Computer Science 134C
Introduction to Computer Science, in Python
Lecture #11 (Files)
October 1, MMXVIII

Keywords
csv, fibo, file, global, mode,
memoization,print,recursion

We consider files in more detail.

1. This week: A focus on debugging. Next week, reading period.
2. Midterm, evening of October 16. Closed book, open mind.
3. Questions?
4. From before: lists and tuples are similar, save mutability. Parallel assignment.
5. From before: dictionaries and sets.
6. Dealing with expensive, deterministic function calls.
   (a) Consider this beautiful function:

```
def fibo(n):
    """Nth fibonacci number: f(0) = 0, f(1) = 1, f(n)=f(n-1)+f(n-2)."""
    if n < 2:
        return n
    else:
        return fibo(n-1) + fibo(n-2)
```

   How can we make this run faster?
   (b) Idea: Never call the function more than once with the same arguments. The notion of *memoization*.
7. The *file*: a persistant store between python sessions.
   (a) Old files are opened, read, closed:

```
f = open('tomsawyer.txt','r')
for line in f:
    ...
f.close()
```

   Here, `'r'` is the *mode*: open the text file (`'t'`, a default) for reading. You can open a file for writing (`'w'`), which creates (or if necessary, deletes the contents of) the file first. There are other modes, as well; see `pydoc3 open`.
   (b) It's hard to remember to close the file, so we can use the `with` statement which, for files, remembers to do this for us:

```
with open('huckfinn.txt','r') as f:
    for line in f:
        ...
# file is closed, implicitly
```

(c) To create files with new data, we open the file for write. Let's print a multiplication table in `table.dat`:

```
with open('table.dat','w') as out:
    for a in range(1,10+1):
        for b in range(1,10+1):
            print('{:4}'.format(a*b),end='',file=out)
        print(file=out)
# the out.close is implied, here
```

The result, in `table.dat`:

```
   1   2   3   4   5   6   7   8   9  10
   2   4   6   8  10  12  14  16  18  20
   3   6   9  12  15  18  21  24  27  30
   4   8  12  16  20  24  28  32  36  40
   5  10  15  20  25  30  35  40  45  50
   6  12  18  24  30  36  42  48  54  60
   7  14  21  28  35  42  49  56  63  70
   8  16  24  32  40  48  56  64  72  80
   9  18  27  36  45  54  63  72  81  90
  10  20  30  40  50  60  70  80  90 100
```

(d) The `csv` package allows you to read and write files of comma separated values. Here's an idiomatic approach:

```
import csv
with csv.reader(open('grades.csv')) as f:
    for row in f:
        # process each row, a list of strings found between commas in file
```

The package supports the *creation* of files, as well. See `pydoc3 csv`.

8. A bit more about `print`:

(a) Print typically writes to the `sys.stdout`. You can write to an file `out` (as above) with `print(...,file=out)`.

(b) Print typically adds a newline (`\n`) at the end of the print. You can override this with a string, `s`, with `print(...,end=s)`.

(c) Print will typically wait to show output until you've written a newline; it *buffers* the output. You can force print to flush this output with `print(...,flush=True)`.

9. Recall: Often, you don't need explicitly manipulate an input or output file. Instead, on unix, you can *redirect* input and output to and from a script from specific files.

(a) The unix `script <f` operator redirects file `f` to be input to `script`.

(b) The unix `script >f` operator redirects output of `script` to file `f`.

(c) In unix `script1 | script2` the output of `script1` is used as input as `script2`.

⋆