Погружение в Python (семинары)

Задание 1. Логирование с использованием нескольких файлов

Напишите скрипт, который логирует разные типы сообщений в разные файлы. Логи уровня DEBUG и INFO должны сохраняться в `debug_info.log`, а логи уровня WARNING и выше — в `warnings_errors.log`.

Подсказка № 1

Создайте логгеры с разными уровнями логирования. Используйте `logger.setLevel()` для установки минимального уровня логирования, который будет обрабатываться логгером.

Подсказка № 2

Используйте logging.FileHandler для записи логов в файлы. Установите `FileHandler` для записи сообщений в указанные файлы и укажите уровень логирования для каждого обработчика с помощью метода `setLevel()`.

Подсказка № 3

Настройте формат сообщений с помощью logging.Formatter. Создайте объект `Formatter` для настройки формата сообщений. Примените его к обработчикам с помощью метода `setFormatter()`.

Подсказка № 4

Добавьте обработчики к логгеру с помощью addHandler(). После настройки обработчиков, добавьте их к логгеру с помощью метода `addHandler()`.

Эталонное решение:

```
import logging



# Настройка логирования

logger = logging.getLogger()

logger.setLevel(logging.DEBUG)
```

```
# Форматтер для сообщений
formatter = logging.Formatter('%(asctime)s - %(levelname)s -
%(message)s')


# Обработчик для сообщений уровня DEBUG и INFO
debug_info_handler = logging.FileHandler('debug_info.log')
debug_info_handler.setLevel(logging.DEBUG)
debug_info_handler.setFormatter(formatter)
logger.addHandler(debug_info_handler)


# Обработчик для сообщений уровня WARNING и выше
warnings_errors_handler = logging.FileHandler('warnings_errors.log')
warnings_errors_handler.setLevel(logging.WARNING)
warnings_errors_handler.setFormatter(formatter)
logger.addHandler(warnings_errors_handler)


# Логирование сообщений различных уровней
logger.debug('Это сообщение уровня DEBUG.')
logger.info('Это сообщение уровня INFO.')
logger.warning('Это сообщение уровня WARNING.')
logger.error('Это сообщение уровня ERROR.')
logger.critical('Это сообщение уровня CRITICAL.')
```

Задача 2. Работа с текущим временем и датой

Напишите скрипт, который получает текущее время и дату, а затем выводит их в формате `YYYY-MM-DD HH:MM:SS`. Дополнительно, выведите день недели и номер недели в году.

Подсказка № 1

Используйте `from datetime import datetime`, чтобы получить доступ к текущему времени и дате, а также к методам для их форматирования.

Подсказка № 2

Используйте `datetime.now()` для получения объекта `datetime`, содержащего текущее время и дату.

Подсказка № 3

Примените метод `strftime()` для форматирования даты и времени в строку с нужным форматом, например, `'%Y-%m-%d %H:%M:%S'`.

Подсказка № 4

Используйте `strftime('%A')` для получения полного названия дня недели.
Используйте `isocalendar()[1]` для получения номера недели в году.

Эталонное решение:

```
from datetime import datetime


def display_current_datetime():
    # Получение текущего времени и даты
    now = datetime.now()

    # Форматирование даты и времени
    formatted_date = now.strftime('%Y-%m-%d %H:%M:%S')

    # Получение дня недели и номера недели
    day_of_week = now.strftime('%A')
    week_number = now.isocalendar()[1]

    print(f'Current date and time: {formatted_date}')
```

```
    print(f'Day of the week: {day_of_week}')

    print(f'Week number: {week_number}')


if __name__ == '__main__':

    display_current_datetime()
```

Задача 3. Планирование задач

Напишите функцию, которая принимает количество дней от текущей даты и возвращает дату, которая наступит через указанное количество дней. Дополнительно, выведите эту дату в формате `YYYY-MM-DD`.

Подсказка № 1

Используйте `from datetime import datetime, timedelta`, чтобы получить доступ к текущей дате и времени, а также к функции для добавления или вычитания дней.

Подсказка № 2

Вызовите `datetime.now()` для получения текущей даты и времени в виде объекта `datetime`.

Подсказка № 3

Создайте объект `timedelta`, который представляет собой интервал времени, а затем добавьте его к текущей дате для получения даты в будущем.

Подсказка № 4

Примените метод `strftime()` для преобразования объекта `datetime` в строку в формате YYYY-MM-DD.

Эталонное решение:

```
from datetime import datetime, timedelta


def future_date(days_from_now):

    """
```

```
    Возвращает дату, которая наступит через указанное количество
дней от текущей даты.

    :param days_from_now: Количество дней от текущей даты.
    :return: Отформатированная дата в формате YYYY-MM-DD.

    Примеры:
    >>> future_date(30)
    '2024-09-08'
    >>> future_date(-10)
    '2024-07-30'
    """
    # Получение текущей даты и времени
    today = datetime.now()

    # Вычисление даты через указанное количество дней
    future_date = today + timedelta(days=days_from_now)

    # Форматирование будущей даты в строку в формате YYYY-MM-DD
    formatted_future_date = future_date.strftime('%Y-%m-%d')

    return formatted_future_date

if __name__ == '__main__':
    days = 30  # Количество дней для вычисления
    print(f'Date {days} days from now: {future_date(days)}')
```

Задача 4. Опции и флаги

Напишите скрипт, который принимает два аргумента командной строки: число и строку. Добавьте следующие опции:

- `--verbose`, если этот флаг установлен, скрипт должен выводить дополнительную информацию о процессе.
- `--repeat`, если этот параметр установлен, он должен указывать, сколько раз повторить строку в выводе.

Подсказка № 1

Используйте `import argparse`, чтобы работать с аргументами командной строки.

Подсказка № 2

Используйте `argparse.ArgumentParser` для создания объекта парсера, который будет обрабатывать входные параметры.

Подсказка № 3

Примените метод `add_argument` для добавления обязательных аргументов `number` и `text`. Укажите типы данных и описания.

Подсказка № 4

Добавьте опцию `--verbose` с `action='store_true'` для флага, который активирует дополнительный вывод, и `--repeat` для указания количества повторений строки.

Эталонное решение:

```
import argparse



def main():

    # Создание парсера аргументов

    parser = argparse.ArgumentParser(description='Процессинг числа и
строки с дополнительными опциями.')
```

```
    # Добавление обязательных аргументов

    parser.add_argument('number', type=int, help='Число для вывода')

    parser.add_argument('text', type=str, help='Строка для вывода')



    # Добавление опций

    parser.add_argument('--verbose', action='store_true',
help='Вывод дополнительной информации')

    parser.add_argument('--repeat', type=int, default=1,
help='Количество повторений строки')



    # Парсинг аргументов

    args = parser.parse_args()



    # Вывод дополнительной информации, если опция verbose
установлена

    if args.verbose:

        print(f'Полученные аргументы: number={args.number},
text="{args.text}", repeat={args.repeat}')



    # Вывод строки, повторенной указанное количество раз

    print(f'Число: {args.number}, Строка: {args.text *
args.repeat}')



if __name__ == '__main__':

    main()
```

Задача 5. Запуск из командной строки

Напишите код, который запускается из командной строки и получает на вход путь до директории на ПК. Соберите информацию о содержимом в виде объектов

namedtuple. Каждый объект хранит: имя файла без расширения или название каталога, расширение, если это файл, флаг каталога, название родительского каталога. В процессе сбора сохраните данные в текстовый файл используя логирование.

Подсказка № 1

Используйте функцию `os.path.join()` для правильного построения путей к файлам и каталогам в зависимости от операционной системы.

Подсказка № 2

Используйте `os.path.isdir()` для проверки, является ли указанный путь директорией перед тем как пытаться получить его содержимое.

Подсказка № 3

Используйте `os.path.splitext()` для разделения имени файла на основную часть и расширение, где расширение можно очистить от начальной точки.

Подсказка № 4

Используйте `logging.basicConfig()` для настройки логирования, указав уровень логирования и формат сообщений.

Подсказка № 5

Определите `namedtuple` для хранения информации о файлах и каталогах, что позволяет легко управлять структурой данных и логированием.

Эталонное решение:

```
import os

import logging

from collections import namedtuple

from argparse import ArgumentParser



# Определение namedtuple для хранения информации о файле/каталоге

FileInfo = namedtuple('FileInfo', ['name', 'extension',
'is_directory', 'parent_directory'])
```

```
# Настройка логирования
logging.basicConfig(filename='directory_contents.log', level=logging.INFO, format='%(asctime)s - %(message)s')

def collect_info(directory_path):
    """Собирает информацию о содержимом директории и сохраняет в лог."""
    if not os.path.isdir(directory_path):
        raise ValueError(f"Указанный путь {directory_path} не является директорией.")

    # Получаем родительский каталог
    parent_directory = os.path.basename(os.path.abspath(directory_path))

    # Перебираем содержимое директории
    for entry in os.listdir(directory_path):
        entry_path = os.path.join(directory_path, entry)

        # Проверяем, является ли элемент директорией
        if os.path.isdir(entry_path):
            file_info = FileInfo(name=entry, extension=None, is_directory=True, parent_directory=parent_directory)
        else:
            name, extension = os.path.splitext(entry)
            file_info = FileInfo(name=name, extension=extension.lstrip('.'), is_directory=False, parent_directory=parent_directory)
```

```
        # Запись в лог

        logging.info(f'{file_info.name} | {file_info.extension if
file_info.extension else "N/A"} | {"Directory" if
file_info.is_directory else "File"} | {file_info.parent_directory}')



def main():

    """Основная функция для обработки командной строки и сбора
информации."""

    parser = ArgumentParser(description="Сбор информации о
содержимом директории и запись в лог.")

    parser.add_argument('directory', type=str, help="Путь до
директории для анализа")



    args = parser.parse_args()

    directory_path = args.directory



    try:

        collect_info(directory_path)

        print(f'Информация о содержимом директории
"{directory_path}" успешно записана в файл
"directory_contents.log".')

    except ValueError as e:

        print(e)



if __name__ == '__main__':

    main()
```