Міністерство освіти і науки України
Міжнародний гуманітарний університет
Кафедра комп’ютерної інженерії та інноваційних технологій

В.В. Педяш

СКРИПТОВІ МОВИ ПРОГРАМУВАННЯ

Методичні вказівки до практичних робіт
для здобувачів вищої освіти,
які навчаються за спеціальностями
галузі «Інформаційні технології»

Одеса 2025

Затверджено Вченою Радою Міжнародного гуманітарного університету.
Протокол № 5 від 20 січня 2025 р.

Педяш В.В. Скриптові мови програмування. Методичні вказівки до виконання практичних робіт для здобувачів вищої освіти, які навчаються за спеціальностями галузі «Інформаційні технології» [Електронне видання]. Кафедра комп'ютерної інженерії та інноваційних технологій Міжнародного гуманітарного університету. Одеса, 2025. 46 с.

Дисципліна «Скриптові мови програмування» спрямована на формування у здобувачів вищої освіти спеціальності Інженерія програмного забезпечення компетентностей, пов'язаних із розробкою програмного забезпечення із використанням сучасних скриптових мов програмування. Вона орієнтована на формування здатності розробляти алгоритми, створювати програмні модулі та програмні системи для розв'язання прикладних задач різної складності, застосовувати стандартні та сторонні бібліотеки для обробки даних, автоматизації процесів та інтеграції програмних компонентів.

ЗМІСТ

Практична робота 1. Реалізація базових алгоритмів

Мета роботи – ознайомитися з основними типами даних у скриптових мовах програмування, навчитися використовувати змінні для збереження даних та реалізовувати базові алгоритми обробки числової і текстової інформації.

Ключові положення

Змінні є базовим елементом будь-якої програми і призначені для збереження даних, які можуть змінюватися під час виконання програми. У скриптових мовах програмування змінні зазвичай не потребують явного оголошення типу, оскільки тип визначається автоматично на основі присвоєного значення. Такий підхід називається динамічною типізацією і спрощує процес розробки, але вимагає уважності під час роботи з різними типами даних.

Типи даних визначають, які значення може містити змінна і які операції можна над ними виконувати. Найбільш поширеними є цілочисельні типи, дійсні числа, рядки та логічні значення. Кожен із цих типів має свої особливості обробки і використання. Наприклад, числові типи дозволяють виконувати арифметичні операції, тоді як рядки призначені для роботи з текстовою інформацією.

Операції над змінними є основою реалізації алгоритмів. До них належать арифметичні операції, операції порівняння та логічні операції. Вони дозволяють виконувати обчислення, перевіряти умови та керувати ходом виконання програми.

Важливим є правильне використання операторів присвоєння. Під час виконання програми змінна може змінювати своє значення багаторазово, що дозволяє реалізувати алгоритмічні процеси, зокрема накопичення результатів, обчислення за формулами та обробку введених даних.

Скриптові мови програмування широко використовуються для швидкої розробки прикладних програм, автоматизації задач та обробки даних. Використання змінних і типів даних є необхідною умовою реалізації будь-якого алгоритму, незалежно від його складності.

Таким чином, змінні та типи даних є фундаментом програмування, що забезпечує можливість збереження, обробки та аналізу інформації у програмних системах.

Практичне завдання

1. Оголосити змінні різних типів даних: цілочисельні, дійсні, рядкові та логічні.
2. Реалізувати введення даних з клавіатури та збереження їх у змінних.
3. Виконати арифметичні операції над числовими змінними.
4. Реалізувати обчислення значення математичного виразу.
5. Виконати операції порівняння між змінними та вивести результати.
6. Реалізувати простий алгоритм обробки текстових даних (наприклад, об’єднання рядків або визначення довжини рядка).
7. Створити програму, що використовує логічні змінні для перевірки умов.
8. Проаналізувати результати виконання програми для різних вхідних даних.

Методичні вказівки до виконання практичного завдання

Перед початком виконання роботи необхідно ознайомитися з основними типами даних, що використовуються у вибраній скриптовій мові програмування. Особливу увагу слід приділити особливостям динамічної типізації та правилам роботи зі змінними.

На першому етапі необхідно створити програму, у якій оголошуються змінні різних типів. Кожній змінній слід присвоїти початкове значення та вивести його у консоль для перевірки правильності роботи.

На наступному етапі необхідно реалізувати введення даних користувачем. Введені значення слід зберігати у змінних відповідного типу. У разі потреби необхідно виконати перетворення типів даних, наприклад перетворення рядка у числовий тип для виконання обчислень.

Далі необхідно реалізувати обчислення математичного виразу. Вираз повинен містити декілька операцій, що дозволить продемонструвати порядок їх виконання та роботу з числовими типами даних. Результат обчислення слід вивести у зручному для сприйняття вигляді.

Після цього необхідно реалізувати операції порівняння. Слід перевірити рівність, нерівність, більшість або меншість значень змінних. Результати таких операцій мають логічний тип і можуть бути використані для подальшої обробки.

На наступному етапі необхідно реалізувати обробку рядкових даних. Наприклад, можна об’єднати декілька рядків, змінити їх регістр або визначити довжину рядка. Це дозволить продемонструвати можливості роботи з текстовою інформацією.

Далі необхідно використати логічні змінні для реалізації простих умов. Наприклад, можна перевірити, чи належить число до певного діапазону або чи виконується задана умова.

У процесі виконання роботи необхідно протестувати програму на різних вхідних даних. Це дозволить перевірити коректність роботи алгоритмів і виявити можливі помилки.

Контрольні питання

1. Що таке змінна у програмуванні?
2. Які типи даних використовуються у скриптових мовах програмування?
3. У чому полягає динамічна типізація?
4. Які операції можна виконувати над числовими типами даних?
5. Які особливості роботи з рядковими даними?
6. Що таке логічний тип даних?
7. Яке призначення операцій порівняння?
8. Навіщо виконують перетворення типів даних?
9. Які помилки можуть виникати при роботі зі змінними?
10. Яке значення мають змінні та типи даних у реалізації алгоритмів?

Зміст звіту

– назва практичної роботи;
– мета виконання;
– опис типів даних і змінних;
– опис реалізованих алгоритмів;

– приклади виконання програми;
– результати тестування;
– висновки.

Практична робота 2. Розробка програм із використанням умовних операторів

Мета роботи – ознайомитися з принципами розгалуження алгоритмів, навчитися використовувати умовні оператори для вибору дій залежно від заданих умов та реалізовувати програми з альтернативними варіантами виконання.

Ключові положення

Умовні оператори є основним засобом реалізації розгалуження в алгоритмах. Вони дають змогу програмі вибирати один із кількох можливих шляхів виконання залежно від істинності або хибності певної умови. Завдяки цьому програма може реагувати на введені дані, аналізувати ситуації та приймати рішення відповідно до заданої логіки.

У скриптових мовах програмування умовні оператори зазвичай реалізуються у вигляді конструкцій if, elif або else. Основна частина такої конструкції містить логічний вираз, результат якого визначає, чи буде виконано певний блок команд. Якщо умова є істинною, виконується одна послідовність команд, а якщо ні – інша.

Логічні вирази у складі умовних операторів будуються на основі операцій порівняння та логічних операцій. За допомогою операторів порівняння можна визначити, чи є два значення рівними, чи одне з них більше або менше за інше. Логічні оператори дозволяють поєднувати декілька умов в один складений вираз.

Умовні оператори широко використовуються у прикладному програмуванні. Вони є основою перевірки коректності введених даних, вибору режиму роботи програми, реалізації елементів керування та обробки різних ситуацій, що виникають у процесі виконання програми.

Важливим є правильне формулювання умов. Некоректно побудований логічний вираз може призвести до неправильного вибору гілки виконання програми. Тому під час розробки алгоритмів необхідно уважно аналізувати умови, їх порядок і можливі варіанти значень вхідних даних.

У багатьох задачах використовується вкладене розгалуження, коли в одній гілці умовного оператора міститься інший умовний оператор. Такий підхід дозволяє реалізувати складніші алгоритми прийняття рішень, однак вимагає чіткої структури коду і зрозумілої логіки.

Таким чином, умовні оператори є необхідним засобом побудови алгоритмів, у яких результат виконання залежить від певних умов, а їх використання забезпечує гнучкість і адаптивність програмного забезпечення.

Практичне завдання

1. Реалізувати програму, що перевіряє, чи є введене число додатним, від’ємним або нульовим.
2. Створити програму для визначення більшого з двох або трьох чисел.
3. Реалізувати перевірку парності або непарності введеного числа.

4. Розробити програму для визначення належності значення до заданого інтервалу.

5. Реалізувати програму, що за введеним балом визначає оцінку за заданою шкалою.

6. Використати складені логічні вирази з декількома умовами.

7. Реалізувати приклад вкладених умовних операторів.

8. Проаналізувати результати виконання програми для різних варіантів вхідних даних.

Методичні вказівки до виконання практичного завдання

Перед початком виконання роботи необхідно ознайомитися з поняттям розгалуження алгоритму, структурою умовних операторів та правилами побудови логічних виразів. Особливу увагу слід приділити використанню операторів порівняння та логічних операторів, які застосовуються для формування умов.

На першому етапі необхідно реалізувати найпростішу програму з одним умовним оператором. Наприклад, можна перевірити, чи є число додатним. Після цього доцільно ускладнити задачу і додати альтернативну гілку для випадку, коли число не задовольняє умову.

Далі необхідно реалізувати повне розгалуження з кількома варіантами. Наприклад, програма може визначати, чи є число додатним, від'ємним або нульовим. У цьому випадку слід застосувати послідовну перевірку умов та виведення відповідного результату.

На наступному етапі необхідно реалізувати задачі, у яких використовуються складені умови. Для цього слід поєднати декілька простих логічних виразів за допомогою логічних операторів. Наприклад, можна перевірити, чи належить число певному діапазону значень.

Після цього необхідно реалізувати вкладені умовні оператори. Один із можливих прикладів полягає у визначенні оцінки за декількома критеріями або у виборі дій залежно від декількох характеристик введених даних. Під час реалізації потрібно звернути увагу на правильне структурування коду, щоб уникнути логічних помилок.

У процесі виконання роботи необхідно протестувати програму для різних наборів вхідних даних. Доцільно перевірити граничні випадки, наприклад нульові значення, однакові числа або значення на межах інтервалів. Це дозволить оцінити коректність реалізації умовних операторів.

Результати виконання програми слід проаналізувати з погляду правильності вибору гілок розгалуження. Особливу увагу слід приділити тому, як змінюється поведінка програми залежно від вхідних параметрів і наскільки логічно побудовано алгоритм прийняття рішень.

Контрольні питання

1. Що таке розгалуження алгоритму?

2. Яке призначення умовного оператора?

3. Які конструкції використовуються для реалізації умовних операторів у скриптових мовах програмування?

4. Що таке логічний вираз?

5. Які оператори порівняння використовуються в умовах?

6. Для чого застосовуються логічні оператори?

7. У чому полягає різниця між простим і повним розгалуженням?

8. Що таке вкладений умовний оператор?

9. Чому важливо перевіряти граничні значення в умовах?

10. Які помилки можуть виникати під час побудови умовних виразів?

Зміст звіту

– назва практичної роботи;
– мета виконання;
– опис принципу роботи умовних операторів;
– опис реалізованих програм;
– приклади вхідних даних і отриманих результатів;
– результати тестування;
– висновки.

Практична робота 3. Організація циклічних обчислень

Мета роботи – ознайомитися з принципами реалізації циклічних алгоритмів, навчитися використовувати оператори циклу для багаторазового виконання дій та реалізовувати обчислювальні процеси з повторенням.

Ключові положення

Циклічні конструкції є основним засобом реалізації повторюваних обчислень у програмуванні. Вони дають змогу виконувати одну і ту ж послідовність дій багаторазово без дублювання коду. Це дозволяє ефективно реалізовувати алгоритми обробки даних, накопичення результатів і перебору значень.

У скриптових мовах програмування найчастіше використовуються два основні типи циклів: цикл із лічильником та цикл із умовою. Цикл із лічильником використовується у випадках, коли кількість повторень відома заздалегідь. Цикл із умовою застосовується тоді, коли кількість повторень залежить від виконання певної умови.

Ключовими елементами циклу є початкове значення, умова виконання та зміна керуючої змінної. Порушення будь-якого з цих елементів може призвести до некоректної роботи програми, зокрема до виникнення нескінченного циклу.

Важливим є використання змінних для накопичення результатів обчислень. Під час виконання циклу значення змінної може змінюватися на кожній ітерації, що дозволяє реалізувати алгоритми сумування, множення або пошуку значень.

У процесі роботи циклу часто застосовуються умовні оператори, які дозволяють змінювати поведінку алгоритму залежно від певних умов. Це дає змогу реалізувати більш складні алгоритмічні структури, зокрема фільтрацію даних або обробку окремих випадків.

Цикли широко використовуються під час обробки масивів, рядків і інших структур даних. Вони дозволяють послідовно обробляти елементи структури та виконувати необхідні операції над кожним із них.

Таким чином, циклічні конструкції є необхідним інструментом програмування, що забезпечує ефективну реалізацію повторюваних обчислень і обробку даних.

Практичне завдання

1. Реалізувати цикл із лічильником для виведення чисел у заданому діапазоні.
2. Реалізувати цикл із умовою для виконання обчислень до досягнення заданої умови.

3. Обчислити суму перших N натуральних чисел із використанням циклу.

4. Реалізувати обчислення факторіала числа.

5. Створити програму для знаходження максимального та мінімального значення серед введених чисел.

6. Реалізувати обробку рядка з використанням циклу (наприклад, підрахунок кількості символів або певних елементів).

7. Використати умовні оператори всередині циклу для реалізації складнішої логіки.

8. Проаналізувати результати виконання програм для різних вхідних даних.

Методичні вказівки до виконання практичного завдання

Перед початком виконання роботи необхідно ознайомитися з основними типами циклів, що використовуються у скриптових мовах програмування, а також з правилами їх побудови. Особливу увагу слід приділити умовам завершення циклу та зміні керуючих змінних.

На першому етапі необхідно реалізувати простий цикл із лічильником. У цьому циклі слід задати початкове значення, кінцеве значення та крок зміни змінної. Результат виконання циклу необхідно вивести у консоль для перевірки правильності роботи.

На наступному етапі необхідно реалізувати цикл із умовою. У цьому випадку виконання циклу має тривати доти, доки виконується задана умова. Важливо забезпечити зміну значень змінних таким чином, щоб цикл завершився коректно.

Далі необхідно реалізувати алгоритми обчислення суми та факторіала. Для цього слід використати змінну-накопичувач, яка змінює своє значення на кожній ітерації циклу. Потрібно звернути увагу на правильну ініціалізацію цієї змінної перед початком циклу.

Після цього необхідно реалізувати обробку послідовності значень. Наприклад, можна організувати введення декількох чисел і визначити серед них максимальне та мінімальне значення. У цьому випадку слід правильно організувати порівняння значень у процесі виконання циклу.

На наступному етапі необхідно реалізувати обробку текстових даних із використанням циклу. Наприклад, можна перебрати символи рядка і виконати певні дії для кожного символу.

Під час реалізації задач необхідно використовувати умовні оператори всередині циклу. Це дозволить реалізувати складніші алгоритми, наприклад обробку лише тих елементів, які задовольняють певну умову.

У процесі виконання роботи необхідно протестувати програми на різних вхідних даних, включаючи граничні випадки. Це дозволить перевірити коректність реалізації циклів і уникнути логічних помилок.

Контрольні питання

1. Що таке цикл у програмуванні?
2. Які основні типи циклів використовуються у скриптових мовах програмування?
3. У чому полягає відмінність між циклом із лічильником і циклом із умовою?
4. Які основні елементи має цикл?
5. Що таке нескінченний цикл і чому він виникає?
6. Яке призначення змінної-накопичувача?

7. Як реалізується обробка послідовності даних за допомогою циклу?
8. Чому важливо правильно задавати умову завершення циклу?
9. Яку роль відіграють умовні оператори всередині циклу?
10. Які помилки можуть виникати при використанні циклів?

Зміст звіту

– назва практичної роботи;
– мета виконання;
– опис принципів роботи циклів;
– опис реалізованих алгоритмів;
– приклади виконання програм;
– результати тестування;
– висновки.

Практична робота 4. Обробка та перетворення даних у програмі

Мета роботи – ознайомитися з основними методами обробки та перетворення даних у скриптових мовах програмування, навчитися виконувати перетворення типів, обробляти текстову та числову інформацію та реалізовувати алгоритми аналізу даних.

Ключові положення

Обробка даних є однією з основних задач програмування, що полягає у зміні, аналізі та інтерпретації інформації з метою отримання корисних результатів. У скриптових мовах програмування обробка даних здійснюється за допомогою вбудованих функцій, операторів і методів роботи з різними типами даних.

Перетворення даних передбачає зміну типу або структури даних. Найпоширенішими є перетворення між рядковими та числовими типами. Такі операції необхідні, наприклад, при введенні даних користувачем, коли значення спочатку надходять у вигляді тексту, а потім використовуються для виконання обчислень.

Обробка текстових даних включає операції об’єднання рядків, розбиття рядка на частини, пошуку підрядків, заміни символів та зміни регістру. Ці операції широко застосовуються під час роботи з текстовою інформацією, файлами та мережевими повідомленнями.

Обробка числових даних передбачає виконання арифметичних операцій, округлення значень, обчислення статистичних показників та аналіз числових послідовностей. Важливим є правильний вибір типу даних, оскільки це впливає на точність і результат обчислень.

У процесі обробки даних часто використовуються структури даних, зокрема списки або масиви. Вони дозволяють зберігати множину значень і виконувати обробку кожного елемента з використанням циклів.

Важливим є забезпечення коректності даних. Перед виконанням обробки необхідно перевіряти правильність введених значень, щоб уникнути помилок під час виконання програми.

Таким чином, обробка та перетворення даних є невід’ємною частиною програмування, що забезпечує можливість роботи з інформацією різного типу та складності.

Практичне завдання

1. Реалізувати введення даних користувачем і виконати перетворення типів даних.
2. Створити програму для обробки числових даних (обчислення середнього значення, суми, максимуму та мінімуму).
3. Реалізувати обробку текстового рядка (визначення довжини, підрахунок символів, зміна регістру).
4. Виконати розбиття рядка на окремі елементи та їх подальшу обробку.
5. Реалізувати заміну символів або підрядків у тексті.
6. Використати структури даних для збереження набору значень і виконати їх обробку.
7. Реалізувати перевірку коректності введених даних.
8. Проаналізувати результати обробки для різних вхідних даних.

Методичні вказівки до виконання практичного завдання

Перед початком виконання роботи необхідно ознайомитися з основними методами роботи з даними у вибраній скриптовій мові програмування. Особливу увагу слід приділити операціям перетворення типів даних та обробці рядків.

На першому етапі необхідно реалізувати введення даних користувачем. Введені значення, як правило, мають рядковий тип, тому їх потрібно перетворити у відповідний тип для подальшої обробки. Наприклад, для виконання арифметичних операцій необхідно перетворити рядок у числовий тип.

Далі необхідно реалізувати обробку числових даних. Для цього слід використати змінні або структури даних для збереження значень і виконати необхідні обчислення, зокрема знаходження суми, середнього значення, максимального та мінімального значення.

На наступному етапі необхідно реалізувати обробку текстових даних. Слід виконати основні операції над рядками, зокрема визначення довжини рядка, зміну регістру символів, пошук і заміну підрядків.

Після цього необхідно реалізувати розбиття рядка на окремі елементи. Наприклад, можна розбити рядок за певним розділювачем і отримати список значень, які надалі можна обробляти у циклі.

Далі необхідно використати структури даних для збереження набору значень. Наприклад, можна створити список чисел або рядків і виконати обробку кожного елемента.

Особливу увагу слід приділити перевірці коректності введених даних. Необхідно передбачити ситуації, коли користувач вводить некоректні значення, і забезпечити обробку таких випадків.

У процесі виконання роботи необхідно протестувати програму на різних вхідних даних, щоб переконатися у правильності реалізації алгоритмів обробки та перетворення даних.

Контрольні питання

1. Що таке обробка даних у програмуванні?
2. Що таке перетворення типів даних?

3. Які основні операції виконуються над рядками?
4. Як здійснюється обробка числових даних?
5. Які структури даних використовуються для збереження наборів значень?
6. Для чого виконується перевірка коректності даних?
7. Як реалізується розбиття рядка на частини?
8. Які помилки можуть виникати під час перетворення типів даних?
9. Які методи використовуються для пошуку та заміни підрядків?
10. Яке значення має обробка даних у прикладному програмуванні?

Зміст звіту

– назва практичної роботи;
– мета виконання;
– опис методів обробки та перетворення даних;
– опис реалізованих програм;
– приклади вхідних і вихідних даних;
– результати тестування;
– висновки.

Практична робота 5. Побудова алгоритмів керування виконанням програм

Мета роботи – ознайомитися з принципами керування виконанням програм, навчитися комбінувати умовні оператори та цикли, використовувати оператори керування потоком виконання та реалізовувати складні алгоритмічні структури.

Ключові положення

Керування виконанням програм є важливою складовою програмування, що визначає порядок виконання команд і дозволяє змінювати цей порядок залежно від умов та стану програми. Основними засобами керування виконанням є умовні оператори, цикли та спеціальні оператори зміни потоку виконання.

Умовні оператори дозволяють вибирати різні варіанти виконання програми залежно від значень змінних або результатів обчислень. Цикли забезпечують багаторазове виконання певного фрагмента коду. Поєднання цих конструкцій дає змогу реалізовувати складні алгоритми, що адаптуються до різних ситуацій.

Оператори керування потоком виконання, такі як break, continue та return, дозволяють змінювати стандартний хід виконання програми. Оператор break використовується для дострокового завершення циклу, continue – для переходу до наступної ітерації циклу, а return – для завершення виконання функції з поверненням результату.

Важливим є правильне проєктування алгоритму перед його реалізацією. Алгоритм повинен бути логічно послідовним, враховувати всі можливі варіанти виконання та забезпечувати коректну обробку даних.

У складних алгоритмах часто використовуються вкладені конструкції, коли цикли містять умовні оператори або інші цикли. Це дозволяє реалізувати багаторівневу логіку обробки даних, але вимагає уважності для забезпечення зрозумілості та коректності коду.

Керування виконанням програм є основою розробки інтерактивних застосунків, систем обробки даних та автоматизованих процесів. Правильне використання відповідних конструкцій дозволяє створювати ефективні та гнучкі програмні рішення.

Таким чином, засоби керування виконанням програм забезпечують гнучкість алгоритмів і дають змогу реалізовувати складні логічні структури у програмному забезпеченні.

Практичне завдання

1. Реалізувати програму з використанням вкладених умовних операторів.
2. Створити програму з використанням вкладених циклів.
3. Реалізувати алгоритм із використанням операторів break та continue.
4. Розробити програму, що виконує обробку даних до виконання певної умови.
5. Реалізувати алгоритм із комбінуванням умовних операторів і циклів.
6. Створити програму з використанням функції та оператора return.
7. Реалізувати обробку послідовності даних із різними умовами завершення.
8. Проаналізувати поведінку програми при зміні вхідних даних.

Методичні вказівки до виконання практичного завдання

Перед початком виконання роботи необхідно ознайомитися з основними конструкціями керування виконанням програм, а також з правилами їх комбінування. Особливу увагу слід приділити використанню операторів break, continue та return.

На першому етапі необхідно реалізувати просту програму з використанням вкладених умовних операторів. Слід забезпечити правильну структуру умов і перевірити всі можливі варіанти їх виконання.

Далі необхідно реалізувати програму з використанням вкладених циклів. Наприклад, можна створити таблицю значень або виконати обробку двовимірної структури даних.

На наступному етапі необхідно використати оператори break та continue. Для цього слід реалізувати цикл, у якому за певних умов виконання переривається або переходить до наступної ітерації.

Після цього необхідно реалізувати алгоритм, у якому виконання програми залежить від певної умови. Наприклад, можна організувати введення даних до моменту отримання коректного значення.

Далі необхідно поєднати умовні оператори та цикли у межах одного алгоритму. Це дозволить реалізувати складнішу логіку обробки даних.

На наступному етапі необхідно реалізувати функцію, яка виконує певні обчислення і повертає результат. Використання оператора return дозволяє організувати завершення виконання функції та передавання результату виклику.

У процесі виконання роботи необхідно протестувати програму для різних вхідних даних, включаючи граничні випадки. Це дозволить перевірити коректність роботи алгоритмів керування виконанням.

Контрольні питання

1. Що таке керування виконанням програми?

2. Які основні конструкції використовуються для керування виконанням?
3. Яке призначення операторів break і continue?
4. У яких випадках використовується оператор return?
5. Що таке вкладені конструкції?
6. Як поєднуються умовні оператори та цикли?
7. Чому важливо правильно проєктувати алгоритм?
8. Які помилки можуть виникати при використанні вкладених циклів?
9. Як забезпечити завершення циклу за умовою?
10. Яке значення має керування виконанням у програмуванні?

Зміст звіту

– назва практичної роботи;
– мета виконання;
– опис засобів керування виконанням програм;
– опис реалізованих алгоритмів;
– приклади виконання програм;
– результати тестування;
– висновки.

Практична робота 6. Створення та використання функцій у програмі

Мета роботи – ознайомитися з принципами декомпозиції програм на функції, навчитися створювати та використовувати функції з параметрами і значеннями, що повертаються, та реалізовувати модульну структуру програм.

Ключові положення

Функції є одним із основних засобів структуризації програмного коду. Вони дозволяють виділити окремі логічні частини програми у самостійні блоки, що мають чітко визначене призначення. Такий підхід забезпечує зручність розробки, повторне використання коду та підвищення його зрозумілості.

Функція має ім’я, список параметрів і тіло, у якому визначено послідовність дій. Параметри функції дозволяють передавати вхідні дані, а результат виконання функції може бути повернений за допомогою оператора return. У скриптових мовах програмування функції можуть працювати з даними різних типів без необхідності явного визначення типів параметрів.

Використання функцій дозволяє реалізувати принцип декомпозиції, коли складна задача розбивається на простіші підзадачі. Кожна функція виконує окрему частину роботи, що спрощує розробку та тестування програмного забезпечення.

Функції можуть бути як вбудованими, так і користувацькими. Вбудовані функції надають базові можливості роботи з даними, тоді як користувацькі функції створюються програмістом для реалізації конкретних задач.

Важливим є правильне використання області видимості змінних. Змінні, оголошені всередині функції, є локальними і доступні лише у межах цієї функції. Це дозволяє уникнути конфліктів і небажаних змін даних.

Функції можуть викликатися багаторазово з різними аргументами, що дозволяє використовувати один і той самий код для обробки різних даних. Це підвищує ефективність розробки та зменшує обсяг коду.

Таким чином, використання функцій забезпечує модульність програм, спрощує їх підтримку та дозволяє реалізовувати складні алгоритми у вигляді сукупності простіших операцій.

Практичне завдання

1. Створити просту функцію без параметрів і без повернення значення.
2. Реалізувати функцію з параметрами для виконання обчислень.
3. Створити функцію, що повертає результат обчислення.
4. Реалізувати функцію для обробки текстових даних.
5. Використати функції для розбиття програми на логічні частини.
6. Реалізувати виклик функції з різними аргументами.
7. Дослідити область видимості змінних у функціях.
8. Проаналізувати результати виконання програми з використанням функцій.

Методичні вказівки до виконання практичного завдання

Перед початком виконання роботи необхідно ознайомитися з поняттям функції, її структурою та правилами використання у скриптових мовах програмування. Особливу увагу слід приділити передачі параметрів і поверненню результатів.

На першому етапі необхідно створити просту функцію, яка виконує певну дію, наприклад виведення повідомлення. Це дозволить зрозуміти базову структуру функції та спосіб її виклику.

Далі необхідно реалізувати функцію з параметрами. Параметри повинні передаватися у функцію під час її виклику і використовуватися для виконання обчислень або інших операцій.

На наступному етапі необхідно створити функцію, що повертає результат. Для цього слід використати оператор return. Результат виконання функції можна зберегти у змінній і використати у подальших обчисленнях.

Після цього необхідно реалізувати функцію для обробки текстових даних. Наприклад, функція може виконувати зміну регістру, підрахунок символів або об’єднання рядків.

Далі необхідно використати декілька функцій у межах однієї програми. Це дозволить реалізувати принцип декомпозиції та структурувати програмний код.

Особливу увагу слід приділити області видимості змінних. Необхідно перевірити, як змінні поводяться всередині функції і поза її межами.

У процесі виконання роботи необхідно протестувати програму з різними наборами вхідних даних, щоб переконатися у правильності роботи функцій.

Контрольні питання

1. Що таке функція у програмуванні?
2. Які основні елементи має функція?
3. Яке призначення параметрів функції?

4. Для чого використовується оператор return?
5. У чому полягає принцип декомпозиції?
6. Яка різниця між вбудованими та користувацькими функціями?
7. Що таке локальна змінна?
8. Як здійснюється виклик функції?
9. Які переваги дає використання функцій?
10. Які помилки можуть виникати при використанні функцій?

Зміст звіту

– назва практичної роботи;
– мета виконання;
– опис принципів роботи функцій;
– опис реалізованих функцій;
– приклади виконання програми;
– результати тестування;
– висновки.

Практична робота 7. Реалізація рекурсивних алгоритмів

Мета роботи – ознайомитися з принципами рекурсії, навчитися реалізовувати рекурсивні функції та порівнювати їх ефективність із ітераційними алгоритмами.

Ключові положення

Рекурсія є методом організації алгоритмів, при якому функція викликає сама себе для розв'язання підзадачі меншого розміру. Такий підхід дозволяє природно реалізовувати задачі, що мають ієрархічну або повторювану структуру.

Основними елементами рекурсивної функції є базовий випадок і рекурсивний виклик. Базовий випадок визначає умову завершення рекурсії і запобігає нескінченному виклику функції. Рекурсивний виклик зменшує розмір задачі і поступово наближає її до базового випадку.

Рекурсивні алгоритми широко застосовуються для обробки структур даних, таких як дерева та графи, а також для реалізації класичних задач, наприклад обчислення факторіала або чисел Фібоначчі.

Під час виконання рекурсивної функції кожен виклик створює новий запис у стеку викликів. Це дозволяє зберігати проміжні значення, але також може призводити до перевитрати пам'яті у випадку глибокої рекурсії.

Важливим є правильне визначення базового випадку і забезпечення того, щоб кожен рекурсивний виклик наближався до нього. У протилежному випадку може виникнути нескінченна рекурсія, що призводить до аварійного завершення програми.

У багатьох випадках рекурсивні алгоритми можна замінити ітераційними. Ітераційні алгоритми часто є більш ефективними з точки зору використання пам'яті, однак рекурсія дозволяє отримати більш компактний і зрозумілий код.

Таким чином, рекурсія є важливим інструментом програмування, що дозволяє ефективно розв'язувати задачі з повторюваною структурою.

Практичне завдання

1. Реалізувати рекурсивну функцію для обчислення факторіала числа.
2. Реалізувати рекурсивний алгоритм обчислення чисел Фібоначчі.
3. Створити рекурсивну функцію для обчислення суми перших N натуральних чисел.
4. Реалізувати рекурсивний обхід структури даних (наприклад, вкладених списків).
5. Порівняти рекурсивну та ітераційну реалізацію одного алгоритму.
6. Проаналізувати глибину рекурсії для різних значень вхідних даних.
7. Реалізувати обробку помилкових випадків (наприклад, від'ємні значення).
8. Оцінити ефективність рекурсивних алгоритмів.

Методичні вказівки до виконання практичного завдання

Перед початком виконання роботи необхідно ознайомитися з поняттям рекурсії, її структурою та особливостями використання. Особливу увагу слід приділити визначенню базового випадку і організації рекурсивного виклику.

На першому етапі необхідно реалізувати просту рекурсивну функцію, наприклад обчислення факторіала. Слід визначити базовий випадок, при якому функція повертає результат без подальших викликів, та рекурсивний випадок, у якому функція викликає сама себе.

Далі необхідно реалізувати обчислення чисел Фібоначчі. У цьому випадку функція викликає сама себе двічі, що дозволяє продемонструвати особливості рекурсивних алгоритмів і їх вплив на продуктивність.

На наступному етапі необхідно реалізувати рекурсивну обробку даних. Наприклад, можна обробити вкладену структуру, у якій елементи можуть містити інші елементи такого ж типу.

Після цього необхідно порівняти рекурсивну та ітераційну реалізації одного алгоритму. Слід звернути увагу на відмінності у структурі коду, швидкодії та використанні пам'яті.

Особливу увагу слід приділити аналізу глибини рекурсії. Для великих значень вхідних даних кількість викликів функції може значно зростати, що впливає на ефективність програми.

У процесі виконання роботи необхідно передбачити обробку некоректних даних. Наприклад, для факторіала від'ємного числа слід вивести повідомлення про помилку.

Результати виконання програм необхідно проаналізувати з точки зору правильності та ефективності.

Контрольні питання

1. Що таке рекурсія?
2. Які основні елементи рекурсивної функції?
3. Що таке базовий випадок?
4. Чому важливо правильно визначати умову завершення рекурсії?
5. Які задачі доцільно розв'язувати за допомогою рекурсії?
6. Що таке стек викликів?

7. Які переваги та недоліки рекурсії?
8. У чому різниця між рекурсивними та ітераційними алгоритмами?
9. Які помилки можуть виникати при використанні рекурсії?
10. Як оцінити ефективність рекурсивного алгоритму?

Зміст звіту

– назва практичної роботи;
– мета виконання;
– опис принципу рекурсії;
– опис реалізованих алгоритмів;
– порівняння рекурсивної та ітераційної реалізації;
– результати тестування;
– висновки.

Практична робота 8. Організація програмного коду у вигляді модулів

Мета роботи – ознайомитися з принципами модульної організації програмного коду, навчитися створювати та використовувати модулі, а також реалізовувати повторне використання програмних компонентів.

Ключові положення

Модульна організація програмного коду є важливим підходом до розробки програмного забезпечення, що передбачає поділ програми на окремі незалежні частини. Кожна така частина, або модуль, виконує певну функцію і може бути використана повторно у різних частинах програми або в інших проєктах.

Модулі дозволяють структурувати код, зменшити його складність та підвищити рівень його повторного використання. Завдяки цьому програмний код стає більш зрозумілим, зручним для супроводження і тестування.

У скриптових мовах програмування модулі зазвичай реалізуються у вигляді окремих файлів, що містять функції, змінні або класи. Для використання модуля у програмі застосовуються механізми імпорту, які дозволяють отримати доступ до його вмісту.

Важливим є правильне визначення меж модулів. Кожен модуль повинен містити логічно пов’язаний функціонал і не залежати від внутрішньої реалізації інших модулів. Це дозволяє забезпечити слабке зв’язування між частинами програми.

Модулі можуть містити як користувацькі функції, так і готові бібліотеки. Використання стандартних бібліотек дозволяє скоротити час розробки і підвищити надійність програмного забезпечення.

У процесі роботи з модулями важливим є правильне використання простору імен. Це дозволяє уникнути конфліктів між однаковими іменами змінних або функцій у різних модулях.

Таким чином, модульна організація коду є ефективним підходом до розробки програм, що забезпечує їх структурованість, повторне використання та зручність супроводження.

Практичне завдання

1. Створити окремий модуль із функціями для виконання обчислень.
2. Реалізувати імпорт створеного модуля у головну програму.
3. Використати функції модуля для обробки даних.
4. Створити модуль для роботи з текстовими даними.
5. Використати стандартні бібліотеки скриптової мови програмування.
6. Реалізувати програму з декількома модулями.
7. Дослідити простір імен при використанні модулів.
8. Проаналізувати структуру програми з точки зору модульності.

Методичні вказівки до виконання практичного завдання

Перед початком виконання роботи необхідно ознайомитися з поняттям модуля, принципами модульної організації коду та способами імпорту модулів у програмі. Особливу увагу слід приділити структурі файлів і правилам доступу до елементів модуля.

На першому етапі необхідно створити окремий файл, що буде виконувати роль модуля. У цьому файлі слід реалізувати декілька функцій, наприклад для виконання арифметичних обчислень.

Далі необхідно створити головну програму, у якій буде використано створений модуль. Для цього слід виконати імпорт модуля і викликати його функції. Необхідно перевірити правильність роботи програми та доступ до функцій модуля.

На наступному етапі необхідно створити ще один модуль, наприклад для обробки текстових даних. Це дозволить продемонструвати використання декількох модулів у межах однієї програми.

Після цього необхідно використати стандартні бібліотеки мови програмування. Наприклад, можна застосувати функції для роботи з математичними обчисленнями або обробки рядків.

Особливу увагу слід приділити простору імен. Необхідно перевірити, як відбувається доступ до функцій і змінних з різних модулів, а також як уникнути конфліктів імен.

У процесі виконання роботи необхідно проаналізувати структуру програми. Слід визначити, наскільки вона є модульною, чи логічно розподілено функціональність між модулями та чи зручно використовувати створені компоненти.

Контрольні питання

1. Що таке модуль у програмуванні?
2. Яке призначення модульної організації коду?
3. Як здійснюється імпорт модуля?
4. Що таке простір імен?
5. Які переваги дає використання модулів?
6. Як уникнути конфліктів імен у програмі?
7. Яка різниця між користувацькими та стандартними модулями?
8. Як організовується структура програмного проєкту?
9. Які помилки можуть виникати при роботі з модулями?
10. Яке значення має модульність у програмній інженерії?

Зміст звіту

– назва практичної роботи;
– мета виконання;
– опис принципів модульної організації;
– опис створених модулів;
– приклади використання модулів у програмі;
– результати тестування;
– висновки.

Практична робота 9. Використання стандартних бібліотек у програмуванні

Мета роботи – ознайомитися з можливостями стандартних бібліотек скриптових мов програмування, навчитися підключати та використовувати готові функції і модулі для розв'язання прикладних задач.

Ключові положення

Стандартні бібліотеки є невід'ємною складовою сучасних мов програмування і містять набір готових функцій, модулів та засобів для виконання типових задач. Їх використання дозволяє значно скоротити час розробки програмного забезпечення і підвищити його надійність.

У скриптових мовах програмування стандартні бібліотеки зазвичай містять засоби для роботи з математичними обчисленнями, текстовими даними, файлами, датою і часом, а також іншими типами інформації. Для доступу до цих можливостей необхідно виконати імпорт відповідного модуля.

Використання стандартних бібліотек дозволяє уникнути повторної реалізації вже існуючих алгоритмів. Це сприяє підвищенню якості програмного коду та його відповідності сучасним практикам розробки програмного забезпечення.

Важливим є правильний вибір бібліотеки для розв'язання конкретної задачі. Програміст повинен розуміти, які можливості надає кожен модуль і як його можна ефективно використати у програмі.

Під час роботи з бібліотеками необхідно враховувати простір імен та спосіб імпорту. Можна імпортувати весь модуль або лише окремі функції, що впливає на зручність використання і читабельність коду.

Стандартні бібліотеки забезпечують перевірені та оптимізовані рішення, що дозволяє підвищити ефективність програм і зменшити ймовірність помилок.

Таким чином, використання стандартних бібліотек є важливим елементом сучасного програмування, що забезпечує ефективність, надійність та зручність розробки програмного забезпечення.

Практичне завдання

1. Підключити стандартну бібліотеку для виконання математичних обчислень.
2. Реалізувати програму з використанням функцій для обчислення математичних виразів.

3. Використати бібліотеку для роботи з датою і часом.
4. Реалізувати програму для обробки рядків із використанням стандартних функцій.
5. Використати бібліотеку для роботи з випадковими числами.
6. Порівняти власну реалізацію алгоритму з використанням функцій бібліотеки.
7. Реалізувати програму з комбінованим використанням декількох бібліотек.
8. Проаналізувати ефективність використання стандартних бібліотек.

Методичні вказівки до виконання практичного завдання

Перед початком виконання роботи необхідно ознайомитися зі структурою стандартних бібліотек вибраної скриптової мови програмування та способами їх підключення. Особливу увагу слід приділити синтаксису імпорту модулів і виклику їх функцій.

На першому етапі необхідно підключити бібліотеку для математичних обчислень. Після цього слід реалізувати програму, яка використовує функції цієї бібліотеки для виконання різних обчислень, наприклад обчислення кореня, степеня або тригонометричних функцій.

Далі необхідно використати бібліотеку для роботи з датою і часом. Наприклад, можна отримати поточну дату, визначити різницю між двома датами або виконати форматування дати.

На наступному етапі необхідно реалізувати обробку текстових даних із використанням стандартних функцій. Це може включати зміну регістру, пошук підрядків або форматування рядків.

Після цього необхідно використати бібліотеку для генерації випадкових чисел. Наприклад, можна реалізувати генерацію випадкового числа у заданому діапазоні або створити просту модель випадкового процесу.

Далі необхідно порівняти власну реалізацію певного алгоритму з використанням функцій бібліотеки. Це дозволить оцінити переваги використання готових рішень.

На завершальному етапі необхідно реалізувати програму, яка використовує декілька бібліотек одночасно. Це дозволить продемонструвати можливості інтеграції різних функціональних засобів у межах одного застосунку.

У процесі виконання роботи необхідно проаналізувати ефективність використання стандартних бібліотек, зокрема з точки зору скорочення обсягу коду, підвищення його зрозумілості та надійності.

Контрольні питання

1. Що таке стандартна бібліотека у програмуванні?
2. Яке призначення стандартних бібліотек?
3. Як здійснюється підключення бібліотек у програмі?
4. Які задачі можна розв’язувати за допомогою стандартних бібліотек?
5. Які переваги дає використання стандартних бібліотек?
6. Що таке простір імен у контексті бібліотек?
7. Які способи імпорту модулів існують?
8. Чому важливо використовувати готові бібліотеки замість власної реалізації?
9. Які помилки можуть виникати при використанні бібліотек?
10. Яке значення мають стандартні бібліотеки у програмній інженерії?

Зміст звіту

– назва практичної роботи;
– мета виконання;
– опис використаних бібліотек;
– опис реалізованих програм;
– приклади виконання програм;
– результати тестування;
– висновки.

Практична робота 10. Підключення та використання сторонніх бібліотек

Мета роботи – ознайомитися з принципами використання сторонніх бібліотек у скриптових мовах програмування, навчитися встановлювати, підключати та використовувати зовнішні програмні компоненти для розширення функціональності програм.

Ключові положення

Сторонні бібліотеки є додатковими програмними компонентами, що розробляються незалежними авторами і призначені для розширення можливостей мови програмування. Вони дозволяють реалізувати складні функції без необхідності створювати їх з нуля.

На відміну від стандартних бібліотек, сторонні бібліотеки потребують попереднього встановлення. У скриптових мовах програмування це зазвичай виконується за допомогою спеціальних менеджерів пакетів, які забезпечують завантаження, встановлення та оновлення бібліотек.

Використання сторонніх бібліотек дозволяє значно пришвидшити процес розробки, оскільки програміст може скористатися вже готовими рішеннями для роботи з мережею, обробки даних, візуалізації або взаємодії з іншими системами.

Важливим є вибір бібліотеки. Необхідно враховувати її функціональність, актуальність, підтримку та сумісність із використовуваною версією мови програмування.

Під час використання сторонніх бібліотек необхідно звертати увагу на документацію. Саме вона містить опис функцій, приклади використання та рекомендації щодо інтеграції бібліотеки у програму.

Слід також враховувати питання безпеки та стабільності. Використання ненадійних або застарілих бібліотек може призвести до помилок або вразливостей у програмному забезпеченні.

Таким чином, сторонні бібліотеки є потужним інструментом розширення можливостей програм, але їх використання потребує уважного підходу та аналізу.

Практичне завдання

1. Встановити сторонню бібліотеку за допомогою менеджера пакетів.
2. Підключити встановлену бібліотеку до програми.
3. Реалізувати програму з використанням функцій сторонньої бібліотеки.
4. Використати бібліотеку для обробки даних або роботи з мережею.

5. Ознайомитися з документацією бібліотеки та застосувати приклади використання.
6. Реалізувати програму, що використовує декілька функцій бібліотеки.
7. Перевірити роботу програми після встановлення бібліотеки.
8. Проаналізувати переваги використання сторонніх бібліотек.

Методичні вказівки до виконання практичного завдання

Перед початком виконання роботи необхідно ознайомитися з поняттям сторонніх бібліотек та принципами їх використання у скриптових мовах програмування. Особливу увагу слід приділити способам встановлення бібліотек за допомогою менеджера пакетів.

На першому етапі необхідно обрати сторонню бібліотеку відповідно до поставленої задачі. Після цього слід виконати її встановлення з використанням відповідного інструмента. Потрібно перевірити успішність встановлення та доступність бібліотеки у середовищі розробки.

Далі необхідно підключити бібліотеку у програмі. Для цього слід використати відповідну конструкцію імпорту та перевірити доступ до її функцій.

На наступному етапі необхідно реалізувати програму, що використовує можливості бібліотеки. Наприклад, можна виконати обробку даних, отримання інформації з мережі або інші дії залежно від обраної бібліотеки.

Після цього необхідно ознайомитися з документацією бібліотеки. Слід вивчити приклади використання та реалізувати декілька функцій на їх основі.

Далі необхідно реалізувати програму, що використовує декілька функцій бібліотеки одночасно. Це дозволить продемонструвати її можливості у комплексному використанні.

У процесі виконання роботи необхідно протестувати програму та переконатися у правильності її роботи. Слід звернути увагу на можливі помилки, що можуть виникати при використанні бібліотеки.

На завершальному етапі необхідно проаналізувати ефективність використання сторонніх бібліотек, зокрема з точки зору скорочення обсягу коду та спрощення реалізації задач.

Контрольні питання

1. Що таке стороння бібліотека?
2. Чим сторонні бібліотеки відрізняються від стандартних?
3. Як здійснюється встановлення сторонніх бібліотек?
4. Яке призначення менеджера пакетів?
5. Як підключити бібліотеку у програмі?
6. Чому важливо використовувати документацію бібліотеки?
7. Які критерії вибору сторонньої бібліотеки?
8. Які ризики пов’язані з використанням сторонніх бібліотек?
9. Які помилки можуть виникати при встановленні або використанні бібліотек?
10. Яке значення мають сторонні бібліотеки у сучасному програмуванні?

Зміст звіту

– назва практичної роботи;
– мета виконання;
– опис використаної бібліотеки;
– опис процесу встановлення та підключення;
– опис реалізованої програми;
– результати тестування;
– висновки.

Практична робота 11. Створення класів та об’єктів у програмі

Мета роботи – ознайомитися з основами об’єктно-орієнтованого програмування, навчитися створювати класи та об’єкти, визначати їх властивості та методи, а також реалізовувати взаємодію між об’єктами.

Ключові положення

Об’єктно-орієнтоване програмування є підходом до розробки програмного забезпечення, що базується на використанні об’єктів. Об’єкт є сутністю, яка містить дані та методи для їх обробки. Дані описують стан об’єкта, а методи визначають його поведінку.

Клас є шаблоном для створення об’єктів. Він визначає набір властивостей і методів, які будуть доступні для об’єктів цього класу. Під час створення об’єкта на основі класу відбувається ініціалізація його властивостей.

Властивості класу визначають дані, що зберігаються в об’єкті. Методи є функціями, які виконують операції над цими даними. Такий підхід дозволяє об’єднати дані та операції над ними в єдину структуру.

Інкапсуляція є одним із основних принципів об’єктно-орієнтованого програмування. Вона полягає в обмеженні доступу до внутрішніх даних об’єкта та забезпеченні доступу до них через методи. Це підвищує надійність і захищеність програмного коду.

Об’єкти можуть взаємодіяти між собою шляхом виклику методів. Це дозволяє реалізувати складні програмні системи, що складаються з взаємопов’язаних компонентів.

У скриптових мовах програмування створення класів зазвичай є простим і гнучким. Це дозволяє швидко реалізовувати об’єктно-орієнтовані підходи до розробки програм.

Таким чином, використання класів та об’єктів забезпечує структурованість програмного коду, сприяє повторному використанню та спрощує супроводження програм.

Практичне завдання

1. Створити клас із набором властивостей.
2. Реалізувати методи для роботи з властивостями класу.
3. Створити об’єкти на основі класу.
4. Реалізувати ініціалізацію об’єктів.
5. Використати методи класу для обробки даних.
6. Реалізувати взаємодію між декількома об’єктами.
7. Дослідити зміну стану об’єкта під час виконання програми.
8. Проаналізувати використання класів у програмі.

Методичні вказівки до виконання практичного завдання

Перед початком виконання роботи необхідно ознайомитися з поняттями класу, об’єкта, властивостей і методів. Особливу увагу слід приділити принципу інкапсуляції та організації доступу до даних об’єкта.

На першому етапі необхідно створити простий клас, що містить декілька властивостей. Наприклад, клас може описувати певний об’єкт реального світу з відповідними характеристиками.

Далі необхідно реалізувати методи класу. Методи повинні виконувати операції над властивостями об’єкта, наприклад змінювати їх значення або виводити інформацію.

На наступному етапі необхідно створити об’єкти на основі класу. Кожен об’єкт має містити власні значення властивостей.

Після цього необхідно реалізувати ініціалізацію об’єктів. Для цього слід використати спеціальний метод, що задає початкові значення властивостей під час створення об’єкта.

Далі необхідно використати методи класу для обробки даних. Це дозволить перевірити правильність реалізації логіки класу.

На наступному етапі необхідно реалізувати взаємодію між об’єктами. Наприклад, один об’єкт може передавати дані іншому або викликати його методи.

У процесі виконання роботи необхідно проаналізувати зміну стану об’єктів і оцінити доцільність використання класів для реалізації задачі.

Контрольні питання

1. Що таке клас у програмуванні?
2. Що таке об’єкт?
3. Яка різниця між класом і об’єктом?
4. Що таке властивості класу?
5. Що таке методи класу?
6. У чому полягає принцип інкапсуляції?
7. Як створюється об’єкт?
8. Яке призначення методу ініціалізації?
9. Як реалізується взаємодія між об’єктами?
10. Які переваги дає використання об’єктно-орієнтованого підходу?

Зміст звіту

– назва практичної роботи;
– мета виконання;
– опис створеного класу;
– опис реалізованих методів;
– приклади створення та використання об’єктів;
– результати тестування;
– висновки.

Практична робота 12. Реалізація інкапсуляції та методів доступу до даних

Мета роботи – ознайомитися з принципом інкапсуляції в об’єктно-орієнтованому програмуванні, навчитися обмежувати доступ до даних об’єкта та реалізовувати методи доступу для керування станом об’єкта.

Ключові положення

Інкапсуляція є одним із базових принципів об’єктно-орієнтованого програмування, що передбачає об’єднання даних та методів їх обробки в межах одного класу та обмеження прямого доступу до внутрішнього стану об’єкта. Такий підхід дозволяє забезпечити цілісність даних і контроль їх зміни.

Дані, що зберігаються в об’єкті, зазвичай робляться недоступними для прямого доступу ззовні. Для роботи з ними використовуються спеціальні методи доступу, які дозволяють отримувати або змінювати значення властивостей. Це забезпечує контроль над тим, як саме змінюється стан об’єкта.

Методи доступу поділяються на методи читання та методи запису. Метод читання повертає значення властивості, а метод запису змінює його значення з урахуванням певних обмежень або перевірок.

Важливим є використання перевірок під час зміни значень властивостей. Це дозволяє запобігти встановленню некоректних значень і забезпечує правильну роботу програми.

Інкапсуляція також сприяє підвищенню безпеки програмного коду, оскільки обмежує доступ до внутрішніх механізмів роботи об’єкта. Це дозволяє змінювати реалізацію класу без впливу на інші частини програми.

У скриптових мовах програмування інкапсуляція часто реалізується за допомогою угод щодо іменування або спеціальних механізмів доступу. Незважаючи на це, її принципи залишаються однаковими для різних мов програмування.

Таким чином, інкапсуляція забезпечує контроль над доступом до даних, підвищує надійність програмного забезпечення та сприяє створенню зрозумілого і структурованого коду.

Практичне завдання

1. Створити клас із закритими властивостями.
2. Реалізувати методи читання значень властивостей.
3. Реалізувати методи зміни значень властивостей із перевіркою.
4. Створити об’єкти класу та перевірити доступ до даних через методи.
5. Реалізувати обмеження на допустимі значення властивостей.
6. Дослідити поведінку програми при спробі некоректного доступу до даних.
7. Реалізувати клас із декількома властивостями та методами доступу.
8. Проаналізувати використання інкапсуляції у програмі.

Методичні вказівки до виконання практичного завдання

Перед початком виконання роботи необхідно ознайомитися з принципом інкапсуляції та його значенням у програмуванні. Особливу увагу слід приділити методам доступу до даних і способам обмеження доступу до властивостей класу.

На першому етапі необхідно створити клас, у якому властивості є недоступними для прямого доступу. Для цього слід використати відповідні механізми мови програмування або прийняті угоди щодо іменування.

Далі необхідно реалізувати методи для отримання значень властивостей. Ці методи повинні повертати значення змінних, не змінюючи їх.

На наступному етапі необхідно реалізувати методи для зміни значень властивостей. У цих методах слід передбачити перевірку вхідних даних, щоб уникнути встановлення некоректних значень.

Після цього необхідно створити об’єкти класу та перевірити, що доступ до властивостей можливий лише через методи. Слід спробувати змінити значення властивостей безпосередньо і проаналізувати результат.

Далі необхідно реалізувати додаткові обмеження для властивостей. Наприклад, можна задати допустимий діапазон значень або тип даних.

У процесі виконання роботи необхідно протестувати програму для різних варіантів вхідних даних, зокрема некоректних. Це дозволить перевірити ефективність реалізованих методів доступу.

На завершальному етапі необхідно проаналізувати результати роботи та оцінити доцільність використання інкапсуляції у розробленій програмі.

Контрольні питання

1. Що таке інкапсуляція?
2. Яке призначення інкапсуляції у програмуванні?
3. Що таке закриті властивості?
4. Які методи доступу використовуються для роботи з даними?
5. У чому різниця між методом читання і методом запису?
6. Чому важливо перевіряти значення при зміні властивостей?
7. Як інкапсуляція впливає на безпеку програмного коду?
8. Які способи реалізації інкапсуляції існують у скриптових мовах?
9. Які помилки можуть виникати при відсутності інкапсуляції?
10. Яке значення має інкапсуляція у об’єктно-орієнтованому програмуванні?

Зміст звіту

– назва практичної роботи;
– мета виконання;
– опис принципу інкапсуляції;
– опис реалізованого класу;
– опис методів доступу;
– результати тестування;
– висновки.

Практична робота 13. Використання колекцій даних для обробки інформації

Мета роботи – ознайомитися з основними типами колекцій даних у скриптових мовах програмування, навчитися використовувати їх для збереження та обробки інформації, а також реалізовувати алгоритми роботи з наборами даних.

Ключові положення

Колекції даних є структурами, що дозволяють зберігати множину значень у межах однієї змінної. Вони широко використовуються для організації та обробки інформації у програмному забезпеченні. До найпоширеніших колекцій належать списки, множини, словники та кортежі.

Списки є впорядкованими колекціями, що дозволяють зберігати елементи різних типів. Вони підтримують доступ за індексом, зміну значень та додавання нових елементів. Це робить їх універсальним інструментом для обробки послідовностей даних.

Множини є невпорядкованими колекціями унікальних елементів. Вони використовуються для виконання операцій над множинами, таких як об’єднання, перетин і різниця. Це дозволяє ефективно працювати з унікальними значеннями.

Словники є структурами, що зберігають дані у вигляді пар ключ–значення. Вони дозволяють швидко отримувати значення за ключем і широко застосовуються для зберігання структурованої інформації.

Кортежі є впорядкованими колекціями, які після створення не змінюються. Вони використовуються для збереження фіксованих наборів даних.

Колекції дозволяють реалізовувати ефективну обробку даних за допомогою циклів і умовних операторів. Вони є основою багатьох алгоритмів обробки інформації.

Важливим є правильний вибір типу колекції залежно від задачі. Це впливає на ефективність роботи програми та зручність її реалізації.

Таким чином, використання колекцій даних дозволяє ефективно організувати збереження та обробку інформації у програмних системах.

Практичне завдання

1. Створити список та виконати основні операції над його елементами.
2. Реалізувати додавання, видалення та зміну елементів списку.
3. Використати множину для роботи з унікальними значеннями.
4. Реалізувати операції над множинами.
5. Створити словник і виконати операції доступу за ключем.
6. Реалізувати обробку даних, що зберігаються у словнику.
7. Використати кортеж для збереження фіксованих даних.
8. Реалізувати обробку колекцій із використанням циклів.

Методичні вказівки до виконання практичного завдання

Перед початком виконання роботи необхідно ознайомитися з основними типами колекцій даних та їх властивостями. Особливу увагу слід приділити відмінностям між списками, множинами, словниками та кортежами.

На першому етапі необхідно створити список і виконати базові операції над ним. Слід додати декілька елементів, змінити їх значення та видалити окремі елементи.

Далі необхідно використати множину для збереження унікальних значень. Слід реалізувати операції об’єднання, перетину та різниці між множинами.

На наступному етапі необхідно створити словник. У словнику слід зберігати дані у вигляді пар ключ–значення. Необхідно реалізувати доступ до значень за ключами та зміну даних.

Після цього необхідно використати кортеж для збереження фіксованого набору значень. Слід перевірити, що значення кортежу не можуть бути змінені після створення.

Далі необхідно реалізувати обробку колекцій за допомогою циклів. Наприклад, можна перебрати всі елементи списку або словника і виконати певні операції над ними.

У процесі виконання роботи необхідно протестувати програму на різних наборах даних. Це дозволить оцінити правильність реалізації алгоритмів обробки інформації.

Контрольні питання

1. Що таке колекція даних?
2. Які типи колекцій використовуються у скриптових мовах програмування?
3. Які особливості списків?
4. У чому полягає призначення множин?
5. Як працюють словники?
6. Що таке кортеж і які його властивості?
7. Які операції можна виконувати над колекціями?
8. Як здійснюється обробка колекцій за допомогою циклів?
9. Які помилки можуть виникати при роботі з колекціями?
10. Як вибрати тип колекції для конкретної задачі?

Зміст звіту

– назва практичної роботи;
– мета виконання;
– опис використаних колекцій;
– опис реалізованих алгоритмів;
– приклади виконання програм;
– результати тестування;
– висновки.

Практична робота 14. Робота з файлами: зчитування та запис даних

Мета роботи – ознайомитися з основами роботи з файлами у скриптових мовах програмування, навчитися виконувати зчитування та запис даних, а також реалізовувати обробку текстової інформації у файлах.

Ключові положення

Робота з файлами є важливою складовою програмування, оскільки дозволяє зберігати дані поза межами виконання програми та повторно використовувати їх у майбутньому. Файли можуть містити текстову або бінарну інформацію, що визначає спосіб їх обробки.

Основними операціями при роботі з файлами є відкриття, читання, запис і закриття. Перед виконанням будь-яких операцій файл необхідно відкрити у відповідному режимі. Режим відкриття визначає, які операції будуть доступні, наприклад лише читання або запис.

Зчитування даних може виконуватися повністю або частинами. При роботі з текстовими файлами часто використовується построкове зчитування, що дозволяє ефективно обробляти великі обсяги інформації.

Запис даних у файл може здійснюватися у режимі створення нового файлу або дописування до існуючого. Важливо враховувати, що у режимі перезапису попередній вміст файлу буде втрачено.

Під час роботи з файлами необхідно забезпечити їх коректне закриття після завершення операцій. Це дозволяє уникнути втрати даних і забезпечує правильне завершення роботи програми.

Важливим є обробка помилок, що можуть виникати при роботі з файлами, наприклад відсутність файлу або відсутність прав доступу. Програма повинна передбачати такі ситуації і коректно на них реагувати.

Таким чином, робота з файлами дозволяє організувати збереження і обробку даних, що є необхідним для більшості прикладних програм.

Практичне завдання

1. Створити програму для відкриття текстового файлу у режимі читання.
2. Реалізувати зчитування даних з файлу.
3. Вивести вміст файлу у консоль.
4. Реалізувати запис даних у файл.
5. Створити новий файл і записати у нього текстову інформацію.
6. Реалізувати дописування даних у існуючий файл.
7. Обробити можливі помилки при роботі з файлами.
8. Проаналізувати результати роботи програми.

Методичні вказівки до виконання практичного завдання

Перед початком виконання роботи необхідно ознайомитися з основними операціями роботи з файлами та режимами їх відкриття. Особливу увагу слід приділити різниці між режимами читання, запису та дописування.

На першому етапі необхідно реалізувати відкриття файлу у режимі читання. Слід перевірити, чи файл існує, і у разі його відсутності передбачити відповідну обробку помилки.

Далі необхідно реалізувати зчитування даних з файлу. Це можна зробити повністю або построково. Результати зчитування слід вивести у консоль для перевірки правильності роботи.

На наступному етапі необхідно реалізувати запис даних у файл. Для цього слід відкрити файл у режимі запису та записати у нього певну текстову інформацію.

Після цього необхідно реалізувати дописування даних у файл. Слід відкрити файл у режимі дописування і додати нові дані до вже існуючого вмісту.

Особливу увагу слід приділити коректному закриттю файлу після завершення операцій. Це можна зробити вручну або за допомогою спеціальних конструкцій, що автоматично закривають файл.

У процесі виконання роботи необхідно реалізувати обробку помилок. Наприклад, слід передбачити ситуації, коли файл не існує або виникає помилка доступу.

На завершальному етапі необхідно протестувати програму та проаналізувати результати виконання.

Контрольні питання

1. Що таке файл у програмуванні?
2. Які основні операції виконуються при роботі з файлами?
3. Які режими відкриття файлів існують?
4. Як здійснюється зчитування даних з файлу?
5. Як виконується запис даних у файл?
6. У чому різниця між режимами запису та дописування?
7. Чому важливо закривати файл після роботи з ним?
8. Які помилки можуть виникати при роботі з файлами?
9. Як реалізується обробка помилок при роботі з файлами?
10. Яке значення має робота з файлами у програмуванні?

Зміст звіту

– назва практичної роботи;
– мета виконання;
– опис принципів роботи з файлами;
– опис реалізованих програм;
– приклади зчитування та запису даних;
– результати тестування;
– висновки.

Практична робота 15. Розробка програм для обробки даних

Мета роботи – узагальнити знання щодо обробки даних у скриптових мовах програмування, навчитися розробляти прикладні програми для аналізу, перетворення та збереження даних.

Ключові положення

Обробка даних є ключовою задачею у програмній інженерії, оскільки більшість програмних систем призначені для отримання, аналізу та перетворення інформації.

Скриптові мови програмування широко застосовуються для реалізації таких задач завдяки простоті синтаксису та наявності потужних засобів роботи з даними.

Процес обробки даних зазвичай містить декілька етапів: отримання даних, їх перевірка, перетворення, аналіз та виведення результатів. Кожен із цих етапів має бути реалізований у програмі з урахуванням можливих помилок і особливостей вхідної інформації.

Для обробки даних використовуються різні структури, зокрема списки, словники та інші колекції. Вони дозволяють організувати збереження інформації та виконувати її обробку за допомогою циклів і умовних операторів.

Важливим є забезпечення коректності даних. Перед виконанням обчислень необхідно перевіряти введені значення та виконувати перетворення типів за потреби. Це дозволяє уникнути помилок і забезпечує правильність результатів.

У процесі обробки даних часто використовуються функції та модулі, що дозволяють структурувати програму та повторно використовувати код. Це підвищує ефективність розробки та спрощує супроводження програмного забезпечення.

Результати обробки можуть виводитися у консоль, записуватися у файл або використовуватися для подальших обчислень. Вибір способу виведення залежить від задачі та вимог до програми.

Таким чином, розробка програм для обробки даних передбачає комплексне використання основних засобів програмування і є важливим етапом підготовки фахівців у галузі програмної інженерії.

Практичне завдання

1. Реалізувати програму для введення даних користувачем або з файлу.
2. Виконати перевірку коректності введених даних.
3. Реалізувати перетворення даних до потрібного формату.
4. Виконати обробку даних із використанням колекцій.
5. Реалізувати обчислення необхідних показників (сума, середнє значення, максимум, мінімум).
6. Використати функції для структурування програми.
7. Реалізувати виведення результатів у консоль або файл.
8. Проаналізувати результати виконання програми.

Методичні вказівки до виконання практичного завдання

Перед початком виконання роботи необхідно визначити задачу обробки даних і структуру програми. Слід визначити, які дані будуть використовуватися, як вони будуть вводитися і які результати необхідно отримати.

На першому етапі необхідно реалізувати введення даних. Дані можуть вводитися з клавіатури або зчитуватися з файлу. Після введення необхідно виконати перевірку їх коректності.

Далі необхідно виконати перетворення даних. Наприклад, текстові значення можуть бути перетворені у числові для виконання обчислень.

На наступному етапі необхідно реалізувати обробку даних. Для цього слід використати відповідні структури даних і виконати необхідні обчислення або аналіз.

Після цього необхідно структурувати програму за допомогою функцій. Це дозволить розділити програму на логічні частини і підвищити її зрозумілість.

Далі необхідно реалізувати виведення результатів. Результати можуть бути виведені у консоль або записані у файл.

У процесі виконання роботи необхідно протестувати програму на різних наборах даних, включаючи граничні випадки. Це дозволить перевірити правильність реалізації алгоритмів.

На завершальному етапі необхідно проаналізувати результати роботи програми та зробити висновки щодо її ефективності.

Контрольні питання

1. Що таке обробка даних у програмуванні?
2. Які етапи містить процес обробки даних?
3. Які структури даних використовуються для обробки інформації?
4. Чому важливо перевіряти коректність даних?
5. Як виконується перетворення типів даних?
6. Які засоби використовуються для структурування програм?
7. Як реалізується виведення результатів?
8. Які помилки можуть виникати при обробці даних?
9. Як забезпечити правильність результатів обчислень?
10. Яке значення має обробка даних у програмній інженерії?

Зміст звіту

– назва практичної роботи;
– мета виконання;
– опис задачі обробки даних;
– опис реалізованої програми;
– приклади вхідних і вихідних даних;
– результати тестування;
– висновки.

Практична робота 16. Реалізація обробки винятків у програмі

Мета роботи – ознайомитися з принципами обробки виняткових ситуацій у скриптових мовах програмування, навчитися виявляти та обробляти помилки під час виконання програми, а також забезпечувати її стійку та передбачувану роботу.

Ключові положення

Під час виконання програми можуть виникати ситуації, які порушують нормальний хід її роботи. До таких ситуацій належать помилки введення даних, спроби виконання недопустимих операцій, відсутність файлів, помилки перетворення типів та інші непередбачені стани. Для керованого реагування на такі випадки у мовах програмування використовуються механізми обробки винятків.

Виняток є повідомленням про помилкову або нестандартну ситуацію, що виникла під час виконання програми. Якщо така ситуація не обробляється, виконання програми може бути аварійно завершене. Обробка винятків дає змогу перехопити помилку, виконати необхідні дії та або продовжити роботу програми, або завершити її коректно.

У скриптових мовах програмування для обробки винятків зазвичай використовуються конструкції try, except, else та finally. У блоці try розміщується код, під час виконання якого може виникнути виняток. Блок except призначений для обробки конкретного типу помилки. Блок else виконується у разі відсутності помилки, а блок finally містить код, який виконується незалежно від результату виконання.

Важливим є правильне визначення типів винятків. Наприклад, помилки ділення на нуль, помилки перетворення типів або помилки доступу до файлів мають різну природу і потребують різних способів обробки. Надто загальна обробка всіх можливих винятків ускладнює аналіз роботи програми, тому доцільно передбачати обробку саме тих ситуацій, які можуть виникати у конкретній задачі.

У деяких випадках програма може самостійно генерувати винятки. Це використовується тоді, коли необхідно примусово повідомити про некоректний стан даних або порушення логіки роботи. Такий підхід дає змогу підвищити контроль над правильністю виконання алгоритму.

Обробка винятків має важливе значення для забезпечення надійності програмного забезпечення. Вона дозволяє уникати аварійного завершення програми, забезпечує інформування користувача про помилки та дає змогу коректно завершувати роботу із зовнішніми ресурсами, зокрема файлами або мережевими з’єднаннями.

Таким чином, механізми обробки винятків є важливим засобом підвищення стійкості програм, що дозволяє коректно реагувати на помилкові ситуації та забезпечувати надійну роботу застосунків.

Практичне завдання

1. Реалізувати програму з використанням конструкції try та except.
2. Обробити помилку введення некоректних числових даних.
3. Реалізувати обробку помилки ділення на нуль.
4. Створити програму з обробкою помилок під час відкриття файлу.
5. Використати декілька блоків except для різних типів винятків.
6. Реалізувати використання блоків else та finally.
7. Створити приклад генерації винятку у програмі.
8. Проаналізувати поведінку програми у разі виникнення різних помилок.

Методичні вказівки до виконання практичного завдання

Перед початком виконання роботи необхідно ознайомитися з поняттям виняткової ситуації та основними конструкціями, що використовуються для її обробки. Особливу увагу слід приділити відмінності між помилками синтаксису, які виявляються до початку виконання програми, і винятками, що виникають під час її роботи.

На першому етапі необхідно реалізувати найпростіший приклад обробки винятку. Для цього слід розмістити у блоці try код, який може спричинити помилку, наприклад

перетворення введеного рядка у число. У блоці except необхідно передбачити виведення повідомлення про помилку.

Далі необхідно реалізувати приклад обробки ділення на нуль. Слід організувати введення двох чисел, після чого виконати операцію ділення. Якщо друге число дорівнює нулю, програма повинна перехопити виняток і вивести зрозуміле повідомлення без аварійного завершення.

На наступному етапі необхідно реалізувати обробку помилок під час роботи з файлами. Для цього слід спробувати відкрити файл, який може бути відсутнім. У разі виникнення помилки програма повинна повідомити про неможливість відкриття файлу.

Після цього необхідно використати декілька блоків except для обробки різних типів помилок. Це дозволить продемонструвати, як програма може по-різному реагувати на різні виняткові ситуації.

Далі необхідно застосувати блок else, у якому розміщується код, що виконується лише за відсутності помилок, а також блок finally, у якому виконується завершальна дія, наприклад закриття файлу або виведення повідомлення про завершення роботи.

На наступному етапі необхідно реалізувати приклад самостійної генерації винятку. Наприклад, можна передбачити ситуацію, коли введене значення не задовольняє встановлену умову, і у такому випадку ініціювати виняток програмним шляхом.

У процесі виконання роботи необхідно протестувати програму для різних варіантів вхідних даних, зокрема коректних і помилкових. Це дозволить оцінити правильність реалізації механізмів обробки винятків і їх вплив на стійкість програми.

Контрольні питання

1. Що таке виняток у програмуванні?
2. Які ситуації можуть призводити до виникнення винятків?
3. Яке призначення блоку try?
4. Для чого використовується блок except?
5. У яких випадках доцільно використовувати декілька блоків except?
6. Яке призначення блоку else?
7. Для чого використовується блок finally?
8. Що таке генерація винятку?
9. Чому важливо обробляти винятки у програмі?
10. Які помилки найчастіше виникають під час роботи з даними та файлами?

Зміст звіту

– назва практичної роботи;
– мета виконання;
– опис принципів обробки винятків;
– опис реалізованих прикладів;
– приклади коректної та помилкової роботи програми;
– результати тестування;
– висновки.

Практична робота 17. Налагодження та тестування програмного коду

Мета роботи – ознайомитися з основними підходами до налагодження та тестування програмного коду, навчитися виявляти, локалізувати та виправляти помилки, а також перевіряти правильність роботи програми на різних наборах вхідних даних.

Ключові положення

Налагодження програмного коду є важливим етапом розробки програмного забезпечення, що полягає у виявленні причин некоректної роботи програми та їх усуненні. Під час написання програм можуть виникати синтаксичні, логічні та виконувані помилки. Кожен із цих типів помилок має свої особливості і потребує відповідних підходів до пошуку та виправлення.

Синтаксичні помилки виникають у разі порушення правил запису конструкцій мови програмування. Такі помилки виявляються ще до початку виконання програми. Помилки виконання проявляються під час роботи програми, наприклад при діленні на нуль або зверненні до неіснуючого елемента. Логічні помилки є найбільш складними для виявлення, оскільки програма виконується без аварійного завершення, але формує неправильний результат.

Налагодження передбачає поетапний аналіз виконання програми. Для цього можуть використовуватися виведення проміжних значень змінних, перевірка виконання окремих ділянок коду, а також спеціальні засоби відлагодження, які дозволяють виконувати програму покроково, контролювати зміну значень змінних та аналізувати послідовність виконання команд.

Тестування програмного забезпечення полягає у перевірці правильності роботи програми на заздалегідь підготовлених наборах даних. Воно дозволяє встановити, чи відповідає програма очікуваним результатам, а також виявити приховані помилки, які не були помічені під час налагодження.

Під час тестування важливо використовувати різні категорії тестових даних. До них належать коректні значення, граничні значення, порожні або нестандартні дані, а також помилкові вхідні значення. Такий підхід дозволяє всебічно оцінити поведінку програми.

Важливою складовою перевірки є зіставлення фактичних результатів із очікуваними. Якщо результат не відповідає очікуваному, необхідно виконати повторний аналіз алгоритму, перевірити обчислення та встановити причину помилки.

Налагодження і тестування є взаємопов’язаними процесами. Налагодження дозволяє усунути виявлені помилки, а тестування підтверджує правильність внесених змін. Без систематичного тестування навіть формально працездатна програма може містити приховані дефекти.

Таким чином, налагодження та тестування програмного коду є необхідними етапами розробки програмного забезпечення, що забезпечують правильність, надійність і передбачуваність роботи програм.

Практичне завдання

1. Реалізувати програму, що містить декілька типових помилок.
2. Виявити та виправити синтаксичні помилки у програмному коді.

3. Виявити та виправити помилки виконання.

4. Проаналізувати логіку роботи програми та знайти логічні помилки.

5. Використати виведення проміжних значень для налагодження коду.

6. Підготувати набір тестових даних для перевірки програми.

7. Виконати тестування програми на коректних, граничних та помилкових даних.

8. Проаналізувати результати тестування та зробити висновки щодо правильності роботи програми.

Методичні вказівки до виконання практичного завдання

Перед початком виконання роботи необхідно ознайомитися з основними типами помилок, що можуть виникати у програмному коді, а також з базовими підходами до їх виявлення. Особливу увагу слід приділити відмінності між синтаксичними, логічними помилками та помилками виконання.

На першому етапі необхідно підготувати або використати програму, що містить навмисно допущені помилки. Спочатку слід перевірити, чи програма запускається, і у разі виявлення синтаксичних помилок виправити їх відповідно до правил мови програмування.

Далі необхідно виконати програму і проаналізувати її роботу. Якщо під час виконання виникають помилки, слід визначити, на якому етапі вони з’являються, які дії до цього призвели та які значення мали змінні у цей момент. Для цього доцільно використати виведення проміжних результатів у консоль.

На наступному етапі необхідно перевірити логіку роботи програми. Якщо програма виконується без аварійного завершення, але формує неправильний результат, слід покроково проаналізувати алгоритм, перевірити правильність умов, циклів, обчислень та обробки вхідних даних.

Після цього необхідно підготувати набір тестових даних. До нього слід включити типові коректні значення, граничні значення, а також некоректні вхідні дані. Для кожного тестового прикладу доцільно визначити очікуваний результат ще до запуску програми.

Далі необхідно виконати тестування програми на підготовлених даних. У процесі тестування слід фіксувати фактичні результати та порівнювати їх із очікуваними. У разі виявлення невідповідностей необхідно повернутися до аналізу коду і виправити виявлені недоліки.

Особливу увагу слід приділити граничним випадкам, оскільки саме вони часто виявляють приховані дефекти у логіці програми. Наприклад, слід перевіряти нульові значення, порожні рядки, мінімальні та максимальні допустимі значення.

На завершальному етапі необхідно проаналізувати результати налагодження та тестування, оцінити, які саме помилки були виявлені, як вони були усунені та наскільки покращилася правильність роботи програми після внесення змін.

Контрольні питання

1. Що таке налагодження програмного коду?
2. Які основні типи помилок можуть виникати у програмі?
3. Чим логічна помилка відрізняється від синтаксичної?
4. Які засоби можна використовувати для налагодження програми?
5. Для чого виконують виведення проміжних значень змінних?

6. Що таке тестування програмного коду?
7. Які види тестових даних доцільно використовувати?
8. Чому важливо перевіряти граничні значення?
9. Як визначити, що програма працює правильно?
10. Який зв’язок між налагодженням і тестуванням?

Зміст звіту

– назва практичної роботи;
– мета виконання;
– опис виявлених помилок у програмі;
– опис процесу налагодження;
– перелік тестових даних і очікуваних результатів;
– результати тестування;
– висновки.

Практична робота 18. Розробка програмного модуля для автоматизації задач

Мета роботи – ознайомитися з принципами створення програмних модулів для автоматизації типових задач, навчитися проєктувати структуру модуля, реалізовувати його функції та використовувати розроблений компонент у прикладній програмі.

Ключові положення

Автоматизація задач є важливим напрямом застосування скриптових мов програмування, оскільки дає змогу скоротити обсяг рутинних операцій, підвищити швидкість обробки даних і зменшити ймовірність помилок, пов’язаних із ручним виконанням дій. Особливо доцільним є використання програмних модулів, які містять набір функцій для багаторазового застосування у різних програмах.

Програмний модуль є окремою частиною програмного коду, що реалізує логічно завершений набір операцій. Зазвичай модуль оформлюється у вигляді окремого файлу, який містить функції, константи, класи або інші програмні елементи, необхідні для виконання певної групи задач. Такий підхід дає змогу відокремити службову логіку від основної програми та спростити супроводження програмного коду.

Під час розробки модуля для автоматизації важливо чітко визначити його призначення. Наприклад, модуль може бути призначений для автоматичного перейменування файлів, обробки текстових даних, формування звітів, виконання типових обчислень або перевірки вхідної інформації. Усі функції модуля повинні бути узгодженими між собою та відповідати спільній задачі.

Однією з переваг модульного підходу є повторне використання коду. Якщо одна й та сама функціональність потрібна у декількох програмах, доцільно винести її в окремий модуль. Це дозволяє уникнути дублювання коду, спрощує внесення змін і полегшує тестування.

Під час автоматизації задач особливого значення набуває коректна організація вхідних і вихідних даних. Функції модуля повинні отримувати параметри у визначеному форматі, виконувати необхідну обробку та повертати результат у зручному для подальшого

використання вигляді. Якщо дані є некоректними, модуль повинен передбачати відповідну перевірку та повідомлення про помилку.

Важливою вимогою до програмного модуля є зрозумілість його структури. Імена функцій, параметрів і змінних повинні відображати їх призначення. Код модуля має бути логічно організованим, щоб його можна було легко використовувати в інших програмах або доповнювати новими можливостями.

Автоматизація задач за допомогою модулів дозволяє підвищити ефективність прикладного програмування. У результаті створюються програмні компоненти, які можна адаптувати до різних умов використання і застосовувати для розв'язання типових практичних задач.

Таким чином, розробка програмного модуля для автоматизації задач є важливим етапом формування навичок модульного проєктування, повторного використання коду та створення прикладних програмних засобів.

Практичне завдання

1. Визначити задачу, що підлягає автоматизації засобами програмного модуля.
2. Спроєктувати структуру модуля та перелік його основних функцій.
3. Реалізувати окремий програмний модуль у вигляді файлу з функціями.
4. Створити функції для автоматизованого виконання типових операцій.
5. Реалізувати перевірку коректності вхідних даних у функціях модуля.
6. Підключити створений модуль до основної програми.
7. Продемонструвати роботу модуля на прикладі автоматизації конкретної задачі.
8. Проаналізувати доцільність використання модуля для повторного застосування.

Методичні вказівки до виконання практичного завдання

Перед початком виконання роботи необхідно визначити типову задачу, яку доцільно автоматизувати засобами скриптової мови програмування. До таких задач можуть належати опрацювання списку даних, автоматичне формування текстових повідомлень, перетворення вхідної інформації, пакетна обробка файлів або виконання серії однотипних обчислень. Особливу увагу слід приділити тому, щоб задача була практично значущою і передбачала багаторазове виконання схожих дій.

На першому етапі необхідно спроєктувати структуру програмного модуля. Слід визначити, які саме функції має містити модуль, які параметри вони повинні приймати та які результати повертати. Доцільно відразу встановити єдині правила іменування функцій і параметрів, щоб модуль був зрозумілим і зручним у використанні.

Далі необхідно створити окремий файл модуля та реалізувати в ньому запроєктовані функції. Кожна функція повинна виконувати окрему логічно завершену дію. Наприклад, одна функція може виконувати перевірку даних, інша — їх перетворення, а ще одна — формування підсумкового результату. Такий підхід дозволяє зробити модуль гнучким і зручним для тестування.

На наступному етапі необхідно передбачити перевірку коректності вхідних даних. Якщо функція працює з числами, слід перевіряти допустимість їх значень. Якщо обробляються текстові дані або списки, доцільно перевірити формат і наявність потрібних

елементів. У разі виявлення некоректних даних функція повинна або повертати повідомлення про помилку, або генерувати виняток залежно від логіки програми.

Після реалізації модуля необхідно створити основну програму, яка імпортує цей модуль і використовує його функції для автоматизації поставленої задачі. У головній програмі слід організувати введення початкових даних, виклик функцій модуля та виведення результатів. Важливо перевірити, наскільки зручно використовувати створений модуль у реальній прикладній ситуації.

У процесі виконання роботи доцільно підготувати декілька тестових прикладів. Слід перевірити роботу модуля на коректних вхідних даних, граничних значеннях і помилкових ситуаціях. Це дозволить оцінити правильність реалізації функцій та стійкість модуля до некоректних вхідних даних.

На завершальному етапі необхідно проаналізувати результати роботи. Слід оцінити, чи вдалося скоротити обсяг основної програми за рахунок винесення функціональності у модуль, наскільки зручно використовувати розроблений компонент повторно та чи можна розширити його додатковими функціями у майбутньому.

Контрольні питання

1. Що таке програмний модуль?
2. Яке призначення модуля у прикладній програмі?
3. Які переваги дає модульна організація коду?
4. Чому автоматизація задач є важливою у програмуванні?
5. Які вимоги ставляться до структури програмного модуля?
6. Як організовується передавання даних між основною програмою і модулем?
7. Чому важливо перевіряти коректність вхідних даних у функціях модуля?
8. Як забезпечити повторне використання програмного модуля?
9. Які помилки можуть виникати під час розробки модуля для автоматизації?
10. Яке значення має модульний підхід у програмній інженерії?

Зміст звіту

– назва практичної роботи;
– мета виконання;
– опис задачі, що автоматизується;
– опис структури програмного модуля;
– опис реалізованих функцій;
– приклади використання модуля в основній програмі;
– результати тестування;
– висновки.

Практична робота 19. Використання бібліотек та API у прикладних програмах

Мета роботи – ознайомитися з принципами використання програмних бібліотек та прикладних програмних інтерфейсів, навчитися інтегрувати зовнішні сервіси у програму та реалізовувати обробку даних, отриманих через API.

Ключові положення

Сучасні прикладні програми часто взаємодіють із зовнішніми сервісами, використовуючи прикладні програмні інтерфейси. API є набором правил і методів, що дозволяють програмам обмінюватися даними між собою. Завдяки цьому можна отримувати доступ до віддалених ресурсів, обробляти інформацію та інтегрувати різні програмні системи.

У скриптових мовах програмування для роботи з API зазвичай використовуються спеціальні бібліотеки, що забезпечують формування запитів, отримання відповідей та обробку даних. Найчастіше взаємодія з API здійснюється через HTTP-запити, зокрема методи GET, POST, PUT та DELETE.

Відповіді від API зазвичай передаються у форматі JSON або XML. Програма повинна виконувати розбір отриманих даних, перетворювати їх у внутрішні структури та використовувати для подальшої обробки або відображення результатів.

Важливим є правильне формування запитів до API. Це включає визначення адреси ресурсу, передачу параметрів, заголовків та, за потреби, автентифікаційних даних. Помилки у формуванні запиту можуть призвести до відмови у доступі або отримання некоректних даних.

Під час роботи з API необхідно враховувати можливі помилки, зокрема відсутність з’єднання, перевищення обмежень запитів або отримання некоректної відповіді від сервера. Програма повинна передбачати обробку таких ситуацій і коректно реагувати на них.

Використання бібліотек спрощує роботу з API, оскільки вони надають готові засоби для формування запитів і обробки відповідей. Це дозволяє зосередитися на логіці прикладної програми, не реалізуючи низькорівневі механізми взаємодії.

Таким чином, використання бібліотек та API є важливим напрямом сучасного програмування, що дозволяє створювати інтегровані програмні системи та отримувати доступ до зовнішніх джерел даних.

Практичне завдання

1. Підключити бібліотеку для виконання HTTP-запитів.
2. Реалізувати GET-запит до відкритого API.
3. Отримати та вивести відповідь сервера у програмі.
4. Виконати розбір отриманих даних у форматі JSON.
5. Використати параметри запиту для отримання різних даних.
6. Реалізувати обробку помилок під час виконання запитів.
7. Створити програму, що використовує API для отримання та обробки даних.
8. Проаналізувати результати роботи програми.

Методичні вказівки до виконання практичного завдання

Перед початком виконання роботи необхідно ознайомитися з поняттям API та принципами взаємодії програм із зовнішніми сервісами. Особливу увагу слід приділити структурі HTTP-запитів і форматам передачі даних.

На першому етапі необхідно підключити бібліотеку, що забезпечує виконання HTTP-запитів. Після цього слід реалізувати простий GET-запит до відкритого API. Для цього потрібно вказати адресу ресурсу та виконати запит із програми.

Далі необхідно отримати відповідь сервера та вивести її у консоль. Це дозволить перевірити правильність виконання запиту та доступ до API.

На наступному етапі необхідно виконати розбір отриманих даних. Якщо відповідь має формат JSON, слід перетворити її у внутрішню структуру даних і отримати окремі значення.

Після цього необхідно реалізувати передачу параметрів у запиті. Наприклад, можна змінювати параметри для отримання різних даних від сервера.

Особливу увагу слід приділити обробці помилок. Необхідно передбачити ситуації, коли сервер недоступний, повертає помилку або дані мають некоректний формат.

На завершальному етапі необхідно створити приклад прикладної програми, що використовує API. Наприклад, можна отримати дані з відкритого сервісу та виконати їх обробку або відображення.

У процесі виконання роботи необхідно проаналізувати результати та оцінити ефективність використання бібліотек і API для реалізації прикладних задач.

Контрольні питання

1. Що таке API?
2. Яке призначення API у програмуванні?
3. Які методи HTTP-запитів використовуються для роботи з API?
4. У якому форматі зазвичай передаються дані через API?
5. Як здійснюється розбір JSON-даних?
6. Яке призначення параметрів запиту?
7. Які помилки можуть виникати при роботі з API?
8. Чому доцільно використовувати бібліотеки для роботи з API?
9. Як перевірити правильність отриманих даних?
10. Яке значення має використання API у прикладних програмах?

Зміст звіту

– назва практичної роботи;
– мета виконання;
– опис використаного API;
– опис реалізації запитів;
– приклади отриманих даних;
– результати тестування;
– висновки.

Практична робота 20. Створення скриптів для автоматизації обробки даних

Мета роботи – закріпити навички розробки скриптів для автоматизації обробки даних, навчитися створювати програми, що виконують послідовність операцій без участі користувача, та реалізовувати повний цикл обробки інформації.

Ключові положення

Автоматизація обробки даних є одним із основних напрямів використання скриптових мов програмування. Скрипти дозволяють виконувати повторювані операції без необхідності ручного втручання, що значно підвищує ефективність роботи та зменшує ймовірність помилок.

Скрипт є програмою, що зазвичай виконує певну послідовність дій: отримання даних, їх обробку, аналіз та збереження результатів. На відміну від інтерактивних програм, скрипти часто працюють у напівавтоматичному або повністю автоматичному режимі.

Основою автоматизації є використання базових конструкцій програмування: умовних операторів, циклів, функцій та модулів. Вони дозволяють реалізувати логіку обробки даних та організувати виконання задач у потрібній послідовності.

Важливим є визначення джерела даних. Дані можуть надходити з файлів, баз даних, мережевих ресурсів або генеруватися у програмі. Вибір джерела визначає спосіб їх обробки та структуру скрипта.

У процесі автоматизації необхідно передбачати перевірку коректності даних і обробку помилок. Це забезпечує стабільну роботу скрипта навіть у разі виникнення нестандартних ситуацій.

Результати роботи скрипта можуть зберігатися у файли, виводитися у консоль або передаватися іншим програмам. Це дозволяє інтегрувати скрипти у більші програмні системи.

Таким чином, створення скриптів для автоматизації обробки даних є важливою практичною навичкою, що дозволяє ефективно розв’язувати прикладні задачі та оптимізувати робочі процеси.

Практичне завдання

1. Визначити задачу для автоматизації обробки даних.
2. Реалізувати скрипт для зчитування даних із файлу або іншого джерела.
3. Виконати обробку даних із використанням колекцій.
4. Реалізувати перетворення та аналіз даних.
5. Використати умовні оператори та цикли для керування виконанням скрипта.
6. Реалізувати обробку помилок під час виконання.
7. Зберегти результати обробки у файл або вивести їх у консоль.
8. Проаналізувати ефективність роботи скрипта.

Методичні вказівки до виконання практичного завдання

Перед початком виконання роботи необхідно визначити конкретну задачу, яку доцільно автоматизувати. Це може бути обробка текстових файлів, аналіз числових даних, формування звітів або інші типові задачі, що потребують багаторазового виконання.

На першому етапі необхідно реалізувати отримання даних. Для цього можна використати зчитування з файлу або інше джерело інформації. Дані слід перевірити на коректність перед початком обробки.

Далі необхідно реалізувати обробку даних. Для цього слід використати відповідні структури даних, наприклад списки або словники, а також цикли для обробки кожного елемента.

На наступному етапі необхідно реалізувати перетворення даних. Наприклад, можна змінити формат даних, виконати обчислення або відфільтрувати значення за певними умовами.

Після цього необхідно реалізувати керування виконанням скрипта. Для цього слід використати умовні оператори та інші конструкції, що визначають логіку обробки.

Особливу увагу слід приділити обробці помилок. Скрипт повинен коректно реагувати на некоректні дані, відсутність файлів або інші проблеми, що можуть виникати під час виконання.

На завершальному етапі необхідно реалізувати збереження результатів. Це може бути запис у файл або виведення у консоль.

У процесі виконання роботи необхідно протестувати скрипт на різних наборах даних і оцінити його ефективність та надійність.

Контрольні питання

1. Що таке скрипт у програмуванні?
2. Яке призначення скриптів для автоматизації?
3. Які етапи містить процес обробки даних у скрипті?
4. Які конструкції програмування використовуються у скриптах?
5. Які джерела даних можуть використовуватися у скриптах?
6. Чому важливо перевіряти коректність даних?
7. Як реалізується обробка помилок у скриптах?
8. Які способи збереження результатів існують?
9. Які переваги дає автоматизація обробки даних?
10. Які помилки можуть виникати під час розробки скриптів?

Зміст звіту

– назва практичної роботи;
– мета виконання;
– опис задачі автоматизації;
– опис реалізованого скрипта;
– приклади вхідних і вихідних даних;
– результати тестування;
– висновки.

Рекомендована література

Основна

1. Педяш В. В., Йона Л. Г., Мазур Г. Д., Русу О. П. Python-програмування: методичні рекомендації для практичних та лабораторних занять [Електронне видання] / кафедра комп'ютерної інженерії та інноваційних технологій Міжнародного гуманітарного університету. Одеса, 2025. 137 с.

2. Костюченко А. О. Основи програмування мовою Python: навчальний посібник. Черкаси: ФОП Баликіна С. М., 2020. 180 с.

3. Яковенко А. В. Основи програмування. Python. Частина 1: підручник для студентів спеціальності 122 Комп'ютерні науки, спеціалізації Інформаційні технології в біології та медицині. Київ: КПІ ім. Ігоря Сікорського, 2018. 195 с.

4. Маттес Е. Пришвидшений курс Python. Київ: Видавництво Старого Лева, 2021. 600 с.

5. Васильєв О. Програмування мовою Python. Тернопіль: Навчальна книга – Богдан, 2019. 204 с.

6. Ceder N. The Quick Python Book. 3rd ed. New York: Manning Publications, 2018. 432 p.

7. Lambert K. A. Fundamentals of Python: First Programs. Boston: Cengage Learning, 2018. 476 p.

8. Lutz M. Learning Python. 5th ed. Sebastopol: O'Reilly Media, 2019. 648 p.

9. Стрелковська І. В., Григор'єва Т. І., Педяш В. В., Русу О. П., Мельник В. В. Використання Python програмування у графових моделях безпеки баз даних. Наука і техніка сьогодні. 2026. № 1 (55). С. 2570–2580. DOI: 10.52058/2786-6025-2026-1(55)-2570-2580.

Допоміжна

10. Fenner M. Machine Learning with Python for Everyone. Addison-Wesley Data & Analytics Series. Boston: Addison-Wesley Professional, 2019. 586 p.

11. Григор'єва Т., Русу О., Горбачов В., Крохмаль М. Багатовимірний підхід в задачах розробки програмного забезпечення. Вісник Хмельницького національного університету. Серія: Технічні науки. 2025. Т. 359, № 6.2. С. 157–160. DOI: https://doi.org/10.31891/2307-5732-2025-359-92

.

Інформаційні ресурси

12. Бібліотека Міжнародного гуманітарного університету: веб-сайт. URL: https://mu.od.ua/naukova-biblioteka

13. Національна бібліотека України ім. В. І. Вернадського: веб-сайт. URL: http://www.nbuv.gov.ua

Навчальне видання

Педяш Володимир Віталійович

СКРИПТОВІ МОВИ ПРОГРАМУВАННЯ

Методичні вказівки до практичних робіт для здобувачів вищої освіти,
які навчаються за спеціальностями галузі «Інформаційні технології»

Українською мовою