

卒業論文

command line による editor 操作の習熟プログラム

関西学院大学理工学部

情報科学科 西谷研究室

27014533

2018 年 3 月

目 次

第 1 章	はじめに	4
1.1	研究の目的	4
1.2	研究の動機	4
第 2 章	基本的事項	5
2.1	Emacs	5
2.2	Ruby	6
2.3	RubyGems	6
2.4	Keybind	7
2.5	CUI(Character User Interface)	8
2.6	使用した gem ファイル	8
2.6.1	diff-lcs	8
2.6.2	Thor	9
2.6.3	Minitest	9
2.6.4	FileUtils	10
2.6.5	open3	10
2.6.6	Bundler	10
2.6.7	Rubocop	11
第 3 章	editor_learner の概要	12
3.1	Installation	12
3.1.1	github による install	12
3.1.2	gem による install	12
3.2	uninstall	12
3.2.1	github から install した場合の uninstall 方法	12

3.2.2	gem から install した場合の uninstall 方法	13
3.3	動作環境	13
3.3.1	error 時の対処法	13
3.4	初期設定	14
3.5	delete	16
3.6	random_h.rb と sequential_h.rb	16
3.7	random_check の動作	19
3.8	sequential_check の動作	19
第 4 章	実装コードの解説	21
4.1	起動時に毎回動作するプログラム	21
4.1.1	プログラム内のインスタンス変数の概要	23
4.1.2	File の作成	24
4.2	ファイル削除処理 delete	24
4.3	random_check	25
4.4	sequential_check	27
4.4.1	インスタンス定数に格納されたパス	29
4.4.2	動作部分	30
4.5	新しいターミナルを開く open_terminal	30
第 5 章	他のソフトとの比較	31
5.1	PTYING	32
5.2	e-typing	32
5.3	寿司打	32
5.4	考察	32
第 6 章	総括	33
第 7 章	謝辞	34

目 次

2.1	カーソル移動系,	7
2.2	文字操作系,	7
3.1	作られるファイルの説明.	15
3.2	random_h.rb.	17
3.3	全ての操作を終えたターミナル画面.	18
3.4	sequential_h.rb.	18
5.1	他のソフトとの比較.	31

第1章 はじめに

1.1 研究の目的

editor_learner の開発の大きな目的は editor(Emacs) 操作, CUI 操作 (キーバインドなど), Ruby 言語の習熟とタイピング速度, 正確性の向上である. editor 上で動かすためファイルの開閉, 保存, 画面分割といった CUI 操作を習熟することができ, Ruby 言語のプログラムを写経することで Ruby 言語の習熟へと繋げる. 更にコードを打つことで正しい運指を身につけタイピング速度の向上も図っている. コードを打つ際にキーバインドを利用することでキーボードから手を離すことなくカーソル移動などのコマンドを GUI ではなく CUI 操作で行うことにより作業の効率化にも力を入れている. これら全てはプログラマにとって作業を効率化させるだけでなく, プログラマとしての質の向上につながる.

1.2 研究の動機

初めはタッチタイピングを習得した経験を活かして, 西谷によって開発された shunkun-type(ターミナル上で実行するタイピングソフト) の再開発をテーマにしていたが, これ以上タイピングに特化したソフトを開発しても同じようなものが Web 上に大量に転がっており, そのようなものをいくつも開発しても意味がなく, それ以外の付加価値を付けたソフトを開発しようと考えた. そこで西谷研究室ではタイピング, Ruby 言語, Emacs による editor 操作, CUI 操作の習熟が作業効率に非常に大きな影響を与えるので習熟を勧めている. そこでこれらの習熟を目的としたソフトを開発しようと考えた.

第2章 基本的事項

2.1 Emacs

本研究において使用する editor は Emacs である。

ツールはプログラマ自身の手延長である。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ります。[1]

と書かれている。そこで西谷研究室で勤められている Emacs の機能については以下の通りである、

1. 設定可能である。 フォント、色、ウィンドウサイズ、キーバインドを含めた全ての外見が好みに応じて設定できるようになっていること。通常の操作がキーストロークだけで行えると、手をキーボードから離す必要がなくなり、結果的にマウスやメニュー駆動型のコマンドよりも効率的に操作できるようになります
2. 拡張性がある。 新しいプログラミング言語が出てきただけで、使い物にならなくなるようなエディタではなく、どんな新しい言語やテキスト形式が出てきたとしても、その言語の意味合いを「教え込む」ことが可能です
3. プログラム可能であること。 込み入った複数の手順を実行できるよう、Editor はプログラム可能であることが必須である。

これらの機能は本来エディタが持つべき基本的な機能である。これらに加えて Emacs は、

1. 構文のハイライト Ruby の構文にハイライトを入れたい場合はファイル名の後に.rb と入れることで Ruby モードに切り替わり構文にハイライトを入れることが可能になる。

2. 自動インデント. テキストを編集する際、改行時に自動的にスペースやタブなどを入力しインデント調整を行ってくれる.

などのプログラミング言語に特化した特徴を備えています. 強力な editor を習熟することは生産性を高めることに他ならない. カーソルの移動にしても, 1 回のキー入力で単語単位, 行単位, ブロック単位, 関数単位でカーソルを移動させることができれば, 一文字ずつ, あるいは一行ずつ繰り返してキー入力を行う場合とは効率が大きく変わってきます. Emacs はこれらの全ての機能を孕んでいて editor として非常に優秀である. よって本研究は Emacs をベースとして研究を進める.

2.2 Ruby

Ruby の基本的な説明は以下の通り,

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ます. エレガントな文法を持ち, 自然に読み書きができます. [2]

本研究は Ruby 言語を使用しています. 大きな理由としては強力な標準ライブラリなどを持っており, 構文も自由度が高く記述量も少ない. よって, 縛りが少ないのでとっかかりやすくプログラミング言語の中で習熟しやすいと考えたからである.

2.3 RubyGems

Rubygem の基本的な説明は以下の通り,

RubyGems は, Ruby 言語用のパッケージ管理システムであり, Ruby のプログラムと ("gem" と呼ばれる) ライブラリの配布用標準フォーマットを提供している. gem を容易に管理でき, gem を配布するサーバの機能を持つ. [3]

要は, Ruby 言語で書かれたプログラムをより便利にするためのツール (ライブラリ) が簡単に使えるように配布されている場所, 機能です. 本研究では RubyGems の gem を利用してファイル操作やパスの受け取りなどを行い, 本研究で開発したソフトも gem に公開してある. これにより非常に簡単に本研究で開発したソフトを install できるようになっている.

2.4 Keybind

Keybind とは,

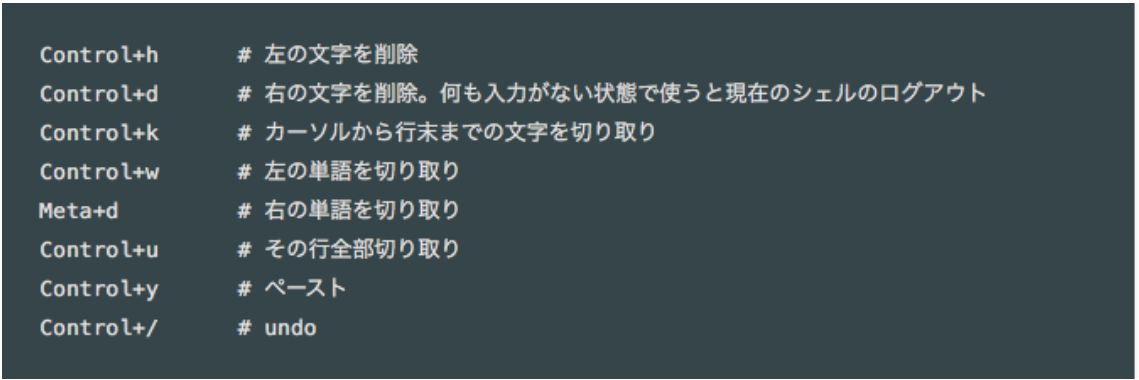
押下するキー (単独キーまたは複数キーの組み合わせ) と, 実行される機能との対応関係のことである. また, キーを押下したときに実行させる機能を割り当てる行為のことである. [4]

以下 control を押しながらを c-と記述する. 本研究における Keybind の習熟は CUI 操作の習熟に酷似している. カーソル移動においても GUI ベースでマウスを使い行の先頭をクリックするより, CUI により c-a を押すことで即座に行の先頭にカーソルを持つていくことができる. 習熟するのであれば, どちらの方が早いかは一目瞭然である. 本研究は Keybind の習熟による CUI 操作の適応で作業の効率化, 高速化に重点を置いている. よく使用されるキーバインドは以下の通り.



Control+f	# 右に移動
Control+b	# 左に移動
Meta+f	# 右に単語単位移動
Meta+b	# 左に単語単位移動
Control+e	# 行末へ移動
Control+a	# 行頭に移動
Control+n	# 下
Control+p	# 上

図 2.1: カーソル移動系,



Control+h	# 左の文字を削除
Control+d	# 右の文字を削除. 何も入力がない状態で使うと現在のシェルのログアウト
Control+k	# カーソルから行末までの文字を切り取り
Control+w	# 左の単語を切り取り
Meta+d	# 右の単語を切り取り
Control+u	# その行全部切り取り
Control+y	# ペースト
Control+/ Control+z	# undo

図 2.2: 文字操作系,

2.5 CUI(Character User Interface)

CUI は,

コンピュータにおいて, キーボード入力と文字表示のみを用いた, ソフトウェアの操作体系. キーボードのコマンド名を入力して操作する方法など. [5]

CUI と GUI にはそれぞれ大きな違いがある. GUI の利点は以下の通り,

- 文字だけでなくアイコンなどの絵も表示できる.
- 対象物が明確な点や, マウスで比較的簡単に操作できる.
- 即座に操作結果が反映される.

CUI の利点は以下の通り,

- コマンドを憶えていれば複雑な処理が簡単に行える.
- キーボードから手を離すことなく作業の高速化, 効率化が行える.

今回 GUI ではなく CUI 操作の習熟を目的にした理由は,

- コマンドを憶えることで作業効率が上がる.
- editor 操作の習熟も孕んでいるから.

カーソル移動においても GUI ではなく CUI 操作により, ワンコマンドで動かした方が効率的である. プログラマにとって editor を使わないことなどない. 上記の理由から, GUI ではなく CUI 操作の習熟を目的としている.

2.6 使用した gem ファイル

2.6.1 diff-lcs

diff-lcs は, 二つのファイルの差分を求めて出力してくれる. テキストの差分を取得するメソッドは, Diff::LCS.sdiff と Diff::LCS.diff の 2 つがある. 複数行の文字列を比較した場合の 2 つのメソッドの違いは以下のとおり.

1. Diff::LCS.sdiff

1. 比較結果を 1 文字ずつ表示する

2. Diff::LCS.diff

1. 比較した結果, 違いがあった行について, 違いがあった箇所のみ表示する.

今回使用したのは後者 (Diff::LCS.diff) である. コード自体が長いので全ての比較結果を表示してしまうとものすごく長くなってしまう. よって, 違いがあった行のみの表示となっている.

2.6.2 Thor

Thor は,

コマンドラインツールの作成を支援するライブラリです. git や bundler のようなサブコマンドツールを簡単に作成することができます. [6]

Thor の使用でサブコマンドを自然言語に近い形で設定できるので, 非常にコマンドが覚えやすくなっている.

2.6.3 Minitest

Minitest はテストを自動化するためのテスト用のフレームワークである. Ruby にはいくつかのテストフレームワークがありますが, Minitest というフレームワークを利用した理由は以下の通りです.

1. Ruby をインストールすると一緒にインストールされるため, 特別なセットアップが不要.
2. 学習コストが比較的低い.
3. Rails のデフォルトのテストフレームワークなので, Rails を開発するときにも知識を活かしやすい.

上記の理由から, sequential.check では minitest を採用しております.

2.6.4 FileUtils

再帰的な削除などの基本的なファイル操作を行うためのライブラリ今回使用したファイル操作は以下の通り.

1. `File.join`: ファイルのパスとパスを結合する
2. `File.exist`: ファイルが存在するかどうかの判定. ファイルが存在して入れば”true”存在していなければ”false”を返り値としている.
3. `FileUtils.mkdir_p`: ディレクトリ `dir` とその親ディレクトリを全て作成する.
4. `FileUtils.touch`: パスに設定されているファイルを作成する.
5. `FileUtils.cp`: ファイルのコピーを行う.
6. `File.expand_path`: パスを絶対パスに展開した文字列を返す.
7. `FileUtils.compare_file`: 二つのファイルの比較を行う. 一致して入れば”true”一致していなければ”false”を返り値としている.

2.6.5 open3

`open3` は,

プログラムを実行し, そのプロセスの標準入力, 標準出力, 標準出力にパイプをつなぎます. [7]

今回は `open3` を利用して `gem` で `install` した場合に `editor_learner` が格納されているパスを取得する.

2.6.6 Bundler

`Bundler` は,

`gem` 同士の互換性を保ちながらパッケージの種類やバージョンを管理してくれる仕組みのことです. 複数人, 複数環境で開発を行う際に書く環境で扱うパッケージの種類やバージョンを合わせてくれる. [8]

2.6.7 Rubocop

Rubocop は Ruby のソースコード解析ツールである。Ruby スタイルガイドや他のスタイルガイドに準拠しているかどうかを自動チェックしてくれるソフトウェアです。自分が打ち込んだ問題文となるソースコードのチェックに使用した。

第3章 editor_learner の概要

3.1 Installation

3.1.1 github による install

github によるインストール方法は以下の通りである。

1. ”https://github.com/souki1103/editor_learner” へアクセス
2. Clone or download を押下, SSH の URL をコピー
3. コマンドラインにて `git clone`(コピーした URL) を行う

上記の手順で開発したファイルがそのまま自分のディレクトリにインストールされる。

3.1.2 gem による install

gem によるインストール方法は以下の通りである。

1. コマンドラインにて `gem install editor_learner` と入力, 実行
2. ファイルがホームディレクトの `.rbenv/versions/2.4.0/lib/ruby/gems/2.4.0/gems` に `editor_learner` が収納される

これで `editor_learner` とコマンドラインで入力することで実行可能となる。

3.2 uninstall

3.2.1 github から install した場合の uninstall 方法

github から install した場合の uninstall 方法は以下の通りである。

1. ホームディレクトで

1. `rm -rf editor_learner` を入力

2. ホームディレクトリから `editor_learner` が削除されていることを確認する.

以上が `uninstall` 方法である.

3.2.2 gem から install した場合の uninstall 方法

gem から install した場合の `uninstall` 方法は以下の通りである.

1. ターミナル上のコマンドラインで

1. `gem uninstall editor_learner` を入力

2. ホームディレクトの `.rbenv/versions/2.4.0/lib/ruby/gems/2.4.0/gems` に `editor_learner` が削除されていることを確認する.

以上が `uninstall` 方法である.

3.3 動作環境

Ruby の version が 2.4.0 以上でなければ動かない. 理由としては, gem に格納されているパスを正しく受け渡しできないからである. 2.4.0 以下で動作させるためには `editor_learner` の最新 version のみを入れることによって動作することが確認できている.

3.3.1 error 時の対処法

error が出た場合は以下の方法を試してください

1. `rm -rf editor_learner` をコマンドラインで入力

これによりファイル生成によるバグを解消できる. もう一つの方法は

1. `gem uninstall editor_learner` をコマンドラインで入力

2. 全ての version を uninstall する.
3. 再度 `gem install editor_learner` で最新 version のみを install する.

上記の手順により Ruby の version によるバグが解消されることが確認できている．現在起こるであろうと予想されるバグの解消法は上記の2つである．Ruby の version が 2.4.0 以上であればなんの不具合もなく動作することが確認できている．

3.4 初期設定

特別な初期設定はほとんどないが起動方法は以下の通りである，

1. コマンドライン上にて `editor_learner` を入力する.
2. `editor_learner` を起動することでホームディレクトリに `editor_learner/workshop` と呼ばれるファイルが作成される．workshop は作業場という意味である．
3. workshop の中に `question.rb` と `answer.rb`, `random_h.rb` と `ruby_1`~`ruby_6` が作成され，`ruby_1`~`ruby_6` の中に `1.rb`~`3.rb` が作成されていることを確認する．

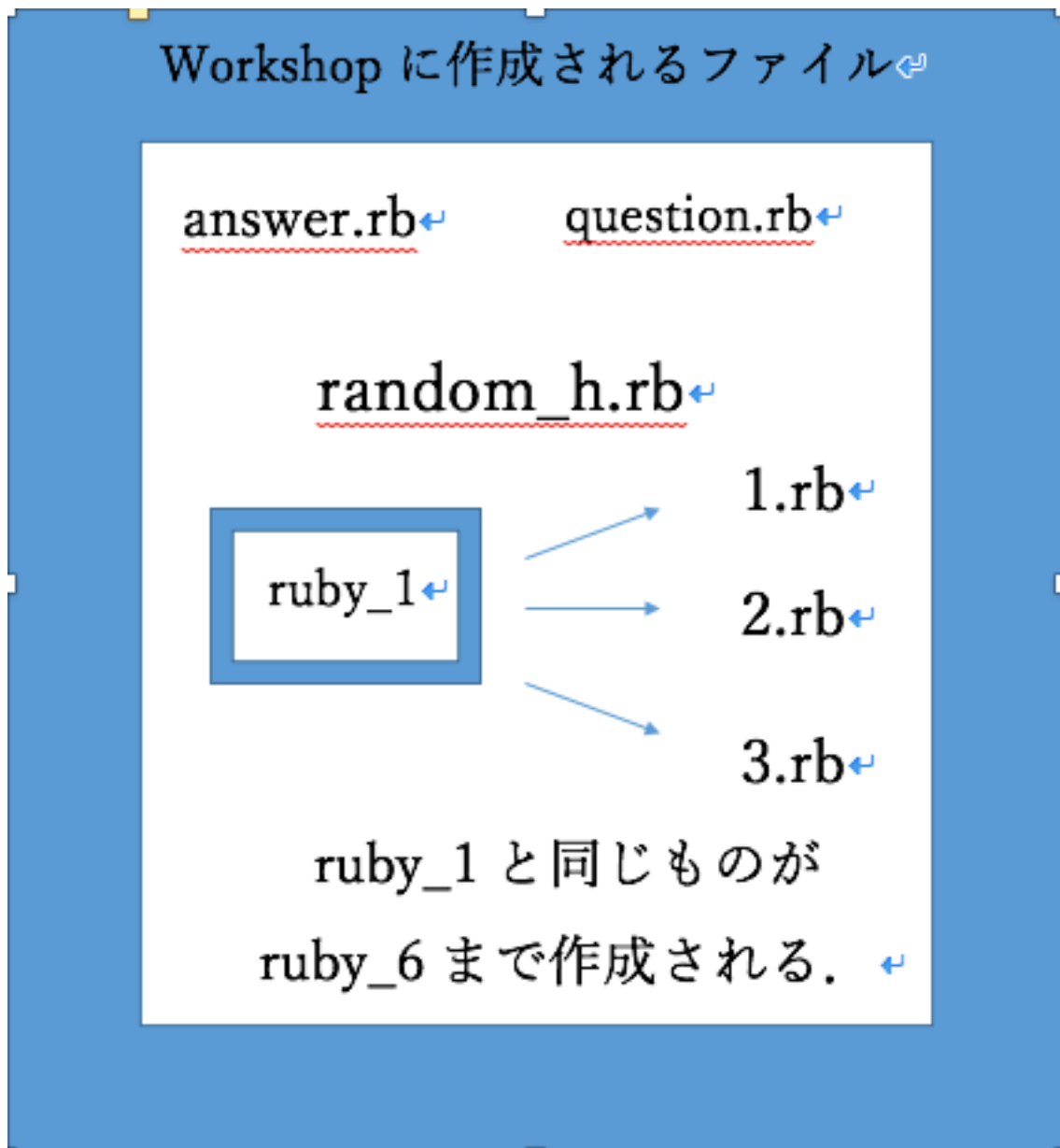


図 3.1: 作られるファイルの説明.

1. 起動すると以下のようなサブコマンドの書かれた画面が表示されることを確認する.

Commands:

```
editor_lerner delete [number~number]
```

```
editor_learner help [COMMAND]
```

```
editor_learner random_check
```

```
editor_lerner sequential_check [lesson_number] [1~3numbers]
```


2. `editor_learner` の後にサブコマンドと必要に応じた引数を入力すると動作する．それぞれのサブコマンドの更に詳しい説明は以下の通りである．

3.5 delete

`editor_learner` を起動することで初期設定で述べたようにホームディレクトリに `editor_learner/workshop` が作成される．`delete` は `workshop` に作成された `ruby_1~ruby_6` を削除するために作成されたものである．`sequential_check` で1度プログラムを作成してしまふと再度実行すると `It have been finished!` と表示されてしまうので，削除するコマンドを作成しました．コマンド例は以下の通りである．

コマンド例

1. `editor_learner delete 1 3`

上記のように入力することで1～3までのファイルが削除される．サブコマンドの後の引数は2つの数字 (char 型) であり，削除するファイルの範囲を入力する．

3.6 random_h.rb と sequential_h.rb

`random_h.rb` と `sequential_h.rb` が初期設定で作成され，`editor_learner` を起動することで自動的に作成され，`random_check` と `sequential_check` を行う際に最初に関くファイルとなる．`random_check` 用と `sequential_check` 用に二つのファイルがある．`random_check` 用のファイルは以下の通りである．

`random_h.rb`

```
File Edit Options Buffers Tools Ruby Help
# to open question.rb
# c-x 2: split window vertically
# c-x c-f: find file and input question.rb
# open a.rb as above
# c-x 3: split window horizontally
# c-x c-f: find file and input answer.rb
# move the other window
# c-x o: other window
# then edit answer.rb as question.rb

# c-a: move ahead
# c-d: delete character
# c-x c-s: save file
# c-x c-c: quit edit
# c-k: kill the line
# c-y: paste of killed line
```

図 3.2: random_h.rb.

上から順に説明すると、1. question.rb を開くために c-x 2 で画面を 2 分割にする。1. c-x c-f で question.rb を探して開く。1. 次に answer.rb を開くために画面を 3 分割する 1. 同様に c-x c-f で answer.rb を探して開く。1. c-x o で answer.rb を編集するためにポインタを移動させる。1. question.rb に書かれているコードを answer.rb に写す。これらの手順が random_h.rb に記述されている。全ての手順を終えたターミナルの状態は以下の通り、

```
File Edit Options Buffers Tools Ruby Help
# coding: utf-8
country = 'italy'

if country == 'japan'
  'こんにちは'
elsif country == 'us'
  'Hello'
elsif country == 'italy'
  'ciao'
else
  numbers = [1, 2, 3, 4]
  sum = 0
  numbers.each do |n|
    sum += n
  end
  sum
end
-UUU:----F1 answer.rb Top L1 G -UU-:----F1 question.rb All L1 Gi
# to open question.rb
# c-x 2: split window vertically
# c-x c-f: find file and input question.rb
# open a.rb as above
# c-x 3: split window horizontally
# c-x c-f: find file and input answer.rb
# move the other window
# c-x o: other window
# then edit answer.rb as question.rb
```

図 3.3: 全ての操作を終えたターミナル画面.

上記の画像では、右上に問題である question.rb が表示され、それを左上にある answer.rb に写す形となる.

次に sequential_h.rb

```
#to open q.rb
# c-x 2: find file and input q.rb
# c-x c-f: find file and input q.rb
# open 1-3.rb as above
# c-x 3: split find file and input 1-3.rb
# move the other window
# c-x o: other window
# then edit 1-3.rb q.rb

# c-a:move ahead
# c-d: delete character
# c-x c-s: save file
# c-x c-c: quit edit
# c-k: kill the line
# c-y: paste of killed line
```

図 3.4: sequential_h.rb.

書かれている内容自体は random_h.rb とほとんど差異がないが、開くファイルの名前が違うため別のファイルとして作成された. この手順に沿って作業することになる. 下に書

かれているのは主要キーバインドであり，必要に応じて見て，使用する形となっている．
上記の手順を行なったターミナル画面の状態は `random_h.rb` の最終形態を同じである．

3.7 random_check の動作

`random_check` の動作開始から終了は以下の通りである．

1. コマンドライン上にて `editor_learner random_check` を入力
2. 新しいターミナル (ホームディレクトリ/`editor_learner/workshop` から始まる) が開かれる．
3. `random_h.rb` を開いて `random_h.rb` に沿って `question.rb` に書かれているコードを `answer.rb` に写す．
4. 前のターミナルに戻り，コマンドラインに”check”と入力することで正誤判定を行ってくれる．
5. 間違っていれば `diff-lcs` により間違った箇所が表示される．
6. 正しければ新しいターミナルが開かれてから終了までの時間と `It have been finished!` が表示され終了となる．

更に次回 `random_check` 起動時には前に書いたコードが `answer.rb` に格納されたままなので全て削除するのではなく，前のコードの必要な部分は残すことができる．

`random_check` の大きな目的は typing 速度，正確性の向上，editor 操作や Ruby 言語の習熟に重点を置いている．いかに早く終わらせるかのポイントが typing 速度，正確性と editor 操作である．

3.8 sequential_check の動作

`sequential_check` の動作開始から終了は以下の通りである．

1. コマンドライン上で `editor_learner sequential_check(1~6の数字) (1~3の数字)` を入力
2. 新しいターミナル (ホームディレクトリ/`editor_learner/workshop/ruby_(1~6の数字)`) が開かれる．

3. sequential.h.rb を開いて sequential.h.rb に沿って q.rb に書かれている内容を第 2 引数の数字.rb に写す.
4. 前のターミナルに戻り, コマンドラインに”check”と入力することで正誤判定を行う.
5. 間違っていれば間違った箇所が表示される. 再度 q.rb と第 2 引数の数字.rb を開いて間違った箇所を修正する.
6. 正しければ ruby_1/1.rb is done! のように表示される.

sequential_check は 1~3 の順に 1.rb がリファクタリングや追加され 2.rb になり, 完成形が 3.rb になるといった形式である. 連続的なプログラムの完成までを写経するので sequential_check と名付けられた.

sequential_check の大きな目的はリファクタリングによる Ruby 言語の学習と CUI 操作によるキーバインドの習熟, タイピング速度, 正確性の向上に重点を置いている. コードがリファクタリングされる様を写経することで自分自身で Ruby のコードを書くときに他の人が見やすくなるようなコードが書けるようになる.

第4章 実装コードの解説

本章では，今回作成したプログラムをライブラリ化し継続的な発展が可能なようにそれぞれの処理の解説を記述する．

4.1 起動時に毎回動作するプログラム

`editor_learner` を起動したときに自動に動く部分である．コードは以下の通りである．

```

def initialize(*args)
  super
  @prac_dir="#{ENV['HOME']}/editor_learner/workshop"
  @lib_location = Open3.capture3("gem environment gemdir")
  @versions = Open3.capture3("gem list editor_learner")
  p @latest_version = @versions[0].chomp.gsub(' (', '-').gsub(')', '')
  @inject = File.join(@lib_location[0].chomp, "/gems/#{@latest_version}/lib")
  if File.exist?(@prac_dir) != true then
    FileUtils.mkdir_p(@prac_dir)
    FileUtils.touch("#{@prac_dir}/question.rb")
    FileUtils.touch("#{@prac_dir}/answer.rb")
    FileUtils.touch("#{@prac_dir}/random_h.rb")
    if File.exist?("#{@inject}/random_h.rb") == true then
      FileUtils.cp("#{@inject}/random_h.rb", "#{@prac_dir}/random_h.rb")
    elsif
      FileUtils.cp("#{ENV['HOME']}/editor_learner/lib/random_h.rb",
        "#{@prac_dir}/random_h.rb")
    end
  end
end

range = 1..6
range_ruby = 1..3
range.each do|num|
  if File.exist?("#{@prac_dir}/ruby_#{num}") != true then
    FileUtils.mkdir("#{@prac_dir}/ruby_#{num}")
    FileUtils.touch("#{@prac_dir}/ruby_#{num}/q.rb")
    FileUtils.touch("#{@prac_dir}/ruby_#{num}/sequential_h.rb")
    if File.exist?("#{@inject}/sequential_h.rb") == true then
      FileUtils.cp("#{@inject}/sequential_h.rb", "#{@prac_dir}/ruby_#{num}/sequential_h.rb")
    else
      FileUtils.cp("#{ENV['HOME']}/editor_learner/lib/sequential_h.rb",
        "#{@prac_dir}/ruby_#{num}/sequential_h.rb")
    end
  end
  range_ruby.each do|n|
    FileUtils.touch("#{@prac_dir}/ruby_#{num}/#{n}.rb")
  end
end

end

end
end
end

```

この部分は基本的にディレクトリやファイルの作成が主である。上から順に説明すると、`@prac_dir` はホームディレクトリ/`editor_learner/workshop` を指しており、ファイルを作る際のパスとして作成されたインスタンス定数である。その後の3つのインスタンス定数(`@lib_location`,`@versions`,`@latest_version`) は gem で install された場合ファイルの場所がホームディレクトリ/`.rbenv/versions/2.4.0/lib/ruby/gems/2.4.0/gems` の `editor_learner` に格納されているため gem で install した人と github で install した人とではパスが変わってしまうためこれらの3つのインスタンス定数を用意した。実際の振る舞いとしては、`File.exist` により `prac_dir` がなければディレクトリを作成しさらにその中に `question.rb` と `answer.rb` を作成する。gem にリリースしていることから gem で install した人と github で install した人のパスの違いを if 文で条件分岐させている。これにより `random_h.rb` を正常にコピーすることができた。

4.1.1 プログラム内のインスタンス変数の概要

インスタンス変数は、`'@'` で始まる変数はインスタンス変数であり、特定のオブジェクトに所属しています。インスタンス変数はそのクラスまたはサブクラスのメソッドから参照できます。初期化されない孫スタンス変数を参照した時の値は `nil` です。

このメソッドで使用されているインスタンス変数は5つである。`prac_dir` はホームディレクトリ/`editor_learner/workshop` を指しており、必要なファイルをここに作るのでパスとして受け渡すインスタンス変数となっている。その後の4つのインスタンス変数は gem から install した場合における、`editor_learner` が格納されているパスを受け渡すためのインスタンス変数である。一つずつの説明は以下の通り、

- `lib_location` はターミナル上で”gem environment gemdir”を入力した場合に出力されるパスを格納している。(自分のターミナル場で実行すると `/Users/souki/.rbenv/versions/2.4.0/lib/`
- `versions` は gem で install された `editor_learner` の version を受け取るためのパスを格納したインスタンス変数である。
- `latest_version` は `versions` で受け取った `editor_learner` の version の最新部分のパスを格納したインスタンス変数である。
- `inject` は実際にこれらのパスをつなぎ合わせてできる gem で install された `editor_learner`

が格納されているパスが格納されているインスタンス変数である。(自分の場合は/Users/souki/.rbenv/
1.1.2となる)

4.1.2 File の作成

全てのパスの準備が整ったら実際に作業する場所に必要なファイル (question.rb や answer.rb) などの作成が行われる。本研究のコードでは editor_learner/workshop がホームディレクトリになれば作成する。さらに、その中に random_check に必要なファイル (question.rb, answer.rb, random_h.rb) が作成される。random_h.rb は gem で install した場合は editor_learner の格納されている部分からコピーを行なっている。次に、sequential_check に必要なファイルを作成する。editor_learner/workshop に ruby_1_ruby6 がなければ作成し、その中に 1.rb, 3.rb と q.rb (問題をコピーするためのファイル) と sequential_h.rb が作成される。sequential_h.rb は random_h.rb と同じで gem から install した場合は editor_learner の格納されている部分からコピーを行なっている。このメソッドの大きな役割はファイル作成である。

4.2 ファイル削除処理 delete

sequential_check で終了した chapter をもう一度したい場合に一度ファイルを削除しなければいけないので、delete メソッドの大きな役割は sequential_check で終了したファイルの削除である。

```
desc 'delete [number~number]', 'delete the ruby_file choose number to delete file'

def delete(n, m)
  range = n..m
  range.each{|num|
    if File.exist?("#{@prac_dir}/ruby_#{num}") == true then
      system "rm -rf #{@prac_dir}/ruby_#{num}"
    end
  }
end
```

コード自体はいたってシンプルで引数を 2 つ受け取ることでその間の範囲の File を削除するようなコードとなっている。system の”rm -rf ファイル名”がファイルを削除するコマンドなのでそこで受け取った引数の範囲でファイルの削除を行っている。

4.3 random_check

random_check のコードは以下の通り,

```

desc 'random_check', 'random check your typing and edit skill.'

def random_check(*argv)
  random = rand(1..15)
  p random
  s = "#{random}.rb"
  puts "check starting ..."
  puts "type following commands on the terminal"
  puts "> emacs question.rb answer.rb"

  src_dir = File.expand_path('../..', __FILE__)
  # "Users/souki/editor_learner"
  if File.exist?("#{@inject}/random_check_question/#{s}") == true then
    FileUtils.cp("#{@inject}/random_check_question/#{s}",
      "#{@prac_dir}/question.rb")
  elsif
    FileUtils.cp(File.join(src_dir, "lib/random_check_question/#{s}"),
      "#{@prac_dir}/question.rb")
  end
  open_terminal

  start_time = Time.now
  loop do
    a = STDIN.gets.chomp
    if a == "check" && FileUtils.compare_file("#{@prac_dir}/question.rb",
      "#{@prac_dir}/answer.rb") == true then
      puts "It have been finished!"
      break
    elsif FileUtils.compare_file("#{@prac_dir}/question.rb",
      "#{@prac_dir}/answer.rb") != true then
      @inputdata = File.open("#{@prac_dir}/answer.rb").readlines
      @checkdata = File.open("#{@prac_dir}/question.rb").readlines
      diffs = Diff::LCS.diff("#{@inputdata}", "#{@checkdata}")
      diffs.each do |diff|
        p diff
      end
    end
  end
end

end_time = Time.now
time = end_time - start_time - 1

```

random_check の概要を簡単に説明すると 15 個ある Ruby のコードから 15 の乱数を取得し、選ばれた数字のファイルが問題としてコピーされて、それを answer.rb に入力することで正解していたら新しいターミナルが開かれてから終了までの時間を評価する仕組みとなっている。

上から解説を行うと、15 の random な乱数を取得、起動と同時にどのファイルがコピーされたか表示される。そして、src_dir でホームディレクトリ /editor_learner のパスが代入される。そして、gem で install した人と github から clone した場合によるファイルコピーのパスの違いを if で条件分岐。そして、15 の乱数のファイルが question.rb にコピーされる。コピーされた後に新しいターミナルが開かれ、時間計測が開始される。そして、check を前の画面に入力できるように gets を使った。初めに gets だけを使用した時改行が入ってしまいうまく入力できなかった。しかし、chomp を入れることで改行をなくすことに成功。しかし、argv と gets を両方入れることが不可能なことが判明した。そこで gets の前に STDIN を入れることで argv との併用が可能なのことがわかり、STDIN.gets.chomp と入力することでキーボードからの入力を受け取ることができた。そして、check が入力されてかつ FileUtils.compare でファイルの比較で正しければ時間計測を終了し、表示する。間違っていた場合はインスタンス定数である input と output に question.rb と answer.rb の中身が格納されて Diff::LCS の diff によって間違っている箇所だけを表示する。一連のコード解説は以上である。

4.4 sequential_check

sequential_check の場合はリファクタリングにあたりたくさんのインスタンス定数を作った。コードは以下の通り、

```

desc 'sequential_check [lesson_number] [1~3number] ',
'sequential check your typing skill and edit skill choose number'
def sequential_check(*argv, n, m)
  l = m.to_i - 1

  @seq_dir = "lib/sequential_check_question"
  q_rb = "ruby_#{n}/#{m}.rb"
  @seqnm_dir = File.join(@seq_dir, q_rb)
  @pracnm_dir = "#{ENV['HOME']}/editor_learner/workshop/ruby_#{n}/#{m}.rb"
  @seqnq_dir = "lib/sequential_check_question/ruby_#{n}/q.rb"
  @pracnq_dir = "#{ENV['HOME']}/editor_learner/workshop/ruby_#{n}/q.rb"
  @seqnl_dir = "lib/sequential_check_question/ruby_#{n}/#{l}.rb"
  @pracnl_dir = "#{ENV['HOME']}/editor_learner/workshop/ruby_#{n}/#{l}.rb"
  puts "check starting ..."
  puts "type following commands on the terminal"
  src_dir = File.expand_path(' ../../..', __FILE__)
  if File.exist?("#{@inject}/sequential_check_question/ruby_#{n}/#{m}.rb")
    == true then
      FileUtils.cp("#{@inject}/sequential_check_question/ruby_#{n}/#{m}.rb"
        , "#{@pracnq_dir}")
    elsif
      FileUtils.cp(File.join(src_dir, "#{@seqnm_dir}"), "#{@pracnq_dir}")
    end
  if l != 0 && FileUtils.compare_file("#{@pracnm_dir}", "#{@pracnq_dir}")
    != true
    FileUtils.compare_file("#{@pracnl_dir}", (File.join(src_dir,
      "#{@seqnl_dir}"))) == true
    FileUtils.cp("#{@pracnl_dir}", "#{@pracnm_dir}")
  end

  if FileUtils.compare_file(@pracnm_dir, @pracnq_dir) != true then
    system "osascript -e 'tell application \"Terminal\" to do script
      \"cd #{@prac_dir}/ruby_#{n} \" ' "
    loop do
      a = STDIN.gets.chomp
      if a == "check" && FileUtils.compare_file("#{@pracnm_dir}",
        "#{@pracnq_dir}") == true then
        puts "ruby_#{n}/#{m}.rb is done!"
        break
      end
    end
  end
end

```

```

    elsif FileUtils.compare_file("#{@pracnm_dir}", "#{@pracnq_dir}")
      != true then
        @inputdata = File.open("#{@pracnm_dir}").readlines
        @checkdata = File.open("#{@pracnq_dir}").readlines
        diffs = Diff::LCS.diff("#{@inputdata}", "#{@checkdata}")
        diffs.each do |diff|
          p diff
        end
      end
    end
  end
else
  p "ruby_#{n}/#{m}.rb is finished!"
end
end
end

```

4.4.1 インスタンス定数に格納されたパス

インスタンス定数に格納されているパスについての説明は上から順に以下の通り，

1. seq_dir は github で clone した人が問題をコピーするときに使うパスである．
2. seqnm_dir はその名の通り seq_dir に引数である n と m を代入したパスである．例として引数に 1 と 1 が代入された時は以下の通り，
 1. editor_learner/sequential_check_question/ruby_1/1.rb となる．
3. pracnm_dir は prac_dir に二つの引数 n と m を代入したものである．実際に作業するところのパスとして使用する．例として引数として 1 と 1 が代入された時は以下の通り，
 1. ホームディレクトリ/editor_learner/workshop/ruby_1/1.rb が格納される．
4. 同様に seq と prac の後についている文字はその後の ruby_(数字)/(数字).rb の数字に入る文字を後につけている．

4.4.2 動作部分

まず gem で install した場合と github で install した場合による違いを条件分岐によりパスを変えている．さらに 1.rb が終了していた場合 2.rb に 1.rb をコピーした状態から始まるように処理が行われている．その後は”check”が入力された時かつ FileUtils.compare で正解していれば終了．間違っていれば Diff::LCS で間違っている箇所を表示．もう一度修正し，”check”を入力，正解していれば終了．以上が一連のコードの解説である．

4.5 新しいターミナルを開く open_terminal

新しいターミナルを開くメソッドである．コードは以下の通りである．

```
def open_terminal
  pwd = Dir.pwd
  system "osascript -e 'tell application \"Terminal\" to do script \"cd #{@prac_dir}\"'"
end
```

新しく開かれたターミナルは prac_dir(editor_learner/workshop) のディレクトリからスタートするように設定されている． random_check では editor_learner/workshop でターミナルが開かれ， sequential_check では editor_learner/workshop/第 1 引数で入力されたファイルの場所が開かれるようになっている．

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。

	UI	プログラムの実行	タイピング以外の付加価値	タイピング文字
editor_learner	CUI	可能	editor操作	プログラミング言語
PTYPING	GUI	不可能	プログラミング言語が豊富	プログラミング言語
e-typing	GUI	不可能	資格取得の練習	ローマ字
寿司打	GUI	不可能	ランキング登録可能	ローマ字

図 5.1: 他のソフトとの比較.

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、

5.1 PTYPING

PTYPING は豊富なプログラム言語が入力可能である。しかし、コードを打つのではなく、コードに使われる `int` などよく使われる単語が 60 秒の間にどれだけ打てるかというソフトです。

5.2 e-typing

e-typing はインターネットで無料提供されているソフトである。ローマ字入力を基本として、単語、短文、長文の 3 部構成となっておりタイピングの資格取得の練習もできる。

5.3 寿司打

自分が一番利用したサイト、GUI ベースによりローマ字入力を基本とし、打てば打つほど秒数が伸びていきどれだけ入力できるかをランキング形式で表示される。

5.4 考察

これら全てのソフトを利用した結果、`editor_learner` はローマ字入力ができない点では他のソフトに遅れをとるが、実際にプログラムを書くようになってからコードを写経することで `{}` や `()` などといったローマ字入力ではあまり入力しないような記号の入力が非常に早くなった。さらに、`editor_learner` は現段階では Ruby の学習のみだが、引数を変えて元となるプログラムを作成することで全てのプログラム言語を学ぶことができる。さらに、実際にコードを入力することができるソフトはたくさんあるが、実行可能なものは少ない (Web で行うものが大半を占めているから。) 実際に西谷研究室で `editor_learner` で学習を行っていない学生と行った自分の `random_check` 平均秒数は前者は 200 秒程なのに対して、自分は 60 秒程である。これらの結果から `editor_learner` による学習により、Ruby 言語の学習にもなり、タイピング速度、正確性の向上、CUI 操作の適応による差が出たと考えた。

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見込める結果となった．

第7章 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．

付録 A プログラムのソースコード

```

# coding: utf-8
require 'fileutils'
require 'colorize'
require 'thor'
require "editor_learner/version"
require 'diff-lcs'
require "open3"

module EditorLearner
  class CLI < Thor

    def initialize(*args)
      super
      @prac_dir="#{ENV['HOME']}/editor_learner/workshop"
      @lib_location = Open3.capture3("gem environment gemdir")
      @versions = Open3.capture3("gem list editor_learner")
      p @latest_version = @versions[0].chomp.gsub(' (', '-').gsub(')', ',')
      @inject = File.join(@lib_location[0].chomp, "/gems/#{@latest_version}/lib")
      if File.exist?(@prac_dir) != true then
        FileUtils.mkdir_p(@prac_dir)
        FileUtils.touch("#{@prac_dir}/question.rb")
        FileUtils.touch("#{@prac_dir}/answer.rb")
        FileUtils.touch("#{@prac_dir}/random_h.rb")
        if File.exist?("#{@inject}/random_h.rb") == true then
          FileUtils.cp("#{@inject}/random_h.rb", "#{@prac_dir}/random_h.rb")
        elsif
          FileUtils.cp("#{ENV['HOME']}/editor_learner/lib/random_h.rb",
            "#{@prac_dir}/random_h.rb")
        end
      end
    end

    range = 1..6
    range_ruby = 1..3
    range.each do |num|
      if File.exist?("#{@prac_dir}/ruby_#{num}") != true then
        FileUtils.mkdir("#{@prac_dir}/ruby_#{num}")
        FileUtils.touch("#{@prac_dir}/ruby_#{num}/q.rb")
        FileUtils.touch("#{@prac_dir}/ruby_#{num}/sequential_h.rb")
      end
    end
  end
end

```

```

        if File.exist?("#{@inject}/sequential_h.rb") == true then
            FileUtils.cp("#{@inject}/sequential_h.rb", "#{@prac_dir}/
            ruby_#{num}/sequential_h.rb")
        else
            FileUtils.cp("#{ENV['HOME']}/editor_learner/lib/sequential_h.rb",
            "#{@prac_dir}/ruby_#{num}/sequential_h.rb")
        end
        range_ruby.each do |n|
            FileUtils.touch("#{@prac_dir}/ruby_#{num}/#{n}.rb")
        end
    end
end
end

desc 'delete [number~number]', 'delete the ruby_file choose number to
delete file'

def delete(n, m)
    range = n..m
    range.each{|num|
        if File.exist?("#{@prac_dir}/ruby_#{num}") == true then
            system "rm -rf #{@prac_dir}/ruby_#{num}"
        end
    }
end

desc 'sequential_check [lesson_number] [1~3number] ', 'sequential check
your typing skill and edit skill choose number'
def sequential_check(*argv, n, m)
    l = m.to_i - 1

    @seq_dir = "lib/sequential_check_question"
    q_rb = "ruby_#{n}/#{m}.rb"
    @seqnm_dir = File.join(@seq_dir, q_rb)
    @pracnm_dir = "#{ENV['HOME']}/editor_learner/workshop/ruby_#{n}/#{m}.rb"
    @seqnq_dir = "lib/sequential_check_question/ruby_#{n}/q.rb"
    @pracnq_dir = "#{ENV['HOME']}/editor_learner/workshop/ruby_#{n}/q.rb"
    @seqnl_dir = "lib/sequential_check_question/ruby_#{n}/#{l}.rb"
    @pracnl_dir = "#{ENV['HOME']}/editor_learner/workshop/ruby_#{n}/#{l}.rb"

```

```

puts "check starting ..."
puts "type following commands on the terminal"
src_dir = File.expand_path('../..', __FILE__)
if File.exist?("#{@inject}/sequential_check_question/ruby_#{n}/#{m}.rb")
  == true then
    FileUtils.cp("#{@inject}/sequential_check_question/ruby_#{n}/#{m}.rb",
      "#{@pracnq_dir}")
  elsif
    FileUtils.cp(File.join(src_dir, "#{@seqnm_dir}"), "#{@pracnq_dir}")
  end
if 1 != 0 && FileUtils.compare_file("#{@pracnm_dir}", "#{@pracnq_dir}")
  != true
    FileUtils.compare_file("#{@pracnl_dir}", (File.join(src_dir,
      "#{@seqnl_dir}"))) == true
    FileUtils.cp("#{@pracnl_dir}", "#{@pracnm_dir}")
  end

if FileUtils.compare_file(@pracnm_dir, @pracnq_dir) != true then
  system "osascript -e 'tell application \"Terminal\" to do script \
    \"cd #{@prac_dir}/ruby_#{n} \\"
loop do
  a = STDIN.gets.chomp
  if a == "check" && FileUtils.compare_file("#{@pracnm_dir}",
    "#{@pracnq_dir}") == true then
    puts "ruby_#{n}/#{m}.rb is done!"
    break
  elsif FileUtils.compare_file("#{@pracnm_dir}", "#{@pracnq_dir}")
    != true then
    @inputdata = File.open("#{@pracnm_dir}").readlines
    @checkdata = File.open("#{@pracnq_dir}").readlines
    diffs = Diff::LCS.diff("#{@inputdata}", "#{@checkdata}")
    diffs.each do |diff|
      p diff
    end
  end
end
else
  p "ruby_#{n}/#{m}.rb is finished!"
end
end
end

```

```

desc 'random_check', 'random check your typing and edit skill.'
def random_check(*argv)
  random = rand(1..15)
  p random
  s = "#{random}.rb"
  puts "check starting ..."
  puts "type following commands on the terminal"
  puts "> emacs question.rb answer.rb"

  src_dir = File.expand_path('../..', __FILE__) # "Users/souki/
  editor_learner"
  if File.exist?("#{@inject}/random_check_question/#{s}") == true then
    FileUtils.cp("#{@inject}/random_check_question/#{s}",
      "#{@prac_dir}/question.rb")
  else
    FileUtils.cp(File.join(src_dir, "lib/random_check_question/#{s}"),
      "#{@prac_dir}/question.rb")
  end
  open_terminal

  start_time = Time.now
  loop do
    a = STDIN.gets.chomp
    if a == "check" && FileUtils.compare_file("#{@prac_dir}/question.rb",
      "#{@prac_dir}/answer.rb") == true then
      puts "It have been finished!"
      break
    elsif FileUtils.compare_file("#{@prac_dir}/question.rb",
      "#{@prac_dir}/answer.rb") != true then
      @inputdata = File.open("#{@prac_dir}/answer.rb").readlines
      @checkdata = File.open("#{@prac_dir}/question.rb").readlines
      diffs = Diff::LCS.diff("#{@inputdata}", "#{@checkdata}")
      diffs.each do |diff|
        p diff
      end
    end
  end
end

end_time = Time.now
time = end_time - start_time - 1

```



```
no_commands do
  def open_terminal
    pwd = Dir.pwd
    system "osascript -e 'tell application \"Terminal\" to do script
      \"cd #{@prac_dir}\" \" ' "
  end
end
end
end
```

参考文献

- [1] 「達人プログラマー」, Andrew Hunt 著, 2016/10/20
- [2] <https://www.ruby-lang.org/ja/> 2017/2/8 アクセス.
- [3] <https://qiita.com/sumyapp/items/5ec58bf3567e557c24d7> 2017/2/8 アクセス.
- [4] <https://www.webl.io/content/キーバインド> 2017/2/8 アクセス.
- [5] <https://www.webl.io/content/CUI> 2017/2/8 アクセス.
- [6] <https://qiita.com/succi0303/items/32560103190436c9435b> 2017/2/8 アクセス
- [7] <https://docs.ruby-lang.org/ja/latest/library/open3.html> 2017/2/8 アクセス
- [8] https://qiita.com/io_fleming/items/14626a9cff44bc87e7db 2017/2/8 アクセス