ДЗ и популярные проблемы

Егор Суворов

Курс «Парадигмы и языки программирования», подгруппа 3

Среда, 28 сентября 2016 года

Организационное

- Начинайте тему с [parad].
- Отдельное домашнее задание — отдельная цепочка писем (только e-mail). Добавляйте [task-02] (или 03, или 05...) после [parad].
- Сохраняйте историю переписки внутри цепочек.
- Вопрос — либо отдельная цепочка писем с подстрочкой «вопрос» в теме и тэгом задания (регистр неважен), либо ВК/Telegram.
- Не сдавайте домашки в письме с темой «вопрос».
- Сразу добавляйте ссылку на страницу с файлом/папкой.

Мотивация: на вопросы я могу ответить быстро и на ходу.

Примеры тем:

- [parad] [task-02] Попытка сдачи
- [parad] [task-02] *Вопрос* по заданию
- ~~[parad] [task-02] Попытка сдачи и вопрос~~
- ~~Домашнее задание 2 из #АУ #nofilter~~

Примерный план занятий

Темы (по каждой надо получить баллы):

1. Основы синтаксиса Python (дз 1)
2. Python (дз 2)
3. Numpy и основы ООП (дз 3-...)
4. Продвинутые классы в Python
5. Паттерны проектирования
6. Представление чисел в пямяти
7. Параллельное программирование
8. Функциональное программирование (Haskell)
9. Введение в SQL
10. Дополнительные главы — в процессе обсуждения.

Получение зачёта

Если не закрыта какая-то тема, то:

- Чётких правил нет, «на усмотрение преподавателя».
- Если не закрыта какая-то одна тема, а остальное выглядит хорошо — надо сделать задачу средней сложности по незакрытой теме.
- Если тема не покрывается одной задачей — их будет несколько.
- Сложность задачи — примерно как домашка. Code Review остаётся, вопросы задавать можно.
- Возможно, потребуется делать не дома, а лично присутствуя в классе («на усмотрение»).

Если мало баллов, то, скорее всего, потребуется решить какое-то количество задач по темам, где у вас мало баллов.

Обратная связь

- Готов обсуждать и даже менять по согласованию критерии оценки, правила игры, оргмоменты.
- Любая критика и жалобы на жизнь также приветствуются. Особенно если есть предложения «как лучше».
- Можно писать и передавать коллективные письма.
- О планируемых завалах (неделя коллоквиумов/презентация проектов/отдых) лучше предупреждать заранее.
- Кому (не)комфортно читать технический английский?
- Чего вы ждёте от этого курса? От университета?
- Делитесь тайными знаниями не только с товарищами, но и со мной. Тогда я знаю, что я упустил на паре.

Фатальные проблемы в дз

1. Несоответствие PEP 8.
2. Shebang не совсем верный, правильно так:

```
/usr/bin/env python3
```

3. Некомпилирующийся код: синтаксические ошибки, опечатки в именах переменных или функций.
4. Несоответствие заданию: в основном проблемы с вводом и выводом (откуда/куда и формат).

Это всё можно проверить автоматически!

- Скрипты на bash/cmd.
- Перенаправление потоков ввода-вывода и команда `diff`.
- Хоткей в редакторе для запуска.

Архитектурные проблемы-1

Разделение на функции. В функции выносится:

1. Повторяющийся код.
2. Делающий ровно одну вещь, которая описывается названием.
3. Логический кусок: «программа обходит папку, *ищет дубликаты*, выводит список».

Смешивание кусков в коде (ввод, решение, вывод). Если куски смешаны, то при изменении требований к одному (например, неверно поняли задание или забыли деталь) надо:

1. Аккуратно искать вхождения кода в программу.
2. Следить за зависимостями между кусками.
3. Тестировать сразу все куски вместе, а не по отдельности.

Архитектурные проблемы-2

Дополнительная проблема: возникает соблазн оптимизировать и подгонять решение под формат вывода. Например, сразу готовить данные к выводу: собирать строку `file1:file2` вместо массива `["file1 "file2"]`.
В чём проблема?

Архитектурные проблемы-2

Дополнительная проблема: возникает соблазн оптимизировать и подгонять решение под формат вывода. Например, сразу готовить данные к выводу: собирать строку `file1:file2` вместо массива `["file1 "file2"]`.
В чём проблема?
Мы предполагаем, что в имени файла нет двоеточия.

Архитектурные проблемы-2

Дополнительная проблема: возникает соблазн оптимизировать и подгонять решение под формат вывода. Например, сразу готовить данные к выводу: собирать строку `file1:file2` вместо массива `["file1 "file2"]`.
В чём проблема?
Мы предполагаем, что в имени файла нет двоеточия.
А это неверно под Linux.

Архитектурные проблемы-2

Дополнительная проблема: возникает соблазн оптимизировать и подгонять решение под формат вывода. Например, сразу готовить данные к выводу: собирать строку `file1:file2` вместо массива `["file1 "file2"]`.
В чём проблема?
Мы предполагаем, что в имени файла нет двоеточия.
А это неверно под Linux.
Если куски отделены:

1. Связи тоньше — легче следить и отлаживать.
2. Легче доказывать корректность из-за изолированности.
3. Можно полностью стереть один кусок и переписать, ничего не сломается.
4. Можно один кусок заменить на «фальшивую» реализацию для тестирования.

Архитектурные запахи

- Есть повторы кода или очень похожего кода. Надо изолировать.
- Код, в котором легко ошибиться (плюс-минус единица, есть крайний случай, перепутать аргументы местами). Можно не ошибиться, но перепутать инварианты в разных кусках кода. Локально корректно, глобально — нет.
- Наличие цикла с инвариантом, который нельзя сразу сформулировать. Или из-за которого лезут случаи. Пример: поиск одинаковых подряд идущих файлов (`std::unique` в C++). Лучше функциональный стиль без изменений состояния программы.
- Наличие случаев, которые не изолированы в отдельной функции. Например, проверка `name[0]` и пустые строки вместо простого `startswith`.
- Неожиданные (нетривиальные) формулы.

Внутри функций тоже можно выделять логические куски и изолировать их друг от друга!

Имена

Имена функций:

1. Обычно содержат глагол: `read_matrix`, `multiply`, `print`.
2. Длиннее одной буквы.
3. Не слишком общие и полностью описывают действие функции: ~~`do`~~, ~~`read`~~, ~~`process_files`~~.

Имена переменных:

1. Одна буква — либо целочисленный счётчик `i`, либо буква из задания/математической формулы из статьи.
2. Единственное число.
3. Множественное число для списков.
4. Для словарей: *key* `_to_` *value*.
5. `_` (нижнее подчёркивание), если не используется дальше в коде.

Запахи имён и переменных

Про имена:

- Имена похожи: `dir_big`, `dir_large` или `res1`, `res2`. В чём отличие переменных?
- Нельзя описать, что делает функция или что хранит переменная. Слишком много на себя берут? Переиспользуются в разных смыслах?
- Временная переменная с именем `temp`.

Про переменные:

- Объявлена в начале функции, а используется потом.
- Объявлена не на том же уровне, где используется.
- Входит в инвариант цикла, хотя может этого и не делать.