Г. О. Козуб, М. А. Семенов

ПРОГРАМУВАННЯ (PYTHON)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ
ДЕРЖАВНИЙ ЗАКЛАД
„ЛУГАНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА”

Г. О. Козуб, М. А. Семенов

ПРОГРАМУВАННЯ (PYTHON)

Методичні рекомендації до лабораторних робіт
для студентів спеціальності
121 – „Інженерія програмного забезпечення”

Старобільськ
ДЗ „ЛНУ імені Тараса Шевченка”
2020

УДК 004.42(072)
ББК 00.000-000
К00

Р е ц е н з е н т и :

Гоменюк С.І. **–** доктор технічних наук, декан математичного факультету Запорізького національного університету.

Ордановська О.І. – доктор педагогічних наук, декан фізико-математичного факультету Державного Південноукраїнського національного педагогічного університету імені К.Д. Ушинського.

К00 **Козуб Г.О.** Програмування : метод. рек. до лаб. робіт для студ. спец. 121 – „Інженерія програмного забезпечення” / Г. О. Козуб, Н. А. Семенов; Держ. закл. „Луган. нац. ун-т імені Тараса Шевченка”. – Старобільськ : ДЗ „ЛНУ імені Тараса Шевченка”, 2020. – 108 с.

Методичні рекомендації структуровано відповідно до розділів навчальної програми курсу „Програмування (Python)” для спеціальності 121 – „Інженерія програмного забезпечення” кафедри інформаційних технологій та систем ДЗ ЛНУ імені Тараса Шевченка. Посібник охоплює основні групи питань з алгоритмізації та розробки програм мовою Python, починаючи з базових основ програмування до основних конструкцій Python (на основі стандартної бібліотеки).

Навчальний посібник призначений для студентів фізико-математичного та технічного профілю, учителів-предметників ліцеїв, коледжів, гімназій, слухачів курсів підвищення кваліфікації, а також для самоосвіти.

УДК 004.42(072)
ББК 00.000-000

Рекомендовано до друку Вченою радою
Луганського національного університету імені Тараса Шевченка
(протокол № 0 від 00 лютого 2020р.)

ЗМІСТ

ВСТУП

На сьогоднішній день Python є однією з найбільш поширених і популярних мов програмування. Python використовується для різних цілей: для створення ігор і веб-додатків, розробки внутрішніх інструментів для різноманітних проектів. Мова також широко застосовується в науковій області для теоретичних досліджень і розв'язування прикладних завдань.. Це обумовлює актуальність предмету вивчення, якому присвячено методичні рекомендації.

Python – багатоцільова мова програмування, яка дозволяє писати код, що добре читається. Відносний лаконізм мови Python дозволяє створити програму, яка буде набагато коротше свого аналога, написаного на іншій мові.

Python - багатоплатформова мова програмування. Це означає, що програми на Python можна запускати в різних операційних системах без будь-яких змін.

Програми, написані на мові програмування Python, можуть бути як невеликими скриптами, так і складними системами. Мова використовується в таких проектах: BitTorrent – протокол для обміну даними, Ubuntu Software Center – вільне програмне забезпечення для пошуку, установки і видалення пакетів в системі Ubuntu Linux, Blender – програма для створення тривимірної комп'ютерної графіки, що включає засоби моделювання, анімації, вимальовування, пост-обробки відео, а також створення відеоігор, GIMP – растровий графічний редактор, із підтримкою векторної графіки, DropBox – файловий хостинг, що включає персональне хмарне сховище, синхронізацію файлів і програму-клієнт, World of Tanks, Вікіпедія, Google, YouTube, Instagram, Яндекс, Facebook і інших. Він легкий, простий у використанні і абсолютно безкоштовний.

Вивчення курсу „Програмування" є невід'ємною частиною у загальному процесі навчання студентів спеціальності 121 – „Інженерія програмного забезпечення". Головна мета полягає у формуванні базових знань з технології програмування мовою Python, умінь створення сучасних програмних продуктів з використанням об'єктно-орієнтованої парадигми і розв'язання різних інженерних завдань за допомогою комп'ютера. Знання,

отримані при вивченні цієї дисципліни, забезпечать професійну підготовку фахівців в галузі інформаційних технологій.

Однією із основних навчальних форм є лабораторні роботи, які відіграють провідну роль у формуванні навичок та застосуванні набутих знань з Python-програмування. Лабораторні заняття логічно продовжують вивчення тем, розпочатих на лекціях. Усі форми лабораторних занять призначені для відпрацювання практичних дій.

Методичні рекомендації з лабораторного практикуму структуровані відповідно до першого модулю навчальної програми курсу „ Програмування" і складаються з шести лабораторних робіт, що підібрані та укладені з матеріалів які розташовані на освітніх сайтах [5, 6, 9, 13]. Тематика робіт спрямована на вивчення як базових основ програмування так і більш складних тем. Для виконання лабораторних робіт використовується мова програмування Python та інтегроване середовище розробки з інтерактивною оболонкою IDLE (Integrated Development and Learning Environment), яке є найбільш адаптованим для навчальних цілей.

Методичні рекомендації містять назву, мету, основні теоретичні відомості та завдання для індивідуального виконання. Для кращого розуміння теми, яка розглядається, кожна лабораторна робота містить приклади програм, результати їх виконання. Наприкінці кожна робота має питання для самоперевірки отриманих знань. Зміст лабораторних занять передбачає роботу в комп'ютерних класах та самостійну роботу студентів.

Методичні рекомендації можуть бути корисними для студентів інших спеціальностей у якості посібника для самостійного опанування технологій Python-програмування.

ВИМОГИ ДО ВИКОНАННЯ Й ЗАХИСТУ ЛАБОРАТОРНИХ РОБІТ

Лабораторні роботи виконуються кожним студентом самостійно. Перед початком виконання лабораторної роботи студент повинен ознайомитися з теоретичним матеріалом, використовуючи джерела, наведені у списку рекомендованої літератури, чи будь-які інші; усвідомити завдання та порядок проведення роботи.

Під час виконання лабораторної роботи студент повинен поетапно виконати індивідуальне завдання відповідно до варіанту. Результатом проведеної роботи можуть бути розроблені програмні додатки мовою програмування Python в інтегрованому середовищі розробки з інтерактивною оболонкою IDLE.

Після виконання лабораторної роботи студент повинен оформити звіт, який має містити наступне: ПІБ студента, № групи, курсу, назву дисципліни, номер та назву теми лабораторної роботи, завдання до роботи (відповідно до номеру варіанту), розв'язання поставленого завдання (короткий опис, алгоритм, програмний код, скріншоти, відповіді на контрольні питання тощо). До звіту додаються файли проекту у вигляді архіву.

Лабораторна робота подається до захисту безпосередньо після її виконання. При захисті роботи студент демонструє результати виконаної роботи та відповідає на контрольні запитання за темою лабораторної роботи.

Кожна лабораторна робота оцінюється за бальною системою, яка встановлена робочою програмою курсу. Оцінюється якість підготовки, повнота виконання роботи, вміст збережених файлів, зміст відповідей та оформлення звітів. Оцінці "відмінно" відповідає виконання більше 90% пунктів роботи, "добре" – більше 75%, "задовільно" – більше 50%.

МОДУЛЬ 1. ОСНОВИ МОВИ ПРОГРАМУВАННЯ PYTHON

ЛАБОРАТОРНА РОБОТА №1

Тема: Установка Python з середою розробки IDLE

Мета: навчитися встановлювати основне програмне забезпечення для розробки Python-програм. Створити перші програми.

Час виконання роботи: 4 години.

Програмне забезпечення: IDLE (Python 3.X).

Короткі теоретичні відомості

Загальні поняття

Python – динамічна інтерпретована об'єктно-орієнтована скриптова мова програмування із строгою динамічною типізацією. Розроблена в 1990 році голандським програмістом Гвідо ван Россумом. Автор назвав мову на честь популярного британського комедійного серіалу 1970-х років "Повітряний цирк Монті Пайтона". Найчастіше вживане прочитання назви мови — "Па́йтон".

Python активно вдосконалюється і в даний час. Часто виходять його нові версії (рис.1.1). Офіційний сайт: http://python.org.

Рис.1.1. Роки виходу версій Python

З появою версії 3.0 розвиток Python як би пішов в двох напрямках: одночасно розвиваються і 2-га і 3-я версії Python. На час написання методичних рекомендацій вже вийшла версія Python 3.7.1.

Python - це високорівнева інтерпретована мова програмування. Назва Python відноситься як до мови програмування, так і до інтерпретатора – комп'ютерної програми, яка зчитує вихідний код (написаний на Python) і виконує інструкції (команди).

Python – досить поширена мова, яка використовується в багатьох галузях:

– розробка прикладного ПЗ;

– розробка веб-додатків (найпотужніший сервер додатків Zope і розроблена на його основі CMS Plone, на основі якої працює, наприклад, сайт ЦРУ, і маса фреймворків для швидкої розробки додатків Plones, Django, TurboGears і багато інших) ;

– використання в якості вбудованої скриптової мови в багатьох іграх, а також в офісному пакеті OpenOffice.org, 3D-редакторі Blender, СУБД Postgre;

– використання в наукових обчисленнях (з пакетами SciPy і NumPy для розрахунків, PyPlot для малювання графіків).

Щоб використовувати програмний продукт, створений мовою програмування Python або розробляти свої власні програми, необхідно завантажити й встановити на комп'ютер інтерпретатор Python з інтерактивною оболонкою IDLE (http://python.org/download/). При написанні програм використовують текстові редактори або інтегровані середовища розробки, які включають в себе різні інструменти для роботи з кодом: засіб для написання коду (текстовий редактор), інтерактивний інтерпретатор, налагоджувач тощо.

Текстові редактори та інтегровані середовища програмування для Python

Платформу Python можна встановлювати як самостійно, так і разом з інтегрованим середовищем розробки. Існує декілька середовищ розробки (Integrated Development Environment - IDE).

Найбільш популярними є PyCharm, PyDev, WingWare, Komodo IDE, Eric, Eclipse, Geany, Spyder, PyScripter.

IDLE — стандартний редактор Python. Встановлюється разом з Python для користувачів Windows, окремим пакетом для користувачів Linux.

Notepad++ - безкоштовний текстовий редактор вихідного коду, який підтримує велику кількість мов, в тому числі і Python. Лише для користувачів Windows.

PyScripter - інтегроване середовище розробки для мови програмування Python, працює під Windows. Поширюється безкоштовно (https://sourceforge.net/projects/pyscripter/).

Wing IDE 101 - вільне інтегроване середовище для Python, розроблене для навчання програмістів-початківців. Для користувачів Linux, Windows і Mac OS X. Поширюється безкоштовно.

Geany - вільний текстовий редактор з базовими елементами інтегрованого середовища розробки, доступний для операційних систем Linux, Mac OS X і Windows (https://www.geany.org/).

Sublime Text 3 - кросплатформенний текстовий редактор вихідних текстів програм та інтегроване середовище розробки. Підтримує плагіни, розроблені за допомогою мови програмування Python. Sublime Text не є вільним чи відкритим програмним забезпеченням, але деякі його плагіни розповсюджуються з вільною ліцензією, розробляються і підтримуються спільнотою розробників. Проте, можна використовувати вільно, хоча часто з'являється повідомлення про придбання ліцензії. Для користувачів Windows, Mac OS X, Linux.

PyCharm — інтегроване середовище розробки для мови програмування Python. Підтримує веб-розробку на Django. PyCharm є власницьким програмним забезпеченням. Присутні безкоштовна версія Community з усіченим набором можливостей і безкоштовна версія Edu для навчальних закладів. PyCharm працює під операційними системами Windows, Mac OS X і Linux (https://www.jetbrains.com/pycharm/).

Spyder — IDE з відкритим кодом для Python, знаходиться у вільному доступі під ліцензією MIT. Містить такі функції як багатомовний редактор, інтерактивна консоль, перегляд

документації, оглядач змінних, пошуковик файлів, пошуковик в файлах, і багато іншого. Хоча Spyder і є автономним IDE, який підтримується безліччю платформ, такими як Windows, Linux, Mac Os, Mac OS X, він також може бути використаний в якості бібліотеки розширення PyQt і може бути вбудований в додатки PyQt5 (https://github.com/spyder-ide/spyder).

Eclipse — платформа загального призначення, повна підтримка усіма операційними системами, відкритий код і підтримка всіх відомих мов, безліч плагінів (https://eclipse.org/).

Eric — IDE і за сумісництвом редактор Python від Detlev Offenbach, з відкритим кодом, підтримка Windows. Містить такі функції як налагоджувач Python і Ruby, покриття коду, автоматична перевірка коду, оболонка Python і Ruby, браузер класу і багато іншого. Також є функції для спільного редагування. Діалоги Regex і Qt, опції для створення сторонніх додатків прямо в редакторі, діаграми додатки, можливості управління проектами, а також інтерактивна оболонка Python. Багатомовний користувальницький інтерфейс, контроль версії для Subversion, Mercurial і Git, використання оголошень в плагінах, і багато іншого (https://eric-ide.python-projects.org/).

Для перекладу мови високого рівня на машинну мову доступні два типи програм:

1. **Компілятор** - перекладає весь вихідний код на машинну мову за один раз, потім машинний код виконується (рис.2).

2. **Інтерпретатор** - перекладає програму з мови високого рівня у машинну мову рядок за рядком, виконуючи кожен з них (рис.1.3).

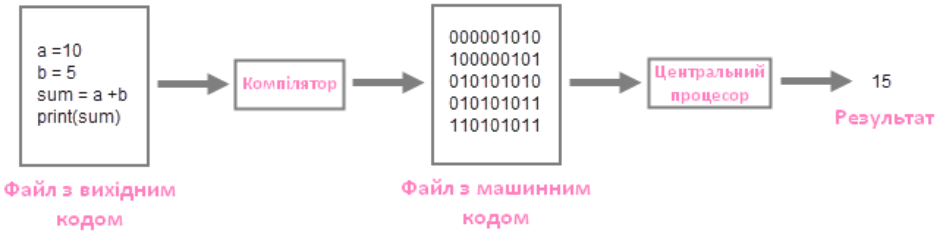


Рис. 1.2. Виконання програми компілятором

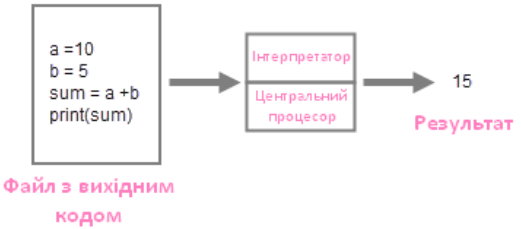


Рис. 1.3. Виконання програми інтерпретатором

Інтерпретатор Python починає свою роботу у верхній частині файлу, перекладає перший рядок на машинну мову, а потім виконує його. Цей процес повторюється до кінця файлу.

Якщо інтерпретатору Пітона дати команду **import this** (імпортувати "сам об'єкт"), то виведеться так званий "Дзен Пітона" (табл.1.1.), який ілюструє ідеологію і особливості даної мови. Глибоке розуміння цього дзену приходить тим, хто зможе освоїти мову Python в повній мірі і отримає досвід практичного програмування.

Таблиця 1.1. - *"Дзен Пітона"*

Фраза	Переклад
1. Beautiful is better than ugly.	Красиве краще за потворне.
2. Explicit is better than implicit.	Просте краще за складне.
4. Complex is better than complicated.	Складне краще за ускладнене.
5. Flat is better than nested.	Плоске краще ніж вкладене.
6. Sparse is better than dense.	Розріджене краще ніж щільне.
7. Readability counts.	Читабельність важлива.
8. Special cases aren't special enough to break the rules.	Виняткові випадки не настільки важливі, щоб порушувати правила.
9. Although practicality beats purity.	Однак практичність важливіша за чистоту.
10. Errors should never pass silently.	Помилки ніколи не повинні замовчуватися.
11. Unless explicitly silenced.	За винятком замовчування, яке задано спеціально.

Фраза	Переклад
12. In the face of ambiguity, refuse the temptation to guess.	У випадку неоднозначності не піддавайтеся спокусі вгадати.
13. There should be one - and preferably only one - obvious way to do it.	Повинен існувати один - і, бажано, тільки один - очевидний спосіб зробити це.
14. Although that way may not be obvious at first unless you're Dutch.	Хоча він може бути з першого погляду не очевидний, якщо ти не голландець.
15. Now is better than never.	Зараз краще, ніж ніколи.
16. Although never is often better than * Right * now.	Проте, ніколи частіше краще, ніж прямо зараз.
17. If the implementation is hard to explain, it's a bad idea.	Якщо реалізацію складно пояснити - це погана ідея.
18. If the implementation is easy to explain, it may be a good idea.	Якщо реалізацію легко пояснити — це може бути хороша ідея.
19. Namespaces are one honking great idea - Let's do more of those!	Простори назв - прекрасна ідея, давайте робити їх більше!

Запуск програм в терміналі Windows

Відкрийте термінальне вікно (вікно командного рядка): натисніть сполучення клавіш Win +R на клавіатурі, введіть команду **cmd**, натисніть OK і виконайте команди:

```
C:\Users>User> cd Desktop 1
C:\Users>User\Desktop> cd python_work 2
C:\Users>User\Desktop\python_work> dir 3
hello.py 4
C:\Users>User\Desktop\python_work> python hello.py 5
Hello, Python! 6
```

Назва **User** *в записі - це ім'я користувача в системі.*

1. Переходимо у папку **Desktop** (Робочий стіл) з використанням команди **cd**.

2. Переходимо у папку **python_work** з використанням команди **cd**.

3. Читаємо вміст папки **python_work** з використанням команди **dir**.

4. Відображення вмісту (єдиний файл **hello.py**).

5. Запускаємо на виконання файл **hello.py** (ввести **python** та ім'я файлу).

6. Відображення результату у вікні терміналу.

Для переходу на рівень вгору у дереві папок використовують інструкцію **cd .**

Щоб закрити вікно терміналу, введіть команду **exit**.

Повідомлення про помилку

В процесі написання і виконання програм можуть з'являтися різноманітні помилки. У таких випадках інтерпретатор Python сам сигналізує про помилку.

Наприклад, коли ми введемо в режимі інтерактивного інтерпретатора інструкцію **'19' + 81**, з'явиться таке повідомлення:

```
>>> '19' + 81
Traceback (most recent call last):
  File "<interactive input>", line 1, in <module>
TypeError: Can't convert 'int' object to str implicitly
```

Введена інструкція некоректна для Python, тому він вказав назву помилки і номер рядка, в якому вона виникла, зупинивши виконання програми.

Якщо помилка зрозуміла для нас, її виправляють. В іншому випадку, щоб дізнатися, що означає повідомлення про помилку, можна здійснити пошук в мережі Інтернет за назвою помилки.

Коментарі

Коментарі надзвичайно корисні в будь-якій мові програмування. У міру зростання обсягу і складності коду в програмі слід додавати коментарі, які описують загальний підхід до розв'язуваної задачі, - свого роду, нотатки, написані на зрозумілій мові.

У мові Python ознакою коментаря є символ #. Інтерпретатор Python ігнорує всі символи в коді після # до кінця рядка.

Наприклад:

```
>>> # Привіт, світ!
... print("Hello, world!")
Hello, world!
```

Продовження рядків

Будь-яка програма стає більш зрозумілою, якщо її рядки короткі. Рекомендована (але не обов'язкова) максимальна довжина рядка дорівнює 80 символам. Якщо ви не можете висловити свою думку в рамках 80символів, скористайтеся символом **продовження рядка** (\). Просто помістіть \ в кінець рядка, а **Python** буде діяти так, ніби це все той самий рядок.

Наприклад:

```
>>> alphabet = 'abcdefg' + \
... 'hijklmnop' + \
... 'qrstuv' + \
... 'wxyz'
>>> alphabet
'abcdefghijklmnopqrstuvwxyz'
>>> 1 + 2 + \
... 3
6
```

Порядок виконання роботи

1. Установка Python.

Спочатку перевіримо, чи встановлен Python на комп'ютері. Якщо програмний додаток встановлено, то у списку, який відкривається за командою *Панель управління, Програми та компоненти,* відображена існуюча версія Python.

2. Інсталяція програмного додатку

Зауваження! Рекомендується робити інсталяцію ПЗ з правами адміністратора.

Скачайте необхідний установник з офіційного сайту (рис.1. 4.) https://www.python.org/downloads/.

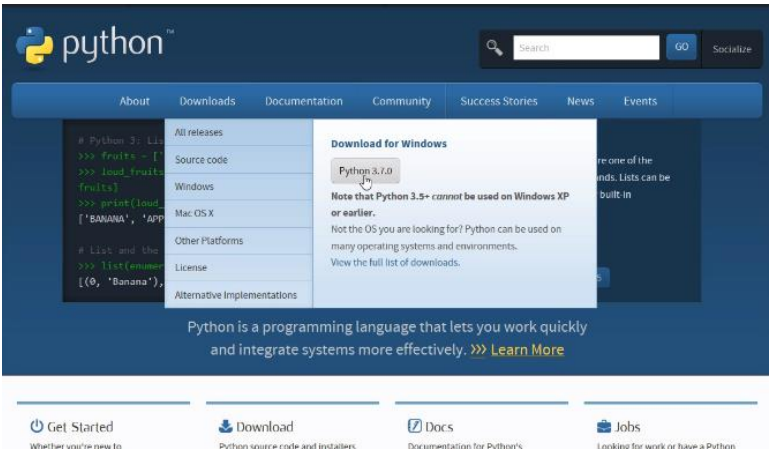


Рис.1. 4. Вікно завантаження Python

По закінченню завантаження установника, запустіть його, щоб почати установку Python. Обов'язково встановіть прапорець на “**Add Python X.Y to PATH**” в майстра налаштування. Переконайтеся, що прапорець “**Add Python X.Y to PATH**” поставлений в установнику, в іншому випадку у вас будуть проблеми з доступом інсталятора Python до командної строки. Натисніть “Встановити”, щоб розпочати встановлення.

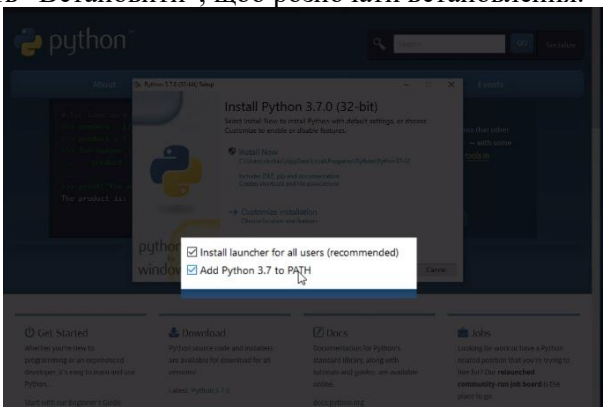


Рис. 1.5. Вікно інсталяції Python

Під час інсталювання вам буде запропоновано обрати наступні компоненти:

– **Register Extensions** дозволяє запускати скрипти **Python (.py)** подвійним клацанням. Рекомендовано, але не обов'язково. (Це налаштування не займає місця на диску, а отже, мало сенсу його видаляти.)

– **Tcl/Tk** - це графічна бібліотека, що використовується середовищем Python, до якої ви звертатиметеся під час виконання робіт

– **Опція документації** встановлює файли довідки, що містять переважну більшість інформації з **docs.python.org**. Рекомендовано, якщо у вас **dialup** або обмежений доступ до інтернету.

– **Utility Scripts** містить скрипт **2to3.py**. Необхідно, якщо ви хочете дізнатися, як переносити наявний код із Python 2 на Python 3. Якщо у вас немає коду на Python 2, можете вилучити цей компонент.

– **Test Suite** - це набір скриптів для тестування самого інтерпретатора Python. Цілком необов'язково.

3. Перевірка установки через менеджер пакетів pip.

По закінченню установки, Python повинен бути встановлений на вашому комп'ютері. Переконаємося в цьому, протестувавши, чи має Python доступ до командної строки (**cmd**) Windows:

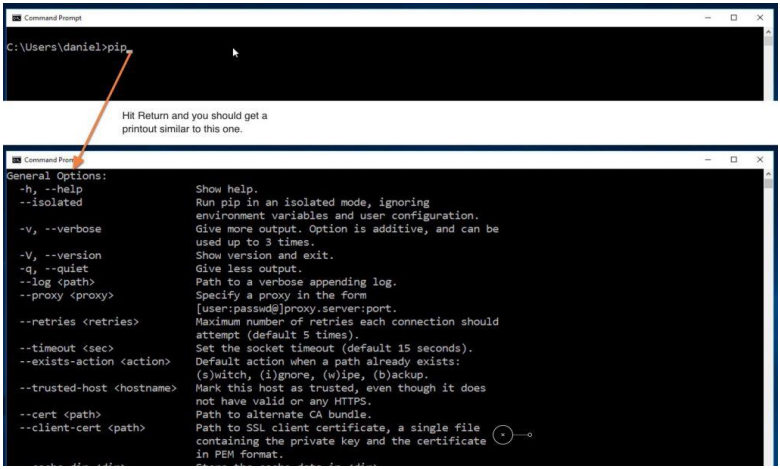


Рис. 1.6. Результат команди “pip”

– Відкрийте командний рядок Windows, запустивши cmd.exe;

– Введіть pip і натисніть “ENTER”;

– Ви побачите довідковий текст менеджера пакетів Python під назвою “pip”. Якщо ви отримаєте повідомлення про помилку, повторіть кроки установки Python, і переконайтеся в тому, що ви маєте робочою версією Python. Велика частина проблем, з якими ви можете зіткнутися, матимуть те чи інше відношення до неправильного налаштуванні PATH. Перевстановлення та підтвердження того, що опція “**Add Python to PATH**” була активована, повинні виправити цю проблему.

4. Запуск інтегрованого середовища розробки IDLE.

Для подальшої роботи відкриваємо IDLE (середовище розробки на мові Python, що поставляється разом з дистрибутивом). IDLE за замовчуванням запускається в інтерактивному режимі.

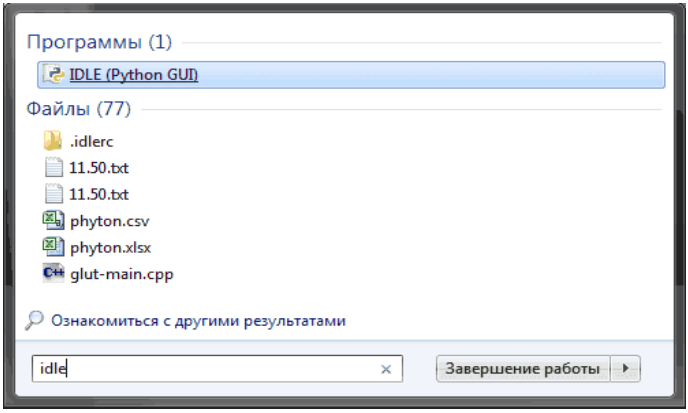


Рис. 1.7. Запуск IDLE через пошук в Windows

Середовище Python - те місце, де можна досліджувати синтаксис Python, отримувати інтерактивну допомогу по командах і зневаджувати короткі програми. Графічне середовище Python (IDLE) також містить і непоганий текстовий редактор, що

підтримує забарвлення синтаксичних елементів Python і безпосередньо поєднано з інтерактивним середовищем.

5. Створення першої програми в інтерактивному режимі.

Для початку попрацюємо в інтерактивні оболочці IDLE.

5.1. Запустіть дистрибутив, в вікні середовища Python зявиться три праві кутові дужки, **>>>,** це запрошення середовища Python. Його не треба набирати. Воно лише означає, що поданий приклад слід вводити у середовищі Python:

```
>>>
```

В основному інтерпретатор виконує команди порядково: пишемо рядок, натискаємо “ENTER”, інтерпретатор виконує її, спостерігаємо результат.

Це дуже зручно, коли людина тільки вивчає програмування або тестує якусь невелику частину коду. Адже якщо працювати на компільованій мові, то довелося б спочатку написати код на вихідній мові програмування, потім скомпілювати і вже потім запустити виконуваний файл на виконання.

5.2. Перш за все, інтерактивне середовище IDLE - чудовий ігровий майданчик для випробування Python. Введіть код:

```
>>> 1 + 1
2
```

де **>>>** - запрошення середовища Python;

1 + 1 - та частина, яку потрібно набирати;

2 - результат обчислення цього виразу.

Як виявилося, ***1 + 1*** є коректним виразом на мові Python, результатом якого, є ***2***.

Спробуйте ще одне:

```
>>>2+5
7
>>>3*(5-8)
-9
>>>2.4+3.0/2
3.9
```

Прокручувати список раніше введених команд можна за допомогою комбінацій Alt + N, Alt + P.

Ви можете набрати будь-який коректний вираз або команду Python в інтерактивному середовищі. Найгірше, що може трапитися - ви отримаєте повідомлення про помилку. Команди виконуються одразу ж (як тільки ви натиснете “ENTER”), значення виразів обчислюються теж одразу, а середовище Python, як калькулятор показує вам результат.

5.3. Першою програмою буде “Hello world”. Для цього на Python достатньо всього одного рядка:

print("Hello world!")

Вводимо цей код в IDLE і натискаємо “ENTER”,результат показано на рис. 1.8.

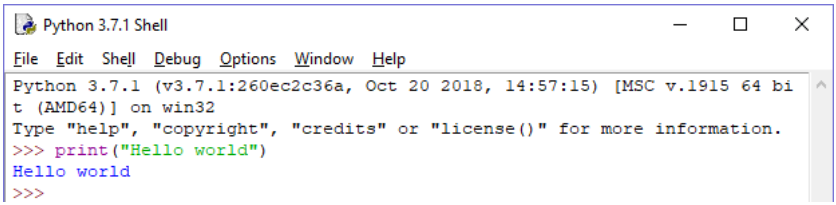


Рис.1.8. Результат роботи програми в IDLE

6. Зміна режиму IDLE та збереження програми.

Інтерактивний режим не є основним при розробці програм. Здебільшого, ви будете зберігати програмний код в файл і запускати вже файл.

6.1. Незважаючи на зручності інтерактивного режиму роботи при написанні програм на Python, зазвичай потрібно зберігати вихідний програмний код для подальшого використання. У такому випадку підготовлюються файли, які передаються потім інтерпретатору на виконання. По відношенню до інтерпретованих мов програмування часто вихідний код називають **скриптом**. Файли з кодом на Python зазвичай мають розширення***.py.** Для того, щоб створити нове вікно, в інтерактивному режимі IDLE виберіть File → New File (або натисніть Ctrl + N). Відкриється нове вікно. Потім бажано відразу зберегти файл під ім’ям «Lr1_6.1_Призвіще.py» (не забуваємо про розширення ***.py**). Після того як код буде підготовлено, знову збережіть файл (щоб оновити збереження). Ну і нарешті, можна запустити скрипт, виконавши команду меню Run → Run Module

(F5). Після цього в першому вікні з'явиться результат виконання коду.

Примітка: якщо набирати код, не зберігши спочатку файл, то підсвічування синтаксису буде відсутнє.

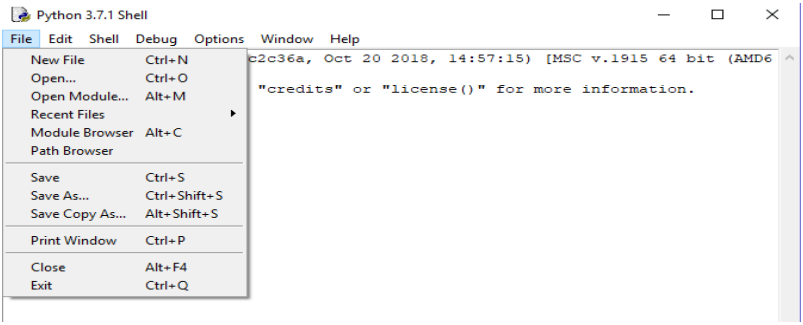


Рис. 1.9. Контекстне меню IDLE

6.2. Підготуйте скрипт. Запустіть його на виконання. У вікні, введіть наступний код:

```
name = input("Як Вас звати? ")
print("Привіт,", name)
```

Перший рядок друкує питання (“Як Вас звати?”), Очікує, поки ви не надрукуєте що-небудь і не натиснете “ENTER”, після чого зберігає введене значення в змінній **name**.

У другому рядку ми використовуємо функцію **print** для виведення тексту на екран, в даному випадку для виведення “Привіт,” і того, що зберігається в змінній “**name**”.

Тепер необхідно натиснути F5 (або вобрати в меню IDLE Run → Run Module) і переконатися в правильності роботи програми. Перед запуском IDLE запропонує зберегти файл. Після чого програма запуститься.

Результати повинен бути таким, як на рис. 1.10. (зліва - результат її роботи, праворуч - файл з написаної вами програмою):

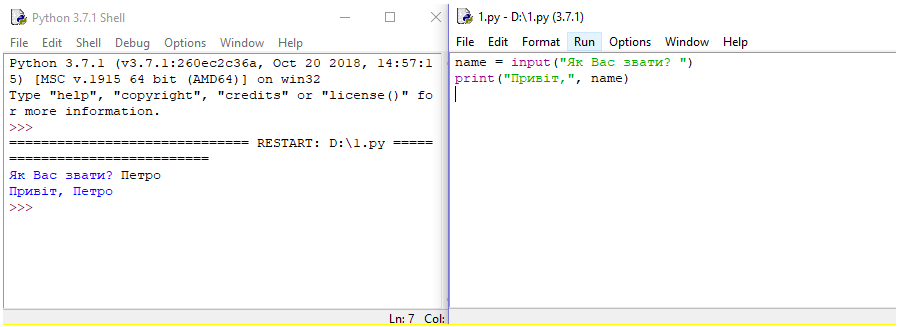


Рис. 1.10. Результат роботи збереженої програми

6.3. *Друге*, що ми вивчимо - це змінну і оператор присвоювання. (Пишемо нову програму):

```
message = ‘Hello, World!’
print(message)
```

Змінна - це величина, що має ім'я, тип і значення. Значення змінної можна змінювати під час роботи програми. У програмі ми створили змінну з ім'ям ***message***, привласнили їй значення-рядок *'Hello, World!*', І, отже, ця змінна прийняла строковий тип.

Знак ***«=»*** - це оператор присвоювання.

Імена змінних можуть складатися з:

– Латинські букви (рядкові і великі літери розрізняються!);

– Російські літери (не рекомендується);

– Цифри (ім'я не може починатися з цифри і складатися тільки з цифр);

– Знак підкреслення _.

Не можна використовувати в іменах змінних:

– Пробіли;

– Знаки +, -,>, <, =, (),! та ін.

– Ключові слова мови Python.

Ключові слова - це слова мови програмування, які мають спеціальне, раз і назавжди закріплене за ними значення. До них відносяться імена функцій, операторів і інше. Наприклад (рис.10), функція “**print**” - ключове слово, яке не можна використовувати в якості імені змінної. Пізніше ми вивчимо і інші функції.

Функція ***print()*** входить у стандартну бібілотеку Python і виводить інформацію, вказану в дужках, на екран або записує у файл.

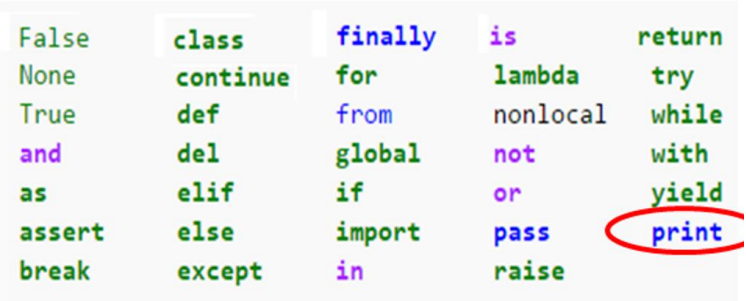


Рис. 10. Ключові слова мови Python

6.4. Продовжимо знайомства з математичними операціями. (Створюємо новий файл “Lr1_6.4_Призвіще.py”).

Створимо дві цілочисельні змінні і попросимо комп'ютер їх скласти.

```
a = 78001457
b = 2546880
c = a + b
print(c)
```

Змінній *c* можна привласнити цілий математичний вираз:

```
c = (a-b) * (a + b) / 27
```

Таблиця 1.2. - *Математичні операції мови Python*

`x + y`	Додавання
`x - y`	Віднімання
`x * y`	Множення
`x / y`	Ділення
`x // y`	Отримання цілої частини від ділення

`x % y`	Остача від ділення
`-x`	Зміна знака числа
`abs(x)`	Модуль числа
`divmod(x, y)`	Пара (`x // y, x % y`)
`x ** y`	Піднесення до ступеня

6.5. Функція вводу. Для того щоб призначити змінній значення, введене з клавіатури, використовується функція ***input ()***. Напишемо і запустимо наступну програму:

name = input("Введите своё имя: ")
print("Привет, ", name)

Змініть програму так, щоб вона виводила в кінці знак оклику.

Введення строки: *s = input("Введите строку: ")*

"Введите строку: "- звернення до користувача (не обов'язково, але дуже бажано)

За замовчуванням всі введені дані інтерпретатор Пітона розуміє, як рядки, тому, якщо ми хочемо отримати число, то рядок доведеться перетворити в число.

Перетворення до цілочисельного типу і введення цілого числа:

n = int(input("Введите число: "))

Тобто на функцію введення ми навішуємо ще одну функцію перетворення в ціле число.

Функція перетворення до цілочисельного типу:

n =int(s)

Функція перетворення до строкового типу:

s = str(n)

Функція генерації випадкового цілого числа з відрізка [x,y]:

import random
a = random.randint(x,y)

Завдання. Напишіть програму, яка отримує на вхід два числа і виводить їх суму:

```
a = input("Введите число a: ")
b = input("Введите число b: ")
sum = a+b
print("a+b= ", sum)
```

Програма працює не правильно, тому що всі введені дані комп'ютером розуміються як рядки, щоб програма працювала правильно треба виправити в:

```
a = int(input("Введите число a: "))
b = int(input("Введите число b: "))
sum = a+b
print("a+b= ", sum)
```

7. Завдання до лабораторної роботи

1) У кожному рядку визначити тип і значення змінної:

```
a = 5
n = input()        #пользователь вводит цифру 8
c = int(n)
d = a*c
d = d–a
s = "Рамамбахарумамбуру"
d = n+a
m = n+s
```

2) Вивести на екран три введених з клавіатури числа в порядку, зворотному їх введення.

3) Ввести з клавіатури два числа і вивести цілу частину від ділення першого на друге.

4) Ввести з клавіатури основа і висоту трикутника і вивести площу трикутника.

5) Ввести з клавіатури два катета трикутника і вивести гипотенузу. (Квадратний корінь - це зведення в ступінь (1/2))

6) Згенерувати випадкове двозначне число, вивести на екран це число, а також суму і добуток його цифр.

Для отримання чисел використовуйте цілочисельне ділення на 10 і взяття залишку від ділення на 10.

Приклад для числа 47:

47 // 10 = 4

47% 10 = 7

7) Ввести основи і висоту трапеції і вивести площу трапеції.

8) Отримати випадкове тризначне число, вивести це число і суму його окремих цифр.

9) Програма, яка розраховує вік людини в годинах.

Контрольні питання

1. Які інтегровані середовища програмування для Python використовують для написання програм?
2. Запуск програм на Python в терміналі Windows.
3. Що виконує функція **print()**?
4. Як у режимі інтерактивного інтерпретатора команди вводяться у термінальному вікні?
5. Які текстові редактори використовують у Python для написання програм?
6. Для яких цілей використовується Python?
7. Як перейти в режим інтерактивного інтерпретатора?
8. Яке розширення мають файли із програмами, написаними на мові Python?
9. Як запустити програму, що міститься у файлі, на виконання у термінальному вікні?
10. У яких випадках, під час виконання програми, може з'явитися повідомлення Traceback (most recent call last)?
11. Який символ використовується у мові Python для позначення коментарів?

Зміст звіту

1. Мета роботи.
2. Завдання до роботи.
3. Відповіді на контрольні питання.
4. Текст розробленого програмного забезпечення.
5. Результати тестування: вхідні дані та результати роботи програми.
6. Висновки, що відображають особисто отримані результати виконання роботи, їх критичний аналіз.

До звіту додаються файли проекту.

ЛАБОРАТОРНА РОБОТА №2

Тема: Типи даних в програмуванні. Визначення змінної

Мета: ознайомитися з типами даних в програмуванні, правилами перетворення цілих та дійсних чисел з однієї системи числення в іншу.

Час виконання роботи: 4 години.

Програмне забезпечення: IDLE (Python 3.X).

Короткі теоретичні відомості

Типи даних (об'єктів) мови Python

Програми на Python виконують дії, які приймають форму операцій над об'єктами (наприклад, складання або конкатенація), над якими виконуються операції. Дані в Python подані у формі вбудованих об'єктів або об'єктів, які створюють із застосуванням конструкцій мови Python або інших інструментів (наприклад, бібліотеки розширень, написані на мові С).

Об'єктами є всі дані, які доводиться обробляти в програмах. Об'єкти (структури даних, призначені для подання складових предметної області) − це область пам'яті зі значеннями і асоційованими з ними наборами операцій.

В Python використовують динамічну типізацію (типи даних визначаються автоматично, їх не потрібно оголошувати в програмному коді), але при цьому він є мовою із чіткою типізацією (ви зможете виконувати над об'єктом тільки ті операції, які застосовані до його типу). Програми на Python можна розкласти на такі складові: модулі, інструкції, вирази і об'єкти. При цьому програми діляться на модулі, модулі містять інструкції, інструкції складаються з виразів, вирази створюють і обробляють об'єкти.

Можна створювати власні типи об'єктів: вбудовані компоненти є стандартними складовими Python, тому вони завжди залишаються незмінними; власні структури мають властивість змінюватися від випадку до випадку. Навіть якщо ви створюєте власні типи об'єктів, вбудовані об'єкти будуть ядром будь−якої програми на Python. У табл. 2.1 наведено деякі вбудовані типи об'єктів і синтаксичні конструкції використання цих об'єктів у вигляді літералів – виразів, які генерують (створюють) ці об'єкти. Типи об'єктів, перераховані в табл. 2.1,

називають базовими, тому що вони вбудовані безпосередньо в мову (для створення більшості з них використовується певний синтаксис). Наприклад, коли виконується програмний код: 'spam', то виконується вираз − літерал, який генерує і повертає новий об'єкт−рядок.

Таблиця 2.1. - *Деякі вбудовані об'єкти*

Тип об'єкта	Приклад літерала
Числа	1234, 3.1415, 3+4j, Decimal, Fraction
Рядки	_spam‘, ―guido‘s‖ , b‘a\x01c‘
Списки	[1, [2, _three‘], 4]
Словники	{_food‘: _spam‘, _taste‘: _yum‘}
Кортежі	(1,‘spam‘, 4, _U‘)
Файли	myfile = open(_eggs‘, _r‘)
Множини	set(_abc‘), {_a‘, _b‘, _c‘}

Схожим чином вираз, розміщений в квадратних дужках, створює список, фігурних дужках - словник тощо. Хоча в Python відсутня конструкція оголошення типу, сам синтаксис виконуваних виразів задає типи створюваних і використовуваних об'єктів. Розглянемо базовий тип об'єктів Python - числа.

Python підтримує звичайні числові типи (цілі і дійсні), а також літерали – для їх створення і вирази – для їх обробки. Нижче наведений повний перелік числових типів та інструментів, які підтримуються в Python:

1. Цілі і дійсні числа
2. Комплексні числа
3. Числа фіксованої точності
4. Раціональні числа
5. Множини
6. Логічні значення
7. Цілі числа необмеженої точності
8. Вбудовані функції, модулі для роботи з числами.

Крім базових типів даних Python використовує звичайні числові типи: цілі числа (додатні та від‘ємні) і дійсні числа (з дробовою частиною), які іноді називають числами з плаваючою точкою. Python дозволяє записувати цілі числа у вигляді шістнадцяткових, вісімкових і двійкових літералів. Він підтримує

комплексні числа і забезпечує необмежену точність подання цілих чисел (кількість цифр в цілих числах обмежується лише обсягом наявної пам'яті). У табл. 2.2 показано, як виглядають числа різних типів в Python в тексті програми (тобто у вигляді літералів).

Таблиця 2.2. - *Числові літерали*

Літерал	Інтерпретація
1234, -24, 0, 9999999999999999999	Звичайні цілі числа (із необмеженою точністю зображення)
1.23, 1., 3.14e-10, 4E210, 4.0e+210	Дійсні числа
0o177, 0x9ff, 0b101010	Вісімкові, шістнадцяткові та двійкові літерали цілих чисел
3+4j, 3.0+4.0j, 3J	Літерали комплексних чисел

Операції. Можна сказати, що операція - це виконання якихось дій над даними (*операндами*). Для виконання конкретних дій потрібні спеціальні інструменти — *оператори*.

Наприклад, символ "+" по відношенню до чисел виконує операцію додавання, а по відношенню до рядків - конкатенацію (з'єднання). Парний знак ****** зводить перше число в ступінь другого.

Вираз	Значення
34.907 + 320.65	355.55699999999996
“Hi,” + "world :)"	'Hi, world :)'
“Hi,” * 10	'Hi, Hi, Hi, Hi, Hi, Hi, Hi, Hi, Hi, Hi, '

Зміна типу даних

Що буде, якщо ми спробуємо виконати в одному виразі операцію над різними типами даними? Наприклад, скласти ціле і дробове число, число і рядок. Однозначну відповідь дати не можна: так, при складанні цілого числа і числа з плаваючою

крапкою, виходить число з плаваючою крапкою, а якщо спробувати скласти будь-яке число і рядок, то інтерпретатор Python видасть помилку.

Вираз	**Результат виконання**
1 + 0.65	1.65
"Hi," + 15	П о м и л к а

Однак, бувають випадки, коли програма отримує дані у вигляді рядків, а оперувати повинна числами (або навпаки). У такому випадку використовуються спеціальні функції (особливі оператори), що дозволяють перетворити один тип даних в іншій. Так функція **int ()** перетворює переданий їй рядок (або число з плаваючою крапкою) в ціле число, функція **str ()** перетворює переданий їй аргумент в рядок, **float ()** - в дробове число.

Вираз	**Результат виконання**
int ("56")	56
int (4.03)	4
int ("comp 486")	П о м и л к а
str (56)	'56 '
str (4.03)	'4.03 '
float (56)	56.0
float ("56")	56.0

Змінні

Дані зберігаються в комірках пам'яті комп'ютера. Коли ми вводимо число, воно поміщається в пам'ять. Але як дізнатися, куди саме? Як надалі звертатися до цих даних? Раніше, при написанні програм на машинній мові, звернення до комірок пам'яті здійснювали за допомогою вказівки регістрів. Але вже з появою асемблерів, при зверненні до даних стали використовувати так звані змінні. Механізм зв'язку між змінними та даними може розрізнятися в залежності від мови програмування і типу даних. Поки досить запам'ятати, що дані

зв'язуються з яких-небудь ім'ям і надалі звернення до них можливо з цього імені.

У програмі на мові Python зв'язок між даними і змінними встановлюється за допомогою знака =. Така операція називається *присвоєнням*. Наприклад, вираз **a = 4** означає, що на об'єкт (дані) у певній області пам'яті посилається ім'я **a** і звертатися до них тепер слід за цим іменем.

Імена змінних можуть бути будь-якими. Однак є кілька загальних правил їх написання:

1. Бажано давати змінним осмислені імена, що говорять про призначення даних, на які вони посилаються.

2. Ім'я змінної не повинно збігатися з командами мови (зарезервованими ключовими словами).

3. Ім'я змінної має починатися з букви або символу підкреслення (_).

Щоб дізнатися значення, на яке посилається змінна, перебуваючи в режимі інтерпретатора, достатньо її викликати (написати ім'я і натиснути **Enter**).

Приклад роботи зі змінними в інтерактивному режимі:

```
>>> apples = 100
>>> eat_day = 5
>>> day = 7
>>> apples = apples - eat_day * day
>>> apples
65
```

Математичні функції

Python має багато математичних функцій стандартної бібліотеки **math.** Щоб отримати до них доступ зі своїх програм, необхідно ввести ***import math.***

Ця бібліотека містить такі константи, як **pi** (число Пі) і **e** (експонента):

```
>>> import math
>>> math.pi
3.141592653589793
>>> math.e
2.718281828459045
```

Розглянемо кілька корисних функцій бібліотеки **math**:

```
>>> math.fabs(-221.1)  -Повертає абсолютне значення.
221.1
>>> math.floor(56.8)  -          Округлення вниз.
56
>>> math.ceil(38.3) -        Округлення вгору.
39
>>> math.factorial(7) -     Обчислення факторіалу.
5040
>>> math.pow(4, 3) -         Піднесення числа до степеня.
64.0
>>> math.sqrt(256) - Обчислення кореня квадратного з числа.
16.0
>>> math.radians(180) -  Перетворення значення в градусах
3.141592653589793             у радіани
>>> math.degrees(math.pi) - Перетворення значення в
180.0                              радіанах у градусну міру
```

В цій бібліотеці також присутні тригонометричні функції: **sin()**, **cos()**, **tan()**, **asin()**, **acos()**, **atan()**.

Система числення

Система числення – це сукупність прийомів та правил зображення чисел, за допомогою символів (цифр, літер тощо), які мають визначені кількісні значення (числовий еквівалент).

Залежно від способів зображення чисел цифрами та способу визначення числового еквіваленту с/ч поділяють на непозиційні та позиційні.

***Непозиційна* с/ч** (наприклад, римська) – це така система, в якій значення символу (значення числового еквіваленту) не залежить від його позиції (розряду) у запису числа, а залежить лише від самого числа.

***Позиційна* с/ч** – це така система, в якій значення символу (числовий еквівалент) залежить від його місця у запису числа.

Будь–яка позиційна с/ч характеризується *основою.* Основа (або базис p) позиційної с/ч – це ціле число, яке визначає кількість символів, що в ній використовують для зображення числа.

В позиційній с/ч з основою p будь-яке число R можна подати у вигляді:

$$R(p) = a_N p^N + a_{N-1} p^{N-1} + \ldots + a_1 p^1 + a_0 p^0 + a_{-1} p^{-1} + \ldots + a_{-k}\, p^{-k}, \quad (1)$$

де коефіцієнти a_i – символи в зображенні числа R, які приймають значення від 0 до $p-1$. Зазвичай число $R(p)$ зображують у вигляді множини коефіцієнтів a_i: $R(p) = a_N a_{N-1} \ldots a_1 a_0 a_{-1} \ldots a_{-k}$.

Наприклад:

$$265{,}23_{10} = 2\cdot 10^2 + 6\cdot 10^1 + 5\cdot 10^0 + 2\cdot 10^{-1} + 3\cdot 10^{-2},$$

$$23{,}78_{16} = 2\cdot 16^1 + 3\cdot 16^0 + 7\cdot 16^{-1} + 8\cdot 16^{-2},$$

$$10001{,}01_2 = 1\cdot 2^4 + 0\cdot 2^3 + 0\cdot 2^2 + 0\cdot 2^1 + 1\cdot 2^0 + 0\cdot 2^{-1} + 1\cdot 2^{-2}.$$

У обчислювальній техніці широко використовують позиційні с/ч (2–ову, 8–ову, 10–ову, 16–ову).

Двійкова система числення – тут використовують дві цифри «0» і «1».

Приклад 2.1. $2_{10} = 2\cdot 10^0 = 1\cdot 2^1 + 0\cdot 2^0 = 10_2$.

Двійкова таблиця додавання

+	0	1
0	0	1
1	1	10

Вісімкова с/ч — тут використовують цифри 0,1,2,3,4,5,6,7.

Приклад 2.2. $8_{10} = 8\cdot 10^0 = 1\cdot 8^1 + 0\cdot 8^0 = 10_8$.

Вісімкова таблиця додавання

+	1	2	3	4	5	6	7
1	2	3	4	5	6	7	10
2	3	4	5	6	7	10	11
3	4	5	6	7	10	11	12
4	5	6	7	10	11	12	13
5	6	7	10	11	12	13	14
6	7	10	11	12	13	14	15
7	10	11	12	13	14	15	16

***Шістнадцяткова* с/ч** – тут використовують цифри 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 і літери A, B, C, D, E, F.

Приклад 2.3. $16_{10}=16\cdot10^0=1\cdot16^1+0\cdot16^0=10_{16}$

Шістнадяткова таблиця додавання

+	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	10
2	3	4	5	6	7	8	9	A	B	C	D	E	F	10	11
3	4	5	6	7	8	9	A	B	C	D	E	F	10	11	12
4	5	6	7	8	9	A	B	C	D	E	F	10	11	12	13
5	6	7	8	9	A	B	C	D	E	F	10	11	12	13	14
6	7	8	9	A	B	C	D	E	F	10	11	12	13	14	15
7	8	9	A	B	C	D	E	F	10	11	12	13	14	15	16
8	9	A	B	C	D	E	F	10	11	12	13	14	15	16	17
9	A	B	C	D	E	F	10	11	12	13	14	15	16	17	18
A	B	C	D	E	F	10	11	12	13	14	15	16	17	18	19
B	C	D	E	F	10	11	12	13	14	15	16	17	18	19	1A
C	D	E	F	10	11	12	13	14	15	16	17	18	19	1A	1B
D	E	F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C
E	F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D
F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E

Множина цілих чисел, розміщених в пам'яті ЕОМ, обмежена і залежить від розміру комірок пам'яті (машинного слова), які використовують для їхнього зберігання.

В *k*-розрядній комірці може зберігатися 2^k різних значень цілих чисел. Зазвичай розряди нумерують справа наліво, починаючи з «0». Нижче показана нумерація біт в двобайтовому машинному слові.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

У даній комірці можна зберігати 2^{16} цілих чисел. Діапазон допустимих значень залежить також від того, чи будуть в комірці зберігатися числа зі знаком чи без знаку. Для запису цілого числа без знаку досить перевести його в двійкову с/ч і при необхідності доповнити результат зліва незначущими нулями.

Приклад 2.4. Отримати внутрішнє машинне зображення цілого числа 129 без знаку в однобайтовій комірці пам'яті.

Розв'язання: $129_{10}=10000001_2$; 1000 0001

Приклад 2.5. Нехай задано внутрішнє машинне подання числа в однобайтовому машинному слові – $9C_{16}$. Визначити, що це за число.

Розв'язання: $9C_{16} = 1001\ 1100_2 = 9 \cdot 16^1 + 12 \cdot 16^0 = 156_{10}$

$2^7+2^4+2^3+2^2=128+16+8+4 = 156_{10}$

Щоб отримати внутрішнє машинне подання цілого числа зі знаком, необхідно знайти його код доповнення.

Подання цілих додатніх чисел

Для їх отримання необхідно виконати такі кроки:

Крок 1. Перевести число *N* в двійкову систему числення.

Крок 2. Отриманий результат доповнити зліва незначущими нулями до k розрядів.

Крок 3. При необхідності перевести число в стислу 16-ову форму.

Приклад 2.6. Отримати внутрішнє машинне зображення цілого числа 1607 в двобайтовій комірці пам'яті. Записати відповідь в 16-овій формі.

Розв'язання: $1607_{10} = 11001000111_2$. Внутрішнє машинне зображення цього числа: 0000 0110 0100 0111; 16-а форма: 0647.

Подання цілих від'ємних чисел

Для подання від'ємних чисел використовується код доповнення, який дозволяє замінити арифметичну операцію віднімання операцією додавання, що спрощує роботу процесора і збільшує його швидкодію. Алгоритм отримання внутрішнього подання цілого від'ємного числа *N*, що зберігається в k-розрядному машинному слові охоплює такі кроки:

Крок 1. Отримати внутрішнє зображення додатнього числа N.

Крок 2. Отримати зворотний код цього числа заміною «0» на «1» і «1» на «0», тобто значення всіх біт інвертувати.

Крок 3. До отриманого числа додати «1» (отримати код доповнення).

Крок 4. При необхідності записати стисле внутрішнє машинне подання.

Приклад 2.7. Отримати внутрішнє подання цілого числа – 1607 в 2-х байтовій комірці пам'яті. Записати відповідь в 16–овій формі.

Розв'язання. Внутрішнє подання цього додатнього числа:
0000 0110 0100 0111
Зворотний код – 1111 1001 1011 1000
Код доповнення – 1111 1001 1011 1001
Стислий 16-ий код – $F9B9_{16}$

У разі подання величини зі знаком найлівіший (старший) розряд вказує на додатне число, якщо містить нуль, і на від'ємне, якщо – одиницю.

Приклад 2.8. Нехай задано стисле 16-е внутрішнє подання числа – $CF18_{16}$. Визначити, що це за число.

Розв'язання: CF18 = 1100 1111 0001 1000

Число від'ємне, оскільки старший розряд дорівнює «1», тому отримуємо зворотний код – 1100 1111 0001 0111 (відняти 1)

Прямий код – 0011 0000 1110 1000; $30E8_{16}= 12520_{10}$

Перетворення чисел з однієї системи числення в іншу

Правило 1. **Переведення змішаного числа** (числа із дробовою частиною) з *p*–ої с/ч в *q*–у с/ч, коли має місце співвідношення $p=q^k$ (*k* – ціле додатне число), здійснюється поразрядно. Кожна *p*–а цифра замінюється рівним їй *k*–розрядним числом, записаним в *q*–ій с/ч (при цьому можна використовувати дані з табл. 2.3).

Приклад

а) $A61,87C_{(16)} = \underbrace{(1010)(0110)(0001)}_{A_{(16)}\ 6_{(16)}\ 1_{(16)}}, \underbrace{(1000)(0111)(1100)}_{8_{(16)}\ 7_{(16)}\ C_{(16)}}{}_{(2)}$

б) $31,471_{(8)} = \underbrace{(011)(001)}_{3_{(8)}\ 1_{(8)}}, \underbrace{(100)(111)(001)}_{4_{(8)}\ 7_{(8)}\ 1_{(8)}}{}_{(2)}$

Таблиця 2.3. - *Запис чисел у різних с/ч*

Десяткове число	Двійкове число	Вісімкове число	Шістнадцяткове число
0	0000	0	0
1	0001	1	1
2	0010	2	2
3	0011	3	3

Десяткове	Двійкове	Вісімкове	Шістнадцяткове
4	0100	4	4
5	0101	5	5
6	0110	6	6
7	0111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

Зворотній переклад з *q*–ої с/ч в *p*–у с/ч здійснюють, розбиваючи *q*–ий запис числа на групи по *k* цифр, рухаючись від коми вправо і вліво і замінюючи кожну групу цифр її *p*–им зображенням. При цьому якщо крайні групи виявляться неповними, то їх доповнюють до *k* цифр незначущими нулями.

Приклад

а) $1110010{,}01101111001_{(2)} = \underbrace{(0111)}_{7_{(16)}}\underbrace{(0010)}_{2_{(16)}}{,}\underbrace{(0110)}_{6_{(16)}}\underbrace{(1111)}_{F_{(16)}}\underbrace{(0010)}_{2_{(16)}}{}_{(2)} = 72{,}6F2_{(16)}$;

б) $1001111{,}011100_{(2)} = \underbrace{(001)}_{1_{(8)}}\underbrace{(001)}_{1_{(8)}}\underbrace{(111)}_{7_{(8)}}{,}\underbrace{(011)}_{3_{(8)}}\underbrace{(100)}_{4_{(8)}}{}_{(2)} = 117{,}34_{(8)}$.

Переведення змішаного числа виконується окремо для цілої і дробової частин числа.

Правило 2. Перетворення цілої частини. Цілу частину числа, записану в *p*–ій с/ч, ділять на основу *q*–ої с/ч, записану в *p*–ій с/ч (всі операції виконуються за правилами *p*–ої с/ч). Отримане в залишку число є молодшою (останньої) цифрою в *q*–ій формі запису числа. Отриману частку знову ділять на основу *q*; залишок – це передостання цифра в шуканій формі запису числа; і т.д. Операцію ділення проводять до тих пір, поки часткою не стане число, менше за *q*; ця частка – це старша (перша) цифра в *q*–ій формі запису числа.

Приклад.

а) $345_{(10)} = 159_{(16)}$;

$$\begin{array}{r|l} 345 & 16 \\ \underline{32} & 21 \\ 25 & \underline{16} \\ \underline{16} & 1 \\ 9 & \end{array} \qquad \begin{array}{r|l} 21 & 16 \\ \underline{16} & 1 \\ 5 & \end{array}$$

б) $24_{(10)} = 11000_{(2)}$.

$$\begin{array}{r|l} 24 & 2 \\ \underline{24} & 12 \\ 0 & \underline{12} \\ & 0 \\ & \end{array} \qquad \begin{array}{r|l} 12 & 2 \\ \underline{12} & 6 \\ 0 & \underline{6} \\ & 0 \\ & \end{array} \qquad \begin{array}{r|l} 6 & 2 \\ \underline{6} & 3 \\ 0 & \underline{2} \\ & 1 \\ & \end{array}$$

Перетворення дробової частини. Дробову частину числа, записану в *p*–ій с/ч, множать на основу *q*–ої с/ч, записану в *p*–ій с/ч. Ціла частина добутку буде старшою цифрою зображення дробу (першої після коми) в *q*–ій формі запису дробу. Дробову частину добутку знову множать на основу *q*; ціла частина – наступна цифра після коми в *q*–ій формі запису дробу. Процес продовжують, поки дробова частина не стане нулем або буде отримано необхідну кількість знаків після коми в дробовій частині запису числа. Цілі частини записують в *q*–ій с/ч.

Приклад.

а) $0{,}2_{(10)} = 0{,}(0011)_{(2)}$;

$$\begin{array}{r} 0{,}2 \\ \underline{\times\ 2} \\ 0_{(2)} = 0{,}4 \\ \underline{\times\ 2} \\ 0_{(2)} = 0{,}8 \\ \underline{\times\ 2} \\ 1_{(2)} = 1{,}6 \\ \underline{\times\ 2} \\ 1_{(2)} = 1{,}2 \\ \underline{\times\ 2} \\ 0_{(2)} = 0{,}4 \\ \underline{\times\ 2} \\ 0_{(2)} = 0{,}8 \end{array}$$

б) $0{,}12_{(10)} = 0{,}1EB_{(16)}$;

$$\begin{array}{r} 0{,}12 \\ \underline{\times\ 16} \\ 1_{(16)} = 1{,}92 \\ 0{,}92 \\ \underline{\times\ 16} \\ E_{(16)} = 14{,}72 \\ 0{,}72 \\ \underline{\times\ 16} \\ B_{(16)} = 11{,}52 \\ \cdots\cdots \end{array}$$

в) $0{,}A_{(16)} = 0{,}625_{(10)}$.

$$\begin{array}{r} 0{,}A \\ \underline{\times\ A} \\ 6_{(10)} = 6{,}4 \\ 0{,}4 \\ \underline{\times\ A} \\ 2_{(10)} = 2{,}8 \\ 0{,}8 \\ \underline{\times\ A} \\ 5_{(10)} = 5{,}0 \end{array}$$

При переведенні змішаного числа результати для цілої та дробової частин, отримані за правилами 2, записують, відокремлюючи комою одну частину від іншої.

Приклад.

$345{,}02_{(10)} = 101011001{,}(0011)_{(2)}$.

$345_{(10)} = 159_{(16)} = \underbrace{(0001)}_{1_{(16)}}\underbrace{(0101)}_{5_{(16)}}\underbrace{(1001)}_{9_{(16)}}{}_{(2)}$;

$0{,}2_{(10)} = 0{,}(0011)_{(2)}$.

$345 \mid 16$
$32 \quad 21 \mid 16$
$25 \quad 16 \quad 1$
$16 \quad 5$
9

$0{,}2$
$\underline{\times\ 2}$
$0_{(2)} = 0{,}4$
$\underline{\times\ 2}$
$0_{(2)} = 0{,}8$
$\underline{\times\ 2}$
$1_{(2)} = 1{,}6$
$\underline{\times\ 2}$
$1_{(2)} = 1{,}2$
$\underline{\times\ 2}$
$0_{(2)} = 0{,}4$
.....

Правило 3. Переведення чисел в 10−ову с/ч рекомендують виконувати додаванням з урахування "ваги" цифри у числі за формулою (1):

$$a_N a_{N-1} \dots a_1 a_0 \,.\, a_{-1} \dots a_{-K} = a_N p^N + a_{N-1} p^{N-1} + \dots + a_1 p^1 + a_0 p^0 + a_{-1} p^{-1} + \dots + a_{-K} p^{-K}$$

Приклад.

а) $1101{,}111_{(2)} = 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 + 1 \cdot 2^{-1} + 1 \cdot 2^{-2} + 1 \cdot 2^{-3} = 13{,}875_{(10)}$;

б) $1A{,}02_{(16)} = 1 \cdot 16^1 + 10 \cdot 16^0 + 0 \cdot 16^{-1} + 2 \cdot 16^{-2} = 26{,}0078125_{(10)}$.

Подання інформації в ПЕОМ типу IBM PC/AT

Інформація в пам'яті ЕОМ зберігається і обробляється в двійковому вигляді. Форма запису даних в пам'яті машини називається внутрішнім поданням інформації в ЕОМ. Одиницею зберігання інформації є один біт, тобто двійковий розряд, який може приймати значення «0» або «1». Застосування двійкової с/ч дозволяє використовувати для зберігання інформації елементи, які мають лише два стійких стани. Один стан служить для зображення одиниці відповідного розряду числа, інший − для зображення нуля. Зазвичай біти пам'яті групуються в більш широкі структури. Група з восьми бітів називається байтом. Поля пам'яті ЕОМ мають спеціальні назви: 16 біт − слово, 32 біта −

подвійне слово, 1024 байта (1 К байт) − лист. Всі байти пронумеровані, починаючи з нуля.

Адресою будь-якої інформації вважається адреса (номер) найпершого байта поля пам'яті, виділеного для її зберігання. Існують два основних способи подання чисел: з фіксованою і з плаваючою комою.

Двійкові числа з фіксованою комою. У вигляді з фіксованою комою можуть зберігатися тільки цілі числа (в пам'яті ЕОМ вони записуються в двійковому вигляді). Ціле двійкове число займає в пам'яті 16 або 32 (або 64) двійкових розряди. Це залежить від довжини числа і способу подання змінної.

Розглянемо, як записується число в подвійному слові (в слові воно записується аналогічно). Сьомий біт першого байту є службовим при зберіганні чисел зі знаком, і його вміст не впливає на абсолютну величину числа. Зазначимо, що для додатних чисел вмістом службового біта є нуль. Молодший двійковий розряд числа записується в 0-ий біт, тобто число заповнюють справа наліво. Якщо число додатне, то біти, що залишилися, заповнюються нулями. Наприклад, число 12710 в 4 байтах буде подано таким чином:

$$127_{(10)} = 7F_{(16)} = 1111111_{(2)} = 1 \cdot 2^6 + 1 \cdot 2^5 + \ldots + 1 \cdot 2^1 + 1 \cdot 2^0$$

76543210	76543210	76543210	76543210	біти
0000 0000	0000 0000	0000 0000	01111111	байти
1 – ий байт	2- ий байт	3– ій байт	4– ий байт	

Із наведеного вище подання видно, що в молодшому розряді записаний коефіцієнт при 2^0, в наступному − при 2^1 і т.д. Очевидно, що якщо у всіх бітах з 0-го по 30-ий знаходяться 1, то максимальне ціле додатне число, яке можна записати в подвійному слові, має вигляд:

$01111111\ 11111111\ 11111111\ 11111111 = 2^{31} - 1 = 2147483647_{10}$

Форму запису додатних цілих двійкових чисел називають прямим кодом. У цьому випадку їх запис у відведеній пам'яті повністю відповідає символічному запису в двійковій системі числення.

Від‘ємні числа записуються в додатковому коді. Використання цього коду дозволяє спростити апаратну реалізацію операції віднімання, яка замінюється операцією додавання зменшуваного, поданого в прямому коді, і від'ємника, поданого в додатковому коді. Додатковий код виходить з прямого шляхом інвертування кожного біта (зворотний код) і додавання “1” до молодшого біту числа. Подамо число -95_{10} як двійкове число з фіксованою точкою:

$$-95_{10} = -5F_{16} = -1011111_2$$

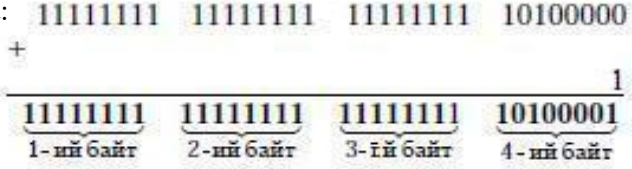
Крок 1. Запишемо прямий код числа в чотири байти:

00000000 00000000 00000000 01011111

Крок 2. Запишемо зворотний код числа:

11111111 11111111 11111111 10100000

Крок 3. Додавши до молодшого біту «1», отримаємо додатковий код числа:



При зберіганні цілих додатних чисел існує можливість майже удвічі збільшити числовий діапазон. Це досягається за рахунок використання службового сьомого біта першого байту для зберігання старшого коефіцієнта в двійковому поданні числа. *Наприклад,* число

$$65535_{10} = FFFF_{16} = 11111111\ 11111111_2,$$

оголошене як ціле без знаку, в двох байтах запишеться у вигляді 11111111 11111111.

Двійкові числа з плаваючою крапкою. Числа, в яких положення крапки не зафіксовано після деякого розряду, а вказується спеціальним числом, називаються числами з плаваючою комою. У загальному вигляді будь−яке число A в системі числення з основою p можна подати у вигляді $A = m \cdot p^k$, де m мантиса числа A, k − порядок числа. При цьому якщо мантиса числа задовольняє нерівності $1 \le |m| < p$, то число нормалізоване.

Розмір областей, які виділяються під числа з плаваючою комою, істотно залежать від апаратної реалізації обчислювальної техніки. Для IBM PC/AT, наприклад, числа з плаваючою комою можуть, в залежності від оголошених атрибутів **float** розташовуватися відповідно в 32 бітах пам'яті у вигляді:

float	±	характеристика		нормалізована мантиса	
біти	31	30	23	22	0

Мантиса числа записується в нормалізованому вигляді (в двійковому зображенні числа перед комою зберігається один значущий двійковий біт). Щоб збільшити діапазон подання чисел в форматі **float,** одиниця перед комою в двійковій нормалізованій формі запису числа в пам'ять не записується. Знак мантиси записується в останньому біті першого байту.

Для спрощення апаратної реалізації арифметичних операцій в поданні числа з плаваючою комою знак порядку явно не зберігається. Замість порядку в пам'ять записується величина, яку називають характеристикою: її отримують шляхом додавання коефіцієнта, який для чисел типу **float** дорівнює $127_{10} = 7F_{16}$.

Подамо число $-18.2_{10} = -10010.(0011)_2$, нормалізований запис якого має вигляд $-1.0010(0011)\cdot 2^{100}$, де нормалізована мантиса $m = -1.0010\ (0011)$, а порядок $k = 4_{10} = 100_2$. Перейдемо від порядку до його характеристики: $4 + 127 = = 131_{10} = 83_{16} = 1000\ 0011_2$. Отримаємо зображення числа -18.2_{10} в форматі **float**:

1	10000011	00100011001100110011010	
31	30	23	22 0

Якщо кількість цифр в числі, яке записують в ПЕОМ або отримано в результаті обчислень, більше виділеного поля пам'яті, то надлишкові цифри відкидаються, а число, що записується в пам'ять, округляється з надлишком.

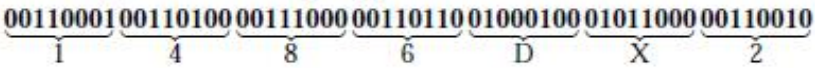
Розмір виділених для зберігання даних областей визначає інтервал допустимих для цього атрибута значень − діапазон (табл. 2.4). Будь-яке число, менше за абсолютною величиною додатного мінімального числа, поданого у відповідному форматі,

буде записано в пам'ять у вигляді нуля. Для заданого формату це так званий машинний нуль. Крім того, числа заданого формату не повинні перевищувати по абсолютній величині максимальне число, подане в цьому форматі. В іншому випадку старші біти числа будуть втрачені, а результат не правильним. Описана ситуація називається переповненням розрядної сітки, а самі числа − машинною нескінченністю.

Таблиця 2.4. - *Інтервал допустимих для атрибута значень*

Тип даних	Кіл-ть біт	Діапазон	
		Abs(min)	Abs(max)
float	32	$3.4 \cdot 10^{-38}$	$3.4 \cdot 10^{38}$

Символьна інформація. В ПЕОМ для внутрішнього подання символьних даних використовується ASCII код, згідно з яким кожному символу відповідає 8-розрядний код, тобто в один байт записується один символ. Наприклад, текст “1486DX2” в пам'яті має вигляд:



Порядок виконання роботи

1. Вивчити теоретичні основи .
2. Загрузіть Python , виконайте завдання:

1) Змінній **var_int** надайте значення **10**, **var_float** - значення 8.4, **var_str** – **“No”**.

2) Змініть значення, збережене у змінній **var_int**, збільшивши його в 3.5 рази, результат зв'яжіть зі змінною **big_int**.

3) Змініть значення, збережене у змінній **var_float**, зменшивши його на одиницю, результат зв'яжіть з тією ж змінною.

4) Розділіть **var_int** на **var_float**, а потім **big_int** на **var_float**. Результат даних виразів не прив'язуйте ні до яких змінних.

5) Змініть значення змінної **var_str** на **“NoNoYesYesYes”**. При формуванні нового значення використовуйте операції конкатенації (**+**) і повторення рядка (*****).

3. Напишіть програму **digit.py**, яка отримує з першого аргументу командного рядка ціле число, а після друкує його в різних системах числення.

Результати повинні бути розділені між собою пробілами і йти в наступному порядку: десяткове число, двійкове число, вісімкове число, шістнадцяткове число.

4. Виведіть значення всіх змінних. Введіть на виконання приклади:

Приклад 1.1. Приклади цілих чисел

```
# числа в 10-овій с/ч dec1 = 8; dec2 = -3
dec3=258028365723567
# числа в 16-овій с/ч
hex1 = 0x9; hex2 = 0xA; hex3 = 0xFF; hex4 = 0x3de
# число у двійковій с/ч
bin1 = 0b11101101
# число у 8-овій с/ч  oct1 = 0o765
```

Приклад 1.2. Приклади дійсних чисел

```
a = 0.5; b = 3.2
# у експоненційній формі запису
c = 3.2e5    # 3.2 * 10**5
d = 1e-3     # 1 * 10**(-3)
# отримання дійсного значення
float("0.5")  # з рядка
float(3))     # з цілого числа
```

Приклад 1.3. Приклади комплексних чисел

```
c1 = 2 + 3j  # 2 + 3i, 2 - дійсна частина, 3 - уявна
c2 = 5 - 5j  # 5 + 5i
# побудова комплексного числа з дійсного
a = 2; b = 3
c3 = complex(a, b); c4 = complex(5, -5)
```

5. Відповідно до свого варіанту отримати внутрішнє зображення:

№	цілого числа без знаку в одно- або дво-байтовій комірці пам'яті та у 8-ій с/ч	цілого числа в двобайтовій комірці, записати відповідь у 16-ій формі		дійсного числа в двійковій системі числення та в експоненційному нормалізованому вигляді (десятковий дріб $0,XXX_{10}$ перевести у двійкову систему числення з точністю до 2^{-3})
1	**2**	**3**		**4**
1	129	1687	-8542	125,1875
2	456	1597	-5687	159,275
3	423	4236	-1667	753,425
4	789	7524	-2546	851,953
5	159	7458	-9874	153,859
6	267	9532	-6523	753,159
7	357	8542	-9832	456,754
8	459	6578	-4596	159,753
9	852	2154	-7425	426,751
10	345	7596	-5623	953,751
11	259	7845	-8742	759,153
12	531	3256	-3657	698,412
13	214	9888	-8563	325,785
14	742	5236	-4253	254,145
15	853	4523	-7425	741,852
16	963	7845	-8632	963,258
17	751	6587	-7823	745,352
18	862	9832	-8523	752,154
19	953	1245	-9632	985,425
20	842	1597	-9874	965,742

6. Зобразити отримані результати на мові Python. Скласти звіт і захистити його по роботі.

Контрольні питання

1. Які основні типи даних визначені в мові Python.
2. Визначити поняття «система числення, основа системи числення». Що є доповненням числа «0» у двійковій с/ч? Що є основою двійкової с/ч?
3. Яку операцію необхідно виконати для перетворення цілої частини числа з однієї с/ч у іншу:
   а) ділення;
   б) віднімання;
   в) множення;
   г) додавання?
4. Які основні типи даних визначені в мові Python?
5. Запишіть програму, яка отримує з першого аргументу командного рядка ціле число, а після друкує його в різних системах числення.

Зміст звіту

1. Мета роботи.
2. Завдання до роботи.
3. Відповіді на контрольні питання.
4. Результати тестування: вхідні дані, скрипти та результати роботи програми.
5. Інтерфейс роботи з програмою в декількох режимах.
6. Висновки, що відображають особисто отримані результати виконання роботи та їх критичний аналіз. До звіту додаються файли проекту.

ЛАБОРАТОРНА РОБОТА №3

Тема: Розробка блок-схеми алгоритму розв'язання задачі.

Мета: ознайомитися з правилами побудови та визначення блок–схем алгоритмів для написання програм, сформувати вміння і навички роботи з графічними елементами та базовими структурами блок-схем.

Час виконання роботи: 2години.

Програмне забезпечення: Microsoft Visio.

Короткі теоретичні відомості

Алгоритм та його властивості

Розв'язання будь-якої задачі на комп'ютері відбувається в кілька етапів: формулювання постановки задачі; конструювання алгоритму розв'язання задачі; складання програми за розробленим алгоритмом; введення в комп'ютер програми і вихідних даних; налагодження і тестування програми; отримання розв'язку та аналіз результатів.

Алгоритм - це кінцева послідовність чітко визначених дій, які призводять до однозначного вирішення поставленого завдання. Головна особливість будь-якого алгоритму - формальне виконання, що дозволяє виконувати задані дії (команди) не тільки людині, але і різним технічним пристроям (виконавцям). Процес складання алгоритму називається алгоритмізацією.

Алгоритми характеризуються обчислювальною і ємнісною складністю. За видом використовуваної обчислювальної моделі алгоритми діляться на послідовні (або детерміновані), паралельні (або недетерміновані), розподілені та ін. Алгоритм можна реалізувати в комп'ютері, якщо він містить тільки елементарні команди. Елементарними, тобто не потребуючими деталізації, можна вважати такі команди (або операції):

1) початок, кінець;

2) список даних;

3) введення, виведення;

4) обчислювальні операції, реалізовані оператором присвоювання.

Для алгоритму характерні такі *властивості*:

1) детермінованість (або визначеність), тобто однозначність його розуміння для будь-якого виконавця, що приводить до точного виконання однієї і тієї ж послідовності дій;

2) результативність, чи спрямованість, тобто властивість досягнення за кінцеву кількість простих кроків шуканого результату задачі;

3) масовість, тобто придатність до розв‘язання будь-якої задачі з деякого класу задач.

Розрізняють такі способи подання алгоритмів: текстовий, операторний і графічний. Найбільше поширення в наш час одержав графічний спосіб, при якому обчислювальний процес розчленовується на окремі операції, що відображаються у вигляді умовних графічних символів (блоків).

Послідовність блоків і сполучних ліній утворюють блок–схему. Блок– схеми розташовуються зверху вниз. Лінії з'єднання окремих блоків показують напрямок процесу обробки в схемі. Кожний такий напрямок називається гілкою. Незалежно від структури алгоритм завжди має по одному блоку *“Початок”* і *“Кінець”*. Його гілки повинні в кінці зійтися, і за якою б гілки не було б розпочато рух, він завжди має привести до блоку *“Кінець”*.

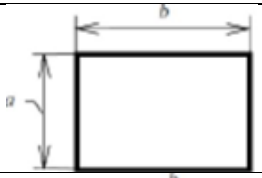
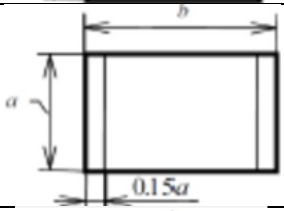
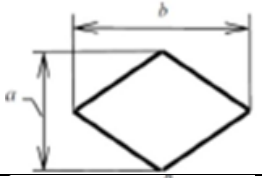
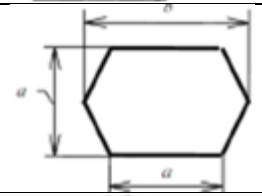
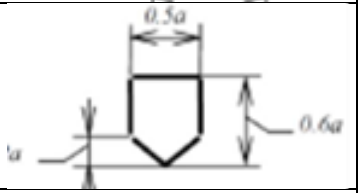
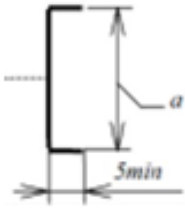
З 01.01.92 введено ДСТ 19.701-90 (ІСО 5807-85), який визначає правила виконання схем і перелік блоків, їхнього найменування, форму і розміри. Блоки з'єднуються між собою у визначеній послідовності стрілками. Усередині блоків у вигляді формул чи тексту вказують інформацію, яка пояснює та характеризує виконувані ними дії. Для визначення алгоритму за допомогою блок-схеми використовуються чітко визначені блоки, основні типи яких наведено в табл. 3.1. Розміри блоків розраховують відповідно до таких правил:

1) найменший геометричний розмір a={10, 15, 20, …} мм;

2) найбільший геометричний розмір $b = 1{,}5a$.

Теорія структурного програмування доводить, що алгоритм будь-якого ступеня складності можна побудувати за допомогою основного базового набору структур: послідовна (лінійна) структура; структура, що розгалужується; циклічна структура.

Таблиця 3.1. - *Графічні елементи, з яких будуються блок-схеми*

Назва	**Графічний блок**	**Функціональне призначення**
Обчислювальний блок		Обчислення або послідовність обчислень
Визначений процес		Виконання підпрограми (функції)
Логічний блок (блок умови)		Перевірка умови
Цикл		Початок циклу
Перехід		З‘єднання між двома сторінками
Коментарі		Пояснення

Найпростішими для розуміння і використання є лінійні структури. ***Лінійним*** називають алгоритм (фрагмент алгоритму), в якому окремі команди виконуються послідовно одна за іншою, незалежно від значень вхідних даних і проміжних результатів. Розглянемо приклад алгоритму лінійної структури (рис. 3.1), який описує алгоритм обчислення площі трикутника зі сторонами *a, b, c* за формулою Герона (спочатку треба визначити значення на півпериметру трикутника, потім розрахувати його площину).

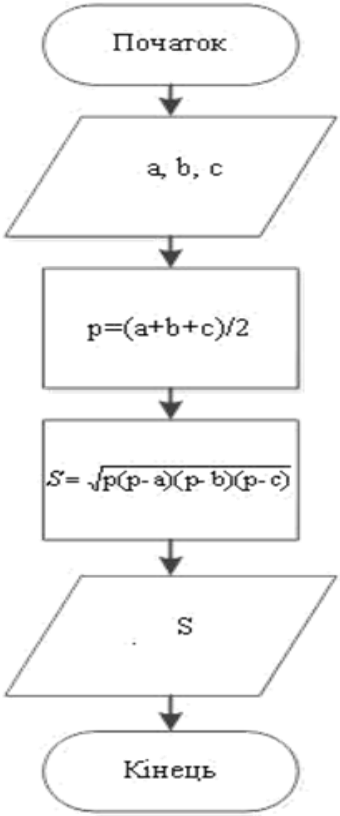


Рис. 3.1. Блок-схема алгоритму обчислення площини трикутника

Створення схем алгоритмів засобами Microsoft Visio

MS Visio – засіб створення різних діаграм, у тому числі інформаційних моделей процесів, блок-схем, структурних схем, графіків робіт, призначений для фахівців; керівників проектів, він дозволяє описувати (моделювати) різні інформаційні технології і системи в керуванні й проектуванні.

Створення будь-якої діаграми розпочинається з команди *Файл-створити - Вибір типу діаграми*. Створення блок схеми розпочинається з того, що на діалоговій панелі вибираємо елемент *Меню – Блок-схема*, потім - *Проста блок-схема*. Після вибору типу діаграми відкривається порожнє вікно (**blank drawing**) **Visio** готове для створення діаграми.

Ліворуч (на зеленому полі) розташовується перелік символів, специфічний для обраного типу *діаграми.*

Символи можуть розташовуватися на бланку діаграми простою операцією перетаскування (ліворуч/праворуч). Для кожного символу надається короткий опис, для цього досить встановити курсор миші на відповідному символі.

Процес побудови діаграм в Visio. Спочатку на робочий аркуш перетягуються необхідні символи-примітиви (**SmartShapes**), потім будуються зв'язки між ними і розташовуються підписи (**текст**). Технологія (**Smart Connector**) дозволяє переміщувати примітиви на робочому аркуші, при цьому зв'язки автоматично перебудовуються, а у випадку перетинання сполучних ліній автоматично створюється “перехрестя” у вигляді арки, яка складається з дуги або декількох (від двох до семи) відрізків.

Перш ніж розпочати створення діаграми, виберемо режим, в якому всі її елементи будуть з'єднуватися автоматично. Для цього використовуємо кнопку *Автосоединение (вкл/выкл),* розташовану на панелі інструментів.

Порядок виконання роботи

1. Вивчити теоретичні основи опису алгоритмів за допомогою блок-схем. Опрацювати приклади.

2. Відповідно до свого варіанту завдання виконати такі кроки:

– побудувати блок-схему алгоритму обчислення значень за даними варіантів у відповідності з індивідуальним завданням в Microsoft Visio,

– сформувати блок-схему у середовищі Microsoft Visio, зберегти її в форматі Microsoft Visio та у текстовому документі.

3. Відповісти на контрольні питання.

4. Скласти звіт і захистити його по роботі.

Варіанти завдань до лабораторної роботи

№	РОЗРАХУНКОВІ ФОРМУЛИ (вихідні дані)	Значення вхідних даних
1	2	3
1	$Y=\sqrt{x^3+ax^2+bx+c}$, де a=x+0,52b; b=$e^{(cx2+1)}$	x=0,35; c=0,8
2	$Y=5\sin^2\left\|\ln(cx^3+1)\right\|$, де x=$\cos^2(a+b)$; b=$\sin^2(a)$	a=0,52; c=1,5
3	$Y=\ln(e^x+bx^2)$, де x=$\cos^2(a+b)$; b=$\sin^2(a)$	a=0,52
4	$Z=\frac{\sqrt{x^4+ax+b}}{\sqrt{x^4+ax+b}}$, де a=x^2+lg(0,08)+e^{-x}; b=x+4a	x=0,75
5	$C=\frac{D+\sqrt{x}}{\ln(\sqrt{R+E})}$, де D=12,5+ln(E); R=8D–x^4+1	x=1,08; E=0,7
6	$Y=(1+z)\frac{x+\frac{y}{x}}{a-\frac{1}{1+x}}$, де $y=e^{\sqrt{x^3+1,75z}}$; $z=5\cdot10^{-1,8}$	$x=0,8\cdot10^{-2}$
7	$Y=\frac{(a+b)^n}{1+\frac{x}{x^m+b^{m-n}}}$, де m=n–1; n=$e^{a/b}$; a=lg(0,083); b=e^a	x=0,81
8	$A=\frac{\alpha_1\beta_1-\alpha_2\beta_2}{\alpha_1-\alpha_2^2}$, де α_1=sin(0,18); $\beta_1=e^{\alpha_1}$; $\alpha_2=\ln(0,05)+e^{\alpha_1}$	β_2=1,7

1	2	3
9	$S = K_1 a_1 + K_2 a_2 - \beta$, де $a_1 = a_0 + \Delta a$; $a_2 = T\sin(30^\circ) + \omega$; $\omega = e^{K_1+K_2}$	K_1=0,05; K_2=0,03; T=1; a_0=1; $\Delta a = 0{,}1$
10	$Z = (a\sqrt{b} - c\sqrt{d})^2 \frac{5{,}6}{a+b+c}$, де $a = \sqrt{(b-c)^3}$; $b=a^2-1$; $d=e^{(b^2-a)}$	c=1,0
11	$Y = \sin\frac{1}{x+0.2} + \lg(0{,}08e^x)$, де $x = -2{,}5a + b\sqrt{c}$; $a=0{,}8c+bx$	c=0,5; b=1
12	$Y = \frac{\cos^2 ax + b}{\sin^2 x}$, де $a = 0{,}5c + e^x$; $b=x^2-ac$; $c=0{,}8\ln(0{,}072)$	x=0,82
13	$Y = \frac{x + 3a - K_1 x}{K_2 x + K_3 x}$, де $x=5a+K_1K_2$	K_1=0,8; K_2=K_3=0,5; a=0,56
14	$Y = \frac{x^4 - x^2 - b}{x-b} + \frac{x^3 - x - a}{x-a}$, де $b=2^x-1$; $x=\lg(0{,}005)+1$	a=1,0
15	$Y = 5\sin^2\left\lvert cx^3 + 1\right\rvert$, де $x=\cos^2(a+b)$; $b=\sin^2(a)$	a=0,52; c=1
16	$Z = \frac{\sqrt{x^4 + ax + b}}{\sqrt{x^4 + ax + b}}$, де $b=x+4a$; $a=-x^2+\lg(0{,}08)+e^{-x}$	x=0,75
17	$Y = (1+z)\frac{x + \frac{y}{x}}{a - \frac{1}{1+x}}$, де $y = e^{\sqrt{x^3+1{,}75z}}$; $z=5\cdot10^{-1,8}$	$x=0{,}8\cdot10^{-2}$; a=1,0
18	$A = \frac{\alpha_1\beta_1 - \alpha_2\beta_2}{m\alpha_1 - \alpha_2^2}$, де $\beta_1 = e^{\alpha_1}$; $\alpha_2 = \ln(0{,}05) + e^{\alpha_1}$; $\alpha_1=\sin(\pi/2)$	β_2=1,7; m=1
19	$Z = (a\sqrt{b} - c\sqrt{d})^2 \frac{5{,}6}{a+b+c}$, де $a = \sqrt{(b-c)^3}$; $b=a^2-1$; $d = e^{(b^2-a^2)}$	c=1
20	$A = \frac{\alpha_1\beta_1 - \alpha_2\beta_2}{\alpha_1 - \alpha_2^2}$, де $\beta_1 = e^{\alpha_1}$; $\alpha_2 = \ln(0{,}05) + e^{\alpha_1}$; $\alpha_1=\sin(\pi/3)$;	β_2=1,7

Контрольні питання

1. Що таке алгоритм? Які існують способи опису алгоритмів?
2. Що таке блок-схема?
3. Основні графічні елементи блок-схем, їх призначення.
4. Які існують правила оформлення блок-схем?
5. Який нормативний документ містить правила оформлення схем алгоритмів?
6. Які існують основні групи графічних форматів в Microsoft Visio?
7. Які шаблони елементів застосовані для формування схеми алгоритму?
8. Які засоби контролю за розмірами елементів передбачені Microsoft Visio?
9. Які засоби вирівнювання і розподілу елементів застосовуються в Microsoft Visio?
10. Як додати текст в діаграмі Microsoft Visio?
11. Як здійснюється експорт фрагментів діаграм Microsoft Visio в текстовий редактор?

Зміст звіту

1. Мета роботи.
2. Завдання до роботи.
3. Відповіді на контрольні питання.
4. Текст розробленого програмного забезпечення.
5. Результати тестування: вхідні дані та результати роботи програми.
6. Висновки, що відображають особисто отримані результати виконання роботи, їх критичний аналіз.

До звіту додаються файли проекту.

ЛАБОРАТОРНА РОБОТА №4

Тема: Алгоритми послідовної (лінійної) структури.

Мета: ознайомитися з алгоритмами послідовної (лінійної) структури та їх реалізацією; можливою обробкою помилок (виключень), з процедурами запуску програм, які реалізують ці алгоритми на мові Python.

Час виконання роботи: 2 години.

Програмне забезпечення: IDLE (Python 3.X).

Короткі теоретичні відомості

Запуск програми користувачем

В преших лабораторних роботах було розглянуто способи запуску програмного коду в Python, який був збережений у файлі. *Повторимо:*

Запуск файлів з командного рядка. Відкрийте текстовий редактор (наприклад, Notepad або редактор IDLE) і збережіть такі інструкції в файлі з ім'ям *script1.py* (перший сценарій на мові Python):

```
import sys # Завантажує бібліотечний модуль
print(sys.platform)
print(2 ** 100) # Підносить число 2 до степеня 100
 x = „Spam!‘
print(x * 8) # Дублює рядок
```

Цей файл:

1. Імпортує модуль ***sys Python*** (бібліотеку додаткових інструментів), щоб пізніше отримати назву платформи ОС.
2. Тричі викликає функцію ***print,*** щоб відобразити результати.
3. Використовує змінну з ім'ям x, утворену в момент, коли їй присвоюють значення у вигляді об'єкта-рядка.
4. Виконує деякі операції над об'єктами.

Ім'я ***sys.platform*** – це змінна-рядок, вміст якої ідентифікує тип комп'ютера, на якому виконується сценарій. Ця змінна знаходиться в модулі з ім'ям ***sys***, який необхідно завантажити з допомогою інструкції ***import.***

Коментар – це текст, розташований за символом #. Коментарі можуть займати окремий рядок або додаватися в рядок з програмним кодом, праворуч від нього. Текст, розташований за символом «#», інтерпретатором ігнорується.

Файл модуля називається ***script1.py.*** Імена файлів з програмним кодом, які передбачається імпортувати з інших файлів, повинні закінчуватися розширенням ****.py***. Якщо зберегти цей текстовий файл, то можна запропонувати інтерпретатору Python виконати його, вказавши повне ім'я файлу в якості першого аргументу команди ***python***, ввівши такий рядок в системній командному рядку:

```
%python script1.py
win32
1267650600228229401496703205376
Spam!Spam!Spam!Spam!Spam!Spam!Spam!Spam!
```

І в цьому випадку також ви повинні використовувати командну оболонку, яка надається операційною системою - у вікні командного рядка *(Command Prompt*) в Windows. Не забувайте замінювати слово «python» на повний шлях до виконуваного файлу інтерпретатора, якщо змінна оточення PATH у вас не налаштована

Якщо все було зроблено правильно, ця команда запустить інтерпретатор Python, який в свою чергу послідовно, рядок за рядком, виконає інструкції в файлі, і ви побачите на екрані результати виконання трьох інструкцій - назва платформи, результат піднесення числа 2 до ступеня 100 і результат багаторазового дублювання рядка. Якщо щось пішло не так, на екрані з'явиться повідомлення про помилку. Оскільки в даній ситуації для запуску програм використовують командну оболонку, можна застосовувати будь-які синтаксичні конструкції, припустимі у командній оболонці. Наприклад, можна направити висновок сценарію Python у файл, щоб детально досліджувати отримані результати пізніше, як показано нижче:

```
% python script1.py > saveit.txt
```

У цьому випадку три рядки, показані в попередньому прикладі запуску сценарію, не будуть виводитися на екран, а будуть записані в файл ***saveit.txt.*** Це – можливість перенаправлення потоків, її можна використати як для виведення тексту, так і для введення. В середовищі Windows цей приклад буде працювати так само, хоча командний рядок буде виглядати інакше:

```
C:\Python30> python script1.py
win32
1267650600228229401496703205376
Spam!Spam!Spam!Spam!Spam!Spam!Spam!Spam!
```

Якщо у вас змінна оточення PATH не налаштована і не був виконаний перехід в каталог інтерпретатора, необхідно вводити повний шлях до виконуваного файлу інтерпретатора Python:

```
D:\temp> C:\python30\python script1.py
win32
1267650600228229401496703205376
Spam!Spam!Spam!Spam!Spam!Spam!Spam!Spam!
```

У новітніх версіях Windows можна просто вводити ім'я файлу сценарію незалежно від того, в якому каталозі ви перебуваєте, бо новітні версії системи Windows відшукують програми, необхідні для запуску файлів, за допомогою реєстру Windows, і вам не потрібно явно вказувати її в командному рядку. *Наприклад*, в сучасних версіях Windows попередню команду можна спростити:

```
D:\temp> script1.py
```

Необхідно вказувати повний шлях до файлу сценарію, якщо він знаходиться в каталозі, відмінному від того, в якому ви працюєте. *Наприклад,* команда:

```
D:\other> python c:\code\otherscript.py
```

буде працювати в каталозі *D:\other*, якщо шлях до команди **python** включений в змінну оточення PATH, при цьому вона повинна запустити сценарій, розташований в каталозі *D:\other.*

Якщо змінна оточення PATH не включає шлях до каталогу з виконуваним файлом інтерпретатора Python і при цьому файл сценарію не знаходиться в поточному робочому каталозі, тоді необхідно вказати повний шлях як до виконуваного файлу інтерпретатора, так і до файлу сценарію:

D:\other> C:\Python30\python c:\code\otherscript.py

Інтерфейс користувача IDLE. Програма IDLE може запропонувати вам графічний інтерфейс користувача для розробки програм на мові Python. IDLE - це набір інструментальних засобів з графічним інтерфейсом, який здатний працювати на самих різних платформах, включаючи Microsoft Windows. IDLE - це програма на мові Python, яка створює графічний інтерфейс за допомогою бібліотеки *tkinter* GUI, що забезпечує її переносимість, але також означає, що для використання IDLE вам доведеться забезпечити підтримку *tkinter* в Python (версія Python для Windows має таку підтримкою за замовчуванням).

Запуск IDLE в Windows – для неї створюється окремий пункт в розділі Python меню кнопки Пуск (Start). У Windows IDLE є сценарієм Python, який за замовчуванням знаходиться в каталозі *C:\Python30\Lib\idlelib*.

В IDLE присутні звичні пункти меню, а для виконання найбільш поширених операцій можна використовувати короткі комбінації клавіш.

Щоб створити (або відредагувати) файл з вихідним програмним кодом в середовищі IDLE, відкрийте вікно текстового редактора: в головному вікні відкрийте меню File (Файл) - пункт New Window (Нове вікно) – відкрити вікно текстового редактора (або Open ... (Відкрити) – щоб відредагувати існуючий файл). Для запуску сценарію, відкритого у вікні редагування, використовують меню «Run» цього вікна (пункт «Run Module»).

IDLE забезпечує підсвічування синтаксису програмного коду, який вводиться як в головному вікні, так і у всіх вікнах текстового редактора – ключові слова виділяються одним кольором, літерали іншим кольором і т. д. Це дозволяє візуально

виділяти елементи програмного коду і розрізняти синтаксичні елементи програмного коду.

Щоб запустити файл з програмним кодом в середовищі IDLE, виберіть вікно, де редагується текст, розкрийте меню Run (Запустити) і виберіть в ньому пункт Run Module (Запустити модуль) або скористайтеся комбінацією клавіш, яка відповідає цьому пункту меню. Якщо з моменту відкриття або останнього збереження файлу його вміст змінювалося, Python запропонує зберегти його.

Коли сценарій запускається таким способом, весь висновок, який він генерує, а також всі повідомлення про помилки з'являються в основному вікні інтерактивного сеансу роботи з інтерпретатором (командна оболонка Python).

Крім основних функцій редагування і запуску середовище IDLE надає цілий ряд додаткових можливостей, включаючи налагоджувач і інспектор об'єктів. Налагоджувач IDLE активується за допомогою меню Debug (Налагодження), а інспектор об'єктів – за допомогою меню File (Файл). Інспектор об'єктів дозволяє переходити, переміщаючись по шляху пошуку модулів, до файлів і об'єктів в файлах – клацання на файлі або об'єкті призводить до відкриття відповідного вихідного тексту в вікні редагування.

Режим налагодження в IDLE ініціюється вибором пункту меню Debug (Налагодження) → Debugger (Отладчик) головного вікна, після цього можна запустити налагоджувач сценарію вибором пункту меню Run (Запустити) → Run Module (Запустити модуль). Як тільки налагоджувач буде активований, клацанням правої кнопки миші на вибраному рядку у вікні редагування ви зможете встановлювати точки зупинки в своєму програмному коді, щоб призупиняти виконання сценарію, переглядати значення змінних тощо Ви зможете стежити за ходом виконання програм – в цьому випадку поточний виконуваний рядок програмного коду виділяється кольором.

У разі появи помилок можна натиснути правою кнопкою миші на рядку з повідомленням про помилку і швидко перейти до рядка програмного коду, який викликав цю помилку. Це дозволяє швидко з'ясувати джерело помилки і ліквідувати її. Крім цього текстовий редактор IDLE володіє великим набором

можливостей, які стануть в нагоді програмістам, включаючи автоматичне оформлення відступів, розширений пошук тексту і файлів тощо.

Числа та операції над ними

Крім числових літералів, наведених в табл. 1.2, Python надає набір операцій для роботи з числовими об'єктами:

Оператори виразів +, –, *, /, >>, **, &, інші.

Вбудовані математичні функції **pow**, **abs**, **round**, **int**, **hex**, **bin**, інші.

Допоміжні модулі **random**, **math**, інші.

Для роботи з числами в основному використовуються вирази, вбудовані функції і модулі (при цьому числа мають ряд власних, специфічних методів). Дійсні числа, наприклад, мають метод **as_integer_ratio**, який зручно використовувати для перетворення дійсного числа в раціональне, а також метод **is_integer_method**, який перевіряє – чи можна подати дійсне число як ціле значення. Цілі числа теж мають різні атрибутами, включаючи метод **bit_length**, він повертає кількість бітів, необхідних для подання значення числа. Крім того, множини підтримують власні методи і оператори виразів.

При роботі з числами часто використовують вирази: комбінації чисел (або інших об'єктів) і операторів, які повертають значення при виконанні інтерпретатором Python. Вирази в Python записуються з використанням звичайної математичної нотації і символів операторів. Наприклад, складання двох чисел X і Y записується у вигляді виразу X+Y, яке наказує інтерпретатору Python застосувати оператор «+» до значень з іменами X і Y. Результатом виразу X+Y буде інший числовий об'єкт. У табл. 3.1 наведено перелік всіх операторів, наявних в Python (математичні оператори: **+**, **–**, *****, **/** і т. д.; оператор обчислює залишок від ділення, оператор «<<» виконує побітовий зсув вліво, оператор «&» виконує побітову операцію «І» і т. д.). Деякі оператори більш характерні для Python. Наприклад, оператор «is» перевіряє ідентичність об'єктів, оператор lambda створює неіменовані функції. Нерівність значень можна перевірити як X! = Y. Операція ділення з округленням вниз (X//Y) усікає дробову частину. Операція ділення X / Y виконує справжнє ділення (повертає результат з дробовою частиною).

Таблиця 4.1. - *Оператори виразів в Python і правила визначення старшинства*

Оператори	Опис
yield x	Підтримка протоколу send у функціях-генераторах
lambda args: expression	Створює анонімну функцію
x if y else z	Тримісний оператор вибору (значення x обчислюється, тільки якщо значення y істинно)
x or y	Логічна операція «АБО» (значення y обчислюється, тільки якщо значення x = хибність)
x and y	Логічний оператор «І» (значення y обчислюється, тільки якщо значення x= істина)
not x	Логічне заперечення
x in y, x not in y	Перевірка на входження (для ітерованих об'єктів і множин)
x is y, x is not y	Перевірка ідентичності об'єктів
x < y, x <= y, x> y, x>= y	Оператори порівняння, перевірка на підмножину і над множину
x == y, x != y	Оператори перевірки на рівність
x | y	Бітова операція «АБО», об'єднання множин
x ^ y	Бітова операція «виключне АБО» (XOR), симетрична різниця множин
x & y	Бітова операція «І», перетин множин
x << y, x >> y	Зсув значення x вліво або вправо на y бітів
x + y, x – y	Додавання, конкатенація Віднімання, різниця множин
x * y, x % y, x / y, x // y	Множення, повторення Залишок, формат Ділення: справжнє і з округленням вниз
-x, +x	Унарний знак «мінус», тотожність
~x	Бітова операція «НЕ» (інверсія)

Оператори	Опис
x ** y	Піднесення до степеню
x[i]	Індексація (в послідовності, відображеннях тощо)
x[i:j:k]	Витяг зрізу
x(...)	Виклик (функцій, класів і інших об'єктів)
x.attr	Звернення до атрибуту
(...)	Кортеж, підвираз, вираз-генератор
[...]	Список, генератор списків
{...}	Словник, множина, генератор словників і множин

Порядок виконання роботи

1. Вивчити теоретичні основи написання алгоритмів послідовної (лінійної) структури. Опрацювати приклади.
2. Відповідно до свого варіанту написати програму, яка, використовуючи складові модуля *math*, розраховує значення виразу (для тригонометричних функцій аргументи задано в радіанах, для введення даних з клавіатури та виведення даних на екран використовувати функції введення-виведення).
3. Засвоїти дві процедури запуску програми:
   – в командному рядку;
   – з інтерфейсу IDLE.
4. Підготувати звіт, в якому повинні бути коди програм з коментарем та скріншоти опрацювання програм, відповіді на контрольні запитання. До звіту додаються файли проекту.

Завдання до лабораторної роботи

Завдання 1. *Запуск програми в командному рядку DOS* (більшість програмістів для розробки програм використовують вікно терміналу з командною оболонкою і вікно текстового редактора)

1.1. За допомогою текстового редактора «Блокнот» («Notepad») створити файл *l_r4.py*, який містить в собі код програми (інструкції мови).

1.2. Запустити цей файл на виконання за допомогою інтерпретатора Python у командному рядку DOS:

меню «Все программы (Programs)», яке з‘являється після кліку на кнопці Пуск (Start)),

меню Accessories (Стандартные) → Command Prompt (Командная строка) або Пуск (Start) → Выполнить… (Run…) → *cmd*.

1.3. Зафіксувати результат виконання коду програми (клавіша PtrScr).

```
>>> import sys
>>> a=sys.path
>>> a
['', 'C:\\WINDOWS\\system32\\python34.zip', 'C:\\Python34\\DLLs', 'C:\\Python34\
\lib', 'C:\\Python34', 'C:\\Python34\\lib\\site-packages']
>>>
```

Завдання 2. *Запуск програми у середовищі IDLE.*

2.1. За допомогою текстового редактора IDLE створити текстовий файл ***l_r4.py***, який містить код програми (IDLE автоматично не додає розширення до імені файла, навіть «****.py***»).

2.2. Запустити цей файл на виконання за допомогою інтерфейсу IDLE.

2.3. Зафіксувати результат виконання коду програми (клавіша PtrScr). Програми мають бути записані в окремих файлах, які можуть виконуватися консольною командою ***python <ім'я файлу>.py.***

Результат обчислень має виводитися на екран командою *print*.

Кожна програма має сприймати довільні значення, які задовольняють описаному в завданні формату, і передаються у вигляді аргументів командного рядка при виклику програми:

python <ім'я файлу>.py

Формат введення даних та виведення результатів має чітко збігатися із вказаним.

Приклад 2.3.1. Вбудована функція **input** – це засіб отримання результату введення з клавіатури – вона виводить підказку, текст якої міститься в необов'язковому аргументі-рядку, і повертає введену користувачем відповідь у вигляді рядка. Функція **print** (засіб виведення) може приймати декілька параметрів, які розділяються комами:

```
print ('first', 'second')
```

Результат – first second

Функція **input** призупиняє виконання програми, поки користувач не введе значення і натисне клавішу ENTER. Вона повертає введене значення у вигляді рядка.

1).*text = input(_Enter some text: _); print(text)*

2).*string = input('Введіть строку: ') # або string = input()*
print('Ви ввели "', string, '"', sep='')
 # введіть два числа
n = int(input('Введіть перше число: '))
m = int(input(' Введіть друге число: '))
print('{} + {} = {}'.format(n, m, n + m))

3).Функції **bin, oct, hex** повертають рядки, які зображують задане число у відповідних с/ч.

number = int(input('Введіть число: '))
print('Двійкова с/ч: ', bin(number))
print('Вісімкова с/ч: ', oct(number))
print('Шістнадцяткова с/ч:', hex(number))

Результат

Введіть число: 10
Двійкова с/ч: 0b1010
Вісімкова с/ч: 0o12
Шістнадцяткова с/ч: 0xa

4).Функції **min, max** повертають мінімальне/максимальне значення

a = 5;b = 7;c = 2
print(min(a, b, c)) # мінімальне значення
print(max(a, b, c)) # максимальне значення

5). Обчислити значення функції $y = \left(sin\left(\pi/2\right)\pi/b\right)\cdot\sqrt{m}$ Значення вхідних даних m=4, b=2.

import math as m1
m=float(input("m=")); b=float(input("b="))
a=m1.sin(m1.pi/2)

```
y=(a/b)*m1.sqrt(m); print("y=", y)
```

Результат

```
m=4
b=2
y= 1.0
```

Приклад 2.3.2. Арифметичні операції

2.3.2.1.

```
# Функція float конвертує значення-рядок у дійсне,
  # з яким можна виконувати арифметичні операції
  x = float(input('Введите первое число: '))
  y = float(input('Введите второе число: '))
  # operation — рядок
  operation = input('Введите знак арифметической операции: ')
  # Присвоїмо змінній result значення None, яке вказує, що
  # значення об‘єкта не відомо
  result = None
  # if-elif-else (якщо - інакше якщо - інакше) — умовний оператор.
  # дозволяє виконувати різні фрагменти кода в залежності від умов
  # Операція «==» перевіряє два значення на рівність
  if operation == '+':
      # до чисел можна застосувати  арифметичні операції
      result = x + y
  elif operation == '-':
      result = x - y
  elif operation == '*':
      result = x * y
  elif operation == '/':
      result = x / y
  elif operation == '^':
      result = x ** y  # ** — операція піднесення до степеня else:
      print('Операція, яка не підтримується')
  # отримаємо результат, якщо операція була припустимою
  if result is not None:  print(result)
```

2.3.2.2.

```
x = 10; y = 8
print(x + y)  # додавання
print(x + 3)
print(x - y)  # віднімання
```

```
print(x * y)  # добуток
print(x / y)  # ділення
print(x // y) # цілочисельне ділення
print(x % y)  # залишок від ділення
print(x ** y) # піднесення до степеню
print(3.2 * 0.8 - 2 * 5 - 3**3)  # арифметичний вираз
 print(4 ** 0.5)  # піднесення до степеню 0.5 – квадратний корінь

z = -2
print(abs(z))           # модуль числа
print(pow(z, 2), z ** 2)  # квадрат числа

m = 3.26
print(round(m), round(m, 1))
# Округлення числа до цілого і до одного знака після коми
```

Приклад 2.3.3.

2.3.3.1.Операції на основі модуля **math**

```
import math # імпортуємо модуль math
x = 3.265
#ціле число, найближче ціле знизу, найближче ціле зверху
print(math.trunc(x), math.floor(x), math.ceil(x))
print(math.pi)       # константа пі
print(math.e)        # число Ейлера
y = math.sin(math.pi / 4) # math.sin – синус
print(round(y, 2))

y = 1 / math.sqrt(2) # math.sqrt – квадратний корінь
print(round(y, 2))
```

2.3.3.2. Деякі математичні функції

```
>>>import math
>>>math.pi, math.e # константи pi, e=2.71
(3.1415926535897931, 2.7182818284590451)
>>>math.sin(2*math.pi/180) # Синус
0.034899496702500969
>>>math.sqrt(144), math.sqrt(2) # Квадратний корінь
```

(12.0, 1.4142135623730951)
*>>>math.pow(2, 4), 2 ** 4 # Піднесення до степеня*
(16, 16)
>>>abs(-42.0), sum((1, 2, 3, 4)) # абсолютне значення, сума
(42.0, 10)
>>>min(3, 1, 2, 4), max(3, 1, 2, 4) # Мінімум, максимум
(1, 4)
>>>math.log(10) #ln(10)
math.log(x[, base]) - with one argument, return the natural logarithm of x (to base e).
math.log1p(x) - return the natural logarithm of 1+x (base e).

Контрольні питання

1. Що таке алгоритм послідовної (лінійної) структури, програма послідовної (лінійної) структури?
2. Які основні типи даних визначені в мові Python?
3. Що виконує функція **input**?
4. Для чого потрібні функції **bin, oct, hex, min, max**?
5. Наведіть приклад операцій (щонайменше 5) на основі модуля **math**.
6. Визначте порядок виконання наступних операцій використовуючи правила визначення старшинства:

x and y, x + y, x * y, x ^ y.

Зміст звіту

1. Мета роботи.
2. Завдання до роботи.
3. Відповіді на контрольні питання.
4. Текст розробленого програмного забезпечення.
5. Результати тестування: вхідні дані та результати роботи програми.
6. Висновки, що відображають особисто отримані результати виконання роботи, їх критичний аналіз.

До звіту додаються файли проекту.

ЛАБОРАТОРНА РОБОТА №5

Тема: Алгоритми розгалуженої структури (інструкція if)
Мета: ознайомитися з алгоритмами розгалуженої структури та їх реалізацією; можливою обробкою помилок (виключень).
Час виконання роботи: 2 години.
Програмне забезпечення: IDLE (Python 3.X).

Короткі теоретичні відомості

Динамічна типізація

У сценаріях на Python не виконують оголошення об'єктів певних типів. Типи даних визначають під час виконання, а не в результаті оголошень у програмному коді.

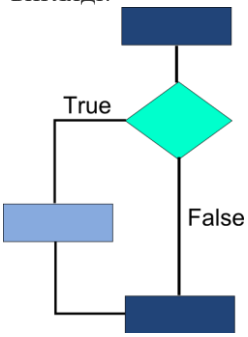
Змінні утворюються при виконанні операції присвоєння, можуть посилатися на об'єкти будь-яких типів і перш ніж до них можна буде звернутися їм необхідно присвоїти деякі значення.

Наприклад, якщо ввести таку інструкцію: *a*=3, інтерпретатор Python виконає цю інструкцію в такі три етапи: 1) створює об'єкт, який подає число 3, 2) створює змінну *a*, якщо вона ще відсутня, 3) у змінну *a* записується посилання на новостворений об'єкт, який подає число «3». Результатом виконання цих етапів буде структура, зображена на рис. 5.1: змінні і об'єкти зберігаються в різних частинах пам'яті і пов'язані між собою посиланням (посилання на рисунку зображено у вигляді стрілки).

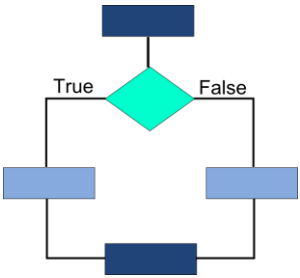
Рис. 5.1. Імена та об'єкти після виконання операції присвоювання *a* = 3. Змінна перетворюється на посилання на об'єкт «3», у внутрішньому поданні змінна є вказівником на простір пам'яті з об'єктом, створеним в результаті інтерпретації літерального виразу «3»

Змінні завжди посилаються на об'єкти і ніколи – на всі інші змінні, але великі об'єкти можуть посилатися на інші об'єкти (наприклад, об'єкт списку містить посилання на об'єкти, які включені в список).

Можна зобразити блок-схему програми, яка містить інструкцію **if**, в такому вигляді:



Зустрічається і більш складна форма розгалуження: **if-else**. Якщо умова при інструкції **if** виявляється хибною, то виконується блок коду при інструкції **else**.



Приклад коду з гілкою **else** на мові програмування Python:

```
tovar1 = 50
tovar2 = 32
if tovar1 + tovar2 > 99:
   print("Сума не достатня")
else:
   print("Чек сплачено" )
```

Якщо використовують змінну (тобто вказівник), інтерпретатор Python переходить за посиланням від змінної до об'єкта. У конкретних термінах: *змінні*

– це записи в системній таблиці, де передбачено місце для зберігання посилань на об'єкти, *об'єкти* – це області пам'яті за об'ємом, достатнім для подання значень цих об'єктів, *посилання* – це вказівники на об'єкти.

Коли в сценарії в результаті виконання виразу створюється нове значення, інтерпретатор Python створює новий об'єкт (тобто виділяє область пам'яті), яка подає це значення. Внутрішня реалізація інтерпретатора для оптимізації кешує і повторно використовує деякі типи незмінних об'єктів, такі як малі цілі числа і рядки (кожен «0» насправді не є новою областю в пам'яті).

Об'єкти мають більш складну структуру, ніж просто простір в пам'яті, необхідний для зберігання значення. Кожен об'єкт має два стандартних поля: описувач типу (який використовують для зберігання інформації про тип об'єкта) і лічильник посилань (який використовують для визначення моменту, коли пам'ять, яку займає об'єкт, може бути звільнена).

Інформація про тип зберігається в об'єкті, але не в змінній. Щоб побачити, як використовується інформація про типи об'єктів, подивимося, що відбувається, якщо виконати кілька операцій присвоювання одній і тій же змінній:

```
>>> a = 3 # це ціле число
>>> a = 'spam' # тепер це - рядок
>>> a = 1.23 # тепер це – дійсне  число
```

Цей програмний код працює таким чином: спочатку створюється ціле число, потім рядок і, нарешті, дійсне число. Тип – це властивість об'єкта, а не імені. У коді змінюється посилання на об'єкт. Тому що змінні не мають типів, ми насправді не змінюємо тип змінної – ми записуємо в змінну посилання на об'єкти інших типів. Змінні посилаються на конкретні об'єкти в конкретні моменти часу. З іншого боку, об'єкти знають, до якого типу вони відносяться, кожен об'єкт містить поле, в якому зберігається інформація про його типі. Цілочисельний об'єкт «3», наприклад буде містити значення «3» плюс інформацію, яка

повідомить інтерпретатор Python, що об'єкт є цілим числом (це вказівник на об'єкт з назвою **int**, який відіграє роль імені цілочисельного типу). Описувач типу для рядка '**spam**' вказує на тип рядок (з ім'ям **str**). Оскільки інформація про тип зберігається в об'єктах, її не потрібно зберігати в змінних.

Таким чином, тип в мові Python – це властивість об'єктів, а не змінних. У програмному коді змінна зазвичай посилається на об'єкти тільки одного типу.

Умовна інструкція if

***Логіка висловлювань. Логічний тип даних*.** Числа можна порівнювати.

Для цього в Python є такі операції порівняння:

> більше	<= менше чи рівне
< менше	== дорівнює
>= більше чи рівне	!= не дорівнює

Наприклад:

6>5 – True

7<1 – False

7==7 – True

7 != 7 – False

Python повертає значення **True** (Істина == 1), коли порівняння істинне, **False** (Хибність == 0) – в іншому випадку. **True** і **False** відносяться до логічного (булевого) типу даних **bool**. В програмах часто використовують більш складні вирази – висловлювання (це – означає деяке твердження, яке може бути істинним або хибним). Кажуть, що істинність (**True**) або хибність (**False**) – це логічні значення висловлювання.

Логіка висловлювань вивчає способи, за допомогою яких з одних висловлювань можна утворювати інші, причому в такий спосіб, щоб істинність або хибність нових висловлювань залежала лише від істинності або хибності старих. Для цього використовуються так звані (логічні) сполучники. Python використовує три логічних оператора **and**, **or**, **not** які відповідають сполучникам І, АБО, НЕ в логіці висловлювань.

A	B	A∧B
T	T	T
T	F	F
F	T	F
F	FF	F

A	B	A ∨B
T	T	T
T	F	T
F	T	T
F	FF	F

A	¬ A
F	T
T	F

Наприклад:

```
x = 8
y = 13
x == 8 and y < 15 # x дорівнює 8 та y меньше 15
x > 8 and y < 15 # x більше 8 та y меньше 15
x != 0 or y >15 # x не дорівнює 0 або y меньше 15
x < 0 or y >15 # x меньше 0 або y меньше 15
```

Для Python істинним або хибним може бути не тільки логічне висловлювання, а й об'єкт. Будь-яке число, яке не дорівнює нулю (або непорожній об'єкт) інтерпретують як «істина». Числа, які дорівнюють нулю, порожні об'єкти та спеціальний об'єкт **None** інтерпретують як «хибність». *Наприклад:* 1) '' and 2 # False and True маємо результат «''»;

2) '' or 2 # False or True – «2»

```
>>> y = 6>8; y False
>>> not y
True
>>> not None
True
>>> not 2
False
>>> 2>4 and 45>3 #  False and  True поверне значення False
False
```

При обчисленні оператора **and** Python обчислює операнди зліва направо і повертає перший об'єкт, який має помилкове значення:

```
>>>0 and 3 # поверне перший помилковий об'єкт-операнд
0
>>>5 and 4 # поверне крайній правий об'єкт-операнд
4
```

Якщо Python не знаходить помилковий об'єкт-операнд, він повертає крайній правий операнд.

Логічний оператор **or** діє схожим чином, але для об'єктів-операндів Python повертає перший об'єкт, який має значення «істина». Python припинить подальші обчислення, як тільки буде знайдений перший об'єкт, який має значення «істина».

```
>>>2 or 3 # повертає перший об'єкт-операнд зі значення
«істина»
2
>>>None or 5 # повертає другий операнд, тому що перший - хибний
5

>>>None or 0 # повертає об'єкт-операнд, що залишився
0
```

Логічні вирази можна комбінувати:

```
>>>1+3 > 7 # пріоритет + вище, ніж >
False
>>> (1+3) > 7 # дужки сприяють наочності і позбавляють від помилок
False
>>>1+(3>7)
1
```

Python можна перевіряти належність до інтервалу:

```
>>>x=0
>>>–5<x<10 # еквівалентно: x > –5 and x<10
True
```

Рядки в Python можна порівнювати аналогічно числам. Символи, як і все інше, подають в комп'ютері у вигляді чисел.

Існує таблиця, яка ставить у відповідність кожному символу деяке число. Визначити, яке число відповідає символу можна за допомогою функції **ord()**:

```
>>>ord ('L')
76
>>>ord ('Ф')
1060
```

Тепер порівняння символів зводиться до порівняння чисел, які їм відповідають:

```
>>>'A' > 'L' 6
False
```

При порівнянні рядків Python їх порівнює посимвольно:

```
>>>'Aa' > 'Ll'

False
>>>
Оператор in перевіряє наявність підрядка в рядку:
>>>'a' in 'abc'
True
>>>'A' in 'abc' # великої літери А немає
False
>>>"" in 'abc' # порожній рядок є в будь-якому рядку
True
>>>'' in ''
True
```

Інструкція if. Розглянемо умовну інструкцію **if**, яку використовують для вибору серед альтернативних операцій на основі результатів перевірки. Інструкція **if** обирає, яку дію необхідно виконати. Вона може містити інші інструкції, в тому числі інші умовні інструкції **if.**

Спочатку записується частина **if** з умовним виразом, далі можуть слідувати одна або більше необов'язкових частин **elif** («**else if**») з умовними виразами і, нарешті, необов'язкова частина **else**. Умовні вирази і частина **else** мають асоційовані з ними блоки

вкладених інструкцій, з відступом щодо основної інструкції. Під час виконання умовної інструкції **if** інтерпретатор виконує блок інструкцій, асоційований з першим умовним виразом, тільки якщо він повертає значення «істина», в іншому випадку виконується блок інструкцій **else**. Загальна форма запису умовної інструкції **if** виглядає таким чином:

```
if <test1>: # Инструкция if с условным выражением test1
<statements1> # Ассоциированный блок
elif <test2>: # Необязательные части elif
<statements2>
else: # Необязательный блок else
<statements3>
```

Розглянемо кілька прикладів. Всі частини цієї інструкції, за винятком основної частини **if** з умовним виразом і пов'язаних з нею інструкцій, є необов'язковими. У найпростішому випадку інші частини інструкції опущено:

```
>>> if 1:
...      print ‘true‘
...
True
```

Запрошення до введення змінюється на «...» для рядків продовження в базовому інтерфейсі командного рядка, що використовується тут (в IDLE текстовий курсор переміщається на наступний рядок вже з відступом, а натискання на клавішу *Backspace* повертає на рядок вгору). Введення порожнього рядка (подвійним натисканням клавіші Enter) завершує інструкцію і призводить до її виконання. Число «1» – це логічна істина, тому дана перевірка завжди буде успішною. Щоб обробити помилковий результат, додайте частину **else**:

```
>>>if not 1:
print _true‘
... else:
print _false‘
false
```

Множинне розгалуження. Розглянемо приклад умовної інструкції **if**, в якій присутні всі необов'язкові частини:

```
>>> x = ‘killer rabbit‘
>>> if x == ‘roger‘:
print “how‘s jessica?”
... elif x ==’bugs‘:
print “what‘s up doc?”
... else:
print ‘Run away! Run away!‘
...
Run away! Run away!
```

Ця багаторядкова інструкція простягається від рядка **if** до кінця блоку **else.** При виконанні цієї інструкції інтерпретатор виконає вкладені інструкції після тієї перевірки, яка дасть в результаті істину, або блок **else**, якщо всі перевірки дадуть результат «хибність». Обидві частини **elif** і **else** можуть бути опущені, і в кожній частині може бути більше однієї вкладеної інструкції. Зв'язок слів **if**, **elif** і **else** визначений тим, що вони знаходяться на одній вертикальній лінії, з одним і тим же відступом.

Python множинне розгалуження оформляють у вигляді послідовності перевірок **if/elif**. Використання інструкції **if** є найбільш простим способом організації множинного розгалуження.

Обробка помилок

Існують три основних типи помилок: помилки етапу компіляції (інтерпретації), етапу виконання та логічні помилки.

Помилки етапу компіляції (або семантичні) відбуваються, коли код порушує правила синтаксису мови. Компілятор не може скомпілювати програму, поки вона не буде містити припустимі оператори і мати правильну структуру. Загальною причиною помилок етапу компіляції є помилки набору (друкарські помилки), посилання на невизначені змінні, пропущені крапка з комою, передача функції неправильних параметрів тощо.

Помилки етапу виконання (або семантичні) відбуваються, коли після компіляції програми під час її виконання робиться

щось неприпустиме (наприклад, програма намагається виконати ділення на нуль або відкрити для запису неіснуючий файл).

Логічні помилки – це помилки проектування та реалізації програми. Ці помилки важко відстежити, оскільки інтерпретатор не може знайти їх автоматично, як синтаксичні та семантичні помилки. Зазвичай засоби налагодження дозволяють їх знайти. Логічні помилки призводять до некоректного або непередбаченого значення змінних, неправильного вигляду графічних зображень або невиконання коду, коли це очікується.

Іноді помилки важко знайти, оскільки вони можуть бути результатом взаємодії різних частин програми. У цьому випадку краще крок за кроком переглянути програму та стан змінних і виразів. Таке виконання – ключовий елемент від лагодження.

Нехай необхідно виконати математичні дії над введеними користувачем числами, наприклад, обчислити квадрати чисел. Для досягнення бажаного ефекту ми могли б спробувати використовувати такі інструкції:

```
while True:
reply = input('Enter text:')

if reply == 'stop': break
print(int(reply) ** 2)
print('Bye')
```

У цьому сценарії використовується однорядкова інструкція **if,** яка виконує вихід із циклу після отримання від користувача рядку «**stop**», а крім того, виконується перетворення введеного рядка для виконання необхідної математичної операції. У даний сценарій також додано повідомлення, яке виводиться в момент завершення роботи сценарію. Оскільки інструкція **print** в останньому рядку не має такого ж відступу, як інструкції вкладеного блоку, вона не вважається частиною тіла циклу і буде виконуватися тільки один раз – після виходу з циклу:

```
Enter text:2
4
Enter text:40
1600
Enter text:stop
```

```
Bye
```

Обробка помилок шляхом перевірки вводу. Якщо користувач введе неправильний рядок, маємо:

```
Enter text:xxx
...текст сообщения об ошибке опущен...
ValueError: invalid literal for int() with base 10: 'xxx'
```

Вбудована функція *int* активує виключення, коли стикається з помилкою. Якщо необхідно забезпечити стійкість сценарію, треба попередньо перевірити вміст рядка за допомогою методу рядків **isdigit:**

```
>>>S = '123'; T ='xxx'; S.isdigit(), T.isdigit()
(True, False)
```

Для цього в наш приклад необхідно додати вкладені оператори. У наступній версії інтерактивного сценарію використовується версія умовної інструкції **if**, яка запобігає *можливість появи винятків:*

```
while True:
        reply = input('Enter text:')
        if reply == 'stop':
          break
        elif not reply.isdigit( ):
          print('Bad!' * 8)
        else:
          print(int(reply) ** 2)
print 'Bye'
```

У повній формі інструкція містить слово **if**, за яким слідують вираз перевірки умови і вкладений блок коду, один або більше необов'язкових перевірок **elif** («**else if**») і відповідних їм вкладених блоків коду і необов'язкова частина **else** із зв'язаним з нею блоком коду, який виконується при недотриманні умови. Інтерпретатор виконує перший блок коду (якщо перевірка дає в результаті значення «істина»), проходячи інструкцію зверху донизу, або частину **else** (якщо всі перевірки дали в результаті значення «хибність»).

Частини **if**, **elif** і **else** в попередньому прикладі належать одній і тій самій інструкції, тому що вертикально вони розташовані на одній лінії (тобто мають однакові відступи). Інструкція **if** простягається до початку інструкції **print** в останньому рядку. У свою чергу, весь блок інструкції **if** є частиною циклу **while**, тому що всю її зсунуто вправо щодо основної інструкції циклу. Тепер новий сценарій буде виявляти помилки перш, ніж вони будуть виявлені інтерпретатором, і виводити повідомлення:

```
Enter text:5
25
Enter text:xyz
Bad!Bad!Bad!Bad!Bad!Bad!Bad!Bad!
Enter text:10
100
Enter text:stop
```

Обробка помилок за допомогою інструкції try. В Python існує більш універсальний спосіб обробки помилок за допомогою інструкції **try**. Використавши цю інструкцію, можна спростити попередній код:

```
while True:
    reply = input('Enter text:')
    if reply == 'stop': break
    try:
        num = int(reply)
    except:
        print('Bad!' * 8)
    else:
        print(int(reply) ** 2)
print 'Bye'
```

Ця версія працює так само, як і попередня, тільки тут ми замінили явну перевірку наявності помилки програмним кодом, який передбачає, що перетворення буде виконано і виконує обробку виключення, якщо таке перетворення неможливо. Ця інструкція **try** складається зі слова **try**, слідом за яким розміщено основний блок коду (дії, які ми намагаємося виконати), з

наступною частиною **except**, де розташовується програмний код обробки винятку. Далі розміщено частину **else**, програмний код якої виконується, якщо в частині **try** виключення не виникло. Інтерпретатор спочатку виконує частину **try**, потім виконує або частина **except** (якщо виник виняток), або частина **else** (якщо виняток не виник).

Тому що слова **try**, **except** і **else** мають однаковий відступ, всі вони вважаються частиною однієї і тієї ж інструкції **try**. В даному випадку частина **else** пов'язана з інструкцією **try**, а не з інструкцією **if**. Ключове слово **else** в мові Python може з'являтися не тільки в інструкції **if**, але і в інструкції **try** і в циклах – величина відступу наочно показує, частиною якої інструкції воно (слово **else**) є. В цьому випадку інструкція **try** розпочинається зі слова **try** і триває до кінця вкладеного блоку коду, розташованого за словом **else**, тому що **else** розміщено на одній відстані від лівого краю, що і **try**. Інструкція **if** в цьому прикладі займає всього один рядок і завершується відразу ж за словом **break**.

Порядок виконання роботи

1. Вивчити теоретичні основи написання алгоритмів розгалуженої структури. Опрацювати приклади.
2. Побудувати блок-схему алгоритму вирішення завдання.
3. Відповідно до свого варіанту:

– визначити умови;

– за допомогою формул описати варіанти виконання необхідний дій;

– написати програму, яка розв‘язує завдання.

– організувати введення даних прикладів з клавіатури, виведення у консоль.

4. Реалізувати обробку помилок (виключень у вигляді рядків):

– за допомогою інструкції **try**;

– перевіривши зміст рядку введення за допомогою методу **isdigit**.

5. Скласти звіт і захистити його по роботі.

Захист роботи включає в себе демонстрацію працездатності програми на різних вхідних даних, демонстрацію трасування

виконання програми, переробку розгалужень, які входять одне до одного.

Вимоги: не можна використовувати масиви, цикли, власні функції.

Завдання до лабораторної роботи

1. Напишіть програмний код, в якому у випадку, якщо значення якоїсь змінної більше 0, виводилося б спеціальне повідомлення (використовуйте функцію **print**). Один раз виконайте програму при значенні змінної більше 0, другий раз - менше 0.

2. Вдоскональте попередній код за допомогою гілки **else** так, щоб залежно від значення змінної, виводилося або 1, або -1.

3. Самостійно придумайте програму, в якій би використовувалася інструкція **if** (бажано з гілкою **else**). Вкладений код повинен містити не менше трьох виразів.

4. Організувати введення з клавіатури, виведення у консоль прикладів.

Приклад 5.1. Приклади логічних операцій та виразів

```
print('and:')
print(False and False);print(False and True)
print(True and False);print(True and True); print()

print('or:')
print(False or False);print(False or True)
print(True or False);print(True or True); print()

print('not:')
print(not False); print(not True); print()

#Логічні вирази
a = True;b = False;c = True
f = a and not b or c or (a and (b or c))
print(f)
```

Приклад 5.2. Приклади порівнянь

```
a = 2; b = 5
print(a < b)                # менше
print(b > 3)                # більше
print(a <= 2)               # менше або дорівнює
print(b >= 7)               # більше або дорівнює
print(a < 3 < b)            # подвійне порівняння
print(a == b)               # рівність
print(a != b)               # нерівність
print(a is b)               # ідентичність об‘єктів в пам‘яті
print(a is not b)           # a и b – різні об‘єкти
#(хоча їхні значення можуть бути однаковими - рівними)
```

Приклад 5.3. Визначити середнє арифметичне заданої непустої послідовності додатних цілих чисел, за якою слідує «0» (це ознака кінця послідовності).

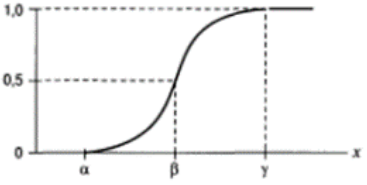
```
from math import *
a=input ('Input first number: ')
if not a.isdigit():
  print("String value can not be entered")
print("or number is negative, restart the program!")
input ("reload the program!")
else: a=int(a)
if a==0:
print("The number can not be zero"); input ()
count=0;ar=0
ar+=a; count+=1
if a!=0:
try:
a=int(input('Input next number or Enter 0 to finish: '))
except:
print("String value can not be entered")
print("or number is negative, restart the program!")
input ("reload the program!")
else:
ar+=a; count+=1
ar=ar/count; print("Average: ",ar)
```

Варіанти завдань до лабораторної роботи

1-13. Напишіть програму, яка обчислює значення визначеної функції (на вхід подають дійсні числа).

1. Задано функцію S з параметрами $\alpha=l$, $\beta=0{,}5(l+r)$, $\gamma=r$ ($l<r$) вигляду:

$$S(x;l,r)=\begin{cases}0, & x\le l;\\ 2\left(\dfrac{x-l}{r-l}\right)^2, & l<x<\dfrac{l+r}{2};\\ 1-2\left(\dfrac{r-x}{r-l}\right)^2, & \dfrac{l+r}{2}<x\le r;\\ 1, & r<x.\end{cases}$$



2. Задано функцію Z з параметрами l та r ($l<r$), яку визначають через функцію S:

$$Z(x;l,r)=1-S(x;l,r),\ \text{де}\ S(x;l,r)=\begin{cases}0, & x\le l;\\ 2\left(\dfrac{x-l}{r-l}\right)^2, & l<x<\dfrac{l+r}{2};\\ 1-2\left(\dfrac{r-x}{r-l}\right)^2, & \dfrac{l+r}{2}<x\le r;\\ 1, & r<x.\end{cases}$$

3.Задано функцію π з параметрами a,c, яку визначають через функції S, Z:

$$\pi(x;a,c)=\begin{cases}S(x;c-a,c), & x\le c;\\ Z(x;c,c+a), & x>c;\end{cases}$$

$$\text{де}\ S(x;l,r)=\begin{cases}0, & x\le l;\\ 2\left(\dfrac{x-l}{r-l}\right)^2, & l<x<\dfrac{l+r}{2};\\ 1-2\left(\dfrac{r-x}{r-l}\right)^2, & \dfrac{l+r}{2}<x\le r;\\ 1, & r<x;\end{cases}\qquad Z(x;l,r)=1-S(x;l,r)$$

4. Задано функцію Гаусса з параметрами c_1, c_2, σ_1, σ_2 вигляду:

$$ts\_gaussian(x;c_1,\sigma_1,c_2,\sigma_2)=\begin{cases} exp\left[-\frac{1}{2}\left(\frac{x-c_1}{\sigma_1}\right)^2\right], & x\le c_1;\\ 1, & c_1<x<c_2;\\ exp\left[-\frac{1}{2}\left(\frac{x-c_2}{\sigma_2}\right)^2\right], & c_2\le x. \end{cases}$$

5. Задано функцію y з параметром b вигляду:

$$y(x,b)=\begin{cases} -x^2+b, & \text{якщо } x<0 \text{ та } b\neq 0;\\ \frac{x}{x-3}+5, & \text{якщо } x>0 \text{ та } b=0;\\ \frac{x}{-3} & \text{інакше.} \end{cases}$$

6. Задано функцію $y(x)$ вигляду: $y(x)=\begin{cases} 1, & \text{якщо } x\le 0;\\ cos(x), & \text{якщо } 0<x\le pi;\\ -1 & \text{інакше.} \end{cases}$

7. Задано функцію ts_π з параметрами $\{a, b, c, d\}$, яку визначають через функції S і Z:

$$ts\_\pi(x;a,b,c,d)=\begin{cases} 0, x\le a;\\ S(x;a,b), a<x<b;\\ 1, b\le x\le c;\\ Z(x;c,d), c<x<d;\\ 0, d\le x. \end{cases}$$

де $Z(x;l,r)=1-S(x;l,r)$, $S(x;l,r)=\begin{cases} 0, x\le l;\\ 2\left(\frac{x-l}{r-l}\right)^2, l<x<\frac{l+r}{2};\\ 1-2\left(\frac{r-x}{r-l}\right)^2, \frac{l+r}{2}<x\le r;\\ 1, r<x. \end{cases}$

8. Задано функцію *triangle* з трьома параметрами $\{a, b, c\}$, де $a < b < c$, вигляду:

$$\mu(x) = triangle(x; a, b, c) = \begin{cases} \dfrac{x-a}{b-a}, & a \le x \le b; \\ \dfrac{c-x}{c-b}, & b \le x \le c; \\ 0, & x \le a \text{ або } c \le x. \end{cases}$$

9. Задано функцію *trapezoid* з чотирма параметрами $\{a, b, c, d\}$, де $a < b \le c < d$, вигляду:

$$\mu(x) = trapezoid(x; a, b, c, d) = \begin{cases} \dfrac{x-a}{b-a}, & a \le x \le b; \\ 1, & b \le x \le c; \\ \dfrac{d-x}{d-c}, & c \le x \le d; \\ 0, & d \le x \text{ або } x \le a. \end{cases}$$

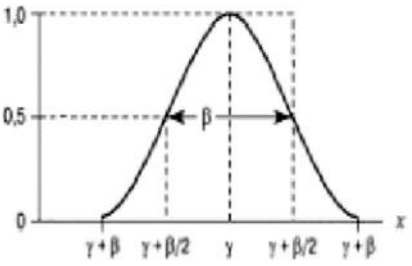
10. Задано функцію S з параметрами α, β, γ ($\alpha<\beta<\gamma$) вигляду:

$$S(x; \alpha, \beta, \gamma) = \begin{cases} 0, & \text{для } x \le \alpha \\ 2\left(\dfrac{x-\alpha}{\gamma-\alpha}\right)^2, & \text{для } \alpha \le x \le \beta \\ 1-2\left(\dfrac{x-\gamma}{\gamma-\alpha}\right)^2, & \text{для } \beta \le x \le \gamma \\ 1, & \text{для } x \ge \gamma \end{cases}$$

11. Задано функцію π на основі функції S з параметрами $\{\beta, \gamma\}$ вигляду:

$$\Pi(x;\ \beta,\ \gamma) = \begin{cases} S(x; \gamma-\beta, \gamma-\beta/2, \gamma), & \text{якщо } x \le \gamma \\ 1-S(x; \gamma, \gamma+\beta/2, \gamma+\beta), & \text{в інших випадках} \end{cases}$$

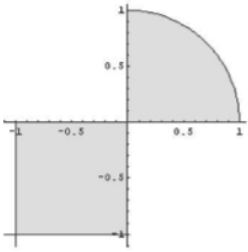
П-функція досягає нуля в точках $x = \beta \pm \gamma$ (параметр β визначає загальну ширину), а координати точок перетину функції з прямою $\pi = 0{,}5$ визначають як $x = \beta \pm \gamma / 2$.



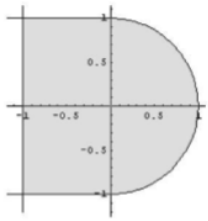
12. Задано функцію y з параметром b вигляду:

$$y(x,b)=\begin{cases} 2x^2+b, & \text{якщо } x<1 \text{ та } b\neq 0; \\ \dfrac{x}{4x-1}+6, & \text{якщо } x>1 \text{ та } b=0; \\ \dfrac{x}{-2} & \text{інакше.} \end{cases}$$

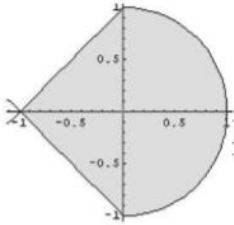
13-**25**. Напишіть програму, яка визначає, чи потрапляє задана точка (x, y) всередину вказаної області:



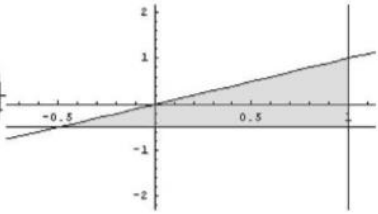
13



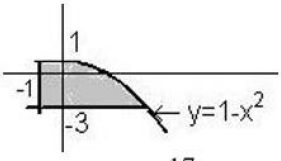
14



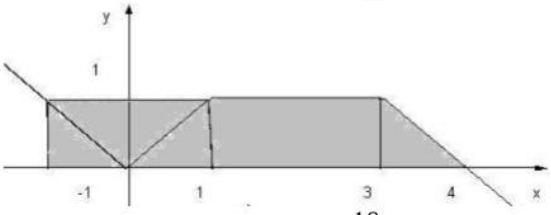
15



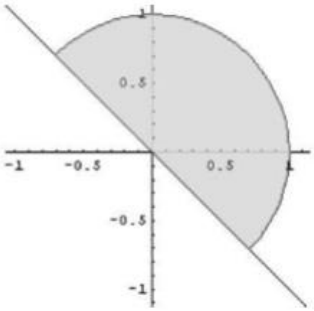
16



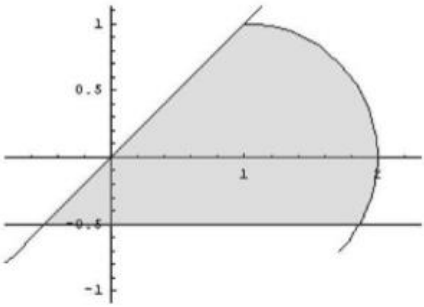
17



18



19



20

21. Нехай задано числа a_1, b_1, c_1, a_2, b_2, c_2. Вивести значення координат точки перетину прямих, які описують рівняння $a_1x+b_1y=c_1$ і $a_2x+b_2y=c_2$, або повідомити, що ці прямі співпадають, не перетинаються або не існують.

22. Нехай задано числа a, b, c - коефіцієнти рівняння $ax^2+bx+c=0$ ($a\neq0$). Вивести значення коренів $x1$, $x2$ (якшо вони існують) та змінної t=**true,** інакше t=**false** (якщо коренів немає, повідомити про це).

Контрольні питання

1. Що таке алгоритм розгалуженої структури?
2. В чому полягає різниця між умовними операторами з однією та двома гілками?
3. Що таке логічне висловлювання? Назвіть логічні операції, які використовують в логічних висловлюваннях при написанні програм на мові Python.
4. Який синтаксис має оператор розгалуження (множинного розгалуження)?

ЛАБОРАТОРНА РОБОТА №6

Тема: Алгоритми циклічної структури (інструкція while)

Мета: ознайомитися з алгоритмами циклічної структури, їх реалізацією на мові Python; вивчити формат оператору циклу while.

Час виконання роботи: 2 години.

Програмне забезпечення: IDLE (Python 3.X)..

Короткі теоретичні відомості

Рядки

Рядки - це впорядковані послідовності символів, що використовують для зберігання і подання текстової інформації (символів і слів, наприклад, ваше ім'я, змісту текстових файлів, завантажених в пам'ять, адрес в Інтернеті, програми на Python тощо). Рядки володіють потужним набором засобів для їх обробки. У Python відсутній спеціальний тип для подання одного символу, тому в разі необхідності використовуються односимвольні рядки.

Рядки відносять до категорії незмінних послідовностей, в тому сенсі, що символи, які вони містять, мають певний порядок розміщення зліва направо і самі рядки неможливо змінити. Рядки – це представник великого класу об'єктів, які називають послідовностями. Зверніть увагу на операції над послідовностями, наведені тут, тому що вони схожим чином працюють і з іншими типами послідовностей, такими як списки і кортежі, які ми будемо розглядати пізніше. У табл. 6.1 наведені найбільш типові літерали рядків і операцій.

Таблиця 6.1. - *Типові літерали рядків та операції над ними*

Операція	Інтерпретація
S = ‘’‘	Пустий рядок
S = “spam‘s”	Рядок у лапках
S = ‘s\np\ta\x00m‘	Екрановані послідовності
block = “’...””	Блоки в потрійних лапках

Операція	Інтерпретація
S = r‘\temp\spam‘	Неформатовані рядки
S = b‘spam‘	Рядки байтів
S1 + S2 S * 3	Конкатенація, повторення
S[i] S[i:j] len(S)	Звернення до символу за індексом Витяг підрядку (зрізу) Довжина
“a %s parrot” % kind	Вираз форматування рядка
“a {0} parrot”.format(kind)	Метод форматування рядка
S.find(‘pa‘)	Виклик методу рядків: пошук
S.rstrip() S.replace(‘pa‘, ‘xx‘) S.split(‘,‘)	Видалення провідних й кінцевих символів пробілу Заміна Розбиття за символом, який є роздільником
S.isdigit()	Перевірка вмісту
S.lower()	Перетворення регістра символів
S.endswith(_spam‘)	Перевірка закінчення рядка
‘spam‘.join(strlist)	Формування рядка зі списку
S.encode(‘latin-1‘)	Кодування рядків Юнікоду
for x in S: print(x)	Обхід в циклі
‘spam‘ in S [c * 2 for c in S] map(ord, S)	Перевірка на входження

Порожній рядок має вигляд пари лапок (або апострофів), між якими нічого немає. Для роботи з рядками підтримуються операції над виразами, такі як конкатенація (об'єднання рядків), виділення підрядка, вибірка символів за індексами (за зсувом від початку рядка) тощо. Python пропонує ряд методів, які реалізують різні завдання роботи з рядками.

Цикл while

Інструкція while. Алгоритм, в якому передбачено неодноразове виконання певної послідовності дій, називається алгоритмом циклічної структури або циклом. Цикл дозволяє істотно скоротити розмір запису алгоритму, зобразити його компактно шляхом відповідної організації пропонуємих дій. Повторювати певні дії має сенс при різних значеннях параметрів, які змінюються. Такі параметри називаються *параметрами циклу*. Блок повторюваних операторів називають *тілом циклу* (це послідовність дій, які виконується багаторазово).

Циклічний процес називається ітераційним, якщо заздалегідь невідома кількість повторень циклу, а кінець обчислення визначається при досягненні деякою величиною заздалегідь заданої точності обчислення.

При програмуванні ітераційних процесів прийнято їх розділяти на цикли з «передумовою» і з «післяумовою». Їх відмінність полягає в тому, що перевірка досягнення деякою величиною заданої точності обчислення здійснюється або на початку циклу, або наприкінці циклу відповідно. Особливість циклу з «післяумовою» полягає в тому, що повторювана ділянка алгоритму виконається хоча б один раз, у той час як в циклі з «передумовою» ця ділянка може не виконатися жодного разу. Процес ініціалізації включає в себе визначення (введення) початкових значень змінних, які використовуються в тілі циклу.

Для запису ітераційних процесів в Python використовують лише один тип операторів циклу **while** - оператор з попередньою умовою (передумовою). Для всіх операторів циклу характерні такі особливості:

- Повторювані обчислення записуються лише один раз.
- Вхід в цикл можливий тільки через його початок.

– Змінні оператора циклу повинні бути визначені до входу в цикл.

– Потрібно передбачити вихід з циклу. Якщо цього не зробити, то обчислення будуть тривати нескінченно довго.

Нескінченний цикл – це циклічна ділянка в програмі, в якій не передбачені засоби виходу з циклу при досягненні деякого умови. Слово "*while*" з англійської мови перекладається як "доки" ("доки логічний вираз має значення **True**, доти будуть виконуватись певні операції"). Конструкцію **while** мовою Python можна описати наступною схемою:

```
while a логічний оператор b:
    дії (або дія)
    зміна a
```

Ця схема приблизна, тому що логічний вираз в заголовку циклу **while** може бути більш складним, а змінюватися може змінна (або вираз) **b**.

Може виникнути питання: "*Навіщо змінювати **a** або **b**?*". Коли виконання програмного коду доходить до циклу **while**, виконується логічний вираз в заголовку, і, якщо було отримано **True**, виконуються вкладені вирази. Після потік виконання програми знову повертається в заголовок циклу **while**, і знову перевіряється умова. Якщо умова ніколи не буде змінюватись, то не буде причин зупинки циклу і програма *зациклиться*. Щоб цього не сталося, необхідно передбачити можливість виходу з циклу. Таким чином, змінюючи значення змінної в тілі циклу, можна довести логічний вираз до **False** і цикл завершиться.

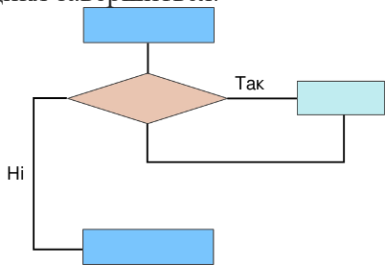


Рис.6.1. Функціональна схема циклу «Поки» (з передумовою)

Цю змінювану змінну, яка використовується в заголовку циклу **while**, зазвичай називають *лічильником*. Як і всякій змінній їй можна давати довільні імена, однак дуже часто використовують букви ***i*** та ***j***. Найпростіший цикл на мові програмування Python може виглядати так:

```
str1 = "+"
i = 0
while i < 10:
    print(str1)
    i = i + 1
```

В останньому рядку коду відбувається збільшення значення змінної ***i*** на одиницю, тому з кожним колом циклу її значення збільшується. Коли буде досягнуте число 10, логічний вираз **i < 10** дасть **False**, виконання тіла циклу буде припинено, а потік виконання програми перейде на наступні команди, що слідують після циклу. Результатом виконання вищевказаного скрипту є виведення на екран десяти знаків “+” в стовпчик. Якщо збільшувати лічильник в тілі циклу не на одиницю, а на 2, то буде виведено тільки п'ять знаків, так як цикл зробить лише п'ять обертів. Більш складний приклад з використанням циклу:

```
fib1 = 0
fib2 = 1
print (fib1)
print (fib2)
n = 10
i = 0
while i < n:
    fib_sum = fib1 + fib2
    print(fib_sum)
    fib1 = fib2
    fib2 = fib_sum
    i = i + 1
```

Цей приклад виводить *числа Фібоначчі* - ряд чисел, в якому кожне наступне число дорівнює сумі двох попередніх: 0, 1, 1, 2, 3, 5, 8, 13 і т.д. Скрипт виводить дванадцять членів ряду: два (0 і

1) виводяться поза циклом і десять виводяться в результаті виконання циклу.

Як це відбувається? Вводяться дві змінні (**fib1** та **fib2**), яким присвоюються початкові значення. Присвоюються значення змінній n та лічильнику i, між якими ті чи інші математичні відносини формують бажане число витків циклу. Всередині циклу створюється змінна **fib_sum**, якій присвоюється сума двох попередніх членів ряду, і її ж значення виводиться на екран. Далі змінюються значення **fib1** та **fib2** (першому присвоюється друге, а другому - сума), а також збільшується значення лічильника.

Інструкція **while** організує цикл з передумовою (перевірка виконується перед початком чергової ітерації), складається з рядка заголовка з умовним виразом, тіла циклу, що містить одну або більше інструкцій з відступами, і необов'язкової частини **else**, яка виконується, коли управління передається за межі циклу без використання інструкції **break**. Інтерпретатор продовжує обчислювати умовний вираз в рядку заголовка і виконувати вкладені інструкції в тілі циклу, поки умовний вираз не поверне значення «хибність»:

while <test>: # Умовний вираз test
<statements1> # Тіло цикла
else: # Необов'язкова частина else
<statements2> # Виконується, якщо вихід із цикла
#виконується не інструкцією break

Оператор **while** дозволяє багаторазово виконувати певні дії в залежності від деякої <умови> (виразу логічного типу, який приймає тільки значення **True** або **False**). Цикл виконується поки <Умова>=«істина». Як тільки <Умова> порушується, виконання циклу завершується. Блок-схема циклу **while** наведена на рис. 6.1.

Інструкція **while** продовжує виконувати блок інструкцій (зазвичай з відступами), поки умовний вираз продовжує повертати значення «істина». Вона називається «циклом», тому що управління циклічно повертається до початку інструкції, поки умовний вираз не поверне значення «хибність». Як тільки в результаті перевірки буде отримано значення «хибність», управління буде передано першій інструкції, яка розташована

відразу ж за вкладеним блоком тіла циклу **while.** Розглянемо приклади простих циклів **while**.

Розглянемо ще приклад (який містить інструкцію **print**, вкладену в цикл **while**), нескінченно виводить повідомлення (**True** – це особлива версія цілого числа «1», вона позначає значення «істина», тому результатом цього умовного виразу завжди буде «істина», і інтерпретатор нескінченно буде виконувати тіло циклу, поки ви не скасуєте його виконання). Такі цикли зазвичай називають нескінченними:

```
>>> while True:
        print(_Type Ctrl-C to stop me!‘)
```

Наступний фрагмент продовжує видаляти з рядка перший символ, поки поки він не стане порожнім, в результаті умова прийме значення «хибність». Перевірка об‘єктів на значення «істина» здійснюється безпосередньо замість використання еквіваленту (while x != _‘:).

```
>>> x =’spam‘
>>> while x: # Поки x – це не пустий рядок
...   print(x, end=‘ ‘)
...   x = x[1:] # Видалити перший символ з x
... spam pam am m
```

Аргумент ***end= ‘’*** забезпечує виведення значень в один рядок через пробіл. Наступний фрагмент перебирає значення від a до b, не включаючи значення b:

```
>>> a=0; b=10
>>> while a < b:
# Один зі способів організації циклів перебору
...   print(a, end=‘’)
...   a += 1     # або a = a + 1
...
0 1 2 3 4 5 6 7 8 9
```

Python відсутній цикл «**do until**», однак його можна імітувати, додавши в кінець тіла циклу умовну інструкцію та інструкцію **break**:

```
while True:
...     <тіло цикла>...
        if exitTest(): break
```

Break, continue, pass і else. Розглянемо дві прості інструкції, які можна використати тільки усередині циклів – інструкції **break** і **continue**. У Python:

– **Break** виконує перехід за межі циклу (всієї інструкції циклу).

– **Continue** виконує перехід на початок циклу (в рядок заголовка).

– **Pass** нічого не робить: це пуста інструкція.

Блок **else** виконується, тільки якщо цикл завершився звичайним чином (без використання інструкції **break**).

З урахуванням інструкцій **break** і **continue** цикл **while** має такий загальний вигляд:

```
while <test1>:
      <statements1> # тіло циклу
   if <test2>: break # Вийти з циклу, пропустивши частину else
   if <test3>:continue #Перейти на початок циклу, до вираз.
test1
  else:
     <statements2>   # Виконується, якщо не була використана
                       #інструкція ‘break‘
```

Інструкції **break** і **continue** можуть з'являтися в будь-якому місці всередині тіла циклу **while** (або **for**), але як правило, їх використовують в умовних інструкціях **if,** щоб виконати необхідну дію у відповідь на деяку умову.

Pass. Інструкція **pass** не виконує ніяких дій, її використовують у випадках, коли синтаксис мови вимагає наявності інструкції, але ніяких корисних дій в цій точці програми виконати не можна. Вона часто використовується в якості порожнього тіла складної інструкції. Наприклад, створити нескінченний цикл, який нічого не робить, можна таким чином:

```
while 1: pass # Натисніть Ctrl-C, щоб припинитицикл!
```

Цей приклад нескінченно робить «ніщо». Ця інструкція може використовуватися, наприклад, для того, щоб ігнорувати виключення в інструкції **try**. Іноді інструкцію **pass** використовують як заповнювач, замість того, «що буде написано пізніше», і в якості тимчасового фіктивного тіла функцій:

```
def func1():
    pass # Реалізація функції буде додана пізніше
def func2(): pass
```

Порожнє тіло функції викличе синтаксичну помилку, тому в подібних ситуаціях можна використати інструкцію **pass.** У Python 3.0 замість будь-якого виразу допускається використовувати три крапки «**...**», які самі по собі не виконують жодної дії, тому їх можна використовувати як альтернативу інструкції **pass,** зокрема замість програмного коду, який буде написано пізніше (примітка: **To Be Done** – слід реалізувати):

```
def func1():
  ... # Альтернатива інструкції pass
def func2():
  ...
func1() # Під час виклику не виконає жодних дій
```

Три крапки може бути присутнім в одному рядку із заголовком інструкції і використовуватися для ініціалізації змінних, коли не потрібно вказувати значення якогось певного типу:

```
def func1(): ... #Може бути присутнім в тому ж рядку
def func2(): ...
>>>X = ... # Альтернатива об'єкту  None
>>>X
Ellipsis
```

Continue. Інструкція **continue** виконує перехід на початок циклу. Вона іноді дозволяє уникнути використання вкладених інструкцій. У наступному прикладі цю інструкцію використовують для пропуску непарних чисел (цей фрагмент виводить парні числа менше «10» і більше або рівні «0»). Число «0» означає «хибність», а оператор **%** обчислює залишок від

ділення, тому даний цикл виводить числа в зворотному порядку, пропускаючи значення, кратні 2 (він виводить 8 6 4 2 0):

```
x = 10
    while x:
    x = x-1 # x -= 1
if x%2!=0: continue # непарне, пропустить вивеедення
print(x, end=‘ ‘)
```

Останній приклад виглядав би зрозуміліше, якби інструкція **print** була складовою інструкції **if:**

```
x = 10
while x:
    x = x-1
        if x % 2 == 0: # парне? - вывести
                print(x, end=‘ ‘)
```

Break. Інструкція **break** виконує негайний вихід з циклу. Програмний код, розташований в циклі за цією інструкцією не виконується, якщо ця інструкція запущена. Нижче наведено інтерактивний цикл, який виконує введення даних за допомогою функції **input** і вихід з циклу, якщо у відповідь на запит імені буде введена рядок «**stop**»:

```
>>> while 1:
...     name = input(_Enter name:‘)
...     if name == _stop‘: break
...     age = input(_Enter age: _)
... print(_Hello‘, name, _=>‘, int(age) ** 2)
...
Enter name:mel
Enter age: 40
Hello mel => 1600
Enter name:bob
Enter age: 30
Hello bob => 900
Enter name:stop
```

Цей приклад виконує перетворення рядка **age** в ціле число за допомогою функції **int,** перед тим як звести його до другого

степеня (це необхідно, тому що функція **input** повертає результат введення користувача у вигляді рядка).

Else. При об'єднанні з частиною **else** інструкція **break** дозволяє позбутися від необхідності зберігати прапор стану пошуку. Наступний фрагмент визначає, чи є додатне ціле число простим числом, виконуючи пошук дільників більше за значення «1»:

```
x = y // 2 # Для значень y>1
while x > 1:
        if y % x == 0: # залишок
        print(y, _has factor‘, x)
       break # Переступити блок else
  x -= 1
else: # Нормальне завершення цикла
print(y, _is prime‘)
```

Замість того щоб встановлювати прапор, який буде перевірений після закінчення циклу, досить вставити інструкцію **break** в місці, де буде знайдений дільник. При такій реалізації управління буде передано блоку **else**, тільки якщо інструкція **break** не була виконана, тобто коли з упевненістю можна сказати, що число є простим. Блок **else** циклу виконується також у разі, коли тіло циклу жодного разу не виконувалося, оскільки в цій ситуації інструкція **break** також не виконується. У циклах **while** це відбувається, коли перша ж перевірка умови в заголовку дає значення «хибність». Внаслідок цього в попередньому прикладі буде отримано повідомлення «**is prime**» (просте число), якщо спочатку ***x*** менше або дорівнює «1» (тобто, коли y=2). Блок **else** в циклах дозволяє обробити «інший» спосіб виходу з циклу, без необхідності встановлювати і перевіряти прапори або умови.

Цей приклад визначає прості числа, але недостатньо точно. Числа, менші «2», не вважають простими у відповідності із суворим математичним визначенням. Якщо бути більш точним, цей код також буде терпіти невдачі при від‘ємних значеннях і виконуватися успішно при використанні дійсних чисел без дробової частини. В Python замість оператора ділення / використовують оператор //, тому що тепер оператор / виконує

операцію «істинного ділення» (початкове ділення необхідно, щоб відсікти залишок!)

Нехай ми створюємо цикл пошуку деякого значення в списку і після виходу з циклу необхідно дізнатися, чи було знайдено це значення. Це завдання можна вирішити таким чином:

```
found = False
# прапор, щоб визначити, чи закінчився пошук успіхом
while x and not found:
        if match(x[0]): # шукане значення є першим?
             print('Ni'); found = True
else:
       = x[1:] # Видалити перше значення й повторити
if not found:
    print('not found')
```

Нижче наводиться еквівалентний фрагмент, де використано блок **else** в циклі:

```
while x: # Вийти, коли x спорожніє
if match(x[0]):
print(_Ni'); break # Вихід, в обхід блоку else
 x = x[1:]
else:
print(_Not found') #цей блок працює, якщо рядок x вичерпано
```

Порядок виконання роботи

1. Проаналізувати та відпрацювати приклади коду, що наведено у теоретичному матеріалі.
2. Вивчити теоретичні основи написання алгоритмів циклічної структури. Опрацювати приклади.
3. Побудувати блок-схему алгоритму вирішення завдання . Відповідно до свого варіанту

– визначити умови;

– за допомогою формул описати варіанти виконання необхідний дій;

– написати програму, яка розв'язує завдання, реалізуючи обробку помилок (виключень);

– організувати введення даних з клавіатури, виведення у консоль рядків або списків.

4. Скласти звіт і захистити його по роботі. Захист роботи включає в себе демонстрацію працездатності програми на різних вхідних даних, демонстрацію трасування виконання програми.

Вимоги: не можна використовувати власні функції; при розв‘язанні задачі треба використовувати вкладені цикли.

Приклад 6.1. Використання циклу **while**

6.1.1.

1) *n = 1*
while n<=10: # повторювати, поки n меньше або дорівнює десяти
print('n =',n) # виводимо поточне значення n
n += 1 # і збільшуємо його на 1
2) *x = 0 while x <= 0: # повторювати, поки x не буде додатним x = int(input('Введіть додатне число: ')) print('Ви ввели число', x)*

3) *print('Усі натуральні числа:') n = 1 while True: # нескінчений цикл print(n); n += 1*

4)*name = None # спочатку ми не знаємо імені користувача # нескінчений цикл while True: print('Меню:') print(—1. Ввести ім‘я\) print('2. Вивести привітання') print('3. Вийти') response = input('Виберіть пункт: ') print() if response == '1': name = input(—Введіть ваше ім‘я: —) elif response=='2': if name: #вітаємося з користувачем, якщо ім'я вже*
введено print('Привіт, ', name, '!', sep='') else: print('Я не знаю вашого імені.') elif response == '3': # оператор break завершує виконання циклу break #якщо користувач вибрав 3, то виходимо з циклу else: print('Неправильне введення.') print()

5) *while True: print('Введіть exit, щоб завершити цикл') response = input('> ') if response == 'exit': break*

6) *x = 0 while x < 10: x += 1 if x == 5: # пропускаємо число 5 continue print(' Поточне число дорівнює ', x)*

7) *x = 5 while x: # поки x не дорівнює нулю print(x); x -= 1 else: print('Цикл виконано') print('Кінцеве значення x:', x)*

8) *x = 3 while x: #цикл буде виконано 3 рази, якщо користувач не завершить його раніше x-=1 response=input('Введіь stop, щоб зупинити цикл (інакше що завгодно): ') if response=='stop': break else: # ця гілка буде виконана, якщо цикл не був перерваний print('Цикл завершився сам') print('Кінець програми')*

Варіанти завдань до лабораторної роботи

1. Обчисліть вираз: b*b*(a-b)/a
2. Обчисліть вираз: $\sqrt{a-b}$
3. Обчисліть вираз: $a\sqrt{b}$
4. Обчисліть вираз: $a\dfrac{b-a}{b+a}$
5. Обчисліть вираз: $\sqrt{a}+\sqrt{b}+\sqrt{a+b}$
6. Обчисліть вираз: $\sqrt{a-b}$ /c
7. Розробити метод **swapXY()**, що може створювати виняткову ситуацію, коли до функції передається параметр **null**. При необхідності викликається виняток **NullPointerException**. Розробіть код обробки цієї виняткової ситуації.
8. Обчислити факторіал числа n, що вводиться з консолі. Обробити всі можливі помилки користувача при введенні цього числа.
9. Реалізуйте клас, призначений для введення цілих чисел з консолі, передбачивши обробку помилок не коректного введення за допомогою винятків.
10. Обчисліть значення функції y = f(x) для x∈[a, b], y=ln(x-1). Обробити всі можливі помилки.

Контрольні питання

1. Що таке цикл, для чого його використовують?
2. Як описується та виконується циклічна інструкція **while**?
3. Як можна організувати нескінченні цикли? Наведіть декілька прикладів і поясніть їх.
4. Як можна вийти з нескінченних циклів? Що відбувається при запуску нескінченного циклу?
5. Чи може оператор циклу не мати тіла? Чому?
6. Для чого служать оператори переривання **break** та **continue**?

Зміст звіту

1. Мета роботи.
2. Завдання до роботи.
3. Відповіді на контрольні питання.
4. Текст розробленого програмного забезпечення.
5. Результати тестування: вхідні дані та результати роботи програми.
6. Висновки, що відображають особисто отримані результати виконання роботи, їх критичний аналіз.

До звіту додаються файли проекту.

РЕКОМЕНДОВАНА ЛІТЕРАТУРА

1. 10 Лучших IDE для Python. URL : https://python-scripts.com/ide-for-python.
2. Бизли Дэвид М. Python. Подробный справочник.СПб. Питер: Символ-Плюс.2012. 864 с.
3. Лутц М. Изучаем Python. 4-е изд. : пер. с англ. СПб. Питер.: Символ-Плюс.2010. 1280 с.
4. Мартелли А., Рейвенскрофт А., Холден С. Python. Справочник. Полное описание языка. 3-е изд.: Пер. с англ. М. : изд-во «Москва». 2018. 896 с.
5. Мізюк О. Путівник мовою програмування Python: Вивчення основ програмування для початківців. URL : http://pythonguide.rozh2sch.org.ua/.
6. Объектно-ориентированное программирование на Python. *Лаборатория линуксоида.* URL : https://younglinux.info/oopython.php.
7. Перегудова Т. С. Python на примерах. Практический курс. 2-е изд. Москва : Издательство «Наука и техника». 2017. 432 с.
8. Плас Дж. Вандер Python для сложных задач: наука о данных и машинное обучение. СПб.: Питер. 2018. 576 с.
9. Програмування на мові Python (3.x). Початковий курс. URL : https://sites.google.com/site/pythonukr/.
10. Прохоренок Н. Python 3 и PyQt 5. Разработка приложений. Питер: издательство «BHV-СПб». 2018. 832с.
11. Прохоренок Н. А., Дронов В. А. Python 3. Самое необходимое. Петербург: BHV- СПб. 2018. 608 с.
12. Саммерфилд М. Программирование на Python 3. Подробное руководство. СПб. Питер: Символ-Плюс. 2009. 608 с.
13. Mark Pilgrim. Dive Into Python 3. URL : https://diveintopython3.net/.

ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ТА ІНТЕРНЕТ-РЕСУРСИ

1. Середовище розробки Python. У вільному доступі. URL : http://www.python.org/.
2. Сайт, що містить у вільному доступі необхідні дистрибутиви, повну інформацію і уроки з програмування на мові Python. Інтерпретатор для Python можна використовувати як програмований високорівневий калькулятор. URL : http://www.enthought.com/products/epdlibraries.php.

ABSTRACT

The main goal of course “Programming” is to form basic knowledge of Python programming technology, skills to create modern software products using the object-oriented paradigm and to solve various engineering problems using a computer. The knowledge gained in the study of this discipline will provide professional training for specialists in the field of information technology.

Python is a multi-purpose and cross-platform programming language that allows you to write code that reads well. The relative laconicism of the Python language allows you to create a program that will be much shorter than its analog written in another language and can be run on various operating systems without any changes.

Programs written in the Python programming language can be either small scripts or complex systems. The language is used in many projects: BitTorrent, Ubuntu Software Center, Ubuntu Linux, Blender, GIMP, DropBox ,, World of Tanks, Wikipedia, Google, YouTube, Instagram, Yandex, Facebook and others. It is easy to use and free.

Methodological recommendations for the laboratory workshop are structured in accordance with the program of the course "Programming (Python)" and consist of six laboratory works covering a wide range of questions on the development of Python applications. Laboratory work logically continues the study of topics begun at lectures. All forms of laboratory exercises are designed to practice practical actions. The subject of work is aimed at studying both the basic foundations of programming and more complex topics. As an integrated development environment, a Python interpreter with an interactive IDLE shell is used. which is the most adapted for educational purposes. Tasks for laboratory work are built taking into account the requirements of the time and practical application in subsequent professional activities. Some practical tasks include self-study and implementation.

Methodological recommendations consist of one module, which includes theoretical information, progress of work, a list of questions, recommended literature and a reference book of basic concepts.

The content of the manual corresponds to the standard requirements of educational programs planned for university students, as well as for self-education.

СЛОВНИК: ОСНОВНІ ПОНЯТТЯ

Алгоритм
набір інструкцій, які описують порядок дій виконавця (комп'ютера, людини), для отримання результату розв'язання задачі за скінченну кількість дій.

Вихідний код
англ. *source code* - будь-який набір інструкцій або оголошень, написаних комп'ютерною мовою програмування у формі, що її може прочитати і модифікувати людина.

Включення
характерна особливість мови, яка дозволяє об'єднувати цикли і умовні перевірки, не використовуючи при цьому громіздкий синтаксис.

Змінні
є іменами, які посилаються на значення в пам'яті комп'ютера.

Генератор
об'єкт, який призначений для створення послідовностей. З допомогою генераторів можна проходити (**ітерувати**) по великим послідовностям без необхідності створення і збереження усієї послідовності у пам'ять відразу.

Інтерпретатор
комп'ютерна програма (або набір комп'ютерних програм), що перетворює (інтерпретує) вихідний код, написаний певною мовою програмування, на семантично еквівалентний код в іншій мові програмування, який, як правило, необхідний для виконання програми комп'ютером.

Ітерація
коли якусь дію необхідно повторити велику кількість разів, у програмуванні використовуються цикли. Один крок циклу називається ітерацією.

Ітератор
об’єкт, що перебирає усі елементи послідовності (переходить від одного елемента до іншого).

Клас
спеціальна конструкція, яка використовується для групування пов’язаних змінних та функцій.

Кортеж
(***tuple***) - послідовність будь-яких елементів. Кортеж аналогічний списку зі значеннями, які не змінюються.

Множина
(***set***) - **«контейнер»**, що містить унікальні елементи у випадковому порядку. Елементом множини може бути будь-який незмінний тип даних: числа, рядки, кортежі.

Мова програмування
мова, яка використовується для запису алгоритмів, призначених для виконання комп’ютером.

Модуль
файл, який містить код на Python. У цих файлах визначені функції, які можна використовувати у власних програмах.

Налагоджувач
(англ. *debugger*) — комп’ютерна програма, яка використовується для тестування і виправлення помилок інших програм.

Об’єкт
є окремою одиницею сховища даних під час роботи програм, яке використовується як базовий елемент побудови програм.

Операнд
аргумент операції; дані, які обробляються командою; граматична конструкція, яка позначає вираз, що задає значення аргументу операції.

Оператор
виконує дію над операндами.

Пакет
дозволяє згрупувати колекцію модулів під загальним ім'ям. Цей прийом дозволяє вирішити проблему конфліктів імен між іменами модулів, які використовуються у різних додатках.

Простори імен
(***namespaces***) - призначені для локалізації імен ідентифікаторів, і попередження їх конфліктів. За замовчуванням, усі ідентифікатори знаходяться у глобальному просторі імен, тому часті випадки існування двох різних об'єктів з однаковими іменами, що призводить до помилок.

Програмування
передання комп'ютеру інструкцій, які вказують йому щось зробити. Наприклад, обчислити математичний вираз, змінити текст, здійснити пошук інформації у файлах або виконати обмін даними з іншими комп'ютерами через мережу Інтернет тощо.

Програміст
фахівець, що займається програмуванням, виконує розробку програмного забезпечення (в простіших випадках – окремих програм) для програмованих пристроїв – комп'ютерів.

Програма
набір команд (вказівок, інструкцій), призначений для виконання комп'ютером у певній послідовності. Такий набір інструкцій називається **вихідним кодом**.

Псевдокод
неформальний запис алгоритму, який використовує структуру поширених мов програмування, але нехтує деталями коду, неістотними для розуміння алгоритму (опис типів, виклик підпрограм тощо). Псевдокод є зрозумілішим, ніж програми, формою запису алгоритмів.

Регулярні вирази
(від англ. *regular expression*) - рядок, що описує або збігається з множиною рядків, відповідно до набору спеціальних синтаксичних правил.

Розгалуження
конструкція мови програмування, що забезпечує виконання/невиконання певної команди (набору команд) за умови істинності/хибності деякого логічного виразу.

Список
(***list***) - послідовність елементів, розташованих у певному порядку. Доступ до елементів здійснюється за індексом. Значення елементів списку можна змінювати.

Словник
(***dict***) - **асоціативний масив** - послідовність неупорядкованих пар **ключ: значення** з вимогою унікальності ключів. На відміну від інших послідовностей, доступ до елементів словника здійснюється по ключу, а не за індексом, ключ може бути будь-якого типу, ключ не допускає змін.

Скрипт (сценарій)
програма, яка автоматизує деяке завдання.

Тип даних
визначає множину допустимих значень, формат їхнього збереження, розмір виділеної пам'яті та набір операцій, які можна виконати над ними.

Функція
частина програми, яка реалізує певний алгоритм і дозволяє звернення до неї з різних частин головної програми; функція повертає результат і може використовуватись як частина виразу.

Цикл
будь-яка багатократно виконувана послідовність команд.

Навчально-методичне видання

КОЗУБ Галина Олександрівна
СЕМЕНОВ Микола Анатолійович

ПРОГРАМУВАННЯ (PYTHON)

Методичні рекомендації до лабораторних робіт
для студентів спеціальності
121 – „Інженерія програмного забезпечення”

Редактор – Козуб Г. О.
Комп’ютерний макет – Козуб Г. О.

Здано до склад. 00.00.2019 р. Підп. до друку 00.00.2020 р.
Формат 60х84 1/16. Папір офсет. Гарнітура Times New Roman.
Друк ризографічний. Ум. друк. арк. 6,39. Наклад 50 прим.Зам. № 00.

Видавець і виготовлювач
Видавництво Державного закладу
„Луганський національний університет імені Тараса Шевченка”
пл. Гоголя, 1, м. Старобільськ, 92703. Тел./факс: (06461) 2-26-70.
e-mail: mail@luguniv.edu.ua
Свідоцтво суб’єкта видавничої справи ДК № 3459 від 09.04.2009 р.