> arr = [ [7, ["a", "b"], 3], [4, 5, 6], [17, 8, 9] ]  
>>> arr[0][1][1]  
'b'
```

15.4. Перебір елементів списку

Перебрати всі елементи списку можна за допомогою циклу `for`:

```
>>> arr = [11, 25, 3, 4, 15]  
>>> for i in arr: print(i, end=" ")  
  
11 25 3 4 15
```

Слід зауважити, що змінну `i` всередині циклу можна змінити, але якщо вона посилається на незмінний тип даних (наприклад, число або рядок), то це не відіб'ється на вихідному списку.

Щоб отримати доступ до кожного елемента, можна, наприклад, скористатися функцією `range()` для генерації індексів.

```
arr = [17, 2, 37, 24]  
for i in range(len(arr)):  
    arr[i] *= 2  
print(arr)
```

Результат виконання:

```
[34, 4, 74, 48]
```

Крім того, перебрати елементи можна за допомогою циклу `while`. Але в цьому випадку слід пам'ятати, що цикл `while` працює повільніше циклу `for`. Як приклад помножимо кожний елемент списку на 3, використовуючи цикл `while`:

```
arr = [11, 2, 5, 1]  
i, c = 0, len(arr)  
while i < c:  
    arr[i] *= 3  
    i += 1  
print(arr)
```

Результат виконання:

```
[33, 6, 15, 3]
```

15.5. Додавання і видалення елементів списку

Для додавання і видалення елементів списку використовуються наступні методи:

- `append(<Об'єкт>)` – додає один об'єкт у кінець списку. Метод змінює поточний список і нічого не повертає.

Приклад:

```
>>> arr = [11, 2, 3, 7]
>>> arr.append(5) # Додаємо число
>>> arr
[11, 2, 3, 7, 5]
>>> arr.append([5, 6]) # Додаємо список
>>> arr
[11, 2, 3, 7, 5, [5, 6]]
```

- `extend(<Послідовність>)` – додає елементи послідовності у кінець списку. Метод змінює поточний список і нічого не повертає. Приклад:

```
>>> arr = [1, 2, 3]
>>> arr.extend([4, 5, 6])
>>> arr.extend((7, 8, 9))
>>> arr.extend("Python")
>>> arr
[1, 2, 3, 4, 5, 6, 7, 8, 9, 'P', 'y', 't', 'h', 'o', 'n']
```

- Додати кілька елементів можна за допомогою операції конкатенації (+) або оператора +=:

```
>>> arr=[6, 12, 3]
>>> arr+[14, 5, 9] #Повертає новий список
[6, 12, 3, 14, 5, 9]
>>> arr+=[4, 5, 6] #Змінює поточний список
>>> arr
[6, 12, 3, 4, 5, 6]
```

- `insert (<Індекс>, <Об'єкт>)` – додає один об'єкт в зазначену позицію. Інші елементи зміщуються. Метод змінює поточний список і нічого не повертає.

Приклади:

```
>>> arr = [11, 5, 13]
>>> arr.insert(0, 0) #Додаємо 0 на початок списку
>>> arr
[0, 11, 5, 13]
>>> arr.insert(-2, 22) #Можна вказати від'ємні числа
>>> arr
[0, 11, 22, 5, 13]
>>> arr.insert(3, 77) #Вставляємо 77 в позицію 3
>>> arr
[0, 11, 22, 77, 5, 13]
>>> arr.insert(6, [44, 55]) #Додаємо список
>>> arr
[0, 11, 22, 77, 5, 13, [44, 55]]
```

- `pop ([<Індекс>])` – видаляє елемент, розташований за вказаним індексом, і повертає його. Якщо індекс не вказано, то видаляє і повертає останній елемент списку. Якщо елемента з вказаним індексом немає або список порожній, збуджується виключення `IndexError`.

Приклади:

```
>>> arr=[7, 4, 22, 77, 14]
>>> arr.pop()
14
>>> arr
[7, 4, 22, 77]
>>> arr.pop(0)
7
>>> arr
[4, 22, 77]
>>> arr.pop(1)
22
>>> arr
[4, 77]
```

- Видалити елемент списку дозволяє також оператор `del`:

```
>>> arr=[5, 6, 7, 8, 9]
>>> del arr[4]
>>> arr
[5, 6, 7, 8]
>>> del arr[1:2]
>>> arr
[5, 7, 8]
>>> del arr[1:3]
>>> arr
[5]
```

- `remove(<Значення>)` – видаляє перший елемент, що містить вказане значення. Якщо елемент не знайдений, то збуджується виключення `ValueError`. Метод змінює поточний список і нічого не повертає.

Приклади:

```
>>> arr=[7, 5, 99, 100, 12, 99]
>>> arr.remove(99)
>>> arr
[7, 5, 100, 12, 99]
```

Якщо необхідно видалити всі повторювані елементи списку, то можна список перетворити у множину, а потім множину назад перетворити в список. Зверніть увагу на те, що список має містити тільки незмінні об'єкти (наприклад, числа, рядки або кортежі). В іншому випадку порушується виключення `TypeError`.

Приклад:

```
>>> arr = [1, 2, 3, 1, 1, 2, 2, 3, 3]
>>> s = set(arr)
>>> s
{1, 2, 3}
>>> arr = list(s)
>>> arr
[1, 2, 3]
```

15.6. Пошук елемента у списку

Виконати перевірку на входження елемента в список дозволяє оператор `in`. Якщо елемент входить до списку, то повертається значення `True`, в іншому випадку – `False`.

Приклад:

```
>>> arr = [7, 5, 100, 12, 99]
>>> 100 in arr
True
>>> 77 in arr
False
```

Проте, оператор `in` не дає ніякої інформації про місцезнаходження елемента всередині списку. Щоб дізнатися індекс елемента всередині списку, слід скористатися методом `index()`. Формат методу:

```
index(<Значення> [, <Початок> [, <Кінець>]])
```

Метод `index()` повертає індекс елемента, що має вказане значення. Якщо значення не входить до списку, то збуджується виключення `ValueError`. Якщо другий і третій параметри не зазначені, то пошук буде проводитися з початку списку.

Приклад:

```
>>> arr=[3, 4, 9, 3, 5, 7, 7]
>>> arr.index(3)
0
>>> arr.index(9, 1, 5)
2
>>> arr.index(6)
Traceback (most recent call last):
  File "<pyshell#3>", line 1, in <module>
    arr.index(6)
ValueError: 6 is not in list
```

Дізнатися загальну кількість елементів із зазначеним значенням дозволяє метод `count(<Значення>)`. Якщо елемент не входить в список, то повертається значення 0.

Приклад:

```
>>> arr=[3, 4, 9, 3, 5, 7, 7, 3]
>>> arr.count(3)
3
>>> arr.count(7)
2
>>> arr.count(6)
0
```

За допомогою функцій `max()` і `min()` можна дізнатися максимальне і мінімальне значення списку відповідно. Приклад:

```
>>> arr=[3, 4, 9, 3, 5, 7, 7, 3]
>>> max(arr)
9
>>> min(arr)
3
```

15.7. Перевертання та перемішування списку

Метод `reverse()` змінює порядок проходження елементів списку на протилежний. Метод змінює поточний список і нічого не повертає.

Приклад:

```
>>> arr = [3, 4, 9, 3, 5, 7, 7, 3]
>>> arr.reverse()
>>> arr
[3, 7, 7, 5, 3, 9, 4, 3]
```

Функція `shuffle(<Список> [, <Число від 0.0 до 1.0>])` з модуля `random` "перемішує" список випадковим чином. Функція перемішує сам список і нічого не повертає. Якщо другий параметр не вказано, то використовується значення, що повертається функцією `random()`.

Приклад:

```
>>> import random
>>> arr = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>> random.shuffle(arr)
>>> arr
[7, 4, 9, 6, 3, 10, 2, 5, 8, 1]
```

15.8. Вибір елементів випадковим чином

Отримати елементи зі списку випадковим чином дозволяють наступні функції з модуля `random`:

- `choice(<Послідовність>)` – повертає випадковий елемент з будь-якій послідовності (рядку, списку, кортежу):

```
>>> arr = [1, 2, 3, 4, 5, 6, 7]
>>> import random
>>> random.choice(arr)
7
>>> random.choice(arr)
4
```

- `sample(<Послідовність>, <Кількість елементів>)` – повертає список із зазначеною кількістю елементів. У цей список потраплять елементи з послідовності, вибрані випадковим чином. Як послідовності можна вказати будь-які об'єкти, що підтримують ітерації.

Приклад:

```
>>> arr = [1, 2, 3, 4, 5, 6, 7]
>>> import random
>>> random.sample(arr, 4)
[7, 2, 1, 3]
```

15.9. Сорткування списку

Відсортувати список дозволяє метод `sort()`. Метод має такий вигляд:
`sort([key = None] [, reverse = False])`

Всі параметри є необов'язковими. Метод змінює поточний список і нічого не повертає. Відсортуємо список по зростанню з параметрами за замовчуванням:

```
>>> arr = [4, 1, 9, 14, 5, 7, 5]
>>> arr.sort()
>>> arr
[1, 4, 5, 5, 7, 9, 14]
```

Щоб відсортувати список за спаданням, слід в параметрі `reverse` вказати значення `True`:

```
>>> arr = [4, 1, 9, 14, 5, 7, 5]
>>> arr.sort(reverse=True)
>>> arr
[14, 9, 7, 5, 5, 4, 1]
```

При сортуванні фактично порівнюються два об'єкти, і який з них "менше", ставиться перед тим, який "більше". Поняття "більше" і "менше" досить умовні. І

якщо для чисел все просто - числа розставляються в порядку зростання, то для рядків і інших об'єктів ситуація складніша. Зокрема, рядки оцінюються по перших символах. Якщо перші символи рівні, оцінюються другі символи і так далі. При чому цифровий символ вважається "менше", ніж алфавітний заглавний символ, а заглавний символ вважається меншим, ніж рядковий. Детальніше ми це розглянемо при вивченні рядків.

Таким чином, якщо в списку поєднуються рядки з верхнім та нижнім регістром, то ми можемо отримати не зовсім коректні результати, так як для нас рядок "bob" повинен стояти до рядка "Tom". І щоб змінити стандартну поведінку сортування, ми можемо передати в метод `sort()` в якості параметра функцію для зміни регістру символів у параметрі `key`:

```
>>> users = ["Tom", "bob", "alice", "Sam", "Bill"]
>>> users.sort(key=str.lower)
>>> users
['alice', 'Bill', 'bob', 'Sam', 'Tom']
```

15.10. Заповнення списку числами

Створити список, що містить діапазон чисел, можна за допомогою функції `range()`. Функція повертає об'єкт, що підтримує ітерації. Щоб з цього об'єкта отримати список, слід скористатися функцією `list()`.

Як приклад заповнимо список числами від 0 до 9:

```
>>> list(range(10))
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Створимо список, що складається з діапазону чисел від 1 до 14:

```
>>> list(range(1, 15))
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14]
```

А тепер список парних чисел з діапазону від 2 до 14:

```
>>> list(range(2, 15, 2))
[2, 4, 6, 8, 10, 12, 14]
```

Тепер змінимо порядок чисел на протилежний:

```
>>> list(range(14, 0, -1))
[14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
```

Якщо необхідно отримати список з випадковими числами (або випадковими елементами з іншого списку), то слід скористатися функцією `sample(<Послідовність>, <Кількість елементів>)` з модуля `random`.

Приклад:

```
>>> import random
>>> arr = [11, 2, 13, 4, 55, 6, 77, 8, 91, 10, 25]
>>> random.sample(arr, 5)
[91, 77, 11, 10, 25]
>>> random.sample(range(700), 3)
[305, 347, 645]
```

Створити список можна також за допомогою `list comprehension` (генерування (компонування) списку). Докладніше познайомитися з `list comprehension` ви можете скориставшись [додатком 7](#) даного посібника.

Нехай, наприклад, потрібно створити список із 20 випадкових цілих чисел таких що $-7 \leq A[i] \leq 7$. Тоді відповідна команда та результат її виконання можуть мати наступний вигляд:

```
>>> from random import randint
>>> A = [randint(-7, 7) for i in range(20)]
>>> A
[6, 4, -3, -1, -1, -3, -2, -2, 1, 5, 7, 7, 2, 2, -7, -5, 1, -6, 0, -2]
```

Якщо ж, наприклад, необхідно створити список із 5 випадкових дійсних чисел таких що $-7 \leq A[i] \leq 7$, то у цьому випадку відповідна команда та результат її виконання можуть мати наступний вигляд:

```
>>> from random import uniform
>>> A=[uniform(-7, 7) for i in range(5)]
>>> A
[6.047908377920264, 5.379689783298021, 1.02030022022792,
-5.9282700633688235, 5.49613669358814]
```

15.11. Кортежі

Кортежі, так само як і списки, є впорядкованими послідовностями елементів. Кортежі багато в чому аналогічні списками, але мають одну дуже важливу відмінність – змінити кортеж не можна. Можна сказати, що кортеж – це список, доступний «тільки для читання». Створити кортеж можна за допомогою таких дій:

- вказавши всі елементи через кому всередині круглих дужок (або без дужок):

```
>>> k1 = () #Створюємо порожній кортеж
>>> k2 = (5,) #Створюємо кортеж з одного елементу
>>> k3 = ('a', 'v', 44, 'Пітон') #Створюємо кортеж з 4 елементів
>>> k1, k2, k3
((), (5,), ('a', 'v', 44, 'Пітон'))
```

Зверніть особливу увагу на другий рядок прикладу. Щоб створити кортеж з одного елемента, необхідно в кінці вказати кому. Саме коми формують кортеж, а не круглі дужки. Якщо всередині круглих дужок немає ком, то буде створено об'єкт іншого типу. Пам'ятайте, що будь-який вираз в мові Python можна взяти у круглі дужки, а щоб отримати кортеж, необхідно вказати коми.

- за допомогою функції `tuple([<Послідовність>])`. Функція дозволяє перетворити будь-яку послідовність в кортеж. Якщо параметр не вказано, то створюється порожній кортеж. Приклад:

```
>>> tuple("String")
('S', 't', 'r', 'i', 'n', 'g')
>>> arr = [11, 2, 13, 4, 55, 6, 77, 8, 91, 10, 25]
>>> tuple(arr)
(11, 2, 13, 4, 55, 6, 77, 8, 91, 10, 25)
```

Позиція елемента в кортежі задається індексом. Зверніть увагу на те, що нумерація елементів кортежу (як і списку) починається з 0, а не з 1. Як і всі послідовності, кортежі підтримують звернення до елемента за індексом, отримання зрізу, конкатенацію (оператор +), повторення (оператор *), перевірку на входження (оператор in). Приклад:

```

>>> k = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
>>> k[0]
1
>>> k[::-1]
(10, 9, 8, 7, 6, 5, 4, 3, 2, 1)
>>> k[3:5]
(4, 5)
>>> 8 in k
True
>>> 15 in k
False
>>> (11, 27, 7) * 3
(11, 27, 7, 11, 27, 7, 11, 27, 7)
>>> (6, 0, 4)+(9, 0, 5, 7)
(6, 0, 4, 9, 0, 5, 7)

```

Кортежи відносяться до незмінних типів даних. Іншими словами, можна отримати елемент за індексом, але змінити його не можна:

```

>>> k = (7, 3, 9)
>>> k[0]
7
>>> k[0] = 55
Traceback (most recent call last):
  File "<pyshell#12>", line 1, in <module>
    k[0] = 55
TypeError: 'tuple' object does not support item assignment

```

Отримати кількість елементів кортежу дозволяє функція `len()`:

```

>>> k = (7, 3, 9, 15, 7)
>>> len(k)
5

```

Кортежі підтримують два методи:

- `index(<Значення>[, <Початок>[, <Кінець>]])` – повертає індекс елемента, що має вказане значення. Якщо значення не входить до кортежу, збуджується виключення `ValueError`. Якщо другий і третій параметри не зазначені, то пошук буде проводитися в усьому кортежі.

Приклад:

```

>>> k = (7, 21, 5, 2, 9)
>>> k.index(21)
1
>>> k.index(2, 1, 3)
Traceback (most recent call last):
  File "<pyshell#2>", line 1, in <module>
    k.index(2, 1, 3)
ValueError: tuple.index(x): x not in tuple
>>> k.index(2, 1, 4)
3

```

- `count(<Значення>)` – повертає кількість елементів із зазначеним значенням. Якщо елемент не входить в кортеж, то повертається значення 0. Приклад:

```

>>> k = (2, 2, 7, 2, 1, 5)
>>> k.count(2)
3
>>> k.count(7)
1
>>> k.count(77)
0

```

Інших методів у кортежів немає. Щоб здійснювати операції над кортежами, слід скористатися вбудованими функціями, призначеними для роботи з послідовностями, які ми розглядали при вивченні списків.

15.12. Практикум. Завдання 15.12

Знайти суму додатних елементів одновимірного масиву (таблиці) всі елементи якого є цілими числами. Збережіть програму у файлі «Завдання 15-12».

Технологія виконання

1. Створіть новий файл та збережіть його під ім'ям «Завдання 15-12».
2. Для розв'язування цієї задачі використаємо тип даних список.
3. Створюємо порожній список.
4. Для введення елементів списку можна скористатися способом, розглянутим у пункті 15.1.

5. Залишається знайти суму додатних елементів списку.

6. Програмна реалізація може бути, наприклад, такою:

```

A=[]
print("Введіть кількість елементів масиву.")
n=int(input("n="))
for i in range(1, n+1):
    print("A[" ,i, "]= ", end=" ")
    A.append(int(input()))
print(A)
s=0
for i in range(0, n):
    if A[i]>0:
        s=s+A[i]
print('S = ', s)
input()

```

7. Виконайте програму для кількох наборів даних.
8. Перегляньте результати виконання та при потребі змініть код програми. Перевірте, виконавши програму.
9. Збережіть зміни та закрийте IDLE.

15.13. Практикум. Завдання 15.13

У масиві $a[1]$, $a[2]$, $a[3]$, ..., $a[n]$ визначте кількість елементів, значення яких дорівнюють c . Метод `count()` використовувати забороняється. Збережіть програму у файлі «Завдання 15-13».

Технологія виконання

1. Створіть новий файл та збережіть його під ім'ям «Завдання 15-13».
2. Для розв'язування задачі використаємо тип даних список.

3. Сутність алгоритму розв'язування задачі полягає у послідовному порівнянні всіх елементів списку, починаючи з першого елемента, із заданим значенням. Якщо значення елемента масиву дорівнює заданому, то показник кількості (k) збільшується на одиницю.

4. Після створення виконайте програму для кількох наборів даних.

5. Перегляньте результати виконання та при потребі змініть код програми. Перевірте, виконавши програму.

6. У випадку виникнення труднощів можете скористатися [лістингом програми](#).

7. Збережіть зміни та закрийте IDLE.

15.14. Практикум. Завдання 15.14

У масиві $a[1], a[2], a[3], \dots, a[n]$ визначте значення максимального елемента та підрахуйте їх кількість. Функцію `max()` та метод `count()` використовувати забороняється. Збережіть програму у файлі «Завдання 15-14».

Технологія виконання

1. Створіть новий файл та збережіть його під ім'ям «Завдання 15-14».

2. Спочатку визначимо значення максимального елемента в масиві. Для цього введемо змінну (наприклад `MAX`) та присвоїмо їй значення, що дорівнює першому елементу. Потім, здійснюючи перебір елементів масиву, будемо їх порівнювати з `MAX`, та якщо воно виявиться більшим, то `MAX` присвоїмо значення цього елемента і т. д.

3. Для підрахунку кількості максимальних елементів скористайтесь досвідом, набутим під час виконання Завдання 15.13.

4. Після створення програми виконайте її для кількох наборів даних.

5. Перегляньте результати виконання та при потребі змініть код програми. Перевірте, виконавши програму.

6. У випадку виникнення труднощів можете скористатися [лістингом програми](#). Запропонований алгоритм не є оптимальним з точки зору часу виконання. Яким чином можна його оптимізувати.

7. Збережіть зміни та закрийте IDLE.

15.15. Практикум. Завдання 15.15

Створіть масив з n елементів ($A[1], A[2], A[3], \dots, A[n]$, n вводиться користувачем) випадкових цілих чисел, таких, що $-100 < A[i] < 100$ та виведіть його на екран. Збережіть програму у файлі «Завдання 15-15».

Технологія виконання

1. Створіть новий файл та збережіть його під ім'ям «Завдання 15-15».

2. Для розв'язування цієї задачі та наступних задач із використанням масивів будемо, як правило, застосовувати тип даних мови Python `список`.

3. Введення кількості елементів списку не являє труднощів.

4. Для отримання списку з випадковими числами можна скористатися функцією [sample](#) з модуля [random](#). Для цього модуль спочатку потрібно приєднати.

5. Тоді програмна реалізація може бути, наприклад, такою:

```

n=int(input('n=')) # Вводимо кількість елементів списку (масиву)
import random      # Під'єднуємо модуль "random"
A=random.sample(range(-99, 100), n) # Створюємо список
print(A)           # Друкуємо список
input()

```

6. Виконайте отриману програму для декількох наборів даних.
7. Збережіть зміни та закрийте IDLE.

15.16. Практикум. Завдання 15.16

Напишіть програму, під час виконання якої, буде створено та виведено на екран список цілих чисел $A[1], A[2], A[3], \dots, A[n]$ з випадковою кількістю елементів n ($2 \leq n \leq 20$). Всі числа повинні відповідати умові: $1000 < A[i] < 1000$. Кількість елементів списку n також повинна бути виведена на екран. Файл для збереження – «Завдання 15-16».

Технологія виконання

1. Створіть новий файл та збережіть його під ім'ям «Завдання 15-16».
2. Задача схожа на розглянуту в попередньому пункті.
3. Єдиною суттєвою відмінністю є випадкова кількість елементів списку. Згенерувати n можна за допомогою функції `randint` з модуля `random`.
4. Створіть програму та виконайте її для кількох наборів даних.
5. Після аналізу вихідних даних, при потребі, змініть код програми. Перевірте, виконавши програму.
6. У випадку виникнення труднощів можете скористатися [лістингом програми](#).
7. Збережіть зміни та закрийте IDLE.

15.17. Практикум. Завдання 15.17

Складіть програму під час виконання якої буде створено та виведено на екран список дійсних чисел $A[1], A[2], A[3], \dots, A[n]$ з випадковою кількістю елементів n ($1 < n < 11$). Всі числа повинні відповідати умові: $10 \leq A[i] \leq 10$. Кількість елементів списку n також повинна бути виведена на екран. Файл для збереження – «Завдання 15-17».

Технологія виконання

1. Створіть файл програми та збережіть його під ім'ям «Завдання 15-17».
2. Скориставшись досвідом, набутим під час виконання попередніх завдань, створіть програму.
3. Для заповнення масиву випадковими дійсними числами скористайтеся функцією `uniform` із модуля `random`.
4. Після створення програми виконайте її для кількох наборів даних.
5. Перегляньте результати виконання та при потребі змініть код програми.
6. У випадку виникнення труднощів є можливість скористатися [лістингом одного з варіантів програми](#).
7. Збережіть зміни та закрийте IDLE.

15.18. Практикум. Завдання 15.18

Створіть функції користувача для створення масиву випадкових цілих чисел $A[1], A[2], A[3], \dots, A[k]$ таких, що $d \leq A[i] \leq e$ та для знаходження середнього арифметичного елементів масиву. Згенеруйте три масиви з кількістю елементів n та знайдіть суму середніх арифметичних значень елементів кожного з них. Масиви також повинні бути виведені на екран. Файл для збереження – «Завдання 15-18».

Технологія виконання

1. Створіть файл програми та збережіть його під ім'ям «Завдання 15-18».
2. Скориставшись досвідом, набутим під час виконання попередніх завдань, спочатку створюємо функцію для генерування масиву:

```
def arr_int(k, d, e):    # Функція для генерування масиву
    import random
    A=random.sample(range(d, e+1), k)
    return A
```

3. Потім функцію для знаходження середнього арифметичного елементів масиву:

```
def sa(B):    # Функція для знаходження сер. арифм. ел. масиву
    s=0
    for i in range(0, len(B)):
        s=s+B[i]
    p=s/len(B)
    return p
```

4. Подальше створення програми не повинно викликати труднощів.
 5. Виконайте її для кількох наборів даних.
 6. Після аналізу вихідних даних, при потребі, зробіть корективи коду програми.
- Перевірте, виконавши програму.
7. У випадку виникнення труднощів можете скористатися [лістингом програми](#).
 8. Збережіть зміни та закрийте IDLE.

15.19. Практикум. Завдання 15.19

Створіть 4 масиви з n випадкових цілих чисел ($0 \leq A[i] \leq 100$), виведіть їх та значення максимального елементу в кожному з них на екран. Функцію `max()` використовувати забороняється. Файл для збереження – «Завдання 15-19».

Технологія виконання

1. Створіть файл програми та збережіть його під ім'ям «Завдання 15-19».
2. Задача доволі проста, за умови виконання попередніх завдань. Новизна полягає у створенні функції для визначення максимального елементу списку. Тому наведемо її тут:

```
def MAX(B):    # Функція для знаходження макс. ел. масиву
    max=B[0]
    for i in range(0, len(B)):
        if B[i]>max:
            max=B[i]
    return max
```

3. Після створення програми виконайте її для кількох наборів даних.
4. Перегляньте результати виконання та при потребі змініть код програми.

5. У випадку виникнення труднощів є можливість скористатися [лістингом програми](#).

6. Збережіть зміни та закрийте IDLE.

15.20. Практикум. Завдання 15.20

Створіть 3 масиви з n випадкових цілих чисел ($0 \leq A[i] \leq 50$), виведіть їх та індекси перших мінімальних елементів у кожному з них на екран. Функцію `min()` та метод `index()` використовувати забороняється. Файл для збереження – «Завдання 15-20».

Технологія виконання

1. Створіть файл програми та збережіть його під ім'ям «Завдання 15-20».
2. Задача, за умови виконання попередніх завдань, доволі проста. Новизна полягає у створенні функції для виведення індексів мінімальних елементів масиву. Одну з можливих реалізацій наведемо тут:

```
def ind_min(B): # Функція для виведення інд. мін. ел. масиву
    min=B[0]
    for i in range(1, len(B)):
        if B[i]<min:
            min=B[i]
    for i in range(1, len(B)):
        if B[i] == min:
            print(i)
```

3. Подальше створення програми не повинно викликати труднощів.
4. Виконайте її для кількох наборів даних.
5. Після аналізу вхідних та вихідних даних, при потребі, зробіть корективи коду програми. Перевірте, виконавши програму.
6. У випадку виникнення труднощів можете скористатися [лістингом програми](#).
7. Збережіть зміни та закрийте IDLE.

15.21. Практикум. Завдання 15.21

Дано 3 масиви (значення елементів користувач вводить з клавіатури під час виконання програми) кожний з яких містить m ($m < 10$) цілих чисел. Обчисліть кількість елементів кожного масиву значення яких дорівнює **a** ($-7 \leq a \leq 7$, значення **a** вводиться з клавіатури, при введенні числа, що не відповідає умові, з'являється повідомлення про помилку та надається нова можливість). Файл для збереження – «Завдання 15-21».

Технологія виконання

1. Розв'язання задачі, за умови виконання попередніх завдань, доволі просте.
2. Дізнатися загальну кількість елементів із зазначеним значенням дозволяє метод `count` (див. пункт [15.6. Пошук елемента у списку](#)).
3. У якості допомоги можете скористатися [лістингом програми](#).

15.22. Практикум. Завдання 15.22

У масиві `a[1]`, `a[2]`, `a[3]`, ..., `a[n]` визначте значення мінімального та максимального елементів та підрахуйте їх відповідні кількості. Збережіть програму у файлі «Завдання 15-22».

Технологія виконання

1. Розв'язання задачі є тривіальним, за умови виконання попередніх завдань,.
2. Дізнатися загальну кількість елементів із зазначеним значенням дозволяють методи та функції з пункту [15.6. Пошук елемента у списку](#).
3. Як допомогу можете використати [лістинг програми](#), який надає один з можливих варіантів розв'язку.

15.23. Практикум. Завдання 15.23

Створіть масив, що містить n ($n < 1000$) цілих чисел таких, що $a \leq A[i] \leq b$ ($-100 < a < b$, $b < 200$). Підрахуйте кількість від'ємних елементів та замініть їх числом 0. Файл для збереження – «Завдання 15-23».

Технологія виконання

1. Оскільки в умові задачі не зазначено спосіб введення масиву то згенеруємо його за допомогою `list comprehension` (генерування списку):

```
from random import randint
A = [randint(a, b) for i in range(n)]
```

2. Інші кроки розв'язання є тривіальними за умови наявності досвіду виконання попередніх завдань.
3. У якості допомоги можете скористатися [лістингом програми](#).

15.24. Практикум. Завдання 15.24

Дано масив з цілих чисел. Виведіть всі елементи масиву, які більші попереднього елемента. Файл для збереження – «Завдання 15-24».

Технологія виконання

1. Оскільки в умові задачі не зазначено спосіб введення масиву то введемо його з клавіатури за допомогою функції `map`:

```
A = list(map(int, input().split()))
```

2. Для виведення елементів масиву, що більше попереднього використаємо цикл `for` та умовний оператор `if`:

```
for i in range(1, len(A)) :
    if A[i-1] < A[i]:
        print(A[i], end=' ')
```

15.25. Практикум. Завдання 15.25

Дано масив з n цілих чисел. Виведіть пари сусідніх елементів одного знаку. Якщо сусідніх елементів одного знаку не має, то не виводьте нічого. Файл для збереження – «Завдання 15-25». [17, стор. 62, 19, №7844]

Технологія виконання

1. За умови виконання попередніх завдань, задача не повинна викликати труднощів.
2. У якості допомоги можете скористатися [лістингом програми](#).

15.26. Практикум. Завдання 15.26

Дано масив з n цілих чисел. Знайти суму двох найбільших елементів масиву. Файл для збереження – «Завдання 1-15-26». [17, стор. 64, 19, 7834]

Технологія виконання

1. За умови виконання попередніх завдань та використання методу `sort()`, задача не повинна викликати труднощів.
2. У якості допомоги можете скористатися [лістингом програми](#).

15.27. Практикум. Завдання для самостійного виконання

1. Учні вели спостереження за температурою повітря на протязі n днів та заносили дані у таблицю. Складіть програму для знаходження середньої температури повітря для цього проміжку часу. Збережіть її у файлі «Завдання 15-27-1-с».
2. Знайти суму елементів цілочисельного масиву (таблиці) кратних числу 10. Збережіть її у файлі «Завдання 15-27-2-с».
3. У масиві $a[1], a[2], a[3], \dots, a[n]$ визначте кількість елементів, значення яких менше числа M . Збережіть програму у файлі «Завдання 15-27-3-с».
4. Створіть список з 7 елементів $A[1], A[2], A[3], \dots, A[7]$, випадкових цілих чисел, таких, що $0 < A[i] < 1000$ та виведіть його на екран. Збережіть програму у файлі «Завдання 15-27-4-с».
5. Створіть програму, під час виконання якої, буде згенеровано та виведено на екран список цілих чисел $A[1], A[2], A[3], \dots, A[n]$ з випадковою кількістю елементів n ($1 < n < 20$). Всі числа повинні належати іншому списку $[7, 37, 25, 44, 15, 9, -7, -10, -12, 0]$. Кількість елементів списку n також повинна бути виведена на екран. Файл для збереження – «Завдання 15-27-5-с».
6. Створіть функції користувача для генерування масиву випадкових цілих чисел $A[1], A[2], A[3], \dots, A[k]$ таких, що $d \leq A[i] \leq e$ та для знаходження суми елементів масиву. Згенеруйте три масиви з кількістю елементів n та знайдіть середнє арифметичне сум елементів цих масивів. Масиви також повинні бути виведені на екран. Файл для збереження – «Завдання 15-27-6-с».
7. Згенеруйте 5 масивів кожний з яких містить n випадкових цілих чисел ($0 \leq A[i] \leq 1000$), виведіть їх на екран та визначте значення мінімального елементу в кожному з них. Функцію `min()` використовувати забороняється. Файл для збереження – «Завдання 15-27-7-с».
8. У масиві $a[1], a[2], a[3], \dots, a[n]$ визначте індекси та підрахуйте кількість максимальних елементів. Збережіть програму у файлі «Завдання 15-27-8-с».
9. Створіть масив, що містить n ($n < 1000$) цілих чисел таких, що $a \leq A[i] \leq b$ ($0 < a < b, b < 1000000$). Підрахуйте кількість елементів менших за 5000 та замініть їх числом 5000. Файл для збереження – «Завдання 15-27-9-с».
10. Дано масив з n цілих чисел. Знайти суму всіх елементів масиву які не дорівнюють максимальному. Файл для збереження – «Завдання 15-27-10-с».

[17, стор. 61, 19, №7831]

11. Дано масив з n цілих чисел. Виведіть всі його елементи з парними індексами. Файл для збереження – «Завдання 15-27-11-с». [17, стор. 61, 19, №7842]
12. Дано масив з n цілих чисел. Визначте, скільки в цьому масиві елементів, які більші двох своїх сусідів і виведіть кількість таких елементів. Крайні елементи списку не враховуються, оскільки у них мало сусідів. Файл для збереження – «Завдання 15-27-12-с». [17, стор. 62, 19, №7845]
13. Дано масив з n цілих чисел. Замінити всі найбільші елементи на найменший, а найменші на найбільший. Файл для збереження – «Завдання 15-27-13-с». [17, стор. 63, 19, №7849]
14. Дано масив з n цілих чисел. Знайти суму та кількість чисел, які більші за середнє арифметичне елементів масиву. Файл для збереження – «Завдання 15-27-14-с». [17, стор. 64, 19, №7833]
15. Дано масив з n цілих чисел. Знайти суму двох найменших елементів масиву. Файл для збереження – «Завдання 15-27-15-с».
16. Маємо N цілих чисел. Який найбільший добуток можна отримати, використавши тільки три з цих чисел?

Вхідні дані

В першому рядку ціле невід’ємне число N ($3 \leq N \leq 10^6$). У другому рядку N цілих чисел, кожне по модулю не перевищує 10^5 .

Вихідні дані

Значення найбільшого добутку трьох з них. [19, №7507]

Приклади

Вхідні дані	Вихідні дані
9 3 5 -9 7 4 0 9 -3 5	315

Файл для збереження програми – «Завдання 15-27-16-с».

Питання для самоконтролю

1. Що спільне та чим відрізняються кортежі та списки у мові Python?
2. Перерахуйте способи створення списків у Python?
3. Яким чином здійснюється звернення до елементів списку в мові програмування Python?
4. З якого числа починається нумерація елементів списку в мові Python?
5. Яка функція в мові Python дозволяє отримати кількість елементів списку?
6. Як можна змінити значення елемента списку?
7. Яким чином у мові Python реалізовані багатовимірні списки?
8. Які методи використовуються для додавання та вилучення елементів списку?
9. Що можна зробити за допомогою методу `index()`?
10. Який оператор дозволяє виконати перевірку на входження елемента до списку?
11. Яке призначення методу `reverse()`?
12. Що можна зробити за допомогою функції `shuffle()` з модуля `random`?
13. Назвіть функції модуля `random` за допомогою яких можна отримати елементи зі списку випадковим чином.

14. Який метод дозволяє відсортувати список?
15. Які функції та яким чином можна використовувати для заповнення списку числами?
16. Яка найважливіша відмінність кортежів від списків?
17. Які є способи створення кортежів?
18. З якого числа починається нумерація елементів кортежу в мові Python?
19. Які методи підтримують кортежі починаючи з Python 2.6? Яке їх призначення?

Гузь Валентина

16. Масиви у Python. Модуль array

Одновимірні масиви, що є аналогами відповідних структур даних у таких мовах програмування як, наприклад, C++ та Free Pascal у Python реалізовані за допомогою модуля array. Масиви дуже схожі на списки, але з обмеженням на тип даних і розмір кожного елемента.

16.1. Коли доцільно використовувати модуль array?

Модуль array може стати в нагоді, коли потрібно реалізувати списки даних з ефективним витрачанням пам'яті і відомо, що всі елементи списку належать одному типу. Наприклад, щоб зберегти список з 10 мільйонів цілих чисел, буде потрібно приблизно 160 Мбайт пам'яті, тоді як масив з 10 мільйонів цілих чисел займе всього 40 Мбайт. Незважаючи на таку економію пам'яті, жодна з основних операцій над масивами не виконується швидше, ніж аналогічні операції над списками, більш того, вони можуть виявитися навіть повільнішими.

При виконанні обчислень за участю масивів слід бути обережними з операціями, які створюють списки. Наприклад, коли генератори списків застосовуються до масивів, вони перетворюють їх у списки цілком, знищуючи всі переваги в економії пам'яті. У подібних ситуаціях створювати нові масиви краще за допомогою виразів-генераторів. Наприклад:

```
from array import *
a = array('i', [1,2,3,4,5])
b = array(a.typecode, (2*x for x in a))
# Створюється новий масив із a
print(type(b))
print(len(b))
for i in range(len(b)):
    print(b[i], end=" ")
```

Результат:

```
<class 'array.array'>
5
2 4 6 8 10
```

Оскільки вигаєш від використання масивів полягає в економії пам'яті, можливо, виявиться більш осмисленим виконати операції безпосередньо у вихідному масиві. У разі великих масивів операції всередині масиву виконуються приблизно на 15 відсотків швидше, ніж аналогічний програмний код, який створює новий масив за допомогою виразу-генератора.

16.2. Тип елементів. Створення масиву

Розмір і тип елемента у масиві визначається при його створенні та може набувати значень, поданих у таблиці 16.2.1.

Таблиця 16.2.1

Код типу	Тип у мові C	Тип у Python	Мінімальний розмір у байтах	Діапазон
'b'	signed char	int	1	-128 .. 127

'B'	unsigned char	int	1	0 .. 255
'h'	signed short	int	2	-32 768 .. 32 767
'H'	unsigned short	int	2	0 .. 65 535
'i'	signed int	int	2	-32 768 .. 32 767
'I'	unsigned int	int	2	0 .. 65 535
'l'	signed long	int	4	-2 147 483 648 .. 2 147 483 647
'L'	unsigned long	int	4	0 .. 4 294 967 295
'q'	signed long long	int	8	-9 223 372 036 854 775 808 .. 9 223 372 036 854 775 807
'Q'	unsigned long long	int	8	0 .. 18446744073709551615
'f'	float	float	4	$\pm 3.4 \cdot (10^{-38}) \dots \pm 3.4 \cdot (10^{+38})$
'd'	double	float	8	$\pm 1.7 \cdot (10^{-308}) \dots \pm 1.7 \cdot (10^{+308})$

Клас `array.array (typename[, initializer])` – новий масив, елементи якого обмежені `typename` (кодом типу), та ініціалізатор, який повинен бути списком, об'єктом, що підтримує інтерфейс буфера, або ітеріруємим об'єктом.

Повертає об'єкт, що являє собою масив елементів типу `typename`. При спробі додати до масиву елементи, тип яких не збігається з типом, що використовувався при створенні масиву, збуджується виключення `TypeError`.

`array.typecodes` – рядок, що містить всі можливі коди типів у масиві.

Наведемо кілька способів створення масивів.

- Масив створюється зі списку `[7, 4, 5, 8]`. Діапазон можливих значень його елементів: -128 .. 127.

```
from array import *
a = array('b', [7, 4, 5, 8])
```
- При створенні масиву значення його елементів вводяться з клавіатури. Діапазон можливих значень: -32 768 .. 32 767:

```
from array import *
a = array('i', list(map(int, input().split())))
```
- Згенеруємо масив цілих чисел з `n` елементів ($-10 \leq A[i] \leq 10$) та виведемо на екран значення його елементів.

```
import array as ar
from random import randint
n=int(input('Введіть кількість елементів масиву:'))
A=ar.array('l',[randint(-10, 10) for i in range(n)])
for i in range(n):
    print(A[i], end=' ')
```

Приклади можливих результатів виконання програми:

```
Введіть кількість елементів масиву:7
-6 -8 0 -9 5 3 7
Введіть кількість елементів масиву:15
-9 1 -5 7 -7 2 5 6 5 10 -10 -8 -9 0 8
```


Масиви відносяться до змінюваних типів даних. Як і всі послідовності, вони підтримують звернення до елемента за індексом, отримання зрізу, конкатенацію (оператор +), повторення (оператор *), перевірку на входження (оператор in).

16.3. Методи для роботи з масивами (array) у Python

`array.typecode` – TypeCode символ, використаний при створенні масиву.

`array.itemsize` – розмір у байтах одного елемента в масиві.

`array.append(x)` – додавання елемента у кінець масиву.

`array.buffer_info()` – кортеж (комірка пам'яті, довжина). Корисно для низькорівневих операцій.

`array.byteswap()` – змінити порядок проходження байтів у кожному елементі масиву. Корисно при читанні даних з файлу, написаного на машині з іншим порядком байтів.

`array.count(x)` – повертає кількість входжень `x` до масиву.

`array.extend(iterable)` – додавання елементів з [ітератора](#) (об'єкт, що підтримує ітерації) до масиву. Якщо ітератором є інший масив, він повинен мати точно такий же тип елементів; якщо ні, то збуджується виключення `TypeError`. Якщо об'єкт не є масивом, він повинен підтримувати ітерації, а його елементи які слід додати до масиву, повинні мати аналогічний тип.

`array.frombytes(s)` – створює масив `array` з масиву байт. Кількість байт повинна бути кратна розміру одного елемента в масиві (так, якщо б його було прочитано з файлу методом `fromfile()`).

`array.fromfile(f, n)` – читає `n` елементів із файлу `f` і додає їх у кінець масиву. Файл повинен бути відкритий на бінарне читання. Якщо він містить менше `n` елементів, генерується виключення `EOFError`, але елементи, які були доступні, додаються до масиву.

`array.fromlist(list)` – додавання елементів зі списку.

`array.index(x)` – номер першого входження `x` у масив.

`array.insert(i, x)` – вставити новий елемент зі значенням `x` до масиву перед номером `i`. Від'ємні значення розглядаються відносно кінця масиву.

`array.pop([i])` – видаляє `i`-й елемент з масиву та повертає його. За замовчуванням видаляється останній елемент.

`array.remove(x)` – видаляє перше входження `x` з масиву.

`array.reverse()` – змінює порядок елементів у масиві на зворотний.

`array.tobytes()` – перетворює масив у рядок двійкових даних так, якщо б запис виконувалася методом `tofile()`.

`array.tofile(f)` – записує всі елементи (у вигляді машинних значень (двійкових кодів)) до файлового об'єкта `f`.

`array.tolist()` – перетворює масив у список.

16.4. Практикум. Завдання 16.4

Дано масив з `n` цілих чисел ($-1000 \leq A[i] \leq 1000$). Замінити всі від'ємні значення елементів їх модулями та вивести на екран значення елементів зміненого масиву. У програмі передбачити введення всіх даних з клавіатури. Масив

створюється за допомогою модуля array. Файл для збереження програми – «Завдання 16-4».

Технологія виконання

1. За умови опрацювання пунктів 16.1-16.3 та виконання завдань до попередніх тем, задача не повинна викликати труднощів.

2. Наведемо один з можливих варіантів програми:

```
from array import *
n=int(input('Введіть кількість елементів масиву:'))
print('Введіть значення елементів масиву через пробіл:')
A=array('i', list(map(int, input().split())))
for i in range(n):
    if A[i]<0:
        A[i]=abs(A[i])
for i in range(n):
    print(A[i], end=' ')
```

3. Результат виконання для одного з наборів даних:

```
Введіть кількість елементів масиву: 7
Введіть значення елементів масиву через пробіл:
-7 0 -200 78 5 -5 6
7 0 200 78 5 5 6
```

16.5. Практикум. Завдання 16.5

Згенерувати масив з **n** довільних цілих чисел ($-100 \leq A[i] \leq 100$). Додати до масиву **k** елементів, вводячи їх з клавіатури. Вивести на екран значення елементів зміненого масиву. Масив створюється за допомогою модуля array. Файл для збереження програми – «Завдання 16-5».

Технологія виконання

1. За умови опрацювання пунктів 16.1-16.4 та виконання завдань до попередніх тем, задача не повинна викликати труднощів.

2. Наведемо один з можливих варіантів програми:

```
from array import *
from random import randint
n=int(input('Введіть кількість елементів масиву: '))
A=array('i', [randint(-100, 100) for i in range(n)])
k=int(input('Введіть кількість елементів, що потрібно додати: '))
for i in range(k):
    A.append(int(input('Введіть елемент масиву: ')))
for i in range(k+n):
    print(A[i], end=' ')
```

3. Наведемо результат виконання для одного з наборів даних:

```
Введіть кількість елементів масиву: 7
Введіть кількість елементів, що потрібно додати: 4
Введіть елемент масиву: -25
Введіть елемент масиву: 7
Введіть елемент масиву: 89
Введіть елемент масиву: 0
20 -81 -86 25 76 -47 -67 -25 7 89 0
```

16.6. Практикум. Завдання 16.6

Дано масив з n цілих чисел ($0 \leq A[i] \leq 1000$). Знайти суму елементів масиву, кратних заданому числу x . У програмі передбачити введення всіх даних з клавіатури. Задачу розв'язати за допомогою модуля `array`. Файл для збереження програми – «Завдання 16-6».

Технологія виконання

1. За умови виконання завдань до попередніх тем, задача не повинна викликати труднощів.

2. Наведемо один з можливих варіантів програми:

```
from array import *
n=int(input('Введіть кількість елементів масиву: '))
x=int(input('Введіть натуральне число: '))
print('Введіть значення елементів масиву через пробіл:')
# Введення масиву.
A = array('i', list(map(int, input().split())))
S=0
for i in range(n):
    if A[i]%x==0:
        S=S+A[i]
print('S=', S)
```

3. Результат виконання для одного з наборів даних:

```
Введіть кількість елементів масиву: 7
Введіть натуральне число: 9
Введіть значення елементів масиву через пробіл:
63 8 9 18 7 2 7
S= 90
```

16.7. Практикум. Завдання 16.7

Дано масив з n цілих чисел ($-1000000 \leq a[i] \leq 1000000$). Знайти пару сусідніх елементів сума яких дорівнює заданому числу. Вивести їх значення та індекси. У програмі передбачити введення всіх даних з клавіатури. Задачу розв'язати за допомогою модуля `array`. Файл для збереження програми – «Завдання 16-7».

Технологія виконання

1. За умови виконання завдань до попередніх тем та опрацювання пунктів 16.1-16.6 задача дуже проста.

2. У якості допомоги можете скористатися [лістингом програми](#).

3. Наведемо результат виконання для одного з наборів даних:

```
Введіть кількість елементів масиву: 5
Введіть ціле число: 10
Введіть значення елементів масиву через пробіл:
5 5 3 4 6
5 5
0 1
4 6
3 4
```

16.8. Практикум. Завдання для самостійного виконання

1. Дано масив з n цілих чисел ($-10000 \leq A[i] \leq 10000$). Замінити всі додатні значення елементів їх квадратами, а від'ємні модулями та вивести на екран

значення елементів зміненого масиву. У програмі передбачити введення всіх даних з клавіатури. Масив створюється за допомогою модуля `array`. Файл для збереження програми – «Завдання 16-8-1-с».

2. Дано масив з n цілих чисел ($-1000000 \leq a[i] \leq 1000000$). Знайти добуток елементів масиву (таблиці) кратних заданому числу. Збережіть її у файлі «Завдання 16-8-2-с».

Питання для самоконтролю

1. За допомогою якого модуля реалізовані одновимірні масиви, що є аналогами відповідних структур даних у таких мовах програмування як, наприклад, C++ та Free Pascal?
2. Коли та яким чином доцільно використовувати у Python модуль `array`?
3. Наведіть приклади створення масивів у програмі за допомогою модуля `array`?
4. Які вам відомі методи для роботи з масивами (`array`) у Python?

Гузь Валентина

17. Рядки

Рядки є впорядкованими послідовностями символів. Довжина рядка обмежена лише об'ємом оперативної пам'яті комп'ютера. Як і всі послідовності, рядки підтримують звернення до елементу за індексом, отримання зрізу, конкатенацію (оператор +), повторення (оператор *), перевірку на входження (оператор in).

Крім того, рядки належать до незмінних типів даних. Тому практично всі рядкові методи в якості значення повертають новий рядок. При використанні невеликих рядків це не призводить до будь-яких проблем, але при роботі з великими рядками можна зіткнутися з проблемою нестачі пам'яті.

17.1. Створення рядків

Будь-яка послідовність символів, взята в лапки, в Python вважається рядком; при цьому рядки можуть бути взяті як в одиночні, так і в подвійні лапки:

```
>>> r1 = 'Це рядок'
>>> r2 = "Це теж рядок"
>>> type(r1)
<class 'str'>
>>> type(r2)
<class 'str'>
```

Це правило дозволяє використовувати внутрішні лапки та апострофи у рядках.

Створити рядок можна також вказавши набір символів між потроєнними апострофами або потроєнними лапками. Такі об'єкти можна розмістити на декількох рядках, а також одночасно використовувати лапки і апострофи. В інших випадках такі об'єкти еквівалентні рядкам в апострофах і лапках. Всі спеціальні символи в таких рядках інтерпретуються. Приклади:

```
>>> print('''Рядок 1
Рядок 2''')
Рядок 1
Рядок 2
>>> print("""Рядок 1
Рядок 2""")
Рядок 1
Рядок 2
```

Створити рядок можна також за допомогою функції `str([<Об'єкт>[, <Кодування>[, <Обробка помилок>]])`. Якщо вказано тільки перший параметр, то функція повертає рядкове представлення будь-якого об'єкта. Якщо параметри не зазначені взагалі, то повертається порожній рядок.

```
>>> s = 55.5
>>> type(s)
<class 'float'>
>>> v = str(s)
>>> v
'55.5'
>>> type(v)
<class 'str'>
```

17.2. Зміна регістру символів у рядках

Одна з найпростіших операцій, що виконуються з рядками, дає змогу змінювати регістр символів. Погляньте на наступний фрагмент коду та спробуйте визначити, що у ньому відбувається:

```
>>> name = "ada lovelace"
>>> print(name.title())
Ada Lovelace
```

У цьому прикладі у змінній `name` зберігається рядок, що складається з букв нижнього регістру `"ada lovelace"`. За ім'ям змінної в команді `print()` йде виклик методу `title()`. Метод являє собою дію, яку Python виконує з даними. Точка `(.)` Після `name` в конструкції `name.title()` наказує Python застосувати метод `title()` до змінної `name`. За ім'ям методу завжди повинна стояти пара круглих дужок, тому, що методам для виконання їх роботи часто потрібна додаткова інформація. Ця інформація вказується в дужках. Функції `title()` додаткова інформація не потрібна, тому в круглих дужках нічого немає.

Метод `title()` перетворює перший символ кожного слова у рядку до верхнього регістру, тоді як всі інші символи виводяться у нижньому регістрі. Наприклад, дана можливість може бути корисна, якщо у вашій програмі вхідні значення `Ada`, `ADA` та `ada` повинні розглядатися як одне й те ж саме ім'я, та всі вони повинні відображатися у вигляді `Ada`.

Для роботи з регістром також існують інші корисні методи. Наприклад, всі символи рядка можна перетворити до верхнього або нижнього регістру:

```
>>> name = "Ada Lovelace"
>>> print(name.upper())
ADA LOVELACE
>>> print(name.lower())
ada lovelace
```

Метод `lower()` особливо корисний для зберігання даних. Нерідко програміст не може розраховувати на те, що користувачі введуть всі дані з точним дотриманням регістру, тому рядки перед збереженням перетворюються до нижнього регістру. Потім, коли буде потрібно вивести інформацію, використовується регістр, найбільш підходящий для кожного рядка.

17.3. Конкатенація

Часто виникає необхідність у об'єднанні рядків. Уявіть, що ім'я та прізвище зберігаються у різних змінних та ви хочете об'єднати їх для виведення повного імені:

```
>>> first_name = "Ada"
>>> last_name = "Lovelace"
>>> full_name = first_name + " " + last_name
>>> print(full_name)
Ada Lovelace
```

Для об'єднання рядків у Python використовується знак «плюс» `(+)`. У наведеному прикладі повне ім'я будується об'єднанням `first_name`, пробіла та `last_name`.

Такий спосіб об'єднання рядків називається конкатенацією. Ви можете використовувати конкатенацію для побудови складних повідомлень з інформацією, що зберігається у змінних. Розглянемо приклад:

```
>>> first_name = "ada"
>>> last_name = "lovelace"
>>> full_name = first_name + " " + last_name
>>> print("Hello, " + full_name.title() + "!")
Hello, Ada Lovelace!
```

Повне ім'я використовується для виведення привітання, а метод `title()` забезпечує правильне форматування імені.

```
>>> first_name = "ada"
>>> last_name = "lovelace"
>>> full_name = first_name + " " + last_name
>>> message = "Hello, " + full_name.title() + "!"
>>> print(message)
Hello, Ada Lovelace!
```

Цей код також виводить повідомлення "Hello, Ada Lovelace!", Але збереження тексту повідомлення у змінній істотно спрощує завершальну команду виведення.

17.4. Табуляції та розриви рядків

У програмуванні терміном «пропуск» (whitespace) називаються такі недруковані символи, як пробіли, табуляції та символи кінця рядка. Пропуски структурують текст, щоб користувачеві було зручніше читати його.

Для включення у текст позиції табуляції використовується комбінація символів `\t`.

```
>>> print("Python")
Python
>>> print("\tPython")
    Python
```

Розриви рядків додаються за допомогою комбінації символів `\n`:

```
>>> print("Мови програмування:\nPython\nC++\nJavaScript")
Мови програмування:
Python
C++
JavaScript
```

Табуляції та розриви рядків можуть поєднуватися у тексті. Скажімо, послідовність `"\n\t"` наказує Python почати текст з нового рядка, на початку якого розташовується табуляція.

Наступний приклад демонструє виведення одного повідомлення з розбивкою на чотири рядки:

```
>>> print("Мови програмування:\n\tPython\n\tC++\n\tJavaScript")
Мови програмування:
    Python
    C++
    JavaScript
```


17.5. Видалення пропусків

Зайві пропуски можуть заважати при обробці рядків. Для програміста рядки 'python' та 'python ' зовні не відрізняються, але для певної програми, що їх обробляє це абсолютно різні рядки. Python бачить зайвий пробіл у 'python ' та вважає, що він дійсно важливий – до тих пір, доки ви не повідомите про протилежне.

Потрібно звертати увагу на пропуски при роботі з рядковим типом даних, тому, що у програмах часто доводиться порівнювати рядки, щоб перевірити на збіг їх вміст. Типовий приклад – перевірка імен користувачів при вході на сайт. Зайві пропуски можуть створювати плутанину й у більш простих ситуаціях. На щастя, Python дозволяє легко видалити зайві пропуски з даних, введених користувачем.

Python може шукати зайві пропуски з лівого та правого краю рядка. Щоб переконатися у тому, що біля правого краю (у кінці) рядка немає пропусків, викличте метод `rstrip()`.

```
>>> favorite_language = 'python '  
>>> favorite_language  
'python '  
>>> favorite_language.rstrip()  
'python'  
>>> favorite_language  
'python '
```

Значення, що зберігається у змінній `favorite_language`, містить зайві пробіли у кінці рядка. Коли ви наказуєте Python вивести це значення у термінальному сеансі, ви бачите пробіл у кінці значення. Коли метод `rstrip()` працює зі змінною `favorite_language`, цей зайвий пробіл видаляється. Втім, видалення лише тимчасове: якщо знову запросити значення `favorite_language`, ми бачимо, що рядок не відрізняється від вихідного, включаючи зайвий пробіл.

Щоб назавжди виключити пропуск з рядка, слід записати усічене значення назад у змінну:

```
>>> favorite_language = 'python '  
>>> favorite_language = favorite_language.rstrip()  
>>> favorite_language  
'python'
```

У цьому випадку спочатку пропуски видаляються у кінці рядка, а потім значення записується у вихідну змінну. Операція зміни значення змінної з наступним його збереженням у вихідній змінній часто виконується у програмуванні. Так значення змінної може змінюватися у ході виконання програми або у відповідь на дії користувача.

Пропуски також можна видалити з лівого краю (на початку) рядка за допомогою методу `lstrip()`, а метод `strip()` видаляє пропуски з обох боків:

```
>>> favorite_language = ' python '
>>> favorite_language.rstrip()
' python'
>>> favorite_language.lstrip()
'python '
>>> favorite_language.strip()
'python'
```

У цьому прикладі початкове значення містить пробіли на початку і в кінці. Потім пробіли видаляються з правого краю, з лівого краю і з обох кінців рядка. Експериментуйте з функціями видалення пропусків, це допоможе вам освоїтися з роботою з рядками. На практиці ці функції найчастіше застосовуються для «очищення» користувацького введення перед його збереженням в програмі.

17.6. Операції над рядками

Рядки належать до послідовностей. Як і всі послідовності, рядки підтримують крім конкатенації, яку ми розглянули, ще звернення до елементу за індексом, отримання зрізу, повторення та перевірку на входження.

До будь-якого символу рядка можна звернутися як до елементу списку. Досить вказати його індекс в квадратних дужках. Нумерація починається з нуля:

```
>>> r = "Python"
>>> r[0]
'P'
>>> r[3]
'h'
```

Якщо символ, що відповідає вказаному індексу, відсутній у рядку, то збуджується виключення `IndexError`:

```
>>> r[7]
Traceback (most recent call last):
  File "<pyshell#3>", line 1, in <module>
    r[7]
IndexError: string index out of range
```

В якості індексу можна вказати від'ємне значення. В цьому випадку зміщення буде відраховуватися від кінця рядка, а точніше, значення віднімається від довжини рядка, щоб отримати позитивний індекс:

```
>>> r = "Python"
>>> len(r)
6
>>> r[-1]
'n'
>>> r[len(r)-1]
'n'
```

Оскільки рядки належать до незмінних типів даних, то змінити символ за індексом не можна. Щоб виконати зміну, можна скористатися операцією витягання зрізу, яка повертає зазначений фрагмент рядка. Формат операції:

```
[<Початок>: <Кінець>: <Крок>]
```

Всі параметри є необов'язковими. Якщо параметр `<Початок>` не вказано, то використовується значення 0. Якщо параметр `<Кінець>` не вказано, то

повертається фрагмент до кінця рядка. Слід також зауважити, що символ з індексом, зазначеним у цьому параметрі, не входить до фрагменту, що повертається. Якщо параметр <Крок> не вказано, то використовується значення 1. В якості значення параметрів можна вказати від'ємні значення.

Тепер розглянемо кілька прикладів. Спочатку отримаємо копію рядка:

```
>>> r = "Python"
>>> r[:] # Повертається фрагмент від позиції 0 до кінця рядка
'Python'
```

А тепер виведемо символи у зворотному порядку:

```
>>> r[::-1] # Вказуємо від'ємне значення у параметрі <Крок>
'nohtyP'
```

Замінімо перший символ у рядку:

```
>>> r = 'Python'
>>> "M" + r[1:] # Відтинаємо фрагмент від символу 1 до кінця рядка
'Mython'
```

Видалимо останній символ:

```
>>> r[:-1] # Повертається фрагмент від 0 до len(r)-1
'Pytho'
```

Отримаємо перший символ у рядку:

```
>>> r[0:1]
'P'
```

А тепер отримаємо останній символ:

```
>>> r[-1:] # Отримуємо фрагмент від len(s)-1 до кінця рядка
'n'
```

І, нарешті, виведемо символи з індексами 3 і 4:

```
>>> r[3:5] # Повертаються символи з індексами 3 і 4
'ho'
```

Дізнатися кількість символів у рядку дозволяє функція len():

```
>>> len('Валерій')
7
```

Знаючи кількість символів, можна перебрати всі символи за допомогою циклу for:

```
>>> s = "Python"
>>> for i in range(len(s)): print(s[i], end=" ")
```

```
P y t h o n
```

Оскільки рядки підтримують ітерації, ми можемо просто вказати рядок як параметр циклу:

```
>>> for i in s: print(i, end=" ")
```

```
P y t h o n
```

Як бачимо, результат отримали аналогічний.

Рядки підтримують операцію повторення і перевірку на входження. Повторити рядок вказану кількість разів можна за допомогою оператора *, а виконати перевірку на входження фрагмента у рядок дозволяє оператор in:

```
>>> "-" * 17
'-----'
>>> "yt" in "Python"
True
>>> "yt" in "Perl"
False
```

17.7. Методи для роботи з рядками

Використання деяких методів (`strip`, `lstrip`, `rstrip`) для роботи з рядками ми вже розглядали вище. Розберемо ще декілька:

- `split([<Роздільник>[, <Ліміт>]])` – розділяє рядок на підрядки за вказаним роздільником і додає їх до списку. Якщо перший параметр не зазначено або має значення `None`, то як роздільник використовується символ пробілу. Якщо в другому параметрі задано число, то у списку буде вказана кількість підрядків. Якщо підрядків більше зазначеної кількості, то список буде містити ще один елемент, у якому буде залишок рядка. Приклади:

```
>>> s = "word1 word2 word3"
>>> s.split()
['word1', 'word2', 'word3']
>>> s.split(None, 1)
['word1', 'word2 word3']
>>> s = "word1\nword2\nword3"
>>> s.split("\n")
['word1', 'word2', 'word3']
```

Якщо у рядку містяться кілька пробілів поспіль і роздільник не вказано, то порожні елементи не будуть додані у список:

```
>>> s = "word1      word2 word3      "
>>> s.split()
['word1', 'word2', 'word3']
```

Якщо роздільник не знайдений в рядку, то список буде складатися з одного елемента, що представляє вихідний рядок:

```
>>> "word1 word2 word3".split("\n")
['word1 word2 word3']
```

- `rsplit([<Роздільник>[, <Ліміт>]])` – метод аналогічний методу `split()`, але пошук символу-роздільника проводиться не зліва направо, а навпаки, справа наліво. Приклади:

```
>>> s = "word1 word2 word3"
>>> s.rsplit()
['word1', 'word2', 'word3']
>>> s.rsplit(None, 1)
['word1 word2', 'word3']
>>> "word1\nword2\nword3".rsplit("\n")
['word1', 'word2', 'word3']
```

- `splitlines([True])` – розділяє рядок на підрядки по символу переводу рядка (`\n`) і додає їх до списку. Символи нового рядка включаються в результат, тільки якщо необов'язковий параметр має

значення True. Якщо роздільник не знайдений в рядку, то список буде містити тільки один елемент. Приклади:

```
>>> "word1\nword2\nword3".splitlines()
['word1', 'word2', 'word3']
>>> "word1\nword2\nword3".splitlines(True)
['word1\n', 'word2\n', 'word3']
>>> "word1\nword2\nword3".splitlines(False)
['word1', 'word2', 'word3']
>>> "word1 word2 word3".splitlines()
['word1 word2 word3']
```

- `partition(<Роздільник>)` – знаходить перше входження символу-роздільника у рядок і повертає кортеж з трьох елементів. Перший елемент буде містити фрагмент, розташований перед роздільником, другий елемент – символ-роздільник, а третій елемент – фрагмент, розташований після символу-роздільника. Пошук проводиться зліва направо. Якщо символ-роздільник не знайдений, то перший елемент кортежу буде містити весь рядок, а інші елементи будуть порожніми.

Приклад:

```
>>> "word1 word2 word3".partition(" ")
('word1', ' ', 'word2 word3')
>>> "word1 word2 word3".partition("\n")
('word1 word2 word3', '', '')
```

- `rpartition(<Роздільник>)` – метод аналогічний методу `partition()`, але пошук символу-роздільника проводиться не зліва направо, а, навпаки, справа наліво. Якщо символ-роздільник не знайдений, то перші два елементи кортежу будуть порожніми, а третій елемент буде містити весь рядок. Приклад:

```
>>> "word1 word2 word3".rpartition(" ")
('word1 word2', ' ', 'word3')
>>> "word1 word2 word3".rpartition("\n")
('', '', 'word1 word2 word3')
```

- `join()` – перетворює послідовність у рядок. Елементи додаються через вказаний роздільник. Формат методу:

`<Рядок>=<Роздільник>.join(<Послідовність>)`

Як приклад перетворимо список і кортеж в рядок:

```
>>> " => ".join(["word1", "word2", "word3"])
'word1 => word2 => word3'
>>> " ".join(("word1", "word2", "word3"))
'word1 word2 word3'
```

Зверніть увагу на те, що елементи послідовностей повинні бути рядками, інакше збуджується виключення `TypeError`:

```
>>> " ".join(("word1", "word2", 5))
Traceback (most recent call last):
  File "<pyshell#6>", line 1, in <module>
    " ".join(("word1", "word2", 5))
TypeError: sequence item 2: expected str instance, int found
```

- Як ви вже знаєте, рядки належать до незмінних типів даних. Якщо спробувати змінити символ за індексом, то виникне помилка. Щоб змінити символ за індексом можна перетворити рядок в список за допомогою функції `list()`, провести зміни, а потім за допомогою методу `join()` перетворити список назад в рядок. Приклад:

```
>>> s = "Python"
>>> arr = list(s) # Перетворюємо рядок у список
>>> arr
['P', 'y', 't', 'h', 'o', 'n']
>>> arr[0] = "J" # Змінюємо елемент за індексом
>>> arr
['J', 'y', 't', 'h', 'o', 'n']
>>> s = "".join(arr) # Перетворюємо список у рядок
>>> s
'Jython'
```

- `isdigit()` – перевіряє чи містить рядок тільки цифри. Приклад:

```
>>> a=('7391')
>>> a.isdigit()
True
>>> b=('n7391')
>>> b.isdigit()
False
>>> c=('nsrob')
>>> c.isdigit()
False
```

- `isupper()` – перевіряє чи містить рядок тільки символи верхнього регістру.

```
>>> a='sbhrk'
>>> b='DRLNA'
>>> c='cvGHw'
>>> b.isupper()
True
>>> c.isupper()
False
>>> a.isupper()
False
```

- `islower()` – перевіряє чи містить рядок тільки символи нижнього регістру.

```
>>> a='sbhrk'
>>> b='DRLNA'
>>> c='cvGHw'
>>> a.islower()
True
>>> b.islower()
False
>>> c.islower()
False
```

17.8. Функції для роботи з символами

Для роботи з окремими символами призначені наступні функції:

- `chr(<Код символу>)` – повертає символ за вказаним кодом:

```
>>> print(chr(1077))  
e
```
- `ord(<Символ>)` – повертає код зазначеного символу:

```
>>> print(ord("e"))  
1077
```

17.9. Пошук та заміна у рядку

Для пошуку та заміни в рядку використовуються такі методи:

- `find()` – шукає підрядок у рядку. Повертає номер позиції, з якої починається входження підрядка в рядок. Якщо підрядок в рядок не входить, то повертається значення `-1`. Метод залежить від регістру символів. Формат методу:

`<Рядок>.find (<Підрядок>[, <Початок>[, <Кінець>]])`

Якщо початкова позиція не вказана, то пошук буде здійснюватися з початку рядка. Якщо параметри `<Початок>` і `<Кінець>` вказані, то проводиться операція витягання зрізу

`<Рядок>[<Початок>: <Кінець>]`

і пошук підрядка буде виконуватися у цьому фрагменті. Приклад:

```
>>> s = "приклад приклад Приклад"  
>>> s.find("при")  
0  
>>> s.find("При")  
16  
>>> s.find("тест")  
-1  
>>> s.find("при", 9)  
-1  
>>> s.find("при", 0, 6)  
0  
>>> s.find("при", 7, 12)  
8
```

- `index()` – метод аналогічний методу `find()`, але якщо підрядок до рядка не входить, то збуджується виключення `ValueError`. Формат методу:

`<Рядок>.index(<Підрядок>[, <Початок>[, <Кінець>]])`

Приклад:


```

>>> s = "приклад приклад Приклад"
>>> s.index("при")
0
>>> s.index("при", 7, 12)
8
>>> s.index("При", 1)
16
>>> s.index("тест")
Traceback (most recent call last):
  File "<pyshell#4>", line 1, in <module>
    s.index("тест")
ValueError: substring not found

```

- `rfind()` – шукає підрядок в рядку. Повертає позицію останнього входження підрядка в рядок. Якщо підрядок у рядок не входить, то повертається значення -1. Метод залежить від регістру символів. Формат методу:

```
<Рядок>.rfind(<Підрядок>[, <Початок>[, <Кінець>]])
```

Якщо початкова позиція не вказана, то пошук буде проводитися з початку рядка. Якщо параметри `<Початок>` і `<Кінець>` вказані, то виконується операція витягання зрізу, і пошук підрядка буде проводитися в цьому фрагменті. Приклад:

```

>>> s = "приклад приклад Приклад Приклад"
>>> s.rfind("при")
8
>>> s.rfind("При")
24
>>> s.rfind("тест")
-1
>>> s.find("при", 0, 6)
0
>>> s.find("При", 10, 20)
16

```

- `rindex()` – метод аналогічний методу `rfind()`, але якщо підрядок в рядок не входить, то збуджується виключення `ValueError`. Формат методу:

```
<Рядок>.rindex(<Підрядок>[, <Початок>[, <Кінець>]])
```

Приклад:

```

>>> s = "приклад приклад Приклад Приклад"
>>> s.rindex("при")
8
>>> s.rindex("при", 0, 6)
0
>>> s.rindex("тест")
Traceback (most recent call last):
  File "<pyshell#9>", line 1, in <module>
    s.rindex("тест")
ValueError: substring not found

```

- `count()` – повертає число входжень підрядка в рядок. Якщо підрядок в рядок не входить, то повертається значення 0. Метод залежить від регістру символів. Формат методу:

`<Рядок>.count(<Підрядок>[, <Початок>[, <Кінець>]])`

Приклад:

```
>>> s = "приклад приклад Приклад Приклад"
>>> s.count("при")
2
>>> s.count("при", 6)
1
>>> s.count("При")
2
>>> s.count("тест")
0
```

- `startswith()` – перевіряє, чи починається рядок з вказаною підрядка. Якщо починається, то повертається значення `True`, в іншому випадку – `False`. Метод залежить від регістру символів. Формат методу:

`<Рядок>.startswith(<Підрядок>[, <Початок>[, <Кінець>]])`

Якщо початкова позиція не вказана, порівняння буде проводитися з початком рядка. Якщо параметри `<Початок>` і `<Кінець>` вказані, то виконується операція витягання зрізу, і порівняння буде проводитися з початком фрагмента. Приклад:

```
>>> s = "приклад приклад Приклад Приклад"
>>> s.startswith("при")
True
>>> s.startswith("При")
False
>>> s.startswith("при", 6)
False
>>> s.startswith("При", 16)
True
```

- `endswith()` – перевіряє, чи закінчується рядок зазначеним підрядком. Якщо закінчується, то повертається значення `True`, в іншому випадку – `False`. Метод залежить від регістру символів.

Формат методу:

`<Рядок>.endswith(<Підрядок>[, <Початок>[, <Кінець>]])`

Якщо початкова позиція не вказана, то порівняння буде проводитися з кінцем рядка. Якщо параметри `<Початок>` і `<Кінець>` вказані, то виконується операція витягання зрізу, і порівняння буде проводитися з кінцем фрагмента. Приклад:

```
>>> p = "підрядок ПІДРЯДОК"
>>> p.endswith("док")
False
>>> p.endswith("ДОК")
True
>>> p.endswith("док", 0, 8)
True
```

- `replace()` – робить заміну всіх входжень підрядка в рядок на інший підрядок і повертає результат у вигляді нового рядка. Метод залежить від регістру символів. Формат методу:
`<Рядок>.replace(<Підрядок для заміни>, <Новий підрядок>[, <Максимальна кількість замін>])`

Приклад:

```
>>> s = "Привіт, Петро!"
>>> r = s.replace("Петро", "Іван")
>>> print(r)
Привіт, Іван!
```

17.10. Практикум. Завдання 17.10

Створіть програму, яка запитує ім'я користувача, а потім його вітає, виводячи слово «Привіт», ім'я користувача та розділові знаки у наступному вигляді: «Привіт, Валерій!». Файл для збереження програми – «Завдання 17-10».

Технологія виконання

1. Задача елементарна. Виводимо запрошення користувачу ввести власне ім'я, а потім, використавши операцію конкатенації, виводимо напис, як це вимагається в умові задачі.

2. Наведемо один з можливих варіантів програми:

```
a=input('Як Вас звати? ')
print('Привіт, '+a+'!')
```

3. Або так:

```
print('Привіт, '+input('Як Вас звати? ')+ '!')
```

4. Результат виконання:

```
Як Вас звати? Валерій
Привіт, Валерій!
```

17.11. Практикум. Завдання 17.11

Створіть програму для знаходження кількості одиниць у двійковому запису заданого числа. Файл для збереження програми – «Завдання 17-11».

Технологія виконання

1. Для представлення цілих чисел у системах числення з основами 2, 8 та 16 у Python використовуються вбудовані функції `bin`, `oct` та `hex`, відповідно. Отже, щоб записати задане число у двійковій формі можна використати функцію `bin`, а метод `count`, поверне кількість символів «1».

2. Отримуємо один з варіантів програми:

```
a=int(input('Введіть натуральне число: '))
b=bin(a)
c=b.count('1')
print(b)
print(c)
```

3. Результат виконання:

```
Введіть натуральне число: 25
0b11001
3
```

4. Або такий варіант:

```
print(bin(int(input('Введіть натуральне число: '))).count('1'))
```

5. Результат виконання:
Введіть натуральне число: 25
3

17.12. Практикум. Завдання 17.12

Непорожній рядок, який містить деяке слово, називається паліндромом, якщо це слово однаково читається як зліва направо, так і з права наліво (наприклад: зараз, корок, ротатор, комок, біб, піп, дід, шалаш, кок, тут, око вимив, вилив, вишив, вишив).

Створіть програму яка аналізує введене слово, що складається з символів одного регістру, та виводить повідомлення про те. чи є дане слово паліндромом. Файл для збереження програми – «Завдання 17-12».

Технологія виконання

1. Щоб з'ясувати чи є введене слово паліндромом його потрібно «перевернути» та порівняти з початковим. Для перевертання слова використаємо [операцію витягання зрізу](#), яка повертає зазначений фрагмент рядка.

2. Отримаємо програму:
- ```
s1=input('Введіть слово: ')
ls=s1[::-1]
if s1==ls:
 print('Це паліндром.')
else:
 print('Це не паліндром.')
```

3. Наведемо результати виконання для різних наборів даних:
- ```
Введіть слово: око
Це паліндром.
>>>
Введіть слово: весна
Це не паліндром.
```

17.13. Практикум. Завдання 17.13

Створіть програму для знаходження кількості певних букв (вводить користувач) у даному тексті. Файл для збереження програми – «Завдання 17-13».

Технологія виконання

1. Ідея розв'язання:
- ⇒ вводимо букву або декілька (що будемо підраховувати);
 - ⇒ вводимо текст;
 - ⇒ за допомогою циклу for будемо перебирати букви у введеному тексті і якщо відповідна літера належить до тих, що потрібно підрахувати то лічильник збільшуватимемо на 1;
 - ⇒ виводимо кількість букв.
2. Програмна реалізація:

```

bukvi=input('Введіть букву або кілька: ')
text=input('Введіть текст: ')
k=0
for bukva in text:
    if bukva in bukvi:
        k=k+1
print(k)

```

3. Наведемо результати виконання для різних наборів даних:

```

Введіть букву або кілька: а
Введіть текст: Слава Україні!
3

```

```

Введіть букву або кілька: сво
Введіть текст: Душу й тіло ми положим за нашу свободу
7

```

4. Зауважимо, що програму можна використовувати для підрахунку й інших символів, наприклад, пробілів, крапок або цифр.

17.14. Практикум. Завдання 17.14

Створити програму для з'ясування того, чи можна скласти слово з літер іншого слова. Файл для збереження програми – «Завдання 17-14».

Технологія виконання

1. Ідея розв'язання полягає у наступному:

⇒ вводимо початкове слово;
 ⇒ вводимо слово для оцінювання;
 ⇒ будемо переглядати літери слова по черзі і якщо не знаходимо певної букви у початковому слові, то це свідчитиме про те, що слово скласти неможливо.

2. Наведемо один з варіантів програми:

```

cl_poch=input('Введіть перше слово: ')
cl=input('Введіть друге слово: ')
pok=True
for bukva in cl:
    if bukva not in cl_poch:
        pok=False
if pok:
    print('Слово скласти можна.')
else:
    print('Слово скласти не можна.')

```

3. Результати виконання для різних наборів даних:

```

Введіть перше слово: інформатика
Введіть друге слово: форма
Слово скласти можна.
>>>
Введіть перше слово: інформатика
Введіть друге слово: мова
Слово скласти не можна.

```

17.15. Практикум. Завдання 17.15

У студента Васі є молодший брат Петя, який пішов до першого класу і почав вивчати арифметику. Додому у першому класі задали розв'язати багато прикладів на додавання та віднімання. Петя попросив Васю перевірити домашнє завдання.

Побачивши дві сторінки написаних каракулями прикладів, Вася прийшов у жах від об'єму роботи і вирішив навчити Петю використовувати для самоперевірки комп'ютер. Для цього Васі потрібно написати програму, яка обчислювала б розв'язки потрібних арифметичних прикладів.

Вхідні дані

Один рядок, у якому можуть зустрічатись цифри та символи «+» і «-». Довжина рядка не перевищує 10000 символів, значення всіх чисел у ньому не перевищують 10000.

Вихідні дані

Вивести одне ціле число – результат обчислень.

Файл для збереження програми – «Завдання 17-15». [19]

Технологія виконання

1. Задача має дуже простий розв'язок, якщо використати функцію [eval](#) (див. додаток 1).

2. Лістинг програми:

```
a=input('Введіть вираз: ')
print(eval(a))
```

3. Результат виконання для одного з наборів даних:

```
Введіть вираз: 7+33-15
25
```

4. За допомогою цієї програми можна обчислювати значення виразів, що містять й інші дії. Наприклад:

```
Введіть вираз: (6+20)*5+2**3
138
```

17.16. Практикум. Завдання 17.16

Створити програму для підрахунку цифр, великих та малих букв у введеному рядку. Файл для збереження програми – «Завдання 17-16».

Технологія виконання

1. Задача розв'язується доволі просто за допомогою методів `isdigit()`, `isupper()`, та `islower()`.

2. Наведемо один з варіантів розв'язання:

```
a=input("Введіть текст: ")
v=0
m=0
c=0
for b in a:
    if b.isupper():
        v=v+1
    if b.islower():
        m=m+1
    if b.isdigit():
        c=c+1
print('Великих літер: ', v)
print('Малих літер: ', m)
print('Цифр: ', c)
```

3. Результат виконання:

Введіть текст: Завдання 1-17-16
Великих літер: 1
Малих літер: 7
Цифр: 5

17.17. Практикум. Завдання для самостійного виконання

1. Створіть програму для знаходження кількості одиниць у двійковому запису заданого числа. Файл для збереження програми – «Завдання 17-17-1-с».
2. Порахувати кількість букв «а» у введеному тексті. Файл для збереження програми – «Завдання 17-17-2-с».
3. Непорожній рядок, який містить деяке слово (або декілька), називається паліндромом, якщо це слово однаково читається як зліва направо, так і з права наліво (наприклад: зараз, корок, ротатор, комок, біб, піп, дід шалаш кок, тут, око вимив, вилив, вишив, вирив).
Створіть програму яка аналізує введений рядок (врахувати, що букви можуть мати різні регістри), та виводить повідомлення про те, чи є дане слово паліндромом. Файл для збереження програми – «Завдання 17-17-3-с».
4. Як відомо, числа в двійковій системі записують за допомогою цифр 0 та 1. Ваше завдання – перевести число з двійкового подання в десяткове.

Вхідні дані

Двійковий запис цілого невід’ємного числа. У записі числа не більше 15 цифр. Запис може починатися з нулів.

Вихідні дані

Вивести десятковий запис вхідного двійкового числа.
Файл для збереження програми – «Завдання 17-17-4-с». [17, стор. 69, 19]

Питання для самоконтролю

1. Що розуміють під рядками у мові Python?
2. Чим обмежена довжина рядка у Python?
3. Чи належать рядки до незмінних типів даних у мові програмування Python?
4. Що з переліченого підтримують рядки: звернення до елемента за індексом, отримання зрізу, конкатенацію (оператор +), повторення (оператор *), перевірку на входження (оператор in)?
5. Які вам відомі способи створення рядків у мові Python?
6. Як змінити регістр символів рядка у Python?
7. Що таке конкатенація?
8. За допомогою яких засобів можна видалити пропуски у мові програмування Python?
9. Які ви знаєте методи для роботи з рядками у Python?
10. Яке призначення функцій chr та ord?
11. Які методи в мові Python використовуються для здійснення пошуку та заміни у рядку?

18. Робота з файлами

18.1. Загальні питання

Ви, звичайно, знайомі з поняттям файлу. Файлом є документ текстового редактора Microsoft Word, зображення, створене в графічному редакторі, презентація Microsoft Power Point тощо. Система програмування мови Python також складається з багатьох файлів. Тобто так називають деяку інформацію, записану на носії (диску, флешці або ін.). З іншого боку, файл – це область на носії з цією інформацією, якій присвоєно ім'я.

При розробці програм на мові Python файли використовують тоді, коли потрібно використовувати одні й ті ж дані (перелік товарів та їх характеристики, список прізвищ тощо) у різних програмах. У такому випадку краще зберегти їх у окремому файлі і при необхідності підключати цей файл до тієї чи іншої програми.

З точки зору програмування, файли бувають двох типів:

- текстові, які містять текст, розбитий на рядки;
- двійкові, у яких можуть міститися будь-які дані та будь-які коди без обмежень; у двійкових файлах зберігаються малюнки, звуки, відеофільми і т. д.

Гігантські обсяги даних доступні в текстових файлах. У них можуть зберігатися погодні дані, соціально-економічна інформація, літературні твори та багато іншого. Читання з файлу особливо актуально для додатків, призначених для аналізу даних, але воно також може стати в нагоді в будь-якій ситуації, що вимагає аналізу або зміни інформації, що зберігається у файлі. Наприклад, програма може читати вміст текстового файлу і переписувати його з форматуванням, розрахованим на відображення інформації в браузері.

Ми будемо розглядати тільки текстові файли.

Робота з файлом в програмі на мові Python включає три основних етапи.

Спочатку треба відкрити файл, тобто зробити його доступним для програми. Якщо файл не відкритий, то програма не може до нього звертатися. При відкритті файлу вказують режим роботи з ним. Коли файл відкритий (доступний), програма виконує всі необхідні операції з ним. Після цього потрібно закрити файл, тобто звільнити його, розірвати зв'язок з програмою. **Саме при закритті всі останні зміни, зроблені програмою в файлі, записуються на диск або інший носій інформації.**

Відкриття файлу проводиться за допомогою інструкції (функції) `open()`. Її формат:

```
<ім'я файлової змінної> = open(<ім'я файлу>, <режим відкриття>)
```

Тут `<ім'я файлової змінної>` – ім'я змінної, за допомогою якої програма буде здійснювати зв'язок з файлом (файловою змінною); `<Режим відкриття>` вказує, з якою метою відкривається файл; `<ім'я файлу>` – ім'я файлу, який використовується, в лапках. Цей файл повинен знаходитися в папці, у якій розташована програма, що його використовує. Якщо це не так, то в параметрі `<ім'я файлу>` вказується повний або відносний шлях до файлу.

Можливі режими відкриття текстових файлів і їх позначення в інструкції `open()` наведені в таблиці 18.1.1:

Таблиця 18.1.1

Режим	Позначення
Тільки читання (значення за замовчуванням). Після відкриття файлу покажчик встановлюється на початок файлу. Якщо файл не існує, то збуджується виключення <code>IOError</code> .	r
Читання і запис. Після відкриття файлу покажчик встановлюється на початок файлу. Якщо файл не існує, то збуджується виключення <code>IOError</code> .	r+
Запис. Якщо файл не існує, то він буде створений. Якщо файл існує, то він буде перезаписаний. Після відкриття файлу покажчик встановлюється на початок файлу.	w
Читання і запис. Якщо файл не існує, то він буде створений. Якщо файл існує, то він буде перезаписаний. Після відкриття файлу покажчик встановлюється на початок файлу.	w+
Запис. Якщо файл не існує, то він буде створений. Запис здійснюється у кінець файлу. Вміст файлу не видаляється.	a
Читання і запис. Якщо файл не існує, то він буде створений. Запис здійснюється у кінець файлу. Вміст файлу не видаляється.	a+

Окрім того, після режиму може слідувати модифікатор:

- **b** – файл буде відкритий в бінарному режимі. Файлові методи приймають і повертають об'єкти типу `bytes`;
- **t** – файл буде відкритий в текстовому режимі (значення за замовчуванням у Windows). Файлові методи приймають і повертають об'єкти типу `str`. У цьому режимі у Windows при читанні символ `\r` буде видалений, а під час запису, навпаки, доданий.

Приклади:

- `f1 = open('input.txt')` – використано режим відкриття за замовчуванням, він знаходиться у папці, в якій розташована програма, що його використовує;
- `f = open('input.txt', 'w')` – файл відкрито для запису, він знаходиться у папці, в якій розташована програма, що його використовує;
- `f = open('Мій каталог/input.txt', 'w')` – вказано відносний шлях до файлу;
- `f = open('C:/Temp/Проба/input.txt', 'w')` – вказано абсолютний шлях до файлу;

Увага! Якщо при відкритті файлу для запису вказати ім'я вже існуючого файлу з інформацією, то вона буде втрачена.

Якщо файлу не існує – створюється новий. Якщо файл, який відкривається для читання, не знайдений, виникає помилка.

Для закриття файлу призначений метод `close()`.

Приклади:

- `f.close()`;
 - `inputf.close()`
- де `f` та `inputf` імена файлових змінних.

18.2. Зчитування всього файлу

Для початку нам знадобиться файл з кількома рядками тексту. Нехай це буде файл з числом « π » з точністю до 30 знаків, по 10 знаків на рядок:

```
pi_digits.txt
3.1415926535
8979323846
2643383279
```

Щоб випробувати приклади, або введіть дані в редакторі і збережіть файл з ім'ям `pi_digits.txt`, або завантажте файл з ресурсів книги на [сторінці ...](#). Збережіть файл в каталозі, в якому будуть зберігатися програми цієї глави.

Наступна програма відкриває цей файл, читає його і виводить вміст на екран:

```
file_reader.py
with open('pi_digits.txt') as file_object:
    contents = file_object.read()
    print(contents)
```

У першому рядку цієї програми багато заслуговує на нашу увагу. Почнемо з функції `open()`. Щоб виконати будь-які операції з файлом – навіть просто вивести його вміст, – спочатку необхідно відкрити файл. Функція `open()` отримує один аргумент: ім'я файлу, що відкривається. Python шукає файл з вказаним ім'ям в каталозі, в якому знаходиться файл поточної програми. У даному прикладі виконується програма `file_reader.py`, тому Python шукає файл `pi_digits.txt` в каталозі, в якому зберігається `file_reader.py`. Функція `open()` повертає об'єкт, який представляє файл. В даному випадку `open('pi_digits.txt')` повертає об'єкт, який представляє файл `pi_digits.txt`. Python зберігає цей об'єкт у змінній `file_object`, з якою ми будемо працювати пізніше у програмі.

Конструкція з ключовим словом `with` закриває файл після того, як потреба в ньому відпаде. Зверніть увагу: в цій програмі є виклик `open()`, але немає виклику `close()`. Файли можна відкривати і закривати явними викликами `open()` і `close()`; але якщо через помилку в програмі команда `close()` залишиться не виконаною, то файл не буде закритий. На перший погляд це не страшно, але некоректне закриття файлів може привести до втрати або псування даних. А якщо функція `close()` буде викликана занадто рано, програма спробує працювати з закритим (тобто недоступним) файлом, що призведе до нових помилок. Не завжди можна заздалегідь визначити, коли потрібно закривати файл, але з наведеною конструкцією Python зробить це за вас. Вам залишається лише відкрити файл і працювати з ним так, як потрібно, сподіваючись на те, що Python закриє його автоматично в потрібний момент.

Після того як у програмі з'явиться об'єкт, який представляє файл `pi_digits.txt`, у другому рядку програми використовується метод `read()`,

який читає весь вміст файлу і зберігає його вміст в одному довгому рядку у змінній `contents`. При виведенні значення `contents` на екрані з'являється весь вміст файлу:

```
3.1415926535
8979323846
2643383279
```

```
>>>
```

Єдина відмінність між виведенням і вихідним файлом – зайвий порожній рядок у кінці виведення. Звідки він узявся? Метод `read()` повертає його при читанні, якщо досягнуто кінець файлу. Якщо ви хочете видалити зайвий порожній рядок, додайте виклик `rstrip()` до команди `print`:

```
file_reader_1.py
with open('pi_digits.txt') as file_object:
    contents = file_object.read()
    print(contents.rstrip())
```

Нагадаємо, що метод `rstrip()` видаляє всі пробіли в кінці рядка. Тепер виведення точно відповідає вмісту вихідного файлу:

```
3.1415926535
8979323846
2643383279
```

```
>>>
```

18.3. Шляхи до файлів

Якщо передати функції `open()` просте ім'я файлу, таке як `pi_digits.txt`, Python шукає файл в тому каталозі, у якому знаходиться файл, що виконується.

У деяких випадках (в залежності від того, як організовані ваші робочі файли) файл, що відкривається, може і не перебувати в одному каталозі з файлом програми. Наприклад, файл програми може знаходитися у каталозі `python_work`; в каталозі `python_work` створюється інший каталог з ім'ям `text_files` для текстових файлів, з якими працює програма. І хоча папка `text_files` знаходиться в `python_work`, проста передача `open()` імені файлу з `text_files` не підійде, тому що Python проведе пошук файлу в `python_work` і на цьому зупиниться; пошук не буде продовжено у вкладеному каталозі `text_files`. Щоб відкрити файли з каталогу, відмінному від того, у якому зберігається файл програми, необхідно вказати шлях – тобто наказати Python шукати файли в конкретному місці файлової системи.

Оскільки каталог `text_files` знаходиться в `python_work`, для відкриття файлу із `text_files` можна скористатися відносним шляхом. Відносний шлях наказує Python шукати файли в каталозі, який задається щодо каталогу, в якому знаходиться поточний файл програми. В системі Linux і OS X це виглядає так:

```
with open('text_files/имя_файла.txt') as file_object:
```

Цей рядок означає, що файл `.txt` слід шукати в каталозі `text_files`; він передбачає, що каталог `text_files` знаходиться в `python_work` (так воно і є). У

системах Windows в шляхах до файлів використовується зворотний слеш (\) , але використання у програмі звичайного також не призводить до помилки:

```
with open('text_files\ім'я_файлу.txt') as file_object:
```

В абсолютному і відносному шляху можна вказати як прямі, так і зворотні слеші. Всі слеші будуть автоматично перетворені з урахуванням значення атрибута `sep` з модуля `os.path`. Значення цього атрибута залежить від операційної системи.

Також можна точно визначити місцезнаходження файлу у вашій системі незалежно від того, де зберігається виконувана програма. Такі шляхи називаються абсолютними і використовуються в тому випадку, якщо відносний шлях не працює. Наприклад, якщо каталог `text_files` знаходиться не в `python_work`, а в іншому місці (скажімо, в каталозі з іменем `other_files`), то передати `open()` шлях `'text_files/filename.txt'` не вийде, тому що Python буде шукати вказаний каталог тільки всередині `python_work`. Щоб пояснити Python, де слід шукати файл, необхідно записати повний шлях.

Абсолютні шляхи зазвичай довше відносних, тому їх краще зберігати в змінних, які потім передаються `open()`. В Linux і OS X абсолютні шляхи виглядають так:

```
file_path = '/home/ehmatthes/other_files/text_files/  
ім'я_файлу.txt'
```

```
with open(file_path) as file_object:
```

У Windows вони виглядають так:

```
file_path = 'C:\Users\ehmatthes\other_files\text_files\  
ім'я_файлу.txt'
```

```
with open(file_path) as file_object:
```

З абсолютними шляхами ви зможете читати файли з будь-якого каталогу вашої системи. Поки буде простіше зберігати файли в одному каталозі з файлами програм або в каталогах, вкладених в каталог з файлами програм (таких як `text_files` з розглянутого прикладу).

ПРИМІТКА

Іноді в системах сімейства Windows слеш у шляху до файлів інтерпретується неправильно. Якщо ви використовуєте Windows, але не отримуєте очікуваних результатів, спробуйте використовувати символи зворотного слеша.

18.4. Читання по рядках

У процесі читання файлу часто буває потрібно обробити кожен рядок. Можливо, ви шукаєте якусь інформацію у файлі або збираєтеся якимось чином змінити текст, наприклад при читанні файлу з метеорологічними даними ви обробляєте кожен рядок, у якому в описі погоди зустрічається слово «сонячно». Або, припустимо, у новинах ви шукаєте кожен рядок з тегом заголовка і замінюєте його спеціальними елементами форматування.

Для послідовної обробки кожного рядка у файлі можна скористатися циклом `for`:

```
file_reader_2.py
```



```
filename = 'pi_digits.txt'
with open(filename) as file_object:
    for line in file_object:
        print(line)
```

Ім'я файлу, з якого зчитується інформація, зберігається у змінній `filename`. Це стандартний прийом при роботі з файлами: бо змінна `filename` не представляє конкретний файл (це всього лише рядок, яка повідомляє Python, де знайти файл), ви зможете легко замінити `'pi_digits.txt'` ім'ям іншого файлу, з яким ви збираєтеся працювати. Після виклику `open()` об'єкт, який представляє файл і його вміст, зберігається у змінній `file_object`. Ми знову використовуємо синтаксис `with`, щоб доручити Python відкривати і закривати файл в потрібний момент. Для перегляду вмісту всі рядки файлу перебираються в циклі `for`.

На цей раз порожніх рядків виявляється ще більше:

```
3.1415926535
```

```
8979323846
```

```
2643383279
```

```
>>>
```

Порожні рядки з'являються через те, що кожен рядок у текстовому файлі завершується невидимим символом нового рядка. Команда `print` додає свій символ нового рядка при кожному виклику, тому в результаті кожен рядок завершується двома символами нового рядка: один прочитаний з файлу, а інший доданий командою `print`. Виклик `rstrip()` в команді `print` видаляє зайві порожні рядки:

```
file_reader_2.py
filename = 'pi_digits.txt'
with open(filename) as file_object:
    for line in file_object:
        print(line.rstrip())
```

Тепер виведення знову відповідає вмісту файлу:

```
3.1415926535
```

```
8979323846
```

```
2643383279
```

```
>>>
```

18.5. Створення списку рядків на основі вмісту файлів

При використанні `with` об'єкт файлу, що повертається викликом `open()`, доступний тільки в межах блоку `with`, що його містить. Якщо ви хочете, щоб вміст файлу залишався доступним за межами блоку `with`, збережіть рядки файлу в списку всередині блоку і надалі працюйте з отриманим списком. Одні частини файлу можна обробити негайно та відкласти інші для обробки у майбутньому.

У наступному прикладі рядки `pi_digits.txt` зберігаються в списку в блоці `with`, після чого виводяться за межами цього блоку:

```
filename = 'pi_digits.txt'
with open(filename) as file_object:
    lines = file_object.readlines()
for line in lines:
    print(line.rstrip())
```

У третьому рядку метод `readlines()` послідовно читає кожен рядок з файлу і зберігає його в списку. Список зберігається у змінній `lines`, з якою можна продовжити роботу після завершення блоку `with`. У четвертому рядку в простому циклі `for` виводяться всі елементи списку `lines`. Так як кожному елементу `lines` відповідає рівно один рядок файлу, виведення точно відповідає його вмісту.

18.6. Робота із вмістом файла

Після того як файл буде зчитаний у пам'ять, ви зможете обробляти дані так, як вважаєте за потрібне. Для початку спробуємо побудувати один рядок з усіма цифрами з файлу без проміжних пробілів:

```
pi_string.py
filename = 'pi_digits.txt'

with open(filename) as file_object:
    lines = file_object.readlines()

pi_string = ''
for line in lines:
    pi_string += line.rstrip()

print(pi_string)
print(len(pi_string))
```

Спочатку програма відкриває файл і зберігає кожен рядок цифр у списку – точно так, як це робилося в попередньому прикладі. У рядку 4 створюється змінна `pi_string` для зберігання цифр числа "π". Далі йде цикл, який додає до `pi_string` кожен рядок цифр, з якої видаляється символ нового рядка. У рядках 7 і 8 програма виводить рядок і його довжину:

```
3.1415926535 8979323846 2643383279
36
```

Змінна `pi_string` містить пробіли, які були присутні на початку кожного рядка цифр. Щоб видалити їх, досить використовувати `strip()` замість `rstrip()`:

```
pi_string_1.py
```



```

filename = 'pi_digits.txt'

with open(filename) as file_object:
    lines = file_object.readlines()

pi_string = ''
for line in lines:
    pi_string += line.strip()

print(pi_string)
print(len(pi_string))

```

У підсумку ми отримуємо рядок, що містить значення «пі» з точністю до 30 знаків. Довжина рядка дорівнює 32 символам, тому що в нього також включаться початкова цифра 3 і крапка:

```

3.141592653589793238462643383279
32

```

ПРИМІТКА

Читаючи дані з текстового файлу, Python інтерпретує весь текст у файлі як рядок. Якщо ви читаєте з текстового файлу число і хочете працювати з ним в числовому контексті, перетворіть його в ціле число функцією `int()` або в дійсне число функцією `float()`.

18.7. Великі файли: мільйон цифр

До цього моменту ми обмежувалися аналізом текстового файлу, який складався всього з трьох рядків, але код цих прикладів буде працювати і з набагато більшими файлами. Починаючи з текстового файлу, що містить значення «пі» до 1 000 000 знаків (замість 30), ви зможете створити один рядок, який містить всі ці цифри. Змінювати програму взагалі не доведеться – достатньо передати їй інший файл. Також ми обмежимося виведенням перших 50 цифр, щоб не довелося чекати, поки в терміналі не прокрутить мільйон знаків:

```

pi_string_2.py
filename = 'pi_million_digits.txt'

with open(filename) as file_object:
    lines = file_object.readlines()

pi_string = ''
for line in lines:
    pi_string += line.strip()

print(pi_string[:52] + "...")
print(len(pi_string))

```

З вихідних даних видно, що рядок дійсно містить значення «пі» з точністю до 1 000 000 знаків:

```

3.14159265358979323846264338327950288419716939937510...
1000002

```

Python не встановлює ніяких обмежень на довжину даних, з якими ви можете працювати. Вона обмежується хіба що об'ємом пам'яті вашої системи.

18.8. Запис у файл

Один з найпростіших способів збереження даних – запис у файл. Текст, записаний у файл, залишиться доступним і після закриття терміналу з виведенням вашої програми. Ви зможете проаналізувати результати після завершення програми або передати свої файли іншим. Ви також зможете написати програми, які знову читають збережений у пам'яті текст і працюють з ним.

Запис у порожній файл

Щоб записати текст до файлу, необхідно викликати `open()` з другим аргументом, який повідомляє Python, що ви збираєтесь записувати дані у файл. Щоб побачити, як це робиться, напишемо просте повідомлення і збережемо його у файлі (замість того щоб просто вивести на екран):

```
write_message.py
filename = 'programming.txt'

with open(filename, 'w') as file_object:
    file_object.write("Мені подобається програмування.")
```

При виклику `open()` в цьому прикладі передаються два аргументи. Перший аргумент, як і раніше, містить ім'я файлу, що відкривається. Другий аргумент `'w'` повідомляє Python, що файл повинен бути відкритий в режимі запису. Файли можна відкривати в режимі читання (`'r'`), запису (`'w'`), приєднання (`'a'`) або в режимі, що допускає як читання, так і запис в файл (`'r+'`). Якщо аргумент режиму не вказано, Python за замовчуванням відкриває файл в режимі тільки для читання.

Якщо файл, що відкривається для запису, ще не існує, функція `open()` автоматично створює його. Будьте уважні, відкриваючи файл в режимі запису (`'w'`): якщо файл існує, то Python *знищить* його дані перед поверненням об'єкта файлу.

У рядку 3 метод `write()` використовується з об'єктом файлу для запису рядка у файл. Програма не виводить дані на термінал, але, відкривши файл **programming.txt**, ви побачите в ньому один рядок:

```
programming.txt
```

Мені подобається програмування.

Цей файл нічим не відрізняється від будь-якого іншого текстового файлу на вашому комп'ютері. Його можна відкрити, записати в нього новий текст, скопіювати / вставити текст тощо.

ПРИМІТКА

Python може записувати в текстові файли тільки рядкові дані. Якщо ви захочете зберегти в текстовому файлі числову інформацію, дані доведеться попередньо перетворити в рядки функцією `str()`.

Багаторядковий запис

Функція `write()` не додає символи нового рядка до записаного тексту. А це означає, що якщо ви записуєте відразу кілька рядків без включення символів нового рядка, отриманий файл може виглядати не так, як ви розраховували:

```
filename = 'programming.txt'
```

```
with open(filename, 'w') as file_object:  
    file_object.write("Мені подобається програмування.")  
    file_object.write("Я люблю створювати нові ігри.")
```

Відкривши файл **programming.txt**, ви побачите, що два рядки «склеїли»:
мені подобається програмування.Я люблю створювати нові ігри.

Якщо включити символи нового рядка в команди `write()`, текст буде складатися з двох рядків:

```
filename = 'programming.txt'
```

```
with open(filename, 'w') as file_object:  
    file_object.write("Мені подобається програмування.\n")  
    file_object.write("Я люблю створювати нові ігри.\n")
```

Результат виглядає так:

```
мені подобається програмування.  
Я люблю створювати нові ігри.
```

Для форматування виводу також можна використовувати пробіли, символи табуляції та порожні рядки по аналогії з тим, як це робилося з виведенням на термінал.

Приєднання даних до файлу

Якщо ви хочете додати у файл нові дані замість того, щоб перезаписувати існуючий вміст, відкрийте файл в режимі приєднання. В цьому випадку Python не знищує вміст файлу перед поверненням об'єкта файлу. Всі рядки, що виводяться у файл, будуть додані в кінець файлу. Якщо файл ще не існує, то Python автоматично створить порожній файл.

Змінимо програму **write_message.py** і доповнимо існуючий файл **programming.txt** новими даними:

```
write_message.py
```

```
filename = 'programming.txt'
```

```
with open(filename, 'a') as file_object:  
    file_object.write("Мені подобається Python.\n")  
    file_object.write("Я люблю розв'язувати олімпіадні задачі.\n")
```

У рядку 2 аргумент 'a' використовується для відкриття файлу в режимі приєднання (замість перезапису існуючого файлу). Далі записуються два нові рядки, які додаються до вмісту **programming.txt**:

```
programming.txt
```

```
мені подобається програмування.  
Я люблю створювати нові ігри.  
мені подобається Python.  
Я люблю розв'язувати олімпіадні задачі.
```

В результаті до вихідного змісту файлу додається новий текст.

18.9. Практикум. Завдання 18.9

Програма зчитує двоцифрове число і виводить через пропуск кожен цифру окремо. Введення і виведення даних повинно здійснюватися за допомогою файлів.

Вхідні дані

Натуральне число на проміжку від 10 до 99 включно.

Вихідні дані

Спочатку першу цифру числа і через пропуск другу. [19, №1]

Файл для збереження програми – «Завдання 18-9».

Технологія виконання

1. У цьому та деяких інших завданнях ми використовуватимемо вимоги, що наведені у якості прикладу програми з використанням файлів для введення / виведення на ресурсі <https://www.e-olymp.com> [19].

2. Створюємо текстовий файл у який заносимо вхідні дані (двоцифрове число, наприклад «63») й зберігаємо під ім'ям «input» розширення «.txt» Windows додасть автоматично.

3. Створюємо файл програми та зберігаємо під ім'ям «Завдання 18-9».

4. Вводимо текст програми та зберігаємо зміни:

```
inputfile = open('input.txt', 'r')
ab = int(inputfile.read())
a = ab // 10
b = ab % 10
inputfile.close()
outputfile = open('output.txt', 'w')
outputfile.write(str(a)+' '+str(b)+'\n')
outputfile.close()
```

5. При виконанні інструкції у першому рядку функція `open()` отримує два аргументи: ім'я файлу, що відкривається та «режим відкриття». Python шукає файл з вказаним ім'ям в каталозі, у якому знаходиться файл поточної програми. У даному прикладі виконується програма «Завдання 18-9.py», тому Python шукає файл «input.txt» в каталозі, у якому зберігається «Завдання 18-9.py». Функція `open()` повертає об'єкт, який представляє файл. В даному випадку `open('input.txt', 'r')` повертає об'єкт, який представляє файл «input.txt». Файл відкрито у режимі для читання ('r'). Python зберігає посилання на цей об'єкт у змінній `inputfile`, з якою ми будемо працювати пізніше у програмі.

6. Виконання наступної інструкції дозволяє зчитати дані з файлу (двоцифрове число) «input.txt» та зберегти посилання на нього у змінній `ab`.

7. Після використання даних з файлу «input.txt» закриваємо його за допомогою команди «`inputfile.close()`».

8. Далі за допомогою інструкції «`outputfile=open('output.txt', 'w')`» Python шукає файл з вказаним ім'ям в каталозі, в якому знаходиться файл поточної програми. У даному прикладі виконується програма «Завдання 1-18-9.py», тому Python шукає файл «output.txt» в каталозі, в якому зберігається «Завдання 1-18-9.py». Функція `open()` повертає об'єкт, який представляє файл. В даному випадку `open('output.txt', 'w')` повертає об'єкт, який представляє файл «output.txt». Файл відкрито у режимі для запису ('w'). Python зберігає посилання на цей об'єкт у змінній `outputfile`.

9. За допомогою інструкції `outputfile.write(str(a)+' '+str(b)+'\n')` здійснюємо запис вихідних даних у файл «output.txt» та закриваємо його за допомогою інструкції `outputfile.close()`.

10. Існує загальна рекомендація відкривати файли, використовуючи інструкцію `with`. Це пов'язано з тим, що вона гарантує автоматичне закриття дескрипторів відкритих файлів після того, як виконання програми залишає контекст інструкції `with`. За її допомогою розглянуту вище програму можна подати у вигляді:

```
with open('input.txt', 'r') as inputfile:
    ab = int(inputfile.read())
    a = ab // 10
    b = ab % 10
with open('output.txt', 'w') as outputfile:
    outputfile.write(str(a)+' '+str(b)+'\n')
```

11. Файл нової програми збережемо під ім'ям «Завдання 1-18-9-1».

12. Виконайте збережені програми за допомогою їх відкриття подвійним натисканням миші для кількох наборів даних, що заносяться до файлу «input.txt» перед кожним виконанням та перевірте у файлі «output.txt» результати кожного виконання відповідної програми.

18.10. Практикум. Завдання 18.10

Записати до текстового файлу «output.txt», який знаходиться у тій же папці, що і програма, текст «Мені подобається Python!».

Файли для збереження програми – «Завдання 18-10-1» (з використанням інструкції `with`) та «Завдання 18-10-2» (без використання інструкції `with`).

Технологія виконання

1. Програма може бути, наприклад, такою:

```
#Відкриваємо файл для запису.
f = open('output.txt', 'w')
#Записуємо до нього текст.
f.write('Мені подобається Python!')
#Закриваємо файл.
f.close()
```

2. Або з використанням інструкції `with` – такою:

```
with open('output.txt', 'w') as f:
    f.write('Мені подобається Python!')
```

3. І у першому і у другому випадках у каталозі (папці) місцезнаходження файлу програми у файл «output.txt» (у випадку його відсутності він буде створений) буде записано текст: `Мені подобається Python!`.

4. Збережіть розглянуті програми у файлах з іменами, що відповідають умові.

18.11. Практикум. Завдання 18.11

Записати до текстового файлу «output.txt» (який знаходиться у тій же папці, що і програма) рядок, значення якого задається користувачем (вводиться з клавіатури) в ході виконання програми.

Файли для збереження програми – «Завдання 18-11-1» та «Завдання 18-11-2».

Технологія виконання

1. Єдиною відмінністю від попереднього завдання є введення рядка користувачем. Врахувавши це отримаємо програму:

```
#Відкриваємо файл для запису.  
f = open('output.txt', 'w')  
#Записуємо до нього текст.  
f.write(input('Введіть текст: '))  
#Закриваємо файл.  
f.close()
```

2. Варіант з використанням інструкції with:

```
with open('output.txt', 'w') as f:  
    f.write(input('Введіть текст: '))
```

3. І у першому і у другому випадках у файл «output.txt» буде записано текст, введений користувачем, наприклад, **Слава Україні!**. Відкрийте його та перегляньте результат.

4. Збережіть розглянуті програми у файлах з іменами, що вказані в умові.

18.12. Практикум. Завдання 18.12

Записати до текстового файлу «Перші 100.txt» (який знаходиться у тій же папці, що і програма) всі числа від 1 до 100 з пробілом між ними. Файл для збереження програми – «Завдання 18-12».

Технологія виконання

1. Наведемо один з можливих варіантів програми:

```
with open('Перші 100.txt', 'w') as f:  
    for i in range(1, 101):  
        f.write(str(i)+' ')
```

2. Виконайте програму та перегляньте результати, відкривши файл «Перші 100.txt».

18.13. Практикум. Завдання 18.13

Записати до текстового файлу «Вивчаємо Python.txt», (який повинен знаходитись у папці «Навчання», що, у свою чергу, знаходиться у тому ж каталозі, що і програма) текст, який буде мати вигляд:

```
Мені подобається вивчати Python!  
давайте це робити разом!
```

Файл для збереження програми – «Завдання 18-13».

Технологія виконання

1. Створюємо та зберігаємо файл програми у потрібній папці. У цьому ж каталозі створюємо папку «Навчання».

2. Один з можливих варіантів програми:

```
with open('Навчання/Вивчаємо Python.txt', 'w') as f:  
    f.write('Мені подобається вивчати Python!\n')  
    f.write('Давайте це робити разом!')
```

3. При відкритті програми ми використали відносний шлях до файлу «Вивчаємо Python.txt», завдяки цьому файл буде створений (у випадку його відсутності) саме у папці «Навчання».

4. До рядку виводу в файл включений керуючий параметр '\n', що відповідає символу переходу на наступний рядок.
5. Виконайте програму та перевірте результати, відкривши файл «Вивчаємо Python.txt».

18.14. Практикум. Завдання 18.14

Записати до текстового файлу «output.txt» кожний символ, введеного користувачем рядка, на окремому рядку файлу (який знаходиться у тому ж каталозі, що і програма). Файл для збереження програми – «Завдання 18-14».

Технологія виконання

1. Створюємо та зберігаємо файл програми.
2. Використавши досвід виконання попередніх завдань 18.9-18.13, отримаємо:

```
with open('output.txt', 'w') as f:
    for sim in input('Ведіть рядок: '):
        f.write(sim+'\n')
```
3. Виконайте програму та перевірте результати, відкривши файл «output.txt».

18.15. Практикум. Завдання 18.15

Записати до текстового файлу «output.txt» кожне з чисел від 1 до 7 на окремому рядку файлу (який знаходиться у тому ж каталозі, що і програма). Файл для збереження програми – «Завдання 18-15».

Технологія виконання

1. Створіть та збережіть файл програми.
2. Завдання майже аналогічне завданню 18.12. У якості допомоги можна скористатися [лістингом програми](#).
3. Виконайте програму та перевірте результати, відкривши файл «output.txt».

18.16. Практикум. Завдання 18.16

Записати до текстового файлу «output.txt» кожний елемент заданого списку зі значень рядкового типу на окремому рядку файлу (який знаходиться у тому ж каталозі, що і програма). Файл для збереження програми – «Завдання 18-16-п».

Технологія виконання

1. Створіть та збережіть файл програми.
2. Розглянемо два з можливих програмних реалізацій алгоритму. Перша:

```
sp=['Мені', 'подобається', 'Python']
with open('output.txt', 'w') as f:
    for i in range(len(sp)):
        f.write(sp[i]+'\\n')
```
3. Збережіть файл програми під ім'ям «Завдання 18-16-1». Виконайте цей варіант програми та перевірте результати, відкривши файл «output.txt».
4. Можна також розглядати не індекси елементів списку, а їх значення:


```
sp=['Я', 'вивчаю', 'Python']
with open('output.txt', 'w') as f:
    for el in sp:
        f.write(el+'\n')
```

5. Збережіть цей варіант у файлі під ім'ям «Завдання 18-16-2». Виконайте його та перевірте результати, відкривши файл «output.txt».

18.17. Практикум. Завдання 18.17

Записати до текстового файлу «Футбольні клуби.txt» кожної з 5 введених назв футбольних клубів на окремому рядку. Файл для збереження програми – «Завдання 18-17».

Технологія виконання

1. Створіть та збережіть файл програми.
2. За умови використання досвіду, отриманого при виконанні завдань 18.9-18.16 розв'язання є доволі простим.
3. У якості допомоги можна скористатися [лістингом програми](#).
4. Виконайте програму та перевірте результат, відкривши файл «Футбольні клуби.txt».

18.18. Практикум. Завдання 18.18

Розробити програму для виведення на екран двох рядків файлу, створеного при розв'язанні завдання 1.18.13. Порожніх рядків між ними бути не повинно. Файл для збереження програми – «Завдання 18-18».

Технологія виконання

1. Створіть та збережіть файл програми.
2. Перемістіть у каталог у якому знаходиться програма файл «Вивчаємо Python.txt», створений при виконанні завдання 18.13.

3. Наведемо три програмних реалізацій алгоритму. Перша:

```
# Відкриваємо файл для читання.
f=open('Вивчаємо Python.txt', 'r')
# Зчитуємо перший рядок файлу та запам'ятовуємо його.
s=f.readline()
# Друкуємо значення змінної s.
print(s, end='')
# Аналогічно з другим рядком.
s=f.readline()
print(s, end='')
```

4. За допомогою засобів Python програму можна оптимізувати, наприклад, так:

```
with open('Вивчаємо Python.txt', 'r') as f:
    print(f.readline(), end='')
    print(f.readline(), end='')
```

5. Або використати для зчитування рядків цикл for наступним чином:

```
with open('Вивчаємо Python.txt', 'r') as f:
    for i in f:
        print(i, end='')
```

6. Для читання всіх рядків файлу можна використати також таку конструкцію:

```
for s in open("ім'я файлу"):
```

```
...
```

У ній у змінній `s` будуть перебиратися значення всіх рядків. Звернемо увагу на те, що:

- 1) відкривати файл для читання окремою інструкцією не потрібно;
- 2) в інструкції `open()` можна не вказувати режиму відкриття файлу для читання;
- 3) закривати файл за допомогою методу `close()` не потрібно, він закриється автоматично після закінчення циклу `for`.

7. Отримуємо найкоротший варіант коду:

```
for i in open('Вивчаємо Python.txt'):  
    print(i, end='')
```

8. Виконавши всі варіанти програми, отримаємо однаковий результат:

```
Мені подобається вивчати Python!  
Давайте це робити разом!
```

18.19. Практикум. Завдання 18.19

Розробити програму для виведення на екран всіх рядків файлу, створеного при розв'язанні завдання 1.18.15. Порожніх рядків між ними бути не повинно. Файл для збереження програми – «Завдання 18-19».

Технологія виконання

1. Створіть та збережіть файл програми.
2. Завдання аналогічне попередньому.
3. У якості допомоги можна скористатися [лістингом програми 1](#) або [лістингом програми 2](#).

18.20. Практикум. Завдання 18.20

Текстовий файл «`input.txt`» містить декілька рядків тексту (наприклад, для простоти, натуральні числа: 1, 2, 3, 4, ..., 7). Розробити програму для виведення на екран n -го рядку файлу. Файл для збереження програми – «Завдання 1-18-20».

Технологія виконання

1. Створіть та збережіть файл програми.
2. Текстовий файл є «файлом послідовного доступу до даних». Це означає, що для того, щоб прочитати 100-е за рахунком значення з файлу, потрібно спочатку прочитати попередні 99. У своїй внутрішній пам'яті система зберігає положення покажчика (файлового курсора), який визначає поточне місце в файлі. При відкритті файлу покажчик встановлюється у самий початок файлу, при читанні зміщується на позицію, наступну за прочитаними даними. Якщо потрібно повторити читання з початку файлу, потрібно його закрити, а потім знову відкрити. Тому для розв'язання завдання потрібно прочитати, але не використовувати перші ($n - 1$) рядків файлу, а потім прочитати і вивести на екран n -й рядок.
3. Врахувавши пункт 2 отримаємо програму:

```

n=int(input('Введіть номер рядка: '))
with open('input.txt', 'r') as f:
    # Пропускаємо перші n-1 рядків.
    for i in range(n-1):
        f.readline()
    # Виводимо n-й рядок.
    print(f.readline(), end='')

```

4. Виконайте програму та проаналізуйте результат.

18.21. Практикум. Завдання 18.21

Текстовий файл «input.txt» містить декілька рядків тексту (наприклад, для простоти, натуральні числа: 1, 2, 3, 4, ..., 7). Розробити програму для запису всіх рядків файлу до списку та вивести значення його елементів через пробіл. Файл для збереження програми – «Завдання 18-21».

Технологія виконання

1. Створіть та збережіть файл програми.
2. В Python є оригінальні можливості записати всі рядки текстового файлу до списку. Зокрема, це можна зробити за допомогою методів `readlines()` і `strip()`. Наведемо кілька програмних реалізацій алгоритму. Перша:

```

with open('input.txt', 'r') as f:
    # Записуємо всі рядки файлу до списку.
    a = [line.strip() for line in f]
    # Виводимо на екран значення елементів списку.
    for i in a:
        print(i, end=' ')

```

3. Друга:

```

with open('input.txt', 'r') as f:
    # Записуємо всі рядки файлу до списку.
    a = f.readlines()
    # Виводимо на екран значення елементів списку.
    for i in a:
        print(i.strip(), end=' ')

```

4. Ще один оригінальний спосіб – одночасне використання методів `read()` і `split()`. Перший метод читає всі рядки файлу і представляє їх як один рядок, а другий розділяє її на вихідні частини і записує їх окремо в список. Відповідна програмна реалізація матиме вигляд:

```

with open('input.txt', 'r') as f:
    # Записуємо всі рядки файлу до списку.
    a = f.read().split()
    # Виводимо на екран значення елементів списку.
    for i in a:
        print(i.strip(), end=' ')

```

5. Виконавши всі варіанти програми, отримаємо однаковий результат:

```

1 2 3 4 5 6 7
>>>

```

18.22. Практикум. Завдання 18.22

Всі рядки файлу, створеного при розв'язанні «Завдання 18-14», записати в інший файл. Список не використовувати. Файл для збереження програми – «Завдання 1-18-22-n» (n=1, 2, 3, ...).

Технологія виконання

1. Створіть та збережіть файл програми.
2. Скопіюйте файл, що був створений при виконанні «Завдання 18-14», до папки у якій збережено програму та перейменуйте його у «input.txt».
3. Розглянемо три з можливих варіантів програмних реалізацій алгоритму розв'язування задачі. Перший:

```
# Відкриваємо існуючий файл для читання.  
f1 = open('input.txt', 'r')  
# Відкриваємо новий файл для запису.  
f2 = open('output.txt', 'w')  
for i in f1: # Для всіх рядків вихідного файла:  
# значення чергового рядка записуємо до іншого файлу.  
    f2.write(i)  
# У тому рахунку символ '\n'.  
f1.close()  
f2.close()
```

4. Другий варіант:

```
with open('input.txt', 'r') as f1:  
    with open('output.txt', 'w') as f2:  
        for i in f1:  
            f2.write(i)
```

5. І остання – найкоротша:

```
with open('output.txt', 'w') as f2:  
    for i in open('input.txt'):  
        f2.write(i)
```

6. Виконайте всі варіанти програми для різних наборів даних та порівняйте результати. Спробуйте знайти інший варіант розв'язання.

18.23. Практикум. Завдання 18.23

У текстовому файлі «countries.txt» на окремих рядках записані назви 7 держав, у файлі «capitals.txt» – їх столиці (також на окремих рядках і в тому ж порядку, що і назви держав). Комп'ютер під час виконання програми після введення користувачем країни повинен вивести її столицю. Файл для збереження програми – «Завдання 18-23».

Технологія виконання

1. Створіть та збережіть файл програми.
2. Алгоритм розв'язання може бути таким:
 - ⇒ вводимо назву країни та присвоюємо значення змінній;
 - ⇒ відкриваємо файл «countries.txt» для читання;
 - ⇒ створюємо список країн;
 - ⇒ визначаємо номер (n) введеної країни у списку;
 - ⇒ відкриваємо файл «capitals.txt» для читання;
 - ⇒ створюємо список столиць;

⇒ виводимо назву столиці, що відповідає номеру n.

3. Наведемо один з можливих варіантів програмної реалізації:

```
c=input('Введіть назву країни: ')
with open('countries.txt') as f1:
    # Створюємо список країн.
    kr=[ln.strip() for ln in f1]
    # Визначаємо номер (n) введеної країни у списку.
    for i in range(len(kr)):
        if kr[i]==c:
            n=i

with open('capitals.txt') as f2:
    # Створюємо список столиць.
    st=[line.strip() for line in f2]
    # Виводимо назву столиці, що відповідає номеру n.
    print(st[n])
```

4. Виконайте програму та проаналізуйте результати. Знайдіть інший (свій) варіант розв'язання.

18.24. Практикум. Завдання 18.24

Дописати до текстових файлів «countries.txt» та «capitals.txt» із «Завдання 18-23» n назв країн та n їх столиць відповідно. Файл для збереження програми – «Завдання 18-24».

Технологія виконання

1. Створіть та збережіть файл програми під ім'ям «Завдання 18-24».
2. Скопіюйте файли «countries.txt» та «capitals.txt» до папки у якій зберегли файл програми.
3. Наведемо алгоритм додавання інформації до файлу:
 - ⇒ відкрити файл на додавання до нього інформації;
 - ⇒ записати до нього новий рядок;
 - ⇒ закрити файл (звертаємо увагу на необхідність використання під час запису керуючого символу).
4. Розглянемо одну з можливих програмних реалізацій:

```
n=int(input('Введіть кількість рядків: '))
# Додаємо n назв країн до файлу 'countries.txt'.
with open('countries.txt', 'a') as f1:
    for i in range(n):
        c=input('Введіть назву країни: ')
        f1.write(c+'\n')
# Додаємо n назв столиць до файлу 'capitals.txt'.
with open('capitals.txt', 'a') as f2:
    for j in range(n):
        cap=input('Введіть назву столиці: ')
        f2.write(cap+'\n')
```

5. Виконайте програму кілька разів та перегляньте результати. Створіть інший варіант програмної реалізації.

18.25. Практикум. Завдання 18.25

Замінити у існуючому текстовому файлі «countries.txt» рядок із заданим номером k . Файл для збереження програми – «Завдання 18-25».

Технологія виконання

1. Створіть та збережіть файл програми під ім'ям «Завдання 18-25».
2. Наведемо алгоритм заміни рядка із заданим номером в існуючому файлі:
 - ⇒ відкрити файл для читання;
 - ⇒ всі рядки файлу записати до списку;
 - ⇒ закрити файл;
 - ⇒ змінити відповідне значення у списку;
 - ⇒ записати всі елементи зміненого списку у той же файл.
3. Одна з можливих програмних реалізацій:

```
k=int(input('Введіть номер рядка: '))
# Додаємо n назв країн до файлу 'countries.txt'.
with open('countries.txt', 'r') as f1:
    sp=f1.readlines()
    nz=input('Введіть нове значення: ')
    sp[k-1]=nz+'\n'
with open('countries.txt', 'w') as f2:
    for el in sp:
        f2.write(el)
```

4. Виконайте програму кілька разів та перегляньте результати.

18.26. Практикум. Завдання 18.26

Маємо N цілих чисел. Який найбільший добуток можна отримати, використавши тільки три з цих чисел? Введення і виведення даних повинно здійснюватися за допомогою файлів.

Вхідні дані

В першому рядку ціле невід'ємне число N ($3 \leq N \leq 10^6$). У другому рядку N цілих чисел, кожне по модулю не перевищує 10^5 .

Вихідні дані

Значення найбільшого добутку трьох з них.

Приклади

Вхідні дані	Вихідні дані
9	315
3 5 -9 7 4 0 9 -3 5	

Файл для збереження програми – «Завдання 1-18-26». [19, №7507]

Технологія виконання

1. У цьому та деяких інших завданнях ми використовуємо вимоги, що наведені у якості прикладу програми з використанням файлів для введення / виведення на ресурсі <https://www.e-olymp.com> [19].

2. Створюємо текстовий файл з ім'ям «input.txt» у який заносимо вхідні дані.
3. Створюємо файл програми та зберігаємо під ім'ям «Завдання 18-26».
4. Ідея розв'язання полягає у тому щоб відсортувати список, що містить числа, задані в умові, у неспадному порядку та обрати найбільший з двох добутоків: трьох найбільших або двох найменших та найбільшого елементу списку.

5. Розглянемо два з можливих варіантів програми:

```
f1=open('input.txt')
# Зберігаємо рядки файлу у списку.
sp=f1.readlines()
f1.close()
# Стіорюємо список з цілих чисел.
x=[int(i) for i in sp[1].split()]
# Відсортовуємо список за неспаданням.
x.sort()
# Знаходимо добутки.
d1=x[0]*x[1]*x[-1]
d2=x[-3]*x[-2]*x[-1]
# Визначаємо максимальний з добутоків.
d=max(d1, d2)
f2=open('output.txt', 'w')
# Записуємо результат до файлу.
f2.write(str(d))
f2.close()
```

6. І більш короткий:

```
with open('input.txt') as f1:
    sp=f1.readlines()
x=[int(i) for i in sp[1].split()]
x.sort()
with open('output.txt', 'w') as f2:
    f2.write(str(max(x[0]*x[1]*x[-1], x[-3]*x[-2]*x[-1])))
```

7. Виконайте програму для різних наборів даних, перегляньте та проаналізуйте результати.

18.27. Практикум. Завдання для самостійного виконання

1. Записати до текстового файлу «output.txt», який знаходиться у тій же папці, що і програма, текст «Привіт, Світ!». Файл для збереження програми – «Завдання 18-27-1».
2. Записати до текстового файлу «Непарні.txt» (який знаходиться у тій же папці, що і програма) всі непарні числа менші 100 через пробіл. Файл для збереження програми – «Завдання 18-27-2».
3. Дано рядок символів. Записати кожен її символ, починаючи з останнього, в окремому рядку текстового файлу. Файл для збереження програми – «Завдання 18-27-3».
4. Отримати і записати до файлу 15 випадкових цілих чисел з інтервалу [50, 100]. Файл для збереження програми – «Завдання 18-27-4».
5. Записати в текстовий файл квадрати чисел від 1 до 10 (кожен на окремому рядку). Файл для збереження програми – «Завдання 18-27-5».

6. Дано 10 назв міст. Записати їх до текстового файлу (кожна назва на окремому рядку). Список не використовувати. Файл для збереження програми – «Завдання 18-27-6».
7. Є список з 20 чисел, серед яких є від'ємні. Записати кожне від'ємне число на окремому рядку текстового файлу. Файл для збереження програми – «Завдання 18-27-7».
8. Записати до текстового файлу рядок з декількох слів, значення якого задається в ході виконання програми (між усіма словами рядка повинен бути один пробіл). Використовуючи цей файл, отримати список, в якому будуть всі слова заданого рядка. Файл для збереження програми – «Завдання 18-27-8».
9. У текстовому файлі gosud.txt на окремих рядках записані назви 10 держав, у файлі stolitsi.txt - їх столиці (також на окремих рядках і в тому ж порядку, що і назви держав). Дано назви двох столиць. Визначити назви відповідних держав. Файл для збереження програми – «Завдання 18-27-9».
10. Є файл, у кожному рядку якого записано слово. Визначити середню «довжину» слова. Файл для збереження програми – «Завдання 18-27-10».
11. Є текстовий файл, що містить декілька рядків. Створити програму для обміну місцями двох рядків цього файлу. Файл для збереження програми – «Завдання 18-27-11».

Питання для самоконтролю

1. Які основні етапи роботи з файлами в програмі?
2. Які можливі режими відкриття файлу?
3. Як здійснюється зчитування всього файлу у мові Python?
4. Яким чином у Python здійснюється зчитування великих файлів?
5. Як здійснюється у мові Python запис даних у порожній файл?
6. Яким чином можна додати нові дані до файлу?
7. Як можна рядки файлу записати до списку?

Додатки

Додаток 1. Елементи функціонального програмування у Python

Функціональне програмування – розділ дискретної математики і парадигма програмування, в якій процес обчислення трактується як обчислення значень функцій в математичному розумінні останніх (на відміну від функцій як підпрограм в процедурному програмуванні).

Функціональне програмування передбачає обходитися обчисленням результатів функцій від вихідних даних та результатів інших функцій, і не передбачає явного зберігання стану програми. Відповідно, не передбачає воно й змінюваність цього стану (на відміну від імперативного, де однією з базових концепцій є змінна, що зберігає своє значення і дозволяє змінювати його в міру виконання алгоритму).

Елементи функціонального програмування в Python можуть бути корисні будь-якому програмісту, бо дозволяють гармонійно поєднувати виразну потужність цього підходу з іншими підходами. Як правило, коли говорять про елементи функціонального програмування в Python, то розглядають такі функції вищого порядку: `lambda`, `map`, `filter`, `zip`.

Анонімні функції

Крім звичайних функцій мова Python дозволяє використовувати анонімні функції, які називаються *лямбда-функціями*. Анонімна функція описується за допомогою ключового слова `lambda` за наступною схемою:

```
lambda [<Параметр 1> [... , <Параметр N>]]: <Значення, що повертається>
```

Після ключового слова `lambda` можна вказати параметри, що передаються. У якості параметра `<Значення, що повертається>` вказується вираз, результат виконання якого буде повернутий функцією. Як видно зі схеми, у лямбда-функцій немає імені. З цієї причини їх і називають анонімними функціями.

У якості значення лямбда-функція повертає посилання на об'єкт-функцію, яку можна зберегти у змінній або передати в якості параметра у іншу функцію. Викликати лямбда-функцію можна, як і звичайну, за допомогою круглих дужок, усередині яких розташовані передані параметри. Приклад використання лямбда-функцій:

```
>>> f1 = lambda: 7 * 11          # Функція без параметрів
>>> print(f1())
77
>>> f2 = lambda x, y: x - y      # Функція з двома параметрами
>>> print(f2(55, 10))
45
>>> f3 = lambda x, y, z: x * y + z # Функц. з трьома параметрами
>>> print(f3(5, 7, 40))
75
```

Як і в звичайних функціях, деякі параметри лямбда-функцій можуть бути необов'язковими. Для цього параметрам у визначенні функції присвоюється значення за замовчуванням:

```
>>> f = lambda a, b=5: a * b
>>> print(f(8))
40
>>> print(f(10, 15))
150
```

Найчастіше не зберігають посилання у змінній, а відразу передають як параметр в іншу функцію. Наприклад, метод списків `sort()` дозволяє вказати функцію користувача в параметрі `key`. Відсортуємо список спочатку з урахуванням регістру, а потім без урахування регістру символів, вказавши в якості параметра лямбда-функцію:

```
arr = ["приклад1", "Приклад3", "Приклад2"]
arr.sort()
print(arr)

arr1 = ["приклад1", "Приклад3", "Приклад2"]
arr1.sort(key=lambda s: s.lower())
print(arr1)
```

Результат виконання:

```
['Приклад2', 'Приклад3', 'приклад1']
['приклад1', 'Приклад2', 'Приклад3']
```

Функція `map`

Вбудована функція `map()` дозволяє застосувати певну функцію до кожного елементу послідовності. Функція має наступний формат:

```
map(<Функція>, <Послідовність 1> [...], <Послідовність N>)]
```

Функція `map()` повертає об'єкт, що підтримує ітерації, а не список, як це було раніше в Python 2. Щоб отримати список у версії Python 3, необхідно результат передати в функцію `list()`.

У якості параметру `<Функція>` вказується посилання на функцію (назва функції без круглих дужок), якій буде передаватися поточний елемент послідовності. У середині функції зворотного виклику необхідно повернути нове значення.

Піднесемо до квадрату кожний елемент списку.

```
def func(el):
    """ Піднесення до квадрату значення кожного ел. списку """
    return el**2
arr = [1, 2, 3, 4, 5, 6, 7]
print( list( map(func, arr) ) )
```

Результат виконання:

```
[1, 4, 9, 16, 25, 36, 49]
```

Функції `map()` можна передати кілька послідовностей. В цьому випадку в функцію зворотного виклику будуть передаватися відразу кілька елементів, розташованих в послідовності на однаковому зміщенні. Підсумуємо елементи трьох списків:

```
arr1 = [1, 2, 3, 4, 5]
arr2 = [10, 20, 30, 40, 50]
arr3 = [100, 200, 300, 400, 500]
print( list( map(lambda x,y,z: x+y+z, arr1, arr2, arr3) ) )
```

Результат виконання:

```
[111, 222, 333, 444, 555]
```

У цьому прикладі ми використали у якості функції зворотного виклику лямбда-функцію.

Якщо кількість елементів у послідовності буде різною, то в якості обмеження вибирається послідовність з мінімальною кількістю елементів:

```
def funk(a1, a2, a3):
    return a1 + a2 + a3
```

```
arr1 = [1, 2, 3, 4, 5]
arr2 = [10, 20, 30]
arr3 = [100, 200, 300, 400]
print( list( map(funk, arr1, arr2, arr3) ) )
```

Результат виконання:

```
[111, 222, 333]
```

Функція zip

Вбудована функція `zip()` на кожній ітерації повертає кортеж, що містить елементи послідовностей, які розташовані на однаковому зміщенні. Функція повертає об'єкт, що підтримує ітерації, а не список, як це було раніше у Python 2. Щоб отримати список у версії Python 3, необхідно результат передати до функції `list()`. Формат функції:

```
zip (<Послідовність 1> [..., <Послідовність N>])
```

Приклад:

```
>>> zip([1, 2, 3], [4, 5, 6], [7, 8, 9])
<zip object at 0x032C3CD8>
>>> list(zip([1, 2, 3], [4, 5, 6], [7, 8, 9]))
[(1, 4, 7), (2, 5, 8), (3, 6, 9)]
```

Якщо кількість елементів в послідовності буде різною, то до результату потраплять тільки елементи, які існують у всіх послідовностях на однаковому зміщенні:

```
>>> list(zip([1, 2, 3], [4, 6], [7, 8, 9, 10]))
[(1, 4, 7), (2, 6, 8)]
```

В якості ще одного прикладу переробимо нашу програму підсумовування елементів трьох списків і використовуємо функцію `zip()` замість функції [map\(\)](#):

```
arr1 = [1, 2, 3, 4, 5]
arr2 = [10, 20, 30, 40, 50]
arr3 = [100, 200, 300, 400, 500]
arr = [x + y + z for (x, y, z) in zip(arr1, arr2, arr3)]
print(arr)
```

Результат виконання:

```
[111, 222, 333, 444, 555]
```

Функція filter

Функція `filter()` дозволяє виконати перевірку елементів послідовності. Формат функції:

```
filter (<Функція>, <Послідовність>)
```

Якщо в першому параметрі замість назви функції вказати значення `None`, то кожен елемент послідовності буде перевірений на відповідність значенню `True`. Якщо елемент в логічному контексті повертає значення `False`, то його не буде додано до результату, що повертається. Функція повертає об'єкт, що підтримує ітерації, а не список або кортеж, як це було раніше в Python 2. Щоб отримати список у версії Python 3, необхідно результат передати в функцію `list()`. Приклад:

```
>>> filter(None, [1, 0, None, [], 2])
<filter object at 0x032D0CB0>
>>> list(filter(None, [1, 0, None, [], 2]))
[1, 2]
```

У першому параметрі можна вказати посилання на функцію. У цю функцію в якості параметра буде передаватися поточний елемент послідовності. Якщо елемент слід додати до повертаемого функцією `filter()` значення, то всередині функції зворотного виклику слід повернути значення `True`, в іншому випадку – значення `False`. Видалимо всі від'ємні значення зі списку:

```
def func(elem):
    return elem >= 0
arr = [-1, 2, -3, 4, 0, -20, 10]
arr = list(filter(func, arr))
print(arr)
```

Результат:

```
[2, 4, 0, 10]
```

Ще дві корисні функції

Функція range()

Функція `range()` має наступний формат:

```
range ([<Початок>,, <Кінець> [, <Крок>])
```

Перший параметр задає початкове значення. Якщо параметр `<Початок>` не вказано, то за замовчуванням використовується значення 0. У другому параметрі вказується кінцеве значення. Слід зауважити, що це значення не входить до значень які повертаються. Якщо параметр `<Крок>` не вказано, то використовується значення 1. Функція повертає об'єкт, що підтримує ітераційний протокол. За допомогою цього об'єкта всередині циклу `for` можна отримати значення поточного елемента.

Наприклад:

```
>>> for i in range(1, 10, 2): print(i, end=' ')

1 3 5 7 9
```

Функція eval()

У Python є вбудована функція `eval()`, яка виконує рядок з кодом і повертає результат виконання. Це дуже потужна, але в той же час і дуже небезпечна інструкція, особливо якщо рядки, які ви передасте в `eval`, отримані не з перевіреного джерела. Ввівши певний код, зловмисник може, наприклад, завдати шкоди даним, що містяться на диску комп'ютера.

Функція `eval()` має наступний формат:

```
eval(expression, globals=None, locals=None)
```

Вона виконує вираз (програмний код) у рядку `expression` і повертає результат.

Приклад:

```
>>> a=5
>>> b=10
>>> eval('3*a+b')
25
```

Додаток 2. Список ключових слів та вбудованих ідентифікаторів

Ключові слова:

['False' (Хиба, Хибний), 'None' (Немає), 'True' (Істина, Істинний), 'and' (і), 'as' (як), 'assert', 'break' (перервати), 'class', 'continue' (продовжити), 'def', 'del', 'elif' (інакше, якщо), 'else' (інакше), 'except', 'finally', 'for' (для), 'from' (із), 'global', 'if' (якщо), 'import' (імпорт), 'in' (в), 'is' (є), 'lambda', 'nonlocal', 'not' (ні), 'or' (або), 'pass', 'raise', 'return' (повернення), 'try', 'while' (доки), 'with' (з), 'yield'].

Вбудовані ідентифікатори:

['ArithmeticError', 'AssertionError', 'AttributeError', 'BaseException', 'BlockingIOError', 'BrokenPipeError', 'BufferError', 'BytesWarning', 'ChildProcessError', 'ConnectionAbortedError', 'ConnectionError', 'ConnectionRefusedError', 'ConnectionResetError', 'DeprecationWarning', 'EOFError', 'Ellipsis', 'EnvironmentError', 'Exception', 'False', 'FileExistsError', 'FileNotFoundError', 'FloatingPointError', 'FutureWarning', 'GeneratorExit', 'IOError', 'ImportError', 'ImportWarning', 'IndentationError', 'IndexError', 'InterruptedError', 'IsADirectoryError', 'KeyError', 'KeyboardInterrupt', 'LookupError', 'MemoryError', 'ModuleNotFoundError', 'NameError', 'None', 'NotADirectoryError', 'NotImplemented', 'NotImplementedError', 'OSError', 'OverflowError', 'PendingDeprecationWarning', 'PermissionError', 'ProcessLookupError', 'RecursionError', 'ReferenceError', 'ResourceWarning', 'RuntimeError', 'RuntimeWarning', 'StopAsyncIteration', 'StopIteration', 'SyntaxError', 'SyntaxWarning', 'SystemError', 'SystemExit', 'TabError', 'TimeoutError', 'True', 'TypeError', 'UnboundLocalError', 'UnicodeDecodeError', 'UnicodeEncodeError', 'UnicodeError', 'UnicodeTranslateError', 'UnicodeWarning', 'UserWarning', 'ValueError', 'Warning', 'WindowsError', 'ZeroDivisionError', '__build_class__', '__debug__', '__doc__', '__import__', '__loader__', '__name__', '__package__', '__spec__', 'abs', 'all', 'any', 'ascii', 'bin', 'bool', 'breakpoint', 'bytearray', 'bytes', 'callable', 'chr', 'classmethod', 'compile', 'complex', 'copyright', 'credits', 'delattr', 'dict', 'dir', 'divmod' (частка-остача), 'enumerate', 'eval', 'exec', 'exit', 'filter', 'float', 'format', 'frozenset', 'getattr', 'globals', 'hasattr', 'hash', 'help', 'hex', 'id', 'input', 'int', 'isinstance', 'issubclass', 'iter', 'len', 'license', 'list', 'locals', 'map', 'max' (максимум), 'memoryview', 'min' (мінімум), 'next', 'object', 'oct', 'open', 'ord', 'pow', 'print', 'property',

'quit', 'range' (діапазон), 'repr', 'reversed', 'round', 'set', 'setattr', 'slice', 'sorted', 'staticmethod', 'str', 'sum' (сума), 'super', 'tuple', 'type', 'vars', 'zip'].

Додаток 3. Модулі у Python

Модулем у мові Python називається будь-який файл з програмою. Кожен модуль може імпортувати інший модуль, отримуючи, таким чином, доступ до ідентифікаторів всередині імпортованого модуля. Слід зауважити, що модуль який імпортується може містити програму не тільки на мові Python. Наприклад, можна імпортувати скомпільований модуль, написаний на мові C.

Інструкція **import**

Імпортувати модуль дозволяє інструкція `import`.

Інструкція `import` має такий вигляд:

```
import <Назва модуля 1> [as <Псевдонім 1>] [, ..., <Назва модуля N> [as <Псевдонім N>]]
```

Після ключового слова `import` вказується назва модуля. Зверніть увагу на те, що назва не повинна містити розширення та шлях до файлу.

За один раз можна імпортувати відразу кілька модулів, перерахувавши їх через кому. Як приклад підключимо модулі `time` і `math`:

```
>>> import time, math
>>> print(time.strftime("%d.%m.%Y"))
21.03.2019
>>> print(math.pi)
3.141592653589793
```

Якщо назва модуля є занадто довгою і її незручно вказувати кожен раз для доступу до ідентифікаторів всередині модуля, то можна створити псевдонім. Псевдонім задається після ключового слова `as`. Створимо псевдонім для модуля `math`:

```
>>> import math as m #Створення псевдоніма
>>> print(m.pi)
3.141592653589793
```

Тепер доступ до атрибутів модуля `math` може здійснюватися тільки за допомогою ідентифікатора `m`. Ідентифікатор `math` у цьому випадку використовувати вже не можна.

Всі ідентифікатори всередині імпортованого модуля доступні тільки через ідентифікатор, зазначений в інструкції `import`.

Інструкція **from**

Для імпортування певних ідентифікаторів з модуля можна скористатися інструкцією `from`.

Інструкція має кілька форматів:

```
from <Назва модуля> import <Ідентифікатор 1> [as <Псевдонім 1>]
[, ..., <Ідентифікатор N> [as <Псевдонім N>]]
from <Назва модуля> import (<Ідентифікатор 1> [as <Псевдонім 1>]
[, ..., <Ідентифікатор N> [as <Псевдонім N>]])
from <Назва модуля> import *
```


Перші два формати дозволяють імпортувати модуль і зробити доступними тільки зазначені ідентифікатори. Для довгих імен можна призначити псевдонім, вказавши його після ключового слова `as`. Як приклад зробимо доступними константу `pi` та функцію `factorial()` з модуля `math`, а для назви функції створимо псевдонім:

```
from math import pi, factorial as frl
print(pi)
print(frl(5))
input()
```

Результат виконання:

```
3.141592653589793
120
```

Ідентифікатори можна розмістити на декількох рядках, вказавши їх назви через кому всередині круглих дужок.

Третій формат інструкції `from` дозволяє імпортувати всі ідентифікатори з модуля. Для прикладу імпортуємо всі ідентифікатори з модуля `math`:

```
>>> from math import *
>>> print(pi)
3.141592653589793
>>> print(cos(pi))
-1.0
```

Додаток 4. Інформація про олімпіадні завдання

Завдання 1.12.3*

II етап Всеукраїнської олімпіади юних програмістів. Чернігівська область. Рік 1996, 11 клас. (Дещо змінене.)

Завдання для самостійного виконання до пункту 1.12 (№2).*

II етап Всеукраїнської олімпіади юних програмістів. Чернігівська область. Рік 1996, 11 клас. (Модифіковане.)

Завдання для самостійного виконання до пункту 1.12 (№3).*

III етап Всеукраїнської олімпіади юних програмістів. Чернігівська область. Рік 2005. (Дещо змінене.)

Завдання 1.13.4*

Задача 1. «Числа Фібоначчі». [11, стор. 9-12]

Завдання 1.14.1*

II етап Всеукраїнської олімпіади юних програмістів. Чернігівська область. Рік 1996, 10 клас.

Додаток 5. Перелік кольорів


	black		bisque		lightgreen		slategrey
	k		darkorange		forestgreen		lightsteelblue
	dimgray		burlywood		limegreen		cornflowerblue
	dimgrey		antiquewhite		darkgreen		royalblue
	grey		tan		green		ghostwhite
	gray		navajowhite		g		lavender
	darkgrey		blanchedalmond		lime		midnightblue
	darkgray		papayawhip		seagreen		navy
	silver		moccasin		mediumseagreen		darkblue
	lightgray		orange		springgreen		mediumblue
	lightgrey		wheat		mintcream		blue
	gainsboro		oldlace		mediumspringgreen		b
	whitesmoke		floralwhite		mediumaquamarine		slateblue
	white		darkgoldenrod		aquamarine		darkslateblue
	w		goldenrod		turquoise		mediumslateblue
	snow		cornsilk		lightseagreen		mediumpurple
	rosybrown		gold		mediumturquoise		blueviolet
	lightcoral		lemonchiffon		azure		indigo
	indianred		khaki		lightcyan		darkorchid
	brown		palegoldenrod		paleturquoise		darkviolet
	firebrick		darkkhaki		darkslategray		mediumorchid
	maroon		ivory		darkslategrey		thistle
	darkred		beige		teal		plum
	red		lightyellow		darkcyan		violet
	r		lightgoldenrodyellow		c		purple
	mistyrose		olive		cyan		darkmagenta
	salmon		y		aqua		m
	tomato		yellow		darkturquoise		fuchsia
	darksalmon		olivedrab		cadetblue		magenta
	coral		yellowgreen		powderblue		orchid
	orangered		darkolivegreen		lightblue		mediumvioletred
	lightsalmon		greenyellow		deepskyblue		deeppink
	sienna		chartreuse		skyblue		hotpink
	seashell		lawngreen		lightskyblue		lavenderblush
	chocolate		sage		steelblue		palevioletred
	saddlebrown		lightsage		aliceblue		crimson
	sandybrown		darksage		dodgerblue		pink
	peachpuff		honeydew		lightslategray		lightpink
	peru		darkseagreen		lightslategray		
	linen		palegreen		slategray		

Додаток 6. Проблема української мови в консолі

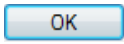
Ця проблема виникає через невідповідність кодувань Windows і консолі Windows, тобто інтерпретації кодів символів в їх графічне представлення. Докладніше про кодування символів – [за посиланням](#).

Для вирішення проблеми необхідно налаштувати шрифт консолі. Для цього потрібно зробити таке:

⇒ відкрити вікно консолі (наприклад виконати подвійне клацання миші на значку програми на Python);

⇒ натиснути на значок  у лівому верхньому кутку екрана та обрати пункт **Свойства**;

⇒ у діалоговому вікні, що відкриється, на вкладці шрифт обрати **Lucida Console**;

- ⇒ можна також змінити інші налаштування;
- ⇒ натиснути кнопку ;
- ⇒ закрити вікно консолі.

Додаток 7. Генератори списків у Python

У мові програмування Python існує спеціальна синтаксична конструкція, яка дозволяє за певними правилами створювати заповнені списки. Така конструкція називається генератором списків (*list comprehension*). Її зручність полягає в більш короткому записі програмного коду, ніж якби список створювався звичайним способом. [21]

Наприклад, треба створити список, заповнений натуральними числами до певного числа. «Класичний» спосіб може виглядати приблизно так:

```
>>> a = []
>>> for i in range(1,15):
    a.append(i)
```

```
>>> a
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14]
```

На створення списку пішло три рядки коду. Генератор же зробить це за один:

```
>>> a = [i for i in range(1,15)]
>>> a
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14]
```

Тут конструкція `[i for i in range(1,15)]` є генератором списку. Вся конструкція береться в квадратні дужки, що як би говорить, що буде створено список. У середині квадратних дужок можна виділити три частини:

- 1) що робимо з елементом (у цьому випадку нічого не робимо, просто додаємо до списку);
- 2) що беремо (в даному випадку елемент `i`),
- 3) звідки беремо (тут з об'єкта `range`).

Частини відокремлені одна від одної ключовими словами `for` та `in`.

Розглянемо наступний приклад:

```
>>> b = [3, -7, -2, 4, -4, 7, 5]
>>> c = [i**2 for i in b]
>>> c
[9, 49, 4, 16, 16, 49, 25]
```

У цьому випадку в генераторі списку береться кожен елемент зі списку `a` і підноситься до квадрату. Таким чином:

- 1) що робимо – підносимо елемент до квадрату;
- 2) що беремо – елемент;
- 3) звідки беремо – зі списку `a`.

В кінець генератора можна додавати конструкцію `if`. Наприклад, треба з рядка витягнути усі цифри:

```
>>> d = "m77sj94k 5 5sd231 9"
>>> s = [int(i) for i in d if '0'<=i<='9']
>>> s
[7, 7, 9, 4, 5, 5, 2, 3, 1, 9]
Або заповнити список числами, кратними 7 або 17:
>>> a = [i for i in range(7,150) if i%7 == 0 or i%17 == 0]
>>> a
[7, 14, 17, 21, 28, 34, 35, 42, 49, 51, 56, 63, 68, 70, 77,
84, 85, 91, 98, 102, 105, 112, 119, 126, 133, 136, 140, 147]
```

Таким чином, генератори дозволяють створювати списки легше і швидше. Однак замінити ними досить складні конструкції не завжди вийде. Наприклад, коли умова перевірки повинна включати гілку `else`.

Гузь Валентина

Термінологічний словник

У зв'язку з неузгодженістю використання деяких термінів у різноманітній літературі з програмування (особливо в книгах, перекладених з інших мов) вважаємо за доцільне навести тлумачення деяких понять і термінів програмування та алгоритмізації, що ми використовували у цьому посібнику.

Ідентифікатор (англ. *identifier*) – ім'я об'єкта програми (змінної, масиву, структури, функції тощо), що дозволяє звернутись до об'єкта; ознака, яка цілком визначає сутність, у наперед визначеному просторі.

Інструкція або **оператор** (англ. *Statement*) – найменша автономна частина мови програмування; команда або набір команд. Програма звичайно являє собою послідовність інструкцій.

Ітерація – одноразове виконання тіла циклу.

Ітератор (від англ. *iterator* – нумератор) – інтерфейс, що надає доступ до елементів колекції (масиву або контейнера) і навігацію по ним. У різних системах ітератори можуть мати різні загальноприйняті назви. У термінах систем управління базами даних ітератори називаються курсором. У найпростішому випадку ітератором у низькорівневих мовах є покажчик. Більше інформації можна отримати [тут](#). [9]

Клас – це спеціальна конструкція, яка використовується для групування пов'язаних змінних та функцій. При цьому, згідно з термінологією ООП, глобальні змінні класу (члени-змінні) називаються полями даних (також властивостями або атрибутами), а члени-функції називають методами класу. Створений та ініціалізований екземпляр класу називають об'єктом класу. На основі одного класу можна створити безліч об'єктів, що відрізнятимуться один від одного своїм станом (значеннями полів). [9] Докладніше дивись, наприклад, [тут](#).

Масив (англ. *array*) (у деяких мовах програмування також таблиця, ряд, матриця) – структура даних, що зберігає набір значень (елементів масиву), які ідентифікуються за індексом або набором індексів, що приймають [цілі](#) (або приводяться до цілих) значення з деякого заданого безперервного діапазону. Одновимірний масив можна розглядати як реалізацію [абстрактного типу даних вектор](#).

Масив може бути одновимірним ([вектором](#), лінійною таблицею), та багатовимірним (наприклад, двовимірною таблицею), тобто таким, де індексом є не одне число, а сукупність з декількох чисел, кількість яких збігається з розмірністю масиву.

У найпростішому випадку масив має сталу довжину за всіма вимірами та може зберігати дані тільки одного, заданого при описі, типу. Ряд мов підтримує також динамічні масиви, довжина яких може змінюватися під час роботи програми, та гетерогенні масиви, які можуть у різних елементах зберігати дані різних типів. [9]

Більше інформації про масиви можна отримати, наприклад, [тут](#).

Об'єктно-орієнтоване програмування (ООП) – це спосіб організації програми, що дозволяє використовувати один і той же код багаторазово. На відміну від функцій і модулів ООП дозволяє не тільки розділити програму на фрагменти, а й описати предмети реального світу у вигляді об'єктів, а також організувати зв'язки між цими об'єктами. Більше інформації можна отримати [тут](#). [9]

Операнд (англ. operand) – аргумент [операції](#); дані, які обробляються командою; граматична конструкція, яка позначає вираз, що задає значення аргументу операції; іноді операндом називають місце або позицію в тексті, де має стояти аргумент операції.

Операція – конструкція в мовах програмування, аналогічна за записом математичних операцій, тобто спеціальний спосіб запису деяких дій.

Найбільш часто застосовуються арифметичні, логічні та рядкові операції. На відміну від функцій, операції часто є базовими елементами мови і позначаються різними символами пунктуації, а не алфавітно-цифровими. ([Докладніше ...](#)) [9].

Підпрограма (англ. subroutine) – частина [програми](#), яка реалізує певний алгоритм і дозволяє звернення до неї з різних частин загальної (головної) програми. В термінах мов програмування: функції ([C](#)), процедури ([Pascal](#)), методи (в термінології [об'єктно-орієнтованого програмування](#) в мовах [C++](#), [Java](#), [C#](#) та ін.). [Докладніше ...](#) [9].

Програмування – процес створення [комп'ютерних програм](#).

Структура даних (англ. Data structure) – програмна одиниця, що дозволяє зберігати і обробляти багато однотипних і / або логічно пов'язаних даних за допомогою комп'ютерної техніки. Для додавання, пошуку, зміни та видалення даних структура даних надає певний набір функцій, з яких складається її інтерфейс.

Таблична величина – це впорядкований набір змінних деякого типу.

Тіло циклу – послідовність інструкцій, призначена для циклічного (багаторазового) виконання (послідовність інструкцій, що повторюються у процесі виконання оператора циклу).

Література та використані джерела

1. Бриггс Дж. Python для детей. Самоучитель по программированию / Джейсон Бриггс; пер. с англ. Станислава Ломакина; [науч. ред. Д. Абрамова] – М. : Манн, Иванов и Фербер, 2017. – 320 с.
2. Мэтиз Эр. Изучаем Python. Программирование игр, визуализация данных, веб-приложения / Эрик Мэтиз – СПб.: Питер, 2017. – 496 с.
3. Программирование для детей / К. Вордерман, Дж. Вудкок, Ш. Макаманус и др.; пер. с англ. С. Ломакина. – М.: Манн, Иванов и Фербер, 2015. – 224 с.
4. Прохоренок Н. А. Python 3 и PyQt. Разработка приложений. – СПб.: БХВ-Петербург, 2012. – 704 с.
5. <https://sites.google.com/site/pythonukr> – Програмування на мові Python (3.x). Початковий курс.
6. <http://pythontutor.ru> – Питонтьютор.
7. <https://www.w3schools.com/python> – Python Tutorial.
8. <https://informatics.mccme.ru> – Дистанционная подготовка по информатике.
9. <https://wikipedia.org> – Вікіпедія.
10. «[Основи програмування на мові Python](#)» – дистанційний курс Оксани Пасічник на YouTube.
11. Мельник В. І. Інформатика. Олімпіадні задачі з розв'язаннями. – Х.: Вид. група «Основа», 2010. – 160 с.
12. https://uk.wikipedia.org/wiki/Python#cite_note-7 – Python. Вікіпедія.
13. Окулов С. М. Основы программирования. – М.: ЮНИМЕДИАСТАИЛ, 2002. – 424 с.
14. Збірник задач та розв'язків II етапу Всеукраїнської учнівської олімпіади з інформатики 2016-2017 навчального року/ С. М. Бондаренко, А. А. Борзаков, А. М. Дасюк та ін.; за заг. ред. Ю. М. Літоша, О. Є. Баранової, О. М. Смірної. – Чернігів: ЧОППО імені К. Д. Ушинського, 2017 – 29 с.
15. Бондаренко С. М., Зуб В. В. II етап Всеукраїнської олімпіади юних програмістів (завдання та розв'язки): Посібник для вчителів та учнів. – Чернігів: РВК ЧОППО, 2002. – 64с.
16. Руденко В. Д. Інформатика для загальноосвітніх навчальних закладів з поглибленим вивченням інформатики: підруч. для 9 кл. загальноосвіт. навч. закл. / В. Д. Руденко, Н. В. Речич, В. О. Потієнко. – Харків : Вид-во «Ранок», 2017. – 240 с.
17. Матвійчук С. В. Практикум програмування Python / C++ на e-olymp.com (збірник задач з рекомендаціями до їх розв'язання) / С. В. Матвійчук, С. С. Жуковський – Житомир: Вид-во ЖДУ ім. І. Франка, 2019. – 232 с.
18. Введение в Python – <https://wm-help.net/lib/b/book/2773507910/>.
19. <https://www.e-olymp.com> – Інтернет-портал організаційно-методичного забезпечення дистанційних олімпіад з програмування для обдарованої молоді навчальних закладів України.

20. Златопольский Д. М. Основы программирования на языке Python. – М.: ДМК Пресс, 2017. – 284 с.
21. <https://younglinux.info/python/feature/generators> – Генераторы списков в Python.

Гузь Валентина

Гузъ Валентина

Для приміток

Гузь Валентина