卒業論文

ruby_learner

-プログラミング言語Rubyの学習効率化アプリ-

関西学院大学理工学部

情報科学科 西谷研究室

27015464

大津隆輝

2019年3月

概 要

本研究室ではプログラミング言語 Ruby[1] を用いてソフト開発を行なっている．しかし，自習で実践的なプログラミングスキルを習得することは初学者にとっては難しいと考える．実践的なプログラミングスキルの例を 3 つ挙げる．1 つ目は期待される振る舞いを実現する為の言語知識である．2 つ目はコーディング規約を理解する為の言語スタイルの知識である．3 つ目は様々な開発環境に対応する為に，基本ツールに慣れることである．本研究ではこの 3 つの習得を目的とした言語学習を提供するアプリケーション ruby_learner を Ruby を用いて開発する．

本アプリケーションの基本的な機能は体系的な言語学習である．それを実現する為に，教材と課題をアプリ使用者にアプリ内で提供する．アプリ使用者はこれらを用いて言語知識のインプットとアウトプットを行う．課題の正誤は期待される振る舞いとコーディング規約の両面でチェックを行う．使用者は一連の動作を基本的なツールで行うことで，基本ツールに慣れると考える．また，課題への回答は履歴として保存しているのでいつでも確認が出来る．

秋学期の期間，このアプリケーションを週に一度の頻度で実際に 9 人の Ruby 未学習者に操作してもらった．その後，使用感についてのアンケートを行なった．肯定的な意見には，コーディング規約や基本ツールに慣れることが出来て良かったとあった．否定的な意見には，課題の難易度の問題や自分好みのコーディング規約を設定したいとあった．また，継続使用を望む者は 9 人中 2 人のみであった．本研究の目的が一部で達成できていることがわかるが，一方で多くの改善点が見つかった．

上記からわかるように，本アプリケーションは多くの問題点がある．それらを 3 つ挙げる．1 つ目は本アプリケーションを使用してレビューを行ってくれる人数が少ない点である．2 つ目はモチベーションの維持を目的とした機能がない点である．3 つ目は個人の要望に対して柔軟に本アプリケーションをカスタマイズする機能がない点である．これらを踏まえて，継続的な本アプリケーションの開発を目的として Github に開発に必要な資料を掲載している [23].

目 次

表 目 次

図 目 次

第1章　はじめに

1.1　研究の目的

西谷研究室ではプログラミング言語 Ruby を使用して，ソフト開発や卒業研究を行なっている．3 年生は Ruby の習得が早ければ自分の研究の為の時間を確保できる．しかし，言語習得を個人で行う事に関してのハードルは高い [2]．そのため，Web が提供する自習環境として，Progate[3] や codecademy[4] 等がある．しかし，それらの Web サービスが提供する，基本的な開発ツールの使用を必要としないサービスが本当に言語習得に最適かは判然としない．

コンピュータ開発では，ある言語の文法習得だけではなく，その言語やフレームを便利に使うエコシステムの習得も必要である．mac では Xcode が，windows では VisualStudio がその代表例である．これらの環境の構築は，コマンド一発で可能である．

新たなチーム開発に参加する際は，そのチームの推奨する環境に柔軟に馴染む必要があると考えられる．しかし，全ての環境を経験することは困難である．そこで，エコシステムより更に細かいツールに馴染むことで，柔軟に環境に馴染むことが出来ると考える．

次に，git などのコード管理の普及に伴って，チームでのコード編集の履歴などを通して，メンバーが他人の書いたコードを確認することがたやすくなった [5]．このような開発では他人の書いたコードを読み解き，それに適切な形で自分のコードを書き加える能力が必要となる．よって，近年の言語学習は動くだけのコードを書くのではなく言語が持つスタイルについてもきちんと学ぶ必要があると考える．

本研究では，Ruby の体系的な言語学習とプログラミングスタイルの学習ができるアプリケーションの開発を Ruby を用いて行う．そのアプリケーションを通して，基本的なツールに馴染むことで，個人の言語学習の効率化を目指す．

第2章　基本動作

2.1　Ruby

本研究で学習者が習得するプログラミング言語を決定する．以下の引用はRubyに関する記述である．

> オープンソースの動的なプログラミング言語で、シンプルさと高い生産性を備えている。エレガントな文法を持ち、自然に読み書きができる。[1]

Rubyとはプログラミング言語の一種であり，少ない記述量で期待する動作を実現できるとわかる．また，西谷研究室で使用を勧めている言語である．本研究では西谷研究室の早期学習を目的の一つとしているので，本研究で用いる言語はRubyとする．

2.2　ユーザーインターフェース

本研究で開発するアプリケーションで用いる操作方法と動作環境を決定するために，CUI[7]とGUI[8]の比較を行う．CUIとはコンピュータの操作を文字のみで行う方法であり，一方でGUIとはコンピュータの操作をマウスを用いて行う方法である．しかし，GUIでの操作には限界がある．それについて以下に引用を記す．

> GUIでしか作業を行わないというのは、使っている環境の持つ能力すべてを使いこなしていないということと同義なのです。共通作業の自動化はできないですし、ツールが持っているすべての力を出し切ることもできません。[9]

引用から，GUIは直感的に操作できるが，複雑な操作を行う際の作業効率は悪い．一方でCUIは習得が難しいが，複雑な操作を単純に行えるので作業効率は良いということがわかる．本研究の目的の一つは，実践環境に近い状況下での言語学習を実現することである．実践環境では作業効率を重視し，CUIの使用が多くなると考える．また，CUIを習

得した場合には，shell を用いた開発がメインであると考える．そこで，本研究のアプリケーションは GUI で操作する Web アプリケーションではなく，CUI で操作する shell で動作するアプリケーションとする．

2.3 RubyGems

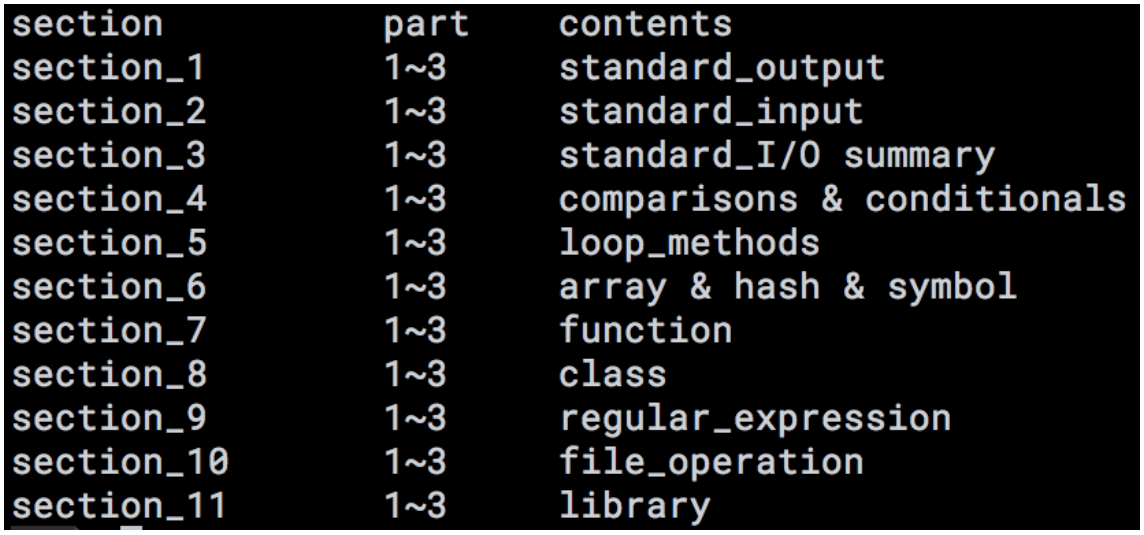
本研究で開発するアプリケーションの提供方法を決定する．RubyGems[12] とは Ruby で用いることのできるパッケージを公開することができるサービスである．公開されたパッケージは gem コマンドを用いることで容易に取得できる．取得したパッケージは shell で動作させることが可能である．本研究では，RubyGems のパッケージとして教育アプリケーションを作成することで一般に公開する．

2.4 学習方法

本アプリケーションが提供する学習方法を決定する．Ruby の言語習得のために，アプリ使用者に知識のインプットとアウトプットを繰り返し行ってもらうことで記憶の定着を図る．その為に，インプットの為の教材とアウトプットの為の課題と答えを用意する．本アプリケーション使用者はこれらの教材を読み課題を回答する．その後，答えで正解を確認することで自身のスキルを向上させることが出来ると考えられる．

2.5 教材

教材は Ruby のリファレンス [13] を参考に執筆した．使用者はこの教材で知識のインプットを行う．全 11 セクションで各 3 パートずつの構成であり，各セクションの単元は以下の通りである．



section	part	contents
section_1	1~3	standard_output
section_2	1~3	standard_input
section_3	1~3	standard_I/O summary
section_4	1~3	comparisons & conditionals
section_5	1~3	loop_methods
section_6	1~3	array & hash & symbol
section_7	1~3	function
section_8	1~3	class
section_9	1~3	regular_expression
section_10	1~3	file_operation
section_11	1~3	library

図 2.1: 教材内容.

2.6 課題

課題は教材の各単元ごとに執筆した．使用者はこの課題で知識のアウトプットを行う．この課題を適切に回答できれば，上記の教材から適切な知識をインプットできていると判断する．

2.7 回答のチェック

使用者が作成した課題への回答が期待される振る舞いをしているかどうか，コーディング規約に従っているかどうかをチェックする．Ruby はインタプリタ言語なのでコンパイルを必要としない．そこで，コンパイルなしで以下のチェックを実施することができる．

期待される振る舞い 期待される振る舞いが出来ている状態とは，コードが目的を果たす動作を適切に行なっている状態のことを示す．この項目をチェックする為にテストフレームワークを用いて各課題に対してのテストを用意する．このテストをクリアした場合に期待される振る舞いが果たされたとする．

コーディング規約 コーディング規約とはコード作成時に設ける一定のルールの事である．コーディング規約を設ける事で可読性と保守性の向上を可能にする．この項目をチェックする為に全ての課題に共通のコーディング規約を設ける．

上記の 2 つのチェックの大きな流れを以下に記す.

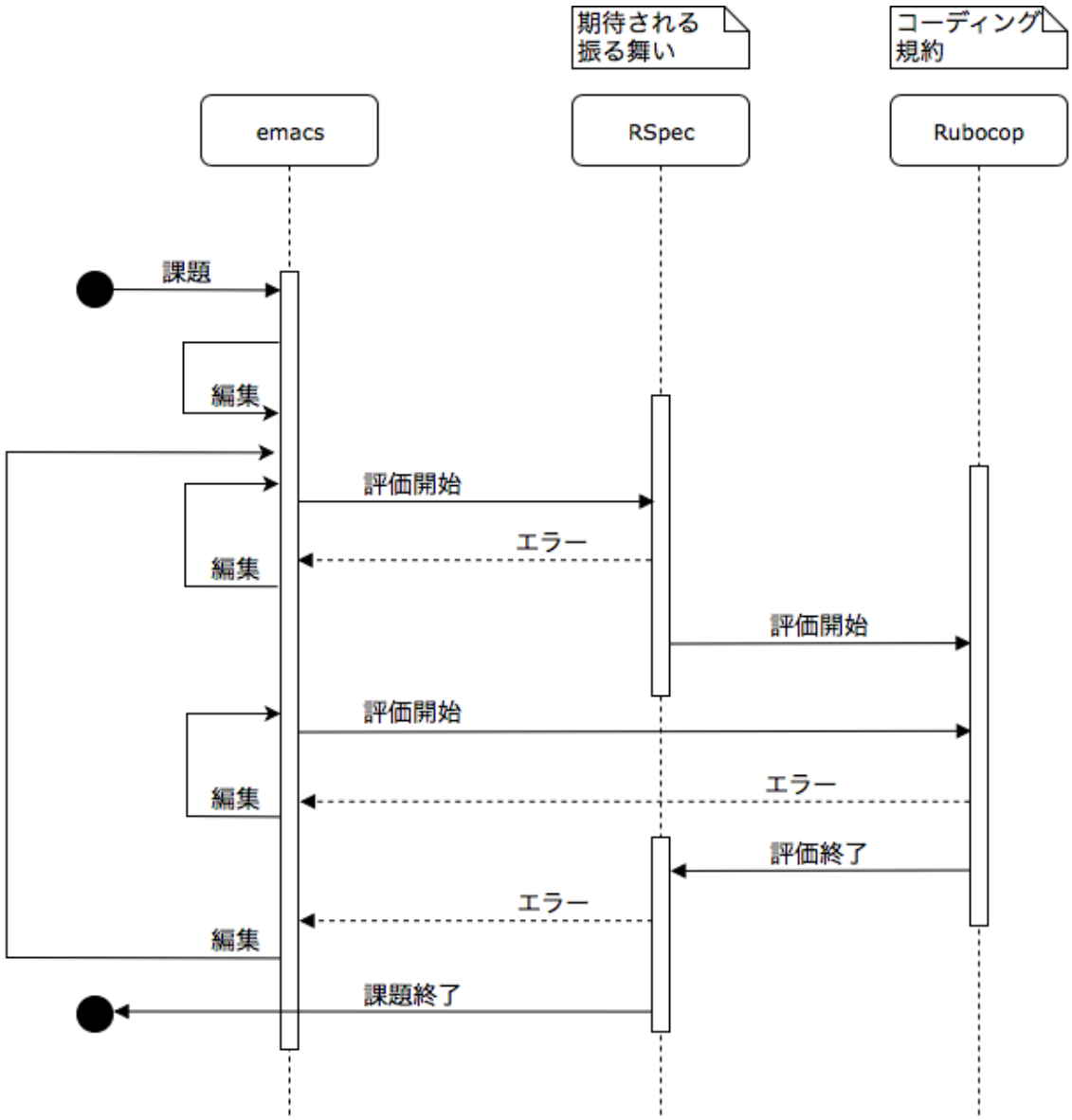


図 2.2: sequential_check のチェックフロー.

2.8 エディタ

本アプリケーション使用者が行う課題の回答に適したエディタを決定する．本研究で開発するアプリケーションは，CUI で操作する shell を用いたアプリケーションである．すなわち，エディタも shell 上で動作する必要がある．また，エディタにはプログラミングする上で便利な複数の機能を設定できるものを使用することで作業効率を上げることが

出来る．

> ツールはプログラマ自身の手の延長である。これは他のどのようなソフトウェアツールよりも Editor に対して当てはまる。テキストはプログラミングにおける最も基本的な生素材なので、できる限り簡単に操作できる必要がある。[9]

それを満たす一般的なエディタには Vim[10] と Emacs[11] の 2 つが挙げられる．西谷研究室で使用を勧めているエディタは Emacs であり，本研究では西谷研究室の早期学習を目的としている．よって本研究で用いるエディタは Emacs とする．

2.9 使用した gem ファイル

2.9.1 RSpec

RSpec[14] とは Ruby 用のテストフレームワークである．本研究の教育アプリケーションが提供する学習用の課題に対して，使用者の回答が期待される振る舞いをしているかを確認する為に用いる．

2.9.2 Rubocop

Rubocop[15] とはコードが任意のコーディング規約に従っているかを判定する gem パッケージである．本研究のアプリケーションが提供する学習用の課題に対して，使用者の回答がコーディング規約に従っているかを確認する為に用いる．

2.9.3 Thor

Thor[16] とは CLI 作成支援の gem パッケージである．本研究で作成するアプリケーションは複数の機能を内包しており，同時に CUI で実行させる必要がある．上記を実現するために Thor を用いて開発を行う．

2.10　CLI 作成ツールの比較

本アプリケーションでは複数の機能を搭載するために Thor を用いて開発を行う．しかし，他のライブラリの方が良いのではないかと疑問を持ったので，それらの比較を行う．CLI 作成ツールのライブラリとして代表的なものをいくつか挙げると，Thor，Optparse[17]，GLI[18] である．それらの比較結果を以下に示す．

表 2.1: CLI 作成ツールの比較．

	DSL	変更	インストール数
Optparse	X	△	-
Thor	○	○	218,997,330
GLI	○	○	11,351,529

(2019/01/21現在)

Optparse では OptionParser クラスを用いて CLI 作成を行う．書きやすさについて他の 2 つと比較した際に，Optparse は独特の書き方である．Ruby 経験者であっても Optparse 用に新たな学習が必要となる．また事前に用意されたクラスを用いるのでカスタマイズが難しく，サブコマンド作成や変更作業が行いにくい．

GLI と Thor の比較を行う．どちらもサブコマンドの追加や変更作業は行いやすいが，コードの書きやすさについて違いがある．Thor は Thor クラスの継承を行うだけで基本的な CLI 作成は終了する．一方で，GLI は Ruby に似た書き方であるが, 設定を細かく書いていかなければいけない．それぞれをまとめた表が上記の表である．次にインストール数には大きな差があった．これより，より広く普及しているのは Thor であることがわかる．

本アプリケーションは継続的な開発を期待しているので，より多くの人が使用している Thor での開発が好ましいと考える．よって，現段階で新規に CLI を作成する場合は Thor を選択することが最適であると考える．

第3章　ruby_learnerの詳細

3.1　Install/Uninstall

Install と Uninstall 方法は以下のコマンドを実行することで可能である．

Install $ gem install ruby_learner

Uninstall $ gem uninstall ruby_learner

3.2　ruby_learnerの全コマンド

本アプリケーションには多くのコマンドを用意している．それぞれのコマンドには学習を効率的に行える機能を実装している．その一覧は以下の通りである．各コマンドの説明は順次行なっていく．

```
Usage: ruby_learner [options]
Options:
        -v                                          - check ruby_learner's version.

        emacs_key (-e)                              - confirm emacs key-bindings.
        emacs_key -image (-i)                       - confirm emacs key-bindings in the image.

        sequential_check (-s) [sec:1~11] [par:1~3]  - learning drill.
        sequential_check -drill (-d)                - confirm drill numbers and contents.
        sequential_check -next (-n)                 - learn next to final history.
        sequential_check -last (-l)                 - learn final history.
        sequential_check -workshop (-w)             - move current_directory to workshop.
        sequential_check -manual (-m)               - change mode manual or nomal.
        sequential_check -copspec (-c)              - check your answer.

        restore (-r)                                - check your restore.
        restore [number]                            - open your number's restore.
        restore -refresh (-r)                       - clear all your restore.

        copspec (-c) [file_path]                    - check your original code in ruby_learner.
        pair_popup (-p) [time]                      - popup a alert per times, for pair_programing.
        install_emacs (-i)                          - install emacs in your mac.
        theme (-t) [black or white]                 - change ruby_learner's theme.
```

図 3.1: ruby_learner のコマンド一覧.

3.3 sequential_check

Ruby に関する教材と課題を体系的に学習できるモードである．教材と課題によって知識のインプットアウトプットを促し，知識の定着を図る．学習効率を高める為に複数のオプションを用意している．基本的な学習画面は以下の通り，画面を 2 分割して上部に回答コード，下部に教材と課題を表示させている．

図 3.2: sequential_check の学習画面.

以下はこのモードの詳細である.

3.3.1 normal_mode

sequential_check が提供する教材と課題の受講方法の 1 つ．デフォルトではこちらがアクティブになっているが，manual_mode から切り替える場合は [$ ruby_learner -s -m] を実行することで可能である．normal_mode の状態では，sequential_check の学習コマンドの実行時に起こる一連の流れ (図 2.2) を半自動で実行される．使用者が操作するのは課題回答後のエラー発生時における [回答修正][答え確認][途中終了] の 3 つのみである．答え確認を選択した場合は，emacs の画面が 3 分割され，新たな領域に答えが記載される．このモードでは最低限の操作で学習に集中できる．言語の学習に集中することで，早期の言語習得が可能であると考える.

3.3.2 manual_mode

sequential_check が提供する教材と課題の受講方法の 1 つ．normal_mode から切り替える場合には [$ ruby_learner -s -m] を実行することで可能である．normal_mode の状態とは違い，sequential_check の学習コマンドの実行後に複数の操作が必要となる．以下にその詳細を記す．

ディレクトリ構造 本アプリケーションは初回起動時にホームディレクトリに.ruby_learner という隠しディレクトリを作成している．そしてこのディレクトリには workshop というディレクトリが存在し，そこで教材と課題の取り組みを行う必要がある．workshop 内には lib と spec という 2 つのディレクトリがある．lib には「教材と課題が一緒になった sentence.org」「使用者が課題の回答を記入する workplace.rb」「課題の答えが記載されている answer.rb」の 3 つのファイルが存在している．spec には課題のチェックを行うテストファイルが存在している．以下はそのディレクトリ構造の一覧である．

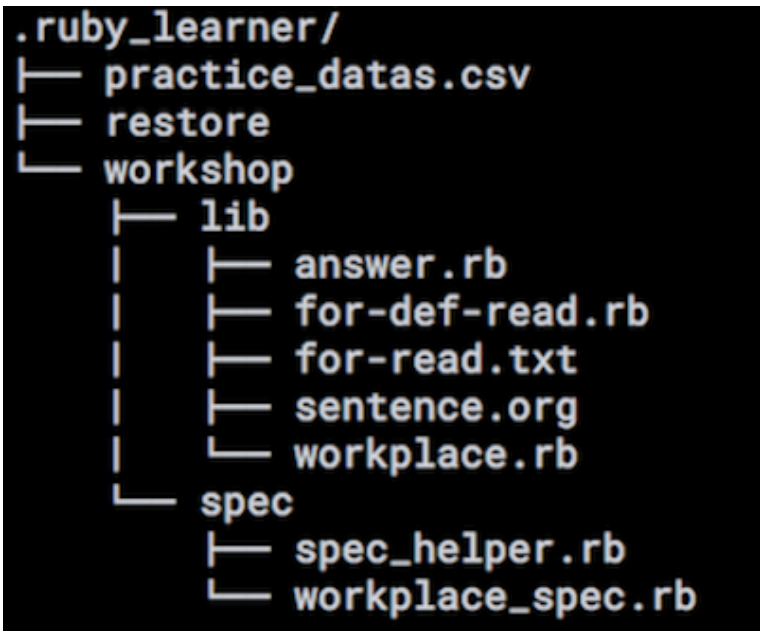


図 3.3: .ruby_learner のディレクトリ構造.

教材と課題の取り組み 使用者は手動で emacs を用いて sentence.org と workplace.rb を開く必要がある．そのコマンドの一例として [$ emacs sentence.org workplace.rb] が挙げられる．そして sentence.org に記された教材を読み, 課題を理解した後に，その課

題に対する答えを workplace.rb に記入することで課題に対して回答を行ったと判断される．

チェックの実行 使用者が回答したコードのチェックを行うには通常の rspec と rubocop のコマンドを使用する方法以外に，ruby_learner がもつコマンドを使用する方法がある．それが [$ ruby_learner -s -c] を実行する方法である．このコマンドでは normal_mode と同様のチェックを workplace.rb に保存されたコードに行うことができる．このチェックが正常に終了すると，使用者の回答コードが期待される振る舞いとコーディング規約の両面で合格していることとなる．

このモードでは多くの操作を手動で行うことで，開発で用いるディレクトリ構造の把握や実際の開発サイクルの習得が可能であると考える．

3.3.3 last

最新の課題回答を再開することができるオプション．.ruby_learner/workshop のディレクト内に存在するファイル全てに一番最後に回答していた回答コードと教材課題等がコピーされる．一度中断した最新の課題回答を途中から再開できる為，学習効率が向上すると考える．

3.3.4 next

最新の課題の次の課題を行うことのできるオプション．.ruby_learner/workshop のディレクト内に存在するファイル全てに次の教材課題等がコピーされる．通常の学習コマンドが [$ sequential_check 1 1] の様にセクション番号とパート番号の入力を必要とするのとは違い，[$ sequential_check -n] のみで次の課題を行える為，学習効率が向上すると考える．

3.3.5 workshop

.ruby_learner/workshop にディレクトリの移動を行うオプション．manual_mode では作業ディレクトリに手動で.ruby_learner/workshop にする必要がある．どのディレクトリに

いてもコマンド一つで作業ディレクトリに移動できるので，操作性の向上を目指してこの機能を実装した．

3.4 restore

今まで取り組んできた課題に対する回答コードの一覧を確認できるコマンド．使用者は学習を行う際に履歴を保存することを意識せずに，学習を行った順に履歴が保存されていく．自動的に履歴を作成することで自分が積み上げてきた過程を確認できるので使用者は次の学習計画を立てやすくなる．

3.4.1 open

詳細なコードを確認したい場合に用いるオプション．回答コードの復習を可能としている．このオプションを利用することで，過去の自分が書いたコードに対してレビューするという学習方法が可能になる．この学習方法は自分の成長を感じれるほかに，コードの修繕を行うことはより可読性の高いコードや動作効率の良いコードの作成することが可能となる．

3.4.2 refresh

履歴の一斉削除を行うためのコマンド．本アプリケーションでは限度なく履歴を保存し続けていく仕様なので，容量が増えすぎた場合は任意のタイミングでこのオプションの使用を推奨する．

3.5 pair_popup

ペアプログラミングを行う際に，役割交代を報告するためのアラートを表示させるコマンド．任意の秒数でアラートを表示させることができる．このコマンドでは，本アプリケーションを用いる学習方法にペアプログラミングを導入することを可能にしている．ペアプログラミングでは自分以外の使用者の意見を得られる点で，個人学習より高い学習効

率を発揮すると考えられる．また，本アプリケーションの目的でもある実戦的な開発に近い学習を実現するためにも，他者と協力するペアプログラミングの導入を実践した．

3.6 install_emacs

emacs をインストールするコマンド．初学者にとって一つの壁である環境構築は基本的に本アプリケーションのインストール時に bundler を用いて自動的に最低限は完了している．しかし，emacs に関してはインストールに非常に時間がかかるので任意のタイミングでインストールできる様にコマンドとしている．コマンドの実行時にはシェルスクリプトを実行することで，半自動で emacs のインストールが開始される．

3.7 emacs_key

emacs のキーバインド [19] を確認することのできるコマンド．本アプリケーションでは CUI での操作を推奨している．emacs にはキーバインドと呼ばれる操作コマンドが存在する．これらの習得を果たすことで emacs での開発効率が向上すると考えられる．そこで，このコマンドでは使用者が確認したい時にすぐにキーバインド一覧を表示できる様にしている．

3.7.1 string

各種キーバインドを文字で表示するオプション．キーバインドがリスト状の一覧で表示され，簡易的に表示できる．ウィンドウが新規に作成されることがないので，本アプリケーション外での開発でも気軽に emacs のキーバインドを確認することができる．実際にコマンドを実行した場合に表示されるキーバインドのリストを以下に記す．

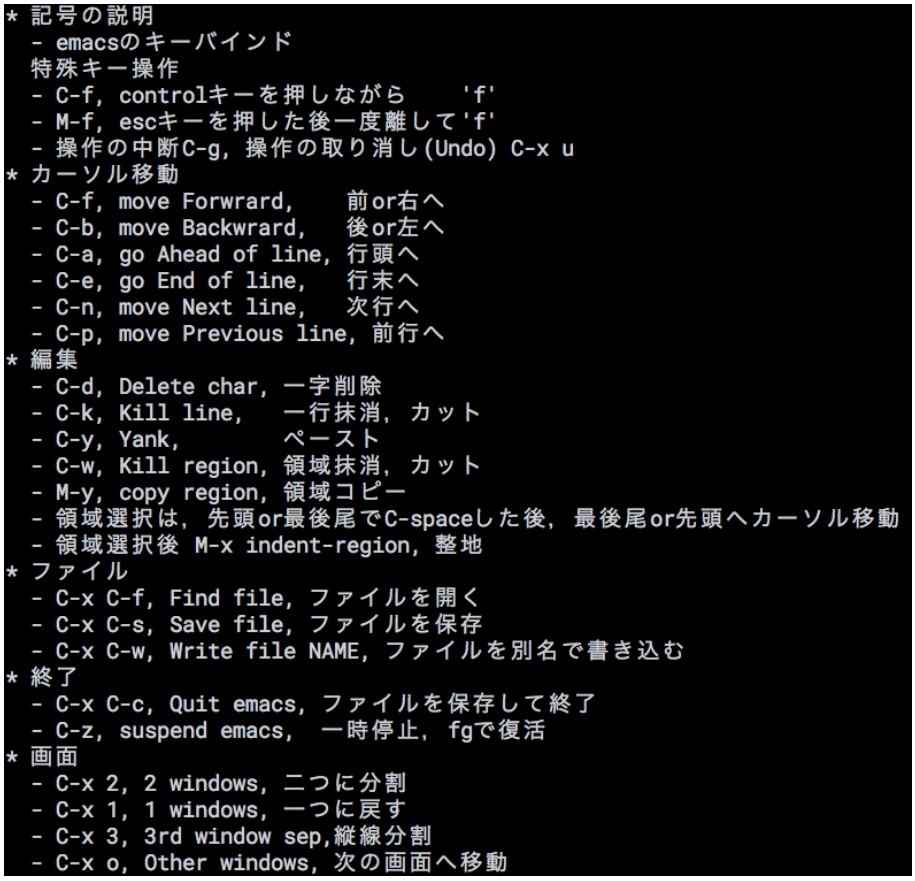


図 3.4: emacs キーバインドの文字表示.

3.7.2 image

各種キーバインドを画像で表示するオプション．キーバインドが視覚的に表現されているため，初学者にとってイメージしやすい表示となっている．しかし，画像表示のために新規ウィンドウが立ち上がるのでウィンドウ操作はマウスでなければならない場合がある．実際にコマンドを実行した場合に表示されるキーバインドの画像を以下に記す.

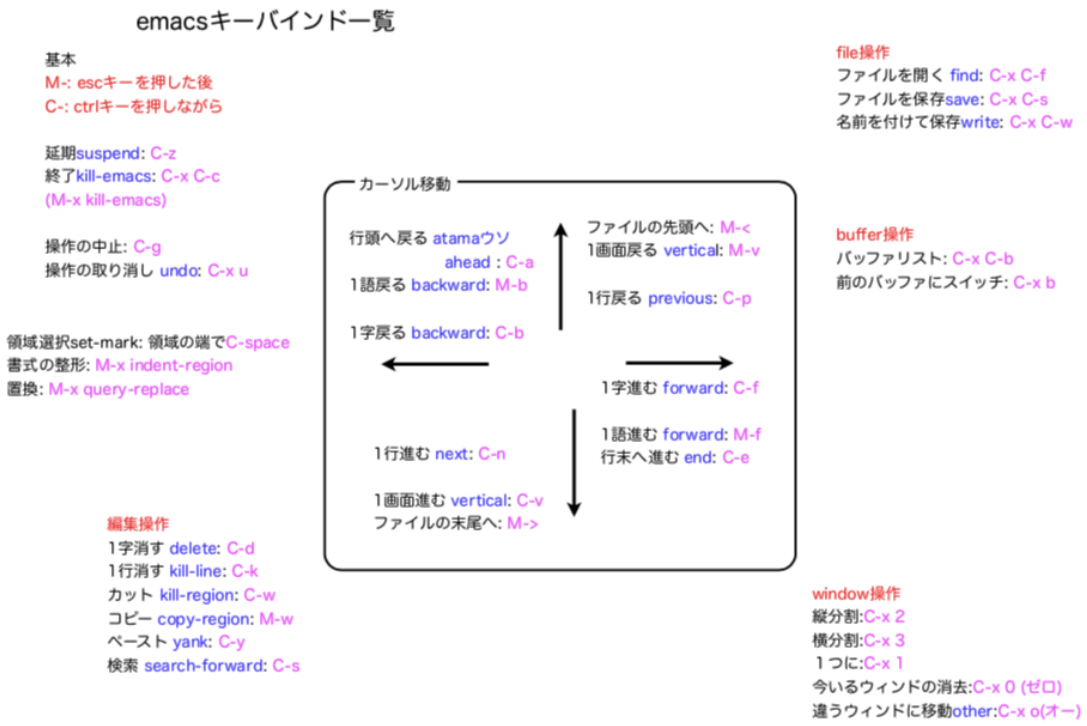


図 3.5: 西谷の Emacs の keybind 早見表. [20].

3.8 theme

sequential_check の normal_mode で使用する emacs のテーマの変更が可能．現在のテーマは黒と白の 2 種類から選ぶことができる．

第4章　考察

4.1　競合するサービスとの比較

現在，インターネット上にはプログラミング言語の学習を目的としたサービスが多く存在し，ドットラーニング [21]，paiza ラーニング [22]，progate，codecademy があげられる．それらのサービスと本研究で開発したアプリケーションとの比較を以下に示す．

表 4.1: 競合サービスの比較．

	サービス形態	コード作成	教材	有料機能
ドットラーニング	Web	なし	動画	動画内コード公開
paizaラーニング	Web	Webエディタ	動画	質問
progate	Web	Webエディタ	テキスト	テキスト
codecademy	Web	Webエディタ	テキスト	質問・上位講義
ruby_learner	Shell	Emacs	テキスト	なし

ruby_learner を除く一般的なサービスは GUI で操作する Web サービスである．しかし，実際の開発環境は Web ではなく shell で行う場合が多いと考えられる．また，サービス内で使用するコード記入に用いられるエディタは ruby_learner 以外はサービス外で行う開発には用いることはできない．上記の 2 点からも分かるように，実際の開発に用いられるツールを使用できる面で ruby_learner が優れている．さらに，有料機能がないので使用者に快適な学習を提供できると考えられる．上記の点より，ruby_learner とその他の競合するサービスについてとの間には明確な違いがあることがわかる．その上で，ruby_learner には優れている点が存在すると考える．

本研究で作成したアプリケーションのソースコードは Github 上に掲載している [23]．これにより今後の機能追加が可能となっているので，さらに学習に必要とされる機能に関しては順次追加することもできる．この点からも分かるように，独自のカスタマイズができる点も ruby_learner の強みであると考える．

4.2 使用者からの評価

実際に 9 名の Ruby 未学習者に本研究で作成したアプリケーションを使用してもらい，本研究の評価を行う．

肯定的な評価をいくつか得られた．以下は被験者の評価の内，肯定的な評価をリスト状にしたものである．

1. Rubocop で優れた書き方へ誘導してくれる点
2. 一つの画面でテキストと課題に取り組める点
3. CUI での操作に慣れることが出来る点
4. Emacs のキーバインドを習得できる点
5. 課題の答えをいつでも確認できる点

CUI 操作や Emacs のキーバインドに慣れる事を実感している使用者がいることがわかる．また，Rubocop を通して得られるコーディング規約の学習に対しても肯定的な評価を得られたことがわかる．

一方で否定的な評価も得られた．以下は被験者の評価の内，否定的な評価をリスト状にしたものである．

1. 課題に関数の使い方などのヒントが欲しい
2. 他言語バージョンが欲しい
3. Rubocop のコーディング規約を変更したい

使用者が課題に対するヒントが欲しいと感じている点は，課題の難易度の変更が出来ないので感じているのではないかと考える．Rubocop の言語スタイルのレベルについても，所属するチームによって違うので，任意で変更出来る仕様にする必要を感じる．

肯定的な評価から分かるように，本アプリケーションは CUI 操作に慣れることが出来る点や使用者のコードのチェック方法について肯定的な評価を得た．使用者のコードのチェックには RSpec と Rubocop の 2 つを用いた．このチェック方法によって，使用者のコードは当事者目線ではなく定量的なチェックを受けることが可能であったため，このような評価に繋がったと考える．以上から，本アプリケーションを用いる事で，使用者は効率的な学習が出来ていると考えられる．

しかし，今回の実験以外で本アプリケーションを使用したいかという問いに対して，使用したいと答えた人は9人中2人のみであった．本アプリケーションには多くの改善点がある．そのうちの一つとして，モチベーション維持を目的とする機能が必要であると考える．またコーディング規約や課題の難易度の部分で，個人個人が求めている学習の詳細なスタイルにバラつきが見受けられた．この解決策として，個人でのカスタマイズ機能の追加も検討する．

第5章　総括

本研究では効率的な学習を通して，西谷研究室で必要となるスキルの習得を目指したアプリケーションを開発した．そして，このアプリケーションは試作品を用意して，毎週使用者に使ってもらうことで意見をもらい改善を行ってきた．これにより，実際に使用者に求めれる機能を実装することができたと考えられる．

しかし，本アプリケーションについてはいくつかの問題点が存在する．前章でも記述したように，まだまだ学習効率をあげるために足りない機能が多い．それを補う機能の実装や改善が必要となる．また，本アプリケーション自体の評価は使用者からの評価としている．さらに学習の効率を向上させる為には多くの使用者からの評価が必要となる．これらの問題点が本アプリケーションの問題点となると考える．

そこで，本アプリケーションの改善のためには追加での研究が必要だと考える．本研究を容易に引き継ぐことが出来るように，ソースコードを含む開発資料を Github 上に掲載している．こうする事で，本アプリケーションの継続的な改良を可能としている [23]．

参考文献

[1] Ruby ホームページ, https://www.ruby-lang.org/ja/, accessed 2019.1.30.

[2] 押見太雄,「Learning Style Inventory を用いたプログラミング初学者のための教材レコメンドシステムの提案」, 慶應義塾大学総合政策学部卒業論文 (2017), p.1.

[3] Progate. https://prog-8.com/mypage, accessed 2019.1.30.

[4] codecademy. https://www.codecademy.com/, accessed 2019.1.30.

[5] Git. https://git-scm.com/book/ja/v2/, accessed 2019.1.30.

[6] Souki Wada, editor_learner 1.1.2. https://rubygems.org/gems/editor_learner/, accessed 2019.1.30.

[7] Weblio, CUI. https://www.weblio.jp/content/CUI, accessed 2019.1.30.

[8] Weblio, GUI. https://ejje.weblio.jp/content/GUI, accessed 2019.1.30.

[9] Andrew Hunt, David Thomas, 「達人プログラマー」, (オーム社, 2016 年).

[10] Vim. https://www.vim.org, accessed 2019.1.30.

[11] Emacs. https://www.gnu.org/software/emacs/, accessed 2019.1.30.

[12] RubyGems. https://rubygems.org, accessed 2019.1.30.

[13] Ruby リファレンス. https://docs.ruby-lang.org/ja/, accessed 2019.1.30.

[14] RSpec. http://rspec.info/about/, accessed 2019.1.30.

[15] RuboCop. http://batsov.com/rubocop/, accessed 2019.1.30.

[16] Thor. https://rubygems.org/gems/thor/versions/0.19.1, accessed 2019.1.30.

[17] Optparse. https://docs.ruby-lang.org/ja/latest/library/optparse.html, accessed 2019.1.30.

[18] gli. https://rubygems.org/gems/gli/versions/2.14.0, accessed 2019.1.30.

[19] Weblio, キーバイド. https://www.weblio.jp/content/キーバインド, accessed 2019.1.30.

[20] 西谷の Emacs の keybind 早見表. https://ist.ksc.kwansei.ac.jp/~nishitani/?cmd=view&p=EmacsPrimary&key=emacs, accessed 2019.1.30.

[21] ドットラーニング. https://dotinstall.com, accessed 2019.1.30.

[22] paiza ラーニング. https://paiza.jp, accessed 2019.1.30.

[23] Takaki Otsu, ruby_learner. https://github.com/Takaki0403/ruby_learner, accessed 2019.1.30.