Практические задания для первого блока по базовому курсу по Python

Предложенные практические задания помогут студентам попрактиковаться в различных аспектах и конструкциях языка программирования Python, включая работу с текстовыми файлами, обработку исключений, использование различных структур данных, условий и логирования, работа с модулями.

Задание 1: Анализ данных из текстового файла

Условие:

У вас есть текстовый файл `data.txt`, содержащий информацию о продажах товаров. Каждая строка файла содержит три поля, разделенных запятыми: название товара, количество проданных единиц и цену за единицу. Пример строки:

Товар1,10,99.99

Товар2,5,149.50

Требования:

1. Импорт модулей:
   - Импортируйте необходимый модуль `logging`.
2. Чтение из файла:
   - Считайте данные из `data.txt` с помощью контекстного менеджера `with`.
3. Обработка исключений:
   - Используйте конструкцию `try-except` для обработки возможных ошибок при чтении файла.
4. Логирование:
   - Запишите в лог файл информацию об успешно считанных данных и об ошибках чтения файла.
5. Обработка данных:
   - Подсчитайте общую выручку по каждому товару и общий доход.
6. Использование функций:
   - Оформите подсчет выручки в виде функции.
7. Работа со структурами данных:
   - Используйте списки и словари для хранения и обработки данных.
8. Вывод данных:
   - Выведите таблицу с названием товара, количеством, ценой и общей выручкой по товару.
9. Логические операторы и условные конструкции:

- Если общий доход превышает определенное значение, выведите сообщение о высоких продажах.

10. Форматирование строк:

- Используйте форматирование строк для красивого вывода.

Пример содержимого файла "data.txt":

Телевизор,5,29999.99

Холодильник,2,49999.50

Смартфон,10,19999.00

Ноутбук,3,69999.99

Пылесос,7,8999.90

Стиральная машина,4,35999.95

Микроволновка,8,7999.99

Кофеварка,15,4999.50

Задание 2: Обработка данных из текстового файла

Условие:

Дан текстовый файл `students.txt`, содержащий информацию о студентах и их оценках по разным предметам. Каждая строка файла содержит имя студента и его оценки, разделенные запятыми. Пример строки:

```
Иван Иванов,5,4,3,5
Мария Петрова,4,4,5,5
```

Требования:

1. Чтение из файла:
   - Откройте и считайте данные из `students.txt` с помощью контекстного менеджера `with`.
2. Обработка исключений:
   - Обработайте возможные исключения при чтении файла.
3. Создание функций:
   - Напишите функцию, которая рассчитывает средний балл каждого студента.
4. Использование циклов:
   - Используйте циклы `for` для перебора студентов и их оценок.
5. Логические операторы и условные конструкции:
   - Определите студентов, у которых средний балл выше определенного порога.
6. Запись в файл:
   - Запишите результаты в новый текстовый файл `top_students.txt`.
7. Использование списков и словарей:
   - Организуйте данные в удобные для обработки структуры.
8. Форматирование строк:
   - Выведите информацию о каждом студенте в читабельном формате.
9. Логирование:
   - Запишите в лог-файл информацию о количестве студентов и об успешно завершенной операции.
10. Импорт модулей:
    - Импортируйте модуль `logging`.

Пример содержимого файла `students.txt`:

```
Алексей Смирнов,5,4,3,5,4
Елена Кузнецова,4,4,5,5,5
Дмитрий Попов,3,3,4,2,3
```

Анна Соколова,5,5,5,5,5

Игорь Лебедев,2,3,3,2,4

Мария Козлова,4,5,4,5,5

Сергей Новиков,5,4,5,5,4

Наталья Морозова,3,4,3,4,3

Задание 3: Создание простой базы данных в текстовом файле

Условие:

Разработайте программу для ведения списка задач (To-Do List), которые сохраняются в текстовый файл `tasks.txt`.

Требования:

1. Создание функций:
   - Реализуйте функции для добавления, удаления и отображения задач.
2. Обработка исключений:
   - Обработайте возможные ошибки при работе с файлом и некорректным вводом пользователя.
3. Использование циклов:
   - Организуйте меню для взаимодействия с пользователем в цикле `while`.
4. Запись и чтение из файла:
   - При добавлении или удалении задач обновляйте `tasks.txt`.
5. Работа с контекстным менеджером:
   - Используйте `with` для открытия файла.
6. Использование списков:
   - Храните задачи в списке для удобной обработки.
7. Логические операторы и условные конструкции:
   - Реализуйте проверку наличия задачи перед удалением.
8. Логирование:
   - Записывайте в лог-файл действия пользователя и ошибки.
9. Форматирование строк:
   - Выводите задачи в формате нумерованного списка.
10. Импорт модулей:
    - Импортируйте модуль `logging`.

Пример содержимого файла `tasks.txt`:

Купить продукты на ужин

Позвонить в больницу и записаться к врачу

Подготовить и отправить отчет директору

Сходить в спортзал – "качаем ноги"

Прочитать книгу "Понедельник начинается в субботу"

Очередной раз попытаться написать программу для решения "всех задач"

Убрать квартиру и приготовить ужин

Встретиться с друзьями и поиграть в настолки

Задание 4: Обработка данных о сотрудниках

Условие:

У вас есть файл `employees.txt`, содержащий информацию о сотрудниках: имя, возраст, должность и зарплата, разделенные запятыми. Пример строки:

Иван Иванов,30,Инженер,70000

Мария Петрова,28,Аналитик,80000

Необходимо обработать эти данные и вывести список сотрудников, чья зарплата выше среднего.

Требования:

1. Чтение из файла:
   - Считайте данные из `employees.txt` с помощью `with`.
2. Обработка исключений:
   - Обработайте ошибки при чтении файла и некорректном формате данных.
3. Создание функций:
   - Напишите функцию `filter_employees(data)`, возвращающую отфильтрованный список.
4. Использование списков и словарей:
   - Храните данные в удобных для обработки структурах.
5. Вычисления с числами:
   - Рассчитайте среднюю зарплату.
6. Логические операторы и условные конструкции:
   - Сравните зарплату каждого сотрудника со средней.
7. Запись в файл:
   - Запишите результаты в `high_earners.txt`.
8. Логирование:
   - Логируйте количество сотрудников и результаты фильтрации.
9. Форматирование строк:
   - Выведите информацию о каждом сотруднике в формате: "Имя - Должность - Зарплата".
10. Импорт модулей:
    - Импортируйте модуль `logging`.

Пример содержимого файла `employees.txt`:

Иван Иванов,35,Инженер,70000

Мария Петрова,28,Аналитик,80000

Алексей Смирнов,40,Менеджер,90000

Елена Кузнецова,30,Разработчик,85000

Дмитрий Попов,25,Тестировщик,60000

Анна Соколова,32,HR,75000

Игорь Лебедев,45,Директор,120000

Мария Козлова,29,Маркетолог,70000

Сергей Новиков,38,Системный администратор,65000

Наталья Морозова,27,Дизайнер,72000

Задание 5: Словарь синонимов

Условие:

Создайте программу, которая читает файл `synonyms.txt`, содержащий слова и их синонимы в формате: слово - синоним1, синоним2, синоним3. Позвольте пользователю вводить слово и получать список его синонимов.

Требования:

1. Чтение из файла:
   - Используйте `with` для чтения `synonyms.txt`.
2. Обработка исключений:
   - Обработайте ошибки отсутствия файла.
3. Использование функций:
   - Создайте функцию `get_synonyms(word)`.
4. Логические операторы и условные конструкции:
   - Проверьте наличие слова в словаре.
5. Использование циклов:
   - Позвольте пользователю вводить слова до тех пор, пока он не захочет выйти.
6. Работа со словарями и списками:
   - Используйте словарь для хранения синонимов.
7. Логирование:
   - записывайте в лог файл запросы пользователя и результаты.
8. Обработка пользовательского ввода:
   - Обработайте ввод в разных регистрах (строчные/прописные буквы).
9. Форматирование строк:
   - Выведите синонимы в виде пронумерованного списка. Например, пользователь ввел слово "большой", вывод должен быть таким:

"большой"

Синонимы 1. огромный, 2. крупный, 3. громадный

10. Импорт модулей:
    - Импортируйте модуль `logging`.

Пример содержимого файла `synonyms.txt`:

большой - огромный, крупный, громадный, масштабный, величавый

маленький - крошечный, небольшой, миниатюрный, малогабаритный, компактный

красивый - прекрасный, симпатичный, привлекательный

умный - интеллектуальный, сообразительный, смышленый

быстрый - скорый, стремительный, резвый

сильный - мощный, крепкий, выносливый, стойкий

добрый - отзывчивый, сердечный, ласковый

интересный - увлекательный, захватывающий, занимательный

Дополнительное задание (по желанию): Обработка текстовых данных

Условие:

Напишите программу, которая считывает текст из файла `input.txt`, удаляет все строки, содержащие определенное слово, и записывает результат в файл `output.txt`.

Требования:

1. Чтение из файла:
   - Используйте `with` для чтения `input.txt`.
2. Обработка исключений:
   - Обработайте ошибки отсутствия файла.
3. Использование функций:
   - Создайте функцию `filter_lines(word, lines)`.
4. Использование циклов:
   - Переберите строки и примените фильтрацию.
5. Логические операторы и условные конструкции:
   - Проверяйте наличие слова в строке.
6. Запись в файл:
   - Запишите отфильтрованные строки в `output.txt`.
7. Логирование:
   - Логируйте количество удаленных строк.
8. Работа со строками:
   - Обработайте строки без учета регистра.
9. Импорт модулей:
   - Импортируйте модуль `logging`.
10. Форматирование строк:
    - Сохраняйте оригинальный формат строк при записи.

Пример содержимого файла `input.txt`:

Я люблю программировать на Python.

Python - очень популярный язык программирования.

Мой друг предпочитает Java.

Мы вместе работаем над проектом.

Изучение новых языков программирования расширяет кругозор.

Python подходит для быстрого прототипирования.

В выходные я планирую отдохнуть.

Дополнительные задачи повышенной сложности которые могут содержать дополнительные темы которые ранее не рассматривались в курсе:

Задание 1: Анализ данных из CSV-файла

Условие:

У вас есть CSV-файл `data.csv`, содержащий информацию о продажах товаров. Каждая строка файла содержит три поля: название товара, количество проданных единиц и цену за единицу.

Требования:

1. Импорт модулей:
   - Импортируйте необходимые модули (`csv`, `logging`).
2. Установка внешних пакетов:
   - Используйте пакет `tabulate` (если его нет, установите через `pip`), чтобы красиво вывести данные в табличном формате.
3. Чтение из файла:
   - Считайте данные из `data.csv` с помощью контекстного менеджера `with`.
4. Обработка исключений:
   - Используйте конструкцию `try-except`, чтобы обработать возможные ошибки при чтении файла.
5. Логирование:
   - Логируйте информацию об успешно считанных данных и об ошибках чтения файла.
6. Обработка данных:
   - Подсчитайте общую выручку по каждому товару и общий доход.
7. Использование функций:
   - Оформите подсчет выручки в виде функции.
8. Работа со структурами данных:
   - Используйте списки и словари для хранения и обработки данных.
9. Вывод данных:
   - Выведите таблицу с названием товара, количеством, ценой и общей выручкой по товару.
10. Логические операторы и условные конструкции:
    - Если общий доход превышает определенное значение, выведите сообщение о высоких продажах.

Задание 2: Парсинг JSON-файла и обработка данных

Условие:

Дан JSON-файл `students.json`, содержащий информацию о студентах и их оценках по разным предметам.

Требования:

1. Импорт модулей:
   - Импортируйте модуль `json`.
2. Чтение из файла:
   - Откройте и считайте данные из `students.json` с помощью контекстного менеджера `with`.
3. Обработка исключений:
   - Обработайте возможные исключения при чтении файла и при парсинге JSON.
4. Создание функций:
   - Напишите функцию, которая рассчитывает средний балл каждого студента.
5. Использование циклов:
   - Используйте циклы `for` для перебора студентов и их оценок.
6. Логические операторы и условные конструкции:
   - Определите студентов, у которых средний балл выше определенного порога.
7. Запись в файл:
   - Запишите результаты в новый JSON-файл `top_students.json`.
8. Использование списков и словарей:
   - Организуйте данные в удобные для обработки структуры.
9. Форматирование строк:
   - Выведите информацию о каждом студенте в читабельном формате.
10. Логирование:
    - Запишите в лог-файл информацию о количестве студентов и об успешном завершении операции.

Задание 3: Калькулятор выражений

Условие:

Создайте программу-калькулятор, которая считывает математическое выражение из файла `expression.txt`, вычисляет результат и записывает его в файл `result.txt`.

Требования:

1. Чтение из файла:

- Считайте выражение из файла `expression.txt`.

2. Обработка исключений:

- Используйте `try-except` для обработки возможных ошибок при чтении файла и вычислении выражения.

3. Создание и использование собственных исключений:

- Создайте собственное исключение `InvalidExpressionError` для случаев некорректного выражения.

4. Использование функций:

- Напишите функцию `evaluate_expression`, которая принимает строку-выражение и возвращает результат.

5. Логические операторы и условные конструкции:

- Проверьте корректность выражения перед вычислением.

6. Запись в файл:

- Запишите результат в файл `result.txt`.

7. Использование finally:

- Обеспечьте закрытие файлов или освобождение ресурсов в блоке `finally`.

8. Логирование:

- Логируйте шаги выполнения программы и возникшие ошибки.

9. Импорт модулей:

- Импортируйте модуль `re` для проверки выражения с помощью регулярных выражений.

10. Работа со строками и числами:

- Парсите и вычисляйте выражение, поддерживая операции сложения, вычитания, умножения и деления.

Задание 4: Обработка текстового файла и анализ слов

Условие:

Напишите программу, которая считывает текст из файла `text.txt`, подсчитывает количество уникальных слов и частоту их встречаемости.

Требования:

1. Чтение из файла:

- Используйте контекстный менеджер `with` для открытия `text.txt`.

2. Обработка исключений:

- Обработайте исключения, связанные с отсутствием файла.

3. Использование функций:

- Создайте функцию `count_words`, которая возвращает словарь с количеством вхождений слов.

4. Использование множеств:

- Определите уникальные слова с помощью множества `set`.

5. Работа со строками:

- Приведите текст к нижнему регистру и удалите знаки препинания.

6. Логические операторы и условные конструкции:

- Исключите из подсчета стоп-слова (например, предлоги, союзы).

7. Запись в файл:

- Запишите результаты в файл `word_count.json` в формате JSON.

8. Импорт модулей:

- Импортируйте модуль `json` для работы с форматом JSON.

9. Использование списков и словарей:

- Организуйте данные для удобной сортировки и вывода.

10. Логирование:

- Логируйте информацию о количестве уникальных слов и общему количеству слов.

Задание 5: Работа с API и обработка данных

Условие:

Используйте внешний API для получения данных о текущей погоде в заданном городе и сохраните эти данные в файл.

Требования:

1. Установка и использование внешнего пакета:

- Используйте пакет `requests` (установите через `pip`, если не установлен).

2. Импорт модулей:

- Импортируйте необходимые модули (`requests`, `json`, `logging`).

3. Обработка исключений:

- Обработайте возможные ошибки сети или некорректного ответа от API.

4. Создание функций:

- Напишите функцию `get_weather(city)`, которая возвращает данные о погоде.

5. Логические операторы и условные конструкции:

- Проверьте корректность введенного города.

6. Запись в файл:

- Сохраните полученные данные в файл `weather.json`.

7. Работа с контекстным менеджером:

- Используйте `with` для работы с файлами.

8. Логирование:

- Логируйте успешные запросы и возникшие ошибки.

9. Использование строк и форматирование:

- Выведите информацию о погоде в читабельном формате.

10. Работа со словарями:

- Обработайте JSON-ответ и извлеките необходимые данные.

Задание 6: Создание простой базы данных в CSV

Условие:

Разработайте программу для ведения списка задач (To-Do List), которые сохраняются в CSV-файл `tasks.csv`.

Требования:

1. Импорт модулей:

- Используйте модуль `csv`.

2. Создание функций:

- Реализуйте функции для добавления, удаления и отображения задач.

3. Обработка исключений:

- Обработайте возможные ошибки при работе с файлом и некорректным вводом пользователя.

4. Использование циклов:

- Организуйте меню для взаимодействия с пользователем в цикле `while`.

5. Запись и чтение из файла:

- При добавлении или удалении задач обновляйте `tasks.csv`.

6. Работа с контекстным менеджером:

- Используйте `with` для открытия файла.

7. Использование списков и словарей:

- Храните задачи в удобной структуре данных для обработки.

8. Логические операторы и условные конструкции:

- Реализуйте проверку наличия задачи перед удалением.

9. Логирование:

- Записывайте в лог-файл действия пользователя и ошибки.

10. Форматирование строк:

- Выводите задачи в формате таблицы с нумерацией.

Задание 7: Калькуляция статистики из Excel-файла

Условие:

У вас есть Excel-файл `data.xlsx` с данными о продажах за месяц. Необходимо вычислить общую сумму продаж и среднюю продажу за день.

Требования:

1. Установка и использование внешнего пакета:
   - Используйте пакет `openpyxl` для работы с Excel-файлом.
2. Импорт модулей:
   - Импортируйте `openpyxl` и необходимые модули.
3. Обработка исключений:
   - Обработайте ошибки при чтении файла и отсутствия данных.
4. Создание функций:
   - Напишите функцию `calculate_sales`, которая возвращает общую и среднюю продажу.
5. Использование циклов:
   - Переберите строки и столбцы в файле для сбора данных.
6. Логические операторы и условные конструкции:
   - Проверьте наличие необходимых столбцов и данных.
7. Работа со списками и числами:
   - Храните продажи в списке для дальнейших вычислений.
8. Запись в файл:
   - Запишите результаты в новый Excel-файл `sales_report.xlsx`.
9. Логирование:
   - Логируйте процесс вычисления и возникшие ошибки.
10. Форматирование строк:
    - Выведите результаты с точностью до двух знаков после запятой.

Задание 8: Словарь синонимов

Условие:

Создайте программу, которая читает файл `synonyms.json`, содержащий слова и их синонимы, и позволяет пользователю вводить слово и получать список его синонимов.

Требования:

1. Чтение из файла:

- Используйте `with` для чтения `synonyms.json`.

2. Обработка исключений:

- Обработайте ошибки отсутствия файла и некорректного JSON.

3. Использование функций:

- Создайте функцию `get_synonyms(word)`.

4. Логические операторы и условные конструкции:

- Проверьте наличие слова в словаре.

5. Использование циклов:

- Позвольте пользователю вводить слова до тех пор, пока он не захочет выйти.

6. Работа со словарями и списками:

- Используйте словарь для хранения синонимов.

7. Импорт модулей:

- Импортируйте `json`.

8. Логирование:

- Логируйте запросы пользователя и результаты.

9. Обработка пользовательского ввода:

- Обработайте ввод в разных регистрах (строчные/прописные буквы).

10. Форматирование строк:

- Выведите синонимы в виде пронумерованного списка.

Задание 9: Игра "Угадай число" с логированием и исключениями

Условие:

Создайте игру, в которой компьютер загадывает число от 1 до 100, а пользователь пытается его угадать.

Требования:

1. Импорт модулей:

- Используйте модуль `random` для генерации числа.

2. Обработка исключений:

- Обработайте ситуации, когда пользователь вводит нечисловые данные.

3. Использование циклов:

- Организуйте процесс игры в цикле `while`.

4. Логические операторы и условные конструкции:

- Подсказывайте пользователю, если загаданное число больше или меньше введенного.

5. Создание функций:

- Напишите функцию `guess_number()`, реализующую логику игры.

6. Логирование:

- Логируйте каждую попытку пользователя и результат.

7. Использование finally:

- В блоке `finally` выведите сообщение о конце игры.

8. Работа со строками и числами:

- Корректно приводите пользовательский ввод к числу.

9. Запись в файл:

- Сохраните результаты игры в файл `game_log.txt`.

10. Форматирование строк:

- Выводите количество попыток после окончания игры.

Задание 10: Обработка данных о сотрудниках

Условие:

У вас есть файл `employees.json`, содержащий информацию о сотрудниках: имя, возраст, должность и зарплата. Необходимо обработать эти данные и вывести список сотрудников, чья зарплата выше среднего.

Требования:

1. Импорт модулей:

- Используйте модуль `json`.

2. Чтение из файла:

- Считайте данные из `employees.json` с помощью `with`.

3. Обработка исключений:

- Обработайте ошибки при чтении файла и некорректном формате данных.

4. Создание функций:

- Напишите функцию `filter_employees(data)`, возвращающую отфильтрованный список.

5. Использование списков и словарей:

- Храните данные в удобных для обработки структурах.

6. Вычисления с числами:

- Рассчитайте среднюю зарплату.

7. Логические операторы и условные конструкции:

- Сравните зарплату каждого сотрудника со средней.

8. Запись в файл:

- Запишите результаты в `high_earners.json`.

9. Логирование:

- Логируйте количество сотрудников и результаты фильтрации.

10. Форматирование строк:

- Выведите информацию о каждом сотруднике в формате: "Имя - Должность - Зарплата".