Лабораторная работа №5-2.

Основные требования:

- каждая функция должна иметь docstring - множ-й комментарий (по типу что делает данная функция)
- написание функций должно быть компактным (не в 100 строчек кода)
- именования функций должны быть нормальными и четко отражать смысл самой функции
- код должен соответствовать стандарту языка Python (PEP).
- классы и объекты должны соответствовать стандартам языка программирования Python.

Общее задание к Л/Р:

Цель - освоить тему логирование (модуль logging).

Создаем основной файл программы - **main.py**

Шаг 1: Определение класса `MemoryAddress`

1. **Определение Класса:**
   - `MemoryAddress` - класс, представляющий адрес памяти.
   - Инициализируется значением по умолчанию (`default_value`).
2. **Конструктор (`__init__`):**
   - Конструктор инициализирует адрес памяти заданным значением `default_value`.
3. **Геттер (`__get__`):**
   - Определяет метод геттера для получения текущего значения адреса памяти.
4. **Сеттер (`__set__`):**
   - Определяет метод сеттера для установки нового значения адреса памяти.
5. **Метод ssvalue:**
   - `ssvalue` - метод, который принимает стек (`stack`) в качестве параметра и возвращает уже новый стек (прочитав текстовый файл с данными найти указанный адрес и перезаписать получается значение стека).

Для чтения текстового файла с адресами:

```
def read_addresses_from_file(self, filename: str) ->
None:

        with open(filename, 'r') as file:
```

Для поиска адреса:

```
def find_address_and_print_value(self, search_address:
str) -> None:

        for memory_address in self.memory_addresses:

.....
```

Шаг 2: Определение класса `WA`

1. **Определение Класса:**
   - `WA` - класс, представляющий рабочую область.
2. **Конструктор (`__init__`):**
   - Инициализирует рабочую область начальным адресом памяти (`address`).
   - Создает пустой словарь (`stack`) для хранения значений, связанных с адресами памяти.
3. **Метод get_address:**
   - Возвращает текущий адрес памяти.
4. **Метод set_address:**
   - Устанавливает новый адрес памяти и генерирует случайное значение, связанное с этим адресом, сохраняя его в словаре `stack`.
5. **Метод ssvalue:**
   - Вызывает метод `ssvalue` связанного экземпляра `MemoryAddress`, передавая в качестве параметра словарь `stack`.

Шаг 3: Создание экземпляра `WA` и тестирование функциональности

1. **Создание экземпляра `WA`:**
   - Создает экземпляр класса `WA` с начальным адресом памяти (`"\x501\x51\x52\x5344\x54\x55\x5655\x57\x59\x67"`).
2. **Вывод начального адреса памяти:**
   - Выводит начальный адрес памяти с использованием метода `get_address`.
3. **Изменение адреса памяти:**
   - Вызывает метод `set_address` для установки нового адреса памяти (`"\x60\x61\x62\x63\x64\x65"`) и связывает с ним случайное значение.
4. **Вывод нового адреса памяти:**
   - Выводит новый адрес памяти с использованием метода `get_address`.
5. **Вызов метода `ssvalue`:**

- Вызывает метод `ssvalue`, который выводит значение, связанное с текущим адресом памяти из словаря `stack`.

Конечный шаг это добавление логирования с сохранением в специальный файл .log

Базовый класс:

```
import random
import logging


class MemoryAddress:
    def __init__(self, default_value: str) -> None:
        self.value = default_value

class WA:

    def __init__(self, address: str) -> None:

……….
```

```
if __name__ == '__main__':

    obj =
WA("\x501\x51\x52\x5344\x54\x55\x5655\x57\x59\x67")

    print(obj.get_address())

    # Change the address

    obj.set_address("\x60\x61\x62\x63\x64\x65")

    print(obj.get_address())

    # Call ssvalue method
```

```
obj.ssvalue()
```

Правильный ответ: *5820*