First Topic for Today

- Ising simulation
- One final Python data type: dictionaries

Exam 2: Wednesday, March 26, 7pm
You can use a calculator

Reading
- Zelle, Chapter 11.6
- Python in a Nutshell, pp 59-61

Plots for Ising Simulation

Consider \( N = 20 \) and 40, vary total energy;
Description limits number of calls made to ising_model

- Visualize grid (Vpython) continuously
- Visualize demon energies after one call to ising_model (make plot inside ising_model)
- 3-D plot on Magnetization vs. Energy & SimulationSteps - one call to ising_model generates one “curve”
- Magnetization versus energy – plot shows three curves
Data types you have used already

- Numbers
- Strings
- Lists
- Arrays

Covered in Python texts/manuals, but not yet used

- **Dictionaries**

Dictionaries

- non-sequential data collection
- very flexible built-in data type
  - not provided as a built-in by all languages
- List = ordered collection of objects
  Dictionary = unordered collection of objects
- A dictionary
  - stores a collection of entries
  - retrieves an entry based on a key-value (not based on the position/index)
  - every key has associated values
Dictionaries: use and operations

D1 =
{'Korb':46184,'Hambrusch':41831,'Vitek':69644}
>>> D1
{'Vitek': 69644, 'Hambrusch': 41831, 'Korb': 46184}
>>> D1['Haugan']=45504
>>> D1['Lint'] = 'no listing'

>>> D1.keys()
['Haugan', 'Lint', 'Korb', 'Hambrusch', 'Vitek']
>>> D1.values()
[45504, 'no listing', 46184, 41831, 69644]

>>> D1.items()
[('Haugan', 45504), ('Lint', 'no listing'), ('Korb', 46184),
('Hambrusch', 41831), ('Vitek', 69644)]

Dictionaries: use and operations

>>> D1
{'Hosking': 46001, 'Haugan': 45504, 'Lint': 'no listing', 'Korb':
46184, 'Hambrusch': 41831, 'Vitek': 69644}
>>> if 'Korb' in D1:
    print D1['Korb']
46184
>>> if D1['Lint'] == 'no listing':
    del D1['Lint']
>>> D1['Korb'] = [46184, 46003]

>>> D1
{'Vitek': 69644, 'Haugan': 45504, 'Hambrusch': 41831, 'Korb':
[46184, 46003]}
```python
>>> D1 = {'Korb':46184, 'Hambrusch':41831, 'Vitek':69644}
>>> D1['Korb'] = [46184, 46003, 413-5667]

Which does not return 3?

A. `len(D1['Korb'])`
B. `len(D1)`
C. `3*len(D1['Vitek'])`
D. `len(D1.items())`
E. `count = 0
  for k in D1: count +=1
  return count`

Matrix = {}
Matrix[(2,3,8)] = 32
Matrix[(2,1,3)] = 16
Matrix[(1,6,3)] = 8

Matrix[1,1,10] = Matrix[(1,6,3)] + Matrix[(2,1,3)]
print Matrix
print Matrix.values()
```

What can be printed?

A. `{(1, 6, 3): 8, (2, 1, 3): 16, (2, 3, 8): 32, (1, 1, 10): 24}
   [8, 16, 32, 24]`
B. `{(1, 1, 10): 24, (1, 6, 3): 8, (2, 1, 3): 16, (2, 3, 8): 32}
   [8, 16, 32, 24]`
C. `[(1, 6, 3): 8, (2, 1, 3): 16, (2, 3, 8): 32, (1, 1, 10): 24]
   [8, 16, 32, 24]`

Dictionaries: main properties

- Access to an entry is by key
- Unordered collection stored in a randomized order
- Dictionaries can grow and shrink
- Dictionaries are mutable (just like lists)
- Concatenation and slicing don’t apply
- Searching is easy
- We can also sort the keys and the values;
  - can also define our own comparisons
- Only use when lists and arrays don’t apply

```
>>> D1
{'Vitek': 69644, 'Haugan': 45504, 'Hambrusch': 41831, 'Korb': [46184, 46003]}

names = D1.keys()
names.sort()

numbers = D1.values()
numbers.sort()

elements = D1.items()
elements.sort()
```
Word Frequency Problem
(Zelle 11.6.3)

- Given a file containing text, count the number of occurrences of each word in the text
- Output a list of words, in sorted order, along with their frequencies
- Dictionaries make this type of problem easy
- Dictionary has words as keys and a counter as value:
  \{
    \{'target': 3\},
    \{'the': 5\},
    \ldots
  \}
- For each word in the file decide if it is in the dictionary
  - If yes, increment the entry’s counter
  - If not, create a new entry

Equivalent statements

```
if counts.has_key(w):
    counts[w] = counts[w] + 1
else:
    counts[w] = 1
```

"Pythonic"

```
    counts[w] = counts.get(w, 0) + 1
```

Returns count[w] if w exists as key in dictionary counts, 0 otherwise
def compareItems((w1,c1), (w2,c2)):
    if c1 > c2:
        return -1
    elif c1 == c2:
        return cmp(w1, w2)
    else:
        return 1

def main():
    print "This program analyzes word frequency in a file"
    print "and prints a report on the n most frequent words."
    # get the sequence of words from the file
    fname = raw_input("File to analyze: ")
    text = open(fname,'r').read()
    text = string.lower(text)
    for ch in """""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
       text = string.replace(text, ch, "")
    words = string.split(text)
    counts = {}
    for w in words:
        counts[w] = counts.get(w,0) + 1
    # output analysis of n most frequent words.
    n = input("Output analysis of how many words? ")
    items = counts.items()
    items.sort(compareItems)
    for i in range(n):
        print "%-12s%5d" % items[i]