

Compsci 101

More Recursion and Modules

d is:

0 -> haiku.txt

1 -> labtemplate.txt

2 -> lecturetemplate.txt

Susan Rodger

Alex Steiger

December 2, 2025



X is for ...

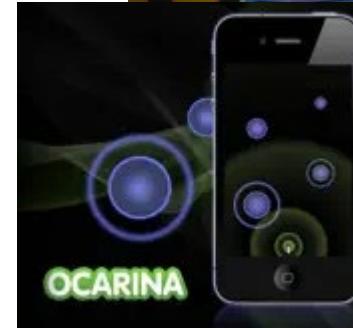


- **XOR**
 - (a or b) and not (a and b), a.k.a. symmetric difference
- **XML**
 - eXtensible Markup Language
- **Xerox Parc**
 - From Mice to Windows

The Power of Collaboration: Ge Wang, Duke alum - Prof. at Stanford

- Duke 2000: Music and Computer Science
 - <https://www.stanforddaily.com/2016/03/09/qa-with-ge-wang-father-of-stanford-laptop-orchestra/>
 - <http://www.youtube.com/watch?v=ADEHmkL3HBg>
- About Design in CompSci 308

Our investment into a huge and meticulous design process was a huge factor in making later progress. 35000+ lines of code / design / documentation gave us a project we were all very happy and proud to be a part of.



Announcements

- Assign 7 Recommender due TODAY!
- APT-7, due Thursday, Dec 4
- Assign 8 due December 5
 - Can be turned in by December 9 with NO PENALTY
- Lab 10 Friday – do prelab before going
- Final Exam – Thursday, December 11, 7pm
 - In French Science 2231<- NOTE different room than where our lecture is – [Block exam time]

Interested in being a UTA?

- Enjoy CompSci101?
- Would like to help others learn it?
- Consider applying to join the team!
- <https://www.cs.duke.edu/undergrad/uta>
- Apply soon - ignore the deadline, they still need UTAs for 101

Assignment 8:

More samples from previous semesters

PFTD

- Review Recursion
- Modules and exceptions
- An APT

Review: Recursion Summary

- **Make simpler or smaller calls**
 - Call itself with different input
- **Must have a base case when no recursive call can be made**
 - Example - The last folder in the folder hierarchy will not have any subfolders. It can only have files. That forms the base case
 - This is the way out of recursion!

Problem: is a number in a list?

- Is 5 in [7, 5, 6, 8] ?
- Is 8 in [5, [[7,4], 9, [3, 4]], [4, [5, [2, [8, 1], 4,]], 5]] ?

Revisit the APT Bagels Recursively

```
filename: Bagels.py
```

```
def bagelCount(orders) :  
    """  
    return number of bagels needed to fulfill  
    the orders in integer list parameter orders  
    """
```

1. `orders = [1,3,5,7]`

Returns: 16

No order is for more than a dozen, return the total of all orders.

- 2.

```
orders = [11,22,33,44,55]
```

Returns: 175 since $11 + (22+1) + (33+2) + (44+3) + (55+4) = 175$

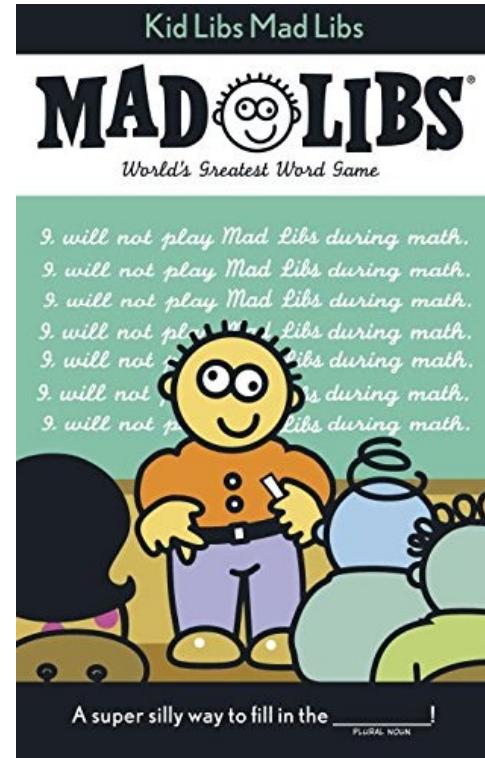
Why use modules?

- Module – Python file (.py file)
- Can have several modules work together
- Easier to organize code
- Easier to reuse code
- Easier to change code
 - As long as the “what” is the same, the “how” can change
 - Ex: `sorted(...)`, one function many sorting algorithms

Modules for Creating

- “~~MadLibs~~” → Tag-a-Story
 - User chooses template
 - Computer fills everything in

In lecture I saw a <color> <noun>
For lunch I had a <adjective> <food>
The day ended with seeing a <animal>
<verb> in <place>



From <noun> to story

In lecture I saw a
<color> <noun>

For lunch I had a
<adjective> <food>

The day ended with
seeing a <animal>
<verb> in <place>



[This Photo](#) by Unknown Author is licensed under [CC BY-NC-ND](#)

In lecture I saw a
magenta house

For lunch I had a
luminous hummus

The day ended with
seeing a cow sleep
in Mombasa



[This Photo](#) by Unknown Author is licensed under [CC BY-NC-ND](#)



[This Photo](#) by Unknown Author is licensed under [CC BY-SA](#)

Demo

- Run storyline.py
- Show Lecture template
- Show Haiku's
- Make modifications

Let's create/modify a story

- Choose a template or make a new one
 - We'll choose `lecturetemplate.txt` first
- Add a new category/replacement
 - We'll choose `number` and list some choices
- Run the program and test our modifications
 - Randomized, hard to test, but doable

Main Parts (3 modules) for tag-a-story

- Put everything together, the template and words
 - Storyline.py
- Loading and handling user choosing templates
 - TemplateChooser.py
- Loading and picking the word for a given tag
 - Replacements.py

Main Parts (3 modules) for tag-a-story

- Put everything together, the template and words
 - Storyline.py
- Loading and handling user choosing templates
 - TemplateChooser.py
- Loading and picking the word for a given tag
 - Replacements.py

Creating a story

- Main steps in `Storyline.py`
 - Get template – use module `TemplateChooser`
 - Go through template
 - Get words for a tag – use module `Replacements`
 - Replace tag with word
- Using modules
 - Assume they work
 - Only care *what* they do, not *how* (abstraction!)

Modules in Action: makeStory() is in Storyline.py

- How can we access TemplateChooser functions?
 - import and access as shown

```
41  def makeStory():
42      """
43          let user make a choice of
44          available templates and print
45          the story from the chosen template
46      """
47
48      lines = TemplateChooser.getTemplateLines("templates")
49      st = linesToStory(lines)
50      print(st)
```

Modules in Action: linesToStory() is in Storyline.py

- We call doWord() – does replacements for words

```
27     def linesToStory(lines):  
28         """  
29             lines is a list of strings,  
30             each a line from a template file  
31             Return a string based on substituting  
32             for each <tag> in each line  
33         """  
34         story = ""  
35         for line in lines:  
36             st = ""  
37             for word in line.split():  
38                 st += doWord(word) + " "  
39             story += st.strip() + "\n"  
40     return story
```

Understanding Code/Module

doWord is in Storyline.py

- What does getReplacement do?
 - How does getReplacement do it?

```
10  def doWord(word):  
11      """  
12          word is a string  
13          if word is <tag>, find replacement  
14          and return it. Else return word  
15          """  
16      start = word.find("<")  
17      if start != -1:  
18          end = word.find(">")  
19          tag = word[start+1:end]  
20  
21          rep = Replacements.getReplacement(tag)  
22          return rep  
23      return word
```

Main Parts for tag-a-story

- Put everything together, the template and words
 - Storyline.py
- Loading and handling user choosing templates
 - TemplateChooser.py
- Loading and picking the word for a given tag
 - Replacements.py

Another module `TemplateChooser.py`

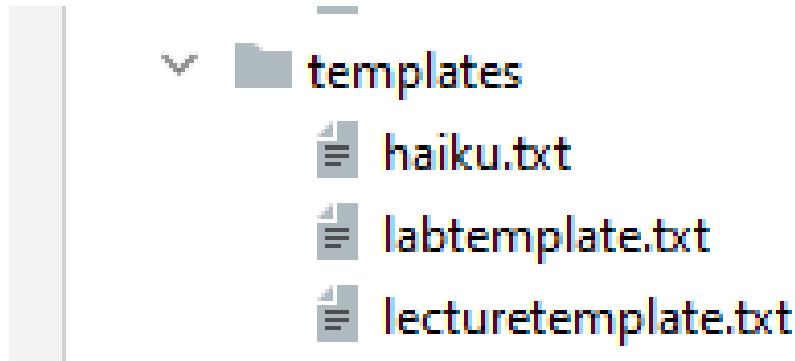
- **Get template**
 - `TemplateChooser.getTemplateLines(DIR)`
 - What:
 - From the templates in the directory DIR (type: str)
 - Return a list of strings, where each element is a line from one of the templates in DIR
- **Word for a tag**
 - `Replacements.getReplacement(TAG)`
 - What:
 - Return a random word that matches TAG (type: str)

Where is it called from?

- In module Storyline.py, function makestory

```
lines = TemplateChooser.getTemplateLines("templates")
```

- Where templates is a folder with three templates:



TemplateChooser.py Steps

- List all templates in the folder
- Get user input that chooses one
- Load that template
- Return as list of strings

These Steps in Code

getTemplateLines in TemplateChooser.py

- Read directory of templates, convert to dictionary
 - Let user choose one, open and return it

```
59  def getTemplateLines(dirname):  
60      """  
61          dirname is a string that's the name of a folder  
62          Prompt user for files in folder, allow user  
63          to choose, and return the lines read from file  
64      """  
65      d = dirToDictionary(dirname)  
66      lines = chooseOne(d)  
67      return lines
```

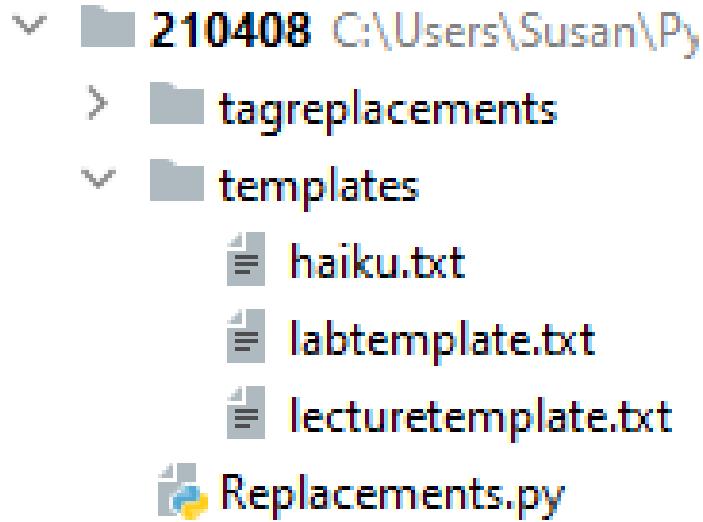
Creating User Menu

dirToDictionary in TemplateChooser.py

- What does this function return? What type?

```
11  def dirToDictionary(dirname):  
12      """..."""  
18      d = {}  
19      index = 0  
20      for one in pathlib.Path(dirname).iterdir():  
21          d[index] = one  
22          # print(type(one))  
23          index += 1  
24      return d
```

Folder in Pycharm



Output:

```
C:\Users\Susan\AppData\Loc  
0 haiku.txt  
1 labtemplate.txt  
2 lecturetemplate.txt  
-----  
choose one> 0  
the slimy bathtub  
reminded them of Africa  
chartreuse squeaky brown
```

pathlib Library

- Path:
“rodger/Pycharm/cps101/lab11/temp/haiku.txt”
- The **pathlib** library is more recent/Python3
 - Simpler, easier to use than functions from **os**
- Handles domain specifics!
 - Doesn't matter if on Windows, Mac, etc.
 - We worry about the *what*, it handles the *how*

pathlib Library cont.

- Path:
“rodger/Pycharm/cps101/lab11/temp/haiku.txt”
- **pathlib.Path(DIR).iterdir()**
 - Returns iterable of Path objects representing each “thing” in the directory DIR
- Path object’s .parts – tuple of strings, each element is a piece of a filename’s path
 - (‘rodger’, ‘Pycharm’, ‘cps101’, ‘lab11’, ‘temp’, ‘haiku.txt’)

Understanding the Unknown chooseOne in TemplateChooser.py

- We will return to this, but analyze parts now
 - What's familiar? What's not familiar ...

```
39  def chooseOne(d):
40      """
41      ...
42      """
43
44 46      while True:
45
46          for key in sorted(d.keys()):
47              print("%d\t%s" % (key, d[key].parts[-1]))
48              print("-----")
49
50          st = input("choose one> ")
51
52          try:
53              val = int(st)
54
55              if 0 <= val and val < len(d):
56                  return reader(d[val])
57
58      except ValueError:
59
60          print("please enter a number")
```

Python exceptions

- What should you do if you prompt user for a number and they enter "one"
 - Test to see if it has digits?
- Use exceptions with `try:` and `except:`
 - See code in function `chooseOne` from *TemplateChooser.py*



Handling Exceptions

- What happens: `x = int("123abc")`

```
46     st = input("choose one> ")
47     try:
48         val = int(st)
49         if 0 <= val and val < len(d):
50             return reader(d[val])
51     except ValueError:
52         print("please enter a number")
--
```

APT WordPlay

APT: WordPlay

Problem Statement

Given a phrase of words, your task is to return a string of the unique words from the phrase, with the words sorted using the following rules.

1. First the unique words should be sorted in reverse order based on their length (number of characters in the word)
2. For words the same length, they should be sorted in alphabetical order based on only the first letter of each such word
3. If there are ties after 1) and 2) criteria, then sort those words in reverse alphabetical order based on the last letter of each such word
4. If there are ties after 1), 2) and 3) criteria, then sort those words in alphabetical order based on the sub-word between the first and last letter of each such word.

APT WordPlay example

"mouse elephant moth zebra mole tiger moose moth mule"

Returns:

"elephant moose mouse tiger zebra moth mole mule"

Surprise