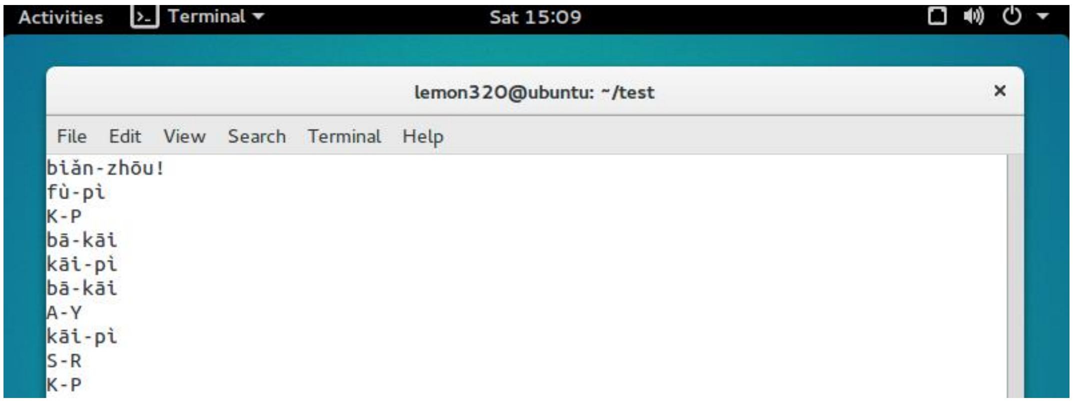
Course: CS562
Name: Yanshen Wang
Date: 03/12/16
Final Report

In this term of CS 562 class, I learned a new kind of testing language, which is called TSTL (Template Scripting Testing Language), and it is mainly used for support programmers to writing a test harness and find bugs from other coding systems. For this report, I would like to use the TSTL tool to test a python library which is use for translating the pronunciation of Chinese characters into Latin alphabet. With the help of this python library, after users input some Chinese characters, they can easily get some results including the letters of pinyin, the tone marks and the initial of each characters. As we know, in currently the technique of Chinese characters' conversion is not very mature, in other words, users will always meet many different problems when they utilize this kind of applications. Therefore, I am confident that at the end of this term, I can utilize the TSTL language to find out some bugs from this python library.

1. My accomplishments

Actually, the whole progress of this testing project is based on my original proposal, and I implement them in the following steps:

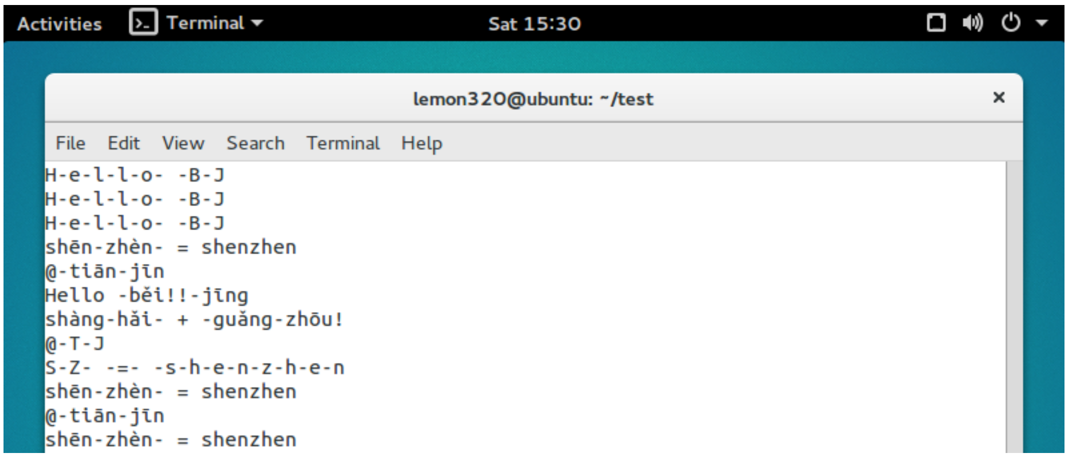
a. First of all, in the syntax of Chinese pronunciation, some characters can have two or more pronunciations. However, after I put those kind of Chinese characters into this library, I found that this program cannot consider this situation, which means each Chinese character only has one pronunciation.



```
biǎn-zhōu!
fù-pì
K-P
bā-kāi
kāi-pì
bā-kāi
A-Y
kāi-pì
S-R
K-P
```

Pic 1. Syntax test

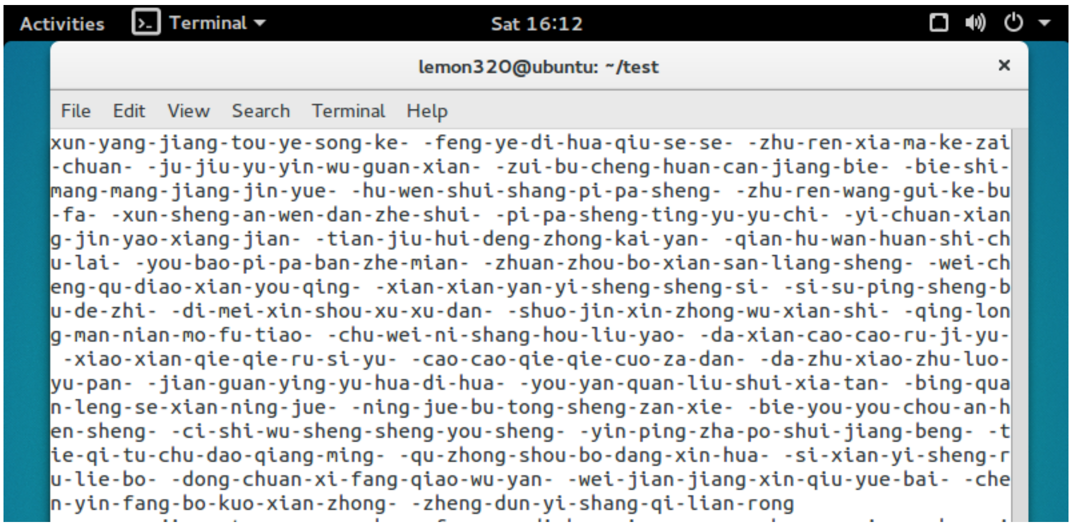
b. In addition, I will try to input some mixed strings into the function to find out if this library will meet some bugs. Actually, this library did really well in this situation since it can smartly recognize which strings are Chinese characters and which strings are not. For the Chinese characters, it can convert them into Latin alphabet; while for the other strings, it just keeps their original type.



```
H-e-l-l-o- -B-J
H-e-l-l-o- -B-J
H-e-l-l-o- -B-J
shēn-zhèn- = shenzhen
@-tiān-jīn
Hello -běi!!-jīng
shàng-hǎi- + -guǎng-zhōu!
@-T-J
S-Z- -=- -s-h-e-n-z-h-e-n
shēn-zhèn- = shenzhen
@-tiān-jīn
shēn-zhèn- = shenzhen
```

Pic 2. Mixed strings

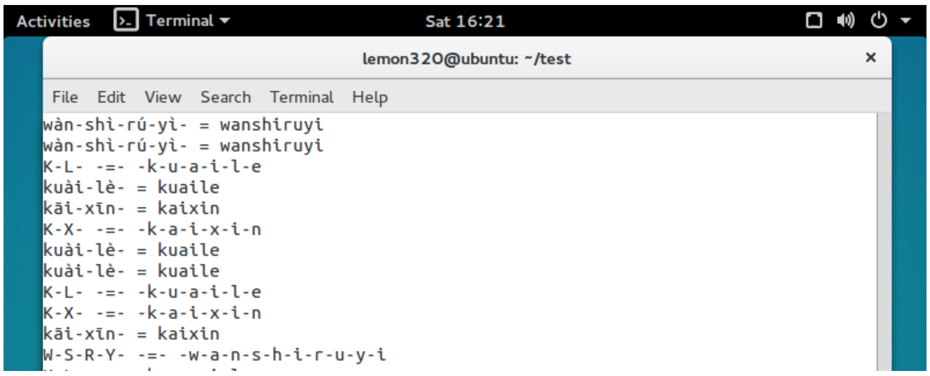
c. Furthermore, I want to test the maximum length of input strings in those functions, and for example in the ***get_pinyin*** function, I will input a long Chinese poem to detect whether or not this python program will meet any problem. In fact, in this situation, no matter how many Chinese characters I put into this library, it can translate them successfully.



```
xun-yang-jiang-tou-ye-song-ke- -feng-ye-di-hua-qiu-se-se- -zhu-ren-xia-ma-ke-zai
-chuan- -ju-jiu-yu-yin-wu-guan-xian- -zui-bu-cheng-huan-can-jiang-bie- -bie-shi-
mang-mang-jiang-jin-yue- -hu-wen-shui-shang-pi-pa-sheng- -zhu-ren-wang-gui-ke-bu
-fa- -xun-sheng-an-wen-dan-zhe-shui- -pi-pa-sheng-ting-yu-yu-chi- -yi-chuan-xian
g-jin-yao-xiang-jian- -tian-jiu-hui-deng-zhong-kai-yan- -qian-hu-wan-huan-shi-ch
u-lai- -you-bao-pi-pa-ban-zhe-mian- -zhuan-zhou-bo-xian-san-liang-sheng- -wei-ch
eng-qu-diao-xian-you-qing- -xian-xian-yan-yi-sheng-sheng-si- -si-su-ping-sheng-b
u-de-zhi- -di-mei-xin-shou-xu-xu-dan- -shuo-jin-xin-zhong-wu-xian-shi- -qing-lon
g-man-nian-mo-fu-tiao- -chu-wei-ni-shang-hou-liu-yao- -da-xian-cao-cao-ru-ji-yu-
 -xiao-xian-qie-qie-ru-si-yu- -cao-cao-qie-qie-cuo-za-dan- -da-zhu-xiao-zhu-luo-
yu-pan- -jian-guan-ying-yu-hua-di-hua- -you-yan-quan-liu-shui-xia-tan- -bing-qua
n-leng-se-xian-ning-jue- -ning-jue-bu-tong-sheng-zan-xie- -bie-you-you-chou-an-h
en-sheng- -ci-shi-wu-sheng-sheng-you-sheng- -yin-ping-zha-po-shui-jiang-beng- -t
ie-qi-tu-chu-dao-qiang-ming- -qu-zhong-shou-bo-dang-xin-hua- -si-xian-yi-sheng-r
u-lie-bo- -dong-chuan-xi-fang-qiao-wu-yan- -wei-jian-jiang-xin-qiu-yue-bai- -che
n-yin-fang-bo-kuo-xian-zhong- -zheng-dun-yi-shang-qi-lian-rong
```

Pic 3. Maximum length

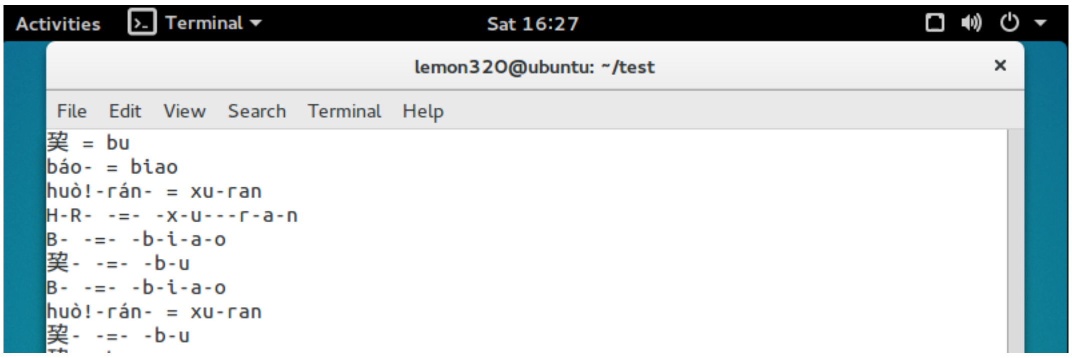
d. Usually, the Chinese characters include two kinds of different types: one is called simplified Chinese characters, and the other is called Traditional Chinese characters. When I test those two types of Chinese characters with this library, I found that it can recognize both of them very well.



```
wàn-shì-rú-yì- = wanshiruyi
wàn-shì-rú-yì- = wanshiruyi
K-L- -=- -k-u-a-i-l-e
kuài-lè- = kuaile
kāi-xīn- = kaixin
K-X- -=- -k-a-i-x-i-n
kuài-lè- = kuaile
kuài-lè- = kuaile
K-L- -=- -k-u-a-i-l-e
K-X- -=- -k-a-i-x-i-n
kāi-xīn- = kaixin
W-S-R-Y- -=- -w-a-n-s-h-i-r-u-y-i
```

Pic 4. Traditional Chinese Characters

e. Finally, I will try to pick up some uncommon used Chinese characters to test if this program can recognize those characters accurately. Actually, in this library most Chinese characters can be translated successfully, but there still exits some characters which are not recognized.



```
菐 = bu
báo- = biao
huò!-rán- = xu-ran
H-R- -=- -x-u---r-a-n
B- -=- -b-i-a-o
菐- -=- -b-u
B- -=- -b-i-a-o
huò!-rán- = xu-ran
菐- -=- -b-u
```

Pic 5. Uncommon used Character

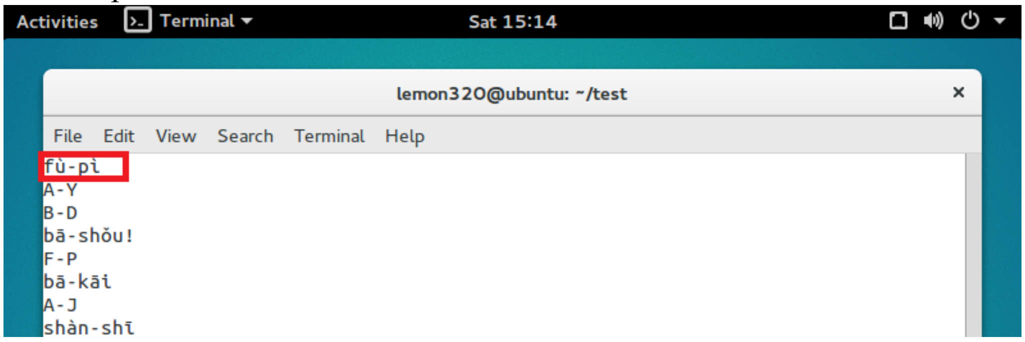
2. Find some bugs

Therefore, after I try lots of times to put different Chinese characters into this library, I successfully find some bugs.

a) According to the syntax of the Chinese characters, under the different situations, one character may have two or more pronunciations. However, after I tested this python library in several times, I found that most Chinese characters only have one pronunciations, and absolutely this causes lots of bugs. For example:

p.get_pinyin(u" 开 辟 ") -->u'kai-pi' p.get_pinyin(u" 复 辟 ") -->u'fu-pi'

correct pronunciation: fu-bi

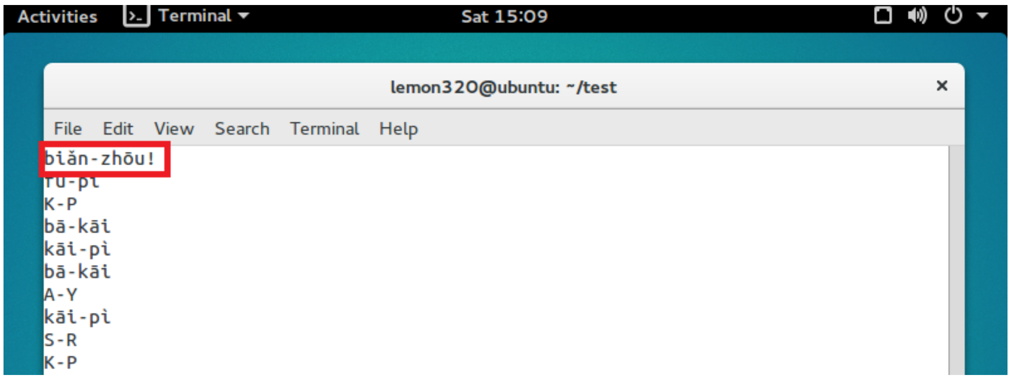


```
fù-pì
A-Y
B-D
bā-shǒu!
F-P
bā-kāi
A-J
shàn-shī
```

Pic 6. Bug1-1

p.get_pinyin(u"扁担") -->u'bian-dan'   p.get_pinyin(u"扁舟") -->u'bian-zhou

correct pronunciation: pian-zhou

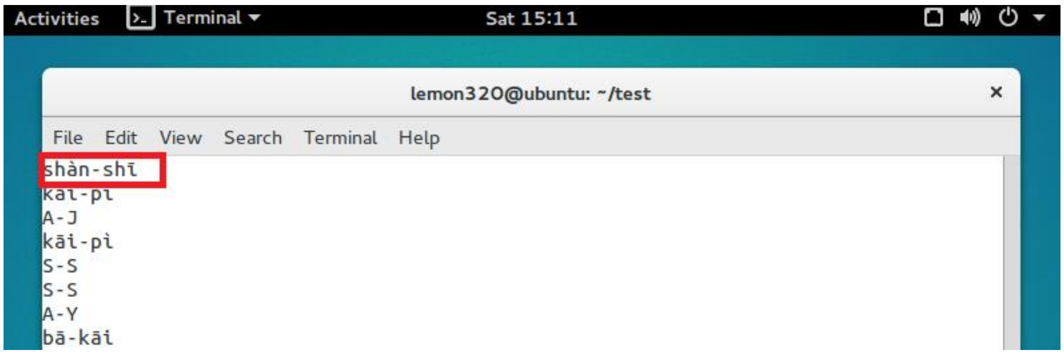


```
biǎn-zhōu!
fu-pi
K-P
bā-kāi
kāi-pì
bā-kāi
A-Y
kāi-pì
S-R
K-P
```

Pic 7. Bug1-2

p.get_pinyin(u" 禅 让 ") -->u'shan-rang' p.get_pinyin(u" 禅 师 ") -->u'shan-shi'

correct pronunciation: chan-shi



```
shàn-shī
kai-pi
A-J
kāi-pì
S-S
S-S
A-Y
bā-kāi
```

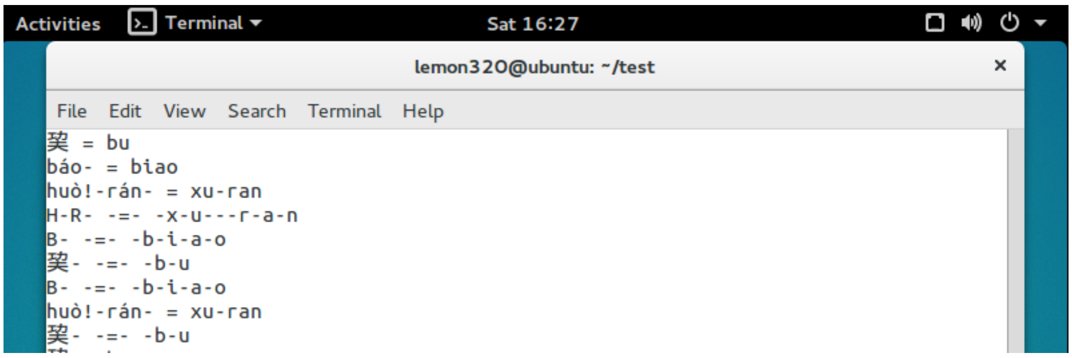
Pic 8. Bug1-3

b) Furthermore, when I tried to pick up some uncommon Chinese characters and put them into the library, I found that even though this program can successfully recognize most of Chinese characters, there still exists some characters which cannot be recognized. For example:

p.get_pinyin(u"踄") -->u'\u5ded'   correct pronunciation:bu

p.get_pinyin(u"覅") -->u'bao'   correct pronunciation:biao

p.get_pinyin(u"湑然") -->u'huo-ran'   correct pronunciation:xu-ran



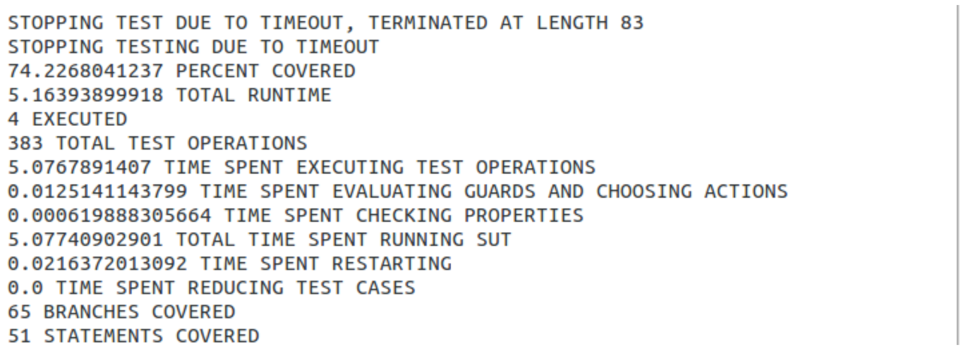
```
巭 = bu
báo- = biao
huò!-rán- = xu-ran
H-R- -=- -x-u---r-a-n
B- -=- -b-i-a-o
巭- -=- -b-u
B- -=- -b-i-a-o
huò!-rán- = xu-ran
巭- -=- -b-u
```

Pic 9. Bug2

3. Identify the tester work

Personally, I think this is a good library, but not a very perfect library since there still exists some bugs should be fixed. For the multiple Chinese pronunciations issue, I think this is a big problem and the designers did not consider well about this aspect of problems when they were building this library. I know it is very difficult to solve this problem, but it absolutely causes lots of errors. For the used Chinese characters issue, I think this is not a very critical problem because most common Chinese characters already have been recognized successfully.

For the coverage, in my last process report, my code coverage is 64.95%, while in my final report, the code coverage has grown to 74.2%, and we can see it in the following picture.



```
STOPPING TEST DUE TO TIMEOUT, TERMINATED AT LENGTH 83
STOPPING TESTING DUE TO TIMEOUT
74.2268041237 PERCENT COVERED
5.16393899918 TOTAL RUNTIME
4 EXECUTED
383 TOTAL TEST OPERATIONS
5.0767891407 TIME SPENT EXECUTING TEST OPERATIONS
0.0125141143799 TIME SPENT EVALUATING GUARDS AND CHOOSING ACTIONS
0.000619888305664 TIME SPENT CHECKING PROPERTIES
5.07740902901 TOTAL TIME SPENT RUNNING SUT
0.0216372013092 TIME SPENT RESTARTING
0.0 TIME SPENT REDUCING TEST CASES
65 BRANCHES COVERED
51 STATEMENTS COVERED
```

Pic. 10 Code coverage

4. Disadvantage of TSTL

During I used the TSTL tool to test my python library, I think there are mainly 2 drawbacks about this language. The first one is that, sometimes the program still generate a series of warning such that: No data was collected. The second one is that my python library is testing about Chinese characters, so in each time, I need to copy the (# -*- coding: utf-8 -*-") into the first line of the "sut.py" to specify this file is the utf-8 encoding. Therefore, in the future work, I hope when compile a tstl file with the utf-8 encoding, the SUT file can generate the command (# -*- coding: utf-8 -*-") automatically.

5. **Advantage of TSTL**

On the other side, I do see that users can get many benefits from TSTL language when they are testing other programing codes. Firstly, the users don't need to spend lot of time on writing a test harness by themselves since the TSTL file can generate the SUT file automatically, and at the same time this "sut.py" file has already been imported from the "randomtester.py", so users just use the command "python randomtester.py" which is really convenient to process the random testing. Furthermore, the grammar of TSTL language is not very difficult for users to learn, so I think this kind of testing language must be spread widely in the future.

6. **Summary**

In this class I am very happy to learn the TSTL language, and I think this is a very convenient and useful tool to test other programs. At the end of this term, I successfully utilize the TSTL language to find some bugs from a python library which is used for translating the Chinese characters to the Latin alphabet. Even through there are still some little drawbacks when I am using TSTL language, I think those bugs will be fixed very soon in the future, and this kind of testing language must be welcomed warmly by more program testers.