МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО‖

О.В. Цеслів

Програмування для аналітичних досліджень

для студентів економічних спеціальностей

Навчальний посібник

Рекомендовано Методичною радою КПІ ім.Ігоря Сікорського
як навчальний посібник для здобувача ступеня бакалавра
За освітньою програмою Економічна аналітика
Спеціальності 051 Економіка

Електронне мережеве видання

Київ
КПІ ім.Ігоря Сікорського
2024

УДК 005.8:316.422

Автор	Цеслів Ольга Володимирівна
Рецензент	Стасюк Олександр Іонович, завідувач кафедри “Автоматизація та комп'ютерно-інтегровані технології транспорту”, Київський університет економіки і технології транспорту, д.т.н., професор.
Відповідальний редактор	Бояринова Катерина Олександрівна завідувач кафедри Економічної кібернетики ,д.е.н., професор

Гриф надано Методичною радою КПІ ім.Ігоря Сікорського
Протокол № 4 від 1.02.2024
за поданням вченої ради факультету маркетингу та маркетингу
протокол № 6 від 29.01.2024

Цеслів О.В.

Програмування для аналітичних досліджень[Електронний ресурс]: навч.посіб. для здобувачів ступеня бакалавр за освіт. програмою Економічна аналітика спец. 051 Економіка/ О.В.Цеслів; КПІ ім.Ігоря Сікорського – Електрон.текст.дані(1 файл)– Київ: КПІ ім.Ігоря Сікорського,2024. – 238с.

В даному навчальному посібнику викладено зміст курсу “Програмування для аналітичних досліджень” студентам другого курсу факультету менеджменту та маркетингу Національного технічного університету України «Київський політехнічний інститут». Викладено основи програмування мовою Python, алгоритмічні алгоритми, синтаксис мови та основи функціонального і модульного програмування. Матеріал супроводжується великою кількістю прикладів.

Призначений для здобувачів ступеня бакалавр за освітньою програмою “Економічна аналітика” спеціальності 051 Економіка

УДК 005.8:316.422

Реєстр № НП_____Обсяг ХХ авт арк
Національний технічний університет України
Київський політехнічний інститут імені Ігоря Сікорського
Проспект Берестейський,37, м.Київ,03056
http://kpi.ua
Свідоцтво про внесення до Державного реєстру видавців, виготовлювачів,
Розповсюджувачів видавничої продукції ДК №5354 від 25.05.2017

Зміст

ВСТУП

Мета вивчення дисципліни “Програмування для аналітичних досліджень” полягає в ознайомленні студентів з основами програмування мовою Python. В дисципліні вивчаються: алгоритми, синтаксис мови та основи функціонального і модульного програмування. Матеріал супроводжується великою кількістю прикладів.

Однією з переваг мови Python є, зокрема, наявність повної стандартної бібліотеки, що дозволяє задовольнити найбуденніші вимоги користувачів, наприклад, завантажити файл з Інтернету, розпакувати архів або створити веб-сервер за допомогою кількох рядків програмного коду. Крім того, існують тисячі додаткових бібліотек сторонніх виробників, які забезпечують складніші та потужніші можливості, наприклад, бібліотека для організації мережних взаємодій Twisted, бібліотека для виконання обчислювальних завдань Numpy або пакет моделювання Simpy. При цьому більшість сторонніх бібліотек можна знайти в Інтернеті. Саме ці обставини і спонукали авторів зупинити свій вибір на мові програмування Python.

Python може використовуватися для програмування у процедурному, об'єктно-орієнтованому і, меншою мірою, у функціональному стилі програмування, хоча загалом Python – це об'єктно-орієнтована мова. Основна мета цього підручника – надати студентам базові відомості про мову Python, необхідні для програмування прикладних задач, і допомогти навчитися писати процедурні й об'єктно-орієнтовані програми для виконання лабораторних робіт із числових методів.

РОЗДІЛ 1. АЛГОРИТМІЧНІ СТРУКТУРИ В МОВІ PYTHON

У 1936 році британський математик Алан Тюрінг запропонував математичну модель обчислювальної машини, яку назвали машиною Тюрінга. Крім того, він продемонстрував, що машина здатна виконувати будь-які обчислення, і ввів концепцію алгоритму – як послідовності дій, необхідних машині для досягнення результату. Сама комп'ютерна програма – це набір письмових команд, які вказують алгоритм у формі, зрозумілій машині. Щоб навчити комп'ютер чогось робити, потрібно спочатку скласти алгоритм.

Алгоритм - це скінченна послідовність точно визначених дій або операцій, спрямованих на досягнення поставленої мети.

Тобто, алгоритм - це система формальних правил, яка визначає зміст і послідовність операцій над вхідними даними та проміжними результатами, необхідних для отримання кінцевого результату при розв'язуванні задачі.

1.1.Властивості алгоритмів

1. Дискретність інформації. Кожний алгоритм має справу з даними: вхідними, проміжними, вихідними. Ці дані представляються у вигляді скінченних слів деякого алфавіту.

2. Дискретність роботи алгоритму. Виконується алгоритм по кроках, на кожному кроці виконується тільки одна операція.

3. Детермінованість алгоритму. Система величин, які отримуються в кожний (не початковий) момент часу, однозначно визначається системою величини, які були отримані в попередні моменти часу.

4. Елементарність кроків алгоритму. Закон отримання наступної системи величин з попередньої повинен бути простим та локальним.

5. Виконуваність операцій. В алгоритмі не має бути не виконуваних операцій. Наприклад, неможна в програмі призначити значення змінній "нескінченність", така операція була би не виконуваною. Кожна операція опрацьовує певну ділянку у слові, яке обробляється.

6. Скінченність алгоритму. Опис алгоритму повинен бути скінченним.

7. Спрямованість алгоритму. Якщо спосіб отримання наступної величини з деякої заданої величини не дає результату, то має бути вказано, що треба вважати результатом алгоритму.

8. Масовість алгоритму. Початкова система величин може обиратись з деякої потенційно нескінченної множини. Тобто, можливість виконання алгоритмів для рішення цілого класу конкретних задач, що відповідають загальній постановці задачі.

1.2.Способи написання алгоритмів

Існує чотири способи написання алгоритмів:

- вербальний (словесний);
- алгебраїчний (за допомогою літерно-цифрових позначень виконуваних дій);
- графічний;
- з допомогою алгоритмічних мов програмування.

Словесна форма запису алгоритмів використовується в різних інструкціях, призначених для виконання їх людиною.

Алгебраїчна форма найчастіше використовується у теоретичних дослідженнях фундаментальних властивостей алгоритмів.

Графічна форма відповідно до державних стандартів оформлення документації, прийнята як основна для опису алгоритмів.

Алгоритм, який за описаний за допомогою алгоритмічної мови програмування - це програма. Алгоритм у такій формі може бути введений у програму і після відповідного оброблення виконаний з метою отримання шуканого результату.

1.3.Мови програмування

Для подання алгоритму у вигляді, зрозумілому комп’ютеру, служать мови програмування. Спочатку завжди розробляється алгоритм дій, а потім він записується однією з таких мов.

У підсумку виходить текст програми – повний, закінчений і детальний опис алгоритму мовою програмування.

Потім цей текст програми спеціальними службовими додатками, які називаються трансляторами, або переводиться в машинний код, або виконується.

Мови програмування – штучні мови. Від природних вони відрізняються обмеженою кількістю "слів", значення яких зрозуміло транслятору, і дуже строгими правилами запису команд-операторів. Сукупність подібних вимог формує синтаксис мови програмування, а сенс кожної команди та інших конструкцій мови – його семантику [2].

Отже, програма, з якою працює процесор, являє собою послідовність чисел, яку називають машинним кодом. Такий запис містить лише номери команд процесора, необхідні дані та адреси комірок пам’яті. Наприклад (для зручності двійкові дані найчастіше записуються у шістнадцятковій формі, де 2 символи відповідають 1 байту даних – шістнадцяткова система числення є досить популярною у програмуванні):

BB 11 01 B9 0D 00 B4 0E 8A 07 43 CD 10 E2 F9 CD 20 48 65 6C 6C

Для написання таких програм застосовуються мови асемблера, які дозволяють записувати команди замість числової форми у текстовій (MOV, ADD, IN, OUT) та містять деякі найпростіші засоби для полегшення написання програми. Тим не менше, навіть в такому "прикрашеному" вигляді програма залишається надзвичайно близькою до машинного коду і тому мови асемблера відносять до низькорівневих мов програмування (тобто таких, що близькі до рівня машинного коду).

Незважаючи на незручність написання, такий код виконується з найвищою швидкістю і може бути максимально оптимізований, так як

програміст має доступ буквально до кожного біту у ньому. Тому найчастіше низькорівневі мови застосовуються для програмування мікросхем та окремих дій у прикладних програмах, де швидкодія є критичною.

Асемблер – це програма, яка перетворює код, написаний мовою асемблера, остаточно у машинний код. Але часто і саму мову називають скорочено "асемблером".

Проте, більшість програм пишеться на високорівневих алгоритмічних мовах програмування. Такі мови зазвичай мають складний синтаксис, використовують слова-оператори близькі до людської мови і – що найголовніше – реалізують алгоритмічні структури для простого і зрозумілого запису програми.

З іншого боку, для виконання комп’ютером така програма має бути перетворена на машинний код або хоча б переписана низькорівневою мовою (а далі вже асемблер забезпечить її розуміння машиною). Причому команди високорівневої мови програмування можуть бути досить складними і відповідати кільком або навіть кільком десяткам машинних команд. Цей процес перетворення називається **трансляцією**, а програми, які його виконують – трансляторами.

Існує два типи трансляторів, що перетворюють вихідний код програм в машинні команди: **інтерпретатори та компілятори**.

Інтерпретатор зчитує вихідний код програми по одній інструкції і, в найпростішому випадку, одразу намагається їх "перекладати" та виконувати. Це дозволяє програмісту швидше перевіряти виконання програми та знаходити помилки в коді. Крім того, така програма може бути легко перенесена на іншу машину і, якщо там є потрібний інтерпретатор, виконана ним –незалежно від операційної системи та процесора. А різницю між особливостями різних комп’ютерів покриває сам інтерпретатор, який, звичайно, буде трохи відрізнятися. Логічно, що в такій схемі виконання програми буде займати трохи більше часу – так як при цьому кожного разу відбувається аналіз коду та його перетворення.

Тому для підвищення швидкодії більшість сучасних інтерпретаторів насправді працює за змішаною схемою, спочатку транслюючи вихідний код програми у деяку проміжну форму –так званий **байт-код**. Він є кодом нижчого рівня і ближчий до асемблеру, але машинно-незалежний – тому виконується не безпосередньо комп'ютером, а деякою віртуальною машиною, яка входить до складу інтерпретатора. Це дозволяє, за відсутності змін в оригінальній програмі, не перечитувати її повністю, а використовувати байт-код як "напівфабрикат" для роботи.

Виконання байт-коду все одно повільніше ніж машинного коду, але такий підхід є компромісом, що намагається поєднати переваги інтерпретації та компіляції.

На відміну від інтерпретаторів, компілятор повністю перетворює вихідний код програми в машинний код, який операційна система може виконати самостійно. Це дозволяє виконувати скомпільовані програми навіть на тих комп'ютерах, на яких немає компілятора. Проте, скомпільована програма прив'язується до операційної системи і набору команд процесора, тому не завжди може бути перенесена і виконана на іншому комп'ютері. Крім того, такі програми виконуються швидше за рахунок того, що комп'ютеру не доводиться кожен раз перед запуском програми виконувати її розбір і перетворення в зрозумілий для себе вигляд. Однак, при сучасних потужностях комп'ютерів і обсягах пам'яті різниця в швидкості виконання програм інтерпретаторами і компіляторами вже майже непомітна, але процес розробки і налагодження програм на інтерпретованих мовах набагато простіший.

Програмування – досить складний процес, і цілком природно, коли програміст припускається помилки. Так повелося, що програмні помилки називають "багами" (від англ. bug – жучок). Процес виявлення і усунення помилок в англомовній літературі прийнято позначати терміном debugging. Процес пошуку помилок в програмі називається тестуванням (testing), процес усунення помилок –налагодженням (debugging).

1.4.Типи помилок

Існує три типи помилок, які можуть виникнути в програмах: синтаксичні помилки, помилки виконання і семантичні помилки.

1.5.Особливості мови програмування Python

Python – молода сценарна мова, історія якого почалася в 1990 році, коли співробітник голандського інституту CWI, тоді ще мало кому відомий Гвідо ван Росум приймав участь в проекті створення мови ABC. Ця мова була призначена для заміни мови BASIC в навчанні студентів основних концепцій програмування.

Паралельно з роботою над основним проектом Гвідо ван Росум вдома на своєму Macintosh написав інтерпретатор іншої простої мови але деякі принципи мови ABC все ж були запозичені.

На честь англійського колективу комічних акторів (яких дуже полюбляв Гвідо) "Monty Python’s Flying Circus" було названо мову та почалося її розповсюдження мережею Internet.

Мова почала швидко розвиватися, оскільки з’явилася велика кількість людей, що були зацікавлені та розумілися в розвитку мов програмування. Спочатку це була досить проста мова, невеликий інтерпретатор, незначна кількість функцій, об’єктно-орієнтоване програмування було відсутнім, але дуже швидко все це з’явилося та до сьогоднішнього дня продовжується її розвиток та виходять нові версії, де кожна наступна має декілька суттєвих відмінностей від попередньої.

Інтерпретатори Python існують під всі можливі платформи: Windows. UNIX та ін. Всі вони розповсюджуються безкоштовно. Python є однією з десяти найпопулярніших мов програмування.

Можна знайти велику кількість додатків, написаних на Python, наприклад:

- командний рядок на моніторі або у вікні терміналу;
- призначені для користувача інтерфейси, включаючи мережеві;

- веб-додатки, як клієнтські, так і серверні;
- бекенд-сервери, що підтримують великі популярні сайти;
- хмари (сервери, керовані сторонніми організаціями);
- додатки для мобільних пристроїв;
- додатки для вбудованих пристроїв;
- призначені для роботи з xml/html файлами;
- додатки призначені для роботи з http запитами;
- призначені для роботи із зображеннями, аудіо та відео файлами;
- додатки призначені для роботи з математичними та науковими розрахунками.

Хоча Python є досить швидким для більшості застосунків, проте його швидкості може виявитися не завжди недостатньою. Якщо програма проводить більшу частину часу за обчисленнями ("обмежена швидкодією процесора" (CPU-bound)), то мови C, C++ або Java впораються із завданням набагато краще, ніж Python. Але не завжди.

Іноді більш якісний алгоритм (покрокове рішення) для Python перевершує за швидкістю неефективний алгоритм для C. Більш висока швидкість розробки для Python дає більше часу для експериментів над альтернативними рішеннями.

Стандартний інтерпретатор Python написаний на C і може бути поліпшений за допомогою додаткового коду.

Інтерпретатори для Python стають швидшими. Мова Java була дуже повільною, коли тільки з’явилася, і для її прискорення було витрачено багато часу і грошей. Мовою програмування Python не володіє ні одна корпорація, тому він поліпшується більш плавно.

Найбільша проблема, з якою можна наразі зіткнутися, – це вибір однієї з двох існуючих версій Python. Остання версія Python 2 має номер 2.7, вона ще довго буде підтримуватися, але версія Python 2.8 ніколи не вийде. Нова розробка буде вестися лише на Python 3. Python – інтерпретована мова програмування, що створює байт-код (файли з розширенням .py, які

з’являються у папці із текстами програм під час їх виконання) для більш швидкої роботи.

1.6.Основні алгоритмічні структури

Етапи розв’язання задачі. Під розв’язанням конкретної задачі розуміють не тільки визначення результатів за допомогою ЕОМ, а також підготовчу роботу, яку необхідно виконати для досягнення поставленої мети. Тому весь процес можна розбити на кілька етапів:

- постановка задачі;
- формалізація (математична постановка задачі);
- вибір методу розв’язку;
- алгоритмізація задачі;
- програмування;
- налагодження програми;
- розв’язок задачі на ЕОМ і аналіз результатів.

Найбільше розповсюджений спосіб опису алгоритму у виді блок-схеми, що являє собою графічну інтерпретацію логічної схеми рішення задач. Блок-схемою - називається графічне зображення алгоритму, коли окремі дії зображуються різними геометричними фігурами – блоками. Правила виконання схем алгоритмів регламентує ДСТУ 2938-94, графічні символи, що використовуються, регламентує ДСТУ 2940-94, ДСТУ 2941-94.

Графічні символи повинні мати порядкові номери, що проставляються в лівій частині верхньої сторони зображення. Графічні символи на схемі алгоритму з'єднуються лініями потоку інформації. Основний напрямок потоку йде зверху вниз і з ліва на право. Вихідна лінія може бути лише одна (виключення - блок перевірки логічних умов і блок модифікації). Лінію потоку інформації підводять, як правило, до середини графічного символу.

У блоках прийняті розміри:

A= 10, 15, 20,.... мм;

B=1,5A (допускається встановлювати B=2A).

При необхідності збільшення розмірів схеми алгоритму допускається збільшення розміру А на число, кратне 5.

При виконанні схем алгоритмів необхідно витримувати мінімальну відстань між рівнобіжними лініями потоку інформації 3 мм і 5 мм – між іншими символами.

Лінійні алгоритми. При складання схем алгоритмів необхідно відрізняти лінійні, розгалужені і циклічні алгоритми. Як правило, вони не використовуються в чистому виді і як правило схема алгоритму досить складної задачі являє собою композицію перерахованих видів алгоритмів.

Лінійним називається обчислювальний процес, у якому дії виконуються послідовно в природному і єдиному порядку проходження. Блокові символи в цій структурі розміщаються в тому ж порядку, у якому повинні бути виконані запропоновані дії.

В алгоритмі лінійної структури використовуються наступні блокові символи:

пуск (початок);

введення;

процес;

виведення даних;

кінець.

Приклад: обчислити висоти трикутника зі сторонами a, b, c, використовуючи формули

$$h_a = \frac{2}{a}\sqrt{p(p-a)(p-b)(p-c)}$$

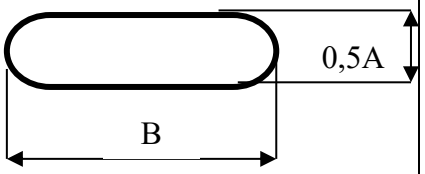
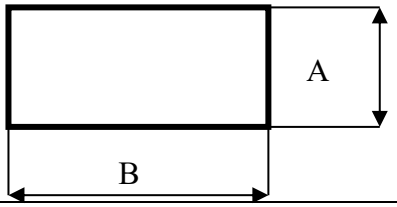
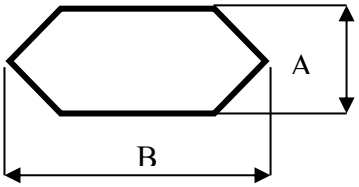
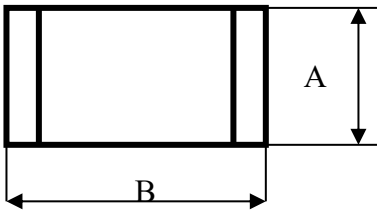
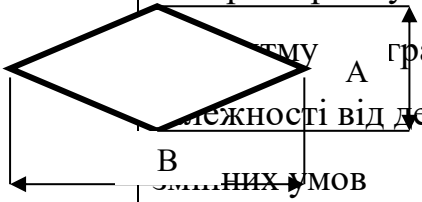
$$h_b = \frac{2}{b}\sqrt{p(p-a)(p-b)(p-c)}$$

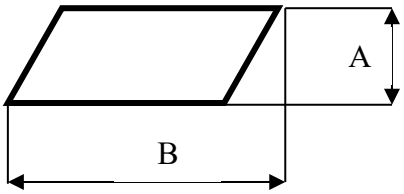
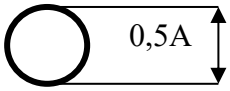
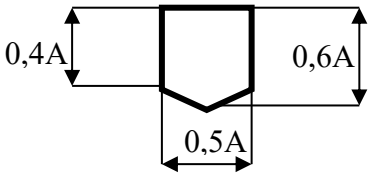
$$h_c = \frac{2}{c}\sqrt{p(p-a)(p-b)(p-c)}$$

$$p = \frac{a+b+c}{2}$$

Таблиця 1.1

Умовні графічні позначення, що застосовуються при складанні схем алгоритмів

Назва блоку	Графічне позначення блоку	Дії, що виконуються
Початок-зупинка		Початок, кінець, переривання процесу чи обробки виконання програми
Процес		Виконання операцій, в результаті якої змінюється значення, форма чи представлення даних
Модифікація (заголовок циклу)		Виконання операцій, що змінюють команди, групи команд, або програму
Визначений процес (підпрограма)		Використання раніше створених, або окремо написаних алгоритмів програм
Рішення		Вибір напрямку виконання [illegible]тму грами в [illegible]лежності від деяких [illegible]них умов

Введення виведення даних		Перетворення даних у форму, придатну для обробки (уведення), чи відображення результатів обробки (виведення)
З'єднувач сторінковий		Розрив лінії потоку інформації
З'єднувач між сторінковий		Розрив лінії потоку між сторінками

Щоб виключити числа, що повторюються, використовуємо проміжну величину

$$t = 2\sqrt{(p-a)(p-b)(p-c)}$$

$$h_a = t/a,\ h_b = t/b,\ h_c = t/c$$

Значення величин p, t, , , зберігаються в комірках пам'яті з відповідними іменами. Алгоритм обчислення представлений на рис. 1.

Розгалужені алгоритми. На практиці часто виникає необхідність, у залежності від отриманих вихідних даних, значень проміжних результатів, здійснювати обчислення по одним чи іншим формулам, тобто в залежності від виконання якої-небудь логічної умови обчислювальний процес повинен йти по одному чи іншому напрямку. Алгоритми, що містять дію вибору напрямку обчислювального процесу, мають назву розгалужених.

Розгалуження на блок-схемах представляється логічним блоком вибору. Умова розгалуження записується всередині блоку логічним відношенням або логічним виразом.

Логічне відношення - послідовний запис констант, змінних, арифметичних виразів, об'єднаних операціями відношення (>=;>;=;<>;<;<=).

Логічний вираз - послідовний запис логічних відносин, розділених знаками логічних операцій:

- логічного множення операції "И" (AND);
- логічного додавання операції "ЧИ" (OR);
- логічного заперечення операції "НЕ" (NOT).

Приклад 1.1. Обчислити корені квадратного рівняння.

$$ax^2 + bx + c = 0$$

$$x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \quad D = b^2 - 4ac \geq 0$$

$$x_{1,2} = \alpha \pm i\beta \quad D = b^2 - 4ac \leq 0$$

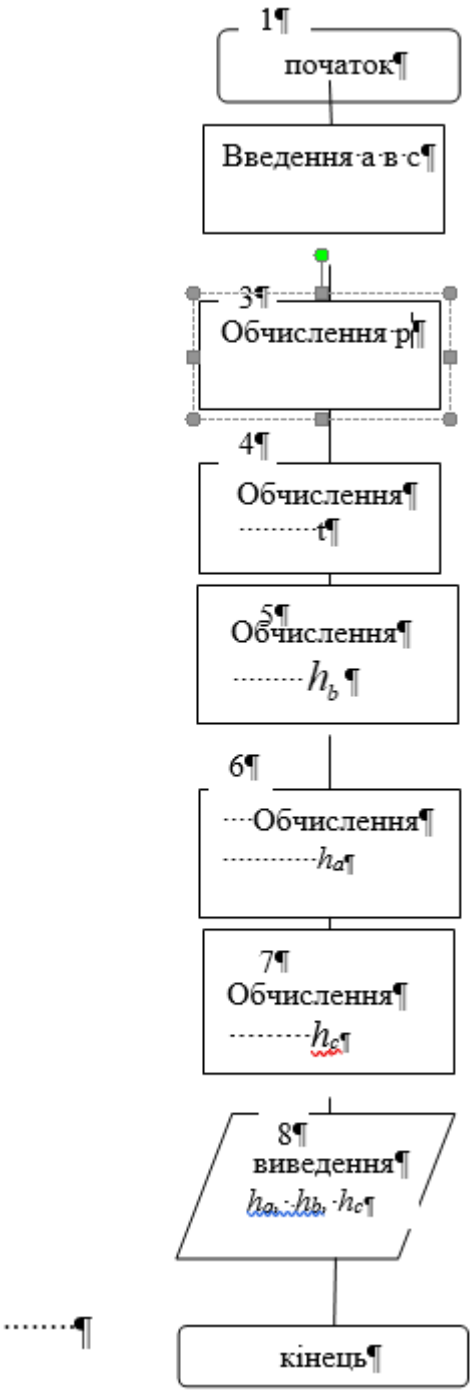


Рис. 1.1. Алгоритм лінійної структури

Основними алгоритмічними структурами є: слідування, розгалуження, цикл.

Слідування – команди виконуються послідовно одна за іншою.

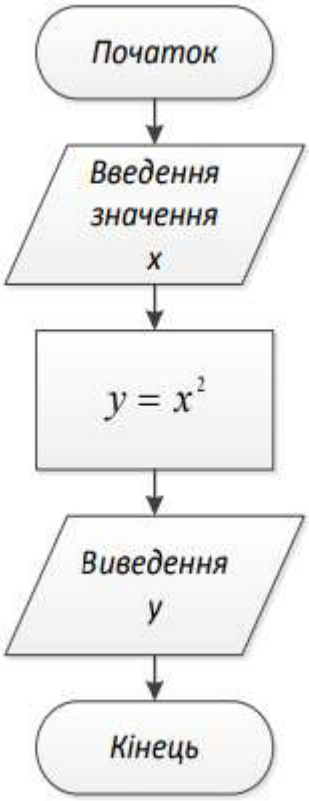


Рис.1.2. Приклад блок-схеми реалізації лінійного алгоритму

Розгалуження – алгоритм, що містить хоча б одну умову в результаті перевірки якої може виконуватись розділення на декілька паралельних гілок. Кожна з гілок може містити також розділення, послідовні дії або цикли.

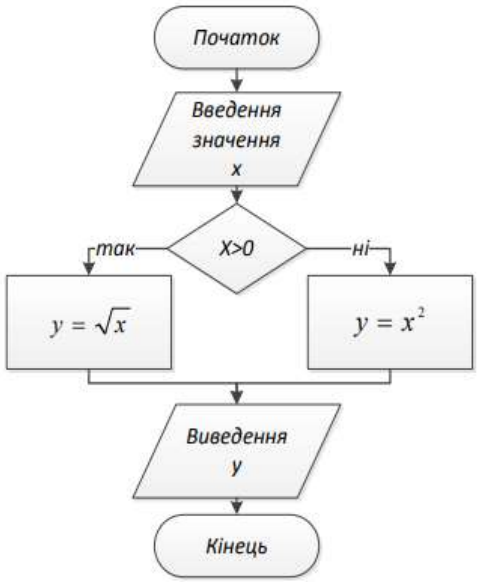


Рис.1.3 Приклад блок-схеми реалізації алгоритму з розгалуженням

Цикл – інструкції що виконують одну і ту ж послідовність дій поки діє задана умова.

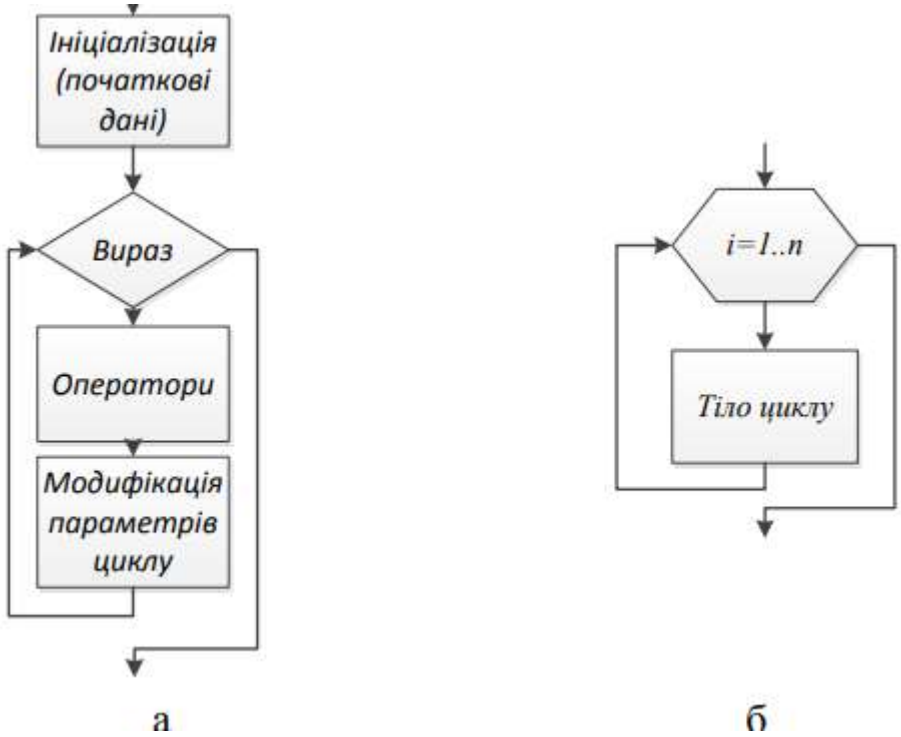


Рис.1.4. Приклад блок-схеми реалізації циклічного алгоритму (а – цикл з передумовою, б – цикл з лічильником)

1.7.Реалізація алгоритмів з розгалуженням

Хід виконання програми може бути лінійним, тобто таким, коли вирази виконуються, починаючи з першого і закінчуючи останнім, по порядку, не пропускаючи жодного рядка коду. Але частіше буває зовсім не так. При виконанні програмного коду деякі його ділянки можуть бути пропущені. Припустимо, в реальному житті людина живе за розкладом (можна сказати, розклад – це своєрідний "програмний код", який слід виконати). У її розкладі о 18.00 стоїть похід в басейн. Однак людині надходить інформація, що басейн не працює. Цілком логічно скасувати своє заняття з плавання. Тобто однією з умов відвідування басейну повинно бути його функціонування, інакше повинні виконуватися інші дії.

Схожа нелінійність дій може бути і в комп’ютерній програмі. Частина коду повинна виконуватися лише при певному значенні конкретної умови.

Найпростішою в Python для опису розгалужуючої структури, де дії виконуються лише у випадку істинності умови, є така конструкція:

IF ЛОГІЧНА УМОВА:

ПОСЛІДОВНІСТЬ ВИРАЗІВ

Цю конструкцію на блок-схемі можна зобразити на рис.1.3

Першим йде ключове слово if (англ. "Якщо"); за ним –логічний вираз; потім двокрапка, що позначає кінець заголовка оператора, а після неї – будь-яка послідовність виразів або тіло умовного оператора, яке буде виконуватися в разі, якщо умова в заголовку оператора істинна.

Приклад 1.2. Розгалудження.

```
x = 2
if x > 0:
 print("x – додатнє")
if x < 0:
 print("x – від’ємне")
```

Результатом виконання даного коду буде: x – додатнє

1. Привласнили значення 2 змінній x.
2. Зробили умовне порівняння за допомогою операторів if, виконуючи різні фрагменти коду в залежності від значень змінної x.
3. Викликали функцію print(), щоб вивести текст на екран.Рядки if в Python є операторами, які перевіряють, чи є значення виразу (в даному випадку змінна x) рівним True.

print() – це вбудована в Python функція для виведення інформації. Вбудовані функції Python – це іменовані фрагменти коду, які виконують певні операції.

Кожен рядок print() відокремлений пробілами під відповідною перевіркою.

У більшості мов програмування символи начебто фігурних дужок ({i}) або ключові слова begin і end застосовується для того, щоб розбити код на розділи. У цих мовах хорошим тоном є використання відбиття пробілами, щоб

зробити програму більш зрозумілою для себе та інших. Існують навіть інструменти, які допоможуть красиво вибудувати код.

Гвідо ван Росум при розробці Python вирішив, що виділення пробілами буде досить, щоб задати структуру програми і уникнути уведення дужок. Python відрізняється від інших мов тим, що пробіли в ньому використовуються для того, щоб задати структуру програми.

Як правило, використовують чотири пробіли для того, щоб виділити кожен підрозділ, хоча можна використовувати будь-яку кількість пробілів, Python чекає, що всередині одного розділу буде застосовуватися однакова кількість пробілів.

Рекомендований стиль – PEP-8 (http://bit.ly/pep-8) – використовувати чотири пробіли. Не рекомендується застосовувати табуляцію або поєднання табуляцій і пробілів – це заважає підраховувати відступи.

З огляду на це, в конструкції if код, який виконується при істинності умови, повинен обов'язково мати відступ вправо. Решта коду (основна програма) повинен мати той же відступ, що і слово if. Зустрічається і більш складна форма розгалуження: if-else.

Якщо умова при інструкції if є хибною, то виконується блок коду при інструкції else:

IF ЛОГІЧНА УМОВА:

ПОСЛІДОВНІСТЬ ВИРАЗІВ_1

else:

ПОСЛІДОВНІСТЬ ВИРАЗІВ_2

Цю конструкцію на блок-схемі можна зобразити:

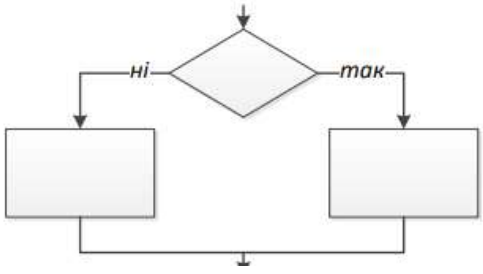


Рис.5. Блок-схема реалізації конструкції if-else

Працює ця конструкція наступним чином. Спочатку перевіряється перша умова і, якщо вона істинна, то виконується перша послідовність виразів. Якщо умова не виконується потік виконання переходить до рядка, який йде після else.

Приклад 1.3. Реалізації конструкції if-else

```
x = int(input("Введіть x= "))
if x > 0:
    y=x**0.5
else:
    y=x**2
print("y =", y)
```

Отримаємо:

```
Введіть x= 9
y = 3.0
```

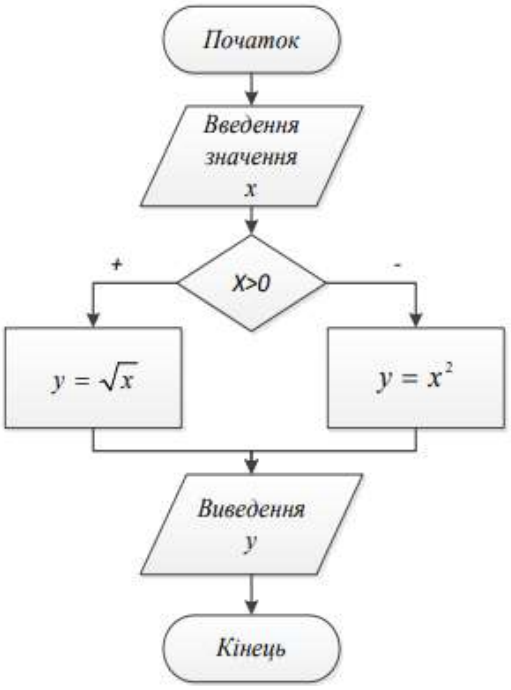


Рис.6. Приклад блок-схеми реалізації конструкції if-else

1.8.Альтернативні гілки програми

Логіка програми що виконується може бути складнішою, ніж вибір однієї з двох гілок.

Умовний оператор if має розширений формат, що дозволяє перевіряти кілька незалежних одна від одної умов і виконувати один з блоків, поставлених у відповідність з цими умовами. У загальному вигляді оператор виглядає так:

IF ЛОГІЧНА УМОВА_1:

ПОСЛІДОВНІСТЬ ВИРАЗІВ_1

else ЛОГІЧНА УМОВА_2:

ПОСЛІДОВНІСТЬ ВИРАЗІВ_2

elif ЛОГІЧНА УМОВА_3:

ПОСЛІДОВНІСТЬ ВИРАЗІВ_3

…

else:

ПОСЛІДОВНІСТЬ ВИРАЗІВ_N

Цю конструкцію на блок-схемі можна зобразити:

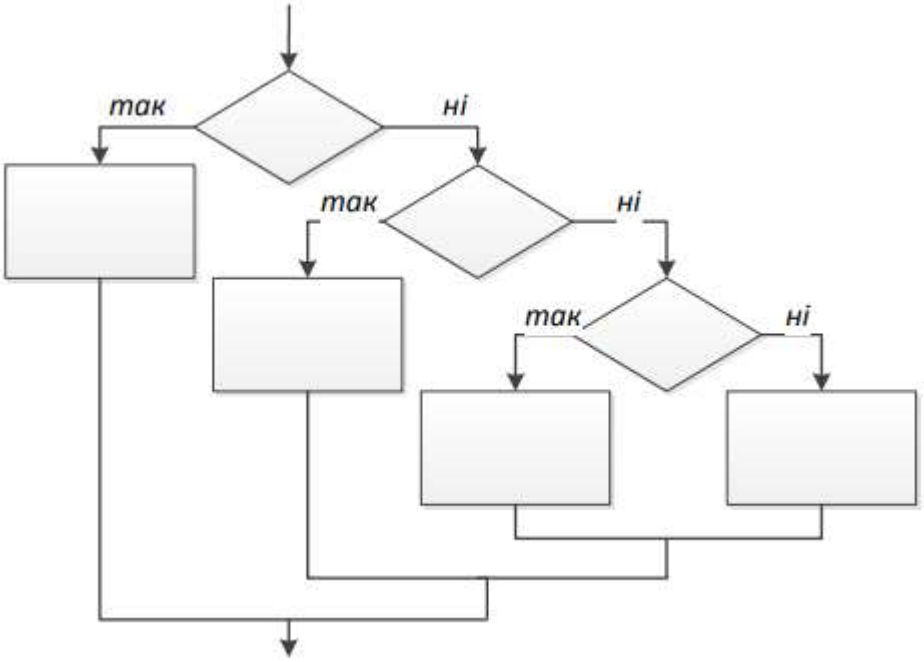


Рис.7. Блок-схема реалізації конструкції if-elif-else.

Працює ця конструкція наступним чином. Спочатку перевіряється перша умова і, якщо вона істинна, то виконується перша послідовність виразів.

Після цього потік виконання переходить до рядку, який йде після умовного оператора (тобто за послідовністю виразів N). Якщо перша умова рівна False, то перевіряється друга умова (наступна після elif), і в разі його істинності виконується послідовність 2, а потім знову потік виконання переходить до рядка, наступного за оператором умови. Аналогічно перевіряються всі інші умови. До гілки програми elsepотік виконання доходить тільки в тому випадку, якщо не виконується жодна з умов.

Ключове слово elif походить від англ. "Else if" – "інакше якщо". Тобто умова, яка слідує після нього перевіряється тільки тоді, коли всі попередні умови хибні.

Контрольні запитання і завдання

1. Як описується та виконується оператор розгалуження?
2. Як описується та виконується оператор множинного розгалуження?
3. Що називається логічним виразом?
4. Які 3 можливих варіанти представлення умови в інструкції if?

РОЗДІЛ 2. . ТИПИ ДАНИХ

Ідентифікатор Python – це ім’я, яке використовується для ідентифікації змінної, функції, класу, модуля або іншого об’єкту.

Ідентифікатор може містити тільки такі символи:

– літери в нижньому регістрі (від "a" до "z");

– літери у верхньому регістрі (від "A" до "Z") (Python є регістро-чутливою мовою програмування);

– цифри (від 0 до 9);

– нижнє підкреслення (_);

– не можуть співпадати з зарезервованими словами

Ідентифікатор не може починатися з цифри. Таким чином, A1 та a1 – два різних ідентифікатори в Python.

Коректними є такі імена: a; a1; a_b_c___95; _abc; _1a.

Наступні імена є некоректними: 1; 1a; 1_.

Як правило великими літерами в Python позначаються константи.
В статичних мовах необхідно вказувати тип кожної змінної, який визначає, скільки місця змінна займе в пам’яті і що з нею можна зробити.

Комп’ютер використовує цю інформацію, щоб скомпілювати програму в дуже низькорівневу машинну мову. Оголошення типів змінних допомагає комп’ютеру знайти деякі помилки і працювати швидше, але це вимагає попереднього продумування і набору коду. Велика частина мов програмування, наприклад C, C ++ та Java, вимагають оголошення типів змінних.

Тип даних – множина значень та множина операцій на цих значеннях. Тобто визначає можливі значення та їх сенс, операції над значеннями та способи зберігання.

Типізація – операція призначення типу інформаційним сутностям. Для різних мов програмування виділяють різні види типізації: статична/динамічна, сильна/слабка.

2.1.Прості типи даних. Числа

Числа бувають різними: цілими, дробовими, комплексними.

Вони можуть мати величезне значення або дуже довгу дробову частину:

– цілі числа (int) – додатні і від'ємні цілі числа, а також 0

(наприклад, 4, 687, -45, 0).

– числа з плаваючою точкою (float) – дробові числа (наприклад, 1.45, -3.789654, 0.00453). Роздільником цілої і дробової частини служить точка.

– комплексні числа (complex) – зберігає пару значень типу float, одне з яких представляє дійсну частину комплексного числа, а інше – уявну (наприклад, 1+2j, -5+10j, 0.44+0.08j

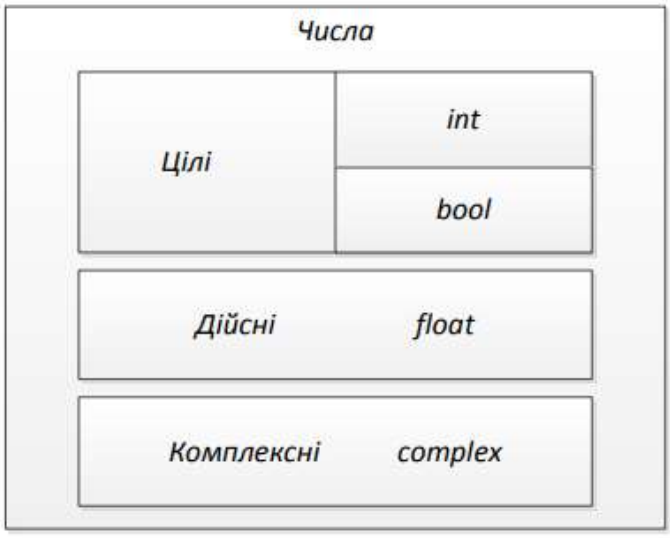


Рис.2.1 Прості типи даних

2.2.Операції над числами

Відомо, що операція – це виконання якихось дій над даними (операндами). Для виконання конкретних дій потрібні спеціальні інструменти – оператори, що наведені у таблиці 2.1.

Таблиця 2.1.

Оператори, що застосовуються у мові програмування Python

Оператор	Оператор	Приклад	Результат
+	Додавання	5 + 8	13
-	Віднімання	90 – 10	80
*	Множення	2*2	4
/	Ділення з плаваючою точкою	7/2	3,5
//	Цілочисельне	7/2	3
%	Залишок	7/2	1
**	Піднесення до степеню	2**2	4

Приклад 2.1 Використання операцій над числами

```
a = 20
b = 10
c = a + b
print("1. Значення c = ", c)
c = a - bprint ("2. Значення c = ", c)
c = a * b
print("3. Значення c = ", c)
c = a / b
print("4. Значення c = ", c)
c = a % b
print("5. Значення c = ", c)
c = a**b
```

```
print("6. Значення c = ", c)
c = a//b
print ("7. Значення c = ", c)
```

Результати виконання

1. Значення c = 30
2. Значення c = 10
3. Значення c = 200
4. Значення c = 2.0
5. Значення c = 0
6. Значення c = 10240000000000
7. Значення c = 2

Операція ділення поділяється на два підвиди:

–- за допомогою оператора / виконується ділення з плаваючою точкою (десяткове ділення);

– за допомогою оператора // виконується цілочисельне ділення (ділення із залишком).

Якщо ділити ціле число на ціле число, оператор / дасть результат з плаваючою точкою.

2.3.Оператори присвоєння

Можна поєднувати арифметичні оператори з привласненням, розміщуючи оператор перед знаком =.

Вираз a -= 3 аналогічний виразу a = a - 3.

Замість знаку – можуть стояти інші оператори: +, *, /, //, %

Приклад 2.2 Використання операторів присвоєння

```
a= 20
b = 10
c = a + b
print ("1. Значення c = ", c)
c += a
```

```
print ("2. Значення c = ", c)
c *= a
print ("3. Значення c = ", c)
c /= a
print ("4. Значення c = ", c)
c = 2
c %= a
print ("5. Значення c = ", c)
c **= a
print ("6. Значення c = ", c)
c //= a
print ("7. Значення c = ", c)
```

Результати виконання

```
1. Значення c =  30
2. Значення c =  50
3. Значення c =  1000
4. Значення c =  50.0
5. Значення c =  2
6. Значення c =  1048576
7. Значення c =  52428
```

2.4.Пріоритет операцій

Коли вираз містить більше одного оператора, послідовність виконання операцій залежить від порядку їх слідування у виразі, а також від їх пріоритету. Пріоритети операторів в Python збігаються з пріоритетами математичних операцій.

```
d = 2 + 3 * 4
print ("Значення d =", d )
```

Результатом виконання даного коду буде: 14

Найвищий пріоритет мають дужки, потім піднесення до степеню, множення та ділення і лише після них додавання, віднімання, далі кон’юкція, диз’юнкція та все інше.

У випадку рівних пріоритетів розрахунок йде справа наліво. Для зміни цього порядку використовують дужки. Краще їх використовувати у всіх сумнівних ситуаціях:

```
d = (2 + 3) * 4
print ("Значення d =", d )
```

Результатом виконання даного коду буде:20

2.5.Перетворення типів

Для того щоб змінити одні типи даних на інші використовуються певні стандартні функції.

Так, для зміни чогось на цілочисельний тип, слід використовувати функцію int(). Вона зберігає цілу частину числа і відкидає залишок.

Приклад 2.3 Використання перетворення типів.

```
a = 20
b = 10.8
q = int(a)
print ("1. Значення q = ", q)
q = int(b)
print ("2. Значення q = ", q)
```

Результатом виконання даного коду буде:

1. Значення q = 20
2. Значення q = 10

Текстовий рядок теж можна зміни та цілочисельний, якщо він буде містити цифрові символи і, можливо, знаки + і -.

Приклад 2.4. Перетворення текстових змінних на цілочисельні.

```
a = "-15"
q = int(a)
print ("1. Значення q = ", q)
```

Результатом виконання даного коду буде: q = -15

Якщо перетворити щось несхоже на число – буде згенеровано виняток:

Приклад 2.5. Винятки при перетворенні даних.

```
a = "xyz"
q = int(a)
print ("1. Значення q = ", q)
```

Результатом виконання даного коду буде:

```
Traceback (most recent call last):
 File "G:\lab.py", line 3, in <module>
 q = int(a)
ValueError: invalid literal for int() with base 10: 'xyz'
```

Для того щоб перетворити інші типи в тип float, слід використовувати функцію float().

Приклад 2.6. Перетворення значення типу int в тип float.

```
a = 98
b = '99'
q = float(a)
print ("1. Значення q = ", q)
q = float (b)
print ("2. Значення q = ", q)
```

Результатом виконання даного коду буде:

```
1. Значення q = 98.0
2. Значення q = 99.0
```

Приклад 2.7. Перетворення рядків з плаваючою точкою.

```
a = '98.6'
b = '-1.5'
c = '1.0e4'
d = True
q = float (a)
print ("1. Значення q = ", q)
```

```
q = float (b)
print ("2. Значення q = ", q)
q = float (c)
print ("3. Значення q = ", q)
q = float (d)
print ("4. Значення q = ", q)
```

Результатом виконання даного коду буде:

1. Значення q = 98.6
2. Значення q = -1.5
3. Значення q = 10000.0
4. Значення q = 1.0

2.6.Прості логічні вирази та логічний тип даних

В усіх мовах програмування високого рівня є можливість розгалуження програми; при цьому виконується одна з гілок програми в залежності від істинності чи хибності умови.

Логічними виразами називають вирази, результатом яких є істина (True) або хибність (False). У найпростішому випадку будь-яке твердження може бути істинним або хибним.

Наприклад, "2 + 2 дорівнює 4" – істинний вираз, а "2 + 2 дорівнює 5" – хибний.

Розмовляючи на природній мові (наприклад, українській) порівняння позначається словами "рівно", "більше", "менше". У мовах програмування використовуються спеціальні знаки, подібні до тих, які використовуються в математичних виразах:

> (більше), < (менше), >= (більше або дорівнює), <= (менше або дорівнює).

В Python використовуються наступні оператори порівняння:

– рівність (==);

– нерівність (! =);

– менше (<);

– менше або дорівнює (<=);

– більше (>);

– більше або дорівнює (> =);

– включення (in ...).

Ці оператори повертають булеві значення True або False. Для перевірки на рівність використовуються два знака "дорівнює" (==) (один знак "дорівнює" застосовується для надання значення змінній)

Приклад 2.8. Використання логічних значень.

```
x = 2 + 2
print('1.Результат роботи логічного виразу:', x == 4)
print('2.Результат роботи логічного виразу:', x == 5)
print('3.Результат роботи логічного виразу:', x != 5)
```

Результатом виконання даного коду буде:

```
1.Результат роботи логічного виразу: True
2.Результат роботи логічного виразу: False
3.Результат роботи логічного виразу: True
```

Результат порівняння двох значень можна записати в змінну:

```
y = x == 5
print (y)
```

Отримаємо:

```
False
```

2.7.Логічні оператори

Логічні вирази типу x>=y є простим. Однак, на практиці не рідко використовуються більш складні. Може знадобитися отримати відповіді "Так" або "Ні" в залежності від результату виконання двох простих виразів. Для об’єднання простих виразів в більш складні використовуються логічні оператори: and, or і not.

Значення їх повністю збігаються зі значенням англійських слів, якими вони позначаються.

Щоб отримати істину (True) при використанні оператора and, необхідно, щоб результати обох простих виразів, які пов’язує цей оператор, були істинними. Якщо хоча б в одному випадку результатом буде False (хибність), то і весь складний вираз буде хибним.

Щоб отримати істину (True) при використанні оператора or, необхідно, щоб результати хоча б одного простого виразу, що входить до складу складного, був істинним. У разі оператора or складний вираз стає хибним лише тоді, коли хибні всі складові його прості вирази.

Оператор not унарний, тобто він працює тільки з одним операндом.

Результатом застосування логічного оператора not (не) відбудеться заперечення операнда, тобто якщо операнд істинний, то not поверне – хибність, якщо хибний, то – істин

Логічний оператор and поверне True або False, якщо його операндами є логічні вирази.

```
print(2>4 and 45>3)
```

Отримаємо:

```
False
```

Для обчислення оператора and Python обчислює операнди зліва направо і повертає перший об’єкт, який має хибне значення.

```
print(0 and 3)
```

Отримаємо:0

Рядки в Python теж можна порівнювати по аналогії з числами. Символи, як і все інше, представлено в комп’ютері у вигляді чисел. Є спеціальна таблиця, яка ставить у відповідність кожному символу деяке число 20. Визначити, яке число відповідає символу можна за допомогою функції ord().

Приклад 2.7 Порівняння рядків.

```
q=ord ('L')
print ("1. Значення 'L' =", q)
q=ord ('Ф')
print ("2. Значення 'Ф' =", q)
```

```
q=ord ('A')
print ("3. Значення 'A' =", q)
q=ord ('a')
print ("4. Значення 'a' =", q)
```

Результатом виконання даного коду буде:

1. Значення 'L' = 76
2. Значення 'Ф' = 1060
3. Значення 'A' = 65
4. Значення 'a' = 97

2.7.Рядкові змінні в Python

Рядки в Python можна порівнювати по аналогії з числами. Символи, як і все інше, представлено в комп'ютері у вигляді чисел. Є спеціальна таблиця, яка ставить у відповідність кожному символу деяке число(ASCII).

Символи розширеного ASCII діапазону друку

https://uk.wikipedia.org/wiki/ASCII

Визначити, яке число відповідає символу можна за допомогою функції ord().

Приклад 2.8. Рядкові змінні.

```
q=ord('L')
print(q)
print ("1. Значення 'L' =", q)
q=ord ('Ф')
print ("2. Значення 'Ф' =", q)
q=ord ('A')
print ("3. Значення 'A' =", q)
q=ord ('a')
print ("4. Значення 'a' =", q)
```

Результатом виконання даного коду буде:

1. Значення 'L' = 76
2. Значення 'Ф' = 1060

3. Значення 'A' = 65

4. Значення 'a' = 97

Порівняння 'A' > 'L' = False

Для порівняння рядків Python їх порівнює посимвольно:

```
q='Aa' > 'Ll'
print ("Порівняння 'Aa' > 'Ll' =", q)
```

Отримаємо: Порівняння 'Aa' > 'Ll' = 76

Порівняння 'Aa' > 'Ll' = False

Оператор in перевіряє входження підрядка в рядок:

```
q='a' in 'abc'
print("Результат входження 'a' in 'abc' -", q)
```

Результат входження 'a' in 'abc' - 76

Отримаємо:

Результат входження 'a' in 'abc' - True

Результат входження 'A' in 'abc' - False

Результат входження '' in 'abc' - True

Результат входження '' in '' - Tru

Порівняння символів зводиться до порівняння чисел, які їм відповідають.

```
q='A' > 'L'
print ("Порівняння 'A' > 'L' =", q)
```

Отримаємо:

Порівняння 'A' > 'L' = False

Для порівняння рядків Python їх порівнює посимвольно:

```
q='Aa' > 'Ll'
print ("Порівняння 'Aa' > 'Ll' =", q)
```

Отримаємо:

Порівняння 'Aa' > 'Ll' = False

Приклад 2.9 Оператор in перевіряє входження підрядка в рядок:

```
q='a' in 'abc'
```

```
print("Результат входження 'a' in 'abc' -", q)
q='A' in 'abc' # Великої літери А немає в рядку 'abc'
print("Результат входження 'A' in 'abc' -", q)
q="" in 'abc' # Порожній рядок міститься в будьякому рядку
print("Результат входження " in 'abc' -", q)
q=" in "
print("Результат входження " in " -", q)
```

Отримаємо:

Результат входження 'a' in 'abc' - True

Результат входження 'A' in 'abc' - False

Результат входження " in 'abc' - True

Результат входження " in " - True

```
a = "використаємо апостроф у слові подвір'я"
print ("Значення a:", a)
```

отримаємо

Значення a: використаємо апостроф у слові подвір'я

2.8.Складні структури даних

2.8.1.Умовні оператори

Інструкція if в Python

Ось синтаксис для основної інструкції if:

```
IF  ЛОГІЧНА УМОВА:
ПОСЛІДОВНІСТЬ ВИРАЗІВ
```

Приклад 2.8 Умовні оператори

```
x = 5
if x < 9:
   print("Привіт!")
```

2.8.2.Умови if-else у Python

Ми можемо додати пункт else до умови, якщо потрібно, аби щось відбувалось, коли умова False.

Ось загальний синтаксис:

```
    if  логічна умова:
    послідовність виразів_1
    else:
    послідовність виразів_2
```

Приклад 2.9 Використання if-else

```
x = 15
if x > 9:
   print("Привіт!")
else:
   print("Бувай!")
print("Кінець")
```

Приклад 2.10 Використання if-else в циклі.

```
number=1
while number < 5:
  if number/2==0:
    print(number)
  else:
   print(3*number+1)
  number=number+1
```

2.8.3.Умови if-elif-else у Python

Приклад 2.11. Використання if-elif-else

```
 x = 5
 if x < 5:
    print("Привіт!")
 elif x < 15:
    print("Рада тебе бачити")
 else:
```

```
        print("Бувай!")


    print("Кінець")
```

результат виконання програми

Рада тебе бачити

Кінець

Приклад 2.12. Використання if-elif-else з оператором and

```
x = 6
if x < 9 and x>3:
   print("Привіт!")
elif x < 15:
   print("Рада тебе бачити")
else:
   print("Бувай!")
print("Кінець")
```

Приклад 2.13. Реалізації конструкції if-elif-else.

```
    if not True:
        print("1")
    elif not (1+1==3):
        print("2")
    else:
        print("3")
```

Результатом запуску даного коду буде:3

Приклад 2.14. Реалізації конструкції if-elif-else.

```
age = 120
if age > 90:
   print("You are too old to party, granny.")
elif age < 0:
   print("You're yet to be born")
elif age >= 18:
```

```
    print("You are allowed to party")
else:
    "You're too young to party"
```

Приклад 2.15. Реалізації конструкції if-elif-else.

```
A=1
B=5
C=10
(a>b)or(c<=a)and(c==a)
```

Зіставлення передбачає визначення при операторі match шуканого значення, після якого можна перерахувати кілька потенційних кейсів, кожен з оператором case. У місці виявлення збігу між match і case виконується відповідний код.

Тут відбувається перевірка кількох умов і виконання різних операцій на основі значення, яке ми знаходимо всередині http_code.

Приклад 2.16. Оператором case

```
http_code = "418"

match http_code:
    case "200":
        print("OK")

    case "404":
        print("Not Found")

    case "418":
        print("I'm a teapot")

    case _:
        print("Code not found")
```

2.8.4. Інструкція switch-case

Приклад 2.16. Використання switch-case.

```
def switch(lang):
    if lang == "JavaScript":
        return "You can become a web developer."
    elif lang == "PHP":
```

```
        return "You can become a backend developer."
    elif lang == "Python":
        return "You can become a Data Scientist"
    elif lang == "Solidity":
        return "You can become a Blockchain developer."
    elif lang == "Java":
        return "You can become a mobile app developer"

print(switch("JavaScript"))
print(switch("PHP"))
print(switch("Java"))
```

2.8.5.Оператор pass

В Python інструкції з розгалуженням, або цикли, або функції з порожнім тілом заборонені, тому в якості тіла використовується "порожній оператор" pass.

Припустимо, що заплановано використання умовного оператора з декількома умовами, але встигли написати тільки один з блоків умовного оператора. При цьому постає питання, як її налагодити, якщо програма не виконується через синтаксичну помилку.

Приклад 2.17. Реалізації конструкції if-elif-else.

```
if not True:
    print("1")
elif not (1+1==3):
elif not (1+1==4):
elif not (1+1==5):
```

Блоки для випадків, коли значення not (1+1==3), not (1+1==4), not (1+1==5), ще не написані, тому програма не виконується через помилку SyntaxError: expected an indented block.

Ключове слово pass можна вставити на місце відсутнього блоку.

Приклад 2.16. Використання ключового слова pass.

```
def switch(lang):
    if not True:
```

```
    print("1")
elif not (1+1==3):
    pass
elif not (1+1==4):
 pass
elif not (1+1==5):
 pass
```

Щоб детальніше налаштувати умови, можна додати один або більше пунктів elif для перевірки та обробки декількох умов. Виконуватиметься лише код першої умови, яка має значення True .

2.9. Розв'язок задач

Приклад 2.17. Знайти найбільше число з трьох даних чисел.

Якщо перше число більше другого і більше третього, то воно найбільше, інакше воно не найбільше, тому розглядаємо два наступних числа. Якщо друге більше третього, то друге найбільше, інакше найбільше - третє число.

Алгоритм розв'язування:

1. Ввести три числа A, B і C;
2. якщо A>B и A>C то найбільше A
   інакше якщо B>C то найбільше B
   інакше найбільше C.

Програмна реалізація

```
import  math
a = int(input('a='))
b = int(input('b='))
c = int(input('c='))
if (a > b) and (a>b):
   max=a
   print("max",a)
```

```
if (b > c) and (b>a):
    max=b
    print("max",b)
if (c > a) and (c>a):
    max=c
    print("max",c)
```

Приклад 2.17. Обчислити значення виразу.

$$Y=\begin{cases} x^2+4x, x<-5; \\ |x|, -5\le x<0; \\ 1+\sqrt{x}, 0\le x<4; \\ x-1, x\ge 4. \end{cases}$$ **и**

Алгоритм розв'язування

1. ввести(x)
2. якщо x< -5

   то y:=x^2+4

   інакше якщо x<0

   то y:=abs(x)

   інакше якщо x<4

   то y:=1+$\sqrt{x}$

   інакше y:=x-1
3. вивести(y)

Програмна реалізація

```
import  math
x=int(input('x='))
print(x)
if (x<-5):
   y=x+4
elif x < 0:
   y = abs(x)
```

```
elif x < 4:
   y = 1 + math.sqrt(x)
else:
   y = x - 1
print(y)
```

отримаємо

```
x=10
10
9
```

Приклад 2.17. Обчислити значення виразу. Дано натуральне число до 100, яке визначає вік людини (в роках). Додати до цього числа найменування "рік", " роки " або " років ", наприклад 1 рік, 42 роки, 26 років.

Проаналізуємо в яких випадках ми додаємо ці слова:

- якщо це числа 11, 12, 13, 14 то - "років"
- якщо 1, 21,31, тобто остання 1, то - "рік"
- якщо 2,3,4,22,23,24,32,33,34..., тобто останні цифри 2..4, то - "роки"
- якщо 10,20...,5,6,7,8,9,15,16,17,18,19,25...29. тобто останні цифри 0, 5...9, то - "років"

*Програмна реалізація***:**

```
import  math
x=int(input('Ваш вік='))
print('Ваш вік=',x)
if (x>10) and (x<15):
   print(x,' років')
if (x ==1) :
      print(x, ' рік')
if (x>=2) and (x<4):
   print(x,' роки')
```

```
if (x==0) or (x==5)or (x==9):
   print(x,' років')
```

Контрольні запитання і завдання

1. Назвіть типи даних, які ви знаєте і які розпізнає Python.
2. Опишіть три варіанти використання функції range.
3. Яку функцію замість range рекомендується застосовувати у разі великого розміру діапазону? Чому це може бути важливо?
4. Як описується та виконується оператор розгалуження?
5. Як описується та виконується оператор множинного розгалуження?
6. Що називається логічним виразом?
7. Які з можливих варіанти представлення умови в інструкції if?
8. Яким чином кодуються логічні значення в мові Python? Чи має Python окремий логічний тип?
9. Які функції можна використовувати для введення й виведення даних?
10. Логічні операції: and, or, not. Для наступних виразів, замініть a, b, c на 1 або 0 так, щоб вираз став істинним (тобто 1). При виконанні необхідно використовувати інтерпретатор Python. Які вирази є логічно еквівалентними? Треба записати деякі з результатів у таблиці істинності:

(a and b)

(not a and b)

(not (a and b))

(a or b)

(a or not b)

(not (a or b))

(not (not a or not b))

(a and (a or b)) Результат залежить від b?

(a and b and c)

(a and b or c)

(a and (b or c))

((a and b) or c)

11. Скласти програму обчислення виразу $y = \cos^2 (2x)^2 + \sin (x)$

12. Скласти програму обчислення виразу $y = \sin(2x) + \sin (x)$

13. Скласти програму обчислення виразу $y = \sqrt{x^2 - y} - 2\sin (x)$

14. Скласти програму обчислення виразу a=1, d=2, c=1.

$$z = \begin{cases} a^2(d-c), y < 10; \\ y + 11ac, y = 10; \\ y^2(\frac{a}{b} - 2c), y > 10. \end{cases}$$

РОЗДІЛ 3. РЕАЛІЗАЦІЯ ЦИКЛІЧНИХ АЛГОРИТМІВ

У реальному житті ми досить часто зустрічаємося з циклами. У комп'ютерних програмах поряд з інструкціями розгалуження (тобто вибором шляху дії) також існують інструкції циклів (повторення дії). Якби інструкцій циклу не існувало, довелося б багато разів вставляти в програму один і той же код поспіль стільки раз, скільки потрібно виконати однакову послідовність дій.

Цикли – це інструкції, які виконують одну й ту ж саму послідовність дій, поки діє задана умова.

Кожен циклічний оператор має тіло циклу – якийсь блок коду, який інтерпретатор буде повторювати поки умова повторення циклу буде залишатися істинною.

В програмуванні розрізняють такі види циклів:

– Цикл з передумовою – виконує дії поки умова є істинною (while).

– Цикл з післяумовою – спочатку виконуються команди, а потім перевіряється умова (do…while).

– Цикл з лічильником – виконується задану кількість разів (for).

– Сумісний цикл – виконує команди для кожного елементу із заданого набору значень (for…which).

В Python присутні лише цикл з передумовою (while) та цикл for, що поєднує в собі два види – цикл з лічильником та сумісний цикл.

3.1.Цикл while з передумовою

Для організації **циклів з передумовою** використовують оператор while.

while <умова>:

<блок команд>

Дія: блок команд тіла циклу буде виконуватись до тих пір, поки умова істинна. Наприклад, щоб вивести слово “Hello” 5 разів, можна написати такий скрипт.

Приклад 3.1. Вивести слово “Hello” 5 разів.

```
i = 5
while i > 0:
  print ('Hello')
  i -= 1          # скорочений запис команди i = i - 1 (зменшення числа на одиницю)
```

3.2.Цикл while з післяумовою

Щоб створити цикл з післяумовою, можна використати таку конструкцію:

while True:

<блок команд>

if <умова>: break

Виконання цієї частини коду:

Оскільки умова циклу True - істина, то ми виконуємо команди тіла циклу. Дійшовши до команди розгалуження, перевіряємо виконання її умови. Якщо вона справджується, то виконується команда break. Вона здійснює переривання циклу, у якому знаходиться. Якщо ж умова команди розгалуження не виконується, тіло циклу виконується ще раз.

Коли програма знаходить команду break, виконання циклу, до якого вона належить, припиняється.

Приклад 3.2. Використання циклу з післяумовою

```
s = 1
while True:
    print(s,'lesson has already passed')
    s = s + 1
    if s>6:
       break
print('End of lessons')
```

Приклад 3.3 Використання циклу з математичними функціями

```
import math
x=2
y=math.cos(x)**2
print(y)

import math
x=2
y=math.sqrt(4)
print(y)

import math
    x=2
    y=math.tan(x)
    print(y)
```

Ще один варіант використання оператора циклу – обчислення формул із змінним параметром.

$$\sum_{i=1}^{n} i^3 = 1^3 + \cdots + n^3$$

В даному випадку, параметром, що змінюються є i, причому I послідовно приймає значення в діапазоні від 1 до n. За допомогою оператора циклу while рішення буде виглядати як в прикладі.

Приклад 3.4 Використання оператора циклу для обчислення формул із змінним параметром

```
n = int(input("Введіть n: "))
sum = 0
i = 1
while i <= n:
  sum += i**3
  i += 1
print ("sum = ", sum)
```

1. Необхідно ввести n – граничне значення i.

2. Ініціалізація змінної sum – в ній буде зберігатися результат, початкове значення – 0.

3. Ініціалізація змінної i (лічильника i) – за умовою, початкове значення – 1.

4. Починається цикл, який виконується, поки i <= n.

5. У тілі циклу в змінну sum записується сума значення з цієї змінної, отриманої на попередньому кроці, і значення i, піднесеної в куб.

6. Лічильнику i присвоюється наступне значення.

7. Після завершення циклу виводиться значення sum після виконання останнього кроку

Наступне значення лічильника отримують додаванням до його поточного значення кроку циклу (в даному випадку крок циклу дорівнює 1). Крок циклу при необхідності може бути від’ємним, і навіть дробовим. Крім того, крок циклу може змінюватися на кожній ітерації (тобто при кожному повторенні тіла циклу).

Приклад 3.5. Функція collatz().

```
correct_choice = False
while not correct_choice:
   choice = input("Введіть число 1 або 2:")
   if choice == "1" or choice == "2":
      correct_choice = True
   else:
     print ("Не правильно введено число, повторіть введеня")
```

Результатом виконання даного коду буде:

```
Введіть число 1 або 2:3
Не правильно введено число, повторіть введеня.
Введіть число 1 або 2:4
Не правильно введено число, повторіть введеня
Введіть число 1 або 2:1
```

1. Визначили логічну змінну correct_choice, присвоївши їй значення False.
2. Оператор циклу перевіряє умову not correct_choice: заперечення False – істина. Тому починається виконання тіла циклу: виводиться запрошення "Enter your choice, please (1 or 2):"і очікується уведення користувача.
3. Після натискання клавіші Enter введене значення порівнюється з рядками "1" і "2", і якщо воно дорівнює одній з цих значень, то змінній correct_choice присвоюється значення True. В іншому випадку програма виводить повідомлення "Не правильно введено число, повторіть введеня".
4. Оператор циклу знову перевіряє умову і якщо вона як і раніше істинна, то тіло циклу повторюється знову, інакше потік виконання переходить до наступного оператору, і інтерпретатор виходить з циклу і виконання програми припиняється або виконуються дії що прописані поза тілом циклу.

Напишемо функцію collatz(), яка має один параметр number.

Якщо число парне то collatz() повинна вивести число // 2 і повернути це значення.

Якщо число непарне, то collatz() повинна вивести і повернути 3 * number + 1.

Підказка: Ціле число є парним, якщо число % 2 == 0, і непарним, якщо число % 2 == 1.

Приклад 3.6. Функція collatz().

```
def collatz(number1):
    number=1
    while number < number1:
        if number / 2 == 0:
            print(number)
        else:
            print(number)
            print(3 * number + 1)
            number = number + 1


print(collatz(5))
```

Приклад 3.7. Коротка програма: Вгадай число

```
import random
secretNumber = random.randint(1, 20)
print('I am thinking of a number between 1 and 20.')
# Ask the player to guess 6 times.
for guessesTaken in range(1, 7):
  print('Take a guess.')
  guess = int(input())
  if guess < secretNumber:
     print('Your guess is too low.')
  elif guess > secretNumber:
     print('Your guess is too high.')
  else:
    break
if guess == secretNumber:
```

```
    print('Good job! You guessed my number in ' + str(guessesTaken) + '
guesses!')
  else:
    print('Nope. The number I was thinking of was ' + str(secretNumber))
```

3.3.Цикли for у Python

Основний синтаксис для написання циклу for у Python:

for <змінна_циклу> in <ітерований_об'єкт>:

<код>

Приклад 3.8. Друк чисел від 1 до 5 з кроком 1

```
for i in range(1,5,1):
   print(i)
```

Приклад 3.9. Використання циклу for

```
for i in range(5):
   print(i)
```

Приклад 3.10 Використання циклу for з арифметичними операціями

```
for j in range(15):
   print(j * 2)
```

Приклад 3.11 Використання циклу for для рядкових змінних

```
for num in range(8):
     print("Привіт" * num)
```

Приклад 3.12 Використання циклу для виведення елементів масиву

```
s = ["a", "b", "c", "d"]
for i in range(len(s)):
        print(s[i])
```

Результати виконання

```
a
b
```

c

d

Приклад 3.13 Використання циклу з двома параметрами

```
for i in range(2, 10):
    print(i)
```

3.4.Знаходження інтегралу функції

Метод прямокутників – метод чисельного інтегрування функції однієї змінної. Якщо розглянути графік підінтегральної функції, то метод буде полягати в наближеному обчисленні площі під графіком. Площа розбивається на прямокутники, ширина яких буде визначатися відстанню між відповідними сусідніми вузлами інтегрування, а висота – значенням підінтегральної функції в цих вузлах.

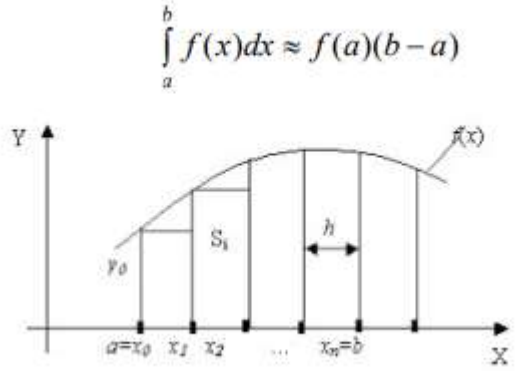


Рис.3.1 **Графік підінтегральної функції**

Приклад 3.14 Обчислення інтегралу$f(x) = \int_0^1 e^x dx$

```
import math
import numpy as np
n = 50
a = 0
b = 1
sum=0
x=0
```

```
h = (b - a)/n
print("Крок: ", h)
for x in np.arange(a, b, h):
    sum=sum+math.exp(x)
print("Результат: ", sum*h)
print("Число прямокутників: ", n)
```

3.4.Нескінченні цикли

Іноді можна зіткнулися з проблемою нескінченного повторення блоку коду, що виникає, наприклад, через семантичну помилку в програмі:

```
i = 0
while i < 10:
 print (i)
```

Такий цикл буде виконуватися нескінченно, тому що умова i <10 завжди буде істинною, адже значення змінної i не змінюється: така програма буде виводити нескінченну послідовність нулів.

Приклад 3.14. Переривання циклу.

```
i = 1
while True:
    if i > 5:
       break # Перериваємо цикл
    print (i)
    i += 1
```

Приклад 3.15 Використання циклу while

```
numbers = [1, 2, 3, 4, 5]
i = 0
while i <=len(numbers)-1:
   print(numbers[i])
 i+=1
```

3.5.Цикл for та else

Мова Python дозволяє використовувати розширений варіант оператора циклу:

WHILE <УМОВА_ПОВТОРЕННЯ_ЦИКЛУ >:

ТІЛО ЦИКЛУ

ELSE:

АЛЬТЕРНАТИВНА_ ГІЛКА_ЦИКЛУ

Приклад 3.15 Використання циклу for та else

```
i = 0
while i < 3:
 print (i)
 i += 1
else:
 print ("кінець циклу")
```

Цикл for дозволяє перебирати всі елементи вказаної послідовності (список, кортеж, рядок). Цикл спрацює рівно стільки разів, скільки елементів знаходиться в послідовності.

Приклад 3.16. Використання for та else зі списками

```
numbers = []
for number in numbers:
 print('Цей список має деякий елемент', number)
 break
else:
# Відсутність переривання означає, що елемент відсутній
 print('Елементів немає в списку, чи не так?')
```

Приклад 3.17. Ітерування за кількома послідовностям

```
days = ['Monday', 'Tuesday', 'Wednesday']
fruits = ['banana', 'orange']
drinks = ['coffee', 'tea', 'beer']
```

```
for day, fruit, drink in zip(days, fruits, drinks):
    print(day, ": drink", drink, "eat", fruit)
```

Результатом запуску даного коду буде:

Monday : drink coffee eat banana

Tuesday : drink tea eat orange

3.6.Генерація числових послідовностей за допомогою функції range()

Функція range() повертає послідовність чисел в заданому діапазоні без необхідності створювати і зберігати велику структуру даних на зразок списку або кортежу.

Це дозволяє створювати великі діапазони, не використавши всю пам’ять комп’ютера і не обірвавши виконання програми.

range (start, end, step)

Якщо опустити значення start, діапазон почнеться з 0.

Необхідною є лише значення end, що визначає останнє значення, яке буде створено прямо перед зупинкою функції. Значення step по замовчуванню дорівнює 1, але можна змінити його на -1 [5 - 13].

```
for x in range(0, 3):
 print(x)
```

3.7.Підходи до створення списків

Створити список цілих чисел можна декількома способами. Можна додавати елементи до списку по одному за раз використовуючи функцію append():

Приклад 3.18. Створення списків.

```
number_list = []
print (number_list)
number_list.append(1)
print (number_list)
number_list.append(2)
number_list.append(3)
```

number_list.append(4)

number_list.append(5)

print (number_list)

результат

[]

[1]

[1, 2, 3, 4, 5]

3.8.Спискове включення

Включення – це компактний спосіб створити структуру даних з одного або більше ітераторів. Включення дозволяють вам об’єднувати цикли і умовні перевірки, не використовуючи при цьому громіздкий синтаксис.

Найпростіша форма такого включення виглядає так:

[ВИРАЗ for ЕЛЕМЕНТ in ІТЕРАЦІЙНИЙ_ОБ’ЄКТ]

Приклад 3.19 Використання спискового включення

number_list = [number for number in range(1,6)]

print (number_list)

результат

[1, 2, 3, 4, 5]

[ВИРАЗ for ЕЛЕМЕНТ in ІТЕРАЦІЙНИЙ_ОБ’ЄКТ if УМОВА]

Наступне включення створює список, що складається тільки з парних чисел, розташованих в діапазоні від 1 до 5 (вираз number% 2 має значення True для парних чисел і False для непарних):

Приклад 3.20 Створює список, що складається тільки з парних чисел.

number_list = [number for number in range(1,6) if number % 2 == 1]

print (number_list)

Отримаємо:

[1, 3, 5]

Приклад 3.21 Створення списку з елементів, що будуть вводитися з клавіатури.

Вираз може містити будь-що. Для створення списку з елементів, що будуть вводитися з клавіатури:

```
number_list = [int(input('введіть число: ')) for number in
range(int(input('введіть кількість елементів: ')))]
print (number_list)
```

результат

```
введіть кількість елементів: 5
введіть число: 5
введіть число: 55
введіть число: 4
введіть число: 3
введіть число: 8
[5, 55, 4, 3, 8]
```

3.9.Вкладені списки

Списки можуть містити елементи різних типів, включаючи інші списки. Вкладеними називаються списки, які є елементами іншого списку.

Вкладені списки зазвичай використовуються для подання матриць. Змінною list_list = [[1, 2, 3], [4, 5, 6], [7, 8, 9]] описано матрицю:

```
1 2 3
4 5 6
7 8 9
```

Приклад 3.22 Виведення вкладених списків.

```
list_list = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
print(list_list[0])
```

результат

```
[[1, 2, 3], [4, 5, 6], [7, 8, 9]]
[1, 2, 3]
```

3.10.Вкладені цикли

Цикли можуть бути вкладені один в одного. При цьому цикли можуть використовувати різні змінні-лічильники.

Найпростіше застосування вкладених операторів циклу –побудова двовимірних таблиць (матриць, масивів), наприклад:

Приклад 3.23 Використання вкладених циклів.

```
i = 1
while i <= 5:
    j = 1
    while j <= 5:
        print (i * j, end="\t")
        j += 1
    print ()
    i += 1
```

результат

```
12      3       4       5
24      6       8       10
36      9       12      15
48      12      16      20
510     15      20      25
```

Даний прийом активно використовується при роботі з вкладеними списками.

Приклад 3.24 Використання вкладених циклів з одним циклом for.

```
lst = [[1, 2, 3], [4, 5, 6]]
for i in lst:
 print (i)
```

результат

```
[1, 2, 3]
[4, 5, 6]
```

Контрольні запитання і завдання

1. Що таке цикл? Навіщо вони потрібні?
2. Як описується та виконується циклічна інструкція while?
3. Як можна організувати нескінченні цикли? Наведіть декілька варіантів і поясніть їх.
4. Як можна вийти з нескінченних циклів?
5. Що відбувається при запуску нескінченного циклу?
6. Чи може оператор циклу не мати тіла? Чому?
7. Для чого служать оператори переривання break та continue? Наведіть приклад.
8. Як працює оператор for?
9. Для організації яких циклів застосовується оператор for?
10. Що таке масиви? Як розташовуються елементи масивів у пам’яті?
11. Як звернутись до першого та останнього елементу масиву?
12. Що буде виведено на екран в результаті виконання команд? *
13. Вивести таблицю множення чисел від 0 до 9.
14. Серед випадкових чисел, що генеруються, знайти: усі від'ємні, усі більші заданого числа та всі числа, що потрапили у відомий діапазон.
15. Вивести коди ASCII латинських літер та символи, що відповідають діапазону 96…127.
16. Напишіть програму переведення миль (від 100 до 1000 миль із кроком 100 миль) у кілометри та навпаки (від 100 до 1000 км із кроком 100 км) для вказаних відстаней. 1 миля = 1,609344 км.
17. Назвемо шестизначне число "щасливим", якщо сума перших трьох його цифр дорівнює сумі останніх трьох. Порахуйте кількість щасливих чисел, у яких сума трьох цифр дорівнює 13.

РОЗДІЛ 4. СТРОКОВІ МЕТОДИ ТА ФУНКЦІЇ

4.1. Створення керуючих символів

Крім цього зворотний слеш (\) дозволяє створювати керуючі послідовності всередині рядків. Найбільш поширена послідовність \n, яка означає перехід на новий рядок. З її допомогою можна створити багаторядкові рядки з однорядкових:

```
alphabet = '\nabcdefg\nhijklmnop\nqrstuv\nwxyz'
print ("Значення alphabet:", alphabet)
```

Результатом виконання даного коду буде:

```
Значення alphabet:
abcdefg
hijklmnop
qrstuv
wxyz
```

Найбільш уживаними є:

\t – знак табуляції

\\ – похила риса вліво

\ ' – символ одинарних лапок

\ " – символ подвійних лапок

Приклад 4.1 Використання керуючих символів.

```
print('\tabc')
print('a\tbc')
print('ab\tc')
```

Отримаємо:

```
abc
a bc
ab c
```

Якщо потрібен зворотний слеш, необхідно надрукувати два:

```
print('abc\\')
```

Отримаємо:

abc\

4.2.Звернення до символу

Для того щоб отримати один символ рядка, задається зміщення всередині квадратних дужок після імені рядка.

Зсув першого (крайнього зліва) символу дорівнює 0, наступного –1 і т.д. Зсув останнього (крайнього праворуч) символу може бути виражено як -1, тому не потрібно рахувати, в такому випадку зміщення подальших символів дорівнюватиме -2, -3 і т.д.:

```
string= 'abcdefghijklmnopqrstuvwxyz'
print (string) # Виведення всього рядку
print (string [0]) # Виведення першого символу рядку
print (string [1]) # Виведення другого символу рядку
print (string [-1]) # Виведення останнього символу рядку
```

З рядка можна вилучати підрядок (частину рядка) за допомогою функції slice. Визначається slice за допомогою квадратних дужок, зміщення початку підрядка start і кінця підрядка end, а також розміру кроку step.

[start: end: step]

Деякі з цих параметрів можуть бути відсутні. У підрядок будуть включені символи, розташовані починаючи з точки, на яку вказує зміщення start, і закінчуючи точкою, на яку вказує зміщення end.

– Оператор [:] дозволяє взяти зріз всієї послідовності від початку до кінця.

– Оператор [start:] дозволяє взяти зріз послідовності з точки, на яку вказує зміщення start, до кінця.

– Оператор [: end] дозволяє взяти зріз послідовності від початку до точки, на яку вказує зміщення end - 1.

– Оператор [start: end] дозволяє взяти зріз послідовності з точки, на яку вказує зміщення start, до точки, на яку вказує зміщення end - 1.

– Оператор [start: end: step] дозволяє взяти зріз послідовності з точки, на яку вказує зміщення start, до точки, на яку вказує зміщення end мінус 1, опускаючи символи, чиє зміщення всередині підрядка кратне step.

Зміщення зліва направо визначається як 0, 1 і т.д., а справа наліво – як -1, -2 і т.д. Якщо не вказати start, функція буде використовувати в якості його значення 0 (початок рядку). Якщо не вказати end, функція буде використовувати кінець рядка.

Python не включає символ, розташований під номером, який вказаний останнім.

Приклад 4.2. Виведення символів.

```
string= 'abcdefghijklmnopqrstuvwxyz'
print (string [2:5]) # Виведення символів починаючи з 3-го до 5-го
print (string [2:]) # Виведення рядку починаючи з 3-го символу
print (string [:]) # Вся послідовність від початку до кінця
print (string [20:]) # Всі символи, починаючи з 20-го і до кінця
print (string [-3:]) # Останні три символи
print (string [18:-3]) # Починаючи з 18-го і закінчуючи 4 з кінця
print (string [-6:-2]) # Закінчуючи 3 з кінця
```

Результатом запуску даного коду буде:

```
cde
cdefghijklmnopqrstuvwxyz
abcdefghijklmnopqrstuvwxyz
uvwxyz
xyz
stuvw
uvwx
```

Щоб збільшити крок, необхідно вказати його після другої двокрапки.

Приклад 4.3. Виведення символів з кроком.

```
string= 'abcdefghijklmnopqrstuvwxyz'
```

print (string [::7]) # Кожен сьомий символ з початку до кінця

print (string [4:20:3]) # Кожен 3 символ, починаючи з 4 та закінчуючи 19-м

print (string [19::4]) # Кожен 4 символ, починаючи з19-го:

print (string [:21:5]) # Кожен п’ятий символ від початку до 20-го:

Отримаємо:

ahov

ehknqt

tx

afkpu

Значення end має бути на одиницю більше, ніж реальне зміщення.

Якщо задати від’ємний крок, Python буде рухатися у зворотний бік.

Приклад 4.4. Виведення символів починаючи з кінця.

string= 'abcdefghijklmnopqrstuvwxyz'

print (string [-1::-1]) # Всі символи, починаючи з кінця і закінчуючи на початку

print (string [::-1]) # Аналогічно

Отримаємо:

zyxwvutsrqponmlkjihgfedcba

zyxwvutsrqponmlkjihgfedcba

4.3.Використання функцій

Приклад 4.5. Використання функції len().

Функція len() підраховує символи в рядку:

string= 'abcdefghijklmnopqrstuvwxyz'

q=len(string)

print ('Довжина рядка string =', q)

Отримаємо:

Довжина рядка string = 26

Довжина порожнього рядка = 0. Функцію len() можна застосовувати до інших послідовностей (кортежі, словники, списки). На відміну від функції len() деякі функції характерні лише для рядків.

Для того щоб використовувати строкову функцію, необхідно ввести ім’я рядка, крапку, ім’я функції і аргументи, які потрібні функції:

рядок.функція(аргументи)

Однією з таких вбудованих функцій є функція split(), що розбиває рядок на список невеликих рядків, спираючись на роздільник:

рядок.split('роздільник')

Для того щоб об’єднати список рядків lines, розділивши їх символами нового рядка, потрібно написати '\n'.join(lines).

Приклад 4.6. Використання функції join().

```
q = ['abcdefgh', 'ijklmnopqr', 'stuvwxyz']
string = ''.join(q) # Об’єднання 3 послідовностей літер без розділення
print(string)
string = ', '.join(q) # Об’єднання 3 послідовностей літер з розділенням їх комами та пробілом
print(string)
```

Результатом запуску даного коду буде:

```
abcdefghijklmnopqrstuvwxyz
abcdefgh, ijklmnopqr, stuvwxyz
```

В Python є велика множина строкових методів (можуть бути використані з будь-яким об’єктом str) і модуль string, що містить корисні визначення.

```
string = "abcdefghijklmnopqrstuvwxyz"
string = ".abcdefghi jklmnopqrs tuvwxyz..."
```

Таблиця 4.1

Приклади використання рядкових функцій

q = string.strip('.') print(q)	# Видалення символів '.' з обох кінців рядка:
q = string.capitalize() print(q)	# Перше слово з великої літер
q = string.title()	# Всі слова з великої літери
q = string.upper()	# Всі слова великими літерами
q = string.lower()	# Всі слова маленькими літерами
q = string.swapcase()	# Зміна регістру літер

4.4. Форматування рядків

Python дозволяє виконати вирівнювання рядків [5, 8]

Приклад 4.7. Використання вирівнювання рядків.

string = ".abcdefghi jklmnopqrs tuvwxyz..."

q=string.center(60) # Рядок вирівнюється всередині заданої кількості пробілів (30) по центру

print(q)

q=string.ljust(60) # Рядок вирівнюється по лівому краю

print(q)

4.5. Форматування рядків з використанням символу %

Таблиця 4.2.

Типи даних

%s	Рядок
%d	Ціле число в десятковій системі числення
%x	Ціле число в шістнадцятковій системі числення
%o	Ціле число в вісімковій системі числення
%f	Число з плаваючою крапкою в десятковій системі
%e	Число з плаваючою крапкою в шістнадцятковій системі
%g	Число з плаваючою крапкою у вісімковій системі

%%	Символ %

Приклад 4.8. Використання символу %.

```
q = '%s' % 42
print(q)
q = '%d' % 42
print(q)
q = '%x' % 42
print(q)
q = '%o' % 42
print(q)
```

Приклад 4.9. Використання символу % для різних типів даних.

```
q = '%s' % 7.03
print(q)
q = '%f' % 7.03
print(q)
q = '%e' % 7.03
print(q)
q = '%g' % 7.03
print(q)
```

отримали

```
7.03
7.030000
7.030000e+00
7.03
```

Приклад 4.10. Інтерполяція деяких рядків і цілих чисел.

```
breed = 'British Shorthair'
cat = 'Lola'
weight = 4
q = "My favorite cat breed is %s" % breed
```

```
print(q)
q = "My cat %s weighs %s kg" % (cat, weight)
print(q)
```

Результатом запуску даного коду буде:

```
My favorite cat breed is British Shorthair
My cat Lola weighs 4 kg
```

4.6.Форматування за допомогою символів {} і функції format

В Python 3 рекомендується застосовувати новий стиль форматування за допомогою методу format(), що має наступний синтаксис:

рядок_спеціального_формату.format(*args, **kwargs)

У параметрі рядок_спеціального_формату всередині символів {} можуть бути вказані деякі специфікатори.

Всі символи, розташовані поза фігурних дужок, виводяться без перетворень. Якщо всередині рядка необхідно використовувати символи {}, то ці символи слід подвоїти, інакше збуджується виняток ValueError.

Приклад 4.11. Інтерполяція деяких рядків і цілих чисел.

```
n = 42
f = 7.03
s = 'string'
q = '{} {} {}'.format(n, f, s)
print(q)
```

Отримаємо:

```
42 7.03 string
```

4.7.Заміна символів

Можна використовувати функцію replace() для того, щоб замінити один підрядок іншим. В функцію передається старий підрядок, новий підрядок і кількість включень старого підрядка, яку потрібно замінити. Якщо опустити останній аргумент, будуть замінені всі включення.

Приклад 4.12. Заміна символів.

```
x='a a aaa a b ba ba ab cb bc'
q = x.replace('a', 'y')
print(q)
q = x.replace('a', 'y', 5) # Заміна 5 входжень
print(q)
```

Отримаємо:

```
y y yyy y b by by yb cb bc
y y yyy a b ba ba ab cb bc
```

4.8.Рядкові змінні в Python

Рядки в Python можна порівнювати по аналогії з числами. Символи, як і все інше, представлено в комп'ютері у вигляді чисел. Є спеціальна таблиця, яка ставить у відповідність кожному символу деяке число(ASCII).

Символи розширеного ASCII діапазону друку

https://uk.wikipedia.org/wiki/ASCII

Визначити, яке число відповідає символу можна за допомогою функції ord().

Приклад 4.13. Використання функції ord().

```
q=ord('L')
print(q)
print ("1. Значення 'L' =", q)
q=ord ('Ф')
print ("2. Значення 'Ф' =", q)
q=ord ('A')
print ("3. Значення 'A' =", q)
q=ord ('a')
print ("4. Значення 'a' =", q)
```

Результатом виконання даного коду буде:

```
1. Значення 'L' = 76
```

2. Значення 'Ф' = 1060

3. Значення 'A' = 65

4. Значення 'a' = 97

Порівняння 'A' > 'L' = False

Для порівняння рядків Python їх порівнює посимвольно.

функції ord().

Приклад 4.14. Використання функції ord() посимвольно.

```
q='Aa' > 'Ll'
print ("Порівняння 'Aa' > 'Ll' =", q)
```

Отримаємо: Порівняння 'Aa' > 'Ll' = 76

Порівняння 'Aa' > 'Ll' = False

Оператор in перевіряє входження підрядка в рядок:

```
q='a' in 'abc'
print("Результат входження 'a' in 'abc' -", q)
```

Результат входження 'a' in 'abc' - 76

Отримаємо:

Результат входження 'a' in 'abc' - True

Результат входження 'A' in 'abc' - False

Результат входження '' in 'abc' - True

Результат входження '' in '' - Tru

Порівняння символів зводиться до порівняння чисел, які їм відповідають.

```
q='A' > 'L'
print ("Порівняння 'A' > 'L' =", q)
```

Отримаємо:

Порівняння 'A' > 'L' = False

Для порівняння рядків Python їх порівнює посимвольно:

```
q='Aa' > 'Ll'
print ("Порівняння 'Aa' > 'Ll' =", q)
```

Отримаємо:

Порівняння 'Aa' > 'Ll' = False

Оператор in перевіряє входження підрядка в рядок.

Приклад 4.15. Перевіряємо входження підрядка в рядок.

```
q='a' in 'abc'
print("Результат входження 'a' in 'abc' -", q)
q='A' in 'abc' # Великої літери А немає в рядку 'abc'
print("Результат входження 'A' in 'abc' -", q)
q="" in 'abc' # Порожній рядок міститься в будьякому рядку
print("Результат входження " in 'abc' -", q)
q=" in "
print("Результат входження " in " -", q)
```

Отримаємо:

```
Результат входження 'a' in 'abc' - True
Результат входження 'A' in 'abc' - False
Результат входження " in 'abc' - True
Результат входження " in " - True
```

```
a = "використаємо апостроф у слові подвір'я"
print ("Значення a:", a)
```

отримаємо

Значення a: використаємо апостроф у слові подвір'я

4.9.Складні структури даних

4.9.1.Списки

Масив – набір фіксованої кількості елементів, що розміщені в пам’яті комп’ютера безпосередньо один за одним, а доступ до них здійснюється за індексом (номер даного елементу в масиві).

В Python для реалізації масиву використовуються списки.

Список – тип даних, що представляє собою послідовність певних значень, що можуть повторюватись. Але на відміну від масиву –кількість елементів у списку може бути довільною.

Списки – гетерогенна, змінювана структура даних, що може містити елементи різних типів, що перераховані через кому та заключені в квадратні дужки. Це дозволяє створювати структури будь-якої складності і глибини.

Списки служать для того, щоб зберігати об’єкти в певному порядку, особливо якщо порядок або вміст можуть змінюватися.

Можна змінювати список, додати в нього нові елементи, а також видалити або перезаписати існуючі. Можна змінити кількість елементів у списку, а також самі елементи. Одне і те ж значення може зустрічатися в списку кілька разів.

Список є об’єктом, тому може бути присвоєний змінній.

int_list=[1, 2, 5, 8]

Отримаємо:

[1, 2, 5, 8]

Список можна створити з нуля або більше елементів, розділених комами і вкладених у квадратні дужки.

Приклад 4.16. Створення списку з нуля.

```
empty_list = [ ]
number_list = [1, 2, 3, 4, 5]
week_days = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday']
print(empty_list)
print(number_list)
print(week_days)
```

Отримаємо:

```
[]
[1, 2, 3, 4, 5]
['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday']
```

Крім того, за допомогою функції list() можна створити порожній список:

```
another_empty_list = list()
print (another_empty_list)
```

Отримаємо:

[]

4.9.2.Функція list() перетворює інші типи даних в списки

Приклад 4.17. Перетворення рядка в список.

```
q = list('cat')
print(q)
```

Отримаємо:

```
['c', 'a', 't']
```

4.9.3.Звернення до елементу

Список містить різні дані, звертатися до яких можна через ім’я списку та вказавши зміщення необхідного елементу:

Приклад 4.18. Перетворення рядка в список.

```
letters_list = ['a', 'b', 'c']
print(letters_list) # Виведення всього списку
print(letters_list[0]) # Виведення першого елементу списку
print(letters_list[1]) # Виведення другого елементу списку
print(letters_list[2]) # Виведення третього елементу списку
print(letters_list[-1]) # Виведення останнього елементу списку
print(letters_list[-2]) # Виведення передостаннього елементу списку
```

Результатом виконання даного коду буде:

```
a
b
c
c
b
```

Зсув повинен бути коректним значенням для списку – воно являє собою позицію, на якій розташовується присвоєне раніше значення. Якщо вказати позицію, яка знаходиться перед списком або після нього, буде згенеровано виняток (помилка).

Приклад 4.19. Помилки при перетворенні рядка в список.

```
letters_list = ['a', 'b', 'c']
print(letters_list[3])
```

Отримаємо:

```
Traceback (most recent call last):
 File "G:\lab.py", line 9, in <module>
 print(letters_list[3])
IndexError: list index out of range
```

За аналогією з отриманням значення списку за допомогою його зміщення можна змінити це значення.

Приклад 4.20. Отриманням значення списку.

```
letters_list = ['a', 'b', 'c']
print (letters_list)
letters_list[2] = 'C'
print (letters_list)
```

Отримаємо:

```
['a', 'b', 'c']
['a', 'b', 'C']
```

4.9.3.Отримання елементів за допомогою діапазону зсувів

Можна отримати зі списку підпослідовність, використавши зріз списку.

Приклад 4.21. Отримання елементів за допомогою діапазону зсувів.

```
letters_list =['a', 'b', 'c', 'd', 'e']
print(letters_list[0:2]) # Виведення елементів починаючи з 1-го до 2-го
print(letters_list[::2]) # Кожен непарний елемент
print(letters_list[::-2]) # Всі елем. з останнього зі зміщенням вліво на 2:
print(letters_list[::-1]) # Інверсія списку
```

Отримаємо:

```
['a', 'b']
['a', 'c', 'e']
['e', 'c', 'a']
['e', 'd', 'c', 'b', 'a']
```

4.9.4.Методи списків

Для додавання елементів в кінець списку – використовують метод append().

Приклад 4.22. Додавання елементів в кінець списку.

```
letters_list =['a', 'b', 'c', 'd', 'e']
print(letters_list)
letters_list.append('f')
print(letters_list)
```

Результатом запуску даного коду буде:

```
['a', 'b', 'c', 'd', 'e']
['a', 'b', 'c', 'd', 'e', 'f']
```

Можна об’єднати один список з іншим за допомогою методу extend().

Приклад 4.23. Об’єднання списків.

```
letters_list =['a', 'b', 'c', 'd', 'e']
others_list = ['g', 'h', 'i']
print(letters_list)
print(others_list)
letters_list.extend(others_list)
print(letters_list)
```

Результатом виконання даного коду буде:

```
['a', 'b', 'c', 'd', 'e']
['g', 'h', 'i']
['a', 'b', 'c', 'd', 'e', 'g', 'h', 'i']
```

4.9.5. Додавання або зміна елемента

Оскільки словники відносяться до змінюваних типів даних, то можна додати або змінити елемент по ключу.

Додати елемент в словник досить легко. Потрібно просто звернутися до елементу по його ключу і привласнити йому значення. Якщо ключ вже існує в словнику, наявне значення буде замінено новим. Якщо ключ новий, він і вказане значення будуть додані в словник.

Приклад 4.24. Об’єднання списків.

```
dict_1 = {1: "a", 3: "c", 4: "d"}
dict_1[2] = "c" # Додавання нового елементу
print(dict_1)
dict_1[2] = "b" # Зміна елементу по ключу
print(dict_1)
```

Отримаємо:

```
{1: 'a', 3: 'c', 4: 'd', 2: 'c'}
{1: 'a', 3: 'c', 4: 'd', 2: 'b'}
```

Можна також використовувати оператор +=.

Приклад 4.25. Використання операторів +=.

```
letters_list =['a', 'b', 'c', 'd', 'e']
others_list = ['g', 'h', 'i']
print(letters_list)
print(others_list)
letters_list += others_list
print(letters_list)
```

Результат:

```
['a', 'b', 'c', 'd', 'e']
['g', 'h', 'i']
['a', 'b', 'c', 'd', 'e', 'g', 'h', 'i']
```

При використанні методу append(), список others_list був би доданий як один елемент списку, замість того щоб об’єднати його елементи зі списком letters_list:

Приклад 4.26. Додавання списків.

```
letters_list =['a', 'b', 'c', 'd', 'e']
others_list = ['g', 'h', 'i']
print(letters_list)
print(others_list)
letters_list.append(others_list)
```

```
print(letters_list)
```

Отримаємо:

```
['a', 'b', 'c', 'd', 'e', 'f', ['g', 'h', 'i']]
```

Функція append() додає елементи тільки в кінець списку.

Коли потрібно додати елемент в задану позицію, використовується функція insert(). Якщо вказати позицію 0, елемент буде додано в початок списку. Якщо позиція знаходиться за межами списку, елемент буде додано в кінець списку, як і у випадку з функцією append(), виняток не буде згенеровано:

Результатом виконання даного коду буде:

```
['a', 'b', 'c', 'd', 'e']
['a', 'b', 'c', 'd', 'e', 'g']
['a', 'b', 'c', 'd', 'e', 'g', 'k']
['a', 'b', 'm', 'c', 'd', 'e', 'g', 'k']
```

4.9.6.Видалення заданого елемента

Коли видаляється заданий елемент, всі інші елементи, які йдуть слідом за ним, зміщуються, щоб зайняти місце видаленого елемента, а довжина списку зменшується на одиницю. Один із варіантів видалення елементу є застосування інструкції del.

Приклад 4.27. Застосування інструкції del.

```
letters_list =['a', 'b', 'c', 'd', 'e']
print(letters_list)
del letters_list [2] # Видалення другого елементу
print(letters_list)
print(letters_list [2])
del letters_list [-1] # Видалення останнього елементу
print(letters_list)
print(letters_list [-1])
```

Отримаємо:

```
['a', 'b', 'c', 'd', 'e']
```

['a', 'b', 'd', 'e']

d

['a', 'b', 'd']

d

Якщо точно не відомо або все одно, в якій позиції знаходиться елемент, використовується функція remove(), щоб видалити його за значенням.

Приклад 4.27. Видалення елемента.

```
letters_list =['a', 'b', 'c', 'd', 'e']
print(letters_list)
letters_list.remove('b')
print(letters_list)
print(letters_list [1])
```

Отримаємо:

['a', 'b', 'c', 'd', 'e']

['a', 'c', 'd', 'e']

c

За допомогою функції pop() можна отримати елемент зі списку і в той же час видалити його. Якщо викликати функцію pop() і вказати зсув, вона поверне елемент, що знаходиться в заданій позиції. Якщо аргумент не вказано – буде використано значення -1. Так, виклик pop(0) поверне головний (початковий) елемент списку, а виклик pop() або pop(-1) – кінцевий елемент.

Приклад 4.28. Видалення елемента за допомогою функції pop().

```
letters_list =['a', 'b', 'c', 'd', 'e']
print(letters_list)
print(letters_list.pop(1)) # Повертаємо значення видаленого елементу
print(letters_list)
print(letters_list [1])
```

Отримаємо:

['a', 'b', 'c', 'd', 'e']

b

['a', 'c', 'd']

c

4.9.7. Визначення зміщення елемента по значенню

Щоб визначити зміщення елемента в списку по його значенню, використовується функція index().

Приклад 4.29 Визначити зміщення за допомогою функції index().

```
letters_list =['a', 'b', 'c', 'd', 'e']
print(letters_list)
print(letters_list.index('d'))
```

Отримаємо:

['a', 'b', 'c', 'd', 'e']

3

В Python наявність елемента в списку перевіряється за допомогою оператора in.

Приклад 4.30. Визначаємо наявність елемента в списку за допомогою оператора in.

```
letters_list =['a', 'b', 'c', 'd', 'e']
print("'d' in letters_list -", 'd' in letters_list)
print("'d' in letters_list -", 'b' in letters_list)
```

Отримаємо:

'd' in letters_list - True

'd' in letters_list - True

Одне і те ж значення може зустрітися більше одного разу. До тих пір поки воно знаходиться в списку хоча б в одному екземплярі, оператор in буде повертати значення True.

Щоб визначити, скільки разів якесь значення зустрічається в списку, використовується функція count().

Приклад 4.31. Визначаємо скільки разів якесь значення зустрічається в списку.

```
letters_list =['a', 'b', 'c', 'd', 'e']
```

```
print(letters_list.count('b'))
print(letters_list.count('o'))
```

Отримаємо:

```
0
2
```

4.9.8.Зміна порядку елементів за допомогою функції sort()

Щоб змінювати порядок елементів за їх значенням, а не за зсувам в Python є дві функції:

– функція списку sort(), яка сортує сам список;

– функція sorted(), яка повертає відсортовану копію списку.

Якщо елементи списку є числами, вони за замовчуванням сортуються по зростанню. Якщо рядками, то сортуються в алфавітному порядку:

Приклад 4.32. Сортируємо в алфавітному порядку.

```
names = ['Alex', 'Olga', 'Helen']
print(names)
sorted_names = sorted(names)
print(sorted_names)
print(names)
```

sorted_names – це копія, її створення не змінило оригінальний список:

```
['Alex', 'Olga', 'Helen']
['Alex', 'Helen', 'Olga']
['Alex', 'Olga', 'Helen']
```

Виклик функції списку sort() для names змінить цей список:

```
names = ['Alex', 'Olga', 'Helen']
print(names)
names.sort()
print(names)
```

Отримаємо:

```
['Alex', 'Olga', 'Helen']
['Alex', 'Helen', 'Olga']
```

Якщо всі елементи списку одного типу, функція sort()відпрацює коректно. Іноді можна навіть змішати типи –наприклад, цілі числа і числа з плаваючою точкою, – оскільки в рамках виразів вони конвертуються автоматично.

Таблиця 4.2.

Методи словників

dict.fromkeys(seq, value=None)	Створює словник dict з ключами з послідовності seq і значенням value (за замовчуванням None)
dict.get(key, default=None)	Повертає значення ключа із словника dict, але якщо ключ відсутній, не створює виняток, а повертає default (за замовчуванням None)
dict.items()	Повертає пари (ключ, значення) із словника dict у вигляді списку кортежів [(key1, value1), (key2, value2), ...]
dict.keys()	Повертає ключі словника dict
dict.pop(key, default)	Видаляє ключ і повертає значення словника dict; якщо ключ відсутній, повертає default (якщо default відсутній - створює виняток KeyError)
dict.popitem()	Видаляє і повертає пару (ключ, значення) словника dict; якщо словник порожній, створює виняток KeyError (пам’ятайте, що словники невпорядковані)
dict.setdefault(key, default=None)	Повертає значення ключа словника dict, але якщо він відсутній, не створює виняток, а створює ключ із значенням default (за замовчуванням None)
dict.update(other)	Оновлює словник dict, додаючи пари (ключ, значення) із other; існуючі ключі перезаписуються (повертає None)
dict.values()	Повертає значення із словника dict у вигляді списку [value1, value2, ...]

Таблиця 4.3.

Методи рядків

Метод	Призначення
s.capitalize()	Повертає копію рядка s, роблячи першу букву у верхньому регістрі

s.lower()	Повертає копію рядка s із символами у нижньому регістрі
s.swapcase()	Повертає копію рядка s, в якій кожна буква матиме протилежний регістр
s.title()	Повертає копію рядка s, в якій кожне нове слово починається з великої літери
s.upper()	Повертає копію рядка s із символами у верхньому регістрі
s.count(x)	Для рядка s повертає кількість входжень в нього зазначеного підрядка x
s.find(x)	Повертає найменший індекс з рядка s, за яким знаходиться початок зазначеного підрядка x (якщо підрядок x не знайдено, повертає -1)
s.index(x)	Повертає найменший індекс з рядка s, за яким знаходиться початок зазначеного підрядка x (якщо підрядок не знайдено, створюється виняток ValueError)
s.rfind(x)	Повертає найбільший індекс з рядка s, за яким знаходиться початок зазначеного підрядка x (якщо підрядок x не знайдено, повертає -1)
s.rindex(x)	Повертає найбільший індекс з рядка s, за яким знаходиться початок зазначеного підрядка x (якщо підрядок не знайдено, створюється виняток ValueError)
s.replace(a, b)	Повертає копію рядка s, де всі входження підрядка a у рядок s замінюються підрядком b
s.startswith(x)	Повертає True, якщо рядок s починається з вказаного префіксу x (якщо ні, то повертає False)
s.endswith(x)	Повертає True, якщо рядок s закінчується вказаним префіксом x (якщо ні, то повертає False)
s.join(x)	Повертає рядок, складений з елементів ітеративного об’єкта x з розділювачами s
s.lstrip(chars)	повертає копію рядка s, на початку якого видалені символм chars
s.rstrip(chars)	повертає копію рядка s, наприкінці якого видалені символм chars
s.strip(chars)	повертає копію рядка s, на початку і у кінці якого видалені символм chars
s.split(char)	Розділення рядка s по розділювачу char і зберігання у список
s.center(width, fill)	Повертає відцентрований рядок, по краях якого стоїть символ fill (*пропуск* за замовчуванням)
s.ljust(width, fillchar)	Робить довжину рядка s не меншою width, в разі потреби заповнюючи останні символи символом fillchar
s.rjust(width, fillchar)	Робить довжину рядка s не меншою width, в разі потреби заповнюючи перші символи символом fillchar
s.isalnum()	Чи складається рядок s з букв і цифр?
s.isalpha()	Чи складається рядок s з букв?
s.isdigit()	Чи складається рядок s з цифр?
s.istitle()	Чи починаються слова в рядку s з великої букви?
s.isupper()	Чи складається рядок s із символів у верхньому регістрі?
s.islower()	Чи складається рядок s із символів у нижньому регістрі?

4.9.9.Форматування рядків. Метод format

Іноді (а точніше, доволі часто) виникають ситуації, коли потрібно зробити рядок, підставивши в нього деякі дані, отримані в процесі виконання програми (користувацьке введення, дані з файлів тощо). Підстановку даних можна зробити за допомогою форматування рядків. Форматування можна зробити за допомогою оператора %, або за допомогою методу format.

Якщо для підстановки потрібен тільки один аргумент, то значення - сам аргумент:

Приклад 4.33. Форматування рядків за допомогою методу format.

```
print('Hello, {} !'.format('Vasya'))
```

Результат

```
Hello, Vasya !
```

Приклади

```
print('Hello, {} !'.format('Vasya'))
print('{0} , {1} , {2} '.format('a', 'b', 'c'))
print( '{} , {} , {} '.format('a', 'b', 'c'))
print( '{2} , {1} , {0} '.format('a', 'b', 'c'))
print( '{2} , {1} , {0} '.format(*'abc'))
print( '{0} {1} {0} '.format('abra', 'cad'))
print( 'Coordinates: {latitude} , {longitude} '.format(latitude='37.24N', longitude='-115.81W'))
coord = {'latitude': '37.24N', 'longitude': '-115.81W'}
print('Coordinates: {latitude} , {longitude} '.format(**coord))
```

результат

```
Hello, Vasya !
a , b , c
a , b , c
c , b , a
c , b , a
```

abra cad abra

Coordinates: 37.24N , -115.81W

Coordinates: 37.24N , -115.81W

Приклад 4.34. Специфікації формату.

```
print("Units destroyed: {players[0]} ".format(players = [1, 2, 3]) )
print( "Units destroyed: {players[0]!r} ".format(players = ['1', '2', '3']))
```

Результат

Units destroyed: 1

Units destroyed: '1'

Python дуже багатий на операції з рядковими об'єктами. Рядки можна визначити у програмі за допомогою рядкових літералів. Літерали записуються з використанням апострофів ', лапок " або цих самих символів, узятих тричі. Усередині літералів зворотна коса риска має спеціальне значення. Вона служить для введення спеціальних символів та для визначення символів через коди. Якщо перед рядковим літералом поставлено r, зворотна коса риска не має спеціального значення (r від англійського слова raw, рядок вказуєтья "як є"). Unicode-літерали вказуються із префіксом u. Наведемо кілька прикладів:

4.10.Масиви

Масив - це структура даних, у якій зберігаються значення одного типу. Масиви в Python можуть містити тільки значення, які відповідають одному й тому самому типу даних. Масив не є основним типом даних, як рядки, ціле число тощо з числа раніше перелічених типів. Для застосування масивів у мові Python, перш за все потрібно імпортувати стандартний array модуль.

Приклад 4.35. Імпорт модуля array у Python.

```
from array import*
import numpy as np
a = np.array([1, 2, 3])
print(a)
a = np.array([1.1, 2.2, 3.3])
print(a)
```

```
a = np.array([[2, 4], [6, 8], [10, 12]])
print(a)
b = np.array([[[1, 2], [3, 4]], [[5, 6], [7, 8]], [[9, 10], [11, 12]]])
print(b)
a = np.array([[2, 4], [6, 8], [10, 12]])
b = np.array([[[1, 2], [3, 4]], [[5, 6], [7, 8]], [[9, 10], [11, 12]]])
np.zeros((3,3))
f=np.array( [[ 0. , 0., 0.] ,
[ 0. , 0., 0.] ,
[ 0. , 0., 0.] ])
print(np .ones ((3,3)))
f1=np.array( [[ 1. , 1., 1.] ,
[ 1. , 1., 1.] ,
[ 1. , 1., 1.] ])
np.ones ((3,3), dtype = complex) # Можно
```

результат

```
[1 2 3]
[1.1 2.2 3.3]
[[ 2  4]
 [ 6  8]
 [10 12]]
[[[ 1  2]
  [ 3  4]]

 [[ 5  6]
  [ 7  8]]

 [[ 9 10]
  [11 12]]]
[[1. 1. 1.]
```

[1. 1. 1.]

[1. 1. 1.]]

4.10.1.Оголошення масиву

Масив є спеціально оголошувана послідовність, що дозволяє компактно зберігати об'єкти одного з базових (раніше оголошених) типів. Масив оголошується після імпорту модуля array.

Приклад 4.36. Оголошення масиву.

```
from array import *
secondArray = array('i', [0,1,2,3,4,5])
print(secondArray)
```

Бібліотека numpy: оголошення масивів

```
import numpy as np
a = np.array([1, 2, 3])
print(a)
type(a)
```

результат

[1 2 3]

Приклад 4.36. Імпорт модуля array у Python.

```
import numpy as np
a = np.array([1, 2, 3])
print(a)
np.zeros((3,3))
f=np.array([[ 0. , 0., 0.] ,
[ 0. , 0., 0.] ,
[ 0. , 0., 0.] ])
print(f)


np.ones((3, 3))
v=np.array( [[ 1. , 1., 1.] ,
```

```
[ 1. , 1., 1.] ,
[ 1. , 1., 1.] ])
print(v)


np.empty([3, 3])
w=np.array([[-2.56e-042, 1.00e-313,1.51],
          [96e-3, 2.00000000e+000,2.61270],
           [1.00000000e+000,-5.41541116e-070, 1.48219694e-320]])
print(w)
```

результат

```
[1 2 3]
[[0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]]
[[1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]]
[[-2.56000000e-042  1.00000000e-313  1.51000000e+000]
 [ 9.60000000e-002  2.00000000e+000  2.61270000e+000]
 [ 1.00000000e+000 -5.41541116e-070  1.48219694e-320]]
```

Приклад 4.37.Індексація в масиві

```
import numpy as np
a = np.arange(10, 16)
print(a)
np.array([10, 11, 12, 13, 14, 15])
print(a[4])
```

Ітерація в масиві

```
for i in a:
    print(i)
```

результат

[10 11 12 13 14 15]

14

10

11

12

13

14

15

Приклад 4.38.Виведення елементів масиву.

```
import numpy as np
A = np.arange(10, 19).reshape((3, 3))
np.array([[10, 11, 12],
[13, 14, 15],
[16, 17, 18]])
for i in A:
    print(i)
```

результат

[10 11 12]

[13 14 15]

[16 17 18

Приклад 4.39.Виведення елементів масиву в стовпчик.

```
mport numpy as np
A = np.arange(10, 19).reshape((3, 3))
np.array([[10, 11, 12],
[13, 14, 15],
[16, 17, 18]])
for item in A.flat:
    print(item)
```

результат

10

11

12

13

14

15

16

17

18

Приклад 4.40.Виведення елементів масиву у вигляді матриці.

```
import numpy as np
np.sinh(0)
x = np.linspace(0, np.pi, num = 7)*1j
print(x)
w=np.array([0.+0.j, 0.+0.52359878j, 0.+1.04719755j, 0.+1.57079633j,
0.+2.0943951j, 0.+2.61799388j, 0.+3.14159265j])
print(w)
d=np.sinh(x)
np.array([ 0.+0.00000000e+00j, 0.+5.00000000e-01j, 0.+8.66025404e-01j,
0.+1.00000000e+00j, -0.+8.66025404e-01j, -0.+5.00000000e-01j,
-0.+1.22464680e-16j])
print(d)
```

результат

```
[0.+0.j         0.+0.52359878j 0.+1.04719755j 0.+1.57079633j
 0.+2.0943951j  0.+2.61799388j 0.+3.14159265j]
[0.+0.j         0.+0.52359878j 0.+1.04719755j 0.+1.57079633j
 0.+2.0943951j  0.+2.61799388j 0.+3.14159265j]
[ 0.+0.00000000e+00j  0.+5.00000000e-01j  0.+8.66025404e-01j
  0.+1.00000000e+00j -0.+8.66025404e-01j -0.+5.00000000e-01j
 -0.+1.22464680e-16j]
```

Контрольні запитання і завдання

1. Створіть список на основі введеної послідовності цілих чисел і надрукуйте другу половину списку як у вихідних даних.

2. Створіть список на основі введеної послідовності цілих чисел і надрукуйте його елементи таким чином: два останні елементи переміщені з кінця в початок списку без зміни їх початкового порядку.

3. Виведіть всі елементи списку з парними індексами. Вводиться список чисел. Всі числа списку знаходяться на одному рядку.

4. Вводиться список чисел. Всі числа списку знаходяться на одному рядку. Виведіть ті його елементи, які зустрічаються в списку лише один раз. Елементи потрібно виводити в тому порядку, в якому вони зустрічаються в списку.

5. Вводиться список цілих чисел в одному рядку через пропуск. Надрукуйте всі елементи, які перевищують попередній елемент списку, через пропуск в новому рядку в порядку їх розміщення у списку.

6. Напишіть програму, яка отримує повне ім’я файлу від користувача та друкує на екрані розширення отриманого файлу.

7. Напишіть програму для сортування за спаданням (порядок, зворотний алфавітному) словника за ключами. Словник зберігає пари ключ-значення у вигляді «країна: столиця». Інформація виводиться як у вихідних даних: сортування має бути проведено за назвами країн.

8. Надрукуйте елементи словника, де ключі - це числа від 1 до n (обидва числа включно), а значення - квадрати ключів. n – ціле число, яке вводить користувач.

9. Створіть словник для зберігання інформації про міста. Використайте назви міст в якості ключів словника. Створіть словник з інформацією про кожне місто: включіть в нього країну, приблизну чисельність населення

10. Виведіть назву кожного міста і всю збережену інформацію про нього як у вихідних даних.

11. Потрібно написати програму, яка здійснює перетворення з однієї одиниці вимірювання довжини в інші. Повинні підтримуватись.

12. Написати програму, яка ставить запитання та пропонує три варіанта відповіді. Якщо відповідь вірна, нараховується бал

Інформаційні ресурси підприємства – це:

а). інформаційні ресурси підприємства - весь обсяг знань, відчужений від їх створювачів, зафіксований на матеріальних носіях і призначений для загального використання;

б). інформаційні ресурси організації можна розуміти як весь наявний обсяг інформації в інформаційній системі;

в). інформаційні ресурси підприємства - весь обсяг знань;

г). інформаційні ресурси підприємства - весь обсяг знань призначений для використання;

д). інформаційні ресурси підприємства - весь обсяг інформації.

Написати словник – каталог відео. Знайти відео по автору

РОЗДІЛ 5. ФУНКЦІОНАЛЬНЕ ПРОГРАМУВАННЯ

Зазвичай реалізація складних задач містить величезні фрагменти коду. Зручним способом організувати великий фрагмент коду в більш зручні фрагменти є створення функцій. Функції в Python є основою при написанні програм. Іноді функцію порівнюють з "чорним ящиком", коли відомо, що на вході і що при цьому на виході, а нутрощі "чорного ящика" часто бувають приховані. Існує велика кількість вбудованих функцій. Наприклад функція abs(), приймає на вхід один аргумент – об’єкт числового типу і повертає абсолютне значення для цього об’єкта.

Результат виклику функції можна присвоїти змінній, використовувати його в якості операндів математичних виразів, тобто складати більш складні вирази.

Приклад 5.1. Використання стандартних функцій.

```
x = abs (-15)
y = x +5
print('y=', y)
```

Результатом виконання даного коду буде: y= 20

Крім складання складних математичних виразів Python дозволяє передавати результати виконання функцій в якості аргументів інших функцій без використання додаткових змінних.

Приклад 5.2. Виведення символів.

```
print(abs (-15))
```

Отримаємо:

```
15
```

Спочатку визначається абсолютне значення цілого числа -15, а потім за допомогою функції print() виводиться на екран результат розрахунку.

5.1. Власні функції

Функція – це іменований фрагмент коду, відокремлений від інших. Вона може приймає будь-яку кількість будь-яких вхідних параметрів і повертати будь-яку кількість будь-яких результатів.

З функцією можна зробити дві речі:

– визначити;

– викликати.

Щоб визначити функцію, використовується наступна конструкція:

def ім’я_функції(вхідні_параметри):

функція

Імена функцій підкоряються тим же правилам, що і імена змінних (вони повинні починатися з літери або _ і містити тільки букви, цифри або _).

Функція може не містити параметри але круглі дужки все рівно необхідно вказувати:

```
def do_nothing():
... pass
```

Тіло функції відділяється пробілами. Використання виразу pass відображає, що функція нічого не робить.

Щоб викликати функцію вказується її ім’я та дужки з параметрами:

```
do_nothing()
```

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

Визначимо і викличемо функцію, яка не має параметрів і виводить на екран одне слово.

Приклад 5.3. Використання функцій.

```
def salute():
    print('Hi!')
salute()
```

Отримаємо:

Hi!

Коли викликається функція salute(), Python виконує код, розташований всередині її опису. У цьому випадку він виводить одне слово і повертає управління основній програмі.

5.2.Параметри функції

Функція може приймати параметри та повертати значення. Параметри функції – звичайні змінні, якими функція користується для внутрішніх розрахунків. Якщо параметрів декілька – вони перераховуються через кому [5 – 10, 13].

Формальні параметри – параметри, що вказуються при оголошенні функції.

Фактичні параметри (аргументи) – параметри, що передаються в функцію при її виклику.

Приклад 5.4. Виведення елементів.

```
def print_numbers(limit):
  for i in range(limit):
    print(i)
n=int(input('Введіть кількість елементів: '))
print_numbers(n)
```

Результатом виконання даного коду буде:

```
Введіть кількість елементів: 5
0
1
2
3
4
```

Значення, які передаються в функцію при виклику, називаються аргументами. Коли функція викликається з аргументами, їх значення копіюються у відповідні параметри всередині функції

Існують функції які просто щось виконують, наприклад вбудована функція print() яка виводить на екран певні значення.

Приклад 5.5. Використання функції print().

```
n=15
print(n)
```

Отримаємо:

```
15
```

Для того щоб переглянути результат роботи такої функції потрібно скористатися функцією print().

Приклад 5.6. Використання функції print() для виведення слова.

```
a=input('Введіть слово: ')
print(a)
```

Отримаємо:

```
Hi!
```

При необхідності повернути результат роботи функції в програму, з якої вона викликалася, для її подальшого оброблення застосовується команда return. Вираз, що стоїть після return буде повертатися в якості результату виклику функції.

Без аргументів return використовується для виходу з функції (інакше вихід відбудеться при досягненні кінця функції).

В Python функції здатні повертати кілька значень одночасно.

Приклад 5.7. Розв’язок квадратного рівняння.

```
import math
def PrintRoots(a, b, c):
  D = b**2 - 4 * a * c
  x1 = (-b + math.sqrt(D)) / 2 * a
  x2 = (-b - math.sqrt(D)) / 2 * a
  return x1, x2
print (PrintRoots(1.0, 0, -1.0))
```

Результатом запуску даного коду буде:

```
(1.0, -1.0)
```

Крім того, результати виконання функції можна привласнювати відразу декільком змінним:

```
x1, x2 = PrintRoots(1.0, 0, -1.0)
print("x1 =", x1, "\nx2 =", x2)
```

Результатом запуску даного коду буде:

```
x1 = 1.0
x2 = -1.0
```

Всередині функції може міститися довільна кількість return. Однак спрацює лише один з них.

Приклад 5.8. Використання return.

```
def traffic_light (color):
  if color == 'red':
    return "STOP!"
  elif color == "green":
    return "GO!"
  elif color == 'yellow':
    return "GET READY!"
  else:
    return "Broken traffic light!"
```

Викликавши функцію traffic_light(), передавши їй в якості аргументу рядок 'blue'.

```
result = traffic_light('blue')
print(result)
```

Функція зробить наступне:

- присвоїть значення 'blue' параметру функції color;
- пройде по логічному ланцюжку if-elif-else;
- поверне рядок;
- присвоїть рядок змінній result.

Результатом буде:

```
'Broken traffic light!'
```

Функція може приймати будь-яку кількість аргументів (включаючи нуль) будь-якого типу. Вона може повертати будь яку кількість результатів (також включаючи нуль) будь-якого типу. Якщо функція не викликає return явно, буде отримано результат None.

```
def do_nothing():
 pass
```

Результатом буде:

None

None – це спеціальне значення в Python, яке заповнює собою порожнє місце, якщо функція нічого не повертає. Воно не є булевим значенням False, незважаючи на те що схоже на нього під час перевірки булевої змінної.

Простори імен та області видимості. Кожна функція визначає власний простір імен. Якщо визначити змінну, яка називається x в основній програмі та іншу змінну x в окремій функції, то вони будуть посилатися на різні значення. В основній програмі визначається глобальний простір імен, а змінні що тут знаходяться називаються глобальними тобто до неї можна звернутися з будь якого місця програми, в тому числі і всередині функції. Змінна є локальною (видно тільки всередині функції), якщо значення їй присвоюється всередині функцій [5, 8, 9].

Приклад 5.9. Локальні та глобальні змінні.

```
a = 3 # Глобальна змінна
y = 8 # Глобальна змінна
def func ():
  print ('func: глобальна змінна a = ', a)
  y = 5 # Локальна змінна
  print ('func: локальна змінна y = ', y)
func () # Виклик функції func()
print ('??? y = ', y) # Відобразиться глобальна змінна
print ('глобальна змінна a = ', a)
print ('глобальна змінна y = ', y)??? y = 8
```

глобальна змінна a = 3

глобальна змінна y = 8

Всередині функції можна звернутися до глобальної змінної a і вивести її значення на екран. Далі всередині функції створюється локальна змінна y, причому її ім’я збігається з ім’ям глобальної змінної – в цьому випадку при зверненні до y виводиться вміст локальної змінної, а глобальна залишається незмінною.

Щоб змінити значення глобальної змінної всередині функції використовується ключове слово global.

Приклад 5.10. Зміна значення глобальної змінної всередині функції.

```
x = 50 # Глобальна змінна
def func():
 global x # Вказуємо, що x-глобальна змінна
 print('x =', x)
 x = 2 # Змінюємо глобальну змінну
 print('Замінюємо глобальне значення x на', x)
func()
x = 50
print('Значення x =', x)
```

Результатом запуску даного коду буде:

x = 50

Замінюємо глобальне значення x на 2

Значення x = 50

З академічної точки зору зміна глобальної змінної всередині функції порушує принципи модульності програми.

Імена функцій в Python є змінними, що містять адресу об’єкта типу функція, тому цю адресу можна привласнити іншій змінній і викликати функцію з іншим ім’ям.

Приклад 5.11. Зміна глобальної змінної всередині функції.

```
def summa (x, y):
```

```
        return x + y
print(summa(5,6))
f = summa
v = f (10, 3) # Викликаємо функцію з іншим ім'ям
print(v)
```

Результатом запуску даного коду буде:

11

13

5.3.Аргументи функцій

Функція може приймати довільну кількість аргументів або не приймати їх зовсім. В функцію можна передавати не лише окремі об’єкти але і колекції послідовності (список, кортеж та ін.). крім того, аргументи можуть бути позиційними, іменованими, обов’язковими та не обов’язковими.

5.4.Позиційні аргументи

Найбільш поширений тип аргументів – це позиційні аргументи, чиї значення копіюються у відповідні параметри згідно з порядком проходження.

Приклад 5.12. Позиційні аргументи.

```
def func(a, b, c):
return a+b*c
print(func(1, 2, 3)) # a = 1, b = 2, c = 3
```

Отримаємо:

7

Незважаючи на поширеність аргументів такого типу, у них є недолік, який полягає в тому, що потрібно запам’ятовувати значення кожної позиції

5.5. Іменовані аргументи

Щоб уникнути плутанини з позиційними аргументами, можна вказати аргументи за допомогою імен відповідних параметрів. Порядок проходження аргументів в цьому випадку може бути іншим.

Приклад 5.13. Позиційні аргументи.

```
 print (func(a =2, b = 1, c = 3))
```

Отримаємо: 5

Можна об’єднувати позиційні аргументи та іменовані аргументи.

```
print (func(2, 2, c = 3))
```

Отримаємо: 8

Якщо викликати функцію, що має як позиційні аргументи, так і іменовані аргументи, то позиційні аргументи необхідно вказувати першими.

5.6.Значення параметра за замовчуванням

Можна вказати значення за замовчуванням для параметрів. Значення за замовчуванням використовуються в тому випадку, якщо викликаючи функцію не було вказано відповідний аргумент.

Приклад 5.14. Значення параметра за замовчуванням.

```
def func(a, b, c=2):
 return a+b*c
```

Викликаючи функцію func() можна не передавати їй аргумент c:

```
print(func(1, 2))
```

Отримаємо: 5

Але якщо надати аргумент, він буде використаний замість аргументу за замовчуванням:

```
print(func(1, 2, 3))
```
Отримаємо: 7

Отримання позиційних аргументів Якщо перед параметром у визначенні функції вказати символ *, то функції можна буде передати будь-яку кількість параметрів.

Якщо у функції є також обов’язкові позиційні аргументи, *args відправиться в кінець списку і отримає всі інші аргументи.

Приклад 5.15. Використання обов’язкових позиційних аргументів.

```
def print_more(numb_1, numb_2, *args):   print(numb_1)
   print(numb_2)
   print(args)
print_more(1, 2, 3, 4, 5, 6)
```

Результат

1

2

(3, 4, 5, 6)

5.7.Отримання іменованих аргументів

Можна використовувати **, щоб згрупувати іменовані аргументи в словник, де імена аргументів стануть ключами, а їх значення – відповідними значеннями в словнику. У наступному прикладі визначається функція *print_kwargs ()*, в якій виводяться її іменовані аргументи.

Приклад 5.16. Використання іменованих аргументів.

```
def print_kwargs(**kwargs):
  print('Іменовані аргументи:', kwargs)
print_kwargs(a = 1, b = 2, c = 3)
```

Отримаємо наступний результат:

Іменовані аргументи: {'b': 2, 'c': 3, 'a': 1}

Усередині функції *kwargs* є словником. Якщо використано позиційні аргументи та іменовані аргументи (*args і **kwargs), вони повинні слідувати в цьому ж порядку. Як і у випадку з args, не обов’язково називати цей словник kwargs.

5.8.Документаційні рядки

Можна додавати документацію до власних функцій, модулів, класів, заключивши рядок на початку тіла функції у лапки. Вона називається ***рядком документації або документаційним рядком (docstring):***

```
def func(anything):
 'Функція повертає введений аргумент'
 return anything
```

Як правило документація містить розгорнуту інформацію про те, що дана функція (модуль) виконує, які аргументи приймає та що повертає в результаті виконання, опис всіх констант (функцій, для модуля). Тож документація може бути досить великого розміру і щоб використати до такої

інформації форматування та вивести багато рядків коментарів необхідно заключити документаційний рядок в три пари подвійних лапок

```
def print_if_true(thing, check):
'''
Prints the first argument if a second argument is true.
 The operation is:
 1. Check whether the *second* argument is true.
 2. If it is, print the *first* argument.
 '''
 if check:
 print(thing)
print(help(print_if_true))
```

На відміну від звичайних коментарів, до документаційних рядків можна звернутися під час виконання програми. Для того щоб вивести рядок документації деякої функції, необхідно викликати функцію help(), передати їй ім’я функції, щоб отримати список всіх аргументів і відформатований рядок документації

```
Help on function print_if_true in module __main__:
print_if_true(thing, check)
 Prints the first argument if a second argument is true.
 The operation is:
 1. Check whether the *second* argument is true.
 2. If it is, print the *first* argument.
None
Довідка про функцію print_if_true у модулі __main__:
print_if_true(thing, check)
Виводить перший аргумент, якщо другий аргумент істинний.
Порядок роботи наступний:
1. Перевірити, чи є *другий* аргумент істинним.
2. Якщо так, то вивести перший аргумент.
```

5.9.Потік виконання

З появою функцій програми перестали бути лінійними, і в зв’язку з цим з’явилася концепція потоку виконання – порядок, у якому виконуються інструкції, що складають програму.

Виконання програми, написаної на Python, завжди починається з першого виразу, а наступні вирази виконуються послідовно зверху вниз. Крім того, визначення функцій жодним чином не впливають на потік виконання, оскільки тіло будь-якої функції не виконується, доки не буде викликана відповідна функція.

Коли інтерпретатор розбирає вихідний код і досягає виклику функції, після обчислення значень параметрів він починає виконувати тіло викликаної функції, і тільки після завершення продовжує розбирати наступну інструкцію.

З тіла будь-якої функції можна викликати іншу функцію, яка також може містити виклики функцій тощо. Однак інтерпретатор Python запам'ятовує, звідки була викликана кожна функція, і якщо під час виконання не відбувається винятків, рано чи пізно він повернеться до вихідного виклику та продовжить виконання наступного оператора.

Приклад 5.17. З функції може бути викликана інша функція.

```
def func1(name):
 print('Привіт, '+name)
def func2():
 return input('Введіть ім\'я ')
func1(func2())
```

Введіть ім'я olga

Привіт, olga

Приклад 5.18. Робота з декількома функціями

```
def func1(name,F):
 print('Привіт, '+name+F)
def func2():
 return input('Введіть ім\'я ')
```

```
def func3():
 return input('Введіть прізвище ')
func1(func2(),func3())
```

Приклад 5.19. Внутрішні функції

Можна визначити функцію всередині іншої функції:

```
def outer(a, b):
 def inner(c, d):
    return c + d
 return inner(a, b)
print(outer(4, 7))
```

Результатом запуску даного коду буде:

11

Внутрішні функції можуть бути корисні при виконанні деяких складних завдань більш ніж один раз всередині іншої функції. Це дозволить уникнути використання циклів або дублювання коду.

Приклад 5.20. Функція, для обчислення інтеграла

$$y = \int_0^1 e^x$$

Перший варіант без власних функцій

```
import math
a=0
b=1
h=(b-a)/18
s=0
x=0
while x<=1:
    s = s + math.exp(x)
    x = x + h
print(s*h)
```

Другий варіант з використанням власних функцій

```
import  math
def f(x):
    return math.exp(x)
def int(a,b,n,f):
    h = (b - a) / n
    x=0
    s=0
    while x <= 1:
        s = s + f(x)
        x = x + h
    return s*h
y=int(0,1,50,f)
print(y)
```

результат: 1.7011562858535672

Третій варіант

```
import  math
def інтеграл(f, a, b, n):
    d = (b - a) / float(n)
    integral = 0
    for i in range(n):
        x_i = a + i * d
        integral += f(x_i)
    integral *= d
    return integral

def f(x):
    return math.exp(x)
a = int(input("Введіть початок інтервалу a: "))
```

```
b = int(input("Введіть кінець інтервалу b: "))
n = 55
результат = інтеграл(f, a, b, n)
print('Значення інтегралу:', результат)
```

результат

```
Введіть початок інтервалу a: 0
Введіть кінець інтервалу b: 1
Значення інтегралу: 1.7027084198907234
```

Приклад 5.21. Напишіть функцію, яка створює список, що складається тільки з парних чисел, розташованих в діапазоні від 1 до 5. (вираз number% 2 має значення True для парних чисел і False. непарних).Напишіть функцію, яка створює список, що складається тільки з парних чисел, розташованих в діапазоні від 1 до 5. (вираз number% 2 має значення True для парних чисел і False. непарних).

```
def is_even(number):
    return number % 2 == 0
def get_even_numbers():
    numbers = range(1, 6)
    even_numbers = list(filter(is_even, numbers))
    return even_numbers
result = get_even_numbers()
print(result)
```

Приклад 5.22. Площа кола.

```
import math
def calculate_circle_area(radius):
    area = math.pi * radius ** 2
    return area
def main():
    radius = float(input("Введіть радіус кола: "))
```

```
        area = calculate_circle_area(radius)
        print(f"Площа кола з радіусом {radius} одиниць - {area:.2f} квадратних одиниць")
    main()
```

Приклад 5.23. Напишіть функцію, яка виводить значення функції y=cos(x)"

```
import math
def func_cos(x):
   y = math.cos(x)
   print ("Косинус Вашого числа становить ", y)

x=input("Будь ласка, введіть число, від якого Ви бажаєте взяти косинус: ")
x=int(x)
func_cos(x)
```

Результат

```
Будь ласка, введіть число, від якого Ви бажаєте взяти косинус: 1
Косинус Вашого числа становить  0.5403023058681398
```

Приклад 5.24. Протабулюємо функцію y=cos(x)

```
import  math
import  numpy as np

def s(x):
   for x in np.arange(0,1,0.09):
      y=math.sin(x)
      print(y)
      x=x+0.1

s(0)
```

отримаємо

0.0

0.08987854919801104

0.17902957342582418

0.26673143668883115

0.35227423327508994

0.4349655341112302

0.5141359916531131

0.5891447579422695

0.6593846719714731

0.7242871743701425

0.7833269096274833

0.8360259786005205

Приклад 5.25. Написати функцію, яка виводить максимальний елемент матриці

```
print("Написати функцію, яка виводить максимальний елемент матриці (стандартні функції не використовувати)")
import random
def max_matrix():
    matrix=[]
    i=1
    while i<4:
        row=[]
        j=1
        while j<5:
            x = random.randrange(1, 50, 1)
            row.append(x)
            print(x, end="\t")
            j+=1
        matrix.append(row)
```

```
        print()
        i+=1
    max_x=matrix[0][0]
    for i in range(3):
        for j in range(4):
            if max_x<matrix[i][j]:
                max_x=matrix[i][j]
    print("Максимальний елемент нашої матриці -- ", max_x)
max_matrix()
```

5.10.Анонімні функції: функція lambda()

В Python лямбда-функція – це анонімна функція, виражена одним виразом. Її можна використовувати замість звичайної маленької функції.

Для того щоб проілюструвати анонімні функції, спочатку створимо приклад, в якому використовуються звичайні функції.

Визначимо функцію edit_story(). Вона має такі аргументи:

– words – список слів;

– func – функція, яка повинна бути застосована до кожного слова в списку words.

– stairs – список слів

– edit_story – функція, яку потрібно застосувати кожного слова списку (записує з великої літери кожне слово і додає знак оклику).

Приклад 5.26. Анонімні функції.

```
import  math
stairs = ['hi', 'hello', 'привіт']

def edit_story(words, func):
    for word in words:
        print(func(word))
def enliven(word):
```

```
        return word.capitalize() + '!'
print(edit_story(stairs, enliven))
```

результат

```
Hi!
Hello!
Привіт!
None
```

Приклад 5.27. Функцію enliven() можна замінити лямбда функцією.

```
def edit_story(words, func):
 for word in words:
 print(func(word))
stairs = ['hi', 'hello', 'привіт']
edit_story(stairs, lambda word: word.capitalize() + '!')
def edit_story(words, func):
 for word in words:
    print(func(word))
stairs = ['hi', 'hello', 'привіт']
edit_story(stairs, lambda word: word.capitalize() + '!')
```

результат

```
Hi!
Hello!
Привіт!
```

Лямбда приймає один аргумент, який в цьому прикладі названий word. Все, що знаходиться між двокрапкою та закриваючою дужкою, є визначенням функції.

Часто використання справжніх функцій на зразок enliven()набагато прозоріше, ніж використання лямбда. Лямбда найбільш корисні у випадках, коли потрібно визначити багато дрібних функцій і запам’ятати усі їх імена.

5.11.Рекурсія

В мові програмування Python функція може викликати будь яку кількість інших функцій. Функції також можуть викликати самі себе, тобто мають властивість рекурсивності.

Рекурсія – спосіб опису об’єктів або обчислювальних процесів через самих себе. Рекурсивне програмування дозволяє описати процес що повторюється без явного використання операторів циклу.

Багато математичних функцій можна описати рекурсивно. Класичним прикладом програмування рекурсії є задача знаходження $n!$.

$$n! = \begin{cases} 1 & n = 0 \\ n * (n-1)! & n > 0 \end{cases}$$

Приклад 5.28. Рекурсія.

```
def factorial (n):
 if n>0:
 return n* factorial(n-1)
 else:
 return 1
print(factorial (5))
```

Отримаємо:

120

x

n = {

1 n = 0

$x * x$

$n-1$ n > 0

```
x=10
def rec_func (n):
 if n>0:
```

```
 return x* rec_func (n-1)
 else:
 return 1
print(rec_func (5))
```

Отримаємо:

```
100000
```

Рекурсивна функція обов’язково повинна містити хоча б одну альтернативу, що не використовує рекурсивний виклик, тобто явне визначення для деяких значень аргументів функції, тобто умову виходу (закінчення рекурсивності), щоб не спричинити зациклення програми. Кожний (новий) виклик вимагає додаткової пам’яті з ресурсу програмного стека. Якщо кількість викликів (глибина рекурсії) надмірно велика, виникає переповнення сегмента стека і операційна система вже не може створити наступний примірник локальних об’єктів функції, що як правило, веде до аварійного завершення програми.

Контрольні запитання і завдання

1. Що називають функцією?
2. Як відбувається звернення до функції?
3. Чи кожна функція повинна мати оператор повернення?
4. Що таке локальні змінні?
5. Що таке глобальні змінні?
6. Що таке фактичні параметри функції?
7. Що таке формальні параметри?
8. Чи можуть ідентифікатори фактичних і формальних параметрів співпадати?
9. Чи обов’язково кількість фактичних і формальних параметрів повинні співпадати?
10. Чи може глобальна змінна бути розташована у тілі програми?
11. Чи можна у середині однієї функції оголошувати іншу

функцію?

РОЗДІЛ 6 ГРАФІКИ

6.1. Побудова графіків в модулі turtle

Створювати графічні зображення на Python легко. Модуль turtle ("черепашка") дає змогу переміщати по екрану черепашку, яка малює картинки.

Після запуску програми з'являється графічне вікно, по якому рухається черепашка, малюючи доброзичливого робота. Можна спостерігати, як вона створює його частину за частиною.

Спочатку створимо функцію, яка малює прямокутники, а потім навчимо програму складати з них робота. Розмір і колір прямокутників можна вибирати, змінюючи аргументи функції: ноги можуть бути довгими та вузькими.

Ніколи не зберігай файли програм, що використовують черепашку, під іменем turtle.py: Python заплутається і видасть купу помилок.

Модуль turtle дає змогу керувати інструментом "черепашка", який оснащений пером. За допомогою функцій модуля черепашку можна змусити рухатися у потрібному напрямку і малювати найрізноманітніші картинки. Також можна вказати, коли опустити перо і почати малювати, а коли підняти і переміститися в іншу частину екрана, не залишаючи за собою сліду.

В оболонці Python імпортуйте всі об'єкти модуля turtle:

from turtle import *

Комбінуючи команди, можна легко створювати складні фігури та малюнки.

Приклад 6.1.Побудова графиків за допомогою модуля turtle

```
from turtle import *
color('green', 'blue')
begin_fill()
while True:
    forward(300)
    left(250)
    if abs(pos()) < 1:
        break
end_fill()
done()
```

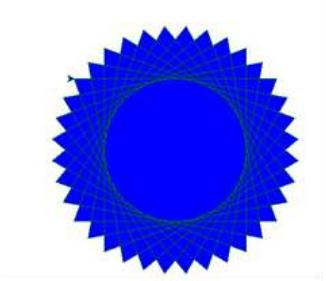


Рис.6.1 Результат виконання прикладу 6.1

6.1.1. Методи RawTurtle/Turtle та відповідні функції

turtle.forward(distance)

turtle.fd(distance)

Параметри

distance – число (ціле чи з плаваючою точкою)

Приклад 6.2. Методи RawTurtle.

Роботизована черепаха починає рух у точці (0, 0) на площині x-y. Після імпорту turtle дайте їй команду turtle.forward(15), і вона переміститься (на екрані!) на 25 пікселів у напрямку, до якого стоїть обличчям. Якщо, застосувати команду turtle.right(25), turtle повернеться на місці на 25 градусів за годинниковою стрілкою.

```
import  turtle
from turtle import *
turtle.position()
turtle.forward(25)
turtle.position()
turtle.forward(-75)
turtle.position()
done()
```

Перемістіть черепаху назад на відстань, протилежну напрямку руху черепахи.

turtle.back(distance)

turtle.bk(distance)

turtle.backward(distance)

Параметри

distance – число

Приклад 6.3. Рух черепахи.

```
import  turtle
from turtle import *
turtle.position()

turtle.backward(30)
turtle.position()
done()
```

turtle.right(angle)

turtle.rt(angle)

Параметри

angle – число (ціле чи з плаваючою точкою)

Поверніть черепаху праворуч на одиниці кут. (За замовчуванням одиницями є градуси, але їх можна встановити за допомогою функцій degrees() і radians().)

turtle.goto(x, y=None)

turtle.setpos(x, y=None)

turtle.setposition(x, y=None)

Параметри

- x – число або пара/вектор чисел
- y – число або жодного

Приклад 6.4. Побудуємо трикутник.

```
import turtle
from turtle import *
import  random
tp = turtle.pos()
```

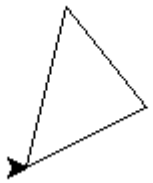


Рис.6.1 Результат виконання прикладу 6.4.

```
turtle.setpos(60,30)
turtle.pos()

turtle.setpos((20,80))
turtle.pos()

turtle.setpos(tp)
turtle.pos()

done()
```

Приклад 6.5.Встановимоь першу координату черепахи на х, другу залишимо без змін.

```
import turtle
from turtle import *
turtle.position()
turtle.setx(10)
turtle.sety(-10)
turtle.position()
done()
```

turtle.home()

Перемістіть черепаху в початкову точку – координати (0,0) – і встановіть її напрям на початкову орієнтацію (яка залежить від режиму

turtle.circle(radius, extent=None, steps=None)

Параметри

- radius – число
- extent – число (або жодного)
- steps – ціле число (або Жодного)

Приклад 6.6. Накреслимо коло із заданим радіусом

```
 rom turtle import *
 position()
 home()
 position()
 heading()
 circle(50)
 position()
 heading()
 circle(120, 380)  #
draw a semicircle
 position()
 heading()
 done()
```

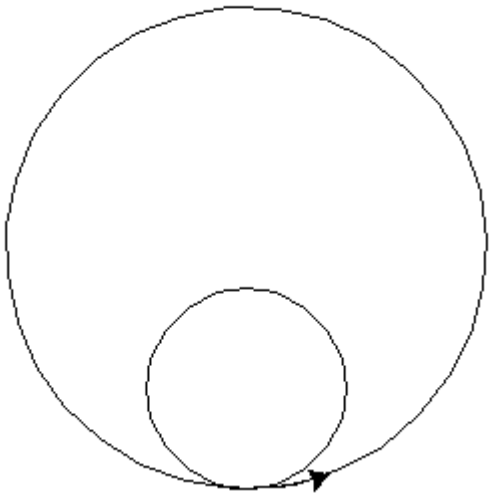


Рис.6.2. Результат виконання прикладу 6.6

turtle.dot(size=None, *color)

Параметри

- size – ціле число >= 1 (якщо задано)
- color – рядок кольорів або числовий кортеж кольорів

Приклад 6.7. Намалюємо круглу точку діаметром *розмір*, використовуючи колір.

```
import turtle
from turtle import *
```

```
turtle.home()
turtle.dot()
turtle.fd(50); turtle.dot(20, "blue"); turtle.fd(50)
turtle.position()
turtle.heading()
done()
```

6.1.2.Методи, які повертають стан об’єкту

turtle.position()

turtle.pos()

Повертає поточне розташування черепахи (x,y).turtle.towards(x, y=None)

Параметри

- *x – число або пара/вектор чисел або екземпляр черепахи*
- *y – число, якщо x є числом, інакше None*

turtle.towards(*x*, *y=None*)

Параметри

- **x** – число або пара/вектор чисел або екземпляр черепахи
- **y** – число, якщо *x* є числом, інакше None

Повертає кут між лінією від положення черепахи до положення, визначеного (x,y), вектором або іншою черепахою

Приклад 6.8. Намалюємо круглу точку діаметром розмір, використовуючи колір.

```
 import turtle
from turtle import *
turtle.goto(10, 10)
r=turtle.towards(0,0)
print(r)
done()
```

6.2.Керування пером

Потягніть ручку вниз – малюйте під час руху.

turtle.pendown()

turtle.pd()

turtle.down()

Потягніть ручку вгору - під час руху не буде малюнка

turtle.penup()

turtle.pu()

turtle.up()

turtle.pensize(width=None)

turtle.width(width=None)

Параметри

width – позитивне число

Приклад 6.9. Намалюємо квадрат.

```
import turtle
from turtle import **
color("blue")
shape("turtle")
speed(10)
pensize(4)
forward(50)#задаємо куди рухатись
right(90)
forward(50)#задаємо куди рухатись
right(90)
forward(50)#задаємо куди рухатись
right(90)
forward(50)#задаємо куди рухатись
right(90)
done()
```

Рис.6.3. Результат виконання прикладу 6.9

Приклад 6.10. Намалюємо трикутник

```
import turtle
from turtle import *color("blue")
shape("turtle")
speed(10)
pensize(4)
forward(50)#задаємо куди рухатись
left(120)
forward(50)#задаємо куди рухатись
left(120)
forward(50)#задаємо куди рухатись
right(90)
done()
```

Рис.6.3. Результат виконання прикладу 6.10

Приклад 6.11. Намалюємо багатокутник

```
import turtle
from turtle import *color("blue")
shape("turtle")
speed(10)
pensize(4)
for x in range(8):
    forward(50)#задаємо куди рухатись
    right(45)
done()
```

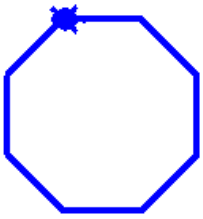


Рис.6.4. Результат виконання прикладу 6.11

Приклад 6.14. Намалюємо різнокольорову сніжинку

```
import turtle
from turtle import *

import  random
colours=['blue','red','white','yellow']
speed(10)
pensize(6)
Screen().bgcolor("turquoise")

def vshape():
  right(25)
  forward(50)
  backward(50)
  left(50)
  forward(50)
  backward(50)
  right(25)

def showflakeArm():
      for x in range(0,4):
         forward(30)
         vshape()
      backward(120)
def showflake():
   for x in range(0,35):
       color(random.choice(colours))
       showflakeArm()
       right(20)
```

Рис.6.7. Результат виконання прикладу 6.14

```
showflake()
done()
```

Приклад 6.15. Створюємо пургу

```
import turtle
from turtle import *
import  random
colours=['blue','red','white','yellow']
speed(10)

Screen().bgcolor("turquoise")

def vshape(size):
  right(25)
  forward(size)
  backward(size)
  left(50)
  forward(size)
  backward(size)
  right(25)
def showflakeArm1(size):
    for x in range(0, 4):
      forward(size)
      vshape(size)
    backward(size * 4)

    def showflake(size):
  for x in range(0,6):
    color(random.choice(colours))
```

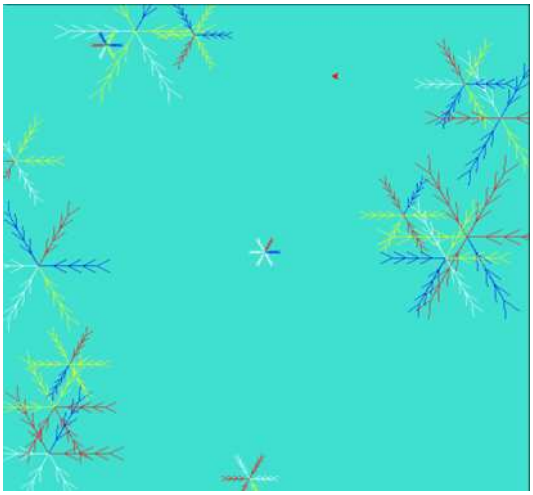


Рис.6.8. Результат виконання прикладу 6.15

```
        showflakeArm1(size)
        right(60)
    for x in range(0, 10):
        size = random.randint(5, 30)
        x = random.randint(-400, 400)
        y = random.randint(-400, 400)
        turtle.penup()
        goto(x, y)
        turtle.pendown()
        showflake(size)

showflake(5)
done()
```

Приклад 6.16. Повертає True, якщо ручка опускається, False, якщо вона піднята

```
from turtle import *
turtle.isdown()
turtle.penup()
print(turtle.isdown())
print(turtle.pendown())
print(turtle.isdown())
done()
```

Результат

```
False
None
True
```

6.2.1.Контроль кольору

turtle.pencolor(*args)

Повернути або встановити колір пера.

Дозволено чотири формати введення:

pencolor()

Повертає поточний колір пера як рядок специфікації кольору або як кортеж (див. приклад). Може використовуватися як вхідні дані для іншого виклику color/pencolor/fillcolor.

pencolor(кольоровий рядок).

Встановіть колір пера на colorstring, який є рядком специфікації кольору Tk, наприклад "red", "yellow" або "#33cc8c".

`` pencolor ((r, g, b))``

Встановіть колір пера на колір RGB, представлений кортежем r, g і b. Кожне з r, g і b має бути в діапазоні 0..colormode, де colormode дорівнює 1,0 або 255

import turtle

Приклад 6.17 Встановлюємо колір пера.

```
from turtle import *
print(colormode())

print(turtle.pencolor())

print(turtle.pencolor("brown"))
print(turtle.pencolor())

tup = (0.2, 0.8, 0.55)
turtle.pencolor(tup)
print(turtle.pencolor())
```

```
print(colormode(255))
print(turtle.pencolor())

print(turtle.pencolor('#32c18f'))
print(turtle.pencolor())

done()
```

результат

```
1.0
black
None
brown
(0.2, 0.8, 0.5490196078431373)
None
(51.0, 204.0, 140.0)
None
(50.0, 193.0, 143.0)
```

urtle.fillcolor(*args)

Повернути або встановити колір заливки.

Дозволено чотири формати введення:

fillcolor()

Повертає поточний колір заливки як рядок специфікації кольору, можливо, у форматі кортежу (див. приклад). Може використовуватися як вхідні дані для іншого виклику color/pencolor/fillcolor.

fillcolor(colorstring)

Встановіть колір заливки на colorstring, який є рядком специфікації кольору Tk, наприклад "red", "yellow" або "#33cc8c".

fillcolor((r, g, b))

Встановіть колір заливки на колір RGB, представлений кортежем r, g і b. Кожне з r, g і b має бути в діапазоні 0..colormode, де colormode дорівнює 1,0 або 255).

fillcolor(r, g, b)

Встановіть колір заливки на колір RGB, представлений r, g і b. Кожне з r, g і b має бути в діапазоні 0..colormode.

Якщо turtleshape є багатокутником, внутрішня частина цього багатокутника буде намальована новим кольором заливки.

Приклад 6.18 Колір заливки.

```
from turtle import *
print(turtle.fillcolor("violet"))
print(turtle.fillcolor())
print(turtle.pencolor())
print(turtle.fillcolor())

print( turtle.fillcolor('#ffffff'))
print(turtle.fillcolor())
done()
```

результат

None
violet
black
violet
None
(1.0, 1.0, 1.0)

Приклад 6.19. Визначаємо колір.

```
import turtle
from turtle import *
turtle.color("red", "green")
print(turtle.color())

print(color("#285078", "#a0c8f0"))
print(color())
done()
```

результат

```
('red', 'green')
None
((0.1568627450980392, 0.3137254901960784, 0.47058823529411764),
(0.6274509803921569, 0.7843137254901961, 0.9411764705882353))
```

Приклад 6.20. Задаємо розміри пензля.

```
import turtle
from turtle import *
print(turtle.begin_fill())
if turtle.filling():
   turtle.pensize(5)
else:
   turtle.pensize(3)
done()
```

Приклад 6.21. Малюємо коло та замальовуємо червоним.

```
import turtle
from turtle import *
```

```
turtle.color("black", "red")
print(turtle.begin_fill())
print(turtle.circle(80))
print(turtle.end_fill())
done()
```

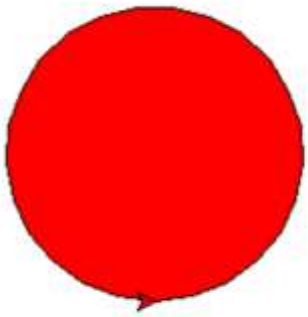


Рис.6.9. Результат виконання прикладу 6.21

Приклад 6.22. Малюємо черепаху.

```
import turtle
from turtle import *
print( turtle.shape())
print(turtle.shape("turtle"))
done()
```

Результат

classic

None

turtle.tiltangle(angle=None)

Параметри

angle – число (необов'язково)

Встановити або повернути поточний кут нахилу.

6.3. Програмний пакет Matplotlib

Matplotlib – програмний пакет на мові програмування Python для візуалізації даних з використанням двовимірної 2D і тривимірної 3D графіки.

Matplotlib – програмний пакет на мові програмування Python для візуалізації даних з використанням двовимірної 2D і тривимірної 3D графіки.

Існує ряд програмних засобів для створення графіків для доповідей та презентацій. Наприклад, можна відкрити CSV-файл в LibreOffice або Google Docs і побудувати в них графіки. Але що, якщо графіки або діаграми потрібно створювати регулярно, то для цього найкраще підходить Python і його пакет Matplotlib.

Пакет Matplotlib за будовою подібний до NumPy, SciPy і IPython та надає можливості, подібні до пакету MATLAB. На даний час пакет працює з декількома графічними бібліотеками, включаючи wxWindows і PyGTK.

Пакет підтримує наступні види графіків та діаграм:

• графіки (line plot);

• діаграми розсіювання (scatter plot);

• стовпчасті діаграми (bar chart) і гістограми (histogram);

• секторні діаграми (pie chart);

• деревоподібні діаграми (stem plot);

• контурні графіки (contour plot);

• поля градієнтів (quiver);

• спектральні діаграми (spectrogram).

Користувач може вказати вісі координат, сітку, додати підписи і пояснення, використовувати логарифмічну шкалу або полярні координати.

Нескладні тривимірні графіки можна будувати з допомогою набору інструментів (toolkit) mplot3d. Існують і інші набори інструментів: для картографії, для роботи з Excel, утиліти для GTK та інші.

З допомогою Matplotlib можна створювати як звичайні, так і анімовані зображення.

6.3.1. Графіки з однією кривою

Приклад 6.23. Приклад побудови простого графіка:

```
from pylab import *
plot(range(1, 20),
 [i * i for i in range(1, 20)], 'ro')
savefig('text.jpg')
```

show()

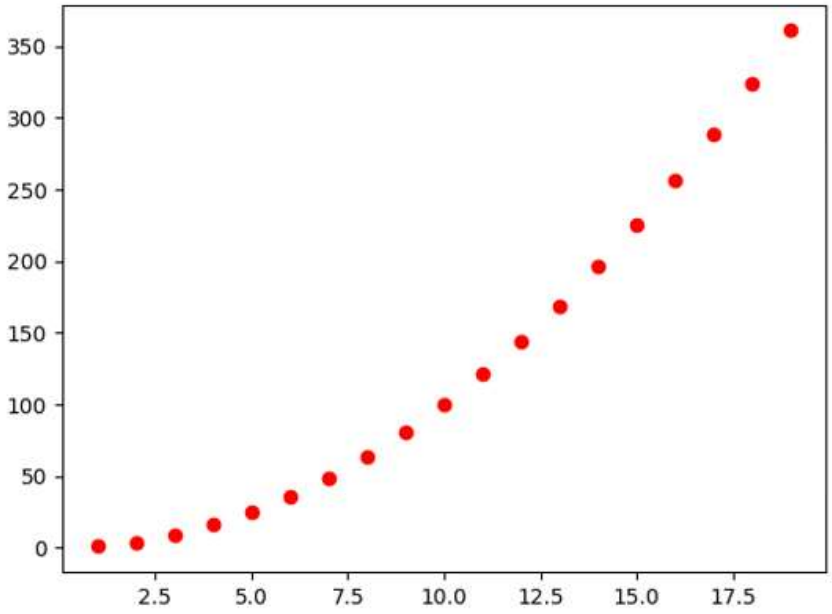
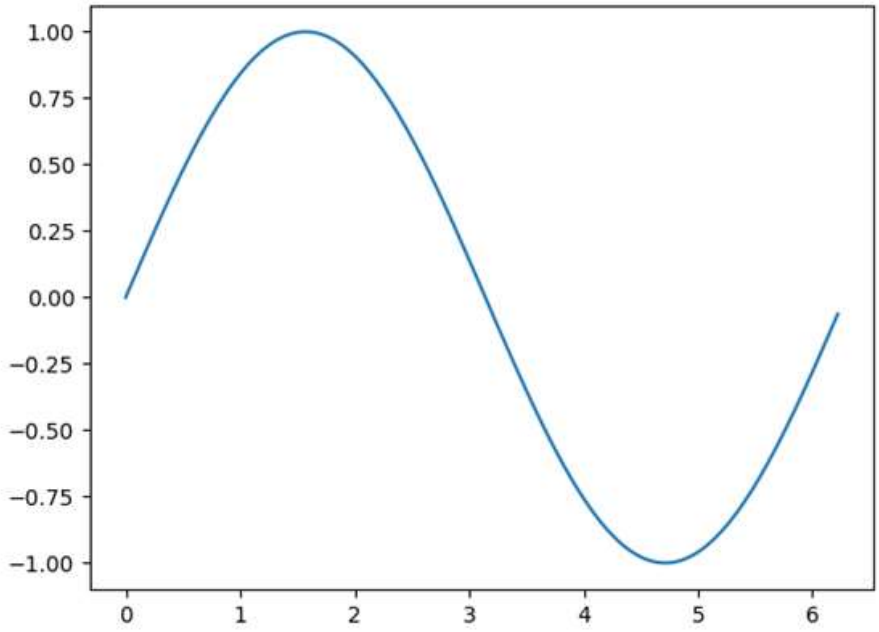


Рис.6.10. Графік прикладу 6.1

Приклад 6.24. Побудова графіка sin(x) з використанням стандартної бібліотеки Python і з пакету NumPy.

```
import math
import matplotlib.pyplot as plt
T = range(100)
X = [(2 * math.pi * t) / len(T) for t in T]
Y = [math.sin(value) for value in X]
plt.plot(X, Y)
plt.show()
import numpy as np
import matplotlib.pyplot as plt
X = np.linspace(0, 2 * np.pi, 100)
Y = np.sin(X)
plt.plot(X, Y)
plt.show()
```



6.11. Графіки sin(x)

Приклад 6.25. Побудова графіка декількох функцій

```
import numpy as np
import matplotlib.pyplot as plt
X = np.linspace(0, 2 * np.pi, 100)
Ya = np.sin(X)
Yb = np.cos(X)
plt.plot(X, Ya)
plt.plot(X, Yb)
plt.show()
```

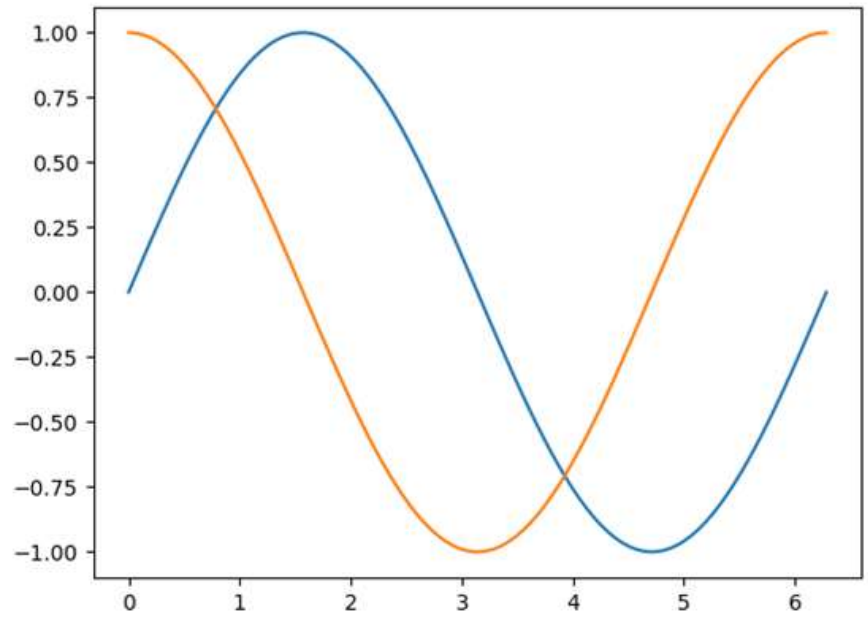


Рис.6.12. Графік декількох функцій

Приклад 6.26. Побудова графіків за даними з файлів Нехай наступні дані записані у файл my_data.txt

```
0 0
1 1
2 4
4 16
5 25
6 36
import matplotlib.pyplot as plt
X = np.linspace(0, 2 * np.pi, 100)
Ya = (X)-5
Yb = (X)+5
plt.plot(X, Ya)
plt.plot(X, Yb)
plt.show()
```

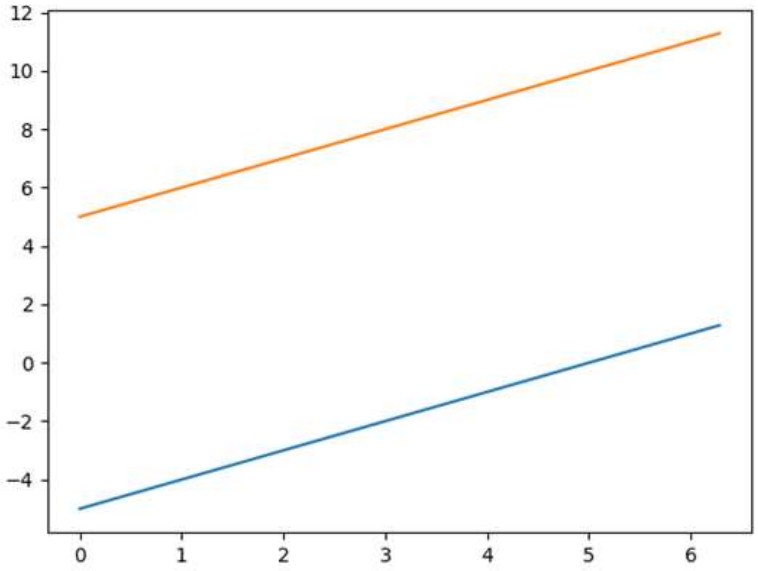


Рис.6.13. Побудова графіків за даними з файлів

Приклад 6.27. Побудова стовпчастих діаграм

```
import matplotlib.pyplot as plt
data = [5., 25., 50., 20.]
plt.bar(range(len(data)), data)
plt.show()
```

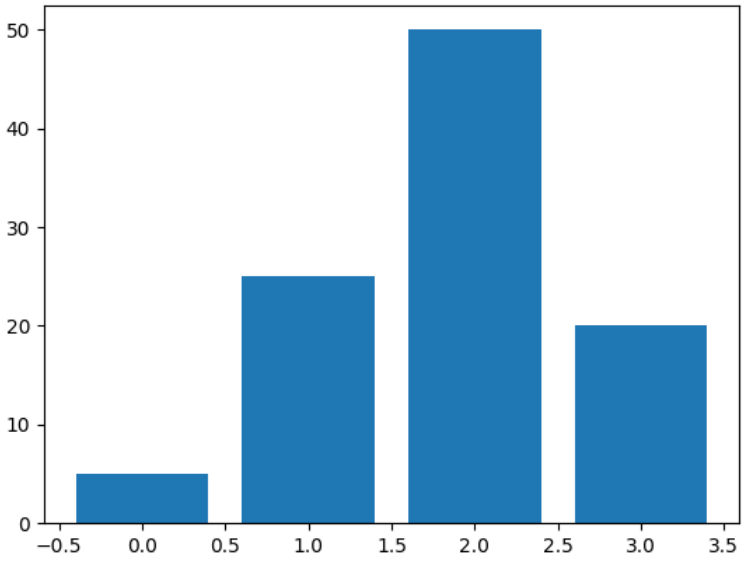


Рис.6.14. **Побудова стовпчастих діаграм**

Приклад 6.28. Побудова секторних діаграм

```
import matplotlib.pyplot as plt
data = [5, 25, 50, 20]
```

```
plt.pie(data)
plt.show()
```

Рис.6.15. Побудова секторних діаграм

Існують формати, які підтримують порядковий доступ до змінних, наприклад, numpy.recarray або pandas.DataFrame.

Matplotlib дозволяє створювати такі об’єкти з використанням ключового слова data і генерувати графіки з рядками, що відповідають цим ключам.

Приклад 6.28. Декілька підграфіків з використанням змінних категорій.

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.tri as tri
names = ["group_a", "group_b", "group_c"]
values = [1, 10, 100]
plt.figure(figsize=(9,3))
plt.subplot(131)
plt.bar(names, values)
plt.subplot(132)
plt.scatter(names, values)
plt.subplot(133)
plt.plot(names, values)
plt.suptitle("Categorucal Plotting")
```

plt.show()

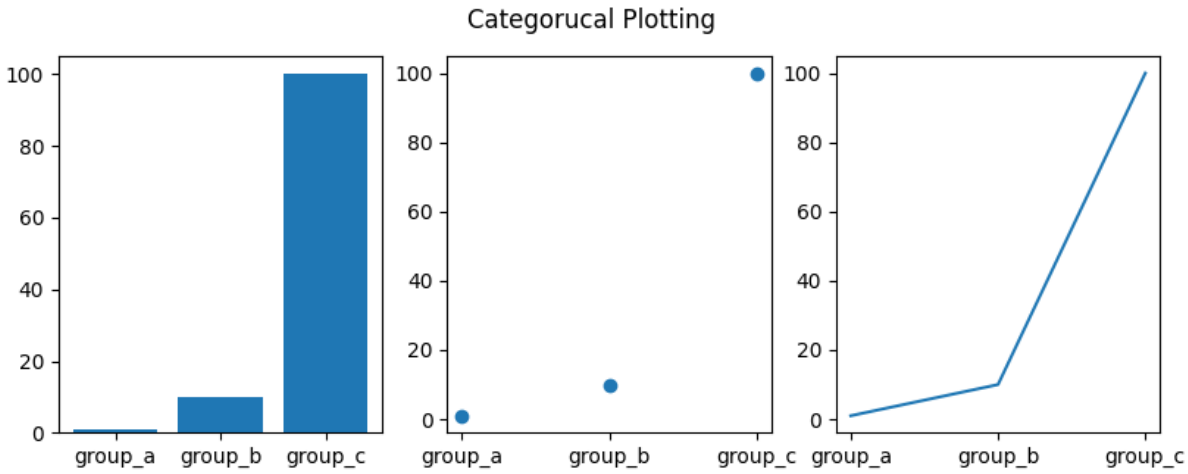


Рис.6.16. Декілька підграфіків з використанням змінних категорій

6.4. Налаштування кольору

Кольори графіків можна задавати наступними способами:

• Трійками: три числа задають значення червоного, зеленого і синього (RGB). Числа мають значення в інтервалі [0,1].

• Четвірками: три числа мають те саме значення як і у трійок, а четверте число задає

прозорість (значення також в інтервалі [0,1]).

• Зарезервовані імена кольорів: Matplotlib використовує стандартні імена кольорів HTML.

Деякі кольори позначаються одною буквою:

b – синій

g – зелений

r – червоний

c – блакитний

m – пурпуровий

y – жовтий

k – чорний

w – білий

Приклад 6.29. Задання кольору параметром color

```
import numpy as np
import matplotlib.pyplot as plt
def pdf(X, mu, sigma):
 a = 1. / (sigma * np.sqrt(2. * np.pi))
 b = -1. / (2. * sigma ** 2)
 return a * np.exp(b * (X - mu) ** 2)
X = np.linspace(-6, 6, 1000)
for i in range(5):
 samples = np.random.standard_normal(50)
 mu, sigma = np.mean(samples), np.std(samples)
 plt.plot(X, pdf(X, mu, sigma), color = 'b')
plt.plot(X, pdf(X, 0., 1.), color = 'r')
plt.show()
```

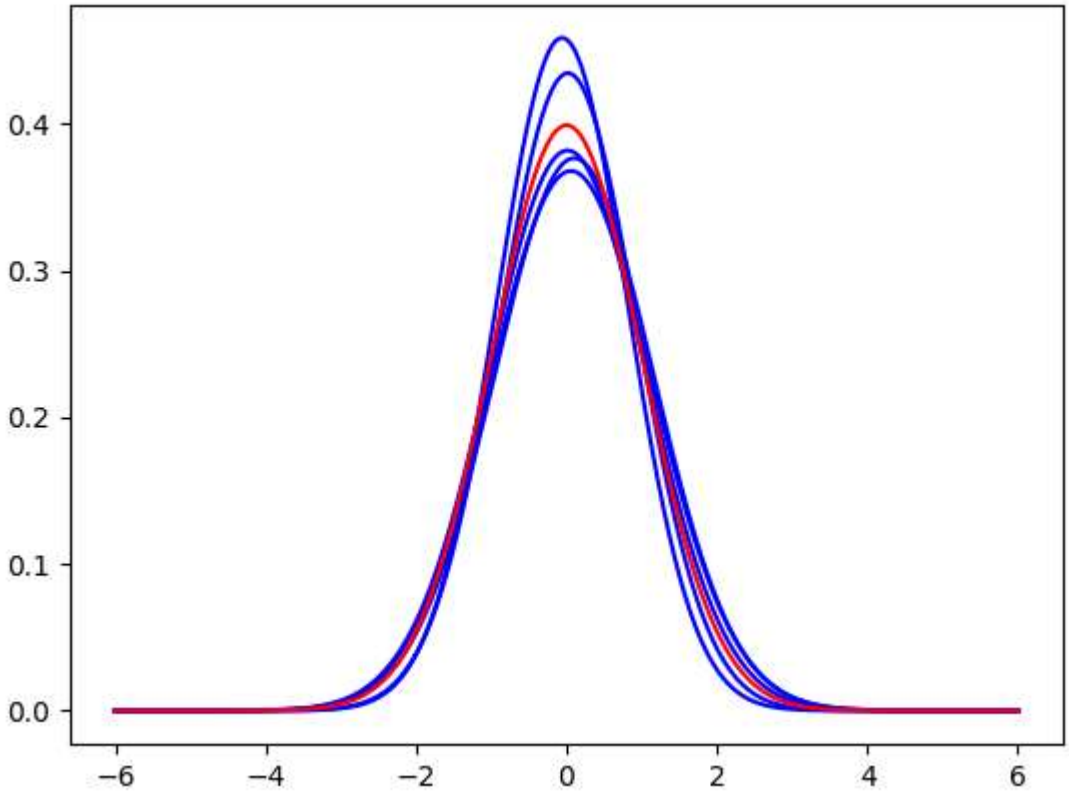


Рис.6.17. Результат виконання прикладу 6.29

Гістограма – спосіб графічного подання табличних даних, у якому кількісні співвідношення деякого показника мають вигляд прямокутників, площі яких пропорційні внеску.

Для ілюстрації цього визначення і можливості отримати гістограму розглянемо таку програму:

Приклад 6.30. Побудова гістограми

```
import matplotlib.pyplot as plt

import numpy as np
y = np.random.randn(1000)
plt.hist(y, 25)
plt.show()
```

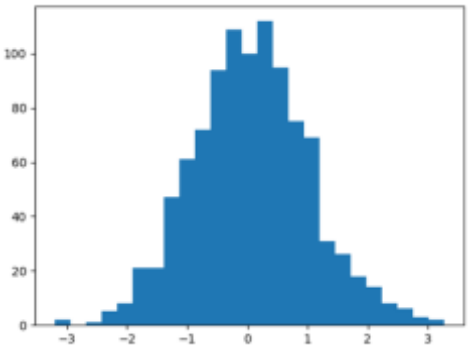


Рис.6.9. Результат виконання прикладу 6.30

Інструкція np.random.randn(1000) створюється масив із випадкових точок відповідно до гауссового розподілу.

Як видно, на відміну від раніше використовуваної функції plot() для кривих, застосовується hist() (histogram). Першим аргументом вона приймає масив чисел, другим необов'язковим аргументом є кількість смуг, на які буде розбито масив. За замовчуванням це число дорівнює десяти, однак у прикладі – 25

Контрольні запитання і завдання

1. Яке призначення пакетів візуалізації даних?
2. Які можливості з настройки вигляду графіків надає пакет Matplotlib?
3. Перерахуйте види маркерів, які можна використовувати для виведення графіків у пакеті Matplotlib.
4. Які види графіків можна будувати за допомогою пакета Matplotlib?
5. Побудуйте графіки простих математичних функцій.

6. Побудуйте гістограми та кругові діаграми за даними своєї академічної групи.

РОЗДІЛ 7. МОДУЛЬНІСТЬ В PYTHON

Якщо програмний код складний, розбиття його на окремі функції може допомогти спростити візуальне сприйняття. Якщо цього недостатньо, має сенс перемістити деякі функції та відповідні оголошення за межі основного файлу програми.

Ці додаткові файли, які містять код, який використовується в програмі, називаються модулями. Найчастіше вони містять оголошення функцій і констант, які потім можна зв'язати (імпортувати) в основну програму і вільно використовувати там. Об’єкти в модулі можна імпортувати в інші модулі. Файл створюється шляхом додавання розширення .py до імені модуля. Під час імпортування модуля інтерпретатор шукає файли спочатку в поточному каталозі, потім у каталозі, зазначеному в змінній середовища PYTHONPATH, потім у залежному від платформи шляху за замовчуванням і в спеціальному файлі з розширенням ".pth", розташованому в стандартний каталог.

Ви можете змінити PYTHONPATH і ".pth", додавши в них шляхи. Шукані каталоги можна побачити у змінній sys.path.

Як правило, великі програми складаються з файлу запуску (файл верхнього рівня) і набору файлів модулів. Керування основним обробником файлів. Водночас модуль — це більше, ніж просто фізичний файл. Модуль — це сукупність компонентів. У цьому сенсі модуль є простором імен, і всі імена всередині модуля також називаються атрибутами, такими як функції та змінні.

Існує безліч вбудованих модулів, які дозволяють виконувати складні математичні операції (math), обробляти дати (datatime)/час (time), випадкові числа (random), операційні та файлові системи (os) тощо.

7.1.Модуль math. Математичні функції

Математичний модуль math надає додаткові функції для роботи з числами, а також стандартними константами. Перед використанням модуля необхідно виконати інструкції для його підключення: import math.

Модуль math надає наступні стандартні константи:

pi – повертає число n.

e – повертає значення константи e.

Приклад 7.1 Перетворення радіан в градуси

```
import math
print(math.pi)
print(math.e)
```

Отримаємо:

3.141592653589793

2.718281828459045

7.1.1. Основні функції для роботи з числами

Стандартні тригонометричні функції: sin (), cos (), tan () – (синус, косинус, тангенс). Значення вказується в радіанах.

Зворотні тригонометричні функції asin (), acos (), atan ()(арксинус, арккосинус, арктангенс). Значення повертається в радіанах.

degrees() – перетворює радіани в градуси:

exp() – експонента;

log() – логарифм;

sqrt() – квадратний корінь;

ceil() – значення, округлене до найближчого більшого цілого;

floor() – значення, округлене до найближчого меншого цілого:

pow(Число, Степень) – підносить число до степені;

fabs() – абсолютне значення;

fmod() – остача від ділення;

factorial() – факторіал числа.

Приклад 7.1 Перетворення радіан в градуси

```
import math
print(math.degrees(math.pi))
```

Отримаємо:

180.0

radians() – перетворює градуси в радіани:

```
import math
print(math.radians(180.0))
```

Отримаємо:

3.1415926535897931

Приклад 7.2. Використання математичних функцій.

```
import math
print(math.sqrt(100), math.sqrt(25))
```

Отримаємо:

10.0, 5.0

```
import math
print(math.ceil(5.49), math.ceil(5.50), math.ceil(5.51))
```

Отримаємо:

6.0, 6.0, 6.0

```
import math
print(math.floor(5.49), math.floor(5.50), math.floor(5.51))
```

Отримаємо:

5.0, 5.0, 5.0

```
import math
print(math.pow(10, 2), 10 ** 2, math.pow(3, 3), 3 ** 3)
```

Отримаємо:

100.0, 100, 27.0, 27

```
import math
print(math.fabs(10), math.fabs(-10), math.fabs(-12.5))
```

Отримаємо:

10.0, 10.0, 12.5

```
import math
print(math.fmod(10, 5), 10 % 5, math.fmod(10, 3), 10 % 3)
```

Отримаємо:

0.0, 0, 1.0, 1

```
import math
print(math.factorial(5), math.factorial(6))
```

Отримаємо:

120, 720

7.2.Модуль random. Випадкові числа

Щоб зробити програму і непередбачуваною, але декілька способів. Один із способів – генерувати випадкові числа та використовувати їх у своїй програмі.

Python має вбудований модуль, який дозволяє генерувати псевдовипадкові числа. З математичної точки зору вони не є справді випадковими.

Модуль random дозволяє генерувати випадкові числа. Перед використанням модуля необхідно виконати інструкції для його підключення.

```
import random
```

7.2.1.Основні функції випадкових чисел

Функція random() – повертає псевдовипадкове дійсне число від 0.0 до 1.0:

Приклад 7.2.Використання випадкових чисел.

```
import random
print(random.random())
print(random.random())
print(random.random())
```

Отримаємо:

0.42888905467511462

0.57809130113447038

0.20609823213950174

Числа, що видаються функцією random(), розподілені рівномірно – це означає, що всі значення рівноймовірні.

uniform(start, end) – повертає псевдовипадкове дійсне число в діапазоні від start до end:

Приклад 7.3.Використання випадкових чисел.

```
import random
print(random.uniform(0, 10))
print(random.uniform(0, 10))
```

Отримаємо:

```
1.6022955651881965
5.206693596399246
```

Функція randint(start, end) – повертає псевдовипадкове ціле число в діапазоні від start до end.

Приклад 7.4.Використання випадкових чисел.

```
import random
print(random.randint(0, 10))
print(random.randint(0, 10))
```

Отримаємо:

```
10
6
```

Функція randrange(start, end, step) – повертає випадковий елемент з числової послідовності. Параметри аналогічні параметрам функції range(). Саме зі списку, що повертається функцією range(), і вибирається випадковий елемент:

Приклад 7.5. Випадковий елемент з числової послідовності.

```
import random
print(random.randrange(10))
print(random.randrange(0, 10))
print(random.randrange(0, 10, 2))
```

Отримаємо:

9

1

8

choice(Послідовність) – повертає випадковий елемент з будь-якої послідовності (рядку, списку, кортежу).

Приклад 7.5. Випадковий символ з рядку.

```
import random
print(random.choice("string")) # Випадковий символ з рядку
print(random.choice(["s", "t", "r"])) # Випадковий елемент зі списку
print(random.choice(("s", "t", "r"))) # Випадковий елемент з кортежу
```

Отримаємо:

't'

's'

'r'

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

Приклад 7.6. Перемішує елементи списку випадковим чином.

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

Отримаємо:

[7, 1, 6, 10, 9, 4, 8, 3, 2, 5]

Функція sample(Послідовність, Кількість елементів) – повертає список із зазначеної кількості елементів. У цей список потраплять елементи з послідовності, вибрані випадковим чином.

Як послідовність – можна вказати будь-який об’єкт, що підтримує ітерації.

Приклад 7.7. Повертаємо список із зазначеної кількості елементів.

```
import random
print(random.sample("string", 2) )
lst = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
print(random.sample(lst, 2))
print(lst) # Сам список не змінюється
print(random.sample((1, 2, 3, 4, 5, 6, 7), 3) )
```

Отримаємо:

```
['s', 'g']
[8, 7]
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
[2, 4, 3]
```

7.3.Імпорт з модулів

Для того щоб отримати доступ до функцій або змінних/констант з модуля та використати їх в основній програмі – його необхідно підключити до програми. Це можна зробити за допомогою інструкції import МОДУЛЬ, де модуль це ім’я іншого файлу Python без розширення .py.

```
import math
```

Дана команда імпортує модуль math. Тепер необхідно викликати з нього одну з функцій. Для того, щоб звернутися до змінної або функції з імпортованого модуля необхідно вказати його ім'я, поставити крапку і вказати необхідне ім’я:

модуль.функція константа

Приклад 7.8. Імпорт з модулів.

```
import math
print (math.e)
```

Отримаємо:

```
2.718281828459045
```

Запис math.e означає, що значення e знаходиться в просторі імен модуля math.

```
import math
print(math.sqrt(9))
```

Отримаємо:

```
3.0
```

Дізнатися, які функції і константи визначені в модулі можна за допомогою функції dir() .

Приклад 7.9. Використання функції dir().

```
import math
print(dir(math))
```

Отримаємо:

```
['__doc__', '__loader__', '__name__', '__package__', '__spec__',
'acos', 'acosh', 'asin', 'asinh', 'atan', 'atan2', 'atanh', 'ceil', 'copysign',
'cos', 'cosh', 'degrees', 'e', 'erf', 'erfc', 'exp', 'expm1', 'fabs',
'factorial', 'floor', 'fmod', 'frexp', 'fsum', 'gamma', 'gcd', 'hypot',
'inf', 'isclose', 'isfinite', 'isinf', 'isnan', 'ldexp', 'lgamma', 'log',
'log10', 'log1p', 'log2', 'modf', 'nan', 'pi', 'pow', 'radians', 'sin', 'sinh',
'sqrt', 'tan', 'tanh', 'trunc']
```

В результаті виконання цієї команди інтерпретатор вивів всі імена, визначені в цьому модулі. У їх числі є і змінна __doc__, що дозволяє вивести опис модуля.

Приклад 7.10. Використання змінна __doc__.

```
import math
print (math.__doc__)
```

Отримаємо:

```
This module is always available. It provides access to the
mathematical functions defined by the C standard.
```

__doc__ є внутрішнім ім’ям рядка документації як змінної всередині функції.

Приклад 7.11. Виведення рядка документації.

```
import math
```

print (math.e.__doc__)

Отримаємо:

float(x) -> floating point number

Convert a string or number to a floating point number, if possible.

Крім того, дізнатися про функції, які містить модуль, можна через функцію help().

Приклад 7.12. Використання функції help().

import math

print(help(math))

Отримаємо:

Help on built-in module math:

NAME

math

DESCRIPTION

This module is always available. It provides access to the mathematical functions defined by the C standard.

FUNCTIONS

acos(...)

acos(x)

Return the arc cosine (measured in radians) of x.

Якщо необхідно ознайомитися з описом конкретної функції модуля, то викликається довідка окремо для неї.

Приклад 7.13. Використання функції help(), для конкретної функції.

import math

print(help(math.sqrt))

Отримаємо:

Help on built-in function sqrt in module math:

sqrt(...)

sqrt(x)

Return the square root of x.

7.4.Імпорт окремої функції з модуля

В Python можна імпортувати окрему функцію з модуля за допомогою наступної конструкції

from МОДУЛЬ import ФУНКЦІЯ/КОНСТАНТА

```
from math import sqrt
print(sqrt(9))
```

Отримаємо:

```
3.0
```

Таким чином, Python не створюватиме змінну math, а завантажить в пам’ять тільки функцію sqrt(). Тепер виклик функції можна робити, не звертаючись до імені модуля math. Через кому можна перерахувати декілька функцій або змінних які необхідні.

Приклад 7.14. Використання функції help(), для конкретної функції.

```
from math import sqrt, factorial
print(sqrt(9))
print(factorial(9))
```

Отримаємо:

```
3.0
362880
```

Або вказати * і тоді можна звертатися до будь-яких функцій.

Приклад 7.15. Використання функції help(), для конкретної функції.

```
from math import *
print(sin(pi/2))
```

Отримаємо:

```
1.0
```

В якості параметра тригонометричні функції приймають значення кута в радіанах.

Приклад 7.16. Використання тригонометричні функції.

```
from math import *
print(help(sin))
```

Отримаємо:

Help on built-in function sin in module math:

sin(...)

sin(x)

Return the sine of x (measured in radians).

7.5.Створення власних модулів

Щоб створити власний модуль необхідно зберегти файл з власним ім'ям ім'я.py (для модулів обов'язково вказується розширення .py), що містить якийсь код (вміст модуля).

Наприклад створимо модуль назвавши його my_math.

Приклад 7.17. Створення власних функції.

```
def my_func ():
 print('test')
import my_math # Імпорт модуля my_math
my_math. my_func() # Виклик функції my_func
```

Результатом виконання даного коду буде:

test

Тепер спробуємо створити власний модуль.

Створіть файл з ім'ям my.py (для модулів обов'язково вказується розширення .py)і містить код (вміст нашого модуля):

```
def f ():
return 4
```

Тепер потрібно сказати Python, де шукати наш модуль. З'ясуємо через звернення до змінної path модуля sys, де Python за замовчуванням зберігає власні модулі (у вас список каталогів може відрізнятися):

Приклад 7.18. Створення власних модулів.

```
import sys
print(sys.path)
def f():
```

```
 return 4
```

отримаємо

```
['C:\\Users\\olgat\\PycharmProjects\\pythonProject3',
'C:\\Users\\olgat\\PycharmProjects\\pythonProject3',
'C:\\Users\\olgat\\AppData\\Local\\Programs\\Python\\Python311\\python311.zip',
'C:\\Users\\olgat\\AppData\\Local\\Programs\\Python\\Python311\\DLLs',
'C:\\Users\\olgat\\AppData\\Local\\Programs\\Python\\Python311\\Lib',
'C:\\Users\\olgat\\AppData\\Local\\Programs\\Python\\Python311',
'C:\\Users\\olgat\\AppData\\Local\\Programs\\Python\\Python311\\Lib\\site-packages']
```

Далі помістимо наш модуль в один з перерахованих каталогів, наприклад, в ‘C:\Users\olgat\PycharmProjects\pythonProject3’.

Якщо ми все правильно зробили, то імпортуємо наш модуль, вказавши тільки його ім'я(без розширення).

Приклад 7.19. Імпортування власних модулів.

```
import sys
print(sys.path)
def f():
 return 4
import my
my.f()
```

Тепер ми через точку можемо викликати функцію, яка знаходиться в модулі mm. Продовжимо вивчення модулів в Python. Створимо ще один модуль (по аналогії зпопереднім), вкажемо для нього інше ім'я - **mtest.py**:

```
print('test')
```

Новий модуль буде містити виклик функції print. Імпортуємо його кілька разів поспіль:

```
import mtest
import mtest
print(‘test’)
```

```
import importlib
importlib.reload(mtest)
```

Продовжимо експерименти зі модулями в Python. Створимо ще один модуль з ім'ям mypr.py.

Приклад 7.20. Імпортування модулів.

```
def func(x):
return x ** 2 + 7
x =int(input("Введіть значення: "))
print(func (x))
```

Імпорт модуля призводить до виконання всієї програми :

```
import mypr
```

7.6.Каталоги пошуку модулів

Для імпорту Python шукає файли, що зберігається в стандартному модулі sys, як змінну path. Можна отримати доступ до цього списку і змінити його (відрізняється для різних операційних систем).

Приклад 7.21. Змінна path.

```
import sys
for place in sys.path: # path містить список шляхів пошуку
модулів
print(place)
```

Результатом виконання даного коду буде:

```
D:\Users\syad\AppData\Local\Programs\Python\Python35\Lib\idl
elib
D:\Users\syad\AppData\Local\Programs\Python\Python35\python
35.zip
D:\Users\syad\AppData\Local\Programs\Python\Python35\DLLs
D:\Users\syad\AppData\Local\Programs\Python\Python35\lib
D:\Users\syad\AppData\Local\Programs\Python\Python35
D:\Users\syad\AppData\Local\Programs\Python\Python35\lib\site
-packages
```

Список sys.path містить шляхи пошуку, одержувані з наступних джерел:

– шлях до поточного каталогу з виконуваним файлом;

– значення змінної оточення PYTHONPATH. Для додавання змінної в меню Пуск необхідно обрати пункт Панель керування (або Налаштування | Панель управління). Обрати пункт Система.

Перейти на вкладку Додатково і натиснути кнопку Змінні середовища. У розділі Змінні середовища користувача натиснути кнопку Створити. В поле Ім’я змінної ввести "PYTHONPATH", а в полі Значення змінної задати шлях до папок, модулів через крапку з комою.

Після цих змін перезавантажувати комп’ютер не потрібно, достатньо заново запустити програму;

– шляхи пошуку стандартних модулів;

– вміст файлів з розширенням pth, розташованих в каталогах пошуку стандартних модулів, наприклад, в каталозі

D:\Users\syad\AppData\Local\Programs\Python\Python35\lib\sitepackages. Назва файлу може бути довільною, головне, щоб

розширення файлу було pth. Кожен шлях (абсолютний або відносний) повинен бути розташований на окремому рядку.

Каталоги повинні існувати, в іншому випадку вони не будуть додані в список sys.path.

При пошуку модуля список sys.path проглядається зліва направо. Пошук припиняється після першого знайденого модуля.

Таким чином, якщо в каталогах D:\Users\\folder1 і D:\Users\folder2

існують однойменні модулі, то буде використовуватися модуль з папки D:\Users\folder1, оскільки він розташований першим у списку шляхів пошуку.

Список sys.path можна змінювати з програми за допомогою спискових методів. Наприклад, додати каталог в кінець списку можна за допомогою методу append().

Приклад 7.22. Використання спискових методів.

import sys

```
sys.path.append(r" D:\Users\folder1") # Додаємо в кінець списку
for place in sys.path:
print(place)
```

Результатом виконання даного коду буде:

```
D:\Users\syad\AppData\Local\Programs\Python\Python35\Lib\idl\elib
D:\Users\syad\AppData\Local\Programs\Python\Python35\python
35.zip
D:\Users\syad\AppData\Local\Programs\Python\Python35\DLLs
D:\Users\syad\AppData\Local\Programs\Python\Python35\lib
D:\Users\syad\AppData\Local\Programs\Python\Python35
D:\Users\syad\AppData\Local\Programs\Python\Python35\lib\site
-packages
D:\Users\folder1
```

Приклад 7.23. Використання методу insert().

```
import sys
sys.path.insert(0, r" D:\Users\folder2") # Додаємо на
початку списку
for place in sys.path:
 print(place)
```

Результатом використання даного коду буде:

```
D:\Users\folder2
D:\Users\syad\AppData\Local\Programs\Python\Python35\Lib\idl
elib
D:\Users\syad\AppData\Local\Programs\Python\Python35\python
35.zip
D:\Users\syad\AppData\Local\Programs\Python\Python35\DLLs
D:\Users\syad\AppData\Local\Programs\Python\Python35\lib
D:\Users\syad\AppData\Local\Programs\Python\Python35
D:\Users\syad\AppData\Local\Programs\Python\Python35\lib\site
```

-packages

D:\Users\folder1

Символ r перед лапками дозволяє не інтерпретувати спеціальні послідовності. Якщо використовуються звичайні рядки, то необхідно подвоїти кожен слеш в шляху:

sys.path.append ("D:\\Users\\folder1\\folder2\\folder3")

Приклад 7.24. Тестування кожного повідомлення.

Створивши випадкове повідомлення, програма тестує з його допомогою функцій шифрування та дешифрування.

```
import random
message='Olga Tsesliv'
message = list(message)
random . shuffle(message)
message = " . join(message)
i=0
print('Test #%s : " % s ... "' % (i + 1, message [:50 ] ))
```

Контрольні запитання і завдання

1. Яким чином можна згенерувати випадкове число?
2. Для чого існує функція random()?
3. Яким чином генеруються цілі випадкові числа на певному інтервалі?
4. Як згенерувати дійсні випадкові числа на певному інтервалі?
5. Що називають функцією?
6. Як відбувається звернення до функції?
7. Чи кожна функція повинна мати оператор повернення?
8. Що таке локальні змінні?
9. Що таке глобальні змінні?

10. Числа m та k ($3 \le k \le 10$) вводяться з клавіатури. Згенерувати та вивести на екран m цілих (дійсних) випадкових чисел з проміжку, вказаному у пункті a.

11. Розробити програму, дотримуючись таких вимог:

a) число n (кількість елементів списку) – іменована константа;

b) елементи списку – псевдовипадкові числа, згенеровані на інтервалі [a, b], де a і b вводяться з клавіатури (a<b).

12. В одновимірному масиві (списку), що складається з n дійсних елементів, обчислити:

a) суму від'ємних елементів;

b) добуток елементів списку, розташованих між максимальним і мінімальним елементами.

13. Розробити програму, дотримаючись таких вимог:

a) розміри масиву n і m – ввести з клавіатури;

b) елементи масиву – псевдовипадкові числа, згенеровані на інтервалі [a, b], де a і b (a<b) вводяться з клавіатури;

c) усі вхідні та вихідні дані і також елементи початкової матриці та отриманої виводити на екран.

14. Реалізувати програму, яка міняє місцями перший і останній стовпці квадратної матриці.

15. Якщо ви запустили наведену нижче програму і вона вивела на екран число 8, то що буде виведено на екран, коли ви запустите програму наступного разу?

```
import random
random . seed ( 9)
print ( random . randint (l, 10) )
```

16. Що виведе на екран така програма?

```
spam = [1, 2, 3]
eggs = spam
ham = eggs
```

ham[0] = 99

print (ham == spam)

17. У якому модулі міститься функція deepcopy ( ) ?

18. Що виведе на екран наступна програма?

import copy

spam = [1, 2, 3]

eggs = copy . deepcopy (spam)

ham = copy . deepcopy (eggs)

ham[O] = 99

print (harn == spam)

РОЗДІЛ 8.ВИНЯТКИ

Коли модулів занадто багато, виникає необхідність їх подальшого групування. Для цього файли модулів розташовуються в папках. Відомо, що інтерпретатор шукає модулі в поточній папці та в спеціальних місцях, тому вам потрібно якимось чином вказати йому, що папка поруч із вашою програмою є не просто папкою з файлами, а також містить модуль, який ви хочете вставити . Для цього файл __init__.py повинен знаходитися в папці - він може бути порожнім, але його наявність сигналізує інтерпретатору, що папка є пакетом модуля, який можна використовувати в програмі.

Пакетом називається каталог з модулями, в якому розташований файл ініціалізації __init__.py.

Як і модулі, пакети створюють нові простори імен:

import my_package.my_math

Модуль my_math шукатиметься в пакеті my_package print(my_package.my_math.exp(1))

Отримаємо:

2.718281828459045

Всі розглянуті види імпорту поширюється також на пакети. Лише в іменах додається додатковий елемент через крапку –назва пакету. Пакети можуть вкладатися в інші пакети, аналогічно додаючи нові простори імен.

Як і модулі, пакети можуть містити код, який буде виконано під час ініціалізації пакету, – він записується в самому файлі __init__.py

8.1.Типи помилок

Винятки – це сповіщення інтерпретатора, порушувані в разі виникнення помилки в програмному коді або при настанні якої небудь події. Якщо в коді не передбачено оброблення винятків, то програма переривається і виводиться повідомлення про помилку.

Існує три типи помилок в програмі:

синтаксичні – це помилки в імені оператора або функції, невідповідність закриваючих та відкриваючих лапок і т.д. Тобто помилки в синтаксисі мови. Як правило, інтерпретатор попередить про наявність помилки, а програма не виконуватиметься зовсім.

Приклад 8.1. Синтаксичні помилки.

```
print("Невідповідність відкритих та закритих лапок!)
```

Результатом запуску даного коду буде:

```
SyntaxError: EOL while scanning string literal
```

Семантичні – це помилки в логіці роботи програми, які можна виявити тільки за результатами роботи скрипта.

Як правило, інтерпретатор не попереджає про наявність помилки. А програма буде виконуватися, оскільки не містить синтаксичних помилок. Такі помилки досить важко виявити і виправити.

Помилки часу виконання – це помилки, які виникають під час роботи скрипта. Причиною є події, які не передбачені програмістом. Класичним прикладом служить ділення на нуль.

Приклад 8.2. Помилки часу виконання.

```
def test (x, y):
 return x/y
```

```
print(test(4, 2))
print(test(4, 0))
```

Результатом виконання даного коду буде:

```
Traceback (most recent call last):
 File "G:\КПИ\2017-2018\Иностранцы\lab2.py", line 4, in
<module>
File "G:\lab.py", line 2, in test
 return x/y
ZeroDivisionError: division by zero
```

В мові Python винятки порушуються не тільки при помилці, але і як повідомлення про настання будь-яких подій. Наприклад, метод index() викликає виняток ValueError, якщо шуканий фрагмент не входить в рядок.

Приклад 8.3. Виняток ValueError.

```
print("String". index ("S"))
```

результат 0

```
print("String". index ("s"))
```

результат

```
Traceback (most recent call last):
 File "<pyshell#13>", line 1, in <module>
 "String". index ("s")
ValueError: substring not found
```

8.1.1.Оброблення винятків

Для оброблення винятків призначена інструкція try [5, 8, 9, 13]. Формат інструкції:

```
try:
 <БЛОК, В ЯКОМУ ПЕРЕХОПЛЮЮТЬСЯ
ВИНЯТКИ>
except <ВИНЯТОК_1> as <ОБ’ЄКТ ВИНЯТКУ>:
```

```
 <БЛОК, ЩО ВИКОНУЄТЬСЯ ПРИ ЗБУДЖЕННІ
ВИНЯТКУ>
…
except <ВИНЯТОК_N> as <ОБ’ЄКТ ВИНЯТКУ>:
 <БЛОК, ЩО ВИКОНУЄТЬСЯ ПРИ ЗБУДЖЕННІ ВИНЯТКУ>
e1se:
 <БЛОК, ЩО ВИКОНУЄТЬСЯ, ЯКЩО ВИНЯТКУ НЕ  ВИНИКЛО>
finally:
 < БЛОК, ЩО ВИКОНУЄТЬСЯ В БУДЬ-ЯКОМУ
ВИПАДКУ>
```

Інструкції, в яких перехоплюються винятки, повинні бути розташовані всередині блоку try. У блоці except в параметрі <Виняток_1> вказується клас оброблюваного винятку.

Приклад 8.4. Оброблення винятку, що виникає при діленні на нуль.

```
try: # Перехоплюється виняток
 x=1/0 # Помилка: ділення на 0
except ZeroDivisionError: # Вказуємо клас винятку
 print ("Обробили ділення на 0")
 x=0
 print(x)
```

Результатом виконяння даного коду буде:

```
Обробили ділення на 0
0
```

Якщо в блоці *try* згенеровано виняток, то управління передається блоку *except.* У разі, якщо виключення не відповідає зазначеному класу, управління передається наступному блоку *except.* Якщо жоден блок except не відповідає винятку, то виняток "спливає" до обробника більш високого рівня. Якщо виняток ніде не обробляється в програмі, то управління передається обробнику за замовчуванням, який зупиняє виконання програми і виводить стандартну інформацію про помилку. Таким чином, в обробнику може бути

кілька блоків *except* з різними класами винятків. Крім того, один обробник можна вкласти в інший.

Приклад 8.5. Вкладений обробник.

```
try: # Обробляється виняток
  try: # Вкладений обробник
     x=1/0 # Помилка: ділення на 0
  except NameError:
     print ("Невизначений ідентифікатор")
  except IndexError:
     print ("неіснуючий індекс")
  print ("Вираз після вкладеного обробника")
except ZeroDivisionError:
      print ("Обробка ділення на 0")
      x=0
      print(x)
```

Результатом виконання даного коду буде:

```
Оброблення ділення на 0
0
```

У вкладеному обробнику не вказано виняток ZeroDivisionError, тому виняток "спливає" до обробника більш високого рівня. Після оброблення винятку управління передається інструкції, розташований відразу після обробника.

В інструкції except можна вказати відразу кілька винятків, перерахувавши їх через кому всередині круглих дужок.

Приклад 8.6. Використання кількох винятків.

```
try:
 x = 1/0
except (NameError, IndexError, ZeroDivisionError):
# Оброблення відразу декількох винятків
 x = 0
```

```
print (x)
```

Якщо в інструкції except не вказано клас винятку, то такий блок перехоплює всі винятки.

```
try:
 x = 1/0
except: # Оброблення всіх винятків
 x = 0
 print (x)
```

На практиці слід уникати порожніх інструкцій except, оскільки можна перехопити виняток, яке є лише сигналом системі, а не помилкою. Якщо в обробнику присутній блок e1se, то інструкції всередині цього блоку будуть виконані тільки при відсутності помилок. При необхідності виконати будь-які завершальні дії незалежно від того, було згенеровано виняток чи ні, слід скористатися блоком finally.

Приклад 8.7. Використання блоком finally.

```
try:
  x = 10/2 # Немає помилки
  #x = 10/0 # Помилка: ділення на 0
except ZeroDivisionError:
   print ("Ділення на 0")
else:
   print ("Блок e1se")
finally:
   print ("Блок finally")
```

Результат виконання при відсутності винятку:

При наявності винятку і відсутності блоку except інструкції всередині блоку finally будуть виконані, але виняток не буде оброблено. Він продовжить "спливання" до обробника більш високого рівня.

Приклад 8.8. Використання інструкції всередині блоку finally.

```
try:
```

```
x = 10/0
finally:
print ("Блок fina11y")
```

Якщо користувацький обробник відсутній, то управління передається обробнику по замовчуванню, який перериває виконання програми і виводить повідомлення про помилку.

```
Traceback (most recent call last):
File "G:\lab.py", line 2, in <module>
x = 10/0
ZeroDivisionError: division by zero
```

8.2.Класи вбудованих винятків

Всі вбудовані виключення в мові Python представлені у вигляді класів.

Основна перевага використання класів для оброблення винятків полягає в можливості вказівки базового класу для перехоплення всіх винятків відповідних класів-нащадків. Наприклад, для перехоплення ділення на нуль було використано клас ZeroDivisionError. Якщо замість цього класу вказати базовий клас ArithrneticError, то перехоплюватимуться винятки класів FloatingPointError, OverflowError і ZeroDivisionError.

```
try:
x = 1/0
except ArithmeticError: # Вказано базовий клас
print ("Обробили ділення на 0")
```

Таблиця 8.1

Ієрархія вбудованих класів винятків

Виняток	Дія
BaseException	Базовий клас для всіх вбудованих винятків. Він не призначений для безпосереднього успадкування класами, визначеними користувачем (для цього використовуйте Exception). Якщо str() викликається для екземпляра цього класу, повертається представлення аргументів для екземпляра або порожній рядок, якщо аргументів не було
Exception	Усі вбудовані винятки, що не виходять із системи, походять від цього класу. Усі визначені користувачем винятки також мають бути похідними від цього класу.

AssertionError	Викликається, коли оператор assert не виконується.
EOFError	Викликається, коли функція input() досягає умови кінця файлу (EOF) без читання жодних даних. (Примітка: методи io.IOBase.read() і io.IOBase.readline() повертають порожній рядок, коли вони досягають EOF.)
ImportError	Викликається, коли оператор import має проблеми при спробі завантажити модуль. Також виникає, коли «зі списку» в from ... import має назву, яку неможливо знайти. The optional name and path keyword-only arguments set the corresponding attributes:
IndexError	Викликається, коли індекс послідовності виходить за межі діапазону. (Індекси фрагментів мовчки скорочуються, щоб потрапити в дозволений діапазон; якщо індекс не є цілим числом, виникає TypeError.)
KeyError	Викликається, коли ключ відображення (словника) не знайдено в наборі існуючих ключів.
KeyboardInterrupt	Викликається, коли користувач натискає клавішу переривання (зазвичай Control-C або Delete). Під час виконання регулярно виконується перевірка на наявність переривань. Виняток успадковується від BaseException, щоб не бути випадково перехопленим кодом, який перехоплює Exception і таким чином запобігти виходу інтерпретатора
NameError	Викликається, коли локальне чи глобальне ім’я не знайдено. Це стосується лише некваліфікованих імен. Пов’язане значення – це повідомлення про помилку, яке містить ім’я, яке не вдалося знайти.
OverflowError	Викликається, коли результат арифметичної операції завеликий для представлення.

Приклад 8.8. Створення тесту.

```
import  math
score=0
print('Тест «Тварини»')
guess1 = input('Який ведмідь живе на полюсі? ')
guess2 = input('Яка сухопутна тварина найшвидша? ')
guess3 = input('Яка тварина найбільша? ')

def check_guess(guess, answer):
   global score
   if guess.lower() == answer.lower() :
      print ('Відповідь вірна')
      score = score + 1
check_guess(guess1, 'білий')
```

```
check_guess(guess2, 'гепард')
check_guess(guess3, 'синій кит')
print(score)
print('Ви набрали: ' + str(score))
```

Результат

```
Тест «Тварини»
Який ведмідь живе на полюсі? білий
Яка сухопутна тварина найшвидша? гепард
Яка тварина найбільша? синій кит
Відповідь вірна
Відповідь вірна
Відповідь вірна
3
Ви набрали: 3
```

Приклад 8.9. Програма опитування з перенесення рядків.

```
import math
score = 0
guess = input('Яка з цих тварин риба?\n \1) Кит\n 2) Дельфін\n 3) Акула\n4) Кальмар\n Введіть 1, 2, 3 або 4.')
def check_guess(guess, answer):
   global score
   if guess.lower() == answer.lower():
      print('Відповідь вірна')
      score = score + 1
check_guess(guess, '3')
```

результат

```
Яка з цих тварин риба?
 1) Кит
 2) Дельфін
```

3) Акула

4) Кальмар

Введіть 1, 2, 3 або 4.3

Відповідь вірна

8.3.Створення функцій

Приклад 8.11. Створення власних функцій.

```
import  math
def print_secondsper_day():
    hours = 24
    minutes = hours * 60
    seconds = minutes * 60
    print(seconds)
print_secondsper_day()
```

Приклад 8.12. Аргумент days

```
import  math
def print_seconds_per_day(days):
    hours = days * 24
    minutes = hours * 60
    seconds = minutes * 60
    print(seconds)


print_seconds_per_day (7)
```

результат

604800

Приклад 8.13. Поверни значення.

```
import  math
def convert_days_to_seconds(days):
    hours = days * 24
    minutes = hours * 60
    seconds = minutes * 60
```

```
    return seconds


print(convert_days_to_seconds(5))
```

результат

```
432000
```

Паролі дають змогу захистити наші комп'ютери, поштові акаунти та облікові записи від зазіхань сторонніх осіб. Мета цього проекту - створити інструмент для генерації надійних паролів, які легко запам'ятовуються.

Приклад 8.14. "Генератор паролів"

```
import random
import string

adjectives = ['sleepy ', 'slow ', 'hot ',
'cold ', 'big ', 'red ',
'orange', 'yellow ', 'green ',
'blue ', 'good', 'old ',
'white ', 'free ', 'brave ']
nouns = ['apple ', 'dinosaur', ' ball ',
' cat ' , ' goat' , 'dragon' ,
' car ' , 'duck ', 'panda']
print ('Добрий день!')
adjective = random .choice(adjectives)
noun = random.choice(nouns)
special_char = random .choice(string. punctuation)

number = random.randrange(0, 100)
password = adjective + noun + str(number) + special_char
print ('Новий пароль: %s' % password)
```

результат

```
Добрий день!
```

Новий пароль: red cat 73%

8.4. Модулі

Модуль - це блок готового коду, який вирішує типове завдання. Включивши в програму модулі, ти зможеш зосередитися на більш цікавих завданнях**.**

Вбудовані модулі. У комплекті з Python йде безліч модулів. Усі разом вони називаються стандартною бібліотекою. Ось кілька цікавих модулів, які можуть тобі знадобитися

datetime Модуль datetime («дата время») предназначен для работы с датами. С его помощью ты сможешь узнать сегодняшнюю дату и выяснить, сколько дней осталось ждать какого-то события

statistics Модуль statistics ("статистика") допоможе знайти середнє значення або з'ясувати, яке число найчастіше зустрічається у списку. З його допомогою можна, наприклад, дізнатися середній бал серед гравців.

Random Ти вже використовував цей модуль для генерації випадкових паролів. Він хороший, якщо в код потрібно додати елемент випадковості.

Webbrowser Цей модуль дає змогу керувати веб-браузером, відкриваючи інтернет-сторінки прямо з твого коду.

Socket Модуль socket ("роз'єм") дає змогу програмам спілкуватися по мережі або через інтернет. Його можна використовувати при створенні онлайн-гри**.**

Ключове слово import відкриває доступ до всього вмісту модуля, однак назву модуля доведеться писати перед викликом кожної належної йому функції. Цей код завантажує всі функції модуля webbrowser і відкриває сайт Python за допомогою функції open() ("відкрити").

from... import...

Якщо тобі потрібна лише частина модуля, ключове слово from ("з") дасть змогу завантажити тільки її. При цьому назви функцій можна вводити як зазвичай.

Ось приклад завантаження з модуля random функції choice(), яка вибирає випадковий елемент зі списку.

from... import... as...

Іноді потрібно змінити ім'я завантажуваного модуля або функції - наприклад, тому що таке ім'я вже існує або воно погано запам'ятовується. Для цього слугує ключове слово as ("як").

Після нього потрібно написати нове ім'я. Ось приклад, у якому функція time()("час"), що повертає поточний час, завантажується під ім'ям time_now()("поточний час"). Результат роботи програми - кількість секунд, що минули з 1 січня 1970 року (дати, від якої веде відлік часу більшість комп'ютерів)

Приклад 8.15. "Генератор паролів" з використанням функції time_now()

```
import webbrowser
webbrowser. open( ' https://docs.python.org /3/library' )
from random  import choice
direction = choice(['С ', 'Ю', 'З ', 'В'])
print(direction )

from time import time as time_now
now = time_now()
print(now)
```

результат

Ю

1697644514.031677

Кількість секунд, що минули з 00:00 1 січня 1970 року

Контрольні запитання і завдання

1. Поняття помилки.

2. Що таке виняткові ситуації і яким чином здійснюється їх оброблення у Python?

3. Блоки try – except.

4. Атрибути винятків, ініціювання винятків.

5. Для чого використовується гілка finally в інструкції try?

6. Чи можна гілку finally поєднувати з гілками except?

7. Які два класи виняткових ситуацій наявні в Python?

8. Який із класів виняткових ситуацій рекомендується використовувати у програма.

9. Написати програму, яка ставить запитання та пропонує три варіанта відповіді. Якщо відповідь вірна, нараховується бал

Амортизація – це:

а). період часу, протягом якого основні засоби використовуються підприємством;

б). втрата знаряддями праці своєї вартості і поступове перенесення її на заново створений продукт з метою нагромадження коштів відтворення цих знарядь праці;

в). частка погашення балансової вартості основних засобів на оновлення;

г). вартість відтворення основних засобів на момент їх переоцінки.

10. Написати словник – каталог книг. Знайти книгу по Автору.

11. Написати програму, яка ставить запитання та пропонує три варіанта відповіді. Якщо відповідь вірна, нараховується бал

Інформаційні ресурси підприємства – це:

а). інформаційні ресурси підприємства - весь обсяг знань, відчужений від їх створювачів, зафіксований на матеріальних носіях і призначений для загального використання;

б). інформаційні ресурси організації можна розуміти як весь наявний обсяг інформації в інформаційній системі;

в). інформаційні ресурси підприємства - весь обсяг знань;

г). інформаційні ресурси підприємства - весь обсяг знань призначений для використання;

д). інформаційні ресурси підприємства - весь обсяг інформації.

РОЗДІЛ 9 СТРУКТУРИ ДАНИХ

9.1.Списки

9.1.1.Функції та методи списків

Списки(list) в Python - упорядковані змінювані колекції об'єктів довільних типів, що змінюються(майже як масив, але типи можуть відрізнятися).

Щоб використовувати списки, їх потрібно створити. Створити список можна кількома способами. Наприклад, можна обробити будь-який ітерований об'єкт (наприклад, рядок) вбудованою функцією list:

Приклад 9.1. Функція list().

```
print(list('список'))
s = []
print(s)
l = ['s', 'p', ['isok'], 2]
print(l)
```

результат

```
['с', 'п', 'и', 'с', 'о', 'к']
[]
['s', 'p', ['isok'], 2]
```

Приклад 9.2. Робота зі списками.

```
c = [c * 3 for c in 'list']
print(c)
```

результат

```
['lll', 'iii', 'sss', 'ttt']
```

Потрібно зазначити, що методи списків, на відміну від рядкових методів, змінюють сам список, а тому результат виконання не потрібно записувати в цю змінну.

Приклад 9.3. Сортування списків.

```
l = [1, 2, 3, 5, 7]
l.sort()
print(l)
l = l.sort()
print(l)
```

отримали

```
[2, 3, 5, 7, 10]
None
```

Приклад 9.4. Вставляння, додавання елементів списків.

```
a = [66.25, 333, 333, 1, 1234.5]
print(a.count(333), a.count(66.25), a.count('x'))
a.insert(2, -1)
a.append(333)
print(a)
a.index(333)
a.remove(333)
print(a)
a.reverse()
print(a)
a.sort()
print(a)
```

результат

```
2 1 0
[66.25, 333, -1, 333, 1, 1234.5, 333]
[66.25, -1, 333, 1, 1234.5, 333]
```

[333, 1234.5, 1, 333, -1, 66.25]

[-1, 1, 66.25, 333, 333, 1234.5]

Приклад 9.4. Складна конструкція генератора списків.

```
c = [c + d for c in 'list' if c != 'i' for d in 'spam' if d != 'a']
print(c)
```

результат

['ls', 'lp', 'lm', 'ss', 'sp', 'sm', 'ts', 'tp', 'tm']

9.2.Функції та методи списків

Для списків доступні основні вбудовані функції, а також методи списків

Таблиця 9.1

Методи списків

Метод	Що робить
list.append(x)	Додає елемент у кінець списку
list.extend(L)	Розширює список list, додаючи в кінець усі елементи списку L
list.insert(i, x)	Вставляє на і-ий елемент значення x
list.remove(x)	Видаляє перший елемент у списку, що має значення x
list.pop([i])	Видаляє і-ий елемент і повертає його. Якщо індекс не вказано,
list.index(x, [start [, end]])	Повертає положення першого елемента від start до end зі значенням x
list.count(x)	Повертає кількість елементів зі значенням x
list.sort([key =функція])	Сортує список на основі функції

list.reverse()	Розгортає список
list.clear()	Очищає список (нове в python 3.3)

Потрібно зазначити, що методи списків, на відміну від рядкових методів, змінюють сам список, а тому результат виконання не потрібно записувати в цю змінну.

Приклад 9.5. Методи сортування списків.

```
l = [1, 2, 3, 5, 7]
l.sort()
print(l)
l = l.sort()
print(l)
l = [1, 2, 3, 5, 7]
print(sorted(l))
```

результат

```
[1, 2, 3, 5, 7]
None
```

Приклад 9.6. Додавання елементів списку.

```
l = [1, 2, 3, 5, 7]
l.append(67)
print(l)
```

результат

```
[1, 2, 3, 5, 7, 67]
```

Приклад 9.7. Вставка елементів списку.

```
a = [66.25, 333, 333, 1, 1234.5]
print(a.count(333), a.count(66.25), a.count('x'))
a.insert(2, -1)
a.append(333)
print(a)
a.index(333)
```

```
print(a)
a.remove(333)
print(a)
a.reverse()
print(a)

a.sort()
print(a)
```

результат

```
2 1 0
[66.25, 333, -1, 333, 1, 1234.5, 333]
[66.25, 333, -1, 333, 1, 1234.5, 333]
[66.25, -1, 333, 1, 1234.5, 333]
[333, 1234.5, 1, 333, -1, 66.25]
[-1, 1, 66.25, 333, 333, 1234.5]
```

9.3.Індекси та зрізи

Приклад 9.8. Взяття елемента за індексом

```
a = [1, 3, 8, 7]
print(a[0])
print(a[3])

a = [1, 3, 8, 7]
print(a[-1])
print(a[-2])
```

результат

```
7
8
```

У Python, крім індексів, існують ще й зрізи. Функція item[START:STOP:STEP] - бере зріз від номера START, до STOP (не включаючи його), з кроком STEP. За замовчуванням START = 0, STOP =

довжина об'єкта, STEP = 1. Відповідно, які-небудь (а можливо, і все) параметри можуть бути відпущені.

Приклад 9.9. Використання зрізів.

```
a = [1, 3, 8, 7]
print( a[:])
print( a[1:])
print(a[:3])
print( a[::2])
```

результат

```
[1, 3, 8, 7]
[3, 8, 7]
[1, 3, 8]
[1, 8]
```

Якщо значення, яке роздруковуються, виявиться за межами діапазону об'єкта, друкується порожній список.

Приклад 9.10. Елементи виходять за межі списку.

```
a = [1, 3, 8, 7]
print( a[10:20])
```

результат

```
[]
```

Також за допомогою зрізів можна не тільки вилучати елементи, але і додавати і видаляти елементи (розуміється, тільки для змінених послідовностей).

Приклад 9.11. Додавання елементів за допомогою зрізів.

```
a = [1, 3, 8, 7]
a[1:3] = [0, 0, 0]
print(a)
[1, 0, 0, 0, 7]
```

```
del a[:-3]
print(a)
```

результат

```
[1, 0, 0, 0, 7]
[0, 0, 7]
```

9.4.Кортежі

Кортеж, по суті – незмінний список. Які переваги:

- Захист от дурака. То є кортеж захищений від змін, як, так і випадкових (що добре).
- Менший розмір.

Приклад 9.12. Порівняння списків та кортежів.

```
a = (1, 2, 3, 4, 5, 6)
b = [1, 2, 3, 4, 5, 6]
print(a.__sizeof__())

print(b.__sizeof__())
```

результат

```
72
88
```

Приклад 9.13. Можливість використовувати кортежі як ключі словника:

```
d = {(1, 1, 1) : 1}
print(d)

d = {[1, 1, 1] : 1}
print(d)
```

З перевагами кортежів розібралися, тепер постає питання - а як з ними працювати. Приблизно так само, як і зі списками.

9.4.1.Створення порожнього кортежу

Приклад 9.14. Створення порожнього кортежу.

```
a = tuple() # За допомогою вбудованої функції tuple()
print(a)
```

результат

```
a = () # За допомогою літерала кортежа
print(a)
```

Приклад 9.15. Створюємо кортеж з одного елемента.

```
a = ('s')
print(a)# рядок
a = 's',# кортеж
print(a)
#або
a = ('s',)
print(a)
```

Приклад 9.16. Створюємо кортеж.

```
a = tuple('hello, world!')
print(a)
htpekmnfn
#('h', 'e', 'l', 'l', 'o', ',', ' ', 'w', 'o', 'r', 'l', 'd', '!')
```

9.5. Словники

Словники в Python - невпорядковані колекції довільних об'єктів із доступом за ключу. Їх іноді ще називають асоціативними масивами або хеш-таблицями.

Щоб працювати зі словником, його потрібно створити. Створити його можна кількома способами. По-перше, за допомогою літерала:

Приклад 9.17. Створюємо словник за допомогою літерала.

```
d = {}
print(d)
d = {'dict': 1, 'dictionary': 2}
print(d)
```

результат

```
{}
{'dict': 1, 'dictionary': 2}
```

Приклад 9.18. Створюємо словник за допомогою функції dict().

```
d = dict(short='dict', long='dictionary')
print(d)

d = dict([(1, 1), (2, 4)])
print(d)
```

Приклад 9.19. Створюємо словник за допомогою методу fromkeys():

```
d = dict.fromkeys(['a', 'b'])
print(d)

d = dict.fromkeys(['a', 'b'], 100)
print(d)
```

результат

```
{'a': None, 'b': None}
{'a': 100, 'b': 100}
```

Приклад 9.20. Створюємо словник за допомогою генераторів словників.

```
d = {a: a ** 2 for a in range(7)}
print(d)
```

9.5.1.Методи словників

Основні методи роботи зі словниками:

ct.clear() - очищає словарь.

dict.copy() - повертає копію словаря.

classmethod dict.fromkeys(seq[, value]) - створює словник з ключами з seq і значенням значення (за умовчанням немає).

dict.get(key[, default]) - повертає значення ключа, але якщо його немає, не скидає виключення, а повертає значення за замовчуванням (за замовчуванням None).

dict.items() - повертає пари (ключ, значення).

dict.keys() - повертає ключі в словарі.

dict.pop(key[, default]) - видаляє ключ і повертає значення. Якщо ключа немає, повертається default (за замовчуванням бросает исключение).

dict.popitem() - видаляє і повертає пару (ключ, значення). Если словарь пуст, бросает выключение KeyError. Помните, что словари неупорядочены.

dict.setdefault(key[, default]) - повертає значення ключа, але якщо його немає, не кидає виключення, а створює ключ зі значенням

default (за замовчуванням None)

Приклад 9.22. Створюємо словник.

```
my_dictionary={}
# Додаємо елементи в масив
my_dictionary['Іван']='1111'
my_dictionary['Микола']='2222'
my_dictionary['Сергій']='3333'
my_dictionary['Петро']='4444'
my_dictionary['Артем']='5555'
my_dictionary['Андрій']='6666'
my_dictionary['Дмитро']='7777'
my_dictionary['Сашко']='8888'
# отримання значень
phona_number=my_dictionary['Сергій']
print(phona_number)
```

результат

```
3333
```

9.5.2.Типи даних та значення ключів

Приклад 9.23. Створюємо словник з ключ та значенням.

```
my_dictionary={}
my_dictionary['Вік']=27
my_dictionary[42]='відповідь'
my_dictionary['бали']=[23,67,98]
phona_number=my_dictionary['Вік']
print(phona_number)
phona_number=my_dictionary['Вік']
print(phona_number)
phona_number=my_dictionary[42]
print(phona_number)
phona_number=my_dictionary['бали']
print(phona_number)
```

9.6.Множини

Множина в python - "контейнер", що містить елементи, що не повторюються, у випадковому порядку.

Приклад 9.24. Створюємо множину.

```
a = set('hello')
print(a)

a = {'a', 'b', 'c', 'd'}
print(a)

a = {i ** 2 for i in range(10)} # генератор множин
print(a)

a = {}
print(type(a))
```

результат

{'l', 'h', 'o', 'e'}

{'c', 'd', 'b', 'a'}

{0, 1, 64, 4, 36, 9, 16, 49, 81, 25}

9.6.1.Класи множин

Найбільш розповсюджені класи множин:

• set.difference(other, ...) або set - other - ... - множина з усіх елементів set, не належні жодному з other.

• set.symmetric_difference(other); set ^ other - безліч з елементів, що зустрічаються в одній множині, але не зустрічаються в обох.

• set.copy() - копія множини.

Операції, що безпосередньо змінюють множини:

• set.update(other, ...); set | = other | ... – об'єднання.

• set.intersection_update(other, ...); set &= other & ... - перетин.

• set.difference_update(other, ...); set -= other | ... - віднімання.

• set.symmetric_difference_update(other); set ^= other - безліч з елементів, що зустрічаються в одній множині, але не зустрічаються в обох.

• set.add(elem) - додає елемент до множини.

• set.remove(elem) - видаляє елемент із множини. KeyError, якщо такого елемента не Існує.

• set.discard(elem) - видаляє елемент, якщо він знаходиться у множині.

• set.pop() - видаляє перший елемент із множини. Оскільки безлічі не впорядковані, не можна точно сказати, який елемент буде першим.

• set.clear() - очищення множини

Приклад 9.25. Приклади модулів.

Нехай в нас є формула $S = \frac{ab}{2}$ Площа в прямокутному трикутнику

```
def P1(a,b):
    return (a*b)/2

print(P1(3,5))
```

{0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36}

Приклад 9.26. Підключення модуля зі стандартної бібліотеки.

```
import os
print( os.getcwd())
import time, random
print(time.time())

print(random.random())
```

результат

```
C:\Users\olgat\PycharmProjects\pythonProject4
1705245341.5570016
0.7014939412107525
```

Приклад 9.27. Використання псевдонімів

```
import math as m
print( m.e)
```

результат

```
2.718281828459045
```

Приклад 9.28.Створення власного модуля на Python

Файл p1.py

```
def hello():
     print('Hello, world!')
def fib(n):
    a = b = 1
    for i in range(n - 2):
        b =  a + b
    return b
hello()
d=fib(5)
print(d)
```

Файл p2.py

```
import p1
p1.hello()


p1.fib(5)
print(p1.fib(5))
```

Приклад 9.29. Приклад запису даних в файл та зчитування з файлу

```
lines = ["one", "two", "three"]
with open("otus.txt", "w") as file:
    for  line in lines:
        file.write(line + '\n')


file.close()


file=open("otus.txt", "r")
text=file.read()
print(text)
file.close()
```

Приклад 9.31 Вивчаємо Python на прикладах

```
        import random
        customare=['Іваненко ','Петренко ','Шевченко ','Степаненко ']
        winner=random.choice(customare)
        flavor='пражський'
        print('Вітаємо,'+winner+'Ви отримали подарунок')
        prompt='Хочете пражський торт'
        answer=input(prompt)
        order=flavor+' торт '
        if answer=='так':
            order=order+'вишенька '
```

```
print('один '+order+ 'для '+winner)
customare=['Іваненко ','Петренко ','Шевченко ','Степаненко ']
print(customare)
```

Результат

```
['Іваненко ', 'Петренко ', 'Шевченко ', 'Степаненко ']
```

Контрольні запитання і завдання

1. Як створити список в Python?
2. Як отримати кількість значень в списку?
3. Як видалити значення із списку за індексом?
4. Як змінити значення уже існуючого елемента в списку?
5. Що так список Python?
6. Індекс елемента списку – це?
7. Як видалити всі елементи зі списку?
8. Як знайти середнє значення елементів списку?
9. Який індекс має елемент із значеням 66 списку a =[5, 4, 66, 37, 55, 7]
10. Дано список A[2;5;7;8;16;13] . Визначте, яких значень набудуть елементи списку після виконання оператора A[1] = A[3] + A[5]
11. Вам передано список, і його елементи необхідно впорядкувати в порядку убування на основі значення зазначеної властивості.

```
[
  {"a": 1, "б": 3},
  {"a": 3, "б": 2},
  {"a": 2, "б": 40},
  {"a": 4, "б": 12}
]
```

Повинен бути:

```
[
  {"a": 4, "б": 12},
```

```
  {"а": 3, "б": 2},
  {"а": 2, "б": 40},
  {"а": 1, "б": 3}
]
```

12. Належить додати довжину слів, розділених пропуском, після кожного слова і повернути її у вигляді масиву.

"apple ban" --> ["apple 5", "ban 3"]

"you will win" --> ["you 3", "will 4", "win 3"]

РОЗДІЛ 10.РОБОТА З ФАЙЛАМИ

Вивчимо як Python працює з файлами. Для читання вмісту файлу потрібно виконати такі три дії:

– відкрити файл,

– завантажити змінну

– закрити файл.

Аналогічним чином, щоб записати новий вміст у файл, необхідно відкрити (або створити) файл, записати в нього новий вміст і закрити файл.

10.1.Відкривання файлів

У Python можна відкрити файл для читання або запису за допомогою функції open( ) . Її першим аргументом є ім'я файлу, що відкривається.

Якщо файл міститься в тій самій папці, що й програма, достатньо вказати тільки ім'я файлу, наприклад ' thet imemachine . txt'. У цьому разі команда для відкриття файлу матиме такий вигляд:

fileObj = open (' dictionary.txt')

Файловий об'єкт зберігається у змінній fileObj, яка буде використовуватися в операціях читання/запису.

Можна також вказати абсолютний шлях до файлу, що включає ім'я папки, в якій знаходиться цей файл, та імена всіх батьківських папок.

Приклад 10.1. Абсолютний шлях до файлу.

```
import mm
```

```
mm.f()
import sys
print(sys.path)
```

Результат

```
['C:\\Users\\olgat\\PycharmProjects\\pythonProject3',
'C:\\Users\\olgat\\PycharmProjects\\pythonProject3',
'C:\\Users\\olgat\\AppData\\Local\\Programs\\Python\\Python311\\python311.zip',
'C:\\Users\\olgat\\AppData\\Local\\Programs\\Python\\Python311\\DLLs',
'C:\\Users\\olgat\\AppData\\Local\\Programs\\Python\\Python311\\Lib',
'C:\\Users\\olgat\\AppData\\Local\\Programs\\Python\\Python311',
'C:\\Users\\olgat\\AppData\\Local\\Programs\\Python\\Python311\\Lib\\site-packages']
```

Не забувайте про те, що в Windows символу зворотної косої риски (\) має передувати такий самий екранувальний символ.

Якщо, наприклад, ви хочете відкрити файл ***dictionary.txt***, то необхідно вказати шлях до файлу у вигляді рядка (формат рядка залежить від операційної системи).

```
dictionaryFile = open('dictionary.txt')
```

Файловий об'єкт підтримує кілька методів, призначених для запису, читання і закриття файлу.

10.2.Запис та закриття файлів

У програмі шифрування нам потрібно буде записувати зашифрований (або дешифрований) вміст у новий файл, а для цього буде потрібно використовувати метод write () .

Попередньо слід відкрити файл у режимі запису, передавши функції open () рядок 'w' як другий аргумент (цей аргумент є необов'язковим, оскільки за замовчуванням файл відкривається для читання).

Наприклад, введіть в інтерактивній оболонці таку інструкцію:

```
fileObj = open('spam.txt ', 'w')
```

Ця команда створює файл spam.txt і відкриває його в режимі запису, тим самим тим самим дозволяючи редагувати його вміст. Якщо файл із таким ім'ям уже існує в тій папці, в якій функція open( ) створює новий файл, то він буде замінений, тому будьте уважні, коли використовуєте функцію open( ) у режимі запису.

Тепер, коли файл spam.txt відкрито в режимі запису, можна виконати запис, викликавши метод write( ) . У цього методу один аргумент - рядок тексту, що підлягає запису у файл. Запишіть рядок ' Hello, world !" у файл spam. txt, ввівши в інтерактивній оболонці таку інструкцію.

Приклад 10.2. Записуємо дані в файл.

```
import sys
#print(sys.path)
dictionaryFile = open('dictionary.txt')
fileObj = open('spam.txt ', 'w')
fileObj.write (' Bello , world! ')
```

У результаті передачі рядка ' Hello, world ! ' методу write ( ) він записується у файл spam. txt, слідом за чим Python виводить на екран число 13 - кількість символів, записаних у файл.

Якщо файл більше не потрібен, повідомте Python про те, що робота з файлом завершена, викликавши метод close () файлового об'єкта:

```
fileObj.close ()
```

Підтримується також режим приєднання, який нагадує режим запису, за винятком того, що новий вміст не замінює вміст, уже наявний у файлі, а додається в його кінець. І хоча у нашій програмі цей режим не використовується, вам буде корисно знати, що для його встановлення необхідно передати рядок ' a ' як другий аргументу методу open () .

Якщо при спробі виклику методу write () ви отримуєте повідомлення про помилку. UnsupportedOperation : not headable, то це може означати, що файл не було відкрито в режимі запису. Коли функція open () викликається без вказівки необов'язкового другого параметра, вона автоматично відкриває файл у

режимі читання ('r') , що дає змогу застосовувати до даного файлового об'єкта тільки метод read ( ) .

10.3.Читання з файлу

Метод read ( ) повертає рядок, що містить весь текст, збережений у файлі. Щоб переконатися в цьому, ми прочитаємо файл spam. txt, який створили перед цим за допомогою методу wri te (). Введіть в інтерактивній оболонці такий код.

Приклад 10.3. Зчитування даних з файлу.

```
fileObj = open('spam.txt ', 'r' )
content = fileObj.read()
print(content)
fileObj. close()
```

10.4.Перевірка існування файлу

Читання файлу - безпечна операція, тоді як щодо запису в файл слід проявляти обережність. У разі виклику функції open ( ) у режимі запису для вже існуючого файлу його вихідний вміст буде замінено новим текстом. За допомогою функції os . path . exists ()можна перевірити, чи існує вказаний файл.

Функція os .path.exists ()

Функція os . path. exists () має єдиний строковий аргумент, що задає ім'я файлу або шлях до файлу, і повертає значення True, якщо вказаний файл існує, в іншому випадку повертається значення False. Функція os.path.exists () знаходиться в модулі path, який, своєю чергою, знаходиться в модулі os, тому, коли ми імпортуємо модуль os, одночасно з ним імпортується також модуль path.

Приклад 10.4. Перевірка існування файлу.

```
import os
print(os.path.exists('spam.txt' ))
print(os . path . exists ('C : \\Windows\\System32\\calc.exe '))
print(os. path . exists (' /usr/local/bin/idle3'))
```

print(os . path . exists (' /usr/bin/idle3 '))

Перевірка існування файлу за допомогою функцій

os.path.exists ()

Ми використовуємо функцію os.path.exists () для того, щоб перевірити, чи існує файл, ім'я якого міститься у змінній input Filename , інакше нам просто нічого буде шифрувати або дешифрувати.

Приклад 10.4. Перевірка існування файлу за допомогою функції os.path.exists () .

```
import sys, os
fileObj = open('spam.txt ', 'r' )
content = fileObj.read()
print(content)
if not os.path.exists('spam.txt' ):
   print ( ' The file %s does not exist . Quitt ing ... ' %( 'spam.txt') )
sys.exit ()
```

Рядкові методи, що використовуються для підвищення гнучкості користувацького введення

10.5. Строкові методи upper () , lower () і title()

Приклад 10.5. Використання методу upper ().

```
print(' Hello '. upper())
HELLO
print(' Hello '. lower())
hello
```

Споріднений метод title() повертає рядок, у якому перша літера кожного слова - великі, а всі інші - малі. Введіть в інтерактивній оболонці такі інструкції.

Приклад 10.6. Використання методу title().

```
print(' Hello '. title())
print(' extra , extra , шап bites shark    '. title())
Extra , Extra , Шап Bites Shark
```

Метод startswith () повертає значення True, якщо його строковий аргумент розташовується на початку рядка. Введіть в інтерактивній оболонці такі інструкції.

Приклад 10.7. Використання методу startswith ()

```
print('hello'.startswith('h'))
```

True

Строковий метод endswith( ) використовується для перевірки того, що одне строкове значення міститься в кінці іншого. Введіть в інтерактивній оболонці такі інструкції

```
print('Bello world'.endswith('world'))
```

Використання рядкових методів в програмі

Програма приймала будь-яке введення, яке починається з літери 'C', незалежно від її регістру. Тобто ми хочемо, щоб вміст файлу замінювався, якщо користувач вводить 'c', ' continue ', 'C' або будь-який інший рядок, що починається з букви 'C' . Для підвищення гнучкості програми ми використовуємо методи lower () і startswith ().

Приклад 10.8. Використання методів lower () і startswith ().

```
import sys, os
if os . path . exists('spam.txt' ) :
  print ( 'This will ove rwrite the file % s . (C) ont i nue or (Q) u it ? ' % ('spam.txt') )
response = input( ' > ')
if not response.lower().startswith('c'):
  sys.exit ()
```

10.7.Запис у вихідний файл

На даному етапі зашифрований (або дешифрований) вміст файлу зберігається у змінній translated. Однак після завершення роботи програми ця змінна буде загублена, тому необхідно зберегти рядок, що міститься в ній, у файлі. Відкривається новий файл (функції open( ) передається аргумент 'w') , а потім викликається метод write() файлового об'єкта.

Приклад 10.9. Запис у файл.

```
outputFileObj = open('spam.txt', 'w')
outputFileObj.write('olga')
outputFileObj.close()
```

Приклад 10.10. Запис декількох значень у файл.

```
f = open('myfile.txt', 'w')
f.write('Hello!\n')
f.write('and goodbye')
f.close()
```

Приклад 10.11. Протабулюємо функцію sin() та записуємо в файл .

```
import math
import sys, os
outputFileObj = open('spam.txt', 'w')
for i in range(0,10):
   y=str(math.sin(i))
   outputFileObj.write(y)
   outputFileObj.write('\n')
outputFileObj.close()
```

результат

```
0.0
0.8414709848078965
0.9092974268256817
0.1411200080598672
-0.7568024953079282
-0.9589242746631385
-0.27941549819892586
0.6569865987187891
0.9893582466233818
0.4121184852417566
```

Writelines() отримує в якості аргумента послідовність символьних рядків (список, кортеж, ...) і записує усі елементи послідовності у файл:

Приклад 10.12. Функція writelines()

```
a = ['Hi','there!']
f = open('myfile.txt', 'w')
f.writelines(a)
f.close()
```

10.8.Методи зчитування даних

Метод read() зчитує усі дані з текстового файла і повертає їх як один символьний рядок.

read()

Зазвичай текстова інформація у файлі розбивається на окремі рядки. Це дуже зручно отримувати дані з файлу якраз окремими рядками.

readline()

Метод readline() при кожному виклику зчитує з файлу черговий рядок і повертає його як символьний рядок. Вважається що окремі рядки тексту у файлі розділяються символом '\n' і його буде включено до результату який повертає метод readline().

readlines()

А чи можна отримати вміст усього файлу але окремими символьними рядками? Ну якщо дуже хочеться, то звісно так.

Метод readlines() зчитує усі дані з файлу, розділяє їх на окремі символьні рядки і повертає список з цих рядків.

Зауважте що символ нового рядка '\n' також буде присутнім у кожному символьному рядку.

10.9.Файл як послідовність символьних рядків

Якщо текст у файлі розбитий на окремі рядки, то його можна вважати за послідовність символьних рядків.

Напишемо програму яка зчитує з текстового файлу рядок за рядком і виводить їх.

Приклад 10.13. Зчитуємо з текстового файлу рядок за рядком.

```
import math
import sys, os
file = open('spam3.txt', 'r')
for line in file:
   print(line)
file.close()
```

Контрольні запитання і завдання

1.Як правильно: os.exists () чи os.path.exists ()?

2. Створити файл.

3. Записати текст в файл.

4. Вивести зміст файлу.

5. Закрити файл.

4. Використати функції строкові методи upper () , lower () і title()

5. Використати функції строкові методи startswith ()endswith ( )

3.Яким буде результат обчислення таких виразів?

' Foobar ' . startswi th (' Foo ')

' Foo ' . sta rtswith (' Foobar ')

' Foobar ' . startswi th (' foo ')

' bar ' . endswi th (' Foobar ')

' Foobar ' . endswi th (' bar ')

"Швидка бура лисиця відштовхнула від ледачого пса." . title ()

РОЗДІЛ 11.СТВОРЕННЯ GUI-ПРОГРАМИ

11.1.Події та програмування подій

Подієво-орієнтоване програмування орієнтоване на події. Тобто та чи інша частина програмного коду починає виконуватися лише тоді, коли трапляється та чи інша подія.

Подія – це зовнішній вплив на елемент керування, на який цей елемент може реагувати.

Функція – іменований блок коду.

Метод – це функція, що пов'язана з певним об'єктом.

Подієво-орієнтоване програмування ґрунтується на структурному та об'єктно-орієнтованому.

Програми з графічним інтерфейсом користувача (GUI- graphical user interface) є подієво-орієнтованими.

Подієво-орієнтоване програмування

У python реалізовано різні події:

■ спрацював часовий фактор,

■ хтось клікнув мишкою або натиснув Enter,

■ хтось почав вводити текст,

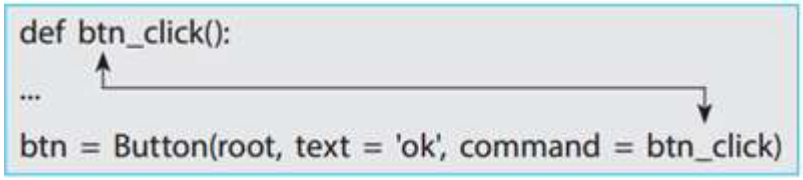
■ хтось перемкнув радіокнопки,

■ хтось прокрутив сторінку вниз

Створення обробників подій для об’єктів класу Button

Командна кнопка застосовується для запуску чи закінчення певного процесу.

Щоб прикріпити до віджета обробник події Натискання на кнопку, необхідно під час створення цього об’єкта в переліку параметрів параметру command присвоїти посилання на функцію, яка виконуватиметься в разі натискання.



Приклад 11.1 Створимо кнопку із заголовком Натисни мене!,

Кнопка виводить повідомлення Клік! (рис. 11.1). У цьому коді функція button_clicked викликається щоразу, коли користувач натисне кнопку button.

```
from tkinter import*
def button_clicked():
    print("Клік!")
root = Tk()
button1 = Button(root, bg = "red", text = "Натисни мене!", command =
button_clicked)
button1.pack()
root.mainloop()
```

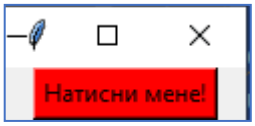


Рис.11.2.Результат виконання прикладу 11.1

11.2. Програмування реакції об’єктів на події

Можна зробити так, щоб об’єкт змінював свої властивості у відповідь на натискання зазначеної клавіші клавіатури або миші.

Таблиця 11.1.

Віджети Python

Label	Віджет Label – клас мітки. Відображає текст у вікні і служить в основному для інформаційних цілей (вивід повідомлень, підпис інших елементів інтерфейсу тощо).
Frame	Віджет Frame – клас фрейму (рамки). Цей віджет призначений для організації віджетів всередині вікна.
Button	Віджет Button – клас кнопки.
Text	Віджет Text – клас багаторядкового текстового поля.
Entry	Віджет Entry – клас однорядкового текстового поля.
Radiobutton	Віджет Radiobutton – клас перемикачів (радіокнопок).
Checkbutton	Віджет Checkbutton – клас прапорців. Цей віджет потрібен користувачам для вибору кількох елементів у вікні, що відрізняє його від перемикача, де користувач може зробити лише один вибір.
Scale	Віджет Scale – клас повзунка (шкала). Це віджет, який дозволяє вибрати будь-яке значення із заданого діапазону.
Listbox	Віджет Listbox – клас списку. Це віджет, який представляє собою список, з елементів якого користувач може вибирати один або кілька пунктів.
Canvas	Віджет Canvas – клас полотна для малювання.
Menu	Menu – клас головного меню

Змінити властивість мітки або будь-якого іншого віджета можна так:

ім'я_віджета["ім'я_властивості"] = значення

Також можна скористатись методом config():

ім'я_віджета.config(ім'я_властивості = значення)

Приклад 11.2. Створення мітки з іменем widget.

```
tkinter import *     #  імпортування графічної бібліотеки
root = Tk()    #  створення головного вікна
widget = Label(root, text='Hello GUI world!') #  створення мітки з іменем
widget, яка відображатиме на екрані взятий у лапки текст
widget.pack()#  розміщення мітки на екрані за допомогою пакувальника pack
widget.mainloop()#  задання команди відображення вікна при запуску
```

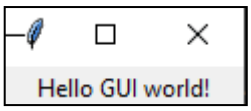


Рис.11.3.Результат виконання прикладу 11.2

Приклад 11.3. Використання віджета Frame

```
from tkinter import *# імпортування графічної бібліотеки
root = Tk()# створення головного вікна
frame1 = Frame(root, bg='blue', bd=5) # створення першої рамки
frame2 = Frame(root, bg='yellow', bd=5) # створення другої рамки
label1 = Label(frame1, text='Перша мітка')  # створення першої мітки
(вказуємо не батьківське вікно root, а рамку frame1)
label2 = Label(frame2, text='Друга мітка')  # створення другої мітки (у
рамці frame2)
frame1.pack()# розміщення у вікні першої рамки
frame2.pack()# розміщення у вікні другої рамки
label1.pack()  # розміщення у вікні першої мітки
label2.pack()  # розміщення у вікні другої мітки
root.mainloop()# задання команди відображення вікна при запуску
```

Рис.11.4.Результат виконання прикладу 11.3

Приклад 11.4. Використання віджету Button

```
from tkinter import *
root = Tk()
butt = Button(text="Змінити колір",  # задання тексту на кнопці
              width=20,  # ширина
              height=5)  # висота
# програмуємо дію за допомогою функції
def change():
    butt['text'] = "Колір змінено"  # текст на кнопці
    butt['bg'] = '#A4C639'  # колір фону після натискання, заданий
шістнадцятковим кодом
    butt['fg'] = '#333399'  # колір тексту після натискання
butt.config(command=change)  # command установлює дію на кнопку
butt.pack()  # розміщення кнопки у вікні
root.mainloop()  # задання команди відображення вікна при запуску
```

Рис.11.5.Результат виконання прикладу 11.4

Приклад 11.5. Використання віджета Text.

```
from tkinter import *  # імпортування графічної бібліотеки
root = Tk()  # створення головного вікна
# створення і розміщення багаторядкового текстового поля
text = Text(width=20,  # ширина текстового поля
            height=7,  # висота текстового поля
            bg="yellow",  # колір фону текстового поля
            fg='blue')  # колір тексту текстового поля
text.pack(side=LEFT)
# створення і розміщення скроллера (смуги прокручування)
```

scroll = Scrollbar(command=text.yview)

scroll.pack(side=LEFT, fill=Y) # властивість fill змушує віджет заповнювати весь доступний простір

Y – по вертикалі

text.config(yscrollcommand=scroll.set) # встановлення можливості прокрутки тексту

root.mainloop() # задання команди відображення вікна при запуску

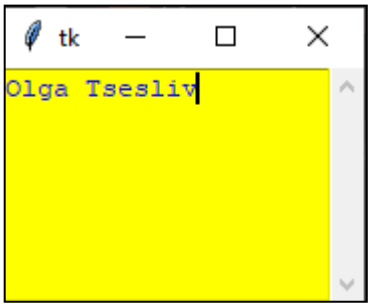


Рис.11.6.Результат виконання прикладу 11.5

https://sites.google.com/comp-sc.if.ua/python-easy/tkinter/%D0%B2%D1%96%D0%B4%D0%B6%D0%B5%D1%82%D0%B8/text-%D1%82%D0%B0-scrollbar

Приклад 11.5. Використання віджета Entry.

```
from tkinter import *# імпортування графічної бібліотеки
# створення функції для дії кнопки
def f():
    label_data['text'] = "Вас звуть %s " % entry_name.get()
root = Tk()  # створення головного вікна
name = Label(root, text="Введіть Ваше ім'я: ")     # створення та розміщення мітки з текстом "Введіть Ваше ім'я: "
name.pack()
entry_name = Entry(root) # створення та розміщення однорядкового текстового поля
entry_name.pack(pady=10)
```

button_get = Button(root, text="Згенерувати", command=f) # створення та розміщення кнопки

button_get.pack()

label_data = Label(root, fg='red') # створення та розміщення пустої мітки, яку змінює створена вище функція

label_data.pack(side=LEFT, pady=10)

root.mainloop()# задання команди відображення вікна при запуску

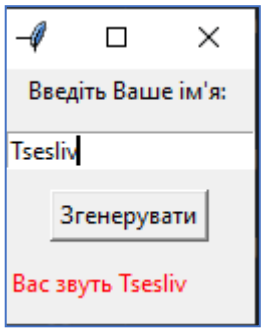


Рис.11.7.Результат виконання прикладу 11.6.

Приклад 11.7. Використання віджета Radiobutton.

```
from tkinter import *  # імпортування графічної бібліотеки
root = Tk()# створення головного вікна
val = StringVar()# створення змінної для зв'язку між перемикачами
val.set("blue")# встановлення першого значення для створеної змінної
def color():# створення функції для зміни кольору фону тексту
  label.config(bg=val.get())
# створення перемикачів
blue = Radiobutton(root, text="Синій", variable=val, value="blue", command=color)  # команда command прив'язує подію
blue.pack()
yellow = Radiobutton(root, text="Жовтий", variable=val, value="yellow", command=color)
yellow.pack()
# створення тексту
label = Label(root, text="Текст")
label.pack()
```

root.mainloop()

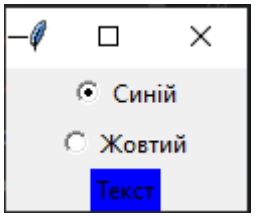


Рис.11.8.Результат виконання прикладу 11.7.

Приклад 11.8. Використання Checkbutton.

```
from tkinter import * # імпортування графічної бібліотеки
root = Tk()# створення головного вікна
# створення прапорців через цикл for
for i in range(10):
  ch_button = Checkbutton(root, text=str(i+1))
  ch_button.pack(side=LEFT) # задаємо параметр ліворуч, щоб прапорці розміщувались по горизонталі
root.mainloop()
```

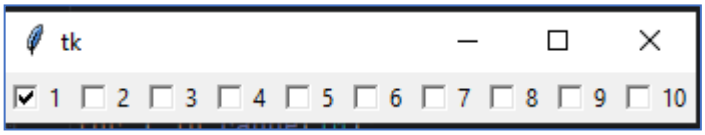


Рис.11.9.Результат виконання прикладу 11.8.

Приклад 11.9. Використання віджета Scale.

```
from tkinter import * # імпортування графічної бібліотеки
root = Tk()# створення головного вікна
# функція для друку значення
def get_value():
   print(scale.get())
# створення шкали
scale = Scale(root,
              orient=HORIZONTAL, # орієнтація повзунка
              length=300, # довжина
              from_=0, # початкове значення на шкалі
              to=10,  # кінцеве значення на шкалі
```

tickinterval=2, # інтервал, через який відображаються мітки на шкалі

resolution=2) # мінімальна відстань пересування повзунка

scale.pack()

button = Button(root, text="Друкувати значення", command=get_value) # створення кнопки

button.pack()

root.mainloop()# задання команди відображення вікна при запуску

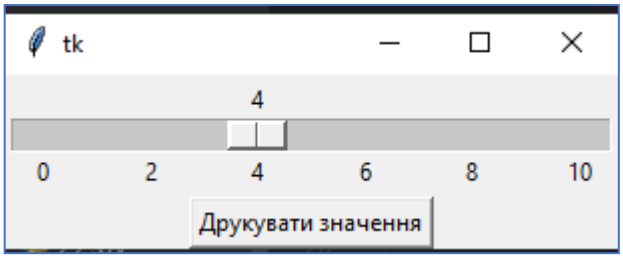


Рис.11.10.Результат виконання прикладу 11.9.

Приклад 11.10. Використання віджета Listbox.

from tkinter import * # імпортування графічної бібліотеки

root = Tk()# створення головного вікна

list1 = ["Python", "Java", "C", "C++"] # задаємо елементи, які повинні потрапити у список

listbox1 = Listbox(root, height=5, width=15, selectmode=EXTENDED) # створюємо віджет – список

for i in list1: # додаємо елементи у віджет

  listbox1.insert(END, i)

listbox1.pack()

root.mainloop()# задання команди відображення вікна при запуску

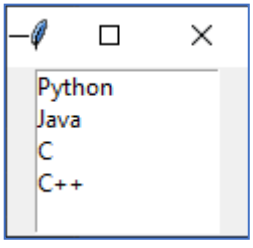


Рис.11.11.Результат виконання прикладу 11.10.

Приклад 11.11. Використання віджета Canvas.

```
from tkinter import * # імпортування графічної бібліотеки
root = Tk()# створення головного вікна
c = Canvas(root, width=200, height=200) # створення полотна шириною та висотою 200
c.pack()
c.create_line(15, 10, 180, 100) # малювання лінії з початком в точці (15,10) та кінцем (180,100)
c.create_line(50, 180, 170, 120, # координати другої лінії
        fill='red', # колір лінії
        width=5, # ширина лінії
        dash=(10, 2), # для малювання штрихами (довжина пунктиру, довжина пропуску)
        activefill='yellow', # колір лінії, коли над нею мишка
        arrow=LAST, # розміщення стрілки в кінці лінії
        arrowshape="10 30 10") # розміри стрілки
root.mainloop()
```

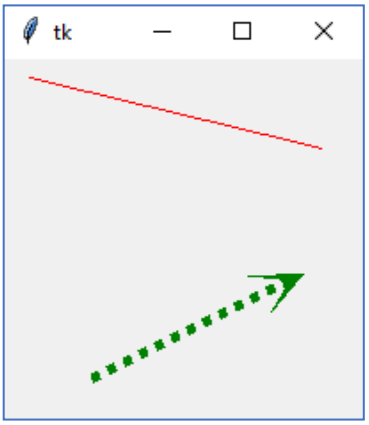


Рис.11.12.Результат виконання прикладу 11.11.

Приклад 11.12. Використання віджета Menu.

```
from tkinter import *
root = Tk()
def hello():
```

```
    print("Привіт!")
menubar = Menu(root)
root.config(menu=menubar)
menubar.add_command(label="Привіт!", command=hello)
menubar.add_command(label="Вийти", command=quit)
root.mainloop()
```

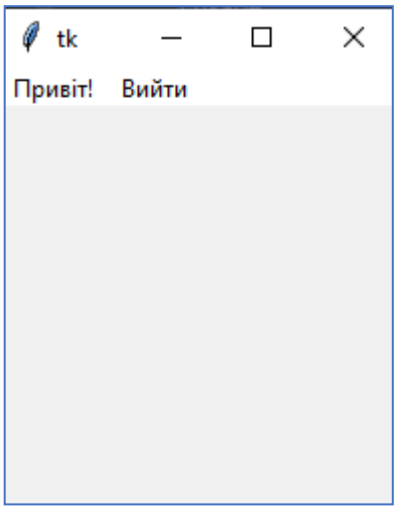


Рис.11.13.Результат виконання прикладу 11.12.

Приклад 11.13. Використання віджета Menu з підменю.

```
from tkinter import *# імпортування графічної бібліотеки
root = Tk()# створення головного вікна
root.geometry("200x150") # задання розмірів вікна
mainmenu = Menu(root) # створення головного меню
root.config(menu=mainmenu)
filemenu = Menu(mainmenu, tearoff=0) # створення підменю "Файл"
filemenu.add_command(label="Новий") # додавання команди "Новий" до підменю "Файл"
filemenu.add_command(label="Відкрити...") # додавання команди "Відкрити..." до підменю "Файл"
filemenu.add_separator()# вставлення горизонтального роздільника
filemenu.add_command(label="Вихід")# додавання команди "Вихід" до підменю "Файл"
```

helpmenu = Menu(mainmenu, tearoff=0) # створення підменю "Довідка"

mainmenu.add_cascade(label="Файл", menu=filemenu) # підв'язуємо підменю "Файл" до головного меню

mainmenu.add_cascade(label="Довідка", menu=helpmenu) # підв'язуємо підменю "Довідка" до головного меню

root.mainloop()# задання команди відображення вікна при запуску

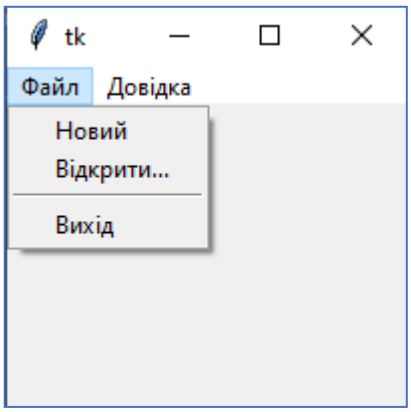


Рис.11.134.Результат виконання прикладу 11.13.

Віджет із подією й функцією-обробником події пов’язують викликом методу bind() (рис.3):

<віджет>.bind('<подія>', <функція>), де '<подія>' – текстова константа, системна назва події; <функція> – назва функції-обробника події.

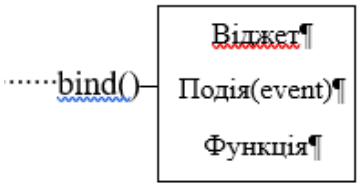


Рис.11.15. Віджет із подією й функцією-обробником події пов’язують викликом методу bind()

Наприклад, віджет – напис Label, подія – клацання по ньому лівою кнопкою миші, дія – пересування напису вправо на 5 пікселів.

Таблиця 11.1.

Перелік подій

Віджет	Подія
Button-1	натиснення лівої кнопки миші
Button-2	натиснення середньої кнопки миші
Button-3	натиснення правої кнопки миші
KeyPress	натиснення клавіші на клавіатурі над віджетом
KeyRelease	відпускання клавіші клавіатури над віджетом
ButtonPress	натиснення кнопки миші над віджетом
ButtonRelease	відпускання кнопки миші над віджетом
Motion	рух миші над віджетом
Configure	зміна розмірів або розташування вікна
Map	показ вікна (розгортання)
Unmap	приховування івікна (згортання)
Destroy	закриття вікна
FocusIn	отримання фокусу віджетом
FocusOut	втрата фокусу віджетом
Enter	курсор миші входить в область вікна
Leave	курсор миші залишає область вікна.

11.3.Створення GUI-програми

GUI, Graphical user interface) – тип інтерфейсу, який дає змогу користувачам взаємодіяти з електронними пристроями через графічні зображення та візуальні вказівки, на відміну від текстових інтерфейсів, заснованих на використанні тексту, текстовому наборі команд та текстовій навігації.

Послідовність дій при створенні GUI-програми:

1. Імпорт модуля tkinter:

from tkinter import *

Для імпорту модуля використаємо наступний код:

2. Створити головне вікно!

кранна форма — це вікно, що містить візуальні графічні елементи або об’єкти управління, такі як меню, кнопки. Для створення вікна, використовується функція Tk(). Вона має наступну структуру:

root = Tk()

11.3.1. Властивості вікна

З попереднього пункту, бачимо, що програма згенерувала вікно стандартного розміру, кольору та деяким заголовком. Якщо ж ми хочемо змінити ці налаштування, нам потрібно звертатись до властивостей вікна, а саме: *title(‘text’)* - задає заголовок вікна (по замовчуванню встановлюється “*tk*”)

root.title(‘Hello’)

geometry(‘W x H + x + y’) - задає розміри та розташування вікна (*W* – ширина, *H* – висота, *x* – відступ від лівого краю, *y* – відступ від правого краю). Задається в пікселях. Параметри *x* та *y* – необов’язкові.

root. geometry(‘W x H + x + y’)

[‘bg’] - задає колір вікна.

root[bg]=’red’

minsize(x, y) - задає мінімальний розмір вікна (x – ширина, y – висота) у пікселях. Якщо не задавати – вікно не матиме обмежень у зменшені.

root. minsize(100, 75)

maxsize(x, y) - задає максимальний розмір вікна (x – ширина, y – висота) у пікселях. Якщо не задавати – вікно не матиме обмежень у збільшені.

root. maxsize(750, 600)

Рис.11.16 Назви кольорів

Послідовність створення програм:

1. Імпорт модуля tkinter:
2. Створити головне вікно!
3. Створити віджети і виконати конфігурацію їхніх властивостей (опцій).
4. Визначити події - те, на що реагуватиме програма.
5. Описати обробники подій - те, як реагуватиме програма.
6. Розташувати віджети в головному вікні.
7. Запустити цикл обробки подій!

Змінна, яка пов'язується з об'єктом, часто називають root (корінь):

root = Tk()

11.4.Віджети

Віджет Label для відображення тексту у вікні. Застосовується в основному для інформаційних цілей (пропаганди) (виведення повідомлень, підпис інших елементів інтерфейсу).

Властивості мітки схожі на властивості кнопки. Але у міток немає опції command. Тому пов'язати мітки з подією можна ТІЛЬКИ за допомогою методу bind.

І написати lbl['command'] = ... НЕ МОЖНА(!)

На прикладі об'єкта типу Label демонстрація властивості font - шрифт.

Приклад 11.14. Використання . Label.

```
from tkinter import *
root = Tk()
l1 = Label(text="Робота над помилками", font="Arial 32")
l2 = Label(text="Розпізнання варіантів", font=("Comic Sans MS",24, "bold"))
l1. config(bd=20, bg='#ffaaaa')
l2. config(bd=20, bg='#aaffff')
l1. pack()
l2. pack()
root.mainloop()
```

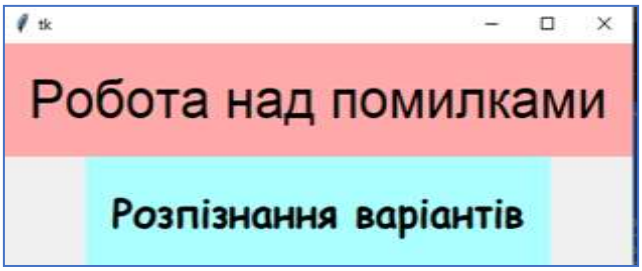


Рис.11.17.Результат виконання прикладу 11.14.

Приклад 11.15. Використання . Label та Button.

```
from tkinter import *
def take():
   lbl['text'] = "Видано"
root = Tk()
Label(root,text="Пункт видачі").pack()
```

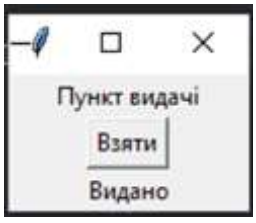


Рис.11.18.Результат виконання прикладу 11.15.

```
Button(root,text="Взяти",
command=take).pack()
lbl = Label(width=10, height=1)
lbl.pack()
root.mainloop()
```

Текстові поля призначені для введення інформації користувачем. Також і для виведення, якщо при цьому передбачається, що текст із них буде скопійовано. Текстові поля як елементи графічного інтерфейсу бувають однорядковими і багаторядковими. У tkinter багаторядковим відповідає клас Text, який буде розглянуто пізніше.

Властивості об'єктів Entry багато в чому схожі з двома попередніми віджетами. А ось методи - НІ.

З текстового поля можна взяти текст. За цю дію відповідає метод get.

У текстове поле можна вставити текст методом insert.

Також можна видалити текст методом delete.

Приклад 11.16. Створити годинник.

```
from tkinter import *
from datetime import datetime as dt
def insert_time():
  t = dt.now().time()
  e1.insert(0, t.strftime('%H:%M:%S
'))
root = Tk()
e1 = Entry(width=50) #
однострочное текстовое поле
but = Button(text="ЧАС",
command=insert_time)
e1.pack()
but.pack()
```

Рис.11.19.Результат виконання прикладу 11.16.

```
root.mainloop()
```

Приклад 11.17. Авторизація.

```
from tkinter import  messagebox
from tkinter import *
def click():
    username=username_entry.get()
    password=password_entry.get()

    messagebox.showinfo('Авторизація',
f'{username}, {password}')
```

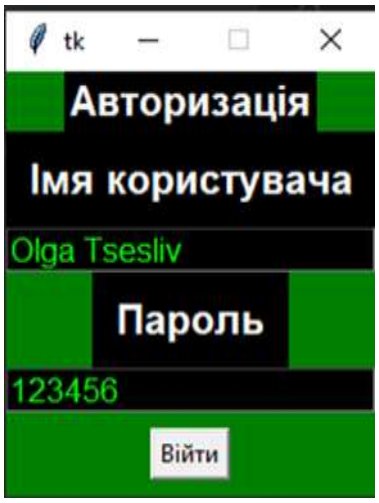


Рис.11.20.Результат виконання прикладу 11.17.

```
root=Tk()
root.title=("Авторизація")
root.geometry=('450x450')
root.resizable(width=False,height=False)
root['bg']='green'
main_label=Label(root,              text="Авторизація",
font="Arial 15 bold", bg='black',  fg='white')
main_label.pack()
username_label=Label(root,text='Імя    користувача',
font='Arial 15 bold', bg='black',  fg='white', padx=10,
pady=10)
username_label.pack()
username_entry=Entry(root,bg='black',        fg='lime',
font='Arial 12')
username_entry.pack()
password_label=Label(root,text='Пароль',  font='Arial
15 bold', bg='black',  fg='white', padx=10, pady=10)
```

```
password_label.pack()
password_entry=Entry(root,bg='black',        fg='lime',
font='Arial 12')
password_entry.pack()
send_button=Button(root,text='Вийти',
command=click)
send_button.pack(padx=10,pady=8)
root.mainloop()
```

11.4.1.Методи позиціонування елементів

Перш ніж продовжити розбиратися з віджетами GUI, бажано прояснити питання їхнього розміщення у вікні.

Під час роботи з Tkinter для позиціонування елементів застосовуються різні методи:

- pack();
- place();
- grid().

Метод grid() дає змогу помістити елемент у конкретний осередок умовної сітки або грида. При цьому

використовуються параметри:

- column - це номер стовпця, відлічується з нуля;
- row - це номер рядка, відлічується з нуля;
- columnspan - вказує число стовпців, зайнятих елементом;
- rowspan - вказує число рядків;
- ipadx і ipady - маються на увазі відступи по горизонталі та вертикалі від меж компонента до тексту компонента;
- padx і pady - аналогічні відступи, але від меж комірки грида до меж компонента;

■ sticky - визначає вирівнювання елемента в комірці в разі, коли комірка більша за компонента.

Приклад 11.18. Створюємо додаток із двома вікнами (rootO, rootl), до root() за замовчуванням чіпляється грід із 9 кнопок.

```
from tkinter import *
root0 = Tk()
root0.title('root0 500x450')
root0.geometry('500x450')
root1 = Tk()
root1.title('root1 300x250')
root1.geometry('300x250')
# Далі — грид:
for x in range(3):
    for y in range(3):
        btn = Button(text="{0}-{1}".format(x, y))
        btn.grid(row = x, column = y,
        ipadx = 10, ipady = 5,
        padx = 10, pady = 10)
# Метод mainloop використовується для виклику вікна віджету.
# Достатньо викликати одне
root0.mainloop()
```

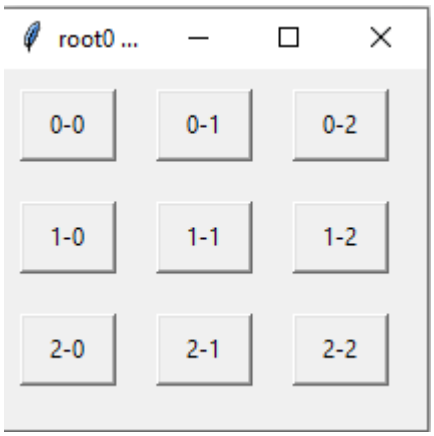


Рис.11.21.Результат виконання прикладу 11.18.

```
from tkinter import *
root0 = Tk()
root0.title('root0 500x450')
root0.geometry('500x450')
root1 = Tk()
root1.title('root1 300x250')
root1.geometry('300x250')
```

```
# Далі — грид:
for x in range(3):
    for y in range(3):
        btn = Button(text="{0}-{1}".format(x, y))
        btn.grid(row = x, column = y,
        ipadx = 10, ipady = 5,
        padx = 10, pady = 10)
# Метод mainloop  використовується для виклику вікна віджету.
# Достатньо викликати одне
root0.mainloop()
```

11.4.2.Метод pack

Це впливає на зручність застосування програми. Розміщення віджетів у вікні - це питання дизайну. Тут програміст і швець, і жнець, і на дудці гравець, і ще інтерфейси розробляє.

У Tkinter, який далі буде розглянуто в окремому розділі, існує три менеджери геометрії (так їх заведено називати) - пакувальник, сітка і розміщення за координатами.

Зараз - про пакувальник. Він найпростіший і часто використовуваний. Решта два - пізніше.

Пакувальник (packer) викликається методом pack, який є у всіх віджетів-об'єктів. Він уже багато разів застосовувався.

Якщо до елемента інтерфейсу не застосувати який-небудь із менеджерів геометрії, то він взагалі не відобразиться у вікні.

При цьому в одному вікні (або будь-якому іншому батьківському віджеті) не можна комбінувати різні менеджери.

Якщо для розміщення віджетів було застосовано метод pack, то тут же застосовувати методи grid

(сітка) і place (місце) НЕ вийде!

Якщо в пакувальники не передавати аргументи, то віджети розташовуватимуться вертикально, один один над одним. Той об'єкт, який першим викличе pack, буде вгорі. Який другим - під першим, і так далі.

У методу pack є параметр (атрибут) side (сторона), який приймає одне з чотирьох значень-констант, заданих у tkinter -

TOP, BOTTOM, LEFT, RIGHT

(верх, низ, ліворуч, праворуч).

За замовчуванням, коли в pack не вказується side, його значення встановлюється в TOP. Через це віджети розташовуються вертикально.

Приклад 11.19. Демонстрація можливостей pack.

```
l1 = Label(width=7, height=4, bg= 'yellow', text='1')
l2 = Label(width=7, height=4, bg= ' orange', text='2')
l3 = Label(width=7, height=4, bg= ' lightgreen', text='3’)
l4 = Label(width=7, height=4, bg= ' lightblue ', text='4')
11. pack()
12. pack()
13. pack()
14. pack()

11. pack(side=BOTTOM)
12. pack(side=BOTTOM)
13. pack(side=BOTTOM)
14. pack(side=BOTTOM)

11. pack(side=LEFT)
12. pack(side=LEFT)
13. pack(side=LEFT)
14. pack(side=LEFT)

11. pack(side=RIGHT)
```

12. pack(side=RIGHT)

13. pack(side=RIGHT)

14. pack(side=RIGHT)

11. pack(side=TOP)

12. pack(side=BOTTOM)

13. pack(side=RIGHT)

14. pack(side=LEFT)

l1.pack(side=LEFT)

12. pack(side=LEFT)

13. pack(side=BOTTOM)

14. pack(side=LEFT)

Приклад 11.20. Мітки та рамки.

```
from tkinter import *
root = Tk()
frm_top = Frame(root)
frm_bot = Frame(root)
# frm top, frm bot – визначають, що класти в мітку
# мітки:
# дві мітки в рамку (фрейм) frm top
lbl1 = Label(frm_top, width=7, height=4, bg='yellow', text="1")
lbl2 = Label(frm_top, width=7, height=4, bg='orange', text="2")
# дві мітки в рамку (фрейм) frm bot
lbl3 = Label(frm_bot, width=7, height=4, bg='lightgreen', text="3")
lbl4 = Label(frm_bot, width=7, height=4, bg='lightblue', text="4")
frm_top.pack()
```

```
        frm_bot.pack()
        # мітки пакуються в рамки
        lbl1.pack(side=LEFT)
        lbl2.pack(side=LEFT)
        lbl3.pack(side=LEFT)
        lbl4.pack(side=LEFT)
        # виклик вікна віджету
        root.mainloop()
```

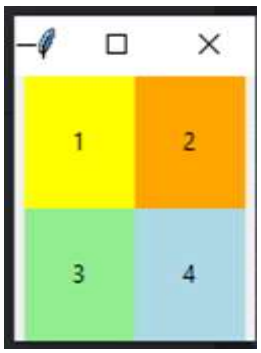


Рис.11.22.Результат виконання прикладу 11.18.

Приклад 11.21. Розв’язок квадратного рівняння.

```
from tkinter import *
from math import sqrt
global D
def solver(a,b,c):
    D = b*b - 4*a*c    # обчислення дискримінанта
    if D >= 0:
        x1 = (-b + sqrt(D)) / (2*a);
        x2 = (-b - sqrt(D)) / (2*a)
        text = "Дискримінант: %s \n x1 = %s \n x2 = %s \n" % (D, x1, x2)
    else:
        text = "Дискримінант: %s \n Рівняння розв'язків немає" % D
    return text
        def inserter(value):
            """ Очищення та вставка значень """
            output.delete("0.0","end")
            output.insert("0.0",value)
```

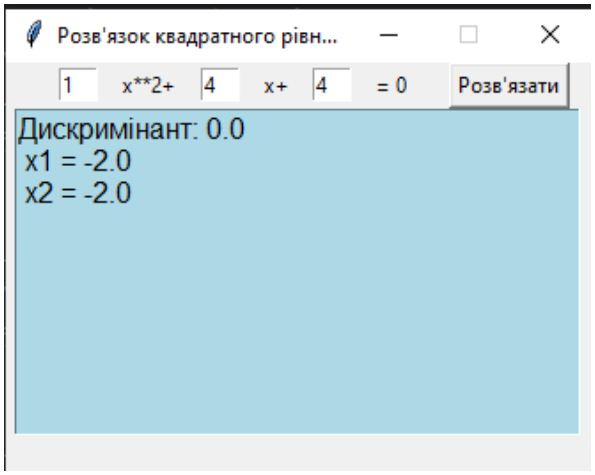


Рис.11.23.Результат виконання прикладу 11.21.

```
def clear(event):
    """ Очистка текстових полів об'єкта Entry  """
    caller = event.widget
    caller.delete("0", "end")

def handler():# функція обробки
# Отримати значення коефіцієнтів з перевіркою на коректність
    try:
        a_val = float(a.get())
        b_val = float(b.get())
        c_val = float(c.get())
        inserter(solver(a_val, b_val, c_val))
        x1 = (-b + sqrt(D)) / 0;
        x2 = (-b - sqrt(D)) /0
    except ZeroDivisionError:
            print("You can't divide by zero")

root = Tk()
root.title("Розв'язок квадратного рівняння")
root.geometry("330x230+300+100")
root.resizable(width=False, height=False)
#
frame = Frame(root)         #
frame.pack()

a = Entry(frame, width=3)   #
a.grid(row=1,column=1,padx=(10,0))
a.bind("<FocusIn>", clear)
al = Label(frame, text="x**2+").grid(row=1,column=2)
```

```
b = Entry(frame, width=3)   #
b.bind("<FocusIn>", clear)
b.grid(row=1,column=3)
bl = Label(frame, text="x+").grid(row=1, column=4)

c = Entry(frame, width=3)   #
c.bind("<FocusIn>", clear)
c.grid(row=1, column=5)
cl = Label(frame, text="= 0").grid(row=1, column=6)

but = Button(frame, text="Розв'язати", command=handler).grid(row=1,
        column=7, padx=(10,0))  #

output = Text(frame, bg="lightblue", font="Arial 12", width=35, height=10)
output.grid(row=2, columnspan=8)

root.mainloop()
```

Приклад 11.22.Побудова графіків з використанням модулів.

```
from tkinter import *

import tsesliv
import numpy as np
def click():
    tsesliv.click1()
    # Creating the tkinter Window
root = Tk()
root.geometry("200x100")
    # Button for closing
button = Button(root, text="Grafik",width=20, height=40,
bg='greenyellow',command=click)
```

```
button.pack(pady=20)
root.mainloop()
def click1():
    import matplotlib.pyplot as plt
    X = np.linspace(0, 2 * np.pi, 100)
    Ya = np.sin(X)
    Yb = np.cos(X)
    plt.plot(X, Ya)
    plt.plot(X, Yb)
    plt.show()
```

файл Tsesliv.py

```
import numpy as np
def click1():
    import matplotlib.pyplot as plt
    X = np.linspace(0, 2 * np.pi, 100)
    Ya = np.sin(X)
    Yb = np.cos(X)
    plt.plot(X, Ya)
    plt.plot(X, Yb)
    plt.show()
```

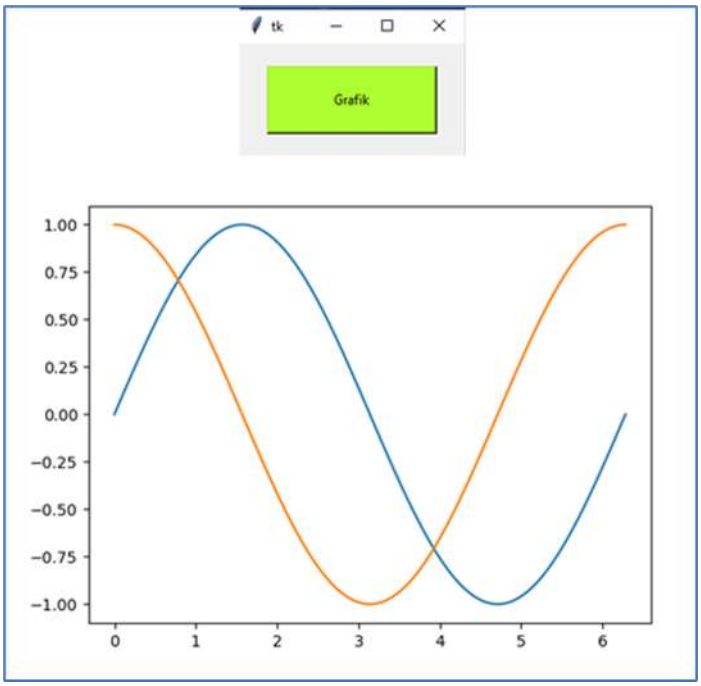


Рис.11.24.Результат виконання прикладу 11.22.

Приклад 11.23. Покомпонентний добуток масивів.

```
import tkinter as tk

def обчислити_добуток():
    масив1 = [int(x) for x in entry1.get().split()]
```

```
            масив2 = [int(x) for x in entry2.get().split()]

            if len(масив1) != len(масив2):
                result_label.config(text="Масиви повинні мати однаковий розмір.")
            else:
                добуток_масивів = [a * b for a, b in zip(масив1, масив2)]
                result_label.config(text=f"Добуток масивів: {добуток_масивів}",
font=("Arial", 12, "bold"), fg="#FF4081")

        window = tk.Tk()
        window.title("Калькулятор добутку масивів")

        window.configure(bg="#FFC0CB")
        label1 = tk.Label(window, text="Введіть перший масив:",
bg="#FFC0CB")
        entry1 = tk.Entry(window)

        label2 = tk.Label(window, text="Введіть другий масив:", bg="#FFC0CB")
        entry2 = tk.Entry(window)

        calculate_button = tk.Button(window, text="Обчислити добуток",
command=обчислити_добуток, bg="#FF69B4", fg="white")
        result_label = tk.Label(window, text="Результат буде виведено тут.",
bg="#FFC0CB")

        window_width = 400
        window_height = 250
        screen_width = window.winfo_screenwidth()
        screen_height = window.winfo_screenheight()
        x_position = (screen_width - window_width) // 2
```

```
y_position = (screen_height - window_height) // 2
window.geometry(f"{window_width}x{window_height}+{x_position}+{y_position}")

label1.pack(pady=10)
entry1.pack(pady=10)

label2.pack(pady=10)
entry2.pack(pady=10)

calculate_button.pack(pady=20)
result_label.pack()

window.mainloop()
```

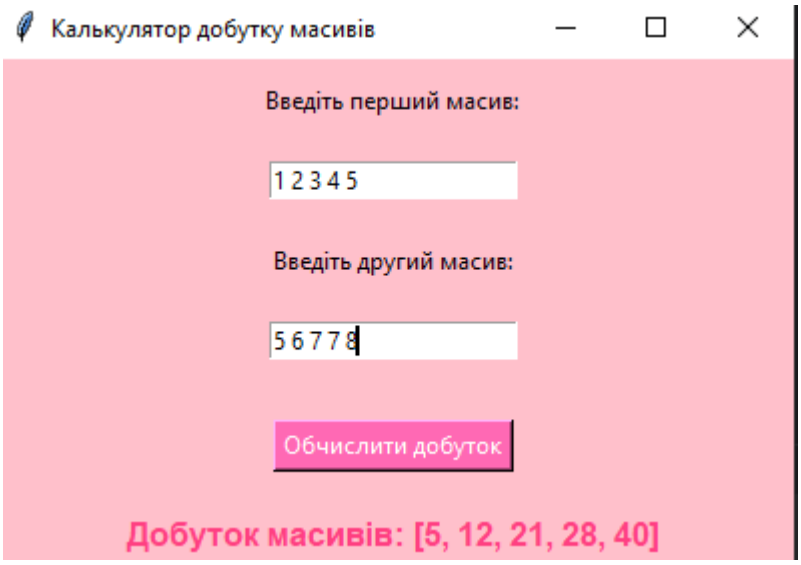
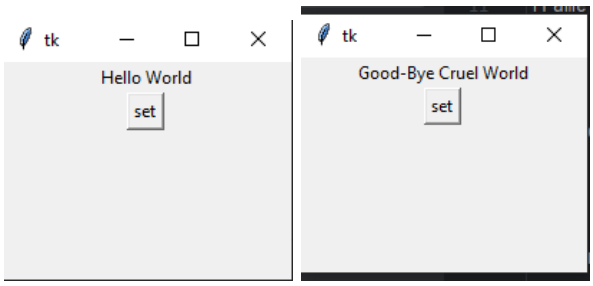


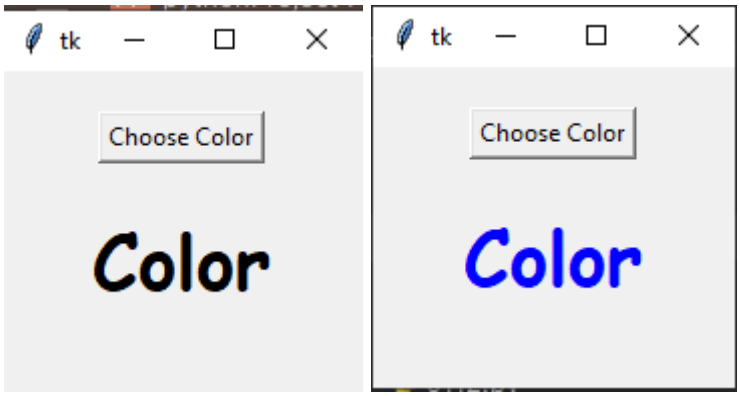
Рис.11.25. Результат виконання прикладу 11.23.

Контрольні запитання і завдання

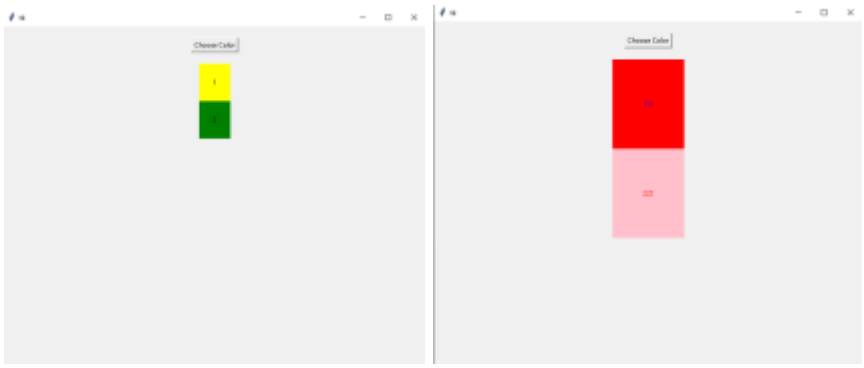
1. Що таке віджети в Python?
2. Які віджети ви знаєте?
3. Які подій ви знаєте?
4. Що таке GUI?



5. Створити програму з кнопкою. При натисканні кнопки змінюється текст на Label



6.Створити програму з кнопкою. При натисканні кнопки змінюється текст на Label



7.Створити програму з кнопкою. При натисканні кнопки змінюється текст та колір на Label1 та Label2

8.Знайти значення Інтеграла

9.Дописати try перевірити ділення на 0 в задачі квадатного рівняння.

10.В задачі авторизації перевірити логін та пароль.

РОЗДІЛ 12. РОБОТА PYTHON+EXCEL

12.1 Підключення бібліотеки

Для роботи з EXCEL, необхідно під’єднати бібліотеку openpyxl. Бібліотека openpyxl для читання/запису файлів Excel 2010 xlsx/xlsm. Документація по бібліотеці знаходиться на сайті https://openpyxl.readthedocs.io/en/stable/. Процес підключення бібліотеки показаний на рис. 12.1 та рис.12.2.

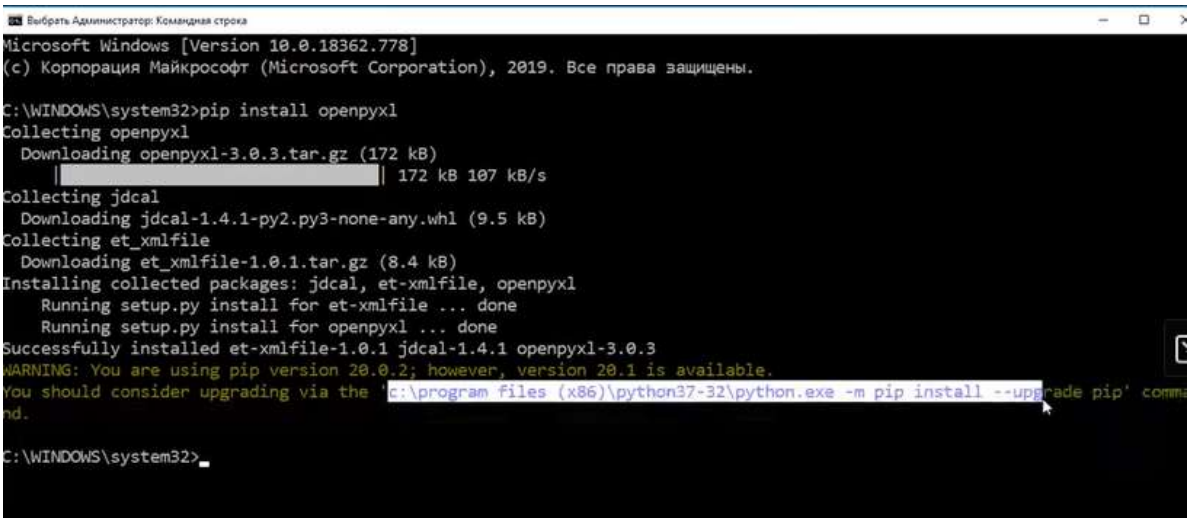


Рис.12.1. Підключення бібліотеки openpyxl

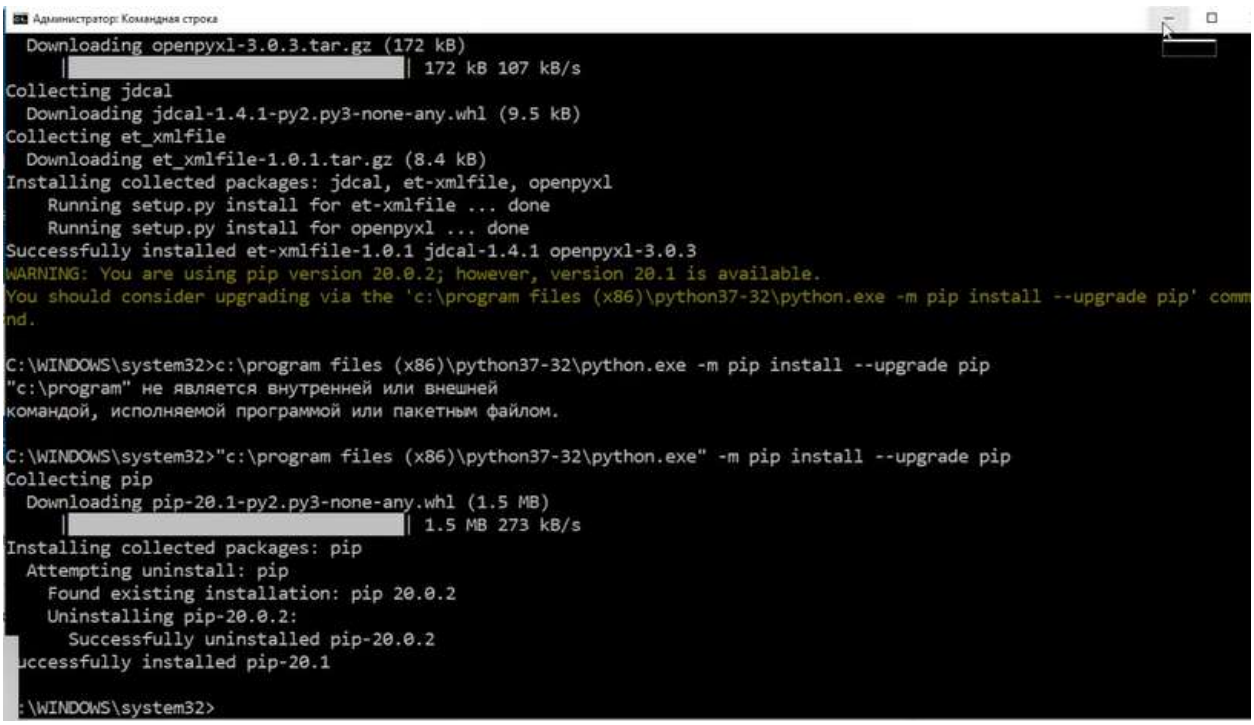


Рис.12.2. Продовження підключення бібліотеки openpyxl

12.2. Робота з бібліотекою openpyxl.

Вся документація про бібліотеку openpyxl.

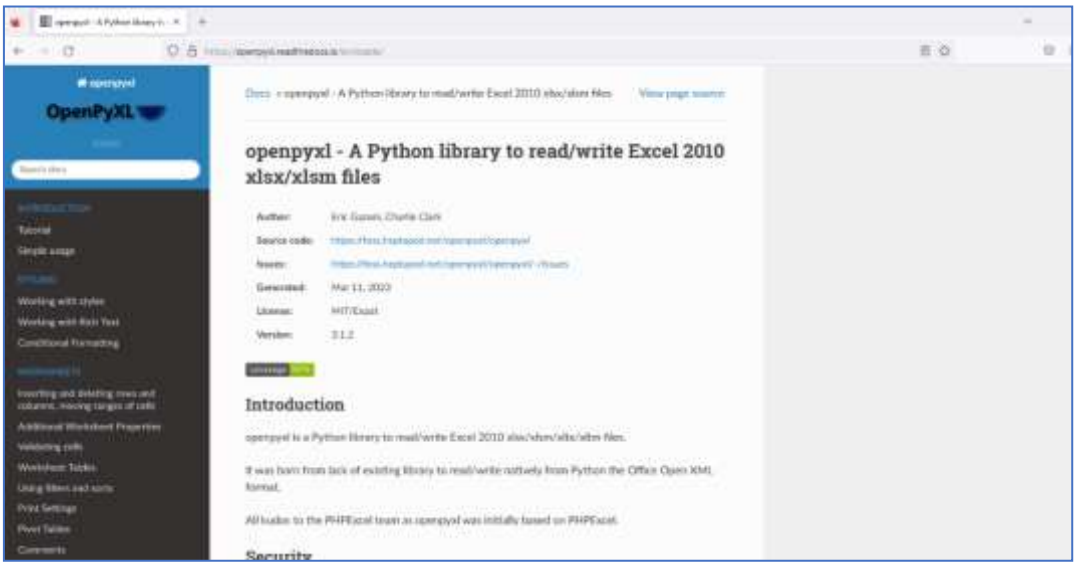


Рис.12.3. Документація бібліотеки openpyxl

Приклад 12.1 Робота з бібліотекою.

```
import  openpyxl
from openpyxl import Workbook
wb = Workbook()
# grab the active worksheet
ws = wb.active

# Data can be assigned directly to cells
ws['A1'] = 42

# Rows can also be appended
ws.append([1, 2, 3])

# Python types will automatically be converted
import datetime
ws['A2'] = datetime.datetime.now()
```

```
# Save the file
wb.save("sample.xlsx")
```

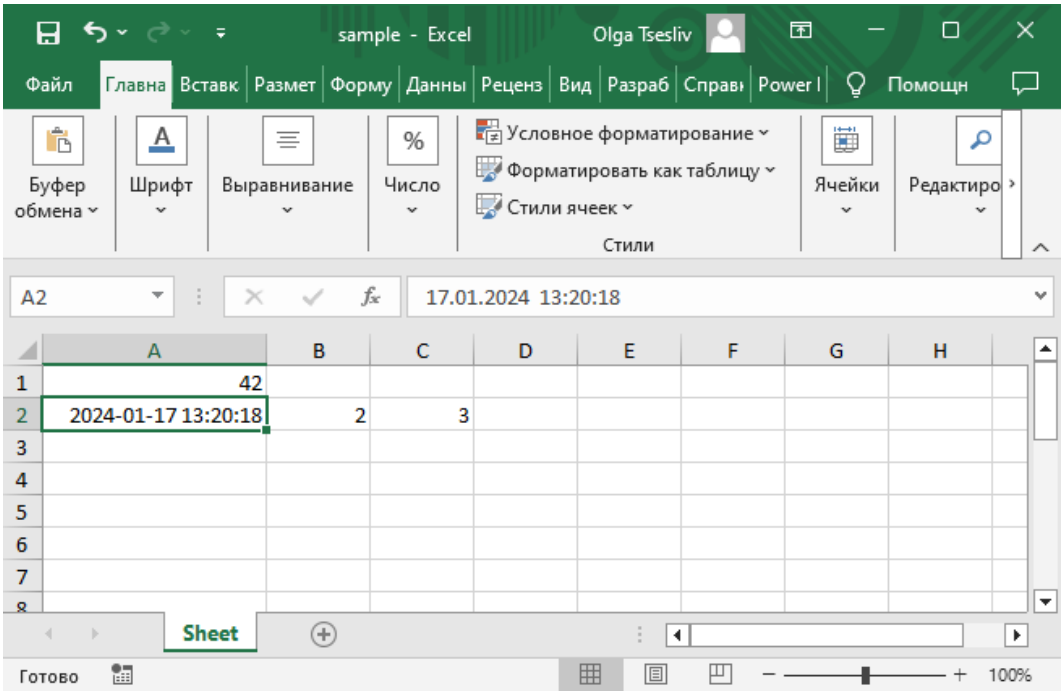
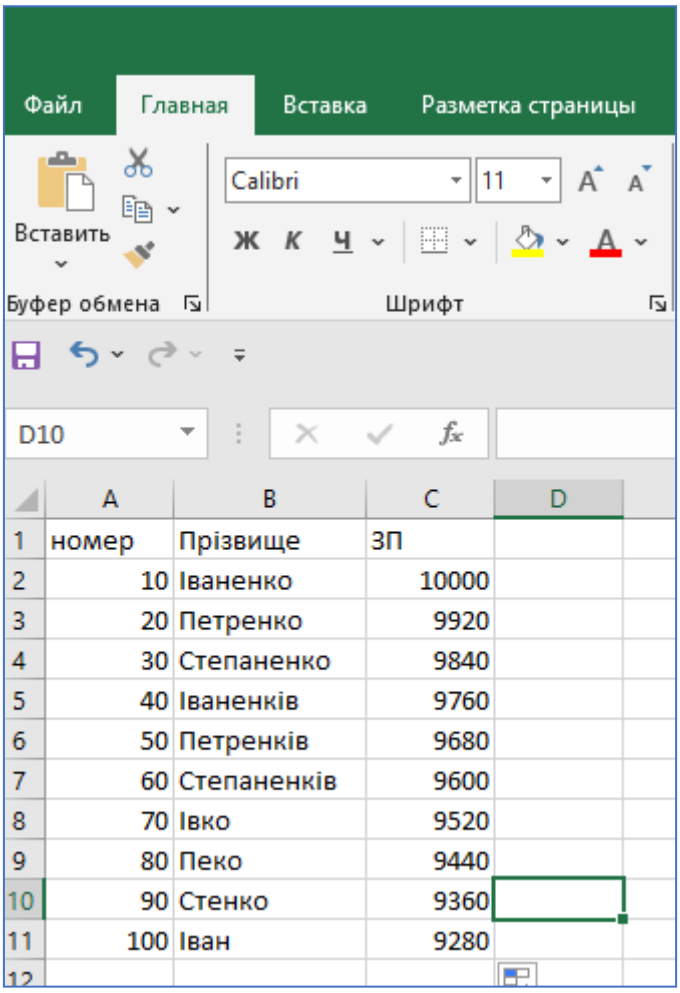


Рис.12.4. Результат виконання прикладу 12.1.

Створюємо лист Excel.



	A	B	C	D
1	номер	Прізвище	ЗП	
2	10	Іваненко	10000	
3	20	Петренко	9920	
4	30	Степаненко	9840	
5	40	Іваненків	9760	
6	50	Петренків	9680	
7	60	Степаненків	9600	
8	70	Івко	9520	
9	80	Пеко	9440	
10	90	Стенко	9360	
11	100	Іван	9280	
12				

Рис.12.5. Файл Excel для прикладу 12.2.

Приклад 12.3 Зчитування даних з файлу Excel.

```
import  openpyxl
from openpyxl import Workbook
wb = openpyxl.reader.excel.load_workbook(filename='sample.xlsx')
print(wb.sheetnames)
```

Результат

['Sheet']

```
import  openpyxl
from openpyxl import Workbook
wb = openpyxl.reader.excel.load_workbook(filename='sample1.xlsx')
print(wb.sheetnames)
```

['Лист1']

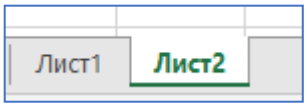


Рис.12.6. Результат виконання прикладу 12.3.

При запуску програми отримаємо

['Лист1', 'Лист2']

Приклад 12.4 Допишемо код до прикладу 12.3.

```
import  openpyxl
from openpyxl import Workbook
wb = openpyxl.reader.excel.load_workbook(filename='sample1.xlsx')
print(wb.sheetnames)
#активізуємо лист
wb.active=0
sheet=wb.active
print(sheet['A1'].value)
```

результат

['Лист1', 'Лист2']

Номер

Приклад 12.4 Зчитаємо дані з файлу.

```
import  openpyxl
from openpyxl import Workbook
wb = openpyxl.reader.excel.load_workbook(filename='sample1.xlsx')
#print(wb.sheetnames)
#активізуємо лист
wb.active=0
sheet=wb.active
for i in range(1,12):
    print(sheet['A'+str(i)].value,sheet['B'+str(i)].value,sheet['C'+str(i)].value)
```

результат

номер Прізвище ЗП

10 Іваненко 10000

20 Петренко =C2-80

30 Степаненко =C3-80

40 Іваненків =C4-80

50 Петренків =C5-80

60 Степаненків =C6-80

70 Івко =C7-80

80 Пеко =C8-80

90 Стенко =C9-80

100 Іван =C10-80

Як бачимо, замість значення прибутку отримали формулу. Щоб отримати значення записуємо наступний код.

Приклад 12.5 Отримання значення прибутку.

```
import  openpyxl
from openpyxl import Workbook
wb = openpyxl.reader.excel.load_workbook(filename='sample1.xlsx', data_only=True)
#print(wb.sheetnames)
#активізуємо лист
wb.active=0
sheet=wb.active
for i in range(1,12):
   print(sheet['A'+str(i)].value,sheet['B'+str(i)].value,sheet['C'+str(i)].value)
```

результат

номер Прізвище ЗП

10 Іваненко 10000

20 Петренко 9920

30 Степаненко 9840

40 Іваненків 9760

50 Петренків 9680

60 Степаненків 9600

70 Івко 9520

80 Пеко 9440

90 Стенко 9360

100 Іван 9280

12.3. Робота з файлами формата CSV

Формат CSV є найбільш часто використовуваним форматом імпорту та експорту для баз даних і електронних таблиць.. Розглянемо робочий приклад, що показує, як читати і записувати дані у файл CSV на Python.

Що таке файл CSV?

Файл CSV (значення, розділені комами) дозволяє зберігати дані в табличній структурі з розширенням .csv. CSV-файли широко використовуються в додатках електронної комерції, оскільки їх дуже легко обробляти. Деякі з областей, де вони були використані, включають в себе:

імпорт і експорт даних клієнтів

імпорт та експорт продукції

експорт замовлень

експорт аналітичних звітів з електронної комерції

12.4.Модулі для читання і запису даних

Модуль CSV має кілька функцій і класів, доступних для читання і запису CSV, і вони включають в себе:

функція csv.reader

функція csv.writer

клас csv.Dictwriter

клас csv.DictReader

Модуль csv.reader приймає такі параметри:

csvfile: зазвичай це об'єкт, який підтримує протокол ітератора і зазвичай повертає рядок щоразу, коли викликається його метод __next__().

dialect='excel': необов'язковий параметр, який використовується для визначення набору параметрів, специфічних для певного діалекту CSV.

fmtparams: необов'язковий параметр, який можна використовувати для перевизначення наявних параметрів форматування.

Ось приклад того, як використовувати модуль csv.reader.

Приклад 12.6. Отримання значення прибутку.

```
import csv
with open("sample26.csv", newline='') as csvfile:
   reader=csv.DictReader(csvfile, delimiter=";")
   for row in reader:
      print( row['lastname'], row['S'])
```

результат

```
Ivahenko 10000
Petrenko 9920
Stif 9840
Anna 9760
Smit 9680
Sur 9600
Ivko 9520
Pero 9440
Shur 9360
dJan 9280
```

Приклад 12.6 Створення простої електронної таблиці та гістограми.

```
from openpyxl import Workbook
wb = Workbook()
ws = wb.active
```

```
        treeData = [["Type", "Leaf Color", "Height"], ["Maple", "Red", 549],
["Oak", "Green", 783], ["Pine", "Green", 1204]]
        for row in treeData:
         ws.append(row)
         from openpyxl.styles import Font

        ft = Font(bold=True)
        for row in ws["A1:C1"]:
            for cell in row:
               cell.font = ft
        from openpyxl.chart import BarChart, Series, Reference
        chart = BarChart()
        chart.type = "col"
        chart.title = "Tree Height"
        chart.y_axis.title = 'Height (cm)'
        chart.x_axis.title = 'Tree Type'
        chart.legend = None
        data = Reference(ws, min_col=3, min_row=2, max_row=4, max_col=3)
        categories = Reference(ws, min_col=1, min_row=2, max_row=4,
max_col=1)

        chart.add_data(data)
        chart.set_categories(categories)
        ws.add_chart(chart, "E1")
        wb.save("TreeData.xlsx")
```

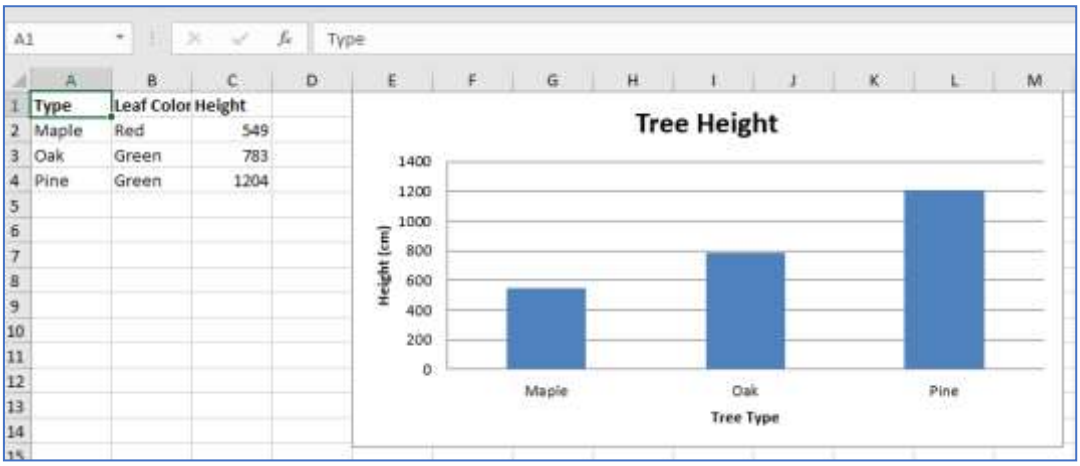


Рис.12.7. Результат виконання прикладу 12.6.

Контрольні запитання і завдання

1. Ввести два масиви знайти добуток.
2. Створити файл в Excel(Прізвище, адреса, телефон) зчитати
3. Створити файл з розширенням .csv(Прізвище, адреса, телефон) Зчитати дані.
4. Ввести два масиви знайти суму
5. Створити файл в Excel(Прізвище, посада, ЗП) зчитати
6. Створити файл з розширенням .csv (Прізвище, посада, ЗП) зчитати
7. Ввести два масиви знайти остачу від ділення
8. Створити файл в Excel(Прізвище, авто, ноутбук) зчитати
9. Створити файл з розширенням .csv (Прізвище, авто, ноутбук) зчитати

Список рекомендованої літератури

1. Васильєв О. Програмування мовою Pythonю. Навч.посіб.Львів.: Вид-во: Навчальна книга Богдан, 2019. 504 с.
2. Маттес Е. Пришвидшений курс Python. Практичний, проєктно-орієнтований вступ до програмування.Львів: Вид-во Старого Лева, 2019.600с.
3. Python. Довідник програміста. Марк Лутц. Львів: Видавництво Науковий світ, 2023. 294с.

4. Цеслів, О. В. Основи програмування та веб-дизайн для студентів економічних спеціальностей: навчальний посібник. КПІ ім. Ігоря Сікорського. Київ : КПІ ім. Ігоря Сікорського, 2020. 150 с. https://ela.kpi.ua/handle/123456789/40499.
5. Цеслів О. В., Коломієць А. С. Технологія проектування та адміністрування баз даних і сховищ даних : навч. посіб. для студ. екон. спец. Київ, КПІ ім. Ігоря Сікорського, Вид-во "Політехніка", 2017. 284 с. https://docplayer.net/92060328-Nacionalniy-tehnichniy-universitet-ukrayini-kiyivskiy-politehnichniy-institut-naukovo-tehnichna-biblioteka-im-g-i-denisenka.html.
6. Fundamentals of Web Programming. Practical Tutorial / L.Oleshchenko, Igor Sikorsky Kyiv Polytechnic Institute, 2021, 138 с. https://ela.kpi.ua/handle/123456789/42208.

Предметний покажчик .

Навчальне видання

Цеслів Ольга Володимирівна

Програмування для аналітичних досліджень

Навчальний посібник