EECS 1210 Introduction to Programming in Python
Spring Semester 2024
Prof. Pai H. Chou

# Self-Check 14

Answer the following questions to check your understanding of your material. Expect the same kind of questions to show up on your tests. This self check is for Python Standard Library part 2.

## 1. Definitions and Short Answers - functions

1. Assuming you have
   `import string`
   What is the value of
   a. `string.ascii_letters`
      `'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ'`
   b. `string.ascii_lowercase`
      `'abcdefghijklmnopqrstuvwxyz'`
   c. `string.punctuation`
      `'!"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}~'`
   d. `string.whitespace`
      `' \t\n\r\x0b\x0c'`
2. Why is it more preferable to check if a character `c` is a white space by
   `if c in string.whitespace:`
   instead of
   `if c in {' ', '\n', '\t'}:`
   ?
   cause string.whitespace includes more
3. Assuming you have
   `from string import Template`
   `s = Template('$x and $y')`
   What is the value of the following expressions?
   a. `s.substitute(x='hello', y='world')`
   b. `s.substitute(x=0x20, y=20)`
   c. `s.substitute(x=2., y=.2)`
4. In vim,
   a. what do you type to search for the string "or"?/or
   b. what do you type to find string "or" **at the beginning of the line**?/^or
   c. what do you type to find string "or" **at the end of the line**?/or$
   d. what do you type to find string "or" as the only string **on the entire line**?/^or$
   e. what do you type to find **words that end with** "or"?/or\>

f. what do you type to find **words that begin with** "or"?/\<or

g. what do you type to find strings "or" **that are not part of another word**?/\<or\>

5. In vim, what is the meaning of
   a. :%s
   b. :%s/abc/xyz/
   c. :%s/abc/xyz/g

6. In python, you can import a module named re. What does it stand for?

7. Given the following functions in the re module
   re.search(*pattern*, *source*) returns matched object for first found
   re.findall(*pattern*, *source*) returns list of found strings
   Assume
   import re
   What are the values of the following expressions?
   a. re.findall('Gary', 'Mary had a little lamb')
   b. re.findall('Mary', 'Mary had a little lamb')
   c. re.findall('Mary', 'Mary and Gary had a little lamb')
   d. re.findall('lamb', 'Mary had a little lamb little lamb')
   e. re.findall('a', 'Mary had a little lamb')

8. Assume the following
   import re
   s = re.findall(*pat*, 'Mary, Gary, and Cary had a scary lamb')
   What is the value of pattern *pat* that will yield the following values for s?

   a. ['Mary', 'Gary', 'Cary']
   b. ['Mary, 'Gary']
   c. ['Mary', 'Gary', 'Cary', 'cary']
   d. ['Cary', 'cary']
   e. ['Mary', 'Gary', 'Cary', 'scary']

9. Assume
   import re
   What are the values of the following expressions?
   a. re.findall('[a-z]*are', 'We scare because we care')
   b. re.findall('[a-z]+are', 'We scare because we care')
   c. re.findall('[a-z]*are', 'We are here to scare and care')
   d. re.findall('[a-z]+are', 'We scare because we care')
   e. re.findall('[a-z]*are', 'He cares that we are scared')
   f. re.findall('[a-z]+are', 'He cares that we are scared')

g. re.findall('[a-z]*are[sd]',
'He cares that we are scared')
h. re.findall('[a-z]*are[sd]?',
'He cares that we are scared')
i. re.findall('[a-z]+are[sd]',
'He cares that we are scared')
j. re.findall('[a-z]+are[sd]?',
'He cares that we are scared')

10. Assume

```
import re
```

What are the values of the following expressions? Explain the difference between c and d.

a. re.findall('[a-z]*','He cares that we are scared')
b. re.findall(r'\w*','He cares that we are scared')
c. re.findall(r'\bare\b','He cares that we are scared')
d. re.findall('\bare\b','He cares that we are scared')

11. Assume

```
import re
```

What are the values of the following expressions?

a. re.findall('Mary','Mary and Mary's lamb like Mary')
b. re.findall('^Mary','Mary and Mary's lamb like Mary')
c. re.findall('Mary$','Mary and Mary's lamb like Mary')
d. re.search('^Mary','Mary and Mary's lamb like Mary')
e. re.search('Mary$','Mary and Mary's lamb like Mary')
f. re.findall('Mary\S+','Mary and Mary's lamb like Mary')

12. Fill in the following blanks

| meaning | backslash notation | equivalent ASCII set |
|---|---|---|
| word boundary | \b | n/a |
| not beginning or end of word | | n/a |
| decimal digit | | [0-9] |
| not decimal digit | \D | |
| white space | | [ \t\n\r\f\v] |
| not whitespace | | |
| alphanumeric | | [a-zA-Z0-9_] |

| | | |
|---|---|---|
| not alphanumeric | | |
| beginning of string | | n/a |
| end of string | | n/a |

13. Assume

```
import re
```

What are the values of the following expressions? Explain the difference between '*' and '*?' as regular expressions

a. re.findall(r'\b.are', 'He cares that we are scared')
   ['care', ' are']
b. re.findall(r'\b.*are', 'He cares that we are scared')
   ['He cares that we are scare']
c. re.findall(r'\b.*?are', 'He cares that we are scared')
   ['He care', ' that we are', ' scare']

14. What are the values of the following expressions? assume

```
import re
```

a. 'To be, or not to be--that is the question!'.split()
b. 'To be, or not to be--that is the question!'.split('-')
c. 'To be, or not to be--that is the question!'.split('--')
d. re.split(' ', 'To be or not--that is the question!')
e. re.split(r'\W', 'To be or not--that is the question!')
f. re.split(r'\W+', 'To be or not--that is the question!')
g. re.split(r'-+', 'To be or not--that is the question!')
h. re.split(r'\s*/+\s*','To/be/// or //not/ that//question')

15.

| reguar expresison | explanation |
|---|---|
| [A-Z][a-z]{2} | one capital letter followed by two lower-case letters |
| 0?[1-9]\|[12]\d\|3[01] | |
| \d\d:\d\d:\d\d | |
| \d{1,4} | |

16. Assume

```
import re
```

Does the following result in match?

```
m = re.fullmatch(r'^.*(\d\d):(\d\d):(\d\d).*(\d{4})$',
                 'Thu Jul 18 14:33:28 PDT 2019')
```

If so, what are the values of

a. m.group(1)
b. m.group(2)
c. m.group(3)
d. m.group(4)

17. Assume

```
import re
```

What is the value of

a. `re.sub('-', '/', 'today 5-20-2019, tomorrow 5-21-2019')`

b. `re.sub('.', '/', 'today 5.20.2019, tomorrow 5-21-2019')`
c. `re.sub(r'\.', '/', 'today 5.20.2019, tomorrow 5-21-2019')`

18. Assume

```
import re
```

What is the value of

```
re.sub(r'(\d+)/(\d+)/(\d+)', r'\3/\2/\1',
    'today 5/20/2019, tomorrow 5/21/2019')
```

19. is the difference between the built-in float type and Decimal class in the decimal module?
    a. What is the value of
    `1.1 + 2.2 == 3.3`
    ?
    b. Assuming you have `from decimal import Decimal`, what is the value of
    `Decimal('1.1') + Decimal('2.2') == Decimal('3.3')`
    ?
20. Assume you have `from fractions import Fraction`, what is the value of
`Fraction(16, -10)`
?

21. Consider the first Tkinter program

```
1 import tkinter
2 root = tkinter.Tk()
3 f = tkinter.Frame(root)
4 f.pack()
5 l = tkinter.Label(f, text='Hello world')
6 l.pack()
7 b = tkinter.Button(f, text='Quit', command=root.destroy)
8 b.pack()
9 tkinter.mainloop()
```

a. What kind of user interface object is tkinter.Label on line 5?

it is read-only text that is displayed in the user interface.

b. What kind of object is root.destroy on line 7?
it is a function (method) that is called when the "Quit" button is clicked.

c. Does line 9 run forever? Or in what condition does line 9 finish?
when root.destroy is called

22. The second version of the Tkinter program is as follows, where the difference is highlighted in pin k:

```
1  import tkinter
2  root = tkinter.Tk()
3  f = tkinter.Frame(root, width=200, height=150)
4  f.pack_propagate(0)
5  f.pack()
6  l = tkinter.Label(f, text='Hello world')
7  l.pack(side=tkinter.TOP)
8  b = tkinter.Button(f, text='Quit', command=root.destroy)
9  b.pack(side=tkinter.BOTTOM)
10 tkinter.mainloop()
```

a. What is the purpose of line 4?

b. What is the purpose of line 7 and line 9?

23. In the Calendar v2 example, the first page of the source code looks like this:

```
1  import tkinter
2  root = tkinter.Tk()
3  f = tkinter.Frame(root, width=250, height=200)
4  f.pack_propagate(0)
5  f.pack()
6  import datetime
7  today = datetime.date.today()
8  current_year, current_month = today.year, today.month
9  import calendar
10 cal = calendar.TextCalendar(6)
11    calstr = tkinter.StringVar()
12 calstr.set(cal.formatmonth(current_year, current_month))
13 l = tkinter.Label(f, textvariable=calstr,
14           justify=tkinter.LEFT,   font=('Courier', 12))
15 l.pack(side=tkinter.TOP)
16 b = tkinter.Button(f, text='Quit',command=root.destroy)
17 b.pack(side=tkinter.BOTTOM)
```

a. On line 13, why does it pass parameter textvariable=calstr (created and set on lines 11-12) instead of parameter text= some string value as before?

24. Continuing with the same example, consider part of the second page of the source code, (abridged)

```
18 def prev_month():
19     global current_year, current_month
20     current_month -= 1
21     if current_month == 0:
22         current_month = 12
23         current_year -= 1
24     calstr.set(cal.formatmonth(current_year,
                  current_month))
...
52 pm = tkinter.Button(f, text='Prev', command=prev_month)
53 pm.pack(side=tkinter.LEFT)
...
56 tkinter.mainloop()
```

a. How is the prev_month function (line 18) called? Can any parameters be passed to it?
b. Why is line 19 necessary?
c. What is the effect of line 24 on the user interface?

25. Continuing with the same example, one improvement is to replace the prev_month and next_month functions with the following:

```
32 def shift_month(add_or_sub):
33     global current_year, current_month
34     current_month = add_or_sub(current_month, 1)
35     if current_month == 0:
36         current_month = 12
37         current_year -= 1
38     elif current_month == 13:
39         current_month = 1
40         current_year += 1
41     calstr.set(cal.formatmonth(current_year,
                  current_month))
42 import operator
43 next_month = lambda : shift_month(operator.add)
44 prev_month = lambda : shift_month(operator.sub)
```

a. Why is it better to combine the functionality of next_month and prev_month functions into one function?
b. Can this new function shift_month be passed as callback to (line 52 in the previous problem), i.e.,

```
pm = tkinter.Button(f, text='Prev', command=shift_month)
```

? Why or why not?

c. Can line 52 be written as

```
pm = tkinter.Button(f, text='Prev',
                    command=shift_month(operator.sub))
```

instead? Why or why not?

d. What is the difference between part c and the following:

```
pm = tkinter.Button(f, text='Prev',
                    command=lambda: shift_month(operator.sub))
```

which one is correct?

26. In calendar v3, the *grid* layout manager is used instead of *pack*. What is the difference?
27. What is the meaning of
    a. `f.grid(row=0, column=0)  # f is an instance of tkinter.Frame`
    b. `l.grid(row=0, column=0, columnspan=3)  # l is a tkinter.Label`
28. What is the inlined equivalent code to

    `[w.grid(row=1,column=i) for i,w in enumerate([pm, b, nm])]`

    where pm, b, nm are all instances of tkinter.Button?
29. In the **word finder** example, the text to search is entered into the Entry widget that is set up using t he following code

```
pat_str = tkinter.StringVar()
pat_entry = tkinter.Entry(f,   textvariable=pat_str)
pat_entry.grid(row=0, column=1)
```

    What is the purpose of the text typed into the Entry? Does it need to be processed first before bein g passed to re.search?
30. Continuing with the **word finder** example, a ListBox instance named result_box is used to display t he matched results. Explain the purpose of the following three lines of code that invoke methods on the list box:

```
result_box.delete(0, tkinter.END)
for i, w in enumerate(matched_words):
    result_box.insert(i, w)
```

31. Continuing with the **word finder** example,
    a. Explain what the second line does when reading the list of words from the file:

```
fh = open('/usr/share/dict/words')
words = list(map(lambda x: x[:-1] if x[-1]=='\n' else x,
                 fh.readlines()))
fh.close()
```

    b. Give an alternative code that uses str split() method in conjunction with fh.read() to achiev e the same as the second line.

```
'\n'.split(fh.read())
```

32. Give the regular expressions and the corresponding description

| | |
|---|---|
| strings containing six (or more) o's | (.*?o){6} |
| string containing exactly six o's | ^(.*?o){6}[^o]$ |

| | |
|---|---|
| words that contain at least a 3-character sequence twice | (.{3}).*\1 |
| 8-character palindrome | ^(.)(.)(.)(.)\4\3\2\1$ |
| word (of any length) in the form of the same substring twice (e.g., soso) | ^(.*)\1$ |

33. In the clock example, can the time be updated every second if you don't use a thread? Why or why not?
34. Continuing with the clock example, does the following code do?

```
import threading
th = threading.Thread(target=update_clock)
th.start()
```

35. In the clock example, why is it necessary to have the line quit = True after f.mainloop()?

```
f.mainloop()
quit = True
```

considering that the update_clock() function looks like

```
import threading, time
def update_clock():
    while not quit:
        now = datetime.datetime.now()
        clockstr.set(f"Date: {now.year}/{now.month}/{now.day} Time: {now.hour:02d}:{now.minute:02d}:{now.second:02d}")
        time.sleep(1)
```

## 2. Programming

1. (Difficulty: ★★☆☆☆) Write a command-line program named dehtml.py to remove tags from an HTML file and write the file into a plain text file.

   Background: HTML, for Hypertext Markup Language, is the way web pages are formatted. It contains tags in the form of of angle bracketed tags. For example,

| tag | description | example |
|---|---|---|

| | | |
|---|---|---|
| <h1>heading 1</h1> | 1st-level heading | heading 1 |
| <a href="link">click me</a> | clickable anchor | click me |
| paragraph 1 <br> paragraph 2 | paragraph | paragraph 1<br><br>paragraph 2 |

The purpose of dehtml.py is to take out these angle-bracketed tags (i.e., formatting) and leave just t he original text.

You can test your code with any web page that is saved as an HTML file and name it with the exte nsion of ".html" (e.g., myweb.html).

Your program should open the file specified on the command line. It should check to make sure th e file name is named with ".html" extension. If it can be opened, it should write the output into a fi le whose name is the same as the HTML file except the ".html" suffix is replaced with a ".txt" suffi x. For example,

$ python3 dehtml.py myweb
Error: File name should have .html suffix
$ python3 dehtml.py myweb.html
Wrote file myweb.txt
$

If successful, it creates a file named myweb.txt without the HTML tags.

Hint: use regular expressions to specify the tags so they can be split.
You may replace each tag with a blank space.

2. (Difficulty: ★★★☆☆) Write a Tkinter version of the "rock, paper, scissors" game. It should di splay three rows. You may use the unicode character for

rock: chr(0x270a) ✊
scissors: chr(0x270c) ✌
paper: chr(0x270b) ✋

| |
|---|
| Your hand: [✋] [✊] [✌] |
| Computer's hand: |
| Status: [Retry] [Quit] |

This initial screen should offer three buttons, one for each of [✋] [✊] [✌].
Upon pressing one of the buttons, the program should generate a random hand and compare agains t the user's hand. The outcome should be displayed on the Status line: "Computer wins", "User wi ns", or "Tied!". In addition, the Status line should also offers two buttons: [Retry] and [Quit].

For example, after you click "paper" button, you may see this:

| |
|---|
| Your hand: [✋] [✊] [✌] you clicked ✋ |
| Computer's hand: ✊ |
| Status: User Wins [Retry] [Quit] |

After clicking Retry, the screen resets back to the initial screen as shown above.

3. (Difficulty: ★★★★☆) Write a Tkinter version of the postfix calculator for dates (from the pre vious assignment).

The interface should look be divided into a left pane and a right pane. The left pane contains butto ns for the verbs: 'today', 'tomorrow', 'yesterday', 'add', 'sub', 'swap'.
It should also contain an Entry widget for text entry and a 'push' button.

The right pane should be a ListBox to show the content of the stack. For simplicity, the stack gro ws "downward" so that the bottom of the stack is on the top side of the ListBox, and new items get pushed to below the last item. Note that [ ] indicates a button that can be clicked.

| | |
|---|---|
| [today]<br>[tomorrow]<br>[yesterday]<br>[add]<br>[sub]<br>[swap]<br><br>[ ] [push]<br>error message here | date(2019, 12, 3)<br>date(2019, 12, 4)<br>date(2019, 12, 2)<br>days(4)<br><br>[Clear] [Quit] |

In the Entry field, you may enter any valid Python expressions that can be pushed. You can also pu sh buttons to enter instead. Using the code from the previous assignment, you should be able to en

ter

date(*year*, *month*, *day*)
days(*d*)
weeks(*w*)
months(*m*)

Hint: you may use the eval() function to convert from the text in the Entry into the corresponding P ython data structure. To be safe, you would want to allow access to only your date, days, weeks, a nd month classes but block out other symbols. Note that the user could type in arbitrary code, but i f an exception occurs during eval(), or if eval() does not return a valid type, then you may indicate error in the "error message here" (as a Label) shown in the example above.

The ListBox should display the same content as shown in the list returned by datecalc() function. Unlike the word finder example, where the entire content is deleted before new content is added on every refresh, here you only need to remove the bottom entry for a pop, or append the new entry to the end for a push.

Hint: It may be easier if the ListBox only serves the display purpose instead of also working as the stack data structure.