COMP 200 & 130 exam 2, Spring 2012

Instructions:

- **Allowed resources:** You can read your notes and assignments, everything posted on the course web site and OWL-Space area, everything directly linked from the course web site and OWL-Space area. You can use a computer for reading and writing the exam, using the Python interpreter and any editor, using a browser for the previously mentioned materials, and using a calculator.
- **Files:**
  - **`exam2.py`**: COMP 200 and COMP 130 students should edit this for problems 1-6.
  - **`exam2_130.py`**: COMP 130 students should edit this for problems 7-8.
  - This text file: Use for instructions only. Do not edit.
- **Time limit:**
  - COMP 200 students are allowed **three and a half contiguous hours**.
  - COMP 130 students are allowed **five contiguous hours**.

  The textual length of the problems may seem long, but that's just because we have attempted to be very clear on what each problem is asking.

- **Asking questions:** Times will be posted online when the instructors will be available for questions. You can take the exam at any time before the due time, but you will likely not get quick responses to any questions during any non-posted time.

1. **COMP 200 & COMP 130** (12 points)

   When writing a program, you need to pick a data structure. Either a list or tuple could be appropriate for your problem. Briefly describe how you would choose between the two options.

2. **COMP 200 & COMP 130** (13 points)

   If you create a Markov chain from a short input text, "riffs" generated from it tend to be very similar to the original text. However, if you create a Markov chain from a long input text, riffs are less similar. Explain why.

3. **COMP 200 & COMP 130** (20 points total)

   Assume the Rice directory is stored as a dictionary mapping a building name to the directory for a single building. In turn, a building directory is a dictionary mapping a room to a single name. For the sake of simplicity, we are assuming each office or college room is occupied by a single person. However, this does allow for someone to have multiple addresses, for example, a college master has both a residential and work address on campus.

   For example,

```
directory = {"Duncan Hall" : {3093 : "John Greiner",
                              3102 : "Stephen Wong",
                              3114 : "Joe Warren",
                              3080 : "Joe Warren"},
             "Keck Hall" : {129 : "Rob Griffin",
                            109 : "Ann Saterbak"},
             "Hanszen College" : {"House" : "Rob Griffin"}}
```

   a. (10 points) Write a function **`lookupBuilding(buildingDir,name)`** that takes a building directory and returns a list of the rooms for that name.

   b. (10 points) Write a function **`lookup(dir,name)`** that takes a directory and a name. It returns a list of the building-room pairs for that name.

   Examples:

   - **`lookupBuilding(directory["Duncan Hall"], "Joe Warren")`** returns **`[3114,3080]`**.
   - **`lookupBuilding(directory["Keck Hall"], "Rob Griffin")`** returns **`[129]`**.

- **`lookup(directory, "Joe Warren")`** returns **`[("Duncan Hall",3114), ("Duncan Hall",3080)]`**.
- **`lookup(directory, "Rob Griffin")`** returns **`[("Keck Hall",129), ("Hanszen College","House")]".`**

4. **COMP 200 & COMP 130** (20 points)

Background: The Python function **`re.findall(regexp,str)`** returns a list of all **non-overlapping** occurrences of the pattern in the string. For example, **`re.findall("a[0-9]a","a1a2a3a4a")`** returns **`["a1a","a3a"]`**. This list does not include **`"a2a"`** because it overlaps **`"a1a"`**, and it doesn't include **`"a4a"`** because it overlaps **`"a3a"`**.

Write a function **`findallOverlapping(regexp,str)`** that returns a list of all **possibly-overlapping** occurrences of the pattern in the string.

Examples:

- **`findallOverlapping("a[0-9]a","a1a2a3a4a")`** returns **`["a1a", "a2a" , "a3a", "a4a"]`**.
- **`findallOverlapping("a*","aaaabaab")`** returns **`["aaaa","aaa","aa","a","","aa","a",""]`**, because it finds occurrences like this:

  input text: **`aaaabaab`**
  occurrences: **`aaaa`**
  `aaa`
  `aa`
  `a`
  `_` Meaning the empty string.
  `aa`
  `a`
  `_` Meaning the empty string.
- **`findallOverlapping("a+b","aaaabaab")`** returns **`["aaaab","aaab","aab","ab","aab","ab"]`**, because it finds occurrences like this:

  input text: **`aaaabaab`**
  occurrences: **`aaaab`**
  `aaab`
  `aab`
  `ab`
  `aab`
  `ab`

Hints:

- Do not do anything with the regular expression other than pass it to one or more functions in the **`re`** library.
- Functions in **`re`** return a **`MatchObject`**, **`m`**, when they find a matching substring. Using **`m.group()`** (Note: no arguments!) then returns the matching substring. E.g.,

```
m = re.somefunction(regexp,somestring)
if m:
    matchedString = m.group()
```

Resource link: The **`re`** library, including specifically **`MatchObject.group()`**.

5. **COMP 200 & COMP 130** (15 points)

Background: When you load a web page, sometimes the network can be very slow. To speed up future access, the browser *caches* the web page, so that it can simply load it from your computer disk drive, rather than loading it from the network again. I.e., it stores the idea that the requested web address maps to this web page. This approach only works with web pages that don't change.

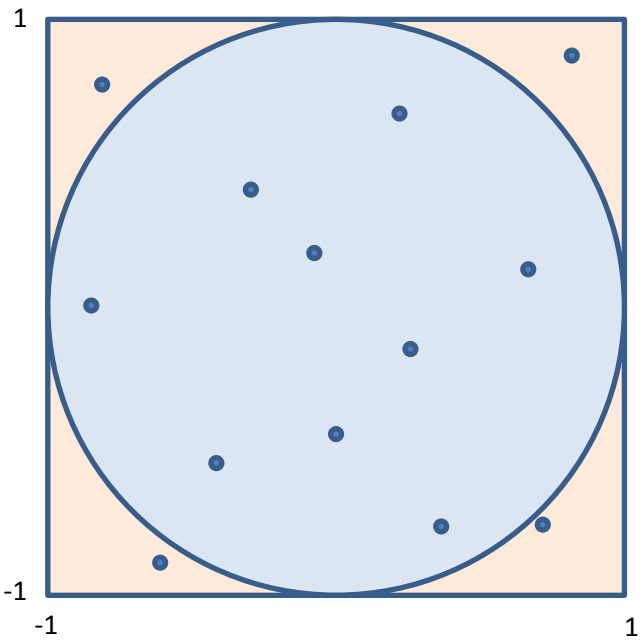
In place of loading a web page, we have provided a function **`f(x)`** that returns some value. We will simply assume that this function is slow. What this function calculates is irrelevant to the problem (although it is known as the Fibonacci function, with a time delay added). We want to speed up using this function by caching its results. The idea is that any time we use the function, we will store the result, so that the next time we use it with the same argument, we will just look up the result, rather than re-calculating it.

Write a function **`fCached(x)`**. You define and maintain a global dictionary that represents the cache. When your function is called, it looks whether the **`f`**'s result for **`x`** has been cached yet. If so, it just returns the result from the cache. Otherwise, it uses **`f(x)`** to calculate the result, stores in the cache that **`x`** maps to this result, and returns the result.

Examples: **`fCached(x)`** always returns the same thing as **`f(x)`**.

6. **COMP 200 & COMP 130** (20 points total)

Background: You might have celebrated Pi Day on 3/14. Historically, calculating the value of pi accurately has been a difficult problem. The following diagram illustrates one approach.



Area of the circle $= \pi \cdot r^2 = \pi \cdot 1^2 = \pi$.
Area of the rectangle $= h \cdot w = 2 \cdot 2 = 4$.
Ratio of circle area to rectangle area $= \frac{\pi}{4}$.

If we randomly generate points in the rectangle, some of them will also land in the circle. If we generate lots of points, we expect the fraction of those points that land in the circle to be approximately the ratio of the areas, i.e., $\frac{\pi}{4}$.

Hints: We can generate a random two-dimensional point in the rectangle by randomly generating two numbers in the range -1 to 1. We can determine if a point is in the circle by seeing whether it is within distance r from the origin, i.e., if $\sqrt{x^2 + y^2} \leq r$, or equivalently $x^2 + y^2 \leq r^2 = 1^2 = 1$.

Resource link: The **`random`** library. You might also want to use the **`math`** library, e.g., for **`math.pi`**.

a. (15 points) Write a function **`approximatePi(numberOfPoints)`** that takes a number of points that you will randomly generate. It generates the points and computes the fraction of those inside the circle, as described above. It returns 4 times that fraction, i.e., an approximation of π.

b. (5 points) Experimentally, how large must **`numberOfPoints`** be to get an approximation that is within 0.01% of the actual value of π? Answer in terms of factors

of 10 for **`numberOfPoints`**, i.e., does **`numberOfPoints`** need to be roughly 1, 10, 100, 1000, 10000, etc.?

Note that 0.01% accuracy implies a value between 3.1412785 and 3.141907, and thus at least the first four digits are correct. We have provided a function **`testApproximatePi(numberOfPoints)`** that will call your function and report your function's result and accuracy.

---

7. **COMP 130 only** (15 points)

   Explain the basic premise by which PCA is used to differentiate between texts of different authorship in terms of multidimensional clustering. Give a short, concise, yet complete answer.

8. **COMP 130 only** (35 points)

   Refer to the provided file **`exam2_130.py`**. It contains code and the directions for four sub-problems.