Outline

MCS 260 Lecture 16
Introduction to Computer Science
Jan Verschelde, 17 February 2016

organization of files manipulating files

files store data permanently

on disk, flash drive, or tape

Random access memory is volatile: all data vanish after a program terminates.

- Files offer permanent storage for data.
  We distinguish between
  1. human readable text files; and
  2. binary files, only readable to the computer.
- Access to files on disk drives is slower.
  The operating system uses buffers:
  1. reads more than one line or character at once;
  2. writes to file, only when the buffer is full.
- Think of files as tapes, sequentially organized:
  1. one needs to read all previous items first;
  2. inserting an item in file leads to a new copy.

files on Unix

work in Terminal on Mac OS X in SEL 2263

- Files are organized in directories. Every file has an absolute path name, `pwd` prints the working directory, with `cd` we change the current directory.

- Access permissions: `r`: read, `w`: write, `x`: execute, organized in three user groups:
  1. the owner of the file
  2. every one in the same group
  3. every one else

  For example: `-rwxr-x--x` (see via `ls -l`) means
  1. the owner can read, write and execute;
  2. members of the group can read and execute;
  3. every one else can only execute.

  `chmod 751 file` changes the file access permissions, into `-rwxr-x--x` (or `-111101001`).

symbolic links: the UNIX command `ln`

The need for symbolic links:

1. programs may require a fixed names for their files
2. data may have move to a different directory

→ need to ensure programs still find their files.

Examples: `ls -lt /usr` on a Mac OS X:

```
lrwxr-xr-x  ...  X11R6 -> X11
lrwxr-xr-x  ...  texbin -> ../Library/Tex
/Distributions/.DefaultTeX/Contents/Programs/i386
```

Syntax of the `ln` command:

```
  ln -s < source >  < name of link >
```

The `-s` means a *soft* link: we create just another name.
Example: `ln -s books books.txt`
creates the link `books.txt` to the file `books`.

a simple library administration

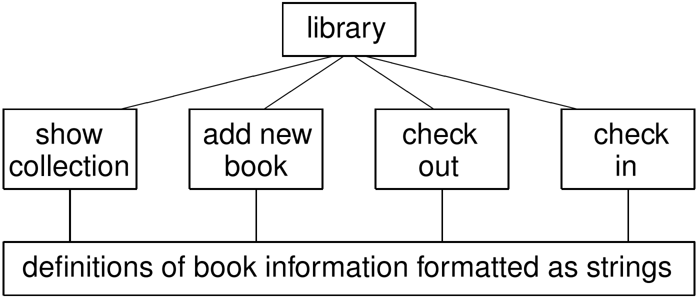
Executing library.py at the command prompt $:

```
$ python library.py
Welcome to our library!  Choose:
  0. leave the program
  1. show all books in the collection
  2. add a new book to the collection
  3. check out a book of the library
  4. return a book to the library
Type 0, 1, 2, 3, or 4 :
```

The collection of books is stored on file
→ save data about collection after program terminates.

functional-modular design

Straightforward top down functional design:



Supported by bottom up module with functions that define how we format the information of books as strings.

information about books: layout of data fields

The information stored for each book is

1. one bit: whether it is available or not;
2. a number: its key value in the collection;
3. a string: its title.

A file is just a sequence of lines.
Every line is a string.

We separate the data fields by colons:

```
1:1:Computer Science, an overview:
1:2:Python Programming in Context:
```

$\rightarrow$ content of file books (or books.txt on Windows)

Modules

components of software systems

A software system consists of

1. a collection of modules; and
2. the relations between the modules.

Modular design defines the decomposition of a system into modules.
A module can have modular components.

Each module has an **interface** and a **body**:

interface	is the set of all elements in a module available to all users of the module, also called the module's **exported resources**
body	is what realizes the functionalities of a module, also called the **implementation**.

A module *imports* resources from another module.
A module *exports* resources via its interface.

modules in Python

The syntax to import a module is

```
import < module >
```

For example: `import math`.
Then we can compute $\sqrt{2}$ via `math.sqrt(2)`.

If we only need one element of a module:

```
from < module > import < element >
```

For example: `from math import sqrt`.
Then we can compute $\sqrt{2}$ simply as `sqrt(2)`.

If `import math` is successful, then
`help(math)` shows information about the module `math`, and
`help(math.cos)` gives help on the function `math.cos`.

To get a quick overview of the contents, do `dir(math)`.

defining our own modules

Our program will have one module `bkform.py`.

The module `bkform`

- defines the formats of the data about books.
- exports routines to
    1. pack data tuple into a line to write to file,
    2. unpack a line read from file into a data tuple.

If we change the data format,
we need to change only one module in the program.

At the start of the file `library.py` we can state

```
from bkform import *
```

which implies that we import every single function.

organization of files
manipulating files

file objects: opening files to read, write, or append

Files are objects to store data permanently on disk.

To use a file, we must first open it. Syntax is

```
< object > = open ( < name > , < access mode > )
```

where

object is the name of the object

name is the name of the file on disk
could be the entire path name

access mode defines how we use the file:
`'r'`: read, `'w'`: write, `'a'`: append

Examples:

```
lib = open('books', 'r') # read from 'books'
lib = open('books', 'w') # write to 'books'
lib = open('books', 'a') # append to 'books'
```

reading from file: methods readline() and readlines()

As objects, files have methods we can use to read.

1. readline() reads the next line from file:

   ```
   >>> lib = open('books', 'r')
   >>> b = lib.readline()
   >>> b
   '1:1:Computer Science, an overview:\n'
   ```

   notice the end of line symbol

2. readlines() reads all lines of the file at once

   ```
   >>> lib = open('books', 'r')
   >>> collection = lib.readlines()
   >>> collection
   ['1:1:Computer Science, an overview:\n',\
    '1:2:Python Programming in Context:\n']
   ```

   readlines() returns a list of strings

end of file and close()

At the end of reading ...

1. When do we reach the end of a file?
   Suppose there are two lines in `books`:

   ```
   >>> lib = open('books', 'r')
   >>> b = lib.readline()
   >>> b = lib.readline()
   >>> lib.readline()
   ''
   ```

   readline() returns an empty string when at the end

2. Closing access to a file:

   ```
   >>> lib.close()
   ```

   it is good practice to use `close()` *when done* although Python will automatically close a file when reassigning the file object, in regards to writing ...

writing to file: methods write() and writelines()

Corresponding to `readline()` and `readlines()` are methods to write lines to file:

1. Suppose we want to append to a file:

   ```
   >>> lib = open('books', 'a')
   >>> lib.write('1:3:a new book:\n')
   ```

   no writeline() needed: add `\n` at end

2. Writing an entire file, e.g.: to copy files

   ```
   >>> lib = open('books', 'r')
   >>> L = lib.readlines()
   >>> newlib = open('backup', 'w')
   >>> newlib.writelines(L)
   ```

   often newlib.close() is needed to empty buffers

organization of files
manipulating files

the file books or books.txt

Recall the content of the file books:

```
1:1:Computer Science, an overview:
1:2:Python Programming in Context:
```

Important to know (by design):

- Every line on file represents one book.
- A line on file is a string of characters.
- For every book, we have 3 data fields:
  1. 1 or 0 for the availability of the book,
  2. the identification number of the book,
  3. the title of the book.
- Data fields are separated by colons `:`.

module bkform: definition of data formats for book

After `import bkform`, typing `help(bkform)` shows

```
FUNCTIONS
    get_key(book)
        Given a string with book information,
        returns the identification number of the book.

    get_status(book)
        Given a string with book information,
        returns True or False for the availability.

    get_title(book)
        Given a string with book information,
        returns the title of the book.

    pack(key, status, title)
        Returns the string with book information
        where key is the book identification number,
        status is True or False for availability,
        title is the title of the book.
```

the function pack() of the module bkform

```
def pack(key, status, title):
    """
    Returns the string with book information
    where key is the book identification number,
    status is True or False for availability,
    title is the title of the book.
    """
    if status:
        result = '1:'
    else:
        result = '0:'
    result += '%d:' % key
    result += title + ':\n'
    return result
```

the function get_status() of the module bkform

```
def get_status(book):
    """
    Given a string with book information,
    returns True or False for the availability.
    """
    splitted = book.split(':')
    return int(splitted[0]) == 1
```

`get_key()` and `get_title()` are similar

the main program library.py

```
from bkform import pack, get_key, get_title, get_status

def show_menu():
    "shows the menu and prompts for a choice"

def main():
    "handles menu selection"
    while True:
        choice = show_menu()
        if choice == '0':
            break
        elif choice == '1':
            show_books()
        elif choice == '2':
            add_book()
        elif choice == '3':
            checkout()
        elif choice == '4':
            checkin()
```

showing the collection: the function show_books()

`show_books()` reads the entire collection from file and shows for every book its key, title, and availability.

```
def show_books():
    "shows the books currently in the collection"
    lib = open('books', 'r')
    collection = lib.readlines()
    for book in collection:
        abook = ' ' + str(get_key(book))
        abook += ' ' + get_title(book)
        if get_status(book):
            print abook + ' available'
        else:
            print abook + ' checked out'
    lib.close()
```

adding a new book

```
def number_of_books():
    "returns the number of books in the collection"
    lib = open('books', 'r')
    collection = lib.readlines()
    result = len(collection)
    lib.close()
    return result

def add_book():
    "adds a book to the collection"
    numb = number_of_books()
    title = input('Give title : ')
    newbook = pack(numb+1, 1, title)
    lib = open('books', 'a')
    lib.write(newbook)
    lib.close()
```

borrowing a book

```
def checkout():
    "checks out a book"
    show_books()
    k = int(input('give book number : '))
    lib = open('books', 'r')
    collection = lib.readlines()
    lib.close()
    lib = open('books', 'w')
    for book in collection:
        if get_key(book) == k:
            if not get_status(book):
                print get_title(book) + ' is unavailable'
                lib.write(book)
            else:
                lib.write(pack(k, 0, get_title(book)))
        else:
            lib.write(book)
    lib.close()
```

returning a book

```
def checkin():
    "return a book to the library"
    k = int(input('give book number : '))
    lib = open('books', 'r')
    collection = lib.readlines()
    lib.close()
    lib = open('books', 'w')
    for book in collection:
        if get_key(book) == k:
            kbook = pack(k, 1, get_title(book))
            lib.write(kbook)
        else:
            lib.write(book)
    lib.close()
```

Summary + Assignments

Read pages 155-160 in *Python Programming in Context.*
for Unix, see §3.2 in *Computer Science, an overview*.

Assignments:

1. Extend the program to also contain the year of publication and author(s) for each book.
2. Write a function `empty_file` that takes on input a file name and returns `True` or `False` depending whether the file is empty or not.
3. Use the design of the library program as a model to stores the identification number, name, and address of every library patron. How will you format your file?
4. With a catalog of books and a file of patrons (from the previous exercise), design a third file that connects the borrowed books to the library patrons.

more assignments

5. Write a Python function `search_book()` that prompts the user to enter the title of a book.
   This title is then used to search for the book with the same title in the file `books`.
6. Write a Python function `delete_book()` that prompts the user for an identification number and removes the corresponding book from the file `books`. When deleting a book, change also the identification numbers so that they match the line numbers.