Варварівська гімназія Славутської міської ради

Збірник завдань та практичних робіт по темі "Алгоритми та програми" з мов програмування Scratch та Python у 5-9 класах

номінація **Інформатика**
Рибчинська Дар'я Вадиміна
вчитель інформатики Варварівської
гімназії Славутської міської ради

Варварівка – 2024р.

Електронний збірник **«Збірник завдань та практичних робіт по темі "Алгоритми та програми" з мов програмування Scratch та Python у 5-9 класах»,** - Варварівка, 2024р.

Автор-упорядник:
Рибчинська Дар'я Вадимівна, вчитель інформатики Варварівської гімназії Славутської міської ради.

Схвалено для використання в освітньому процесі на засіданнях педагогічної ради Варварівської гімназії Славутської ради, Шепетівського райо, Хмельницької обл. (протокол №4 від 27.01.2024р.)

Даний збірник містить електронні практичні та контрольні завдання по темі "Алгоритми та програми" у 5-9 класах, що дає вчителю можливість перевірити знання здобувачів освіти для ефективної організації навчання.

Посібник адресовано вчителям інформатики

ВСТУП

У сучасному світі, де технології швидко розвиваються, вміння програмування стає ключовим елементом освіти, навіть для учнів молодшого та середнього шкільного віку. Програмування не лише розвиває логічне мислення і креативність, але й підготовлює до майбутніх викликів, що стосуються розвитку технологій і цифрової грамотності.

Завдання на розділ "Алгоритми та програми" для 5-9 класів можуть бути різноманітними і включати в себе розуміння основних концепцій алгоритмів, використання блок-схем для представлення алгоритмів, основи програмування за допомогою візуальних інструментів (наприклад, Scratch), а також практичні завдання на розвиток логічного мислення.

Вивчення Scratch та Python в 5-9 класах відкриває двері до нових можливостей у навчанні, особистому розвитку та майбутній кар'єрі учнів, підготовлюючи їх до активної участі у світі, де технології стають необхідністю.

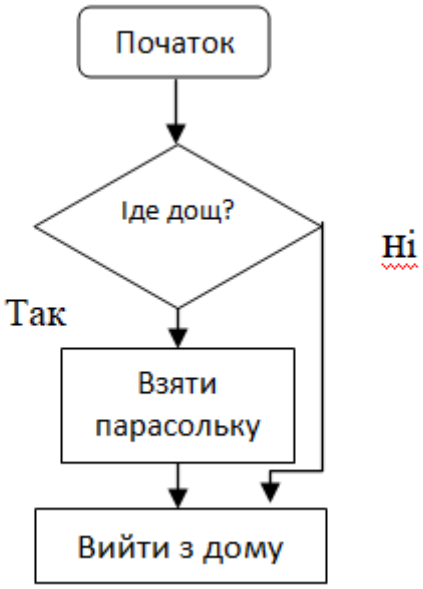
1. Завдання та практичні роботи по темі "Алгоритми та програми" з мови програмування Scratch для 5 класу.

1. Розуміння алгоритмів

Завдання: Напиши алгоритм приготування улюбленої випічки (за потреби можливе використання інтернету). Використовуй прості команди, як-от рецепт смачних мафінів: Ретельно змішайте сухі інгредієнти, це - борошно пшеничне (2,5 склянки), розпушувач (2 ч. л. без гірки), сіль (на кінчику ножа). Не забудьте просіяти борошно з розпушувачем, тощо.

2. Використання блок-схем

Завдання: Намалюй блок-схему для алгоритму, який описує, як вирішити,чи потрібно брати парасольку, виходячи з дому. Алгоритм має включати перевірку погоди (ясно, хмарно, дощ) і рішення, засноване на цій інформації.

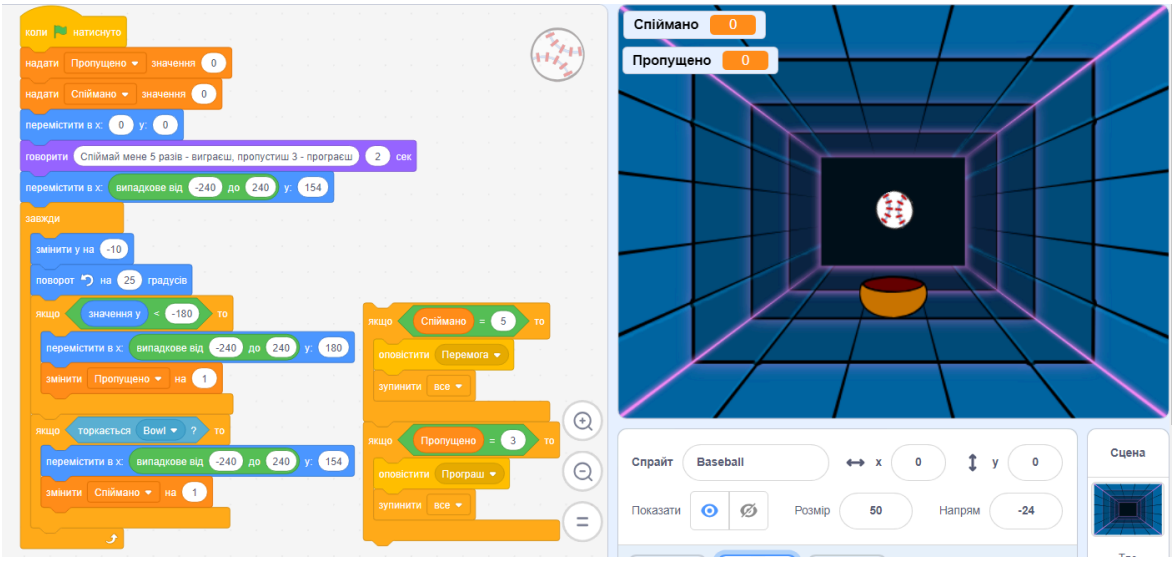


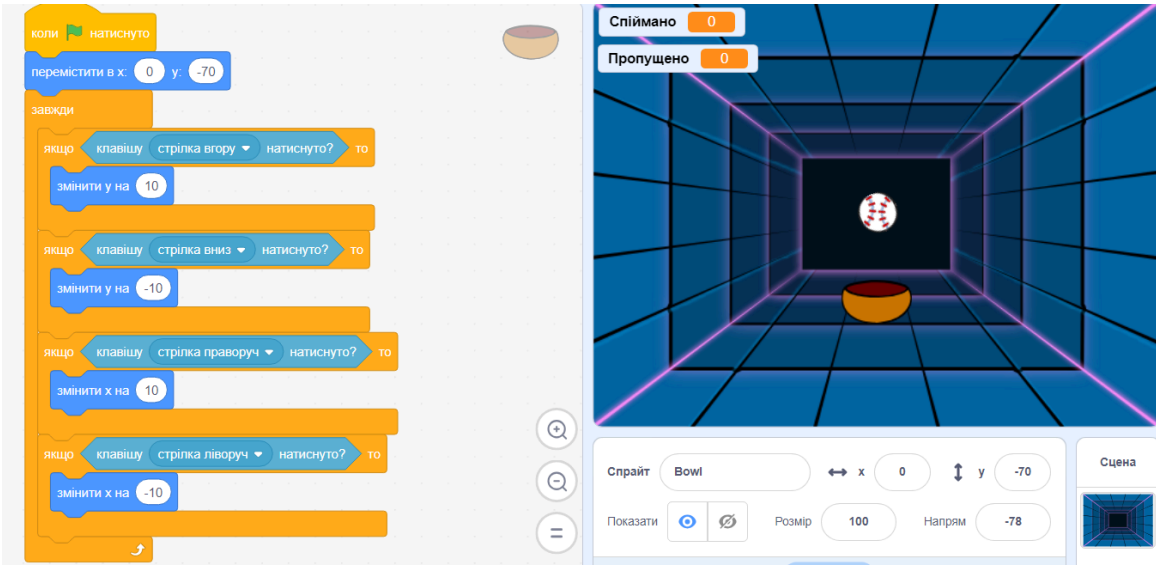
3. Блок-схема процесу

Завдання: Створи блок-схему алгоритму, який визначає, чи є число парним або непарним. Згадай, що для цього потрібно поділити число на 2 та перевірити залишок від ділення.

4. Вступ до програмування з використанням Scratch

Завдання: Створи просту гру або анімацію в Scratch. Наприклад, можна зробити гру, де котик (або будь-який інший спрайт) повинен збирати предмети, що падають зверху. Використовуй цикли, умовні оператори, та змінні для рахунку очок.





5. Прості алгоритмічні задачі

Завдання: Напиши програму на Scratch, яка виводить числа від 1 до 10 та їх квадрати. Використовуй цикл для повторення дій.

6. Програмування на Scratch

Завдання: Створи проєкт на Scratch, де спрайт (персонаж) розмовляє з користувачем, запитуючи його ім'я та вітаючись з ним по імені. Додай можливість спрайту також запитати, як справи у користувача, та відповісти на його відповідь позитивним коментарем.

7. Логічна задача

Завдання: У класі 28 учнів. Серед них, 16 вважають математику улюбленим предметом, а 19 – англійську мову. Як мінімум, скільки учнів можуть вважати улюбленими обидва цих предмети? Спробуй розв'язати цю задачу логічно.

8. Розвиток алгоритмічного мислення

Завдання: Напиши алгоритм, який описує процес вибору одягу в залежності від погоди за вікном. Твій алгоритм має включати перевірку температури (холодно, тепло, жарко) та опадів (іде дощ, сніг, сухо) і на основі

цього пропонувати, що одягнути (шорти, джинси, светр, куртку, парасольку тощо).

9. Гра "Вибір відповідей":

Завдання: Створити гру, де гравець вибирає одну з кількох відповідей на запитання.

Практична робота: Учні можуть створити різні сценарії з питаннями та варіантами відповідей. За допомогою умовних блоків, їм потрібно створити логіку, яка реагує на вибір гравця та визначає, чи є відповідь правильною.

10. Гра "Постріл на координати":

Завдання: Створити гру, де гравець керує персонажем, який стріляє в ціль на заданих координатах.

Практична робота: Учні можуть створити персонажа, який може стріляти у вказані координати на екрані. Вони можуть створити ціль, яка з'являється на випадкових або заздалегідь визначених координатах. Потім за допомогою блоків керування, їм потрібно створити логіку для визначення, чи попав гравець у ціль чи ні.

11.Гра "Клавіатурний тренажер":

Завдання: Створити гру, де гравець повинен натискати вірні клавіші відповідно до показів на екрані.

Практична робота: Учні можуть створити гру, де з'являються зображення клавіш на екрані, і гравець повинен натискати відповідні клавіші на клавіатурі. За допомогою блоків керування, вони можуть визначити, які клавіші повинен натискати гравець в різних сценаріях.

12. Гра «Пам'ять»:

Завдання: Створити гру, в якій гравець повинен відкривати парні картки на полі пам'яті.

Практична робота: Учні можуть створити поле з картками, які мають бути відкриті парами. Вони можуть застосувати блоки умов та змінні, щоб відслідковувати відкриті та закриті картки, а також визначати, чи є вони парою чи ні.

2. Завдання та практичні роботи по темі "Алгоритми та програми" з мови програмування Scratch для 6 класу.

1. Пригодницька гра "Втрачений ключ"

Опис:

Створіть пригодницьку гру, де гравець відправляється на пошук втраченого ключа в таємничому лабіринті. Гравець повинен уникати перешкод, збирати підказки та знайти ключ, щоб відкрити вихід.

Розв'язок:

1) Створення лабіринту:
   - Створіть фонове зображення лабіринту або використайте малюнок лабіринту.
   - Розмістіть гравця та вихід на початковій позиції лабіринту.
2) Створення гравця:
   - Створіть спрайт для гравця (наприклад, героя або дослідника) і помістіть його на початкову позицію лабіринту.
   - Додайте блоки коду для керування гравцем за допомогою стрілок клавіатури або клавіш WASD.
3) Додавання перешкод:
   - Створіть перешкоди на лабіринті, такі як стіни або водоймища, які гравець повинен уникати.
   - Додайте блоки коду для перевірки зіткнень гравця з перешкодами. Якщо гравець зіткнеться з перешкодою, то гра закінчується.
4) Додавання ключа та виходу:
   - Створіть спрайт для ключа та виходу та помістіть їх на різних місцях лабіринту.
   - Додайте блоки коду, які перевіряють, чи зіткнеться гравець з ключем. Якщо так, то відкривається вихід.
5) Додавання підказок:
   - Створіть спрайти для підказок, які розташовані в різних частинах лабіринту.
   - Додайте блоки коду, щоб показувати підказки гравцю при зіткненні з ними.
6) Завершення гри:
   - Додайте блоки коду для перевірки, чи досяг гравець виходу після збору ключа. Якщо так, то гравець перемагає і гра закінчується.
7) Додаткові можливості:
   - Додайте різні рівні складності лабіринту.
   - Реалізуйте можливість вибору персонажа для гравця.
   - Додайте звукові ефекти та анімації для покращення геймплею.

2. Гра «Різдвяний ельф»:

Завдання: Створити гру, де різдвяний ельф збирає подарунки, уникаючи перешкод.

Практична робота: Учні можуть створити головного героя (ельфа), який рухається по екрану, щоб зібрати подарунки. Вони можуть додати перешкоди,

такі як дерева або сніговики, які ельф повинен уникати. Логіка руху та збирання подарунків може бути реалізована за допомогою блоків керування.

1) Створення спрайта ельфа:

- Створіть новий спрайт і назвіть його «Ельф».
- Завантажте або намалюйте зображення ельфа.

2) Додавання коду для руху ельфа:

- Додайте блок коду «Коли зелена стрілка натиснута» і розмістіть в ньому блоки «Перемістити Ельфа вперед на 10 кроків».
- Повторіть цей процес для стрілок вліво, вправо і вниз, змінюючи напрямок руху відповідно.

3) Додавання подарунків:

- Створіть новий спрайт або використайте готовий для подарунка.
- Додайте блок коду «Коли зелена стрілка натиснута» для переміщення подарунка на випадкову позицію на екрані.

4) Перевірка зіткнення ельфа з подарунками:

- В спрайті ельфа додайте блок коду «Коли Ельф доторкнеться до подарунка», щоб зробити його невидимим та збільшити рахунок на одиницю.

5) Додавання перешкод:

- Створіть новий спрайт або використайте готовий для перешкод (наприклад, дерево).
- Додайте блок коду «Коли зелена стрілка натиснута» для переміщення перешкоди на випадкову позицію на екрані.

6) Перевірка зіткнення ельфа з перешкодами:

- В спрайті ельфа додайте блок коду «Коли Ельф доторкнеться до перешкоди», щоб зробити його невидимим та завершити ГРУ.

7) Додавання лічильника подарунків:

- Створіть змінну «Рахунок» і встановіть її значення на початку гри на 0.
- При зіткненні ельфа з подарунком збільшуйте значення цієї змінної на 1.

8) Відображення рахунку на екрані:

- Додайте блок коду «Показати (значення Рахунок)» для відображення рахунку на екрані гри.

3. Гра «Лабіринт»

1) Підготовка середовища. Створення нового проекту:
   - Відкрийте Scratch та створіть новий проект.
   - Видаліть спрайт кота, що йде за замовчуванням, натиснувши на нього правою кнопкою миші та вибравши "видалити".
2) Додавання та налаштування спрайтів:
   - Додайте спрайт, який буде персонажем у лабіринті. Це може бути будь-який спрайт на ваш вибір.
   - Додайте спрайт, який використовуватиметься як стіни лабіринту. Це може бути простий прямокутник або більш складний образ.
3) Створення лабіринту. Малювання лабіринту:

- Використовуючи інструменти малювання у розділі спрайтів, створіть кілька стін лабіринту.
- Ви можете скопіювати спрайт стіни кілька разів, щоб утворити різні частини лабіринту.

4) Програмування персонажа. Рух персонажа:
   - Додайте наступні блоки до спрайту персонажа:
     - Коли клікнуто на зелений прапорець, установіть початкові координати персонажа.
     - Додайте блоки управління рухом за допомогою клавіш (наприклад, стрілок).
     - Зробіть так, щоб персонаж рухався вперед, коли натиснута клавіша і зупинявся, коли він зіткнеться зі стіною.

5) Додавання логіки зіткнення:
   - Додайте умову, що якщо спрайт персонажа торкається спрайта стіни, персонаж має зупинитися або відскочити назад.

6) Тестуйте гру:
   - Запустіть гру, натиснувши на зелений прапорець, і перевірте, чи правильно працює рух персонажа.
   - Переконайтеся, що персонаж не проходить крізь стіни та зупиняється або відскакує при зіткненні.

7) Налагодження:
   - Якщо виникають проблеми з рухом або колізіями, зверніть увагу на координати та умови блоків.
   - Виправляйте помилки, модифікуючи або замінюючи блоки коду.

8) Поліпшення гри:
   - Додайте фінішну точку або цілі.
   - Включіть таймер або рахунок.
   - Прикрасьте ігровий інтерфейс, додавши фон та музику.

4. Гра «Хрестики-нулики»

1) Створення нового проекту:
   - Відкрийте Scratch і створіть новий проект.
   - Видаліть стандартний спрайт кота.

2) Створення ігрового поля:
   - Намалюйте спрайт, який виглядатиме як ігрове поле для «Хрестики-нулики». Це може бути простий квадрат з розділеними на дев'ять рівних частин лініями.

3) Створення спрайтів "X" та "O":
   - Створіть два окремі спрайти: один для "X" та один для "O".
   - Зробіть ці спрайти клікабельними, щоб гравці могли розміщувати їх на полі.

4) Вибір гравця:
   - Використайте змінну (наприклад, «Черга»), щоб визначити, чий хід зараз (X або O).
   - Змінюйте цю змінну після кожного ходу.

5) Розміщення "X" та "O":
- Програмуйте спрайти "X" та "O" таким чином, щоб вони з'являлися на полі в місці кліку мишкою, але лише якщо ця клітинка ще не зайнята.
- Використовуйте ліст (список) або декілька змінних для контролю заповнення поля.

6) Перевірка перемоги:
- Напишіть код, який перевіряє, чи є у когось з гравців три "X" або "O" в ряду (горизонтально, вертикально, або діагонально).
- Повідомте про перемогу відповідного гравця.

7) Тестуйте гру:
- Грайте у «Хрестики-нулики» кілька разів, щоб перевірити, чи правильно працює розміщення знаків, зміна ходів та перевірка перемоги.

8) Налагодження:
- Переконайтеся, що всі можливі сценарії перемоги перевіряються коректно.
- Виправляйте помилки, якщо спрайти "X" або "O" можуть розміщуватися в одній і тій же клітинці.

9) Поліпшення гри:
- Додайте звуки для різних подій (наприклад, хід гравця).
- Встановіть фонове зображення для ігрового поля.
- Додайте кнопку «старт», щоб можна було легко почати гру заново після закінчення партії.

5. Проект «Анімація погоди»

1) Створення нового проекту:
- Відкрийте Scratch та створіть новий проект.
- Видаліть стандартний спрайт кота.

2) Додавання спрайтів та фонів:
- Додайте спрайти, які будуть представляти різні погодні умови: сонце, хмари, дощ (може бути у вигляді крапель), сніжинки.
- Виберіть або створіть фон, який пасуватиме для погодної анімації.

3) Програмування спрайту сонця:
- Спрайт сонця може з'являтися на сцені, коли вибрано «сонячну» погоду.
- Додайте блок «показати», коли клікнуто на кнопку «сонячно».

4) Програмування спрайту хмар:
- Хмари можуть плавно рухатися по сцені з одного краю на інший.
- Додайте блоки для руху хмар та блок «показати» або «сховати» в залежності від вибору погоди.

5) Програмування дощу та снігу:
- Для дощу і снігу використовуйте блоки «клонувати себе» для створення ефекту опадів.
- Кожен клон може падати згори вниз, зникаючи, коли досягає дна.

6) Створення інтерфейсу користувача:

- Додайте кнопки або перемикачі для вибору різних типів погоди.
- Програмуйте кнопки так, щоб вони активували показ відповідних спрайтів і анімацій.

7) Тестування анімацій:
- Перевірте, чи правильно працюють всі погодні анімації при виборі різних погодних умов.
- Переконайтеся, що спрайти коректно зникають і з'являються.

8) Налагодження:
- Виправляйте будь-які помилки у русі спрайтів або взаємодії з користувачем.

9) Додавання фінальних штрихів:
- Можна додати звуки дощу, вітру, або інші аудіо ефекти, щоб покращити іммерсивність анімації.
- Вдоскональте дизайн фону або спрайтів, якщо це потрібно.

6. Практична робота «Математична вікторина»

1) Створення нового проекту:
- Відкрийте Scratch і створіть новий проект.
- Видаліть стандартний спрайт кота.

2) Створення спрайта ведучого:
- Додайте спрайт, який буде виступати в ролі ведучого вікторини. Це може бути людина, тварина або анімований персонаж.
- Спрайт буде задавати питання та оголошувати правильні відповіді.

3) Генерація математичних питань:
- Використайте блоки для генерації випадкових чисел, щоб створити математичні питання. Наприклад, випадково оберіть два числа та тип операції (додавання, віднімання тощо).
- Задайте питання за допомогою блоку «говорити».

4) Введення відповідей користувачем:
- Використайте блок «запитати» для отримання відповіді від користувача.
- Збережіть відповідь у змінній.

5) Перевірка правильності відповідей:
- Використайте умовні блоки для перевірки правильності відповіді.
- Якщо відповідь правильна, виведіть повідомлення про успіх, інакше повідомте про помилку.

6) Таймер:
- Встановіть таймер, який обмежує час для відповіді на питання.
- Якщо час вийшов, перейдіть до наступного питання.

7) Рахунок:
- Створіть змінну «Рахунок», яка буде відстежувати кількість правильних відповідей.
- Збільшуйте рахунок за кожну правильну відповідь.

8) Тестування гри:

- Грайте в вікторину декілька разів, щоб переконатися, що всі математичні питання генеруються коректно, таймер працює, і рахунок нараховується правильно.

9) Налагодження:

- Виправте будь-які помилки, які ви знайдете під час тестування.

10) Додавання фінальних штрихів:

- Покращте візуальний дизайн гри, додавши анімації, змінюючи фони або додаючи звуки для взаємодії.
- Додайте інструкції для користувачів на початку гри.

7. Гра «Лабіринт з містером Потрібним»:

Завдання: Створити гру, де гравець керує персонажем, який шукає вихід з лабіринту, уникаючи містера Потрібного.

Практична робота: Учні можуть створити лабіринт, де гравець повинен знайти вихід. Вони також можуть додати персонажа містера Потрібного, який переслідує гравця. За допомогою умовних блоків, учні можуть визначити, що відбувається, коли гравець зіткнеться з містером Потрібним.

8. Гра «Потоп кораблів»:

Завдання: Створити гру, де гравець стріляє по координатах, намагаючись потопити кораблі противника.

Практична робота: Учні можуть створити поле для гри, де гравець може вибирати координати для стрільби. Вони можуть також додати кораблі противника, які повинні бути потоплені. За допомогою блоків умов та змінних, вони можуть визначити, чи було попадання та чи було потоплення корабля.

9. Анімація «Створення мультфільму»:

Завдання: Створити короткий мультфільм, використовуючи різні персонажі та сценарії.

Практична робота: Учні можуть створити власні персонажі та сцени для мультфільму за допомогою графічних блоків. Вони можуть додати діалоги та рух персонажів за допомогою блоків керування. Кожен учень може створити власну коротку історію або адаптувати існуючу казку.

10. Гра «Сліди динозавра»:

Завдання: Створити гру, де гравець керує динозавром, який повинен стрибати та уникати перешкод.

Практична робота: Учні можуть створити персонажа динозавра, який рухається по екрану, стрибаючи через перешкоди. Вони також можуть додати різноманітні перешкоди, такі як каміння або ріки, які динозавр повинен уникати. Логіка руху та стрибків може бути реалізована за допомогою блоків керування.

11. Проєкт «Годинник»

Завання: Створіть цифровий або аналоговий годинник використовуючи Scratch.

Практична робота:

1. Створити спрайти для годинникових стрілок (для аналогового годинника).
2. Використати блоки часу для отримання реального часу.

3. Налаштувати рух стрілок залежно від часу.

12. Проєкт «Вікторина»

Завдання: Створіть просту вікторину з кількома питаннями та варіантами відповідей.

Практична робота:

1. Створити список питань і варіантів відповідей.
2. Вивести питання і варіанти відповідей на екран.
3. Запрограмувати перевірку відповідей і нарахування балів.

3. Завдання та практичні роботи по темі «Алгоритми та програми» з мови програмування Python для 7 класу

1. Завдання: Обчислення площі прямокутника

Напишемо програму, яка запитує в користувача довжину та ширину прямокутника, а потім обчислює та виводить його площу.

```
# Запитати в користувача довжину та ширину прямокутника
length = float(input("Введіть довжину прямокутника: "))
width = float(input("Введіть ширину прямокутника: "))

# Обчислити площу прямокутника
area = length * width

# Вивести площу на екран
print("Площа прямокутника:", area)
```

Ця програма спочатку запитує в користувача довжину та ширину прямокутника, конвертує введені значення у десяткові числа, обчислює площу, перемножуючи довжину на ширину, а потім виводить результат на екран.

Роз'яснення:

- input() використовується для отримання введених користувачем значень.
- float() використовується для конвертації введених значень у десяткові числа, щоб мати можливість виконувати математичні операції.
- Площа прямокутника обчислюється за формулою площа = довжина * ширина.
- Результат виводиться на екран за допомогою функції print().

2. Завдання: Перевірка числа на парність

Напишемо програму, яка перевіряє, чи є введене користувачем число парним чи непарним.

```
# Запитати користувача про число
number = int(input("Введіть число: "))

# Перевірка на парність
if number % 2 == 0:
    print(number, "є парним числом.")
else:
    print(number, "є непарним числом.")
```

Ця програма спочатку запитує у користувача число, конвертує введене значення у ціле число за допомогою int(), потім перевіряє, чи ділиться введене число на 2 без залишку. Якщо так, вона виводить повідомлення про те, що

число є парним, в іншому випадку виводить повідомлення про те, що число є непарним.

Роз'яснення:

- input() використовується для отримання введеного користувачем значення.
- int() використовується для конвертації введеного значення у ціле число.
- % це оператор, що повертає залишок від ділення одного числа на інше.
- if умовний оператор, який виконує певний блок коду, якщо вираз умови істинний.
- else частина умовного оператора, яка виконується, якщо умова if не є істинною.

3. Завдання: Обчислення середнього арифметичного

Напишемо програму, яка запитує у користувача три числа, а потім обчислює та виводить їх середнє арифметичне.

```
# Запитати у користувача три числа
number1 = float(input("Введіть перше число: "))
number2 = float(input("Введіть друге число: "))
number3 = float(input("Введіть третє число: "))

# Обчислити середнє арифметичне
average = (number1 + number2 + number3) / 3

# Вивести середнє арифметичне на екран
print("Середнє арифметичне:", average)
```

Ця програма спочатку запитує у користувача три числа, конвертує введені значення у десяткові числа, обчислює їх суму, розділену на 3 (кількість чисел), щоб отримати середнє арифметичне, а потім виводить результат на екран.

Роз'яснення:

- input() використовується для отримання введеного користувачем значення.
- float() використовується для конвертації введеного значення у десяткове число, щоб мати можливість виконувати математичні операції.
- Середнє арифметичне обчислюється за формулою середнє = (число1 + число2 + число3) / 3.
- Результат виводиться на екран за допомогою функції print().

4. Завдання: Перевірка простого числа

Напишемо програму, яка перевіряє, чи є введене користувачем число простим.

```
# Запитати користувача про число
number = int(input("Введіть число: "))

# Перевірка на простоту
is_prime = True
```

```
if number <= 1:
    is_prime = False
else:
    for i in range(2, int(number/2) + 1):
        if number % i == 0:
            is_prime = False
            break

# Вивести результат перевірки на екран
if is_prime:
    print(number, "є простим числом.")
else:
    print(number, "не є простим числом.")
```

Ця програма спочатку запитує у користувача число, конвертує введене значення у ціле число за допомогою int(), а потім перевіряє, чи є число простим. Для перевірки на простоту використовується метод перебору всіх чисел від 2 до половини введеного числа. Якщо знайдено дільник, то число не є простим, якщо жоден дільник не знайдено, то число є простим.

Роз'яснення:

- input() використовується для отримання введеного користувачем значення.
- int() використовується для конвертації введеного значення у ціле число.
- range(start, stop) генерує послідовність чисел від start до stop - 1.
- Перевірка на простоту виконується шляхом перевірки, чи є введене число дільником для будь-якого числа від 2 до половини введеного числа.
- Результат перевірки виводиться на екран за допомогою функції print().

5. Завдання: Обчислення суми чисел до заданого користувачем значення

Напишемо програму, яка запитує у користувача ціле число n, а потім обчислює суму всіх цілих чисел від 1 до n.

```
# Запитати у користувача ціле число n
n = int(input("Введіть ціле число n: "))

# Ініціалізувати змінну для збереження суми чисел
sum_of_numbers = 0

# Обчислити суму чисел від 1 до n
for i in range(1, n + 1):
    sum_of_numbers += i

# Вивести суму на екран
print("Сума чисел від 1 до", n, "дорівнює", sum_of_numbers)
```

Ця програма спочатку запитує у користувача ціле число n, а потім використовує цикл for, щоб пройтися по всім цілим числам від 1 до n і додати їх до змінної sum_of_numbers. Нарешті, вона виводить суму на екран.

Роз'яснення:

- input() використовується для отримання введеного користувачем значення.
- int() використовується для конвертації введеного значення у ціле число.
- Цикл for використовується для проходження по всім цілим числам від 1 до n.
- Сума чисел обчислюється за допомогою операції додавання +=.
- Результат виводиться на екран за допомогою функції print().

6. Практична робота: Розробка простого калькулятора

Завдання: Напишіть програму-калькулятор, яка запитує у користувача два числа та операцію (додавання, віднімання, множення або ділення) та виводить результат обчислення на екран.

Послідовність дій:

1. Запитати у користувача перше число.
2. Запитати у користувача операцію (за допомогою символів +, -, *, /).
3. Запитати у користувача друге число.
4. Виконати обчислення відповідно до введеної операції.
5. Вивести результат на екран.

```
Введіть перше число: 10
Введіть операцію (+, -, *, /): *
Введіть друге число: 5
Результат: 50

# Запитати у користувача перше число
num1 = float(input("Введіть перше число: "))

# Запитати у користувача операцію
operation = input("Введіть операцію (+, -, *, /): ")

# Запитати у користувача друге число
num2 = float(input("Введіть друге число: "))

# Виконати обчислення та вивести результат
if operation == "+":
    result = num1 + num2
elif operation == "-":
    result = num1 - num2
elif operation == "*":
    result = num1 * num2
elif operation == "/":
    # Перевірити, чи друге число не дорівнює 0 для уникнення ділення на 0
```

```
    if num2 != 0:
        result = num1 / num2
    else:
        result = "Помилка: ділення на нуль!"
else:
    result = "Невідома операція!"

print("Результат:", result)
```

У цьому прикладі програма працює правильно, обчислюючи різницю між числами та виводячи результат.

7. Практична робота: Обчислення середнього балу учня

Завдання:

Напишіть програму для обчислення середнього балу учня за його оцінками з різних предметів. У програмі має бути можливість введення оцінок з кількох предметів, обчислення середнього балу та виведення результату на екран.

Послідовність дій:

1. Запитати у користувача кількість предметів.
2. Пройтися по кожному предмету та запитати у користувача його оцінку.
3. Обчислити середній бал за формулою: середній бал = (сума оцінок) / (кількість предметів).
4. Вивести середній бал на екран.

```
Скільки предметів: 4
Введіть оцінку з предмету 1: 90
Введіть оцінку з предмету 2: 85
Введіть оцінку з предмету 3: 88
Введіть оцінку з предмету 4: 92
Середній бал: 88.75

# Запитати у користувача кількість предметів
num_subjects = int(input("Скільки предметів: "))

# Ініціалізувати змінну для збереження суми оцінок
total_marks = 0

# Пройтися по кожному предмету та запитати у користувача оцінку
for i in range(1, num_subjects + 1):
    mark = float(input(f"Введіть оцінку з предмету {i}: "))
    total_marks += mark

# Обчислити середній бал
```

```
average_mark = total_marks / num_subjects

# Вивести середній бал на екран
print("Середній бал:", average_mark)
```

У цьому прикладі програма запитує у користувача кількість предметів та оцінки з кожного предмету, обчислює їх суму та середній бал, і виводить результат на екран.

8. Практична робота: Гра «Камінь, ножиці, папір» проти комп'ютера

Завдання:

Напишіть програму для гри «Камінь, ножиці, папір» проти комп'ютера. У цій грі гравець вибирає один із трьох об'єктів («камінь», «ножиці» або «папір»), а комп'ютер також робить свій вибір, після чого визначається переможець.

Послідовність дій:

1. Створіть список із трьох можливих варіантів вибору: «камінь», «ножиці» та «папір».
2. Запитайте у користувача його вибір.
3. Зробіть вибір комп'ютера випадковим чином.
4. Порівняйте вибір гравця та комп'ютера та визначте переможця.
5. Виведіть результат гри на екран.

```
Ваш вибір (камінь, ножиці, папір): ножиці
Вибір комп'ютера: камінь
Ви перемогли! Ножиці перемагають камінь.
import random

# Список варіантів вибору
options = ["камінь", "ножиці", "папір"]

# Запитати вибір гравця
player_choice = input("Ваш вибір (камінь, ножиці, папір): ")

# Зробити вибір комп'ютера випадковим чином
computer_choice = random.choice(options)

# Вивести вибір комп'ютера на екран
print("Вибір комп'ютера:", computer_choice)

# Порівняти вибір гравця та комп'ютера та визначити переможця
if player_choice == computer_choice:
    print("Нічия!")
```

```
elif (player_choice == "камінь" and computer_choice == "ножиці") or \
    (player_choice == "ножиці" and computer_choice == "папір") or \
    (player_choice == "папір" and computer_choice == "камінь"):
   print("Ви перемогли!")
else:
   print("Ви програли!")
```

У цьому прикладі програма дозволяє гравцеві зіграти в гру «Камінь, ножиці, папір» проти комп'ютера. Користувач вводить свій вибір, після чого програма обирає випадковий варіант для комп'ютера і визначає переможця.

9. Практична робота: Конвертер температури

Завдання: Напишіть програму, яка конвертує температуру з градусів Цельсія у градуси Фаренгейта або навпаки.

Послідовність дій:

1. Запитайте у користувача, яку операцію він хоче виконати: перевести з градусів Цельсія у градуси Фаренгейта або навпаки.
2. Запитайте у користувача температуру.
3. Зробіть необхідний розрахунок для конвертації температури.
4. Виведіть результат на екран.

```
Що ви хочете зробити? (c2f або f2c): c2f
Введіть температуру в градусах Цельсія: 25
Температура в градусах Фаренгейта: 77.0

# Запитати користувача, яку операцію він хоче виконати
operation = input("Що ви хочете зробити? (c2f або f2c): ")

# Запитати температуру
temperature = float(input("Введіть температуру: "))

# Зробити розрахунок для конвертації температури
if operation == "c2f":
   converted_temperature = (temperature * 9/5) + 32
   print("Температура в градусах Фаренгейта:", converted_temperature)
elif operation == "f2c":
   converted_temperature = (temperature - 32) * 5/9
   print("Температура в градусах Цельсія:", converted_temperature)
else:
```

```
    print("Невірна операція. Введіть 'c2f' або 'f2c'.")
```

10. Практична робота: Перевірка на паліндром

Завдання: Напишіть програму, яка перевіряє, чи є задане слово або фраза паліндромом (слово, яке читається однаково ззаду наперед).

Послідовність дій:

1. Запитайте у користувача слово або фразу для перевірки.
2. Перевірте, чи є введений текст паліндромом.
3. Виведіть результат перевірки на екран.

```
Введіть слово або фразу: radar
Так, це паліндром!
```

```
# Запитати у користувача слово або фразу для перевірки
word = input("Введіть слово або фразу: ")

# Перевернути слово або фразу
reversed_word = word[::-1]

# Перевірити, чи є введений текст паліндромом
if word.lower() == reversed_word.lower():
    print("Так, це паліндром!")
else:
    print("Ні, це не паліндром.")
```

11. Практична робота: Генерація паролю

Завдання: Напишіть програму для генерації надійного паролю. Пароль має містити випадкову комбінацію великих та малих літер, цифр та спеціальних символів.

Послідовність дій:

1. Використайте модуль random для вибору випадкових символів з різних наборів.
2. Створіть пароль, який містить великі та малі літери, цифри та спеціальні символи.
3. Виведіть згенерований пароль на екран.

```
Згенерований пароль: 3aB&5p$z
```

```
import random
import string

# Створити набір символів для генерації паролю
characters = string.ascii_letters + string.digits + string.punctuation

# Згенерувати пароль
password = ''.join(random.choice(characters) for _ in range(8))

# Вивести згенерований пароль на екран
print("Згенерований пароль:", password)
```

12. Практична робота: Генерація чисел Фібоначчі

Завдання: Напишіть програму, яка генерує перші n чисел послідовності Фібоначчі.

Послідовність дій:

1. Запитайте у користувача кількість чисел, які треба згенерувати.
2. Використайте цикл для генерації перших n чисел послідовності Фібоначчі.
3. Виведіть згенеровані числа на екран.

```
Скільки чисел послідовності Фібоначчі потрібно згенерувати? 10
Перші 10 чисел послідовності Фібоначчі: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34

# Запитати у користувача кількість чисел для генерації
n = int(input("Скільки чисел послідовності Фібоначчі потрібно згенерувати? "))

# Ініціалізувати перші два числа послідовності
fibonacci_sequence = [0, 1]

# Згенерувати перші n чисел послідовності Фібоначчі
for i in range(2, n):
    next_number = fibonacci_sequence[-1] + fibonacci_sequence[-2]
    fibonacci_sequence.append(next_number)

# Вивести згенеровані числа на екран
```

```
print("Перші", n, "чисел послідовності Фібоначчі:", ', '.join(map(str, fibonacci_sequence)))
```

4. Завдання та практичні роботи по темі «Алгоритми та програми» з мови програмування Python для 8 класу

1. Задача на шифрування та дешифрування тексту:

Напишіть програму, яка шифрує або дешифрує введений користувачем текст за допомогою шифру Цезаря. Користувач повинен мати можливість вибрати зсув (ключ шифрування), а також обрати, чи хоче він зашифрувати текст, чи дешифрувати його.

```
def caesar_cipher(text, shift, decrypt=False):
    result = ''
    for char in text:
        if char.isalpha():
            shift_amount = shift if not decrypt else -shift
            if char.isupper():
                result += chr((ord(char) - 65 + shift_amount) % 26 + 65)
            else:
                result += chr((ord(char) - 97 + shift_amount) % 26 + 97)
        else:
            result += char
    return result

# Функція для вибору дії (шифрування чи дешифрування)
def choose_action():
    while True:
                action = input("Виберіть дію ('e' для шифрування, 'd' для дешифрування): ").lower()
        if action in ['e', 'd']:
            return action == 'd'
        else:
            print("Неправильний вибір. Спробуйте знову.")

# Введення тексту та ключа користувачем
text = input("Введіть текст: ")
shift = int(input("Введіть ключ шифрування (ціле число): "))

# Вибір дії (шифрування чи дешифрування)
decrypt = choose_action()

# Виведення зашифрованого або розшифрованого тексту
if decrypt:
    print("Розшифрований текст:", caesar_cipher(text, shift, decrypt=True))
else:
    print("Зашифрований текст:", caesar_cipher(text, shift))
```

У цій програмі функція caesar_cipher шифрує або дешифрує текст за допомогою шифру Цезаря з використанням заданого ключа зсуву. Користувач має можливість вибрати, чи хоче він зашифрувати чи дешифрувати текст, а також вказати ключ шифрування. Завдання допоможе учням краще зрозуміти принцип роботи з рядками та умовними конструкціями у Python, а також збагатить їхні знання з шифрування та дешифрування даних.

2. Задача на перевірку власної ваги на різних планетах:

Напишіть програму, яка дозволяє користувачеві ввести свою вагу на Землі та виводить його вагу на інших планетах. Нехай користувач обирає планету з наступного списку: Марс, Юпітер, Венера, Сатурн, Меркурій, Уран, Нептун.

```
def calculate_weight_on_planet(weight, planet):
    planets_gravity = {
        "Марс": 0.38,
        "Юпітер": 2.34,
        "Венера": 0.91,
        "Сатурн": 1.06,
        "Меркурій": 0.38,
        "Уран": 0.92,
        "Нептун": 1.19
    }
    if planet in planets_gravity:
        return weight * planets_gravity[planet]
    else:
        return "Неправильно введено назву планети"

# Введення ваги користувачем
weight = float(input("Введіть вашу вагу на Землі (кг): "))

# Введення назви планети користувачем
planet = input("Введіть назву планети (Марс, Юпітер, Венера, Сатурн, Меркурій, Уран, Нептун): ")

# Обчислення ваги на вибраній планеті та виведення результату
print("Ваша вага на", planet, "дорівнює", calculate_weight_on_planet(weight, planet), "кг")
```

У цій програмі функція calculate_weight_on_planet приймає вагу користувача та назву планети, на яку він хоче порахувати свою вагу. За допомогою словника planets_gravity вона визначає відповідність між назвою планети та гравітаційним прискоренням на цій планеті. Потім вона обчислює вагу користувача на обраній планеті та повертає результат.

3. Завдання: Розробка програми «Бібліотека книг»

Опис завдання: Розробити програму для управління бібліотекою книг. Програма повинна дозволяти додавати нові книги, видаляти книги, шукати книги за назвою, автором, або роком видання, а також відображати весь список книг у бібліотеці.

Вимоги до програми:

1. Використання функцій для реалізації різних частин програми.
2. Використання циклів та умовних операторів для обробки даних.
3. Введення даних користувачем та виведення результатів на екран.
4. Зберігання даних про книги у вигляді списку словників.

Кроки виконання завдання:

1. Створення основної структури програми:

```
def main():
    library = []
    while True:
        print("\n1. Додати книгу")
        print("2. Видалити книгу")
        print("3. Шукати книгу")
        print("4. Відобразити всі книги")
        print("5. Вийти")
        choice = input("Виберіть опцію: ")
        if choice == '1':
            add_book(library)
        elif choice == '2':
            remove_book(library)
        elif choice == '3':
            search_books(library)
        elif choice == '4':
            display_books(library)
        elif choice == '5':
            break
        else:
            print("Невірний вибір. Спробуйте ще раз.")

if __name__ == "__main__":
    main()
```

2. Функція для додавання нової книги:

```
def add_book(library):
    title = input("Введіть назву книги: ")
    author = input("Введіть автора книги: ")
    year = input("Введіть рік видання: ")
    book = {"Назва": title, "Автор": author, "Рік видання": year}
    library.append(book)
    print("Книга додана до бібліотеки.")
```

3. Функція для видалення книги:

```
def remove_book(library):
    title = input("Введіть назву книги для видалення: ")
    for book in library:
        if book["Назва"] == title:
            library.remove(book)
            print("Книгу видалено.")
            return
    print("Книгу не знайдено.")
```

4. Функція для пошуку книги:

```
def search_books(library):
    search_option = input("Шукати за (1) Назвою, (2) Автором, (3) Роком
видання: ")
    search_term = input("Введіть пошуковий запит: ")
    results = []
    for book in library:
        if search_option == '1' and search_term.lower() in book["Назва"].lower():
            results.append(book)
        elif search_option == '2' and search_term.lower() in
book["Автор"].lower():
            results.append(book)
        elif search_option == '3' and search_term == book["Рік видання"]:
            results.append(book)
    if results:
        print("Знайдені книги:")
        for book in results:
            print(f"Назва: {book['Назва']}, Автор: {book['Автор']}, Рік видання:
{book['Рік видання']}")
    else:
        print("Книг не знайдено.")
```

5. Функція для відображення всіх книг:

```
def display_books(library):
    if library:
        print("Список книг у бібліотеці:")
        for book in library:
            print(f"Назва: {book['Назва']}, Автор: {book['Автор']}, Рік видання:
{book['Рік видання']}")
    else:
        print("Бібліотека порожня.")
```

Додаткові ідеї для покращення програми:

- Додати можливість зберігати дані про книги у файл і завантажувати їх при старті програми.
- Реалізувати сортування книг за різними критеріями (назва, автор, рік видання).
- Додати можливість редагувати інформацію про книги.

4. Завдання: Розробка гри «Вгадати число»

Опис завдання: Розробити програму гри, в якій користувач повинен вгадати число, яке загадала програма. Програма повинна давати підказки користувачу: чи є його припущення більшим або меншим за загадане число. Користувач повинен мати обмежену кількість спроб на вгадування.

Вимоги до програми:

1. Використання функцій для реалізації різних частин програми.
2. Використання циклів та умовних операторів для обробки даних.
3. Введення даних користувачем та виведення результатів на екран.
4. Використання генератора випадкових чисел для загадування числа.

Кроки виконання завдання:

1. Створення основної структури програми:

```
import random

def main():
    print("Ласкаво просимо до гри 'Вгадати число'!")
    while True:
        play_game()
        play_again = input("Хочете зіграти ще раз? (так/ні): ").lower()
        if play_again != 'так':
            break
    print("Дякую за гру! До побачення.")

if __name__ == "__main__":
```

```
    main()
```

2. Функція для гри:

```
def play_game():
    number_to_guess = random.randint(1, 100)
    attempts_left = 10

    print("Я загадала число від 1 до 100.")
    print(f"У вас є {attempts_left} спроб, щоб вгадати його.")

    while attempts_left > 0:
        guess = int(input("Введіть ваше припущення: "))
        if guess < number_to_guess:
            print("Загадане число більше.")
        elif guess > number_to_guess:
            print("Загадане число менше.")
        else:
            print(f"Вітаємо! Ви вгадали число {number_to_guess}.")
            return

        attempts_left -= 1
        print(f"У вас залишилося {attempts_left} спроб.")

    print(f"На жаль, ви не вгадали. Загадане число було {number_to_guess}.")
```

3. Функція для введення чисел з перевіркою:

```
def get_guess():
    while True:
        try:
            guess = int(input("Введіть ваше припущення: "))
            return guess
        except ValueError:
            print("Будь ласка, введіть ціле число.")
```

4. Оновлення функції гри для використання функції введення:

```
def play_game():
    number_to_guess = random.randint(1, 100)
    attempts_left = 10

    print("Я загадала число від 1 до 100.")
    print(f"У вас є {attempts_left} спроб, щоб вгадати його.")
```

```
    while attempts_left > 0:
        guess = get_guess()
        if guess < number_to_guess:
            print("Загадане число більше.")
        elif guess > number_to_guess:
            print("Загадане число менше.")
        else:
            print(f"Вітаємо! Ви вгадали число {number_to_guess}.")
            return

        attempts_left -= 1
        print(f"У вас залишилося {attempts_left} спроб.")

    print(f"На жаль, ви не вгадали. Загадане число було {number_to_guess}.")
```

Додаткові ідеї для покращення програми:

- Додати можливість вибору діапазону чисел для вгадування.
- Додати режим гри з двома гравцями, де кожен по черзі вгадує число.
- Зберігати статистику гри (кількість спроб, найкращий результат тощо).

5. Завдання: Розробка програми «Менеджер завдань»

Опис завдання: Розробити програму, яка допоможе користувачу керувати своїми завданнями. Програма повинна дозволяти додавати нові завдання, видаляти виконані завдання, відзначати завдання як виконані, а також відображати список всіх завдань і тільки невиконаних завдань.

Вимоги до програми:

1. Використання функцій для реалізації різних частин програми.
2. Використання циклів та умовних операторів для обробки даних.
3. Введення даних користувачем та виведення результатів на екран.
4. Зберігання даних про завдання у вигляді списку словників.

Кроки виконання завдання:

1. Створення основної структури програми:

```
    def main():
        tasks = []
        while True:
            print("\nМеню:")
            print("1. Додати завдання")
            print("2. Видалити завдання")
            print("3. Відзначити завдання як виконане")
```

```
        print("4. Показати всі завдання")
        print("5. Показати невиконані завдання")
        print("6. Вийти")
        choice = input("Виберіть опцію: ")

        if choice == '1':
            add_task(tasks)
        elif choice == '2':
            remove_task(tasks)
        elif choice == '3':
            complete_task(tasks)
        elif choice == '4':
            show_all_tasks(tasks)
        elif choice == '5':
            show_pending_tasks(tasks)
        elif choice == '6':
            break
        else:
            print("Невірний вибір. Спробуйте ще раз.")

if __name__ == "__main__":
    main()
```

2. Функція для додавання нового завдання:

```
def add_task(tasks):
    task_name = input("Введіть назву завдання: ")
    task = {"назва": task_name, "виконано": False}
    tasks.append(task)
    print("Завдання додано.")
```

3. Функція для видалення завдання:

```
def remove_task(tasks):
    task_name = input("Введіть назву завдання для видалення: ")
    for task in tasks:
        if task["назва"] == task_name:
            tasks.remove(task)
            print("Завдання видалено.")
            return
    print("Завдання не знайдено.")
```

4. Функція для відзначення завдання як виконаного:

```
def complete_task(tasks):
    task_name = input("Введіть назву завдання для відзначення як виконане: ")
    for task in tasks:
        if task["назва"] == task_name:
            task["виконано"] = True
            print("Завдання відзначено як виконане.")
            return
    print("Завдання не знайдено.")
```

5. Функція для відображення всіх завдань:

```
def show_all_tasks(tasks):
    if tasks:
        print("Список всіх завдань:")
        for task in tasks:
            status = "виконано" if task["виконано"] else "не виконано"
            print(f"{task['назва']} - {status}")
    else:
        print("Список завдань порожній.")
```

6. Функція для відображення невиконаних завдань:

```
def show_pending_tasks(tasks):
    pending_tasks = [task for task in tasks if not task["виконано"]]
    if pending_tasks:
        print("Список невиконаних завдань:")
        for task in pending_tasks:
            print(task["назва"])
    else:
        print("Немає невиконаних завдань.")
```

Додаткові ідеї для покращення програми:

- Додати можливість встановлення дедлайну для завдань.
- Додати пріоритети для завдань (високий, середній, низький).
- Зберігати дані про завдання у файл і завантажувати їх при старті програми.

Приклад роботи програми:

Меню:

1. Додати завдання
2. Видалити завдання
3. Відзначити завдання як виконане
4. Показати всі завдання
5. Показати невиконані завдання

6. Вийти
Виберіть опцію: 1
Введіть назву завдання: Зробити домашнє завдання
Завдання додано.
Меню:
1. Додати завдання
2. Видалити завдання
3. Відзначити завдання як виконане
4. Показати всі завдання
5. Показати невиконані завдання
6. Вийти
Виберіть опцію: 4
Список всіх завдань:
Зробити домашнє завдання - не виконано
Меню:
1. Додати завдання
2. Видалити завдання
3. Відзначити завдання як виконане
4. Показати всі завдання
5. Показати невиконані завдання
6. Вийти
Виберіть опцію: 3
Введіть назву завдання для відзначення як виконане: Зробити домашнє завдання
Завдання відзначено як виконане.
Меню:
1. Додати завдання
2. Видалити завдання
3. Відзначити завдання як виконане
4. Показати всі завдання
5. Показати невиконані завдання
6. Вийти
Виберіть опцію: 5
Немає невиконаних завдань.

6. Завдання: Розробка програми «Генератор паролів»

Опис завдання: Розробити програму, яка генерує надійні паролі відповідно до вимог користувача. Користувач може вказати довжину пароля, а також вибрати, чи повинні пароль містити великі та малі літери, цифри та спеціальні символи.

Вимоги до програми:

1. Використання функцій для реалізації різних частин програми.

2. Використання циклів та умовних операторів для обробки даних.
3. Введення даних користувачем та виведення результатів на екран.
4. Використання бібліотеки random для генерації випадкових символів.

Кроки виконання завдання:

1. Створення основної структури програми:

```
import random
import string

def main():
    print("Ласкаво просимо до Генератора Паролів!")
    while True:
        length = get_password_length()
        use_upper = ask_yes_no("Чи повинні пароль містити великі літери? (так/ні): ")
        use_lower = ask_yes_no("Чи повинні пароль містити малі літери? (так/ні): ")
        use_digits = ask_yes_no("Чи повинні пароль містити цифри? (так/ні): ")
        use_symbols = ask_yes_no("Чи повинні пароль містити спеціальні символи? (так/ні): ")

        password = generate_password(length, use_upper, use_lower, use_digits, use_symbols)
        print(f"Згенерований пароль: {password}")

        play_again = input("Хочете згенерувати ще один пароль? (так/ні): ").lower()
        if play_again != 'так':
            break
    print("Дякую за використання Генератора Паролів! До побачення.")

if __name__ == "__main__":
    main()
```

2. Функція для введення довжини пароля:

```
def get_password_length():
    while True:
        try:
            length = int(input("Введіть довжину пароля: "))
            if length > 0:
```

```
            return length
        else:
            print("Довжина пароля повинна бути більше нуля.")
    except ValueError:
        print("Будь ласка, введіть ціле число.")
```

3. Функція для запиту 'так' або 'ні':

```
def ask_yes_no(prompt):
    while True:
        answer = input(prompt).lower()
        if answer in ['так', 'ні']:
            return answer == 'так'
        else:
            print("Будь ласка, введіть 'так' або 'ні'.")
```

4. Функція для генерації пароля:

```
def generate_password(length, use_upper, use_lower, use_digits,
use_symbols):
    characters = ''
    if use_upper:
        characters += string.ascii_uppercase
    if use_lower:
        characters += string.ascii_lowercase
    if use_digits:
        characters += string.digits
    if use_symbols:
        characters += string.punctuation

    if not characters:
        print("Ви повинні вибрати принаймні один тип символів.")
        return ''

    password = ''.join(random.choice(characters) for _ in range(length))
    return password
```

Додаткові ідеї для покращення програми:

- Додати можливість збереження згенерованих паролів у файл.
- Додати функцію для перевірки надійності згенерованого пароля.
- Додати можливість вибору заборонених символів.

Приклад роботи програми:

Ласкаво просимо до Генератора Паролів!

Введіть довжину пароля: 12
Чи повинні пароль містити великі літери? (так/ні): так
Чи повинні пароль містити малі літери? (так/ні): так
Чи повинні пароль містити цифри? (так/ні): так
Чи повинні пароль містити спеціальні символи? (так/ні): ні
Згенерований пароль: Aa1b2c3D4e5F

Хочете згенерувати ще один пароль? (так/ні): ні
Дякую за використання Генератора Паролів! До побачення.

7. Завдання: Розробка програми «Міні-гра: Змійка»

Опис завдання: Розробити просту версію популярної гри «Змійка», в якій гравець керує змійкою, яка рухається по екрану, збираючи їжу та збільшуючись у довжину. Гра закінчується, якщо змійка зіштовхується з самим собою або з межами екрану.

Вимоги до програми:

1. Використання бібліотеки Pygame для графічного інтерфейсу.
2. Використання циклів та умовних операторів для обробки гри.
3. Використання функцій для реалізації різних частин програми.
4. Створення логіки для руху змійки, генерації їжі та перевірки зіткнень.

Кроки виконання завдання:

1. Встановлення бібліотеки Pygame:
    - Перед початком роботи переконайтеся, що у вас встановлена бібліотека Pygame. Ви можете встановити її за допомогою команди: pip install pygame
2. Основна структура програми:

```
import pygame
import time
import random

pygame.init()

white = (255, 255, 255)
yellow = (255, 255, 102)
black = (0, 0, 0)
red = (213, 50, 80)
green = (0, 255, 0)
blue = (50, 153, 213)

dis_width = 800
dis_height = 600

dis = pygame.display.set_mode((dis_width, dis_height))
pygame.display.set_caption('Змійка')
```

```
clock = pygame.time.Clock()
snake_block = 10
snake_speed = 15

font_style = pygame.font.SysFont("bahnschrift", 25)
score_font = pygame.font.SysFont("comicsansms", 35)

def our_snake(snake_block, snake_list):
    for x in snake_list:
        pygame.draw.rect(dis, black, [x[0], x[1], snake_block, snake_block])

def message(msg, color):
    mesg = font_style.render(msg, True, color)
    dis.blit(mesg, [dis_width / 6, dis_height / 3])

def gameLoop():
    game_over = False
    game_close = False

    x1 = dis_width / 2
    y1 = dis_height / 2

    x1_change = 0
    y1_change = 0

    snake_List = []
    Length_of_snake = 1

    foodx = round(random.randrange(0, dis_width - snake_block) / 10.0) * 10.0
    foody = round(random.randrange(0, dis_height - snake_block) / 10.0) * 10.0

    while not game_over:

        while game_close == True:
            dis.fill(blue)
            message("Вы проиграли! Нажмите Q для выхода или C для
продолжения", red)
            pygame.display.update()

            for event in pygame.event.get():
                if event.type == pygame.KEYDOWN:
                    if event.key == pygame.K_q:
```

```
                game_over = True
                game_close = False
            if event.key == pygame.K_c:
                gameLoop()

    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            game_over = True
        if event.type == pygame.KEYDOWN:
            if event.key == pygame.K_LEFT:
                x1_change = -snake_block
                y1_change = 0
            elif event.key == pygame.K_RIGHT:
                x1_change = snake_block
                y1_change = 0
            elif event.key == pygame.K_UP:
                y1_change = -snake_block
                x1_change = 0
            elif event.key == pygame.K_DOWN:
                y1_change = snake_block
                x1_change = 0

    if x1 >= dis_width or x1 < 0 or y1 >= dis_height or y1 < 0:
        game_close = True
    x1 += x1_change
    y1 += y1_change
    dis.fill(blue)
    pygame.draw.rect(dis, green, [foodx, foody, snake_block, snake_block])
    snake_Head = []
    snake_Head.append(x1)
    snake_Head.append(y1)
    snake_List.append(snake_Head)
    if len(snake_List) > Length_of_snake:
        del snake_List[0]

    for x in snake_List[:-1]:
        if x == snake_Head:
            game_close = True

    our_snake(snake_block, snake_List)
    pygame.display.update()

    if x1 == foodx and y1 == foody:
```

```
            foodx = round(random.randrange(0, dis_width - snake_block) / 10.0) *
    10.0
            foody = round(random.randrange(0, dis_height - snake_block) / 10.0) *
    10.0
            Length_of_snake += 1

        clock.tick(snake_speed)

    pygame.quit()
    quit()

gameLoop()
```

Додаткові ідеї для покращення програми:

- Додати систему балів, яка підраховує кількість з'їдених їжинок.
- Додати різні рівні складності, які збільшують швидкість змійки.
- Реалізувати функціонал збереження найвищого результату.

Приклад роботи програми:

Гра «Змійка» запускається у вікні Pygame, де гравець може керувати змійкою за допомогою клавіш стрілок. Змійка збирає їжу, збільшуючись у довжину. Гра закінчується, якщо змійка зіштовхується з самою собою або з межами екрану. Гравець може розпочати гру заново, натиснувши клавішу 'C', або вийти з гри, натиснувши клавішу 'Q'.

8. Практична робота: Створення текстової гри «Пригоди в підземеллі»

Опис завдання:

Розробити текстову гру, де гравець переміщується по підземеллю, збираючи предмети та уникаючи пасток. Гра повинна мати кілька кімнат, у яких гравець може знайти різні предмети або зіткнутися з небезпекою. Гравець може переміщатися між кімнатами, вибираючи напрямок руху.

Вимоги до програми:

1. Використання функцій для реалізації різних частин гри.
2. Використання циклів та умовних операторів для обробки гри.
3. Введення даних користувачем та виведення результатів на екран.
4. Зберігання інформації про стан гри у змінних (наприклад, інвентар гравця, поточна кімната).

Структура програми:

1. Ініціалізація гри:
    - Створення картки підземелля (кілька кімнат).
    - Визначення початкової позиції гравця.
    - Ініціалізація інвентаря гравця.
2. Головний цикл гри:
    - Виведення інформації про поточну кімнату.

- Запит на введення дії гравця (переміщення, огляд кімнати, взяти предмет тощо).
- Обробка дії гравця.
- Оновлення стану гри.

3. Використання функцій для різних дій:
   - Переміщення між кімнатами.
   - Огляд кімнати.
   - Взаємодія з предметами.

Приклад реалізації:

```
def init_game():
    # Ініціалізація картки підземелля
    rooms = {
        "вхід": {"опис": "Ви знаходитесь біля входу в підземелля.", "північ": "коридор", "предмети": ["факел"]},
        "коридор": {"опис": "Ви в темному коридорі.", "південь": "вхід", "схід": "кімната зі скарбами", "предмети": []},
        "кімната зі скарбами": {"опис": "Ви знайшли кімнату зі скарбами!", "захід": "коридор", "предмети": ["скарб"]},
    }
    player_position = "вхід"
    inventory = []
    return rooms, player_position, inventory

def show_room_description(rooms, position):
    room = rooms[position]
    print(room["опис"])
    if room["предмети"]:
        print("Тут є такі предмети:", ", ".join(room["предмети"]))

def get_player_action():
    action = input("Що ви хочете зробити? (йти/взяти/вихід): ")
    return action.lower()

def move_player(rooms, position):
    direction = input("В який напрямок ви хочете йти? (північ/південь/схід/захід): ")
    if direction in rooms[position]:
        return rooms[position][direction]
    else:
```

```
        print("Ви не можете йти в цьому напрямку.")
        return position

def take_item(rooms, position, inventory):
    room = rooms[position]
    if room["предмети"]:
        item = room["предмети"].pop()
        inventory.append(item)
        print(f"Ви взяли {item}.")
    else:
        print("Тут немає нічого, що можна взяти.")

def main():
    rooms, player_position, inventory = init_game()

    print("Ласкаво просимо до гри 'Пригоди в підземеллі'!")

    while True:
        show_room_description(rooms, player_position)
        action = get_player_action()

        if action == "йти":
            player_position = move_player(rooms, player_position)
        elif action == "взяти":
            take_item(rooms, player_position, inventory)
        elif action == "вихід":
            print("Дякую за гру! До побачення.")
            break
        else:
            print("Невідома дія. Спробуйте ще раз.")
        print(f"Ваш інвентар: {', '.join(inventory)}")

if __name__ == "__main__":
    main()
```

Додаткові ідеї для покращення гри:

- Додати більше кімнат та предметів.
- Ввести можливість зустрічати монстрів та битися з ними.
- Додати пастки, яких потрібно уникати.

- Реалізувати систему здоров'я гравця та можливість лікування.
- Додати головоломки для відкриття нових кімнат.

Приклад роботи гри:
Ласкаво просимо до гри 'Пригоди в підземеллі'!
Ви знаходитесь біля входу в підземелля.
Тут є такі предмети: факел
Що ви хочете зробити? (йти/взяти/вихід): взяти
Ви взяли факел.
Ваш інвентар: факел
Що ви хочете зробити? (йти/взяти/вихід): йти
В який напрямок ви хочете йти? (північ/південь/схід/захід): північ
Ви в темному коридорі.
Що ви хочете зробити? (йти/взяти/вихід): йти
В який напрямок ви хочете йти? (північ/південь/схід/захід): схід
Ви знайшли кімнату зі скарбами!
Тут є такі предмети: скарб
Що ви хочете зробити? (йти/взяти/вихід): взяти
Ви взяли скарб.
Ваш інвентар: факел, скарб

9. Практична робота. Симулятор магазинних покупок

Завдання:

- Створити програму, яка симулює процес покупок у віртуальному магазині, де користувач може додавати товари у кошик, переглядати кошик та оформляти замовлення.

Кроки:

1. Ініціалізація проекту:
    - Створити простий інтерфейс з головним меню (перегляд товарів, додавання в кошик, перегляд кошика, оформлення замовлення).
2. Створення класів товарів:
    - Визначити клас Product з атрибутами (назва, ціна, кількість).
    - Створити кілька різних товарів для демонстрації.
3. Перегляд товарів:
    - Реалізувати функціонал для перегляду списку доступних товарів з їх описами та цінами.
    - Додати можливість сортування товарів за ціною, назвою, категорією.
4. Додавання в кошик:
    - Додати можливість додавання товарів в кошик та перегляду поточного стану кошика.

- Реалізувати функції для зміни кількості товарів у кошику та видалення товарів.

5. Оформлення замовлення:
   - Додати можливість оформлення замовлення з підрахунком загальної вартості.
   - Додати підтвердження замовлення та можливість його скасування.

Ініціалізація проекту

1. Створення основного файлу програми:

```
import tkinter as tk
from tkinter import messagebox

# Ініціалізація головного вікна програми
root = tk.Tk()
root.title("Магазин")

# Головне меню
def main_menu():
    for widget in root.winfo_children():
        widget.destroy()

    title = tk.Label(root, text="Ласкаво просимо до нашого магазину!",
font=('Helvetica', 16))
    title.pack(pady=10)

    view_products_btn = tk.Button(root, text="Переглянути товари",
command=view_products)
    view_products_btn.pack(pady=5)

    view_cart_btn = tk.Button(root, text="Переглянути кошик",
command=view_cart)
    view_cart_btn.pack(pady=5)

    checkout_btn = tk.Button(root, text="Оформити замовлення",
command=checkout)
    checkout_btn.pack(pady=5)

main_menu()
root.mainloop()
```

- Ініціалізація головного вікна програми: Тут створюється основне вікно програми за допомогою бібліотеки tkinter.
- Головне меню: Функція main_menu створює головне меню з кнопками для перегляду товарів, перегляду кошика та оформлення замовлення. Під час

виклику ця функція очищає всі поточні віджети у вікні, а потім додає нові елементи інтерфейсу.

Створення класів товарів

2. Створення класу Product:

```
class Product:
    def __init__(self, name, price, quantity):
        self.name = name
        self.price = price
        self.quantity = quantity

# Створення списку товарів
products = [
    Product("Яблуко", 5.0, 10),
    Product("Хліб", 15.0, 20),
    Product("Молоко", 10.0, 15)
]

cart = []
```

- Клас Product: Цей клас представляє товар у магазині. Він має три атрибути: назва товару (name), ціна товару (price) та кількість товару на складі (quantity). Конструктор класу ініціалізує ці атрибути.
- Список товарів: products — це список об'єктів класу Product, що представляють товари, доступні в магазині.
- Кошик: cart — це список, який буде зберігати товари, додані до кошика.

Перегляд товарів

3. Функція перегляду товарів:

```
def view_products():
    for widget in root.winfo_children():
        widget.destroy()

    title = tk.Label(root, text="Список товарів", font=('Helvetica', 16))
    title.pack(pady=10)

    for product in products:
        product_label = tk.Label(root, text=f"{product.name} - {product.price} грн
(в наявності: {product.quantity})")
        product_label.pack()

        add_to_cart_btn = tk.Button(root, text="Додати в кошик",
command=lambda p=product: add_to_cart(p))
        add_to_cart_btn.pack(pady=5)

    back_btn = tk.Button(root, text="Назад", command=main_menu)
    back_btn.pack(pady=5)
```

- Функція view_products: Ця функція відображає всі товари у списку products.
    - Очищає всі поточні віджети у вікні.
    - Створює заголовок "Список товарів".
    - Для кожного товару у списку products створює мітку (Label), яка відображає назву, ціну та кількість товару.
    - Додає кнопку "Додати в кошик" для кожного товару, яка викликає функцію add_to_cart, передаючи їй поточний товар як параметр.
    - Додає кнопку "Назад", яка повертає користувача до головного меню.

Додавання товарів у кошик

4. Функція додавання товарів у кошик:

```
def add_to_cart(product):
    for item in cart:
        if item['product'].name == product.name:
            item['quantity'] += 1
            product.quantity -= 1
            update_product_list()
            return

    if product.quantity > 0:
        cart.append({'product': product, 'quantity': 1})
        product.quantity -= 1
        update_product_list()
    else:
        messagebox.showinfo("Помилка", "Цей товар закінчився на складі.")
```

- Функція add_to_cart: Ця функція додає товар у кошик.
    - Перевіряє, чи товар вже є у кошику. Якщо так, збільшує кількість товару в кошику та зменшує кількість товару на складі.
    - Якщо товару ще немає у кошику і він є в наявності, додає його у кошик з кількістю 1 та зменшує кількість товару на складі.
    - Якщо товар закінчився, виводить повідомлення про помилку.

5. Функція оновлення списку товарів:

```
def update_product_list():
    for widget in root.winfo_children():
        widget.destroy()
    view_products()
```

- Функція update_product_list: Очищає всі віджети у вікні та знову викликає функцію view_products для оновлення списку товарів після змін.

Перегляд кошика

6. Функція перегляду кошика:

```
def view_cart():
    for widget in root.winfo_children():
```

```
    widget.destroy()

  title = tk.Label(root, text="Ваш кошик", font=('Helvetica', 16))
  title.pack(pady=10)

  if not cart:
    empty_label = tk.Label(root, text="Ваш кошик порожній.")
    empty_label.pack()
  else:
    for item in cart:
      product = item['product']
      quantity = item['quantity']
      cart_item_label = tk.Label(root, text=f"{product.name} - {quantity} шт.
(ціна за одиницю: {product.price} грн)")
      cart_item_label.pack()

  back_btn = tk.Button(root, text="Назад", command=main_menu)
  back_btn.pack(pady=5)
```

- Функція view_cart: Ця функція відображає товари у кошику.
  - Очищає всі поточні віджети у вікні.
  - Створює заголовок "Ваш кошик".
  - Якщо кошик порожній, відображає повідомлення "Ваш кошик порожній".
  - Якщо у кошику є товари, для кожного товару створює мітку, яка відображає назву товару, кількість та ціну.
  - Додає кнопку "Назад", яка повертає користувача до головного меню.

Оформлення замовлення

7. Функція оформлення замовлення:

```
def checkout():
  for widget in root.winfo_children():
    widget.destroy()

  title = tk.Label(root, text="Оформлення замовлення", font=('Helvetica',
16))
  title.pack(pady=10)

  total = sum(item['product'].price * item['quantity'] for item in cart)
  total_label = tk.Label(root, text=f"Загальна сума: {total} грн")
  total_label.pack()
```

```
    confirm_btn = tk.Button(root, text="Підтвердити замовлення",
command=confirm_order)
    confirm_btn.pack(pady=5)

    back_btn = tk.Button(root, text="Назад", command=main_menu)
    back_btn.pack(pady=5)
```

- Функція checkout: Ця функція відображає екран оформлення замовлення.
  - Очищає всі поточні віджети у вікні.
  - Створює заголовок "Оформлення замовлення".
  - Обчислює загальну суму замовлення.
  - Відображає загальну суму.
  - Додає кнопку "Підтвердити замовлення", яка викликає функцію confirm_order.
  - Додає кнопку "Назад", яка повертає користувача до головного меню.

8. Функція підтвердження замовлення:

```
def confirm_order():
    global cart
    message
```

10. Практичне завдання «Створення віртуального саду»

Ініціалізація проекту

1. Створення основного файлу програми:

```
import tkinter as tk
from tkinter import messagebox

# Ініціалізація головного вікна програми
root = tk.Tk()
root.title("Віртуальний сад")

# Головне меню
def main_menu():
    for widget in root.winfo_children():
        widget.destroy()

    title = tk.Label(root, text="Віртуальний сад", font=('Helvetica', 16))
    title.pack(pady=10)

    plant_btn = tk.Button(root, text="Посадити нову рослину",
command=plant_new)
```

```
    plant_btn.pack(pady=5)

    water_btn = tk.Button(root, text="Полити рослину",
command=water_plant)
    water_btn.pack(pady=5)

    fertilize_btn = tk.Button(root, text="Внести добрива",
command=fertilize_plant)
    fertilize_btn.pack(pady=5)

    view_garden_btn = tk.Button(root, text="Переглянути сад",
command=view_garden)
    view_garden_btn.pack(pady=5)

main_menu()
root.mainloop()
```

- Ініціалізація головного вікна програми: Тут створюється основне вікно програми за допомогою бібліотеки tkinter.
- Головне меню: Функція main_menu створює головне меню з кнопками для посадки нової рослини, поливу рослини, внесення добрив та перегляду саду. Під час виклику ця функція очищає всі поточні віджети у вікні, а потім додає нові елементи інтерфейсу.

Створення класів рослин

2. Створення класу Plant:

```
class Plant:
    def __init__(self, name, water_need, fertilizer_need):
        self.name = name
        self.age = 0
        self.water_need = water_need
        self.fertilizer_need = fertilizer_need
        self.watered = 0
        self.fertilized = 0
        self.alive = True

    def grow(self):
        if self.watered >= self.water_need and self.fertilized >=
self.fertilizer_need:
            self.age += 1
```

```
            self.watered = 0
            self.fertilized = 0
        else:
            self.alive = False

    # Створення списку рослин
    garden = []
```

- Клас Plant: Цей клас представляє рослину у саду. Він має кілька атрибутів:
  - name: назва рослини.
  - age: вік рослини.
  - water_need: потреба рослини у воді.
  - fertilizer_need: потреба рослини у добривах.
  - watered: кількість води, яку отримала рослина.
  - fertilized: кількість добрив, які отримала рослина.
  - alive: стан рослини (жива або мертва).

  Клас також має метод grow, який збільшує вік рослини, якщо вона отримала достатньо води та добрив. Якщо рослина не отримала достатньо води або добрив, вона вмирає.
- Список рослин: garden — це список, який буде зберігати всі рослини у саду.

Посадка нової рослини

3. Функція посадки нової рослини:

```
def plant_new():
    for widget in root.winfo_children():
        widget.destroy()

    title = tk.Label(root, text="Посадити нову рослину", font=('Helvetica', 16))
    title.pack(pady=10)

    name_label = tk.Label(root, text="Назва рослини")
    name_label.pack(pady=5)
    name_entry = tk.Entry(root)
    name_entry.pack(pady=5)

    water_need_label = tk.Label(root, text="Потреба у воді")
```

```
water_need_label.pack(pady=5)
water_need_entry = tk.Entry(root)
water_need_entry.pack(pady=5)

fertilizer_need_label = tk.Label(root, text="Потреба у добривах")
fertilizer_need_label.pack(pady=5)
fertilizer_need_entry = tk.Entry(root)
fertilizer_need_entry.pack(pady=5)

def save_plant():
    name = name_entry.get()
    water_need = int(water_need_entry.get())
    fertilizer_need = int(fertilizer_need_entry.get())
    plant = Plant(name, water_need, fertilizer_need)
    garden.append(plant)
    main_menu()

save_btn = tk.Button(root, text="Посадити", command=save_plant)
save_btn.pack(pady=5)

back_btn = tk.Button(root, text="Назад", command=main_menu)
back_btn.pack(pady=5)
```

- Функція plant_new: Ця функція відображає форму для посадки нової рослини.
  - Очищає всі поточні віджети у вікні.
  - Створює заголовок "Посадити нову рослину".
  - Додає поля для введення назви рослини, потреби у воді та добривах.
  - Додає кнопку "Посадити", яка викликає внутрішню функцію save_plant, зберігаючи введені дані і додаючи нову рослину у список garden.
  - Додає кнопку "Назад", яка повертає користувача до головного меню.

Догляд за рослинами

4. Функція поливу рослин:

```
def water_plant():
    for widget in root.winfo_children():
```

```
        widget.destroy()

    title = tk.Label(root, text="Полити рослину", font=('Helvetica', 16))
    title.pack(pady=10)

    for plant in garden:
        if plant.alive:
            plant_label = tk.Label(root, text=f"{plant.name} - Вік: {plant.age},
Потреба у воді: {plant.water_need}, Полита: {plant.watered}")
            plant_label.pack()

            water_btn = tk.Button(root, text="Полити", command=lambda
p=plant: water(p))
            water_btn.pack(pady=5)

    back_btn = tk.Button(root, text="Назад", command=main_menu)
    back_btn.pack(pady=5)

def water(plant):
    plant.watered += 1
    water_plant()
```

- Функція water_plant: Ця функція відображає список всіх рослин у саду, які можна полити.
    - Очищає всі поточні віджети у вікні.
    - Створює заголовок "Полити рослину".
    - Для кожної живої рослини у списку garden створює мітку з інформацією про її вік, потребу у воді та кількість разів, коли вона була полита.
    - Додає кнопку "Полити" для кожної рослини, яка викликає функцію water, збільшуючи кількість поливів для відповідної рослини.
    - Додає кнопку "Назад", яка повертає користувача до головного меню.

5. Функція внесення добрив:

```
def fertilize_plant():
    for widget in root.winfo_children():
        widget.destroy()

    title = tk.Label(root, text="Внести добрива", font=('Helvetica', 16))
    title.pack(pady=10)
```

```
    for plant in garden:
        if plant.alive:
            plant_label = tk.Label(root, text=f"{plant.name} - Вік: {plant.age},
Потреба у добривах: {plant.fertilizer_need}, Добриво внесено:
{plant.fertilized}")
            plant_label.pack()

            fertilize_btn = tk.Button(root, text="Внести добрива",
command=lambda p=plant: fertilize(p))
            fertilize_btn.pack(pady=5)

    back_btn = tk.Button(root, text="Назад", command=main_menu)
    back_btn.pack(pady=5)

def fertilize(plant):
    plant.fertilized += 1
    fertilize_plant()
```

- Функція fertilize_plant: Ця функція відображає список всіх рослин у саду, які можна підживити.
  - Очищає всі поточні віджети у вікні.
  - Створює заголовок "Внести добрива".
  - Для кожної живої рослини у списку garden створює мітку з інформацією про її вік, потребу у добривах та кількість разів, коли вона була підживлена.
  - Додає кнопку "Внести добрива" для кожної рослини, яка викликає функцію fertilize, збільшуючи кількість підживлень для відповідної рослини.
  - Додає кнопку "Назад", яка повертає користувача до головного меню.

Перегляд саду

6. Функція перегляду саду:

```
def view_garden():
    for widget in root.winfo_children():
        widget.destroy()

    title = tk.Label(root, text="Переглянути сад", font=('Helvetica', 16))
```

```
    title.pack(pady=10)

    for plant in garden:
        plant_status = "Жива" if plant.alive else "Мертва"
        plant_label = tk.Label(root, text=f"{plant.name} - Вік: {plant.age}, Статус:
{plant_status}")
        plant_label.pack()

    back_btn = tk.Button(root, text="Назад", command=main_menu)
    back_btn.pack(pady=5)
```

- Функція view_garden: Ця функція відображає всі рослини у саду.
  - Очищає всі поточні віджети у вікні.
  - Створює заголовок "Переглянути сад".
  - Для кожної рослини у списку garden створює мітку з інформацією про її назву, вік та статус (жива або мертва).
  - Додає кнопку "Назад", яка повертає користувача до головного меню.

Це завдання дозволяє учням ознайомитися з основами об'єктно-орієнтованого програмування, графічним інтерфейсом користувача за допомогою tkinter, а також управлінням станом програми та взаємодією з користувачем.

11. завдання «Інтерактивний текстовий квест»

Опис завдання

Створимо інтерактивну текстову гру, де користувач може приймати рішення, що впливають на подальший хід подій. Гра матиме кілька різних сценаріїв з різними виборами та результатами.

Реалізація

1. Створення класу Scene для представлення сценарію:

```
class Scene:
    def __init__(self, title, description, choices=None):
        self.title = title
        self.description = description
        self.choices = choices if choices else {}

    def add_choice(self, choice, next_scene):
        self.choices[choice] = next_scene

    def get_choice(self, choice):
```

```
        return self.choices.get(choice)

    def __str__(self):
        return f"{self.title}\n{self.description}\n"

# Функція для введення користувача
def get_user_choice(scene):
    print(scene)
    choices = scene.choices.keys()
    for index, choice in enumerate(choices, start=1):
        print(f"{index}. {choice}")

    choice = input("Оберіть ваш варіант: ")
    return int(choice) if choice.isdigit() and int(choice) <= len(choices) else 0
```

- Опис: Клас Scene представляє окремий сценарій у грі з заголовком, описом та варіантами вибору (як словник). Методи add_choice() для додавання варіантів вибору та get_choice() для отримання наступної сцени за вибором користувача.

2. Створення сценаріїв для гри:

```
# Сцена 1
scene1 = Scene(
    "Сцена 1",
    "Ви опинилися в загадковому лісі. Перед вами розгалуження стежини.",
    {"Піти наліво": Scene("Ліва стежина", "Ви пішли ліворуч і виявили стародавній храм."),
     "Піти направо": Scene("Права стежина", "Ви пішли праворуч і натрапили на ведмедя.")}
)

# Сцена 2
scene2 = Scene(
    "Сцена 2",
    "Ви стоїте перед двома дверима. Одна червона, інша синя.",
    {"Відкрити червону двері": Scene("Зал скарбів", "Ви знайшли скарби та виграли гру!"),
     "Відкрити синю двері": Scene("Кабінет чаклунства", "Ви потрапили до чаклуна, він вас закляв.")}
)
```

```
# Додавання варіантів вибору до сцен
scene1.add_choice("Повернутися назад", scene1)
scene2.add_choice("Повернутися до початку", scene1)

# Початкова сцена
current_scene = scene1
```

- Опис сцен: Визначаються дві початкові сцени scene1 і scene2 з різними описами та варіантами вибору. Кожен вибір веде до нової сцени або до завершення гри.

3. Головний цикл гри:

```
# Головний цикл гри
while True:
    choice = get_user_choice(current_scene)
    if choice == 0:
        print("Невірний вибір. Гра завершується.")
        break

    choices = list(current_scene.choices.keys())
    selected_choice = choices[choice - 1]
    next_scene = current_scene.get_choice(selected_choice)

    if next_scene:
        current_scene = next_scene
    else:
        print("Невідомий вибір. Гра завершується.")
        break
```

- Опис: В головному циклі гри користувач може обирати варіанти зі списку. Вибір користувача перевіряється, і якщо він коректний, змінюється поточна сцена на наступну згідно з вибором. Гра продовжується до тих пір, поки користувач не зробить невірний вибір або не дійде до кінцевої сцени.

Це завдання демонструє створення інтерактивного текстового квесту з використанням класів, методів, умовних операторів та циклів. Воно дозволяє користувачеві взаємодіяти зі світом гри через текстовий інтерфейс, приймаючи рішення, які впливають на подальший хід подій.

12. Завдання «Генератор коміксів»

Опис завдання

Ми створимо програму, яка генерує комікси на основі випадкових комікснних стрічок. Кожен комікс буде складатися з трьох випадкових стрічок, які об'єднуються в одну картинку.

Реалізація

1. Створення класу ComicStrip для представлення комікснної стрічки:

```
import random

# Клас комікснної стрічки
class ComicStrip:
    def __init__(self, strip):
        self.strip = strip

    def __str__(self):
        return "\n".join(self.strip)

# Функція для генерації коміксів
def generate_comic():
    beginnings = [
        ["Кішка сідає на стіл."],
        ["Собака грає з м'ячем."],
        ["Лисиця крадеться вночі."],
        ["Птахи співають на сходах."],
        ["Мавпа стрибає на гілці."],
    ]
    middles = [
        ["Однак вона випадково впадає у відро!"],
        ["Але раптом починається дощ!"],
        ["І зустрічає іншу лисицю."],
        ["Вони радіють ранковому сонцю."],
        ["Та піднімається високо!"],
    ]
    ends = [
        ["Кішка вибігає з нього, обливаючись водою."],
        ["Собака радіє і бігає додому."],
        ["Та разом вони грають у відкритому полі."],
        ["Птахи співають у зеленому лісі."],
        ["І всім це дуже подобається!"],
    ]
```

```
    beginning = random.choice(beginnings)
    middle = random.choice(middles)
    end = random.choice(ends)

    comic_strip = ComicStrip(beginning + middle + end)
    return comic_strip
```

- Опис: Клас ComicStrip приймає список стрічок і об'єднує їх в одну стрічку для представлення коміксу. Функція generate_comic() містить списки з можливими початками, серединами та кінцями комікснних стрічок. З використанням random.choice() вибираються випадкові стрічки з кожного списку для створення випадкового коміксу.

2. Тестування генератора коміксів:

```
# Генерація та виведення декількох випадкових коміксів
for _ in range(5):
    comic = generate_comic()
    print(comic)
    print("-" * 20)
```

- Опис: У цьому фрагменті коду ми генеруємо і виводимо п'ять випадкових коміксів за допомогою функції generate_comic(). Кожен комікс складається з трьох випадкових стрічок і виводиться у вигляді тексту, розділеного дефісами.

Завершення

Це завдання демонструє творчий підхід до використання випадкових елементів для створення нових творінь. Воно дозволяє відчути та відтворити характерні риси коміксів, використовуючи мову програмування Python.

5. Завдання та практичні роботи по темі "Алгоритми та програми" з мови програмування Python для 9 класу.

1. Шифрування та дешифрування Цезаря

Опис завдання

Створіть програму для шифрування та дешифрування тексту за допомогою шифру Цезаря. Це класичний метод шифрування, де кожна буква тексту замінюється на іншу згідно зі зсувом ключа.

Реалізація

```
def caesar_cipher(text, shift):
    encrypted_text = ""
    for char in text:
        if char.isalpha():  # перевіряємо, чи символ є буквою
            shifted_char = chr((ord(char.lower()) - 97 + shift) % 26 + 97)
            encrypted_text += shifted_char.upper() if char.isupper() else shifted_char
        else:
            encrypted_text += char
    return encrypted_text

def caesar_decipher(text, shift):
    return caesar_cipher(text, -shift)

# Приклад використання
plaintext = "Hello, World!"
shift = 3

encrypted_text = caesar_cipher(plaintext, shift)
decrypted_text = caesar_decipher(encrypted_text, shift)

print("Оригінальний текст:", plaintext)
print("Зашифрований текст:", encrypted_text)
print("Дешифрований текст:", decrypted_text)
```

- Опис: У цій програмі функції caesar_cipher та caesar_decipher використовуються для шифрування та дешифрування тексту відповідно за допомогою шифру Цезаря з заданим зсувом. В програмі також виводяться оригінальний текст, зашифрований текст та дешифрований текст.

2. Система керування списком завдань

Опис завдання

Створіть програму для керування списком завдань. Користувач може додавати, видаляти та виводити завдання. Кожне завдання має мати опис і статус (виконано чи ні).

Реалізація

```
class Task:
    def __init__(self, description):
        self.description = description
        self.completed = False

    def complete(self):
        self.completed = True

    def __str__(self):
        status = "Завершено" if self.completed else "Не завершено"
        return f"{self.description} - {status}"

class TaskManager:
    def __init__(self):
        self.tasks = []

    def add_task(self, task):
        self.tasks.append(task)

    def remove_task(self, index):
        if 0 <= index < len(self.tasks):
            del self.tasks[index]
        else:
            print("Невірний індекс завдання.")

    def display_tasks(self):
        if not self.tasks:
            print("Список завдань порожній.")
        else:
            for index, task in enumerate(self.tasks, start=1):
                print(f"{index}. {task}")

# Приклад використання
task_manager = TaskManager()
```

```
task1 = Task("Написати програму для керування завданнями.")
task2 = Task("Підготувати презентацію з алгоритмів.")
task3 = Task("Вивчити новий розділ з математики.")

task_manager.add_task(task1)
task_manager.add_task(task2)
task_manager.add_task(task3)

task_manager.display_tasks()

task1.complete()

task_manager.remove_task(1)

print("Оновлений список завдань:")
task_manager.display_tasks()
```

- Опис: У цьому прикладі створюються класи Task для представлення окремого завдання з описом і статусом, і TaskManager для керування списком завдань. Користувач може додавати нові завдання, виконувати їх, видаляти та виводити усі завдання.

3. Симуляція роботи черги

Опис завдання

Реалізуйте програму для симуляції роботи черги. Користувач може додавати елементи у чергу, видаляти їх та виводити поточний стан черги.

Реалізація

```
class Queue:
    def __init__(self):
        self.items = []

    def is_empty(self):
        return len(self.items) == 0

    def enqueue(self, item):
        self.items.append(item)

    def dequeue(self):
        if not self.is_empty():
```

```
            return self.items.pop(0)
        else:
            return None

    def peek(self):
        if not self.is_empty():
            return self.items[0]
        else:
            return None

    def display(self):
        if not self.is_empty():
            print("Елементи у черзі:", self.items)
        else:
            print("Черга порожня.")

# Приклад використання
queue = Queue()

queue.enqueue("Елемент 1")
queue.enqueue("Елемент 2")
queue.enqueue("Елемент 3")

queue.display()

queue.dequeue()
queue.display()

print("Перший елемент у черзі:", queue.peek())
```

- Опис: У цьому прикладі створюється клас Queue для представлення черги з методами для додавання (enqueue), видалення (dequeue) елементів та перегляду першого елемента (peek). Користувач може додавати елементи, видаляти їх та виводити поточний стан черги.

4. Шифрування та дешифрування з використанням шифру Перестановки

Опис завдання

Реалізуйте програму для шифрування та дешифрування повідомлень з використанням шифру Перестановки. Користувач повинен мати можливість вводити текст для шифрування та задавати ключ для зміщення символів.

Реалізація

```
import math

def encrypt_message(message, key):
    # Видаляємо пробіли з повідомлення
    message = message.replace(" ", "")
    # Обчислюємо кількість рядків для таблиці
    rows = int(math.ceil(len(message) / float(key)))
    # Заповнюємо таблицю символами
    table = [''] * rows
    row, col = 0, 0

    for symbol in message:
        table[row] += symbol
        row += 1
        if row == rows or (row == rows - 1 and col >= key - 1):
            row, col = 0, col + 1

    # Перетворюємо таблицю в зашифрований текст
    encrypted_text = ''.join(table)
    return encrypted_text

def decrypt_message(encrypted_text, key):
    # Обчислюємо кількість рядків для таблиці
    rows = int(math.ceil(len(encrypted_text) / float(key)))
    # Знаходимо кількість заповнених клітинок у кожному рядку
    cols = int(math.ceil(len(encrypted_text) / float(rows)))
    # Розраховуємо кількість пустих клітинок
    empty_cells = rows * cols - len(encrypted_text)

    # Розбиваємо зашифроване повідомлення на рядки
    table = [''] * rows
    row, col = 0, 0

    for symbol in encrypted_text:
        table[row] += symbol
        row += 1
        if (row == rows) or (row == rows - 1 and col >= cols - empty_cells):
```

```
            row, col = 0, col + 1

    # Зчитуємо рядки таблиці для отримання дешифрованого тексту
    decrypted_text = ''.join(table)
    return decrypted_text

# Приклад використання
message = "Це секретне повідомлення"
key = 4

encrypted_message = encrypt_message(message, key)
print(f"Зашифроване повідомлення: {encrypted_message}")

decrypted_message = decrypt_message(encrypted_message, key)
print(f"Дешифроване повідомлення: {decrypted_message}")
```

- Опис: У цій програмі функції encrypt_message та decrypt_message використовуються для шифрування та дешифрування повідомлення за допомогою шифру Перестановки з використанням заданого ключа. Повідомлення розбивається на рядки у таблиці, а потім перетворюється у зашифроване чи дешифроване повідомлення відповідно.

5. Симуляція квитків на кіносеанси

Опис завдання

Створіть програму для управління квитками на кіносеанси, яка дозволяє купувати, резервувати та скасовувати квитки для різних кіносеансів. Програма повинна підтримувати різні фільми, час кіносеансу, кількість місць та інші деталі.

Реалізація

```
class MovieTicket:
    def __init__(self, movie_title, show_time, num_seats):
        self.movie_title = movie_title
        self.show_time = show_time
        self.num_seats = num_seats
        self.available_seats = num_seats
        self.reserved_seats = 0

    def reserve_seats(self, num_seats):
        if self.available_seats >= num_seats:
            self.available_seats -= num_seats
```

```
            self.reserved_seats += num_seats
            return True
        else:
            return False

    def cancel_reservation(self, num_seats):
        if self.reserved_seats >= num_seats:
            self.available_seats += num_seats
            self.reserved_seats -= num_seats
            return True
        else:
            return False

    def __str__(self):
        return f"{self.movie_title} - {self.show_time}, Залишилось місць:
{self.available_seats}/{self.num_seats}"

class Cinema:
    def __init__(self, name):
        self.name = name
        self.movies = {}

    def add_movie(self, movie_title, show_time, num_seats):
        movie_key = (movie_title, show_time)
        if movie_key not in self.movies:
            self.movies[movie_key] = MovieTicket(movie_title, show_time, num_seats)
        else:
            print(f"Фільм '{movie_title}' на час {show_time} вже існує.")

    def reserve_tickets(self, movie_title, show_time, num_seats):
        movie_key = (movie_title, show_time)
        if movie_key in self.movies:
            movie_ticket = self.movies[movie_key]
            if movie_ticket.reserve_seats(num_seats):
                print(f"Квитки на '{movie_title}' на час {show_time} успішно
зарезервовані.")
            else:
```

```
            print(f"На жаль, не вдалося зарезервувати {num_seats} квитків на '{movie_title}' на час {show_time}.")
        else:
            print(f"Фільм '{movie_title}' на час {show_time} не знайдено.")

    def cancel_reservation(self, movie_title, show_time, num_seats):
        movie_key = (movie_title, show_time)
        if movie_key in self.movies:
            movie_ticket = self.movies[movie_key]
            if movie_ticket.cancel_reservation(num_seats):
                print(f"Резервацію {num_seats} квитків на '{movie_title}' на час {show_time} скасовано.")
            else:
                print(f"На жаль, не вдалося скасувати резервацію {num_seats} квитків на '{movie_title}' на час {show_time}.")
        else:
            print(f"Фільм '{movie_title}' на час {show_time} не знайдено.")

    def display_movies(self):
        if not self.movies:
            print("Наразі немає сеансів у кінотеатрі.")
        else:
            for key, movie_ticket in self.movies.items():
                print(movie_ticket)

# Приклад використання
cinema = Cinema("Мегаплекс")

cinema.add_movie("Прокляття Ла Ллорони", "18:00", 100)
cinema.add_movie("Чорна вдова", "20:00", 80)
cinema.add_movie("Шазам!", "17:30", 120)

cinema.display_movies()

cinema.reserve_tickets("Прокляття Ла Ллорони", "18:00", 2)
cinema.reserve_tickets("Чорна вдова", "20:00", 5)

cinema.display_movies()
```

```
cinema.cancel_reservation("Чорна вдова", "20:00", 3)

cinema.display_movies()
```

- Опис: У цій програмі створюються класи MovieTicket для представлення квитків на фільми з можливістю резервування та скасування місць, і Cinema для управління кінотеатром з функціями для додавання фільмів, резервування та скасування квитків. Користувач може переглядати фільми, резервувати квитки та скасовувати резервації.

6. Менеджер завдань (To-Do List)

Опис завдання

Створіть програму для управління задачами (To-Do List), яка дозволяє користувачеві додавати, видаляти та відмічати виконані задачі. Програма повинна зберігати список задач у файлі та підтримувати базове взаємодію через консоль.

Реалізація

```
class ToDoList:
    def __init__(self):
        self.tasks = []

    def load_tasks(self, filename):
        try:
            with open(filename, 'r', encoding='utf-8') as file:
                self.tasks = [line.strip() for line in file.readlines()]
            print("Завдання успішно завантажено.")
        except FileNotFoundError:
            print("Файл не знайдено. Створіть новий список завдань.")

    def save_tasks(self, filename):
        with open(filename, 'w', encoding='utf-8') as file:
            file.write('\n'.join(self.tasks))
        print("Завдання успішно збережено.")

    def add_task(self, task):
        self.tasks.append(task)
        print(f"Завдання '{task}' додано до списку.")

    def delete_task(self, index):
```

```
        if 0 <= index < len(self.tasks):
            deleted_task = self.tasks.pop(index)
            print(f"Завдання '{deleted_task}' видалено зі списку.")
        else:
            print("Некоректний індекс завдання.")

    def mark_task_done(self, index):
        if 0 <= index < len(self.tasks):
            self.tasks[index] = f"{self.tasks[index]} - виконано"
            print(f"Завдання '{self.tasks[index]}' позначено як виконане.")
        else:
            print("Некоректний індекс завдання.")

    def display_tasks(self):
        if not self.tasks:
            print("Список завдань порожній.")
        else:
            for i, task in enumerate(self.tasks):
                print(f"{i + 1}. {task}")

# Приклад використання
todo_list = ToDoList()

todo_list.load_tasks("tasks.txt")

while True:
    print("\nМеню:")
    print("1. Показати список завдань")
    print("2. Додати завдання")
    print("3. Видалити завдання")
    print("4. Позначити завдання як виконане")
    print("5. Зберегти список завдань")
    print("6. Вийти з програми")

    choice = input("Виберіть дію: ")

    if choice == '1':
        todo_list.display_tasks()
```

```
    elif choice == '2':
        task = input("Введіть нове завдання: ")
        todo_list.add_task(task)
    elif choice == '3':
        index = int(input("Введіть індекс завдання для видалення: ")) - 1
        todo_list.delete_task(index)
    elif choice == '4':
        index = int(input("Введіть індекс завдання для позначення як виконане: ")) - 1
        todo_list.mark_task_done(index)
    elif choice == '5':
        todo_list.save_tasks("tasks.txt")
    elif choice == '6':
        print("Програма завершена.")
        break
    else:
        print("Некоректний вибір. Спробуйте ще раз.")
```

- Опис: У цій програмі клас ToDoList дозволяє користувачеві додавати, видаляти, позначати як виконані та зберігати список завдань у файлі. Користувач може взаємодіяти з програмою через консоль, виконуючи різні операції зі своїм списком завдань.

7. Простий текстовий редактор

Опис завдання

Створіть простий текстовий редактор з базовими функціями, такими як введення тексту, редагування, збереження та завантаження файлів. Програма повинна мати можливість відображення рядків та стовпців тексту, а також змінювати шрифт та кольори тексту.

Реалізація

```
class TextEditor:
    def __init__(self):
        self.text = ""
        self.filename = ""

    def load_file(self, filename):
        try:
            with open(filename, 'r', encoding='utf-8') as file:
                self.text = file.read()
            self.filename = filename
            print(f"Файл '{filename}' успішно завантажено.")
```

```
        except FileNotFoundError:
            print("Файл не знайдено. Створіть новий файл.")

    def save_file(self):
        if self.filename:
            with open(self.filename, 'w', encoding='utf-8') as file:
                file.write(self.text)
            print(f"Файл '{self.filename}' успішно збережено.")
        else:
            print("Файл не був раніше збережений. Використовуйте 'Save As' для
збереження.")

    def edit_text(self):
        print("Редагування тексту:")
        print(self.text)
        print("Для завершення редагування введіть ':q' на новому рядку.")

        while True:
            line = input()
            if line == ":q":
                break
            self.text += line + '\n'

    def display_text(self):
        print(self.text)

# Приклад використання
text_editor = TextEditor()

print("1. Новий файл")
print("2. Завантажити файл")

choice = input("Виберіть опцію: ")

if choice == '1':
    text_editor.edit_text()
elif choice == '2':
    filename = input("Введіть ім'я файлу для завантаження: ")
```

```
    text_editor.load_file(filename)

text_editor.display_text()

text_editor.save_file()
```

- Опис: У цій програмі клас TextEditor дозволяє користувачеві створювати нові текстові файли, завантажувати їх для редагування, а також зберігати зміни у вибраний файл. Користувач може взаємодіяти з програмою через консоль, використовуючи команди для введення, редагування та збереження тексту.

8. Система управління продуктами у магазині

Опис завдання

Створіть програму для управління продуктами у магазині. Програма повинна мати можливість додавання нових продуктів, видалення продуктів, зміни інформації про продукти, відображення інформації про всі продукти та збереження цієї інформації у файлі.

Реалізація

```
class Product:
    def __init__(self, id, name, price, quantity):
        self.id = id
        self.name = name
        self.price = price
        self.quantity = quantity

    def __str__(self):
        return f"ID: {self.id}, Назва: {self.name}, Ціна: {self.price}, Кількість:
{self.quantity}"

class StoreManager:
    def __init__(self):
        self.products = []

    def add_product(self, product):
        self.products.append(product)
        print(f"Продукт '{product.name}' доданий до магазину.")

    def remove_product(self, product_id):
        for product in self.products:
```

```
            if product.id == product_id:
                self.products.remove(product)
                print(f"Продукт з ID {product_id} видалений з магазину.")
                return
        print(f"Продукт з ID {product_id} не знайдений у магазині.")

    def update_product(self, product_id, new_name, new_price, new_quantity):
        for product in self.products:
            if product.id == product_id:
                product.name = new_name
                product.price = new_price
                product.quantity = new_quantity
                print(f"Інформація про продукт з ID {product_id} оновлена.")
                return
        print(f"Продукт з ID {product_id} не знайдений у магазині для оновлення.")

    def display_all_products(self):
        if self.products:
            print("Список усіх продуктів у магазині:")
            for product in self.products:
                print(product)
        else:
            print("У магазині немає жодного продукту.")

    def save_to_file(self, filename):
        with open(filename, 'w', encoding='utf-8') as file:
            for product in self.products:

file.write(f"{product.id},{product.name},{product.price},{product.quantity}\n")
        print("Інформацію про продукти збережено у файлі.")

    def load_from_file(self, filename):
        try:
            with open(filename, 'r', encoding='utf-8') as file:
                lines = file.readlines()
                for line in lines:
                    data = line.strip().split(',')
                    id = int(data[0])
```

```
                name = data[1]
                price = float(data[2])
                quantity = int(data[3])
                self.products.append(Product(id, name, price, quantity))
            print("Інформація про продукти завантажена з файлу.")
        except FileNotFoundError:
            print("Файл не знайдено. Створіть новий список продуктів.")

# Приклад використання
store_manager = StoreManager()

# Завантаження даних з файлу (якщо вже є база)
store_manager.load_from_file("products.txt")

while True:
    print("\nМеню:")
    print("1. Додати продукт")
    print("2. Видалити продукт")
    print("3. Оновити інформацію про продукт")
    print("4. Показати інформацію про всі продукти")
    print("5. Зберегти список продуктів у файл")
    print("6. Вийти з програми")

    choice = input("Виберіть дію: ")

    if choice == '1':
        id = int(input("Введіть ID продукту: "))
        name = input("Введіть назву продукту: ")
        price = float(input("Введіть ціну продукту: "))
        quantity = int(input("Введіть кількість продукту: "))
        product = Product(id, name, price, quantity)
        store_manager.add_product(product)
    elif choice == '2':
        product_id = int(input("Введіть ID продукту для видалення: "))
        store_manager.remove_product(product_id)
    elif choice == '3':
        product_id = int(input("Введіть ID продукту для оновлення: "))
        new_name = input("Введіть нову назву продукту: ")
```

```
            new_price = float(input("Введіть нову ціну продукту: "))
            new_quantity = int(input("Введіть нову кількість продукту: "))
            store_manager.update_product(product_id, new_name, new_price,
new_quantity)
        elif choice == '4':
            store_manager.display_all_products()
        elif choice == '5':
            store_manager.save_to_file("products.txt")
        elif choice == '6':
            print("Програма завершена.")
            break
        else:
            print("Некоректний вибір. Спробуйте ще раз.")
```

- Опис: У цій програмі клас Product використовується для представлення кожного продукту у магазині (з ID, назвою, ціною та кількістю). Клас StoreManager управляє списком продуктів, дозволяючи додавати, видаляти, оновлювати та виводити інформацію про продукти, а також зберігати цю інформацію у файлі. Користувач може взаємодіяти з програмою через консоль, виконуючи різні операції зі списком продуктів у магазині.

9. Система управління замовленнями в ресторані

Опис завдання

Створіть просту систему управління замовленнями для ресторану у консольному інтерфейсі. Програма повинна дозволяти додавати нові замовлення, видаляти існуючі, переглядати список поточних замовлень та зберігати їх у файлі.

Реалізація

```
class Order:
    def __init__(self, order_id, table_number, items):
        self.order_id = order_id
        self.table_number = table_number
        self.items = items

    def __str__(self):
        return f"Замовлення ID {self.order_id} для столика {self.table_number}: {',
'.join(self.items)}"

class RestaurantManager:
```

```
    def __init__(self):
        self.orders = []

    def add_order(self, order):
        self.orders.append(order)
        print(f"Замовлення ID {order.order_id} додане.")

    def remove_order(self, order_id):
        for order in self.orders:
            if order.order_id == order_id:
                self.orders.remove(order)
                print(f"Замовлення ID {order_id} видалене.")
                return
        print(f"Замовлення ID {order_id} не знайдене.")

    def display_orders(self):
        if self.orders:
            print("Список поточних замовлень:")
            for order in self.orders:
                print(order)
        else:
            print("Наразі немає активних замовлень.")

    def save_to_file(self, filename):
        with open(filename, 'w', encoding='utf-8') as file:
            for order in self.orders:

file.write(f"{order.order_id},{order.table_number},{','.join(order.items)}\n")
        print("Замовлення збережені у файлі.")

    def load_from_file(self, filename):
        try:
            with open(filename, 'r', encoding='utf-8') as file:
                lines = file.readlines()
                for line in lines:
                    data = line.strip().split(',')
                    order_id = int(data[0])
                    table_number = int(data[1])
```

```
                items = data[2:]
                self.orders.append(Order(order_id, table_number, items))
            print("Замовлення завантажені з файлу.")
        except FileNotFoundError:
            print("Файл не знайдено. Створіть новий список замовлень.")

# Приклад використання
restaurant_manager = RestaurantManager()

# Завантаження даних з файлу (якщо вже є список замовлень)
restaurant_manager.load_from_file("orders.txt")

while True:
    print("\nМеню:")
    print("1. Додати нове замовлення")
    print("2. Видалити замовлення за ID")
    print("3. Показати всі поточні замовлення")
    print("4. Зберегти список замовлень у файл")
    print("5. Вийти з програми")

    choice = input("Виберіть дію: ")

    if choice == '1':
        order_id = int(input("Введіть ID замовлення: "))
        table_number = int(input("Введіть номер столика: "))
        items = input("Введіть список товарів через кому: ").split(',')
        order = Order(order_id, table_number, items)
        restaurant_manager.add_order(order)
    elif choice == '2':
        order_id = int(input("Введіть ID замовлення для видалення: "))
        restaurant_manager.remove_order(order_id)
    elif choice == '3':
        restaurant_manager.display_orders()
    elif choice == '4':
        restaurant_manager.save_to_file("orders.txt")
    elif choice == '5':
        print("Програма завершена.")
        break
```

```
    else:
        print("Некоректний вибір. Спробуйте ще раз.")
```

Пояснення

1. Клас Order: Відображає замовлення в ресторані з унікальним ID, номером столика і списком товарів.
2. Клас RestaurantManager: Управляє списком замовлень. Має методи для додавання нових замовлень (add_order), видалення замовлень за ID (remove_order), відображення поточних замовлень (display_orders) і збереження замовлень у файл (save_to_file). Метод load_from_file завантажує існуючі замовлення з файлу.
3. Головний цикл програми: Користувач може взаємодіяти з програмою через консоль, вибираючи операції з меню. Для кожної операції є відповідний блок коду, що викликає методи класу RestaurantManager.
4. Збереження у файл: Після завершення програми дані зберігаються у файл orders.txt. При наступному запуску програми можна завантажити попередні замовлення з цього файлу.

10. Керування списком контактів

Опис завдання

Створіть програму для керування списком контактів. Програма повинна дозволяти додавати нові контакти, видаляти їх, редагувати інформацію про контакти та здійснювати пошук контактів за іменем.

Реалізація

```
class Contact:
    def __init__(self, name, phone, email):
        self.name = name
        self.phone = phone
        self.email = email

    def __str__(self):
        return f"{self.name}: {self.phone}, {self.email}"

class ContactList:
    def __init__(self):
        self.contacts = []

    def add_contact(self, contact):
        self.contacts.append(contact)
        print(f"Контакт {contact.name} доданий до списку.")
```

```
def remove_contact(self, name):
    removed = False
    for contact in self.contacts:
        if contact.name.lower() == name.lower():
            self.contacts.remove(contact)
            print(f"Контакт {contact.name} видалений зі списку.")
            removed = True
    if not removed:
        print(f"Контакт з іменем {name} не знайдений у списку.")

def edit_contact(self, name, phone, email):
    found = False
    for contact in self.contacts:
        if contact.name.lower() == name.lower():
            contact.phone = phone
            contact.email = email
            print(f"Інформація про контакт {contact.name} оновлена.")
            found = True
    if not found:
        print(f"Контакт з іменем {name} не знайдений у списку.")

def search_contact(self, name):
    found = False
    for contact in self.contacts:
        if contact.name.lower() == name.lower():
            print(f"Знайдено контакт: {contact}")
            found = True
    if not found:
        print(f"Контакт з іменем {name} не знайдений у списку.")

def display_all_contacts(self):
    if self.contacts:
        print("Список контактів:")
        for contact in self.contacts:
            print(contact)
    else:
        print("Список контактів порожній.")
```

```
# Приклад використання
contact_list = ContactList()

# Додавання контактів
contact_list.add_contact(Contact("Іван Петров", "+380991234567",
"ivan@email.com"))
contact_list.add_contact(Contact("Марія Сидорова", "+380997654321",
"maria@email.com"))

# Видалення контакту
contact_list.remove_contact("Іван Петров")

# Редагування контакту
contact_list.edit_contact("Марія Сидорова", "+380991234567",
"maria.new@email.com")

# Пошук контакту
contact_list.search_contact("Марія Сидорова")

# Виведення усіх контактів
contact_list.display_all_contacts()
```

Пояснення

1. Клас Contact: Представляє контакт з основними атрибутами: ім'я, номер телефону і електронна пошта.
2. Клас ContactList: Управляє списком контактів. Має методи для додавання нових контактів (add_contact), видалення контактів за іменем (remove_contact), редагування інформації про контакти (edit_contact), пошуку контактів за іменем (search_contact) та виведення усіх контактів у списку (display_all_contacts).
3. Головний блок коду: Ілюструє використання різних методів класу ContactList для додавання, видалення, редагування, пошуку та виведення контактів.

ВИСНОВОК

Розвиток логічного мислення та алгоритмічного підходу**:** Програмування вчить учнів розуміти логіку роботи програм, декомпозиції складних завдань на менші частини та розробки алгоритмів для їх вирішення.

Стимулювання креативності та творчого мислення**:** Scratch дозволяє учням створювати власні інтерактивні ігри, анімації та інші проекти, що сприяє розвитку їх креативних здібностей.

Підготовка до цифрової грамотності**:** Вивчення програмування дозволяє учням зрозуміти, як працюють комп'ютери та як їх використовувати для розв'язання реальних проблем.

Підготовка до майбутньої професійної діяльності**:** Python, як більш продвинута мова програмування, дозволяє глибше вивчати принципи програмування та готує учнів до подальшого вивчення ІТ-технологій.

Збільшення конкурентоспроможності**:** Оволодіння навичками програмування відкриває учням нові можливості у навчанні, кар'єрному розвитку та особистому рості, збільшуючи їх конкурентоспроможність на ринку праці.

Ці завдання допоможуть учням краще зрозуміти, як працюють алгоритми та як можна їх використовувати для розв'язання практичних завдань. Вони також сприяють розвитку навичок програмування та логічного мислення.

Вивчення мов програмування Scratch та Python у 5-9 класах є важливою складовою освіти, що надає учням не лише технічні навички, а й розвиває креативність, критичне мислення та підготовлює до активного участі у сучасному інформаційному суспільстві.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Навчальна програма з інформатики для 5-9 класів, затверджена Наказом Міністерства освіти і науки України від 07.06.2017 № 804