**1. Основи теорії алгоритмів.**

Алгоритми є ключовим поняттям у світі програмування та інформаційних технологій загалом. Вони представляють собою послідовність дій, які виконуються для розв'язання конкретної задачі. Розуміння алгоритмів та їх властивостей є важливим елементом будь-якого програміста.

*Алгоритм* - це набір чітких та конкретних кроків, які дозволяють вирішити певну задачу. Ідея полягає в тому, щоб розбити складну задачу на простіші етапи, які можна виконати послідовно. Алгоритми можуть бути знаходити застосування в різних областях, від обробки даних до штучного інтелекту.

Основні властивості алгоритмів

При проектуванні та аналізі алгоритмів важливо розуміти їх характеристики. Ось деякі ключові властивості алгоритмів:

Визначеність: Кожен крок алгоритму повинен бути чітко визначеним та несуперечним. Відсутність двозначності допомагає уникнути непорозумінь при виконанні алгоритму.

Дискретність: Алгоритм повинен складатися з обмеженого числа окремих кроків. Це означає, що кожен крок може бути виконаний за обмежений час та не вимагає "нескінченності".

Завершеність: Алгоритм повинен завершити свою роботу за скінченну кількість кроків. Це означає, що він вирішує задачу незалежно від вхідних даних та не залишає завдання невирішеним.

Коректність: Алгоритм повинен вирішувати задачу правильно для всіх можливих вхідних даних. Це означає, що він має відповідати специфікації та не виконувати зайвих або неправильних дій.

Практична задача 1: Знаходження суми перших N натуральних чисел

Для закріплення розуміння основ алгоритмів ми розглянемо практичну задачу - написання алгоритму для знаходження суми перших N натуральних чисел.

Припустимо, нам потрібно знайти суму перших N натуральних чисел. Наприклад, якщо N = 5, ми маємо знайти суму: 1 + 2 + 3 + 4 + 5 = 15.

Ми можемо використовувати цикл для обчислення суми. Ось приклад алгоритму на мові Python:

def sum\_of\_natural\_numbers(n):

sum = 0

for i in range(1, n+1):

sum += i

return sum

# Зчитування значення N від користувача

n = int(input("Введіть значення N: "))

result = sum\_of\_natural\_numbers(n)

print(f"Сума перших {n} натуральних чисел: {result}")

Цей приклад ілюструє використання послідовності дій для вирішення конкретної задачі - знаходження суми натуральних чисел.

Практична задача 2: Обчислення факторіала числа

Завдання: Напишіть алгоритм для обчислення факторіала числа N (позитивного цілого числа). Факторіал числа N позначається як N! і визначається як добуток всіх натуральних чисел від 1 до N.

def factorial(n):

if n == 0 or n == 1:

return 1

else:

return n \* factorial(n - 1)

# Зчитування значення N від користувача

n = int(input("Введіть ціле невід'ємне число N: "))

result = factorial(n)

print(f"Факторіал числа {n} дорівнює {result}")

У цьому прикладі використовується рекурсія для обчислення факторіала числа.

Практична задача 3: Визначення простого числа

Завдання: Напишіть алгоритм, який перевіряє, чи є задане число N простим (має тільки два дільники - 1 і самого себе).

def is\_prime(n):

if n <= 1:

return False

for i in range(2, int(n \*\* 0.5) + 1):

if n % i == 0:

return False

return True

# Зчитування значення N від користувача

n = int(input("Введіть ціле число N: "))

if is\_prime(n):

print(f"{n} є простим числом.")

else:

print(f"{n} не є простим числом.")

У цьому прикладі використовується цикл for для перевірки дільників числа та оператор % для перевірки дільності.

Ці практичні задачі допоможуть учасникам засвоїти більше понять про алгоритми та використання різних конструкцій мови Python для вирішення різних типів завдань.

**2. Поняття відлагодження та тестування програмних засобів.**

Відлагодження (debugging) - це важлива частина процесу розробки програм. Це процес знаходження та виправлення помилок у вашому коді. Відлагодження допомагає вам забезпечити, що ваша програма працює правильно та відповідає очікуванням.

Інструменти відлагодження в VS Code

VS Code надає потужні інструменти для відлагодження вашого коду. Ось деякі з них:

Точки зупинки (Breakpoints): Ви можете встановлювати точки зупинки у своєму коді, щоб програма зупинялася під час їх досягнення, дозволяючи вам перевірити стан змінних.

Режими відлагодження: VS Code має різні режими відлагодження, такі як крок за кроком (Step Over), виклик функції (Step Into) та вихід із функції (Step Out), що допомагають керувати виконанням програми.

Вікно "Перегляд змінних" (Watch): Ви можете стежити за значеннями змінних у реальному часі під час відлагодження.

Практична задача 1: Відлагодження обчислення факторіала

Візьмемо код з попередньої практичної задачі "Обчислення факторіала числа" та відлагодимо його в VS Code:

Встановіть точку зупинки на початку функції factorial.

Запустіть виконання програми в режимі відлагодження.

Крок за кроком виконайте код, спостерігаючи за значеннями змінних.

Практична задача 2: Відлагодження пошуку простих чисел

Візьмемо код з попередньої практичної задачі "Визначення простого числа" та відлагодимо його в VS Code:

Встановіть точку зупинки на початку функції is\_prime.

Запустіть виконання програми в режимі відлагодження.

Використовуйте режим "Step Into" для входу в цикл та крок за кроком перевіряйте значення змінних.

Практична задача 3: Відлагодження алгоритму сортування бульбашкою

Завдання: Напишіть алгоритм сортування бульбашкою для списку цілих чисел та відлагодьте його в VS Code:

def bubble\_sort(arr):

n = len(arr)

for i in range(n):

for j in range(0, n-i-1):

if arr[j] > arr[j+1]:

arr[j], arr[j+1] = arr[j+1], arr[j]

# Тестовий список для сортування

numbers = [64, 34, 25, 12, 22, 11, 90]

bubble\_sort(numbers)

print("Відсортований список:", numbers)

Встановіть точку зупинки на початку функції bubble\_sort.

Запустіть виконання програми в режимі відлагодження.

Спостерігайте, як елементи списку обмінюються та виконуйте кроки в алгоритмі.

Тестування програм є необхідною складовою розробки програмного забезпечення. Це процес перевірки роботи програми з метою виявлення помилок та неполадок. Правильне тестування допомагає підтвердити коректність програми та забезпечити стабільність та надійність в роботі.

Види тестування

Ручне тестування: Тестування вручну шляхом введення різних значень та спостереження за результатами.

Автоматичне тестування: Використання автоматизованих тестів для перевірки програми. Відбувається автоматично і допомагає виявити помилки при змінах у коді.

Модульне тестування з unittest

Python надає бібліотеку unittest, яка дозволяє писати модульні тести для перевірки функцій вашої програми.

python

Copy code

import unittest

def add(a, b):

return a + b

class TestAddFunction(unittest.TestCase):

def test\_positive\_numbers(self):

self.assertEqual(add(2, 3), 5)

def test\_negative\_numbers(self):

self.assertEqual(add(-2, -3), -5)

if \_\_name\_\_ == '\_\_main\_\_':

unittest.main()

Практична задача 1: Модульні тести для факторіала числа

Потрібно написати модульні тести для функції обчислення факторіала числа з попередньої практичної задачі.

Практична задача 2: Модульні тести для визначення простого числа

Напишіть модульні тести для функції визначення простого числа з попередньої практичної задачі.

Практична задача 3: Модульні тести для алгоритму сортування бульбашкою

Напишіть модульні тести для функції сортування бульбашкою з попередньої практичної задачі.

Використання Git та GitHub для Розробки Програмних Засобів

Git є розподіленим системою керування версіями, яка дозволяє вам зберігати та керувати змінами в вашому коді. GitHub - це платформа для спільної розробки, де ви можете спільно працювати над проектами, зберігати репозиторії та використовувати інші корисні інструменти.

Основні Поняття Git

Репозиторій (Repository): Місце, де зберігається ваш код та всі зміни до нього.

Коміт (Commit): Зміни в коді, які ви зберігаєте. Коміт включає опис та вказує, які файли були змінені.

Гілка (Branch): Відгалуження від основної лінії розробки, яке дозволяє вам розробляти функції чи внесення змін без впливу на основний код.

Злиття (Merge): Об'єднання змін з однієї гілки в іншу, наприклад, об'єднання роботи з функціями у головний код.

Робота з Git та GitHub

Ініціалізація Репозиторію:

Ініціалізуйте новий репозиторій за допомогою команди git init у терміналі.

Створення та Використання Гілок:

Створіть нову гілку за допомогою git branch <назва\_гілки>.

Перейдіть на гілку командою git checkout <назва\_гілки>.

Внесіть зміни, створюючи коміти у новій гілці.

Злиття Гілок:

Перейдіть на гілку, до якої хочете злити зміни: git checkout <цільова\_гілка>.

Виконайте злиття за допомогою git merge <гілка\_джерело>.

Використання GitHub:

Створіть обліковий запис на GitHub та створіть новий репозиторій.

Додайте віддалений репозиторій до вашого локального репозиторію: git remote add origin <URL\_вашого\_репозиторію>.

Завантажте зміни на GitHub: git push origin <назва\_гілки>.

Робота з Віддаленими Змінами:

Отримайте зміни з віддаленого репозиторію: git pull origin <назва\_гілки>.

Практична Задача 1: Створення та Злиття Гілок

Створіть новий репозиторій на GitHub.

Склонуйте репозиторій на свій комп'ютер: git clone <URL\_вашого\_репозиторію>.

Створіть нову гілку для розробки функції: git branch feature.

Перейдіть на нову гілку: git checkout feature.

Внесіть зміни у коді та збережіть їх комітом.

Перейдіть на головну гілку: git checkout main.

Зліть зміни з гілки feature у головну: git merge feature.

Завантажте зміни на GitHub: git push origin main.

Практична Задача 2: Робота з Віддаленими Змінами

Отримайте зміни з віддаленого репозиторію: git pull origin main.

Внесіть зміни у коді та збережіть їх комітом.

Відправте зміни на GitHub: git push origin main.

Git та GitHub допомагає забезпечити ефективне керування версіями вашого коду та спрощує спільну розробку програмних засобів. Розуміння основних понять та операцій Git допомагає вам стати більш впевненим розробником та співпрацювати з іншими колегами.

Використання Fork на GitHub

Функція "fork" на GitHub дозволяє вам створити копію чужого репозиторію на вашому обліковому записі. Це корисний інструмент для спільної роботи над відкритими проектами, внесення змін та внесення пропозицій зі змінами.

Кроки для Використання Fork

Зайдіть на веб-сайт GitHub та перейдіть до репозиторію, який ви хочете форкнути.

Натисніть кнопку "Fork" у верхньому правому куті сторінки. Це створить копію репозиторію на вашому обліковому записі.

Після форкання ви будете перенаправлені до вашої копії репозиторію. Тепер ви можете вносити зміни до цього репозиторію.

Внесення Змін та Відправлення Pull Request

Склонуйте свою копію репозиторію на свій комп'ютер за допомогою команди git clone <URL\_вашого\_репозиторію>.

Внесіть зміни у коді та збережіть їх комітом.

Відправте зміни на GitHub за допомогою команди git push origin <назва\_гілки>.

Перейдіть до своєї копії репозиторію на GitHub та натисніть кнопку "New Pull Request". Виберіть базову та порівняльну гілки.

Заповніть деталі для свого pull request, додайте опис змін, які ви внесли.

Натисніть кнопку "Create Pull Request" для надсилання запиту на злиття змін до вихідного репозиторію.

Синхронізація з Вихідним Репозиторієм

Якщо оригінальний репозиторій також розвивається, ви можете синхронізувати свою копію з останніми змінами:

Додайте віддалений репозиторій оригінального репозиторію за допомогою команди git remote add upstream <URL\_оригінального\_репозиторію>.

Отримайте останні зміни з оригінального репозиторію: git pull upstream <назва\_гілки>.

Ваша копія репозиторію тепер міститиме останні зміни з оригінального репозиторію.

Використання функції "fork" на GitHub дозволяє легко співпрацювати над відкритими проектами та вносити зміни до репозиторіїв інших розробників. Це потужний інструмент для спільної роботи та обміну ідеями у відкритих програмних проектах.