Reminders

- APT Quiz 2
  - 11/12 (8am) - 11/15 (11pm)
Key instructions

• Input ✔
• Output ✔
• Assignments* ✔
• Math/Logic ✔
• Conditionals ✔
• Repetition ✔

*not listed in book
Python Data Types

- int, float, bool ✔
- Collections
  - Strings ✔
  - Lists ✔
  - Tuples ✔
  - Sets ✔
  - Dictionaries ✔
PFTD

• Clever Hangman –
  • How does it work? Greedy Algorithm

• Modules
KISS Principle

• Think of the non-computing context for any word/terms
• KISS model
  • Work smarter, not harder!!
• “Good programmers are simply good designers.”
  • -Dr. Washington
• Design first and always!
• Importance of reusability
• USE PyCharm/PythonTutor IF YOU HAVE QUESTIONS!
People to Know: Matthew Yazzie

- BS, MS (Stanford)
- Organizational and behavioral scientist
- Strategic Director/Facilitator
  - Collective-A DEI Lab
- Navajo Nation
Sometimes there will be letters

- The letter “u” has been guessed and is the 2nd letter
  Ex: _ u _ _ _ _ and user guesses ‘r’

- ["ruddy", "rummy", "rungs", ... "rusty"]
  - 5 words start with “ru” and no other “r” or “u”
- ["burch", "burly", "burns", ... "turns"]
  - 17 words only ‘u’ as second letter and only ‘r’ third letter
- ["bucks", "bucky", ... "tufts"]
  - 98 words with only “u” second letter and no ‘r’
- What should our secret word be? "ruddy", "burch" or "bucks"?
More Details on Game

• Pick 8-letter word at random: *catalyst*
  • User guesses 'a', what should computer do?
  • Print _ a _ a _ _ _ _ and continue?

• Look at all groups of words and decide on a new word that is more likely to stump player
  • Why “designed” better choice than “tradeoff”?
  • 3,475 words with no 'a', 498 with 'a' 3rd letter
Creating Groups/Categories

• For each of 7,070 words (8 letters), given word and ‘a’, find its group, represented by a template
• Use dictionary
  • Template is KEY, the VALUE is a list of matching words
• Choose biggest list
• Repeat
• # words smaller over time

<table>
<thead>
<tr>
<th>Group/Template</th>
<th>Size of Group</th>
</tr>
</thead>
<tbody>
<tr>
<td>_ a _ _ _ _ _ _</td>
<td>587</td>
</tr>
<tr>
<td>_ a _ a _ _ _ _</td>
<td>63</td>
</tr>
<tr>
<td>_ _ a _ _ _ _ _</td>
<td>498</td>
</tr>
<tr>
<td>_ _ _ a _ _ _ _</td>
<td>406</td>
</tr>
<tr>
<td>_ _ _ _ _ _ _ _</td>
<td>3,475</td>
</tr>
</tbody>
</table>
Changes to Regular Hangman

• List of words from which secret word chosen
  • Initially this is all words of specified length
    • User will specify the length of the word to guess
  • After each guess, word list is a new subset

• Keep some functions, modify some, write new ones

• Changes go in another function to minimize changes to working program
  • Minimizing changes helps minimize introducing bugs into a working program
Details from Assignment

• We've missed four times, what's happening?
  • "belted" is one of 20 words that fit guesses

```
letters not yet guessed: bc fgh jklmn pq t vwxyz
misses remaining = 2
_ e _ _ e d
(word is belted)
# possible words: 20
letter> l
_e__ed : 10
_el_ed : 4
_elled : 5
le__ed : 1
# keys = 4
you missed: l not in word
```
Greedy Algorithms

• “Choosing largest group” -> *greedy algorithm*
  • Make a locally optimal decision that works in the long run
  • Choose largest group to make game last …

• Greed as in “it chooses the best current choice every time, which results in getting the best overall result”

• Canonical example? Change with coins
  • Minimize # coins given for change: 57 cents
Making change for 57 cents

- When choose next coin, always pick biggest
- With half-dollar coins

- With quarters and no half dollars
When greedy doesn't work

- What if no nickels? Making change for 31 cents:

- Can we do better? Yes!
Activity 1:

Hangman Words

Want to frustrate your friends? Use these techniques to pick a good hangman word or just pick from the list of words that have proved to be the most challenging to guess.

Hard Hangman Words:
- abruptly
- askew
- azure
- bayou
- blizzard
- absurd
- avenue
- bagpipes
- beekeeper
- boggle
- abyss
- awkward
- bandwagon
- bikini
- bookworm
- affix
- axiom
- banjo
- blitz
- boxcar
Clever vs Plain Hangman

• Minor changes, though they require coding
  • Regular: show ‘a e t w’
  • Clever: show ‘bcd fghijklmnopqrs uv xyz’
  • User inputs added – debug mode, length of word
  • processUserGuessClever

• Major changes
  • Debug mode
  • List of potential words changes at each turn
  • Function getNewWordList and createTemplate
Testing your code

• Alternative to lowerwords.txt? This is large!
  • Create your own file of words. Small file
  • Facilitates testing

• Call random.seed(…)
  • If seeded with same number
  • Same words/order every time you play
  • Reproduce errors more easily
Testing your methods

• CheckMyFunctions.py
  • Can you add anything to check/test things?

• Testing `getNewWordlist(guess, letter, words)`
  • From watching the debug game play?
  • Better: Test in isolation from game

• `getNewWordList` calls `createTemplate(template, word, letter)`
  • How we test one without the other?
  • Test `createTemplate` function first and separately
Edge Case

• Words left: ['trim', 'trio']
• Hangman template is: ‘tri_’
• Users guesses ‘m’
  • What should the secret word be? ‘trio’!
  • But the dictionary has a tie!
    • ‘tri_’ : ['trio'] # length 1
    • ‘trim’ : ['trim'] # length 1
• getNewWordList should take this into account
  • Pick the template with most ‘_’
Activity 2:
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
Reminders

• Work smarter, not harder
• Design first
• Try to identify where you are stuck
  • Identify resources to help solve problem
• Leverage your design and PythonTutor to understand program flow of control
  • [http://pythontutor.com](http://pythontutor.com)