Если вы видите что-то необычное, просто сообщите мне.

# Часть 3: Haskell и взвешенная практика

Вы были в ситуации когда вы пытаетесь изучить что-то в определенное время, и застряли на этом? Шанс, что вы изучаете предмет не лучшим способом. Но как вы можете узнать, что входит в это "хорошее" изучние вашей темы? Оно может разочаровать при попытке найти что-то в интернете. Большинство людей не думают о том способе которым они изучают новое. Они изучают, но они не могут выразить и обучить других людей тому, что они делают, потому что для этого нет инструкции.

Часть 1 этой главы говорит нам о том, почему вы не должны бояться пробовать изучить Haskell. Часть 2 обсуждает некоторые техники на высшем уровне. Эта часть пройдется по паре ключевых идей из "Art of Learning" Джоша Вайтцкина. Мы рассмотрим на то, как избежать проблему застрявания.

Первая идея, о которой мы загооврим это идея взвешенной практики. Цель этой практики прицелиться в конкретную идею и попробовать улушчить определенные навыки до тех пор, пока они будут только в подсознании. Вторая идея - роль ошибок в изчении любого нового навыка. Мы будем использовать это для стимула ведущего дальше, которы предостережот нас от этой же идеи в будущем.

Мы раскроем это в 4 части, в которой разберем применение этой техники в изучении Haskell.

Некоторые техники проще понять, если вы уже имеете какую-то практику в изучении Haskell.

# Взвешенная практика

Предположим, на минуточку, вы учите композицию на фортепьяно. Наибольшее искушение здесь это "учите" часть повторяя композицию от начала до конца. Часть у вас получлается, часть - нет. В итоге, вы изучили большую часть. Это заманчивый способ практки по нескольким причинам:

1. Это "очевидный" выбор.
2. Он позволяет нам проиграть часть, которую мы уже узнали и получить удовольствие от этого.
3. Скорей всего в результате мы выучим композицию.

Однако, это не оптимальный метод со стороны обучения. Если вы хотите улучшить вашу возможность играть вашу композицю от начала до конца, вы должны сфокусироваться на слабых местах. Вам нужно найти определенные пассажи, которые вам даются хуже всего. Как только вы их определите, вы можете разбирать их дальше. Вы можете найти определенную размерность или даже ноты с которыми у вас проблемы. Вы должны практиковать слабые места снова и снова, исправляя одну маленькую вещь за другой. Отсюда, вам нужно пройти весь путь и это заставляет вас ожидать, что вы будете уверенны что вы изучили все свои "слабыу" места.

Фокусируясь на маленьких вещях - главная часть. Вы не можете взять хитрый пассаж о котором ничего не знаете и сыграть его правильно от начала до конца. Вы можете начать с одной части, которая заставляет вас ускорится. Вы можете тренироваться много раз просто фокусируясь на получении последней ноты и двигаться руку дальше. Вас не волнует ничего кроме реппетиции. Как только движение вашей руки переходит в подкорку, вы можете идти дальше к следующей части. Следующий шаг уже нужен, чтобы убедиться что вы достигли первых трех нот.

Это идея взвешенной практики. Мы фокусируемся на одной вещи во времени, и практикуем её неспеша. Бездумная практика, или практика рази практики будет давать вам маленький прогресс. Даже может уменьшить прогресс если мы заимеем плохие привычки. Мы можем применять это к любому навыкы, включая программирование. Мы хотим нахватать маленькие привычки, которые будут постепенно делать нас лучше.

# Ошибки

Взвешенная практика это система построения навыков, которую мы хотим. Однако, есть множество привычек, которые мы не хотим. Мы делаем много вещей, ошибки которых мы поймем позже. И наихудшая вещь это когда мы понимаем, что мы повторяем одну и ту же ошибку.

Вайцкин отмети в "Art of Learning": Если студент любого предмета может избежать хотябы повтора одной и той же ошибки дважды, они быстро достигнут высот в изучаемой теме. Мы так же можем сделать это если будем избегать ошибок в целом, но это не возможно. Мы всегда делаем ошибки, пробуя что-то впервые.

Сначала нужно принять, что ошибки будут случаться. Как только мы это сделаелм, у нас будет решение как этого избежать. Невозможно избежать повторения ошибок, но мы можем предпринять шаги, которые сократят их количество. И если мы такое можем сделать, то увидим существенное улучшение. Наше решение будет хранить все ошибки которые мы сделаем. Описывая всё, что происходит, мы резко сократим повторение ошибок.

# Практикуем HASKELL

Теперь нам нужно сделать шаг назад в мир написания кода и спросить себя, как мы можем применить эти идеи к Haskell. На каких темах практики написания кода мы можем остановиться?

Вы можете написать приложение, и сфокусироваться на приобретении следующей привычки: прежде чем вы напишете функцию, нужно вынести неопределенности и убедиться, что типы сигнатур комплируются(мы это обсудим дальше в части 4). Не важно если вы всё остальное сделали правильно! Приобретя эту привычку можноприступать к следующему шагу. Вы можете убедиться в том, что вы всегда пишете вызов функции прежде чем вы её реализуете.

Я выбрал эти примеры, потому что есть две вещи которые тормозят нас больше всего при написании функции. Первое - это нехватка ясности, потому что мы не знаем точно как наш код будет использовать функцию. Второе - повторение работы когда мы поняли. что нам нужно переписать функцию. Это случается, когда мы не учитываем дополнительные возможности. Эти привычки, продуманны таким образом, чтобы ваша жизни становилась

легче, когда нужно реализовать их.

Есть еще несколько похожих идей:

1. Прежде чем писать функцию, напишите коммент описывающий эту функцию.
2. Прежде чем использовать выражение из библиотеки, добавьте её в ваш `.cabal` файл. Затем, напишите выражение `import` чтобы убедиться, что вы используете правильную зависимость.

Еще один способ - знать как проверить кусок функциональности прежде чем реализовать её. В идеале, написать юнит тесты для функции. Но если это что-то простое, как получение строки ввода и её обработка каким-то способом, вы можете опустить простые идеи. Вы можете вложить в рабочую программу, для командной строки, два вида входных данных. Если вы знаете свой подход до того, как начнете программировать, это имеет значение. Имеет смысл написать план тестирования в каком-то документе для начала.

В большинстве случаев триггером для приобретения этой привычки это написание новой функции. Триггер это важная часть приобретения новой ошибки. Это действие которое подсказывает вашему мозгу, что вы должны делать что-то не привычное. В этом случае, триггером будет написание `::` для сигнатуры. Каждый раз, выполняя это, вспоминайте о вашей цели.

Есть еще идея с множеством триггеров. Каждый раз вы выбираете структуру для хранения ваших данных(списко, последовательность, набор, карты и т.д.), как минимум три разных типа. Как только выходите за предели базовых вещей, вы найдете уникальную силу в каждой из структур. Будет ползено если вы выпишите условия сделанного выбора. Триггер, в этом случае, будет проявляться каждый раз, когда вы пишете ключевые данные. Для более сложной версии, триггером может быть быть написание левой скобки для начала списка. Каждый раз, делая это, спросите себя: можете ли вы использовать другую структуру?

И последняя возможность. Каждый раз делая синонимы типов, спросите: стоит ли создавать новый тип? Это часто ведет к улучшеню времени компиляции. Вы скорей всего видели сообщения об ошибках и большую часть получите еще при сборке. Триггер тут тоже простой: каждый раз когда пишете ключевое слово типа.

Это важдые вещи. Не пытайтесь выучить сразу все! Вам нужно выбрать одну, практиковать её пока она будет работать подсознательно, и затем двигайтесь к другой. Это самая сложная часть взвешенной практики: управлять своим терпением. Самое большое влечение это двигаться и пробовать новые вещи, прежде чем усвоится привычка. Как только вы переключитесь на другие вещи, вы можете потерять, то над чем работали. Помните, обучение сложный процесс! Вам нужно сделать небольшое исследование которое требует определенное время. Вы не сможете ускорить этот процесс.

# Отслеживание ошибок

Давайте представим различные способы, которые помогут нам избежать повторения ошибок в будущем. Опять, эти различные навыки вы приобретаете с помощью взвешенной практики. Они не появляются часто, и вам не нужно их "практиковать". Вам нужно просто помнить, как исправить некоторые ошибки, чтобы в тот момент когда они появятся вы сможете это применить еще раз. Вы можете вести список ужасных ошибок в электронном виде, которые встречаете во время программирования.

1. Как себя повел сборщик?
2. В чем заключалась проблема с кодом?
3. Как это можно исправить?

Для примера, подумайте о времени когда вы были уверенны о том, что код верен. Взгляните на ошибку, затем на ваш код, опять на ошибку. И вы всё еще уверенны, что код правильный. Конечно, компилятор почти всегда прав. Вы хотите это записать, чтобы вас нельзя было обвести вокруг пальца еще раз.

Другой хороший кандидат, это ошибки исполнения где вы не можете никаким образом понять где ошибка возникает в вашем коде. Вы хотите записать этот опыт таким образом, чтобы в следующий раз вы могли быстро решить проблему. Выписывание заставляет вас избегать повторений ошибок.

Есть еще глупые ошибки, которые вы должны записывать потому, что они учат вас быстрее. Например, вначале вы можете использовать `(+)` оператор для складывания двух строк, вместо `(++)` оператора. Выписывание таких ошибок позволит выучить свойство гораздо

быстрее.

Последняя группа вещей, которые нужно выписывать отличноее решение. Не просто исправление багов, но решение ваших главных проблем программирования. Для пример, вы находите вашу программу слишком медленной, но вы использовали наилучшие сткруктуры данных для улучшения. У вас так же имеются записи того, что вы делали. Таким образом, у вас будет возможность применить решение еще и в следующий раз. Эти вещи дают пищу для размышления в собеседованиях. Собеседования часто можно услышать вопросы о проблемах которые вы решали, чтобы понять ваши мотивационные способности.

У меня есть один пример, который показывает хорошее и плохое распознавание ошибок. У меня есть жуткий баг, когда я пытаюсь собрать Haskell проект используя Cabal. Я помню была ошибка компоновщика, которая не указаывала на отдельный файл. Я сделал отличную мысленную заметку, что решение было добавить что-то в файл `.cabal`. Но я не записал полнстью контекст или решение. В будущем, я увидел ошибку компоновщика и зная что нужно что-то сделать в `.cabal` файле, но я не помнил, что именно нужно. Поэтому мне приходилось повторять эту ошкбу пока я не выписал полностью решение вопроса.

# Заключение

Это часто повторяемая мантра, что практика делат всё лучше. Но кто улучшает свои навыки, может вам сказать, только хорошие практики улучшают. Плохие или безумные практики будет мешать вам двигаться дальше. Или хуже, будут прививать вам плохие привычки, которые будут требовать дополнительное время для их отмены. Взвешенная практика это процесс укрепления знания с помощью создания привычки. Вы выбираете одну и фокусируетесь на ней, и огнорируете всё остальное. Затем вы её используете пока она не попадет вам на закорку. И только потом вы двигаетесь дальше. Этот подход требует огромного терпения.

Последняя вещь, которую нужно понять о обучении нужно принять возможность ошибки. Как только это будет сделано, мы можем сделать план записи этих ошибок. Тогда мы можем изучить их и не повторять больше. Это резко увеличит скорость разработки.

---

Revision #4
Created 11 March 2022 05:27:26 by gasick

Updated 4 September 2022 11:11:26 by gasick