USES FOR DICTIONARIES
Three (Main) Uses

1. Translation (encoding/decoding)
2. Accumulators (counting/summing)
3. Merging (joining/linking)
Dictionaries to Encode

• We can use dictionaries to encode/decode data
• We can use dictionaries to translate from one representation to another
d = {}
for a, b in zip('DOG', 'CAT'):
    d[a] = b
a = []
for k in "OGD":
    a.append(d[k])
s = 'Q'.join(a)

a) 'AQTQC'  c) 'QOQGQDQ'
b) 'QOGDQ'  d) 'QAQTQCQ'
Exercise

• Encode all of the words in a file with the Caesar cypher
• Decode all of the words in the file
def encypher(word):
    word=word.upper()
    x="ABCDEFGHIJKLMNOPQRSTUVWXYZ"
    y=x[1:]+x[0]
    e=
    for a,b in zip(x,y):
        e[a]=b
    encoded=""
    for c in word:
        if c in encoded:
            encoded+=e[c]
        else:
            encoded+=c
    return encoded

for line in open("words.txt"):
    line=line.strip()
    print(encypher(line))
Dictionaries as Accumulators

• We can use dictionaries as a collection of counters for many things at once

```python
x = "UUDDLRLRBABA"
d = {}
for c in x:
    if c not in d:
        d[c] = 1
    else:
        d[c] += 1
```
s = "ABACAB"
d = {}
for a, b in enumerate(s):
    if b not in d:
        d[b] = 1
    else:
        d[b] += a

b) d = {"A": 3, "B": 2, "C": 1}
c) d = {"A": 6, "B": 6, "C": 3}
d) d = {"A": 0, "B": 0, "C": 0}
What is the final value of x?

a) 0
b) 1
c) 2
d) 3
s = "ABACAB"

d = {}

for a, b in enumerate(s):
    if b not in d:
        d[b] = a
    else:
        d[b] += a
Exercise

• Count category frequencies in Jeopardy questions
• Count bigram frequencies in Jeopardy clues
Counting bigrams

counter={}
for word in open("words.txt"):  
    word=word.strip().upper()
    for i in range(len(word)-1):
        bi=word[i:i+2]
        if bi not in counter:
            counter[bi]=1
        else:
            counter[bi]+=1
Dictionaries to Join/Merge Data

• We can link data based on a common field

```python
text = {"Bill":60644,
        "Jim":41073,"Beth":63103}
city = {60644:"Chicago",
        41073:"Cincinnati",
        63103:"St. Louis"}
for name in zip:
    print(name,city[zip[name]])
```
d1={3:'Apple',2:'Banana',4:'Carrot'}
d2={'Apple':'A','Banana':'B','Carrot':'C'}

s=''
for k1 in sorted(d1):
    k2=d1[k1]
    s+=d2[k2]

a) "ABC"    c) "BAC"
b) "BCA"      d) "CBA"
Modules

• A collection of Python specialized functions, variables, and types

• We need to `import` the module

```python
import math
```

• Can then access things within the module using `attribute operator`

```python
math.sqrt(math.pi)
```
From

• Can choose what to import with `from`
  
  ```python
  from cmath import phase
  phase(1+1j)
  ```

• We don’t have to type the module name all the time

• Import multiple items with a comma
  
  ```python
  from cmath import phase, rect
  ```
What should replace the `???
math.degrees(2*pi)`

a) `from math import pi
   import math`
b) `from math import pi, degrees`
c) `import pi
   import math`
d) `import math`
exp(pi) – pi

What should replace the ???

a) from math import pi
   import math

b) from math import pi, exp

c) import pi, exp
   import math

d) import math
Useful Python Modules

- math, cmath
- random
- csv
- sys, os
- time, datetime
- itertools
- logging

- NumPy
- SciPy
- matplotlib
Writing readable code

• We should always strive to write code that is easy to read.
  – Our variables should have descriptive names.
  – We should also annotate our code.

• **REMEMBER**: A program is set of instructions a computer executes to achieve a goal.
Commenting

- **Comments** are text that the interpreter ignores
- Comments help *a person* read a program
- The # symbol indicates a comment
  - Anything after that symbol is ignored

# Hello, I am a comment
Docstring

- A string literal that behaves like a comment
- Use triple quotes
- Especially useful after function definition

"""Hello, I am a docstring."""
What is the final value of `x`?

a) ABCD
b) ABCD1
c) ABCD12
d) ABCD123
Why comment/document?

- Allows us to *explain* our code to others.
- But mostly… to ourselves.
- Yes, *ourselves*. 
Documenting Modules

• Every script (.py) file you write is a module.
• Your modules should have a docstring at the beginning describing them and you.

"""
CS101 class demonstration
Author: Ryan Cunningham
"""
Documenting Functions

• Use doc string and describe what function does.
• Describe all parameters by name.
• Describe all return values.

def sqrt(n):
    """
    Computes square root of a number.
    n: an integer or float
    returns: the square root of n
    """
    return n**.5
category_counts={} # accumulator for counting categories
for line in open("jeopardy.txt"):
    if line[0]!="#": # ignore comment lines
        split=line.split("\t") # split on tab
        category=split[2] # category is 3rd column
        # increment count of this category
        if category not in category_counts:
            category_counts[category]=0
        category_counts[category]+=1

# sort category counts for display
category_tuples=[]
for c in category_counts:
    n=category_counts[c]
    category_tuples.append((n,c))
category_tuples.sort() # tuples are sorted by first item
print category_tuples
MATPLOTLIB MODULE
Matplotlib

- Plotting library (module) for Python
- Not included in standard Python libraries
- Pylab interface imitates Matlab

```python
import matplotlib.pyplot as plt
x=[0,1,2]
y=[1,2,3]
plt.plot(x,y)
plt.show()
```
Title and Axis Labels

• Should always label axes and title our graphs

```python
plt.title("Example plot")
plt.xlabel("X data")
plt.ylabel("Y data")
```
Plot

• Function takes at least two arguments
  – List of x and y coordinates
• Can optionally take a string indicating the color/shape of the line
  
  ```python
  plt.plot(x, y, 'r. ')
  ```
• Can also take a `label` keyword argument
  
  ```python
  plt.plot(x, y, 'r.', label=“Fun!”)
  ```
Plot

• Can plot multiple lines at once

```python
x = [1, 2, 3, 4]
y1 = [2, 4, 6, 8]
y2 = [3, 6, 9, 12]
plt.plot(x, y1, 'r-', label="2x")
plt.plot(x, y2, 'g--', label="3x")
```
Legend

- With multiple lines, a legend is helpful

plt.legend()
Bar charts

• Used to show amounts associated with a set of values

```python
a=[1,2,3]
b=[3,4,4]
ticks=['Donna','Jo','Sandy']
plt.xticks(a,ticks)
plt.bar(a,b,label='Bar')
```
Which of these will correctly display a plot?

a) import matplotlib.pyplot as plot
   plot.plt([[1,2,3],[4,5,6]])

b) import matplotlib.pyplot as plt
   plt.plot([[1,2,3],[4,5,6]])

c) from matplotlib import pyplot
   pyplot.plt([[1,2,3],[4,5,6]])

d) None of the other answers
```python
x = []
y = []
ticks = []
for year in homers:
    hr = homers[year]
x.append(year)
y.append(hr)
ticks.append(str(year))

plt.xticks(x, ticks)
plt.bar(x, y)
plt.title("Year vs. HR in MLB")
plt.show()
```
Histograms

• Used to show *distribution* of values
  
  counts = [1, 1, 2, 3, 1, 2, 1, 3, 1, 2, 1]
  
  `plt.hist(counts)`

• Bins the values and counts *frequency* of values that range
Exercise

- Plot distribution of word use in Jeopardy questions
counts = {}
for line in open("jeopardy.txt"):  
    if line[0] != ":":
        split = line.split("\t")
        air_date, answer, category, question,
        rnd, show_number, value = split
        value = value.replace("","").strip()
        if value == ":":
            continue
        words = question.lower().split()
        for word in words:
            if word not in counts:
                counts[word] = 0
            counts[word] += 1
x = []
for word in counts:
    x.append(counts[word])

x.sort()
x = x[-100:]  # plot only the top 100 counts

import matplotlib.pyplot as plt
plt.hist(x)
plt.xlabel("Number of times used")
plt.ylabel("Number of words")
plt.show()
Scatter Plot

- Scatter plots graph points in 2D

```python
plt.scatter(x, y,
            label="Data",
            color='g',
            marker='*')
plt.show()
```
Exercise

• Plot the year vs. RBI for Pete Rose
import csv
import matplotlib.pyplot as plt

f = open("./lahman-csv_2015-01-24/Batting.csv")
years = []
rbis = []
for r in csv.DictReader(f):
    if r["playerID"] == "rosepe01":
        y = int(r["yearID"])
        rbi = int(r["RBI"])
        years.append(y)
        rbis.append(rbi)
f.close()

plt.scatter(years, rbis)
plt.show()