Яндекс Маркет

Как тестировать питонячьи либы

Евгений Блинов
Руководитель команды интеграции робототехнических решений

Обо мне

Работаю в Yandex Robotics тимлидом в команде интеграции робототехнических проектов

Обожаю Python и регулярно на нем пишу.
В свободное время делаю опенсорс

Telegram: @blinof

GitHub: github.com/pomponchik

LinkedIn: linkedin.com/in/pomponchik

Личный сайт: pomponchik.org

Мотивационная часть

Яндекс Маркет

Пишите библиотеки

- 99.(9)% нужного вам кода уже написано

Пишите библиотеки

- 99.(9)% нужного вам кода уже написано
- Код становится лучше, когда мы решаем не *конкретную задачу*, а целый *класс задач*

Пишите библиотеки

- 99.(9)% нужного вам кода уже написано
- Код становится лучше, когда мы решаем не *конкретную задачу*, а целый *класс задач*
- В нем лучше выделены абстракции и поэтому легче вносить изменения

Пишите библиотеки

- 99.(9)% нужного вам кода уже написано
- Код становится лучше, когда мы решаем не *конкретную задачу*, а целый *класс задач*
- В нем лучше выделены абстракции и поэтому легче вносить изменения
- Это лучший способ повышать производительность труда и сделать богаче всех. Это, а не LLM

Особенности либ и их тестирования

Яндекс
Маркет

Код имеет иерархию

- Любой код — часть пирамиды абстракций
  - ... -> Python-код -> сторонние библиотеки -> стандартная библиотека + интерпретатор -> Си + его компилятор и куча вспомогательных технологий -> языки ассемблера -> машинный код -> железо и его микрокод -> ...

Код имеет иерархию

- Любой код — часть пирамиды абстракций
  - ... -> Python-код -> сторонние библиотеки -> стандартная библиотека + интерпретатор -> Си + его компилятор и куча вспомогательных технологий -> языки ассемблера -> машинный код -> железо и его микрокод -> ...
- На самом деле слоев еще больше и они сложно перепутаны между собой

Код имеет иерархию

- Любой код — часть пирамиды абстракций
  - ... -> Python-код -> сторонние библиотеки -> стандартная библиотека + интерпретатор -> Си + его компилятор и куча вспомогательных технологий -> языки ассемблера -> машинный код -> железо и его микрокод -> ...
- На самом деле слоев еще больше и они сложно перепутаны между собой
- Мы балансируем на ее вершине

Код имеет иерархию

- Любой код — часть пирамиды абстракций
  - ... -> Python-код -> сторонние библиотеки -> стандартная библиотека + интерпретатор -> Си + его компилятор и куча вспомогательных технологий -> языки ассемблера -> машинный код -> железо и его микрокод -> ...
- На самом деле слоев еще больше и они сложно перепутаны между собой
- Мы балансируем на ее вершине
- Чем глубже вниз, тем больше скрытой сложности

Код имеет иерархию

- Любой код — часть пирамиды абстракций
  - ... -> Python-код -> сторонние библиотеки -> стандартная библиотека + интерпретатор -> Си + его компилятор и куча вспомогательных технологий -> языки ассемблера -> машинный код -> железо и его микрокод -> ...
- На самом деле слоев еще больше и они сложно перепутаны между собой
- Мы балансируем на ее вершине
- Чем глубже вниз, тем больше скрытой сложности
- Чем ниже в пирамиде находится код, тем он важнее

Важность опоры

- Все абстракции «текут»

Важность опоры

- Все абстракции «текут»
- Мы не можем это совсем исправить, но можем влиять на вероятности

Важность опоры

- Все абстракции «текут»
- Мы не можем это совсем исправить, но можем влиять на вероятности
- Надежность всей пирамиды — это произведение надежности ее слоев

Важность опоры

- Все абстракции «текут»
- Мы не можем это совсем исправить, но можем влиять на вероятности
- Надежность всей пирамиды — это произведение надежности ее слоев
- Надежные программы возможны только при условии надежности всех слоев абстракций

Итак, мы хотим

- Писать либы, чтобы повысить производительность труда

Итак, мы хотим

- Писать либы, чтобы повысить производительность труда
- Сделать продукты более надежными

Итак, мы хотим

- Писать либы, чтобы повысить производительность труда
- Сделать продукты более надежными

Это конфликт.

Решение: пишем либы, но лучше их тестируем

- Либа — это основа логики

Решение: пишем либы, но лучше их тестируем

- Либа — это основа логики
- Любой дефект в либах реплицируется, а значит обходится дороже

Решение: пишем либы, но лучше их тестируем

- Либа — это основа логики
- Любой дефект в либах реплицируется, а значит обходится дороже
- Каждый тест библиотеки принесет в разы больше выгоды, чем такой же тест в использующей его логике

Зачем опенсорс?

Почему бы не ограничиться внутренними либами?

Зачем опенсорс?

Почему бы не ограничиться внутренними либами?

Потому что:

- Многие инструменты, полезные для создания библиотек, работают лучше или дешевле для опенсорса

Зачем опенсорс?

Почему бы не ограничиться внутренними либами?

Потому что:

- Многие инструменты, полезные для создания библиотек, работают лучше или дешевле для опенсорса
- Готовность выложить в опенсорс — показатель качества. Пользуйтесь качественным :)

Как сделать либы более тестируемыми?

- Бьем либы на подлибы

Как сделать либы более тестируемыми?

- Бьем либы на подлибы
- Все элементы с сайд-эффектами должны инжектиться внутрь:
  - Логгеры

Как сделать либы более тестируемыми?

- Бьем либы на подлибы
- Все элементы с сайд-эффектами должны инжектиться внутрь:
  - Логгеры
  - Функция print

Как сделать либы более тестируемыми?

- Бьем либы на подлибы
- Все элементы с сайд-эффектами должны инжектиться внутрь:
  - Логгеры
  - Функция print
  - Что-то, что ходит по сети

Как сделать либы более тестируемыми?

- Бьем либы на подлибы
- Все элементы с сайд-эффектами должны инжектиться внутрь:
  - Логгеры
  - Функция print
  - Что-то, что ходит по сети
  - etc.

Бьем либы на подлибы?

- При росте либы у нас часто возникает развилка:

Бьем либы на подлибы?

- При росте либы у нас часто возникает развилка:
  - Наращивать размер основной либы

Бьем либы на подлибы?

- При росте либы у нас часто возникает развилка:
  - Наращивать размер основной либы
  - Выделить отдельный компонент в подлибу

Бьем либы на подлибы?

- **При росте либы у нас часто возникает развилка:**
  - Наращивать размер основной либы
  - Выделить отдельный компонент в подлибу

- **Если не выделять, со временем:**
  - Абстракции становятся все менее явными и изолированными

Бьем либы на подлибы?

- **При росте либы у нас часто возникает развилка:**
  - Наращивать размер основной либы
  - Выделить отдельный компонент в подлибу

- **Если не выделять, со временем:**
  - Абстракции становятся все менее явными и изолированными
  - Стоимость внесения изменений растет, поскольку растет контекст

Бьем либы на подлибы?

- **При росте либы у нас часто возникает развилка:**
  - Наращивать размер основной либы
  - Выделить отдельный компонент в подлибу

- **Если не выделять, со временем:**
  - Абстракции становятся все менее явными и изолированными
  - Стоимость внесения изменений растет, поскольку растет контекст
  - Вы становитесь заложниками старых неправильных решений

Бьем либы на подлибы?

- **При росте либы у нас часто возникает развилка:**
  - Наращивать размер основной либы
  - Выделить отдельный компонент в подлибу

- **Если не выделять, со временем:**
  - Абстракции становятся все менее явными и изолированными
  - Стоимость внесения изменений растет, поскольку растет контекст
  - Вы становитесь заложниками старых неправильных решений
  - Ваш CI становится медленным, появляются сложности с мутационным тестированием и т. д.

Инжектим сайд-эффекты?

- Даже самые простые компоненты с сайд-эффектами лучше инжектить

Инжектим сайд-эффекты?

- Даже самые простые компоненты с сайд-эффектами лучше инжектить
- Это упрощает тестирование и делает код менее связанным

Кстати, а вы тестируете логирование?

- Проблемы с логами выстреливают в самый неприятный момент
- Тестируйте логи с помощью emptylog:
  - https://github.com/pomponchik/emptylog

```python
from emptylog import LoggerProtocol, EmptyLogger, MemoryLogger, PrintingLogger

def function(logger: LoggerProtocol = EmptyLogger()):
    logger.info('Some text.')

memory_logger = MemoryLogger()
loggers = memory_logger + PrintingLogger()

function(logger=loggers)
#> 2024-07-19 17:51:53.765054 | INFO      | Some text.

print(memory_logger.data.info)
#> [LoggerCallData(message='Some text.', args=(), kwargs={})]
```

Что, если либа имеет внутренние зависимости?

- Если в либе есть зависимости от чего-то внутреннего, что вы не готовы вынести в опенсорс, лучше:
  - Не делать внутренней всю либу

Что, если либа имеет внутренние зависимости?

- Если в либе есть зависимости от чего-то внутреннего, что вы не готовы вынести в опенсорс, лучше:
  - Не делать внутренней всю либу
  - Вынести в опенсорс универсальную часть

Что, если либа имеет внутренние зависимости?

- Если в либе есть зависимости от чего-то внутреннего, что вы не готовы вынести в опенсорс, лучше:
  - Не делать внутренней всю либу
  - Вынести в опенсорс универсальную часть
  - Проприетарную часть оставить внутри в виде коннектора или плагина для основной либы

Что, если либа имеет внутренние зависимости?

- Если в либе есть зависимости от чего-то внутреннего, что вы не готовы вынести в опенсорс, лучше:
  - Не делать внутренней всю либу
  - Вынести в опенсорс универсальную часть
  - Проприетарную часть оставить внутри в виде коннектора или плагина для основной либы
  - Основная часть должна уметь принимать инжект таких зависимостей

Есть три типа либ:

- Содержащие только логику и зависимости от стандартной библиотеки

Есть три типа либ:

- Содержащие только логику и зависимости от стандартной библиотеки
- Основанные на сайд-эффектах

Есть три типа либ:

- Содержащие только логику и зависимости от стандартной библиотеки
- Основанные на сайд-эффектах
- Являющиеся по сути продолжениями или расширениями других библиотек

Тестировать их нужно по-разному

- Не смешивать!

Тестировать их нужно по-разному

- Не смешивать!
- Факт смешения обычно указывает на то, что библиотеку нужно разделить на несколько

Тестировать их нужно по-разному

- Не смешивать!
- Факт смешения обычно указывает на то, что библиотеку нужно разделить на несколько
- Первый тип тестируем только юнит-тестами

Тестировать их нужно по-разному

- Не смешивать!
- Факт смешения обычно указывает на то, что библиотеку нужно разделить на несколько
- Первый тип тестируем только юнит-тестами
- Второй и третий юнитами + тестами высшего порядка (интеграционные, тесты совместимости и т. д.)

Обязательно считаем покрытие

- Покрытие кода тестами можно считать автоматически

Обязательно считаем покрытие

- Покрытие кода тестами можно считать автоматически
- Норма покрытия для библиотек: 100%

Обязательно считаем покрытие

- Покрытие кода тестами можно считать автоматически
- Норма покрытия для библиотек: 100%
- Покрытие чаще всего считается по строкам и по бранчам, есть куча других способов

Обязательно считаем покрытие

- Покрытие кода тестами можно считать автоматически
- Норма покрытия для библиотек: 100%
- Покрытие чаще всего считается по строкам и по бранчам, есть куча других способов
- Нам надо и то, и то

Обязательно считаем покрытие

- Покрытие кода тестами можно считать автоматически
- Норма покрытия для библиотек: 100%
- Покрытие чаще всего считается по строкам и по бранчам, есть куча других способов
- Нам надо и то, и то
- Покрытие 100% — это не 100% протестированности

Обязательно считаем покрытие

- Покрытие кода тестами можно считать автоматически
- Норма покрытия для библиотек: 100%
- Покрытие чаще всего считается по строкам и по бранчам, есть куча других способов
- Нам надо и то, и то
- Покрытие 100% — это не 100% протестированности
- Даже когда у нас 100%, обычно еще есть, что протестировать — делайте это

Посчитали покрытие, что дальше?

- Загрузим в какой-нибудь сервис:
  - codecov — https://app.codecov.io/

Посчитали покрытие, что дальше?

- Загрузим в какой-нибудь сервис:
  - codecov — https://app.codecov.io/
- Желательно настроить гейт на 100%

За пределами 100% покрытия

- Никогда не ставим покрытие как цель. Цель — протестировать как можно больше. 100% — это просто минимальный порог

За пределами 100% покрытия

- Никогда не ставим покрытие как цель. Цель — протестировать как можно больше. 100% — это просто минимальный порог
- Не скупимся на количество вариантов в parametrize

За пределами 100% покрытия

- Никогда не ставим покрытие как цель. Цель — протестировать как можно больше. 100% — это просто минимальный порог
- Не скупимся на количество вариантов в parametrize
- Стараемся сгенерить как можно больше идей о том, что можно протестировать

За пределами 100% покрытия

- Никогда не ставим покрытие как цель. Цель — протестировать как можно больше. 100% — это просто минимальный порог
- Не скупимся на количество вариантов в parametrize
- Стараемся сгенерить как можно больше идей о том, что можно протестировать
- Когда больше нет идей, запускаем мутационные тесты

Кстати, а что писать первым: тесты или код?

Код.

Мутационное тестирование

- Суть: делаем микроизменения в коде и проверяем, что тесты падают.
  Если не падают, значит тесты плохие

Мутационное тестирование

- Суть: делаем микроизменения в коде и проверяем, что тесты падают.
  Если не падают, значит тесты плохие

- Я использую mutmut

Мутационное тестирование

- Суть: делаем микроизменения в коде и проверяем, что тесты падают.
  Если не падают, значит тесты плохие
- Я использую mutmut
- Пока не стоит совать это в CI

Мутационное тестирование

- Суть: делаем микроизменения в коде и проверяем, что тесты падают.
  Если не падают, значит тесты плохие
- Я использую mutmut
- Пока не стоит совать это в CI
- Инструменты в этой области крайне неразвиты, вы можете сделать крупный вклад

Мутационное тестирование

- Суть: делаем микроизменения в коде и проверяем, что тесты падают.
  Если не падают, значит тесты плохие
- Я использую mutmut
- Пока не стоит совать это в CI
- Инструменты в этой области крайне неразвиты, вы можете сделать крупный вклад
- Если мутационные тесты прогоняются долго, это указывает на слишком большой размер проекта.
  Часто это хороший повод разделить либу на несколько либ поменьше

Значение количества тестов как цифры

- На мой взгляд, эта цифра важна

Значение количества тестов как цифры

- На мой взгляд, эта цифра важна
- Необычно маленькое число тестов на фоне высокого покрытия может говорить о плохом качестве тестов (например, что запускается почти весь код, но почти без ассертов; или на один тест приходится слишком много ассертов)

Значение количества тестов как цифры

- На мой взгляд, эта цифра важна
- Необычно маленькое число тестов на фоне высокого покрытия может говорить о плохом качестве тестов (например, что запускается почти весь код, но почти без ассертов; или на один тест приходится слишком много ассертов)
- Количество ассертов, наверное, важнее

Значение количества тестов как цифры

- На мой взгляд, эта цифра важна
- Необычно маленькое число тестов на фоне высокого покрытия может говорить о плохом качестве тестов (например, что запускается почти весь код, но почти без ассертов; или на один тест приходится слишком много ассертов)
- Количество ассертов, наверное, важнее
- pytest по умолчанию скрывает точное количество уникальных тестов, а показывает количество параметризованных тестов. Что с этим делать?

Бейджики

- Засовываем в ридмиху бейджики:
  - С покрытием — 

  - С результатом последнего прогона тестов — 

Бейджики

- Засовываем в ридмиху бейджики:
  - С покрытием — 

  - С результатом последнего прогона тестов — 

- Гордимся и показываем друзьям

Бейджики

- Засовываем в ридмиху бейджики:
  - С покрытием — 

  - С результатом последнего прогона тестов — 

- Гордимся и показываем друзьям
- На гитлабе свои тулы

Тестируем документацию

- Любое утверждение о коде, которое есть в документации, должно быть подкреплено каким-либо тестом

Тестируем документацию

- Любое утверждение о коде, которое есть в документации, должно быть подкреплено каким-либо тестом
- Желательно создавать по отдельному тесту для каждого примера кода

Тестируем документацию

- Любое утверждение о коде, которое есть в документации, должно быть подкреплено каким-либо тестом
- Желательно создавать по отдельному тесту для каждого примера кода
- Есть два пути:
  - Завести отдельную папочку с тестами на документацию и вести ее вручную
  - Автоматизировать это

Файловая структура для тестов

- Все тесты лежат в папке tests в корне

Файловая структура для тестов

- Все тесты лежат в папке tests в корне
- В норме повторяют файловую структуру исходников

Файловая структура для тестов

- Все тесты лежат в папке tests в корне
- В норме повторяют файловую структуру исходников
- Если есть более одного типа тестов, создаем внутри папки tests подпапки для разных типов:
  - e2e — для end-to-end тестов
  - documentation — для доки
  - smokes — для смоук-тестов
  - units — для юнит-тестов

Абстракции в тестах

- Избегаем создания тестовых абстракций

Абстракции в тестах

- Избегаем создания тестовых абстракций
- Тесты должны быть концептуально максимально простыми

Абстракции в тестах

- Избегаем создания тестовых абстракций
- Тесты должны быть концептуально максимально простыми
- Если вы вынуждены создать абстракцию, вынесите ее в отдельную либу

Используем matrix

- Есть во многих инструментах CI, конкретно я использую GitHub Actions

Используем matrix

- Есть во многих инструментах CI, конкретно я использую GitHub Actions
- Как стандарт для matrix’а — все поддерживаемые (можно +1-2 устаревших) версии питона и три базовых ОС: Linux, MacOS, Windows

Используем matrix

- Есть во многих инструментах CI, конкретно я использую GitHub Actions
- Как стандарт для matrix’а — все поддерживаемые (можно +1-2 устаревших) версии питона и три базовых ОС: Linux, MacOS, Windows
- Не делаем утверждений о поддержке чего-либо, если запуск этой штуки не поддержан в CI

Используем matrix

- Есть во многих инструментах CI, конкретно я использую GitHub Actions
- Как стандарт для matrix’а — все поддерживаемые (можно +1-2 устаревших) версии питона и три базовых ОС: Linux, MacOS, Windows
- Не делаем утверждений о поддержке чего-либо, если запуск этой штуки не поддержан в CI
- Ставим классифаеры про версии питона и ОС только в случае поддержки в CI

Используем matrix

- Есть во многих инструментах CI, конкретно я использую GitHub Actions
- Как стандарт для matrix’а — все поддерживаемые (можно +1-2 устаревших) версии питона и три базовых ОС: Linux, MacOS, Windows
- Не делаем утверждений о поддержке чего-либо, если запуск этой штуки не поддержан в CI
- Ставим классифаеры про версии питона и ОС только в случае поддержки в CI
- Также можно посмотреть в сторону Tox, Pyenv и пр.

Не забываем устанавливать библиотеку в CI

- Питонячьи либы могут работать даже без установки, если мы в директории проекта

Не забываем устанавливать библиотеку в CI

- Питонячьи либы могут работать даже без установки, если мы в директории проекта
- Не соблазняемся этим: некоторые проблемы всплывают только при попытке установки (например, конфликты зависимостей)

Проверяем совместимость плагинов

- Если вы пишете плагины и дополнения к основной библиотеке, не забывайте тестировать совместимость

Проверяем совместимость плагинов

- Если вы пишете плагины и дополнения к основной библиотеке, не забывайте тестировать совместимость
- Совместимость нужно тестировать с диапазоном версий

Проверяем совместимость плагинов

- Если вы пишете плагины и дополнения к основной библиотеке, не забывайте тестировать совместимость
- Совместимость нужно тестировать с диапазоном версий
- Не гарантируем совместимость, если не проверили

Проверяем совместимость плагинов

- Если вы пишете плагины и дополнения к основной библиотеке, не забывайте тестировать совместимость
- Совместимость нужно тестировать с диапазоном версий
- Не гарантируем совместимость, если не проверили
- Пайплайн проверки совместимости нужно регулярно перезапускать при выходе новых версий основной либы

Для проверки совместимости можно использовать instld

- instld — это инструмент скачивания либ в рантайме
  - https://github.com/pomponchik/instld
- Там есть контекстный менеджер для скачивания либ
- Используем его в цикле

Пишем докстринги для тестов

- Описываем на человеческом:
  - В чем смысл теста

Пишем докстринги для тестов

- Описываем на человеческом:
  - В чем смысл теста
  - Все неочевидное, что в нем есть, со ссылками на источники

Тест как стандарт для issue

- В идеале просить пользователей репортить баги в формате готовых тестов, которые можно запустить

Тест как стандарт для issue

- В идеале просить пользователей репортить баги в формате готовых тестов, которые можно запустить
- Это можно включить в шаблон issue

Тест как стандарт для issue

- В идеале просить пользователей репортить баги в формате готовых тестов, которые можно запустить
- Это можно включить в шаблон issue
- Если баг-репорт не сопровожден тестом, сначала пишем тест, воспроизводящий проблему, затем фиксим ее. К коммиту с тестом прикрепляем id issue

Как быть с многопоточкой?

- Отделяем логическую часть от менеджмента многопоточки:
  - Желательно на разные модули
  - Тестируем логику как логику

Как быть с многопоточкой?

- Отделяем логическую часть от менеджмента многопоточки:
  - Желательно на разные модули
  - Тестируем логику как логику
- Критические части желательно тестировать на гонки

Резюме

- Пишем либы, выкладываем в опенсорс и очень хорошо тестируем

Резюме

- Пишем либы, выкладываем в опенсорс и очень хорошо тестируем
- Бьем либы на подлибы

Резюме

- Пишем либы, выкладываем в опенсорс и очень хорошо тестируем
- Бьем либы на подлибы
- Инжектим сайд-эффекты (в т. ч. логирование)

Резюме

- Пишем либы, выкладываем в опенсорс и очень хорошо тестируем
- Бьем либы на подлибы
- Инжектим сайд-эффекты (в т. ч. логирование)
- Если есть «внутренние» зависимости, оформляем в виде плагинов

Резюме

- Пишем либы, выкладываем в опенсорс и очень хорошо тестируем
- Бьем либы на подлибы
- Инжектим сайд-эффекты (в т. ч. логирование)
- Если есть «внутренние» зависимости, оформляем в виде плагинов
- Минимальное покрытие — 100%

Резюме

- Пишем либы, выкладываем в опенсорс и очень хорошо тестируем
- Бьем либы на подлибы
- Инжектим сайд-эффекты (в т. ч. логирование)
- Если есть «внутренние» зависимости, оформляем в виде плагинов
- Минимальное покрытие — 100%
- Вешаем бейджики

Резюме

- Пишем либы, выкладываем в опенсорс и очень хорошо тестируем
- Бьем либы на подлибы
- Инжектим сайд-эффекты (в т. ч. логирование)
- Если есть «внутренние» зависимости, оформляем в виде плагинов
- Минимальное покрытие — 100%
- Вешаем бейджики
- Тестируем доку

Резюме

- Пишем либы, выкладываем в опенсорс и очень хорошо тестируем
- Бьем либы на подлибы
- Инжектим сайд-эффекты (в т. ч. логирование)
- Если есть «внутренние» зависимости, оформляем в виде плагинов
- Минимальное покрытие — 100%
- Вешаем бейджики
- Тестируем доку
- Используем matrix

Резюме

- Пишем либы, выкладываем в опенсорс и очень хорошо тестируем
- Бьем либы на подлибы
- Инжектим сайд-эффекты (в т. ч. логирование)
- Если есть «внутренние» зависимости, оформляем в виде плагинов
- Минимальное покрытие — 100%
- Вешаем бейджики
- Тестируем доку
- Используем matrix
- Проверяем плагины на совместимость

Резюме

- Пишем либы, выкладываем в опенсорс и очень хорошо тестируем
- Бьем либы на подлибы
- Инжектим сайд-эффекты (в т. ч. логирование)
- Если есть «внутренние» зависимости, оформляем в виде плагинов
- Минимальное покрытие — 100%
- Вешаем бейджики
- Тестируем доку
- Используем matrix
- Проверяем плагины на совместимость
- Просим тесты в багрепортах и пишем докстринги

Что я не покрыл в докладе

- Тесты производительности и бенчмарки

Что я не покрыл в докладе

- Тесты производительности и бенчмарки
- Нагрузочные тесты

Что я не покрыл в докладе

- Тесты производительности и бенчмарки
- Нагрузочные тесты
- Особенности тестирования мультиязычных либ (например с вставками из Си)

Что я не покрыл в докладе

- Тесты производительности и бенчмарки
- Нагрузочные тесты
- Особенности тестирования мультиязычных либ (например с вставками из Си)
- Что-то за пределами pytest

На этом все

:)

Голосуйте за доклад и задавайте вопросы