ECE12: Introduction to Programming
Lecture 23

Rainer Dömer

doemer@uci.edu

The Henry Samueli School of Engineering
Electrical Engineering and Computer Science
University of California, Irvine

Lecture 23: Overview

- File Processing
  - File I/O
    - Introduction
    - Functions and methods
    - Example **`print_file.py`**
  - Module **`sys`**
    - Standard I/O streams
    - Environment functions
    - Command line arguments
    - Example **`print.py`**

File Processing

- Introduction
  - Persistent data
    - Up to now, all data processed was available only during program run time; when the program terminates, all data was lost
    - *Persistent data* is stored even after a program exits
    - Persistent data is stored in files
      - on the harddisk
      - on a removable disk (floppy disk, etc.)
      - on a tape, ...
  - File I/O
    - Input/output operations on *file streams*
      - Creating files
      - Writing data to files
      - Reading data from files
      - Modifying data in files

File Processing

- File I/O Functions
  - built-in function **`open`** creates a file object (stream)
    - **`open(filename, “r”)`**
      - opens an existing file named **`filename`** for input (read)
      - returns a **`file`** object with file position at the beginning, or raises **`IOError`** (i.e. the file could not be found)
    - **`open(filename, “w”)`**
      - creates a new file named **`filename`** for output (write)
      - returns a **`file`** object with file position at the beginning, or raises **`IOError`** (i.e. the file could not be created)
    - **`open(filename, “a”)`**
      - opens an existing file named **`filename`** for extension (append)
      - returns a **`file`** object with file position at the end, or raises **`IOError`** (i.e. the file could not be found)

File Processing

- Methods provided by a file object (stream)
  - **`read(size)`**
    - reads data from the file and returns it as a string
    - optionally **`size`** specifies the maximum number of bytes to be read
    - returns an empty string at the end of the file
  - **`readline(size)`**
    - reads a line of text from the file and returns it as a string
    - optionally **`size`** specifies the maximum number of bytes to be read
    - returns an empty string at the end of the file
  - **`readlines(size)`**
    - reads all lines of text from the file and returns it as a list of strings
    - optionally **`size`** specifies the maximum number of bytes to be read
  - **`write(data)`**
    - writes the string **`data`** to the file
  - **`writelines(list)`**
    - writes each string in the **`list`** to the file

File Processing

- Methods provided by a file object (stream)
  - **`flush()`**
    - flushes the stream buffer
    - completes a read/write operation
  - **`close()`**
    - closes the stream (and releases the file resource)
  - **`fileno()`**
    - returns the file descriptor (an integer)
  - **`isatty()`**
    - returns **`1`** if the stream is an interactive terminal, otherwise **`0`** (standard disk files)
  - **`seek(offset)`**
    - moves the file position to the specified **`offset`** in bytes
  - **`tell()`**
    - returns the current file position in bytes
  - **`truncate(size)`**
    - truncates the file to the specified **`size`**
    - **`size`** is optional; if not specified, the file is truncated at position zero, so it will be empty

File Processing

- Example: **`print_file.py`**

```
# print_file.py: print the contents of a file

filename = raw_input("Please enter a file name: ")
try:
    file = open(filename, "r")
except IOError, message:
    print "Cannot open file %s for reading: %s"   \
                          % (filename, message)
else:
    print "File %s contains the following lines:" \
                          % (filename)
    lineno = 0
    while 1:
        line = file.readline()
        if not line:
            break
        lineno += 1
        print "%4d" % lineno, line,
    print "End of file %s reached." % filename
    file.close()
```

File Processing

- Module **`sys`** (system environment)
  - Standard I/O streams (opened by the environment)
    - **`sys.stdin`** standard input stream (read access)
    - **`sys.stdout`** standard output stream (write access)
    - **`sys.stderr`** standard error stream (write access)
  - Standard functions
    - **`sys.exit(value)`**
      - terminates the program execution
      - returns result **`value`** to the calling shell
        - » return **`value 0`** indicates normal program exit
        - » return **`value>0`** indicates exit due to an error condition
  - Command line arguments
    - **`sys.argv`** list of command line arguments (strings)
      - **`sys.argv[0]`** program name (always given)
      - **`sys.argv[1]`** first argument (if any)
      - **`sys.argv[2]`** second argument (if any), ...

File Processing

- Example: **`print.py`** (part 1/2)

```
#!/usr/bin/env python
#
# print.py: command to print the contents of a file
# author: Rainer Doemer
# 03/14/04 RD    initial version

import sys

# parse command line arguments
if len(sys.argv) != 2:
    sys.stderr.write("Usage: %s <file_name>\n"    \
                     % sys.argv[0])
    sys.exit(10)
filename = sys.argv[1]

# open the file for reading
try:
    file = open(filename, "r")
except IOError, message:
    print "Cannot open file %s for reading: %s"   \
                     % (filename,message)
    sys.exit(10)
...
```

File Processing

- Example: **`print.py`** (part 2/2)

```
...

# read the file
lines = file.readlines()

# close the file
file.close()

# output the contents page by page
lineno = 0
pageno = 0
pages = len(lines) / 10 + 1
for line in lines:
    if lineno % 10 == 0:
        pageno += 1
        print "\nFile \"%s\" Page %d/%d\n"     \
                 % (filename,pageno,pages)
    lineno += 1
    print "%4d" % lineno, line,
```