Проверочная 2.0

Форма сдачи

Создайте папку на рабочем столе вашего компьютера, которая называется Фамилия_Имя (например, *Гослинг_Райан* или *Булгаков_Михаил*). В этой папке могут находиться:

- одна IPYNB-тетрадка `tasks.ipynb`, которая **явно** разделена на блоки по заданиям (например, так)
- три PY-файла: `task1.py`, `task2.py` и `task3.py`, - в каждом из которых находится код к соответствующему заданию

NB! Не забывайте про кодстайл: `.md` и `.pdf`!

Задачи

Задача №1

Вам надо реализовать программу, которая бы "передразнивала" пользователя. То есть на введенную им фразу выдается аналогичная фраза, слова которой изменены на другие, но сохранены грамматические показатели из "оригинала". Например, "*Мама читала книгу*" может превратиться в "*Дочка ела кашу*", где:

- *мама* и *дочка*: существительное, одушевленное, женский род, единственное число, именительный падеж;
- *читала* и *ела*: глагол, несовершенный вид, переходный, женский род, единственное число, прошедшее время, изъявительное наклонение;
- *книгу* и *кашу*: существительное, неодушевленное, женский род, единственное число, винительный падеж.

У вас есть ZIP-файл с TSV-файлом с самыми частотными словоформами из НКРЯ. Попросите пользователя ввести фразу и выведите ее "передразненный" вариант, ища аналоги слов в файле.

P.S. Давайте считать, что фраза вводится пользователем **без** знаков пунктуации. Дополнительный балл можно получить, если вы не будете изменять стоп-слова в оригинальной фразе.
NB! Чтобы не было проблем при установке pymorphy на Python 3.11+, устанавливайте pymorphy3 (`pip install pymorphy3`) и импортируйте MorphAnalyzer из него (`from pymorphy3 import MorphAnalyzer`). Или пользуйтесь Google Colab для этого задания: там версия 3.10.

Задача №2

Ограничение по памяти	256 мегабайт
Ограничение по времени	2 секунды

Вовочка записал на доске все целые числа от `l` до `r`. Вы можете сделать следующее:

- стереть два числа a и b с доски и вместо них записать $3 * a$ и $floor(b/3)$.

За сколько действий минимум можно обратить все числа в 0?

На вход программе подаётся число `n`. Затем подаётся `n` пар чисел — `l` и `r`. Дла каждой пары надо вывести нужное число действий.

Ввод	**Вывод**
4 1 3 2 4 1999999 2000000 19 84	 5 6 36 263

NB! При решении задачи запрещается использовать сторонние библиотеки.

Задача №3

Реализовать класс `Logger` по следующему шаблону:

```
class Logger:
    def __init__(self, log_level: str = "warning") -> None:
        ...

    def log(self, log_level: str) -> Callable:
        ...
```

Объект класса создаётся с одним из возможных уровней логирования: `debug`, `info`, `warning`, `error`. Второй метод `log` является декоратором. В него в качестве параметра передаётся уровень логирования. При вызове декорированной функции происходит следующее:

- Если уровень логирования функции меньше, чем уровень логирования объекта класса, все вызовы функции `print(*args, **kwargs)` заменяются на `print()`, то есть удаляются все аргументы.

- В остальных случаях происходит преобразование вида: `print(*args, **kwargs)` → `print(f'{log_level.upper()}: ', *args, **kwargs)`, то есть перед результатом вывода печатается уровень логирования.

Аргументами функции `print` могут выступать числа, логические константы, строки, переменные. Гарантируется, что вызов функции умещается на одной строке

Рекомендации по написанию кода:

- Использовать библиотеку `inspect` для получения кода функции.
- Производить получение функции по измененному коду следующим образом:

```
exec_globals = func.__globals__.copy()
exec(modified_code, exec_globals)
modified_func = exec_globals[func.__name__]
```

- Для поиска вызовов фурнкции print рекомендуется использовать следующий паттерн::

```
r"^(?P<indent>\s*)print\((?P<args_and_kwargs>.*?)\)\s*$"
```

- А для поиска аргументов:

```
r"""
    (?P<arg>
        (?:'[^']*'                                  # Single-quoted string
        |"[^"]*"                                    # Double-quoted string
        |\d+                                        # Number
        |True|False|None                            # Boolean and None
        |[a-zA-Z_]\w*                               # Variable name
        )
    )
    |
    (?P<kwarg>
        (?P<key>\w+)\s*=\s*                         # Keyword argument key
        (?P<value>
            ('[^']*'|"[^"]*"|\d+|True|False|None|[a-zA-Z_]\w*)
        )
    )
"""
```

После этого данный код можно вызвать следующим образом

```
modified_func(*args, **kwargs)
```

Пример кода для проверки:

```
logger = Logger("info")


@logger.log("warning")
def func1():
    print("Hello from func1")


@logger.log("info")
def func2():
    print("Hello from func2")


@logger.log("debug")
def func3():
    print("Hello from func3")


func1()
func2()
func3()
```

Данный код должен вывести следующее:

```
WARNING: Hello from func1
INFO: Hello from func2
```