Outline for October 31, 2023

1. Dictionary
   (a) Collection of key-value pairs

2. Creating dictionaries
   (a) Using \( d = {} \)
   (b) Using \( d = \text{dict()} \)

3. Methods for dictionaries
   (a) \( \text{k in D}: \) True if dictionary \( D \) has key \( k \); else False
   (b) \( D\text{.keys()}: \) list of keys in \( D \)
   (c) \( D\text{.values()}: \) list of values in \( D \)
   (d) \( D\text{.items()}: \) list of tuples (key, value) in \( D \)
   (e) \( D\text{.get}(k, d): \) if key \( k \) in \( D \), return associated value; else return \( d \)
   (f) \( \text{del D}[k]: \) delete tuple with key \( k \) from \( D \)
   (g) \( D\text{.clear()}: \) delete all entries in \( D \)

4. Example: memos
   (a) Remember how slowly the recursive Fibonacci number program \textit{rfib.py} ran? Here is a faster recursive version that uses memos \textit{[rfibmemo.py]}

5. Sorting the dictionary
   (a) \text{sorted} sorts based on keys

6. Example: word frequency count
   (a) Unsorted \textit{[wfc-1.py]}
   (b) Sorted alphabetically \textit{[wfc-2.py]}
   (c) Sorted alphabetically, but dictionary order \textit{(note key=\text{str.lower}() in sorted [wfc-3.py]}
   (d) Sorted by frequency \textit{(treat \text{lambda } x: x[1] as an idiom to reference the value of the dictionary entry, not the key—to go from highest to lowest, replace x[1] with -x[1])\textit{[wfc-4.py]}
   (e) Sorted by frequency first, then alphabetically—note use of function \textit{alphafreq(x); you can use any function here, and the parameter is the item [wfc-5.py]}

7. Handling exceptions
   (a) \text{except} \textit{[except0.py]}
   (b) \text{except exceptcode} \textit{[except1.py]}
   (c) else \textit{[except2.py]}
   (d) except exceptcode as msgvar \textit{[except3.py]}
   (e) finally \textit{[except4.py]}
   (f) Exceptions in a function: who handles them? \textit{[except5.py, except6.py]}
   (g) Using global variables as error flags \textit{[except7.py]}
   (h) \text{raise exceptcode message} \textit{[except8.py]}

8. Writing a program to play rock-paper-scissors: top-down design
   (a) Problem statement and general algorithm idea
(b) Data representation and program structure [rps-1.py]
(c) Figure out who wins [rps-2.py]
(d) Get computer choice [rps-3.py]
(e) Get user input [rps-4.py]
(f) Make it user-friendly [rps-5.py]