

卒業論文

プログラミング言語 Ruby 学習の効率化アプリの  
開発

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

# 目次

<b>第1章</b>	<b>はじめに</b>	<b>5</b>
1.1	研究の目的 . . . . .	5
1.2	研究の動機 . . . . .	5
<b>第2章</b>	<b>基本動作</b>	<b>6</b>
2.1	Ruby . . . . .	6
2.2	ユーザーインターフェース . . . . .	6
2.3	RubyGems . . . . .	6
2.4	学習方法 . . . . .	7
2.5	教材 . . . . .	7
2.6	課題 . . . . .	7
2.7	回答の評価 . . . . .	8
2.8	エディタ . . . . .	9
2.9	使用した gem ファイル . . . . .	9
2.9.1	RSpec . . . . .	9
2.9.2	Rubocop . . . . .	9
2.9.3	Thor . . . . .	9
2.10	CLI 作成ツールの比較 . . . . .	10
<b>第3章</b>	<b>ruby_learner の概要</b>	<b>11</b>
3.1	Install/Uninstall . . . . .	11
3.2	ruby_learner の全コマンド . . . . .	11
3.3	sequential_check . . . . .	12
3.3.1	normal_mode . . . . .	12
3.3.2	manual_mode . . . . .	13

3.3.3	last . . . . .	14
3.3.4	next . . . . .	14
3.3.5	workshop . . . . .	14
3.4	restore . . . . .	14
3.4.1	open . . . . .	15
3.4.2	refresh . . . . .	15
3.5	pair_popup . . . . .	15
3.6	install_emacs . . . . .	15
3.7	emacs_key . . . . .	16
3.7.1	string . . . . .	16
3.7.2	image . . . . .	17
3.8	theme . . . . .	18
<b>第 4 章</b>	<b>競合するサービスとの比較</b>	<b>19</b>
4.1	考察 . . . . .	19
<b>第 5 章</b>	<b>他のソフトとの比較</b>	<b>20</b>
5.1	考察 . . . . .	20
<b>第 6 章</b>	<b>総括</b>	<b>21</b>

# 表 目 次

2.1	CLI 作成ツールの比較.	10
4.1	競合サービスの比較.	19

# 目 次

2.1	教材内容.	7
2.2	sequential_check の評価フロー.	8
3.1	ruby_learner のコマンド一覧.	11
3.2	sequential_check の学習画面.	12
3.3	emacs キーバインドの文字表示.	17
3.4	emacs キーバインドの文字表示.	18

# 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、言語学習や卒業研究を行なっている。つまり、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、個人で言語学習を行う際のハードルは高いものである [1]。その原因として、言語に対しての知識がない初心者にとって環境構築や学習の教材選びは非常に困難であることが要因の一つではないかと考えられる。環境構築なしで教材選びも必要ない言語学習サービスには Progate[2] や codecademy[3] 等があるが、環境構築という部分を排斥しているが故にそのサービス外での学習に一定の壁が生じているのではないかと考える。また、近年ではコードを Git で管理することで、チーム内でのコード編集の履歴をメンバーがそれぞれ確認することが出来るので、チームでの開発が管理しやすくなった [4]。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。そういった意味でも、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。そこで本研究では、環境構築の自動化に加え、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を開発し、Ruby を容易に学習できる環境を整え、実践的な学習を提供することで、個人のスキル向上の効率化を目指す。

## 1.2 研究の動機

本研究室に在籍していた和田によって開発されたタイピングアプリケーション editor\_learner の再開発を検討していたが、本研究室で行う開発に必要なスキルの習得に焦点を当てた開発に切り替えた。ここでのスキルとは、動くだけでなくコーディング規約に則ったコードが書けることや開発に必要な基本的なツールの使い方の習得のことである。

## 第2章 基本動作

### 2.1 Ruby

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．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 と GUI 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．CUI は GUI よりも習得が難しいが，複雑な操作を単純に行えるので作業効率は良い．GUI は CUI よりも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．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．私は，実践環境では作業効率を重視し，shell を用いた開発がメインであると考え．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

RubyGems 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 のライブラリ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

アプリが提供する学習方法を決定する。Ruby の言語学習のために、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。インプットの為の教材とアウトプットの為の課題と解答を用意する。アプリ使用者はこれらの教材を読み課題を回答し、解答で正解を確認することで自身のスキルを向上させる。

## 2.5 教材

教材は Ruby のリファレンスを参考に執筆した。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。使用者はこの教材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 回答の評価

使用者が作成した課題への回答が期待される振る舞いとコーディング規則に従っているかどうか評価する。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、以下で解説する評価項目にシンタックスチェックは含まない。

**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。この項目を評価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ことである。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。この項目を評価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つの評価の大きな流れを以下に記す。

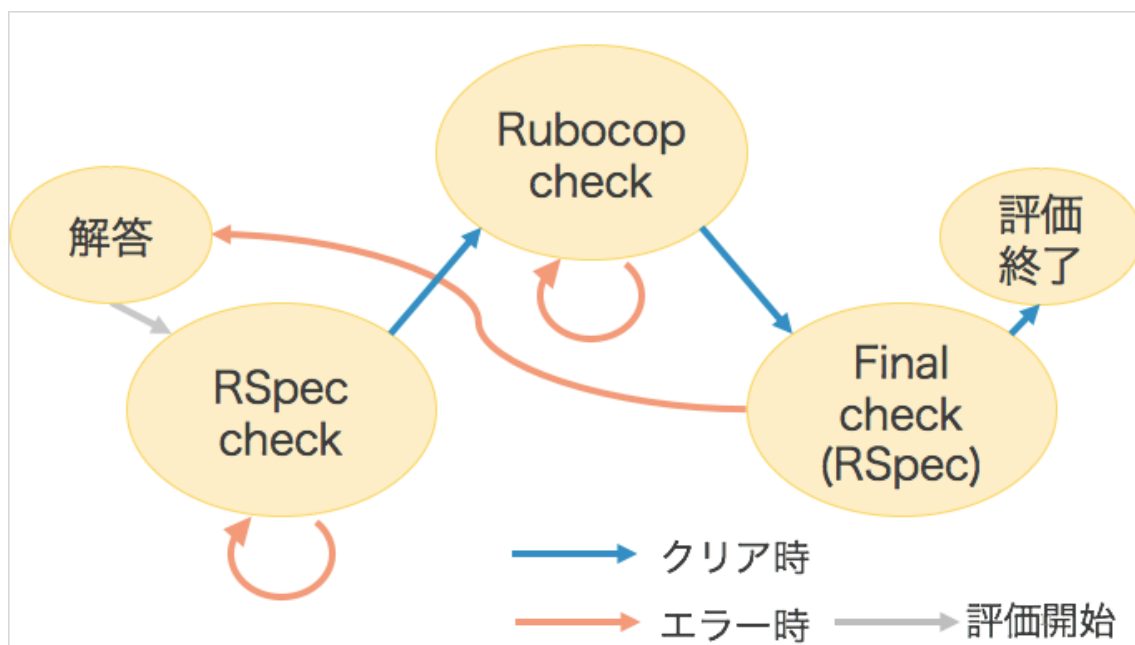


図 2.2: sequential\_check の評価フロー。

## 2.8 エディタ

アプリ使用者が行う課題の回答に適したプログラミング可能なエディタを決定する．上記で本研究で開発するアプリケーションは，CUIで操作する shell で操作するアプリケーションとしているので，エディタも shell 上で動作する必要がある．それを満たす一般的なプログラミング可能なエディタは Vim と Emacs の 2 つである．本研究が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ので，本研究で用いるテキストエディタは Emacs とする．

## 2.9 使用した gem ファイル

### 2.9.1 RSpec

RSpec 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 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ライブラリである．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 とは CLI 作成支援のライブラリである．本研究で作成する教育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、GLI 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。

	書きやすさ	サブコマンド追加	変更の行い易さ
Optparse	X	△	△
Thor	○	○	○
GLI	△	○	○

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ので Ruby 経験者であっても Optparse 用に学習が必要となる。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 作成は終了するが、GLI は Ruby に似た書き方であるが1から細かく書いていかなければいけない。このそれぞれをまとめた表が上記の表である。上記の表からも分かる通り、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える。

## 第3章 ruby\_learner の概要

### 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.

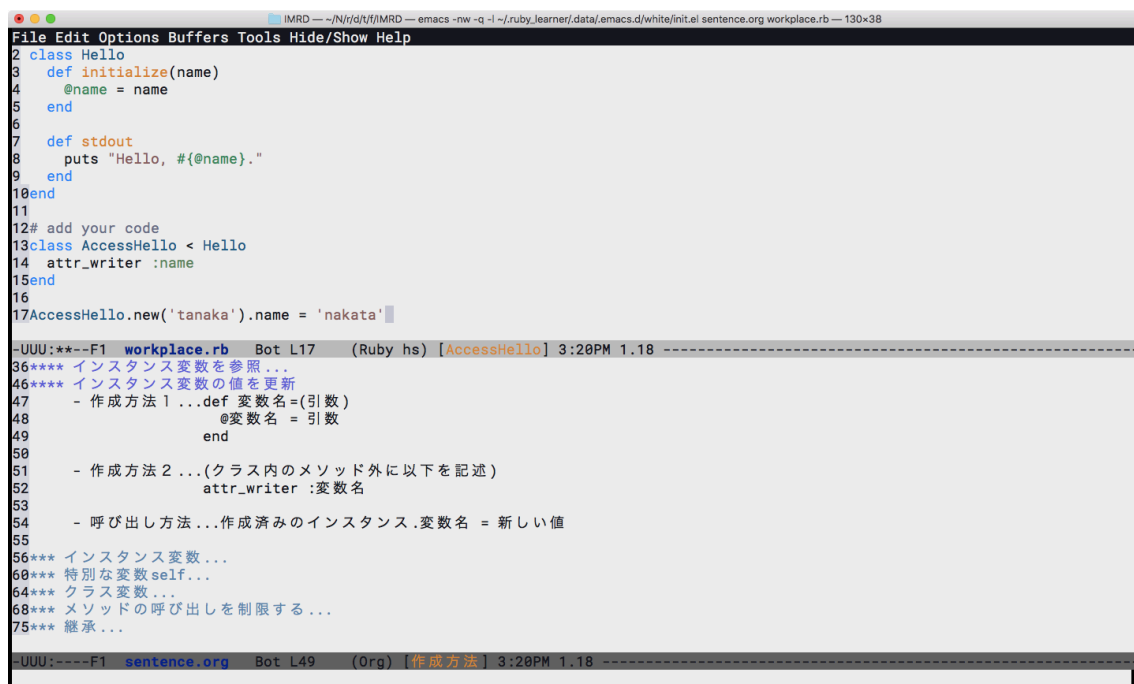
  copspec (-c) [file_path]          - check your original code in ruby_learner.
  pair_popup (-p) [time]            - popup a alert per times, for pair_programing.
  install_emacs (-i)                - install emacs in your mac.
  theme (-t) [black or white]       - change ruby_learner's theme.
```

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

プッシュを用意している。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。



```
File Edit Options Buffers Tools Hide/Show Help
2 class Hello
3   def initialize(name)
4     @name = name
5   end
6
7   def stdout
8     puts "Hello, #{@name}."
9   end
10 end
11
12 # add your code
13 class AccessHello < Hello
14   attr_writer :name
15 end
16
17 AccessHello.new('tanaka').name = 'nakata'

-UUU:--F1 workplace.rb Bot L17 (Ruby hs) [AccessHello] 3:20PM 1.18 -----
36*** インスタンス変数を参照...
46*** インスタンス変数の値を更新
47   - 作成方法1...def 変数名=(引数)
48       @変数名 = 引数
49   end
50
51   - 作成方法2...(クラス内のメソッド外に以下を記述)
52       attr_writer :変数名
53
54   - 呼び出し方法...作成済みのインスタンス.変数名 = 新しい値
55
56*** インスタンス変数...
60*** 特別な変数self...
64*** クラス変数...
68*** メソッドの呼び出しを制限する...
75*** 継承...

-UUU:--F1 sentence.org Bot L49 (Org) [作成方法] 3:20PM 1.18 -----
```

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][解答確認][途中終了] の3つのみである。

### 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`」「課題の解答が記載されている `answer.rb`」の3つのファイルが存在している。 `spec` には課題の評価を行うテストファイルが存在している。

**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、その課題に対する解答を `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。

**評価の実行** 使用者が回答したコードの評価を行うには通常 `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考えられる。

### 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` のディレクトリ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` のディレクトリ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践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。

```
* 記号の説明
- emacsのキーバインド
特殊キー操作
- C-f, controlキーを押しながら 'f'
- M-f, escキーを押した後一度離して 'f'
- 操作の中断C-g, 操作の取り消し(Undo) C-x u
* カーソル移動
- C-f, move Forward, 前or右へ
- C-b, move Backward, 後or左へ
- C-a, go Ahead of line, 行頭へ
- C-e, go End of line, 行末へ
- C-n, move Next line, 次行へ
- C-p, move Previous line, 前行へ
* 編集
- C-d, Delete char, 一字削除
- C-k, Kill line, 一行抹消, カット
- C-y, Yank, ペースト
- C-w, Kill region, 領域抹消, カット
- M-y, copy region, 領域コピー
- 領域選択は, 先頭or最後尾でC-spaceした後, 最後尾or先頭へカーソル移動
- 領域選択後 M-x indent-region, 整地
* ファイル
- C-x C-f, Find file, ファイルを開く
- C-x C-s, Save file, ファイルを保存
- C-x C-w, Write file NAME, ファイルを別名で書き込む
* 終了
- C-x C-c, Quit emacs, ファイルを保存して終了
- C-z, suspend emacs, 一時停止, fgで復活
* 画面
- C-x 2, 2 windows, 二つに分割
- C-x 1, 1 windows, 一つに戻す
- C-x 3, 3rd window sep, 縦線分割
- C-x o, Other windows, 次の画面へ移動
```

図 3.3: 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。

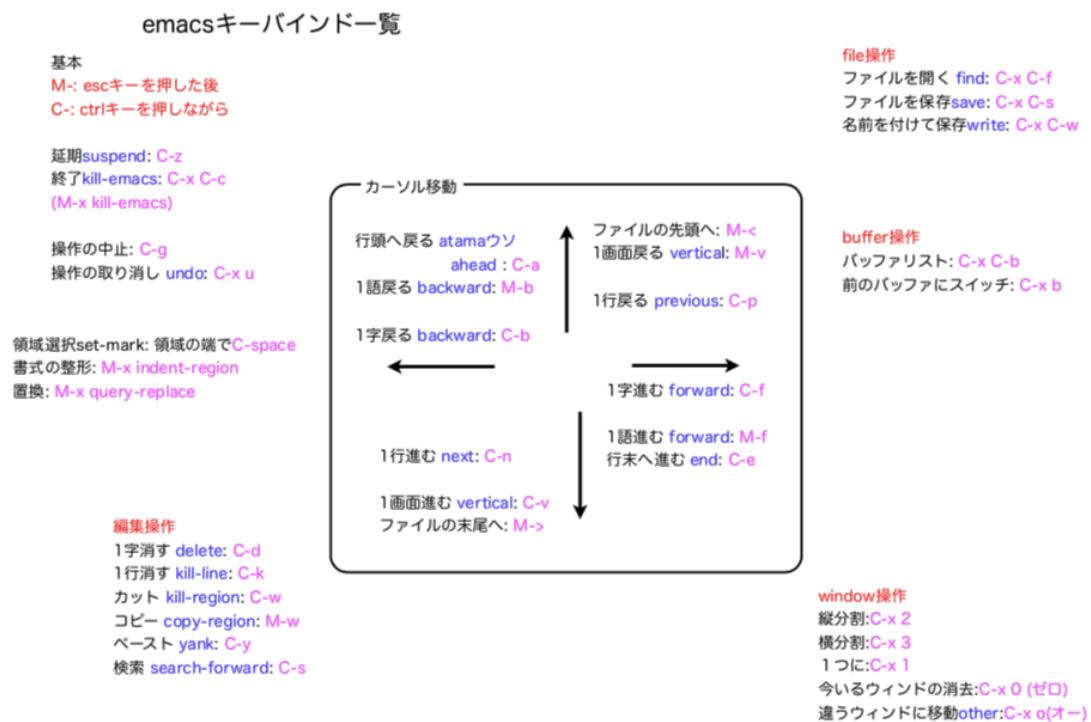


図 3.4: emacs キーバインドの文字表示.

## 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 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する．その中でも人気のサービスは，ドットラーニング，paiza ラーニング，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

### 4.1 考察

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，実際の開発体験に近い環境での学習では ruby\_learner が優れており，サービス外での学習へのサポートも兼ね備えていることがわかる．また有料機能がない点で，使用者にストレスレスな学習を提供できると考えられる．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 上に掲載しており，OSS 開発を行えるようにしている．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．

## 第5章 他のソフトとの比較

他のタイピングソフトとの比較を行った表が以下の通りである。上記のタイピングソフトは自分もよく使っていたタイピングソフトであり、評価も高いソフトである。それぞれの特徴は以下の通り。

1. PTYPING: 豊富なプログラミング言語がタイピング可能
2. e-typing: 資格取得にもつながる練習が可能。間違いが多い箇所を指摘してくれる。
3. 寿司打: 自分が一番よく使ったソフト、GUI ベースで飽きずに継続しやすい。

それぞれの特徴があるが、人気ソフトの中でもプログラムの実行が可能なソフトは発見できなかった。プログラマにとってコードを書いて実行しないのは、テストを受けて結果を見ないのと同義である。また、これらのソフトは全て Web 上で行なっており、editor は全く使わない。プログラマにとってコードだけ書いて editor を使って実行しないことなどない。よっていかに editor\_learner がプログラマ向けのソフトかがわかる。

### 5.1 考察

editor\_learner は様々な機能を有しており、{} や () などの約物の記入、editor 操作やプログラムの実行などが可能な点であり、editor 操作によるキーバインドの習熟が期待される。

GUI ベースのソフトに比べて継続性では劣るが、CUI ベースの editor\_learner ではファイルの保存、開閉による操作を習熟することができ作業を効率化、高速化することができる。これらの理由により editor\_learner では目的に沿った技術の向上が期待される。

## 第6章 総括

実際に今までたくさんのタイピングソフトやプログラムコードの打てるタイピングソフトを数多く利用してきたが，editor 操作の習熟が可能なソフトは見たことも聞いたこともなかった．実際にタイピングだけが早い学生はたくさんいるが editor 操作やキーバインドも使いこなせる学生は少なかった．本研究で開発した editor\_learner によりそれらの技術も上達し，作業効率などの向上が期待される．

# 謝辞

本研究を行うにあたり，終始多大なるご指導，御鞭撻をいただいた西谷滋人教授に対し，深く御礼申し上げます．また，本研究の進行に伴い，様々な助力，知識の供給をいただきました西谷研究室の同輩，先輩方に心から感謝の意を示します．本当にありがとうございました．

## 参考文献

- [1] Andrew Hunt, David Thomas, 「達人プログラマー」, (オーム社, 2016 年).
- [2] Ruby ホームページ, <https://www.ruby-lang.org/ja/>, accessed 2018.2.8.
- [3] S. Koichiro, Rubygems のススメ, <https://qiita.com/sumyapp/items/5ec58bf3567e557c24d7>, accessed 2018.2.8.
- [4] Weblio, キーバインド, <https://www.weblio.jp/content/キーバインド>, accessed 2018.2.8.
- [5] Weblio, CUI, <https://www.weblio.jp/content/CUI>, accessed 2018.2.8.
- [6] S. Kouichiro, Thor の使い方, <https://qiita.com/succi0303/items/32560103190436c9435b>, accessed 2018.2.8.
- [7] 伊藤淳一, 「プロを目指す人のための Ruby 入門」, (技術評論社, 2017 年)
- [8] SideCI, Rubocop の使用, <http://blog-ja.sideci.com/entry/2015/03/12/160441>, accessed 2018.2.9
- [9] Ruby, 相対パス, <https://www.xmisao.com/2013/11/21/ruby-require-relative.html>, accessed 2018.2.8.