Инструменты для программирования

Дмитрий Халанский

28 сентября 2020 г.

Парадокс Питона

`http://www.paulgraham.com/pypar.html` (2004)

- "Python programmers are smart. It's a lot of work to learn a new programming language. And people don't learn Python because it will get them a job; they learn it because they genuinely like to program and aren't satisfied with the languages they already know."
- "Thee Python paradox: if a company chooses to write its software in a comparatively esoteric language, they'll be able to hire better programmers, because they'll attract only those who cared enough to learn it."

2004: Python vs Java

- Python не требует типовых аннотаций, меньше мусора в коде.

2004: Python vs Java

- Python не требует типовых аннотаций, меньше мусора в коде. В современной Java можно писать меньше мусора, да и почти всё автодополнит IDE, к тому же лучше, чем в Python.

2004: Python vs Java

- Python не требует типовых аннотаций, меньше мусора в коде. В современной Java можно писать меньше мусора, да и почти всё автодополнит IDE, к тому же лучше, чем в Python.
- Python более читаемый: он заставляет программистов писать код с вменяемым форматированием.

2004: Python vs Java

- Python не требует типовых аннотаций, меньше мусора в коде. В современной Java можно писать меньше мусора, да и почти всё автодополнит IDE, к тому же лучше, чем в Python.
- Python более читаемый: он заставляет программистов писать код с вменяемым форматированием. В Java никто не пишет без IDE, так что почти весь код уже ровный; если нет, можно читать его тоже в IDE и там автоматически выровнять.

2004: Python vs Java

- Python не требует типовых аннотаций, меньше мусора в коде. В современной Java можно писать меньше мусора, да и почти всё автодополнит IDE, к тому же лучше, чем в Python.
- Python более читаемый: он заставляет программистов писать код с вменяемым форматированием. В Java никто не пишет без IDE, так что почти весь код уже ровный; если нет, можно читать его тоже в IDE и там автоматически выровнять.
- В Python быстрее прототипировать: у него есть REPL.

2004: Python vs Java

- Python не требует типовых аннотаций, меньше мусора в коде. В современной Java можно писать меньше мусора, да и почти всё автодополнит IDE, к тому же лучше, чем в Python.
- Python более читаемый: он заставляет программистов писать код с вменяемым форматированием. В Java никто не пишет без IDE, так что почти весь код уже ровный; если нет, можно читать его тоже в IDE и там автоматически выровнять.
- В Python быстрее прототипировать: у него есть REPL.У Java он недавно тоже появился.

2004: Python vs Java

- Python не требует типовых аннотаций, меньше мусора в коде. В современной Java можно писать меньше мусора, да и почти всё автодополнит IDE, к тому же лучше, чем в Python.
- Python более читаемый: он заставляет программистов писать код с вменяемым форматированием. В Java никто не пишет без IDE, так что почти весь код уже ровный; если нет, можно читать его тоже в IDE и там автоматически выровнять.
- В Python быстрее прототипировать: у него есть REPL.У Java он недавно тоже появился.

Урок: инструменты настолько мощные, что позволяют преодолеть даже огромный разрыв в эргономике.

Успех Python

- Маркетинг для непрофессиональных программистов;
- Простой для чтения синтаксис;
- *Быстрое прототипирование*.

1. Успех Питона

2. Статические инструменты
   - REPL
   - Линтеры
   - Форматтеры
   - Всё в одном: IDE
   - Всё в одном: LSP
   - Работа с Haskell

1. Успех Питона

2. Статические инструменты
   - REPL
   - Линтеры
   - Форматтеры
   - Всё в одном: IDE
   - Всё в одном: LSP
   - Работа с Haskell

REPL

REPL — интерактивное окружение, исполняющее вводимые в него высказывания и тут же демонстрирующее результат.
Языки, известные за свой REPL:

- `sh` — по большей части используется как REPL;
- Python — имеет собственный REPL по команде `python`, а также улучшенную версию, `ipython`.
- LISP — в этом языке REPL был придуман;
- Haskell — `ghci`;
- JavaScript — F12 в браузере.

Концепция REPL обрела популярность и есть почти везде: Java, Scala, Kotlin, Rust (`evcxr`), C/C++ (`gdb`, хотя это не совсем честно)...

Зачем нужен REPL

- В динамических языках — с помощью TAB получать актуальный список методов и полей;
- Быстро проверять свои догадки по поводу того, как работает то или иное;
- Строить сложную последовательность действий, корректность которой сложно проверять в голове, а потом скопировать результат в файл.

1. Успех Питона

2. Статические инструменты
   - REPL
   - Линтеры
   - Форматтеры
   - Всё в одном: IDE
   - Всё в одном: LSP
   - Работа с Haskell

Линтеры

Линтер — программа, которая находит типичные ошибки и предлагает способы их починить. Помогает программисту улучшить стиль, избежать ошибок и получше изучить язык.

- Haskell — `hlint` (с флагами ––refactor ––refactor–options=–is сам всё чинит);
- sh — `shellcheck` (написан на Haskell);
- C++ — `clang-tidy`;
- Python — `pylint`;
- Rust — `clippy`...

Пример вывода hlint

```
Warning: Redundant ==
Found:
  a == True
Perhaps:
  a

Suggestion: Redundant bracket
Found:
  (plus 3) 4
Perhaps:
  plus 3 4
```

1. Успех Питона

2. Статические инструменты
   - REPL
   - Линтеры
   - Форматтеры
   - Всё в одном: IDE
   - Всё в одном: LSP
   - Работа с Haskell

Форматтеры

Очень частая ошибка у даже опытных программистов — форматировать код руками. Это пустая трата времени, которая к тому же более вредная, чем полезная: для читаемости важнее консистентность, чем красота отдельной строчки, а придерживаться одного стиля вручную редко выходит.

- Go — начавший популярность среди форматтеров `gofmt`;
- Python — `yapf`;
- C++ — `clang-format`;
- Haskell — `brittany`, `ormolu`.

1. Успех Питона

2. Статические инструменты
   - REPL
   - Линтеры
   - Форматтеры
   - Всё в одном: IDE
   - Всё в одном: LSP
   - Работа с Haskell

IDE

Можно не устанавливать груду инструментов, а воспользоваться IDE.
Плюсы очевидны.
Минусы:

- Потребляет много ресурсов;
- Иногда ломается, нужно уметь работать и без IDE;
- Сложно интегрировать отдельные инструменты (линтеры, форматтеры) с системой контроля версий;
- Сильно вторгается в весь процесс разработки, добавляя ещё один уровень того, где что-то может пойти не так.

Haskell: есть плагин к Atom `https://atom-haskell.github.io/`.

1. Успех Питона

2. Статические инструменты
   - REPL
   - Линтеры
   - Форматтеры
   - Всё в одном: IDE
   - Всё в одном: LSP
   - Работа с Haskell

LSP

LSP — Language Server Protocol — это протокол взаимодействия между движками IDE и текстовыми редакторами.
Есть language server для большинства языков.
Haskell:
`https://github.com/haskell/haskell-language-server/releases`
Работает *только* для той же версии ghc, которая реально используется.
Большинство редакторов кода поддерживает LSP.

Мораль

Обязательно нужно:

- Установить `hlint`;
- Поставить либо плагин к atom, либо language server;
- Настроить форматтер, чтобы минимально выравнивать код руками, если в редакторе нет кнопки для такой функциональности;
- После того, как вас научат pre-commit hooks, настроить hook на hlint.

Сэкономит тучу времени и сил в изучении языка, хотя и потребует какого-то начального вложения времени.

1. Успех Питона

2. Статические инструменты
   - REPL
   - Линтеры
   - Форматтеры
   - Всё в одном: IDE
   - Всё в одном: LSP
   - Работа с Haskell

Настройка окружения через stack

stack — программа для управления версиями ghc и библиотек: `https://docs.haskellstack.org/en/stable/README/`. Если пользоваться ей, то вместо `ghc` надо вызывать `stack ghc`, а вместо `ghci` — `stack ghci`.
Далее, чтобы пользоваться LSP, надо с `https://github.com/haskell/haskell-language-server/releases` скачать сборку под версию 8.8.3, распаковать и добавить исполняемый файл в `$PATH`. Убедитесь, что из консоли получается вызвать `haskell-language-server-Linux-8.8.3`.
Выкачайте проект с домашними заданиями и вызовите `stack test`. Чтобы выполнить эту команду, Stack попутно выкачает всё нужное.
Теперь можно настроить редактор для поддержки LSP и проверять результат.

Настройка окружения без stack

Проверьте пакетные менеджеры, ищите `ghc` и `cabal-install`.
Помните, что LSP есть только для версии 8.6.4 и выше.
Если там нет достаточно современного компилятора, попробуйте скачать всё с сайта ghc. Затем на `https://www.stackage.org/` посмотрите версию LTS, соответствующую Вашей версии компилятора, в “Latest LTS per GHC version”, либо, если компилятор слишком новый, версию nightly в “Snapshots”. Например, `lts-16.16` для `ghc-8.8.4` или `nightly-2020-09-08` для `ghc-8.10.2`.

Настройка окружения без stack (2)

Настройте редактор. Если собираетесь это делать через LSP, то скачайте с `https://github.com/haskell/haskell-language-server/releases` версию сервера под нужный компилятор.
В проекте, который будет выдан для домашних работ, замените `resolver` в `stack.yaml` на нужную версию. Вызовите `cabal test`.
Настройте редактор для поддержки LSP и проверьте.