touch typing 習得環境 shunkuntype

関西学院大学・理工学部・西谷滋人

2016 年 2 月 7 日

概要

Touch typing は programmer に必須のスキルである．裏をかえせば，プログラミングが苦手な学生ほど，このスキルが身についていないことが多い．より気にすべきは，ペアプログラミングにおいて熟練者がついた場合，運指がおかしい初心者の手の動きを不快と感じること．あらゆる作業において型が崩れていると親方が叱咤するのに通じる．所作の基本は，成功体験の集大成なのだから．touch typing は正にプログラマの「箸の上げ下ろし」なんだな．

さらに，運指がスムーズになると単語の綴りを指が覚えるようになるが，これはもう少し先の話．

さて，Touch typing の習得には，ピアノやギターの運指練習とおなじで単純な繰り返しが必要である．したがって飽きやすい．どのようにこの「単純な繰り返し練習」を「おもしろくするか」の一つの提案として，習得管理ツールを開発する．キーは「データの公開」である．

目次

1　【はじめに】

1.1　touch typing のコツ

touch typing の習得においてはスピードよりも運指が重要である．すなわち，スピードはそれほど速くなくても，運指を常に意識しておなじ指でおなじキーを押す癖をつけることが，初心者には必要である．これが後々のスピードアップにつながるだけでなく，

- タイプの正確性
- 視点固定による，疲労度の抑制

につながる．

1.2　非陳述記憶の性質

長期記憶には大きく分けて陳述記憶 (declarative memory) と非陳述記憶 (non declarative memory) がある．タイピングは自転車の乗り方と同じ，非陳述記憶の中の手続き記憶に分類され，同じ経験を反復することで形成される．いちど形成されると自動的に機能し，長期間保たれるという特徴がある [1]．したがって，タッチタイピングの習得には繰り返し練習が有効である．

1.3　繰り返しの工夫，個人の資質から仲間力へ

しかし，繰り返し練習は単調であるためにどうしても継続が難しい．その単調さを克服するためには，個人の資質として，賢明な学習者やマインドセットなどが要望される．しかし，そこから鍛えていたのでは．相当な時間がかかってしまう．より即効性のある手法として，継続を促すために少数のグループによる「あおり」を助長する工夫を，開発した shunkuntype では，取り入れている．

1.4 e-Learning から active learning へ

このような単純な知識の習得に対して，e-Learning がいくつも開発されている．ところが，単に e-Learning システムをやらせるだけでは，学習が継続しないという結果がでている．そこで，e-Learning の個別進捗に管理者がコメントをつけることによって，学習を継続させる方策が練られている．しかし，これによってもあまりいい結果は出ない．

これに対して，今回構築するシステムでは，参加者どうしがお互いの学習履歴，成果を閲覧することを可能にしている．これによりお互いの刺激となるように工夫している．すなわち，管理者による監視ではなく，参加者による「あおり」を学習継続の動機にすることを意図している．

これが active learning の基本となる．

2 【システムの概要】

shunkuntype は拡張子によって振る舞いを制御している．

2.1 個別練習操作

ユーザが touch typing の練習を行う環境を提供する．

```
bob% shunkuntype
Shunkuntype says 'Hello world'.
Usage: shunkuntype [options]
    -v, --version                   show program Version.
    -i, --init                      Initialize data files
    -c, --check                     Check speed
    -d, --drill [VAL]               one minute Drill [VAL]
    -h, --history                   view training History
    -p, --plot                      Plot personal data
    -s, --submit                    Submit data to dmz0
        --review [TAG]              Review training, TAGs=html or hike
```

詳しい操作法，振る舞いの様子は **ユーザーズマニュアル**（Shunkuntype_manual）にある．

2.2 あおり操作 (–review)

あおりのために用意された環境が–review である．ユーザが submit したデータを集めて，進捗状況の表，およびグラフを作成し，表示する．–review の引数として html あるいは hiki を指定し，'tmp.html ' あるいは'tmp.hiki' を作成．

2.2.1 review の出力例

以下は，hiki での出力 tmp.hiki を貼り付けた状態．work.png,speed.png が自動生成されるが，添付ファイルとして upload しておく必要がある．

- Shunkun typer 2016-01-28 08:45:24 +0900

グラフの詳しい解説は次節で行う．

表 1

	speed[sec]		training	
	init	current	total time[min]	step
bob	37.68	40.39	57.00	45.00
donkey	92.40	39.30	153.00	14.00
fujimoto	90.63	39.81	118.00	22.00
iwasa	61.01	56.36	360.00	20.00
kiyohara	45.79	44.52	23.00	5.00
sakaki	32.83	32.72	8.00	7.00
sumita	127.51	48.94	152.00	15.00
tomoyafujioka	101.12	43.66	456.00	34.00
yoshiokatatsuro	79.59	52.36	526.00	24.00

表 2

`{{attach_view(work.png)}}`	`{{attach_view(speed.png)}}`
練習時間 (work.png)	speed 結果 (speed.png)

3 【評価】

3.1 継続性

図はそれぞれの被験者の練習の様子をグラフで示している.

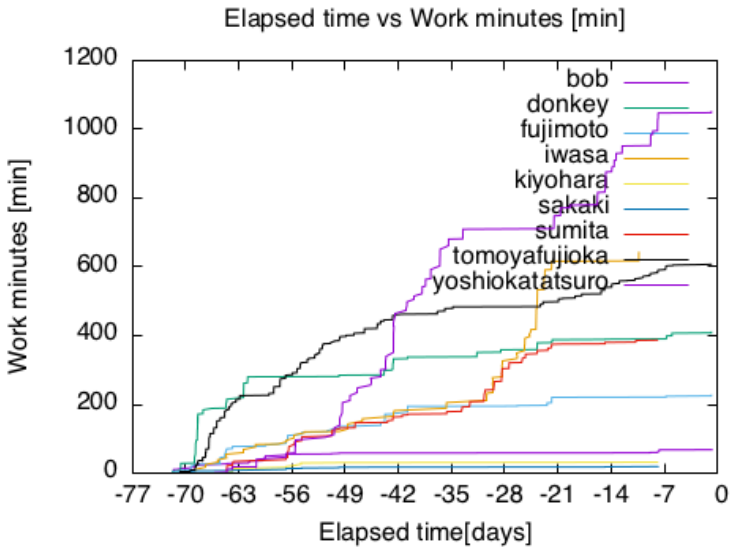

図 1　練習時間の継続状況を示すグラフ.

- 横軸に経過時間を日単位で，縦軸に練習時間を分単位でとっている.

- 経過時間は現在を０点にして，左方向にマイナスでとっている
- それぞれの被験者の総練習量はそれぞれのグラフの最高位置で示される．
- 被験者の立ち上がりがそれぞれ違うのは，実験に参加した時期が違うからである．
- 16 日経過時にもっとも進んでいるのは，fujioka で，その次に donkey が続いている．
- それ以外の被験者の練習量は総じて低い．

被験者の練習量の推移はそれぞれの被験者の取り組みに対する特徴を示している．すなわち

- 毎日５分程度を継続しておこなう
- 気がついたときにちょこちょこと継続する
- まとめてやるがあまり継続しない．

などである．

3.2 スピード改善

図はそれぞれの被験者のタイプ speed の様子をグラフで示している．

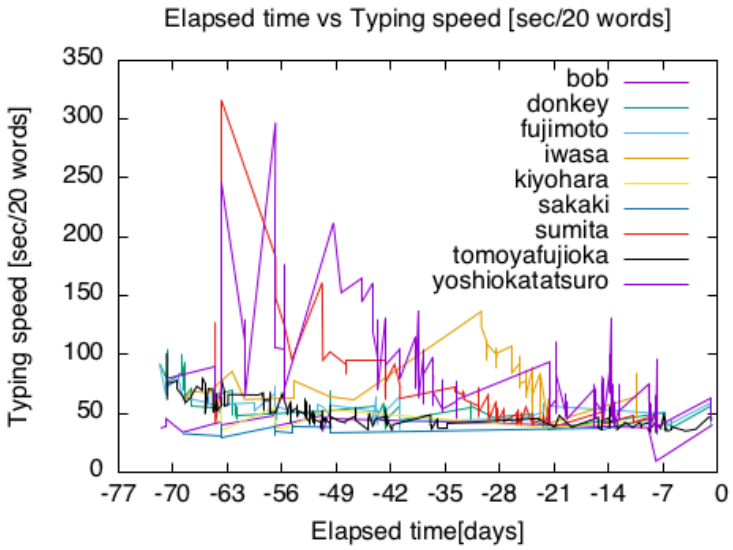

図 2　タイプスピードの変化を示すグラフ．

- 横軸のスケールは同じ．
- 縦軸には 20words を打つのに要する時間がプロットされている．
- speed には training で記録されているデータから計算したデータも含まれている．
- 途中で speed が 300 秒などまで極端に落ちた理由は，後述のハンカチ試験を導入したため．
- よりミクロに個人の成績を見れば，speed が徐々に上がっていることが読み取れる．
  - 例えば，fujioka の-70– -49days, sumita の-63– -21, iwasa の-35 – -21

3.3 ハンカチテスト

運指がうまくいっているかを自覚する方法としてハンカチテストが有効である．これは，手の甲の上にハンカチを置いて，手元のキーを見ずに打ち込んでいく方法である．これによって，

- キーを覚えているか
- ホームポジションへ帰っているか
- それぞれ決まった指でキーを押さえているか

のチェックができる．特に単なる speed 計測では 50 秒程度まで短縮している被験者も，ハンカチテストでは 300 秒もかかっている場合があった．

ただし，初心者はまったく key 配置を記憶できていないため，key 配置表を用意しておく必要がある．これはそのように speed プログラムを改修した (2015/12/02)．

4 【参加者の意識調査】

参加者に対してグループヒアリングを行った [2]．出てきた意見は次の通りである．

4.1 スキルの定着度合い

touch typing のスキルの定着について聞いたところ，

- ホームポジションに戻す癖がついた
- 運指がバラバラであったのが，５本指で打つようになった．
- 手元を見ていた
- 普段の生活での打ち方が変わった
- 画面と本を見るだけで打てるようになった
- パソコンでしゃべるときが速くなった
- フリップ入力よりは圧倒的に速い
  - フリップの速さは予測が効いているのかも
- 日本語は打てていたが，英語はダメだったのが改善した
- カッコとかの位置も覚えた
- 左手一本で全領域を打っていたのが全部の指で打てるようになった．
- 変な癖がなくなった．

4.2 「あおり」は効いたか？

speed の成績や練習時間を共有する「あおり」の効果について質問した．

- Y は S に負けたくないのでがんばった．
- みんなの speed をみて，いままで普通と思っていた入力速度に焦りが出た．自覚した．
- D は練習をやり始めたらずっとやるので，F1 もやらなと思った．
- 淡々としたんが好きで，おもしろい．レベル上げに通じる．
- 図で出るとちょっと意識する
- F2 は煽る側で優越感を感じて...

4.3 習慣になったか？

練習が習慣として定着したかと問うてみた．

- つかなんだ
- クリアしたら練習しない
- time 計測の結果を見てあせった
- パソコン開いたらまずやるという習慣がついた

4.4 不足している機能

不足している機能や，不満な点を指摘してもらった．

- ハンカチはよくない．余分な感覚があるから．
  - 黒塗りのキーボードカバーはいいかも

- speed に普通の単語をもっと増やす
  - 普通の単語は work とか table は変数によく使われるのでいいかも.
- 日本語入力の練習もできるように.
  - 将来プログラマーになるとは限らないので，練習の動機があがる.
- speed の単語には，” [.()[]=_”なども入れるべき
- ノルマを課すようにしては.
  - 図に理想線とか，目標線とか，ガイドラインをいれては？
- speed をタイムアップ方式にしてもいいかも．1 分試験
- レース完走方式にしてもいいかも.
- レベル上げ．装備をつけるなんかも...

5 【まとめ】

- 時間はかかったが参加者全員が 20 words/45secs をクリアした.
  - ただし，「クリアできなんだら卒研指導しない」というムチが働いた可能性は大きい.
- shunkuntype はその後 gem による command line interface アプリケーションとして完成.
- web 化が必要かもしれない.

その場合には,

- processing による GUI
- 登録グループによる競争の実現
  - 内部の煽りだけでなく，仮想敵を作って「くまさんチーム対うさぎさんチーム」で練習をあおる.
- db への data 移行
  - ssh より安全なデータ収集
  - 平文テキスト修正によるチートをなくす

などを考慮して，Rails を用いるのか sinatora などでベタで作るのかの検討が必要.

6 【付録】

A **マニュアル**（Shunkuntype_manual）
B gem **化メモ**（TouchTyping_shunkuntype_gemizing）
C **初期版のコード解説**（TouchTyping_Coding）

7 【参考資料】

1. 「陳述記憶・非陳述記憶」, 鈴木 麻希, 藤井 俊勝, 脳科学辞典, https://bsd.neuroinf.jp/wiki/(https://bsd.neuroinf.jp/wiki/) 陳述記憶・非陳述記憶 (2016/1/28 アクセス).
2. グループヒアリングメモ{{attach_anchor(Shunkuntype_GroupHearing_20160128.pdf)}}

付録 A　マニュアル

A.1　【概要】

touch typing 習得をめざした command line interface アプリ.

A.2　【特徴】

- 軽い.
- こまめな練習が可能な一分間練習.
- 練習履歴の report による仲間内の「あおり」をおこなう工夫.

A.3　【操作法】

```
bob% shunkuntype
Shunkuntype says 'Hello world'.
Usage: shunkuntype [options]
    -v, --version                    show program Version.
    -i, --init                       Initialize data files
    -c, --check                      Check speed
    -d, --drill [VAL]                one minute Drill [VAL]
    -h, --history                    view training History
    -p, --plot                       Plot personal data
    -s, --submit                     Submit data to dmz0
        --review [TAG]               Review training, TAGs=html or hike
```

A.4　【具体的な手順】

A.4.1　インストール

基本的な操作は全て terminal から行う.

■作業 directory の mk

```
mkdir new_shunkuntype
```

■proxy の登録と server の追加　関学内では gem install に proxy を通さないといけない.

```
for tcsh
bob% setenv HTTP_PROXY http://proxy.ksc.kwansei.ac.jp:8080
bob% setenv HTTPS_PROXY http://proxy.ksc.kwansei.ac.jp:8080
for bash, zsh
bob% export HTTP_PROXY=http://proxy.ksc.kwansei.ac.jp:8080
bob% export  HTTPS_PROXY=http://proxy.ksc.kwansei.ac.jp:8080
```

gem 標準の server には登録してないので，local server を登録する必要がある.

```
bob% gem source -a 'http://nishitani0.kwansei.ac.jp/~bob/nishitani0/gems/'
http://nishitani0.kwansei.ac.jp/~bob/nishitani0/gems/ added to sources
bob% gem search shunkuntype -r
```

■install　gem が install する directory の owner の権限によっては sudo が必要かもしれない.

```
bob% sudo gem install shunkuntype
```

A.4.2 **立ち上げ** (-i)

練習，スピードデータを作成しておく必要がある．

```
bob% shunkuntype --init
```

過去のデータを移行は別に後述.

A.4.3 **スピードチェック** (-c)

```
bob% shunkuntype -c
Shunkuntype says 'Hello world'.
./shunkuntype_speed_data.txt opened succcessfully
require next help directory temporary chomp delete catenate puts kill move working scan line backgroun

20 words should be cleared.
Type return-key to start.
 q \ w \ e \ r t \ y u \ i \ o \ p
  a \ s \ d \ f g \ h j \ k \ l \ ; enter
sh z \ x \ c \ v b \ n m \ , \ . \  shift
1
require
require
```

で 20 文字を何秒で打てるかをチェックできる.

A.4.4 one minute drill(-d)

```
bob% shunkuntype -d 5
Shunkuntype says 'Hello world'.
 q \ w \ e \ r t \ y u \ i \ o \ p
  a \ s \ d \ f g \ h j \ k \ l \ ; enter
sh z \ x \ c \ v b \ n m \ , \ . \  shift
deed did feed freed red deer
duke did free red deeded dike
jeered duke fed jeered jeerer

Repeat above sentences. Type return-key to start.
deed did feed freed red deer
```

-d の後の引数で指定した No の drill を実行.

A.4.5 report **の** submit(-s)

```
bob% shunkuntype -s
Shunkuntype says 'Hello world'.
"bob"
"bob"
shunkuntype_training_data.txt                 100% 3618     3.5KB/s   00:00
shunkuntype_speed_data.txt                    100%  862     0.8KB/s   00:00
```

nishitani0 のアカウントが必要．login でうまくいかない時には西谷に聞け.

A.4.6 **過去のデータを移行** (-i)

gem 化する前の古い shunkuntype で練習していた人は，データを移行する必要がある．

```
bob% cp ../shunkuntype/*_data.txt .
bob% shunkuntype --init
Shunkuntype says 'Hello world'.
./shunkuntype_speed_data.txt exits.
./shunkuntype_training_data.txt exits.
```

A.4.7 **練習履歴の表示** (-h)

練習履歴の表示は-h でされる．

```
bob% shunkuntype --history
Shunkuntype says 'Hello world'.
You've finished Basic drills of...
hour | contents              | step
   0 | frdejuki      1,2,3,4 | 1,2,3,4,
   1 | tgyh        5,6,7,8,9 | 5,6,7,8,9,
   2 | vbc    10,11,12,13,14 | 10,11,12,13,14,
   3 | mn,    15,16,17,18,19 | 15,16,17,18,19,
   4 | consol 20,21,22,23,24 | 20,
   5 | swx    25,26,27,28,29 |
   6 | lo.    30,31,32,33,34 | 32,
   7 | aqz    35,36,37,38,39 |
   8 | ;p  40,41,42,43,44,45 |
   9 | consol 46,47,48,49,50 | 47,48,49,
To continue one minute training: shunkuntype -d 21.
```

A.4.8 **練習履歴の表示** (-p)

個人の連取履歴のグラフを plot．gnuplot がないと error となる．

付録 B　gem 化メモ

B.1　目的

shunkuntype の配布を容易にするため gem を使って開発をすすめるためのメモ.

B.2　経緯

TouchTyping_Coding(TouchTyping_Coding) にあるとおり，command lin interface アプリとして shunkuntype は動作していた．しかし，幾つかの問題点があった.

1. 開発途中で user による検証を行っていたので，最新版の install に手間がかかる
2. install 時に練習履歴などの data file をいちいち移行しないといけない.
3. Rakefile.rb から動くようにしていたため,
   （a）各々のプログラムを起動
   （b）code files が分散
   （c）directory 構造が定まっていない

などである.

そこで gem 標準での開発に移行した．これによって

1. 標準的な gem install で最新版を常に維持可能
2. data file と実行ファイルが別の場所にあるので，いちいち移行する必要がない
3. directory 構造や optparse による標準的な記述が可能

になった．また,

- class 化

が強制されるため code の隠蔽や yard ドキュメントでの参照がしやすくなった.

ここで一つ懸念事項であった教材 data files の公開の是非は,

- local server を立てて，そこから down load

することで回避した.

その他の設計思想は TouchTyping_Coding(TouchTyping_Coding) を踏襲している.

B.3　gem 開発環境の構築

{{attach_anchor_string(「パーフェクト Ruby」15 章,PerfectRuby_c15_gems.pdf)}}を参照して作成していく.「パーフェクト Ruby」では test 駆動になっていなかったが，ある程度までは test 駆動で進めたほうがよさそう.

- test に使い方や仕様を盛り込む
- CVM モデルから外れた場合
  - DB を使うのが面倒なので，txt とする．その置き場は local にする.
- 追記：test 駆動では，すでに開発が終了して，test が十分なされているコードの移植では無駄が多い. たんにクラス化して動作を確認する方が早い.

B.3.1　bundle 他の記述

```
bob% bundle gem shunkuntype --test=minitest
```

で必要なファイルが自動生成される．このあと，minitest を unit-test に変更する必要がある．詳しくは, RubyGems_development(RubyGems_development) にある.

B.4 起動

rake から yard が動かせるように.

```
<   spec.add_development_dependency "yard", "~> 0.8"
```

を shunkuntype.gemspec に追加したあと,

```
bob% bundle install
bob% bundle exec rake yard
Files:           2
Modules:         1 (    1 undocumented)
Classes:         0 (    0 undocumented)
Constants:       1 (    1 undocumented)
Methods:         0 (    0 undocumented)
 0.00% documented
```

bundle install しないと，この diretory で bundle が yard を使えない.

B.4.1 exe

exe がないので自動生成．conflict が起こっている時には'n' で飛ばした.

```
bob% bundle gem shunkuntype -b -t
Creating gem 'shunkuntype'...
Code of conduct enabled in config
MIT License enabled in config
   identical  shunkuntype/Gemfile
   identical  shunkuntype/.gitignore
   identical  shunkuntype/lib/shunkuntype.rb
   identical  shunkuntype/lib/shunkuntype/version.rb
    conflict  shunkuntype/shunkuntype.gemspec
Overwrite /Users/bob/Ruby/shunkuntype/shunkuntype.gemspec? (enter "h" for help) [Ynaqdh] n
        skip  shunkuntype/shunkuntype.gemspec
      create  shunkuntype/Rakefile
    conflict  shunkuntype/README.md
Overwrite /Users/bob/Ruby/shunkuntype/README.md? (enter "h" for help) [Ynaqdh] n
        skip  shunkuntype/README.md
   identical  shunkuntype/bin/console
   identical  shunkuntype/bin/setup
   identical  shunkuntype/CODE_OF_CONDUCT.md
   identical  shunkuntype/LICENSE.txt
   identical  shunkuntype/.travis.yml
      create  shunkuntype/.rspec
      create  shunkuntype/spec/spec_helper.rb
      create  shunkuntype/spec/shunkuntype_spec.rb
      create  shunkuntype/exe/shunkuntype
Initializing git repo in /Users/bob/Ruby/shunkuntype
```

exe/shunkuntype は

```
#!/usr/bin/env ruby

require "shunkuntype"
```

```
    Shunkuntype::Command.run(ARGV)
```
となっている．shunkuntype.rb の class Command に以下の記述を追加.
```
        def self.run(argv=[])
            print "Hello world.\n"
    #      new(argv).execute
        end

        def initialize(argv=[])
          @argv = argv
        end
```
すると
```
    bob% bundle exec exe/shunkuntype
    Hello world.
```
で動いていることが確認できる．あとは，これを普通に改良していくだけ.

B.4.2 build のための改良

```
    bob% bundle exec rake build
    rake aborted!
    WARNING:  See http://guides.rubygems.org/specification-reference/ for help
    ERROR:  While executing gem ... (Gem::InvalidSpecificationException)
        ["Rakefile.rb"] are not files
```
なんで,
```
    ln -s Rakefile Rakefile.rb
```
次に,
```
    bob% bundle exec rake install:local
    rake aborted!
    WARNING:  See http://guides.rubygems.org/specification-reference/ for help
    ERROR:  While executing gem ... (Gem::InvalidSpecificationException)
        "FIXME" or "TODO" is not a description

    rake aborted!
    WARNING:  See http://guides.rubygems.org/specification-reference/ for help
    ERROR:  While executing gem ... (Gem::InvalidSpecificationException)
        missing value for attribute summary
```
なんで,
```
    <   spec.summary       = %q{type training.}
    <   spec.description   = %q{type training for shunkun.}
    ---
    >   spec.summary       = %q{TODO: Write a short summary, because Rubygems requires one.}
    >   spec.description   = %q{TODO: Write a longer description or delete this line.}
```
とちゃんと書かんと.

B.5 gem server の起動と check

公開できない可能性のあるアプリの開発のため，非公開のサーバを立てた場合の動作検証を進めておく.

今は,http://nishitani0.kwansei.ac.jp/~bob/nishitani0/gems/(http://nishitani0.kwansei.ac.jp/~bob/nishitan
に置いている．この下の gems に作成した shunkuntype-0.1.0.gem を置くと gem search で見つかる.
```
    bob% sudo gem install builder
```

```
Password:
Fetching: builder-3.2.2.gem (100%)
Successfully installed builder-3.2.2
Parsing documentation for builder-3.2.2
Installing ri documentation for builder-3.2.2
1 gem installed
bob% gem generate_index
Generating Marshal quick index gemspecs for 8 gems
........
Complete
Generated Marshal quick index gemspecs: 0.006s
Generating specs index
Generated specs index: 0.000s
Generating latest specs index
Generated latest specs index: 0.000s
Generating prerelease specs index
Generated prerelease specs index: 0.000s
Compressing indicies
Compressed indicies: 0.001s

bob% ls -r *
specs.4.8.gz            prerelease_specs.4.8.gz  latest_specs.4.8.gz
specs.4.8               prerelease_specs.4.8     latest_specs.4.8

quick:
Marshal.4.8/

gems:
shunkuntype-0.1.0.gem

bob% setenv HTTP_PROXY http://proxy.kwansei.ac.jp:8080
bob% setenv HTTP_PROXY http://proxy.ksc.kwansei.ac.jp:8080
bob% gem source -a 'http://nishitani0.kwansei.ac.jp/~bob/nishitani0/gems/'
http://nishitani0.kwansei.ac.jp/~bob/nishitani0/gems/ added to sources
bob% gem search shunkuntype -r

*** REMOTE GEMS ***

shunkuntype (0.1.0)
```

と無事見えた.

B.5.1 install

```
bob% gem install shunkuntype
Successfully installed shunkuntype-0.1.0
Parsing documentation for shunkuntype-0.1.0
Done installing documentation for shunkuntype after 0 seconds
WARNING:  Unable to pull data from 'http://localhost:8080': bad response Gateway Timeout 504 (http://l
1 gem installed
```

```
[bobsMacBookAir:~/Ruby/shunkuntype] bob% shunkuntype
shunkuntype: Command not found.
```
で，がくっときたが，
```
bob% ls /usr/local/lib/ruby/gems/2.2.0/gems/shunkuntype-0.1.0/
CODE_OF_CONDUCT.md   LICENSE.txt          Rakefile             bin/                 lib/
Gemfile              README.md            Rakefile.rb          exe/                 shunkuntype.gemspe
bob% ls /usr/local/lib/ruby/gems/2.2.0/bin
shunkuntype*
bob% /usr/local/lib/ruby/gems/2.2.0/bin/shunkuntype
Hello world.
```
無事完動．あとは path を書いとくのがいいのかな... /usr/local/bin にいれるというのも一つかも．

■追補 install し直すと/usr/local/bin に追加された．これは，
```
gem update systems
```
で gem の version が 2.4.5-¿2.5.1 に上がったため？．なぜなら，正しくは，
```
bob% gem environment
RubyGems Environment:
  - RUBYGEMS VERSION: 2.5.1
  - RUBY VERSION: 2.2.2 (2015-04-13 patchlevel 95) [x86_64-darwin13]
  - INSTALLATION DIRECTORY: /usr/local/lib/ruby/gems/2.2.0
  - USER INSTALLATION DIRECTORY: /Users/bob/.gem/ruby/2.2.0
  - RUBY EXECUTABLE: /usr/local/opt/ruby/bin/ruby
  - EXECUTABLE DIRECTORY: /usr/local/bin
  - SPEC CACHE DIRECTORY: /Users/bob/.gem/specs
  - SYSTEM CONFIGURATION DIRECTORY: /usr/local/Cellar/ruby/2.2.2/etc
  - RUBYGEMS PLATFORMS:
    - ruby
    - x86_64-darwin-13
  - GEM PATHS:
     - /usr/local/lib/ruby/gems/2.2.0
     - /Users/bob/.gem/ruby/2.2.0
     - /usr/local/Cellar/ruby/2.2.2/lib/ruby/gems/2.2.0
  - GEM CONFIGURATION:
     - :update_sources => true
     - :verbose => true
     - :backtrace => false
     - :bulk_threshold => 1000
     - :sources => ["https://rubygems.org/", "http://nishitani0.kwansei.ac.jp/~bob/nishitani0/gems/"]
  - REMOTE SOURCES:
     - https://rubygems.org/
     - http://nishitani0.kwansei.ac.jp/~bob/nishitani0/gems/
```
の EXECUTABLE DIRECTORY に install される．source による PATH の update を忘れずに．

B.6 shunkuntype の移植

B.6.1 shunkuntype.gemspec の files への登録

新しく code files を足していく．この時，gem build でちゃんと認識するには，spec.files に必要なファイルを足していく必要がある．

bundle gem で環境を作った場合，この操作は

```
    spec.files         = `git ls-files -z`.split("\x0").reject { |f| f.match(%r{^(test|spec|features)/})
```
により代行される．従って，git での操作
```
    git add -A
    git commit
```
などが必要.

B.6.2 directory の参照

shunkuntype をどこに install あるいは，どこで起動されるかにかかわらず data などが適切に参照されるようにするために,
```
      data_dir=File.expand_path('../../../lib/data', __FILE__)
      file_name="#{data_dir}/STEP-47.txt"
```
などとしている．ここでの__FILE__は exe/shunkuntype が起点.

B.6.3 optparse

perfect ruby では subcommand を扱うために todo/options.rb に二重構造の parse を使う例を示している．shunkuntype では一重でいいので，shunkuntype.rb に以下の通り記述.
```
       def execute
         @argv << '--help' if @argv.size==0
         command_parser = OptionParser.new do |opt|
           opt.on('-v', '--version','Show program version.') { |v|
             opt.version = Shunkuntype::VERSION
             puts opt.ver
           }
           opt.on('-i', '--init','initialize data files.') {|v| InitDataFiles.new }
           opt.on('-s', '--speed','speed check.') {|v| SpeedCheck.new }
           opt.on('-m', '--minute [VAL]','minute training of file Integer.', Integer) {|v| Training.new(v
           opt.on('-h', '--history','view training history.') {|v| FinishCheck.new }
           opt.on('-r', '--report','submit data to dmz0') { |v| report_submit()}
         end
         command_parser.parse!(@argv)
         exit
       end
```
ここで [VAL] は「Integer の引数をとってもいい（とらなくてもいい)」ことを表している．この場合，Training.new(v) の v には nil がはいって引き渡されるため，initialize(val=47) として default を定義できなかった．そこで,
```
     val ||= 47
```
としている.

B.6.4 テスト環境の手順と注意

テスト環境での検証を進めるときには，いちど build したものを install する必要がある．server 経由で install すると version が変わらないと新たに install を行わない．build ではなく直接 install すると最新版が version の違いにかかわらず install されて動作検証が可能.
```
    bundle exec rake install
```
では固まることがある．これは，ネット経由で通常の gem サーバーの情報を取ろうとしているため.
```
    bundle exec rake install:local
```
で local のみに install される．install したのを直接叩いてテストすると便利.

B.7 未対応

- debug と pry のために readline version を
- reference を local で参照できるように.

B.8 開発履歴

```
bob% git log
commit f59be87486ae39678e4530e80378c958170d4793
Author: Shigeto R. Nishitani <shigeto_nishitani@me.com>
Date:   Thu Jan 28 09:39:54 2016 +0900

    今んところの凍結版.

commit 40e2b8e1eeb7b1a9eedd69a5f206265075e382fb
Author: Shigeto R. Nishitani <shigeto_nishitani@me.com>
Date:   Wed Jan 27 17:33:03 2016 +0900

    参加者の training, speed data を集計して表示が可能に.

commit d028555812e1af01322d83a626feffe10a357741
Author: Shigeto R. Nishitani <shigeto_nishitani@me.com>
Date:   Wed Jan 20 16:28:44 2016 +0900

    0.5.0 user 側での操作は view 以外を全て実装.

commit ff3a4fe7ef2d09302d2f8413507fac3f669f528e
Author: Shigeto R. Nishitani <shigeto_nishitani@me.com>
Date:   Wed Jan 20 11:55:12 2016 +0900

    for avoiding the error "["training_data.txt"] are not files".

commit 18ff1793a22fddaaafa8e9b2bb592c66f6274140
Author: Shigeto R. Nishitani <shigeto_nishitani@me.com>
Date:   Wed Jan 20 10:56:00 2016 +0900

    version 表示と speed check を options に入れた.

commit 3728051eda7f62752526865ccf3792843b7ed706
Author: Shigeto R. Nishitani <shigeto_nishitani@me.com>
Date:   Tue Jan 19 13:44:41 2016 +0900

    えっと, donkey への demo のための version.

commit 3a8117135228f766d1ef462dea7aecde933f63da
Author: Shigeto R. Nishitani <shigeto_nishitani@me.com>
Date:   Tue Jan 19 12:56:33 2016 +0900
```

one minite training を組み込んだ version.

```
commit 46f8f97193291bf6da298d74aa08715f079bf3ca
Author: Shigeto R. Nishitani <shigeto_nishitani@me.com>
Date:   Tue Jan 19 12:09:31 2016 +0900
```

もっとも最初の shunkun 動作検証 version. speed と word list のみ.

付録 C　初期版のコード解説

C.1　【plot ライブラリの導入】

web のためのデータ加工において次のような変更を行った.

- gnuplot ライブラリの導入
- PlotData クラスの生成

これにより,

- 動作の目的別によるファイルの分離
- 中間ファイルの省略

などが達成され, 記述の簡略化が相当進んだ．詳しくは `TouchTyping_Coding_gnuplot`, `TouchTyping_Coding_PlotData` を参照.

C.2　【開発途中での使用の問題点】

ユーザーの使用，特に情報の収集をしながら，コードの開発を行った．そのため以下のような不便さが出てきた.

1. どの version を使用しているかで挙動が少しずつ変わる
2. どれが最新版かわからない
3. 最新版の更新を tgz でおこなった
4. 個人データを残すため cp で必要な書類を上書きする必要がある.

このあたりこそ Chef の出番なのに，donkey に不幸な事故が起こって...

C.3　【code 記述の基本原則】

動作はすべて rake で指示するようにしている．これにより,

- 操作法の一括化，標準化
- ruby code の分離
  - 環境に依存しやすいところは Rakefile.rb で
  - 変わらないところを*.rb に

という基本原則にのっとっている.

しかし，個別の*.rb は system call から呼び出している．この代案としては,

- require して直接 proc を呼び出す
- class 化して隠蔽する

が考えられる．class 化による隠蔽によって細かいところを気にしなくても良くなる．しかし，隠蔽によってどのように code が挙動するかがわからなくなり，code をいじることが難しくなる.

この code 整理は ruby の基本から逸脱している．これは,「見える化」によりどこに何があるかを明示する整理法に由来している．今後はこの基本原則を CoC 化する方策を考える必要がある.

C.4　【directory 構造と動作の概要】

C.4.1　ユーザ環境

ユーザ環境の directory 構造は以下の通りである.

```
ShunkunType
├── GerardStrong_data
│   ├── STEP-01-home.txt
│   ├── ・・・
```

```
|   └── contents.txt
├── Rakefile.rb
├── data
|   ├── STEP-1.txt
|   ├── ・・・
|   └── word.list
├── finished_check.rb
├── speed.rb
├── speed_data.txt
├── training.rb
└── viewer.rb
```

Rakefile.rb に以下の動作が用意されており，各 file を起動するようにしている．

```
rake size    # start size training: Gerard Strong data
rake speed   # speed check
rake submit  # submit data to dmz0
rake time    # start time training: one minite drills
rake view    # view training history: display the finished step table and the graphs
```

動作の詳細は**ユーザマニュアル**（TouchTyping）にある．

C.4.2 Web 集計環境

Web 集計環境の directory 構造は以下の通りである．

```
Web/
.
├── Rakefile.rb
├── ShunkunTyper2015
├── mem_data
|   ├── bob_speed_data.txt
|   ├── bob_training_data.txt
...
├── mk_shunkunTyper.rb
├── old_rbs
|   ├── #Rakefile2.rb#
...
├── plot_data.rb
└── res.png
```

Rakefile.rb に動作がすべて用意されている．

```
rake edit        # edit related hiki webs
rake speed_all   # make speed graphs for all members
rake speed_view  # make speed graphs for a specific member
rake table       # make web table
rake work_all    # view whole working time including speed_test for all members
rake work_view   # view whole working time including speed_test for a specific member
```

■table　table によって mk_shunkunTyper.rb が起動され，

- 個人のトレーニング履歴データを mem_data にダウンロード
- hiki 形式で table を作成，
- ShunkunTyper2015 に追加

する．

■speed, work, view, all　speed および training のデータをグラフ化する．rake コマンドとして，

```
rake work_all
```

とした場合には，speed および training データの作業時間を総計する．view だけの場合は特定の被験者に対するグラフを作成．例えば，

```
rake speed_view donkey
```

とした場合には，donkey に関する speed ダータを表示.

典型的な Rakefile.rb でのデータ処理は以下の通り．

```
desc "view whole working time including speed_test."
task :work_all => [:mk_mem_names] do
  all_data= $mem_names.inject([]){|all,name| all << work_view(name) }
  text="Work minutes [min]"
  plot(all_data,text)
  plot(all_data,textopts={:png=>true})
  exit
end

def work_view(name)
  p name0 = "#{$source_dir}/#{name}_training_data.txt"
  p name1 = "#{$source_dir}/#{name}_speed_data.txt"
  data0 = PlotData.new(name0,0,3,name)
  data0.add_general_data(name1, 0, 2)
#  start=Time.parse(data0.data[0][0])
  start=Time.parse(Time.now.to_s)
  x_func = proc{|x| ((Time.parse(x)-start)/3600/24) }
  y_func = proc{|x| x.to_f/60.0 }
  data0.mk_plot_data(x_func,y_func)
  data0.sort
  data0.sum_data
  return data0
end
```

- all_data に work_view を追加していく．
  - work_view では
  - file 名を生成
  - data0 を name0 から作成し，name1 からデータを追加.
  - x_func, y_func でデータを修正
  - x 軸を sort
  - y 軸は積分値に変換
- plot に all_data を渡す.

■edit　関連する hiki ページの編集環境を open.

C.5　【data の収集方法】

練習記録データをどのように収集するかが最初の課題であった．一般的には，

- 提出用の web を設けてそこへ提出

という方法が正当であろう．そのためのサイト構築には，

- hiki
- rails

- cgi

が考えられる．しかし，user が

- いちいちあげる手数が面倒

と思ってしまうと情報収集がスムースに進行しない．data 収集後の操作は，

- table 作成
- graph 作成
- web での表示

であるが，これらを自動化するために，hiki の plugin で実装するのも少し手間がかかる．

そこで user 自身が scp により web の特定 directory に submit するという方法をとることにした．理由は以下の通り．

- terminal での type 練習をしているので，CGI での操作が自然．
- ssh のアカウントを被験者全員が保有している
- scp 先の directory の permission を world writable にしてもだれが upload するかの管理はできている．
- scp で手元で data を取れた方が graph や table の改良が容易．

もちろん ruby による自動化は用意されていて，

```
rake submit
```

と user は打ち込むだけにしている．

```
desc "submit data to dmz0."
task :submit do
  dir ='/Users/bob/Sites/nishitani0/ShunkunTyper/mem_data'
  system "scp ./training_data.txt #{true_name}@dmz0:#{dir}/#{true_name}_training_data.txt"
  system "scp ./speed_data.txt #{true_name}@dmz0:#{dir}/#{true_name}_speed_data.txt"
end
```