

Compsci 101

Modules, Exceptions, How to Solve it

Live Lecture

Susan Rodger
Nicki Washington
April 8, 2021

```
{ 'ITA': [1, 0, 0], 'JPN': [0, 1, 0], 'AUS': [0, 0, 1], 'KOR':  
[('AUS', [0, 0, 1]), ('CHN', [0, 1, 0]), ('GBR', [0, 0, 1]), (  
[('AUS', [0, 0, 1]), ('GBR', [0, 0, 1]), ('TPE', [0, 1, 1]), (  
[('TPE', [0, 1, 1]), ('CHN', [0, 1, 0]), ('JPN', [0, 1, 0]), (  
[('KOR', [3, 1, 0]), ('ITA', [1, 0, 0]), ('TPE', [0, 1, 1]), (  
['KOR 3 1 0', 'ITA 1 0 0', 'TPE 0 1 1', 'CHN 0 1 0', 'JPN 0 1
```

v is for ...



- **Viral Video**
 - Husky Dog sings with iPad – 18 million views
 - <https://www.youtube.com/watch?v=Mk4bmK-acEM>
- **Virtual Memory**
 - It is and is not there!
- **Virtual Reality**
 - Augmenting IRL
 - <http://bit.ly/vr-playlist>

Announcements

- APT-7 due TODAY!
- APT-8 out Tuesday, Apr 13
- Assign 6 Recommender, due Apr 22
 - One grace day, NO LATE DAYS, must be in Apr 23
- APT Quiz 2 – now through Sunday, April 11
 - Two Parts, Start on Sakai
- No Class Tues, April 13
 - Instead take Exam 3 on GradeScope

CompSci 101 – In-Person and Zoom

Last week of class (April 20 and 22)

- Limit is 60 people per day
- Can only come one of those days
- Sign up and I will let you know which day you can come (subject to space) and where to come
- Room is on West Campus
- Sign up here:
 - <http://bit.ly/101s21-inperson>

PFTD

- **Collaboration and Creativity**
 - The power of working together with code
- **Review modules and exceptions**
 - Concepts used in Lab 11, leveraging creativity
- **How to solve it**
 - Thinking about steps 5-7, Python, and scale
- **MedalTable APT**

The Power of Collaboration:

Ge Wang, Duke 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.



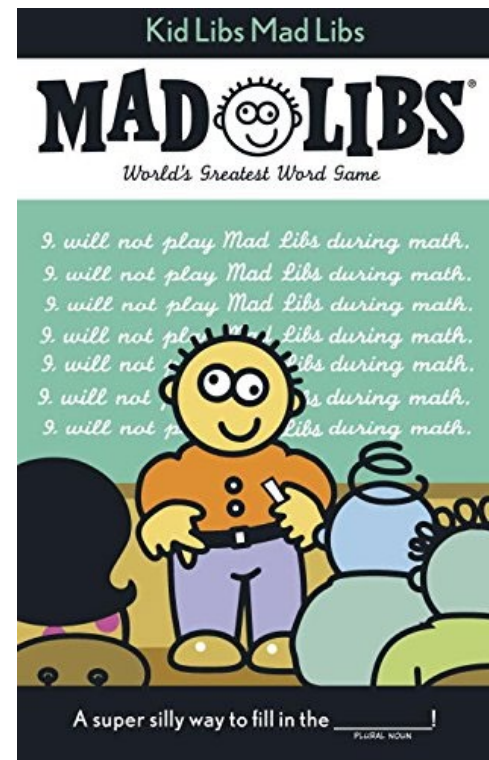
Why use modules?

- 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

In laterLab, 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>

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-NC-ND](#)



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

Demo

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 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 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 a module
 - Go through template
 - Get words for a tag – use a module
 - 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      lines = TemplateChooser.getTemplateLines("templates")
48      st = linesToStory(lines)
49      print(st)
```

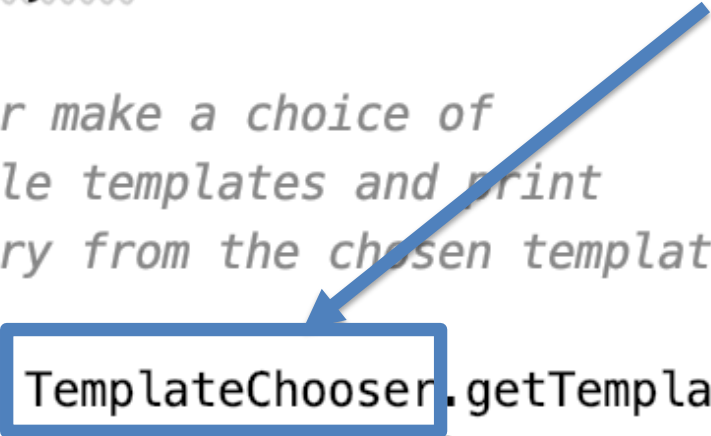
Modules in Action:

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     lines = TemplateChooser.getTemplateLines("templates")  
48     st = linesToStory(lines)  
49     print(st)
```

Another module (file)




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     lines = TemplateChooser.getTemplateLines("templates")
48     st = linesToStory(lines)
49     print(st)
```

A function in the
file:
TemplateChooser.py



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
```

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
```

Another
module (file)

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
```

A function in the
file:
Replacements.py

The other module's “what”

- **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)

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

TemplateChooser.py Steps

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

TemplateChooser.py Steps

- List all templates in the folder
 - pathlib Library
- Get user input that chooses one
 - Handle bad input → try...except
- Load that template
 - Open file, .readlines()
- 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
```

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
```

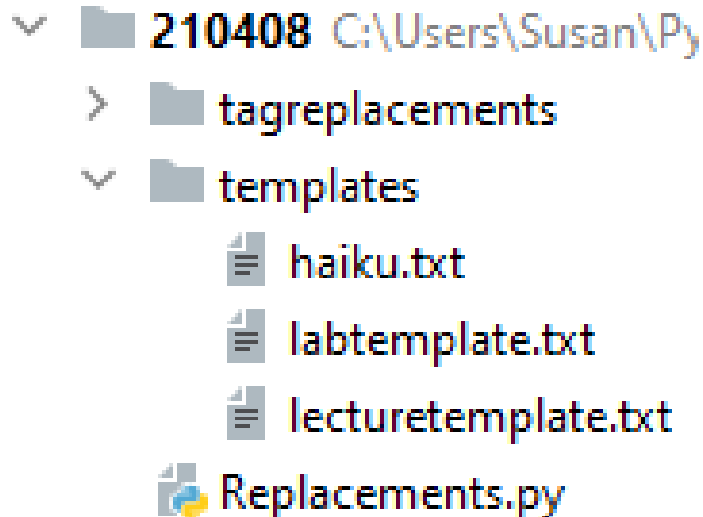
d is:

0 -> haiku.txt

1 -> labtemplate.txt

2 -> lecturetemplate.txt

Folder in Pycharm



Output:

```
C:\Users\Susan\AppData\Local\Programs\Python\Python38-32\Scripts\python.exe
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      """ ... """
46  while True:
47      for key in sorted(d.keys()):
48          print("%d\t%s" % (key, d[key].parts[-1]))
49      print("-----")
50      st = input("choose one> ")
51      try:
52          val = int(st)
53          if 0 <= val and val < len(d):
54              return reader(d[val])
55      except ValueError:
56          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")
53
```

WOTO-1 Modules
<http://bit.ly/101s21-0408-1>

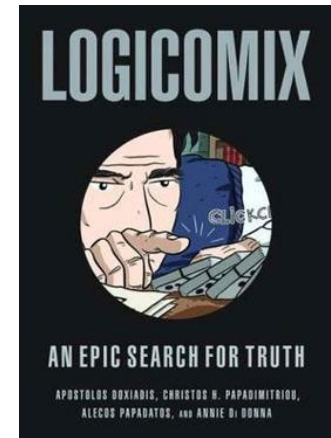


MODULE

Christos Papadimitriou

- Gödel prize for concept of "price of anarchy"

I would say that, quite generally, computer scientists are going to find themselves interacting more with other fields. I encourage my students to go completely wild in their curriculum, to go out and learn not only that which they think they should learn in order to be good computer scientists—usually mathematics and programming and engineering and so on—but learn about everything else, about psychology, economics, about business, about biology, about the humanities. (2008)



When and What's in CompSci 101

- **Problem to solve**
 - Use 7 steps
 - Step 5: How do you translate algorithm to code?
 - What do you use to solve it?
 - When do you use it?

What are the “what’s”?

- Data Structures: list, set, dictionary, tuple
- Loops and iterables: from for to while to iterdir()
- Other:
 - List comprehensions
 - Parallel lists
 - Lambda
 - If...if...if
 - If...elif...else

Quick When's and What's for 101

- Whichever makes more sense to you:
 - Parallel lists vs dictionaries
 - If...if...if vs if...elif...else
 - List comprehension vs for loop
 - Tuples vs Lists
 - If you want to prevent mutation -> tuples
 - Need single line function
 - Lambda vs create normal helper function

APT – Sorted Freqs

APT SortedFreqs

Problem Statement

The frequency with which data occurs is sometimes an important statistic. In this problem you'll determine how frequently strings occur and return a list representing the frequencies of each different/unique string. The list returned contains as many frequencies as there are unique strings. The returned frequencies represent an

alphabetic/lexicographic ordering of the unique words, so the first frequency is how many times the alphabetically first word occurs and the last frequency is the number of times the alphabetically last word occurs.

Consider these strings (quotes for clarity, they're not part of the strings).

```
["apple", "pear", "cherry", "apple", "cherry", "pear", "apple", "banana"]
```

The list returned is `[3, 1, 2, 2]` since the alphabetically first word is "apple" which occurs 3 times; the second word alphabetically is "banana" which occurs once, and the other words each occur twice.

Specification

```
filename: SortedFreqs.py
```

```
def freqs(data):  
    """  
    return list of int values corresponding  
    to frequencies of strings in data, a list  
    of strings  
    """
```

What's the best way to ...

- **SortedFreqs**
 - <https://www2.cs.duke.edu/csed/pythonapt/sortedfreqs.html>
- **Count how many times each string occurs**
 - Create `d = {}`, iterate over list updating values
 - Use `data.count(w)` for each `w`

APT: SortByFreqs

APT SortByFreqs

Problem Statement

The frequency with which data occurs is sometimes an important statistic. In this problem you are given a list of strings and must determine how frequently the strings occur. Return a list of strings that is sorted (ordered) by frequency. The first element of the returned list is the most frequently occurring string, the last element is the least frequently occurring. Ties are broken by listing strings in lexicographic/alphabetical order. The returned list contains one occurrence of each unique string from the list parameter.

Consider these strings (quotes for clarity, they're not part of the strings).

```
["apple", "pear", "cherry", "apple", "pear", "apple", "banana"]
```

The list returned is:

```
[ "apple", "pear", "banana", "cherry" ]
```

since the most frequently occurring string is "apple" which occurs 3 times; the string "pear" occurs twice and the other strings each occur once so they are returned in alphabetical order.

Specification

```
filename: SortByFreqs.py

def sort(data):
    """
    return list of strings based on
    the list of strings in parameter data
    """
```

Wait, wait, but what's ...

- **SortByFreqs**
 - <https://www2.cs.duke.edu/csed/pythonapt/sortbyfreqs.html>
- **Sort by # occurrences high to low**
 - Tuples with count/lambda and reverse=True?
 - Break ties in alphabetical order: two passes

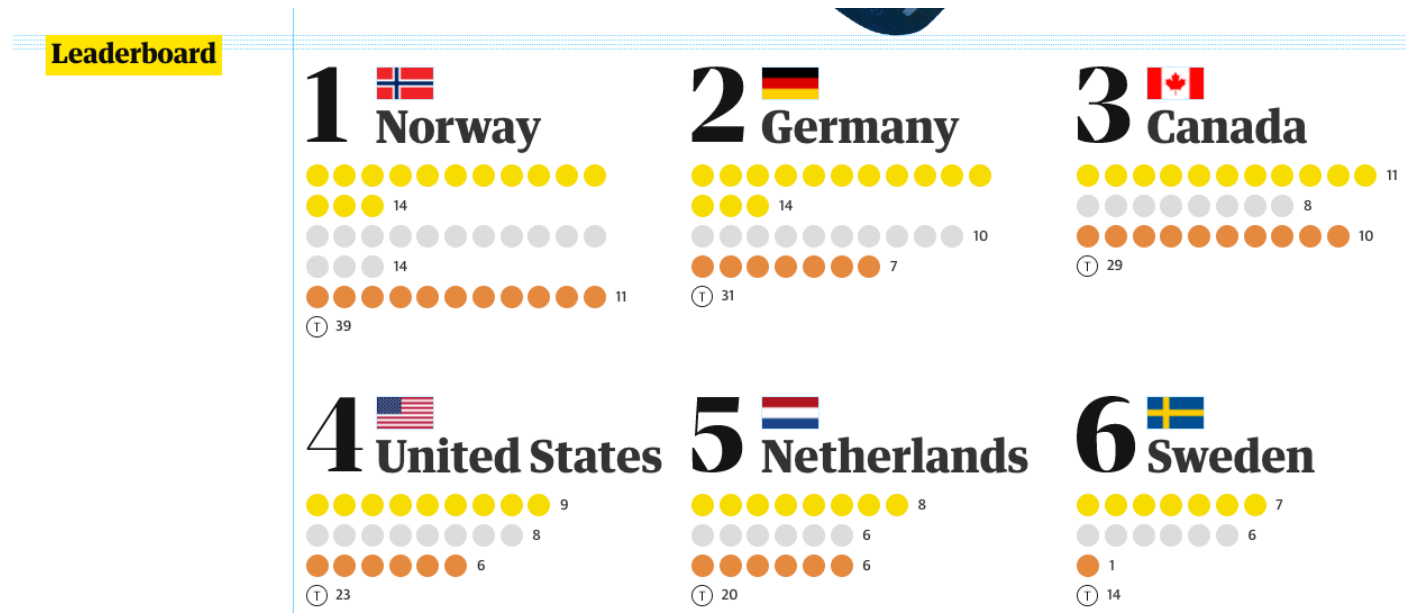
WOTO-2

<http://bit.ly/101s21-0408-2>



APT: MedalTable

- <http://bit.ly/apt-medal-table>



APT: MedalTable

Problem Statement

The Olympic Games will be held, and have been held (and might be being held). Given the results of the olympic disciplines, generate and return the medal table.

The results of the disciplines are given as a `String list` `results`, where each element is in the format "GGG SSS BBB". GGG, SSS and BBB are the 3-letter country codes (three capital letters from 'A' to 'Z') of the countries winning the gold, silver and bronze medal, respectively.

The medal table is a `String list` with an element for each country appearing in results. Each element has to be in the format "CCO G S B" (quotes for clarity), where G, S and B are the number of gold, silver and bronze medals won by country CCO, e.g. "AUT 1 4 1". The numbers should not have any extra leading zeros.

Sort the elements by the number of gold medals won in decreasing order. If several countries are tied, sort the tied countries by the number of silver medals won in decreasing order. If some countries are still tied, sort the tied countries by the number of bronze medals won in decreasing order. If a tie still remains, sort the tied countries by their 3-letter code in ascending alphabetical order.

Specification

```
filename: MedalTable.py

def generate(results):
    """
    return list of strings
    based on data in results, a list of strings
    """

    # you write code here
    return []
```

```
1. ["ITA JPN AUS", "KOR TPE UKR", "KOR KOR GBR", "KOR CHN TPE"]
```

Returns:

```
[ "KOR 3 1 0", "ITA 1 0 0", "TPE 0 1 1", "CHN 0 1 0", "JPN 0 1 0",
  "AUS 0 0 1", "GBR 0 0 1", "UKR 0 0 1"
]
```

Tracking the Data

- **What do we need to obtain for each country?**
 - What's the data, how do we store it?
 - What's the data, how do we calculate it?
- **Method and code to transform input**
 - What will we store, how do we initialize/update
 - Verifying we've done this properly

Use a dictionary?

- **3 dictionaries**
 - Country to Gold count
 - Country to Silver count
 - Country to Bronze count

Example: dictionary d

- Process first string: **"KOR TPE UKR"**
- Process second string: **"KOR KOR TPE"**

Sorting the Data

- Use dictionary to get list of tuples

```
[ ('JPN' , [0 , 1 , 0]) , ('KOR' , [3 , 1 , 0]) ,  
  ('TPE' , [0 , 1 , 1]) , ('UKR' , [0 , 0 , 1]) ]
```

- Sort by what first?
- Sort by what second?
- Sort by what third?
- Etc.
- Any of those sorts by reverse order?
- After sorting, convert to strings to return
- Print to verify, two passes like SortByFreqs

Sorting dictionary in MedalTable

- Write helper function to create that dictionary
- Sort by [gold, silver, bronze]. Just sort?
 - Tuples and lists sorted in index order