

COMPUTING FOR SCIENCE AND ENGINEERING

COPYRIGHT ©2009 Robert D. Skeel. All Rights Reserved.

Contents

1	Introduction	1
1.1	Nature of Computational Science and Engineering	1
1.2	Using Unix	2
1.2.1	Shell scripts	4
1.2.2	Unix tools	5
1.3	Installation of Software	6
1.3.1	Remote access and file transfer	6
1.3.2	Unix, C compiler, OpenMP, and MPI	6
1.3.3	Python, NumPy, and MPI for Python	8
2	Files and Text Processing	10
2.1	Scripting Languages	10
2.1.1	Python	10
2.1.2	Other scripting languages	10
2.2	Using Python	11
2.2.1	Doing computations	11
2.2.2	Developing a script	11
2.2.3	Developing a module	12
2.2.4	Documentation	13
2.2.5	The <code>os</code> module	13
2.2.6	The <code>subprocess</code> module	13
2.3	Files and I/O	14
2.4	Regular Expressions	16
2.5	Data	18
2.5.1	XML	18
2.5.2	Relational databases	18
3	Mathematical Computing	19
3.1	Computing with Python	19
3.1.1	Control structures	19
3.1.2	Variables and data	20
3.1.3	Functions	22

3.1.4	Quality assurance and verification	23
3.1.5	Debugging	24
3.2	Floating-point Computation	26
3.2.1	Floating-point numbers	26
3.2.2	Rounding	27
3.2.3	Floating-point operations	27
3.2.4	Other types of arithmetic	28
3.3	Pseudo-Random Numbers	28
3.4	Graphics	29
3.5	Differentiating Computer Code	30
3.6	Computer Algebra	30
3.6.1	Python	30
3.6.2	<i>Mathematica</i>	30
3.6.3	Maple and Matlab	31
3.7	NumPy	31
3.7.1	Array constructors	31
3.7.2	Indexing	32
3.7.3	Array operations	32
3.8	Other Tools	33
3.8.1	SciPy	33
3.8.2	Matlab and Octave	33
3.8.3	R	33
3.8.4	Fortran, C, and C++	34
4	C Programming	35
4.1	Pointers, Arrays, and Memory Management	36
4.2	Parameter Passing	37
4.3	Scope	38
4.4	Structures	39
4.4.1	Abstract data types	40
4.4.2	Data structures	42
4.4.3	C shortcuts	43
4.5	Compiling and Debugging	44
4.6	Interfacing Python to Compiled Code	44
4.6.1	Creating dynamically linked libraries	45
4.6.2	The <code>ctypes</code> Python module	46
4.6.3	The Numpy <code>ctypes</code> attribute	47
5	Parallel Computing	48
5.1	Computer Organization	48
5.2	Distributed Memory Programming Paradigm	49
5.2.1	Point-to-point operations	49
5.2.2	Collective operations	50

5.2.3	* MPI with Python	51
5.2.4	Grid computing and Condor	51
5.3	Computing on Radon and PBS Job Submission	52
5.4	Parallel Algorithms	53
5.4.1	Divide and conquer	54
5.5	Shared Memory Programming Paradigm	54
5.5.1	POSIX Threads	54
5.5.2	OpenMP	55
5.5.3	General purpose computing on graphics processing units	56
5.5.4	Unified Parallel C (UPC)	57
6	Developing Software	58
6.1	Object-oriented Programming	58
6.2	GUI Programming	60
6.2.1	Web interfaces	61
6.3	Software Engineering	61
6.3.1	Version control	61
6.3.2	Automated builds	62
6.3.3	Coding standards	63
6.3.4	Documentation	63
6.3.5	Development process	63
6.3.6	Integrated development environments	63
7	Error	65
7.1	Approximation	65
7.2	Error Propagation	67
7.2.1	Addition/subtraction	67
7.3	Computational Error	68
7.3.1	Rounding error	68
7.3.2	Statistical error	69
7.4	Algorithms and Numerical Stability	69
8	Functions	72
8.1	Polynomial approximation	72
8.1.1	Polynomial representation	72
8.1.2	Existence and uniqueness	73
8.1.3	Lagrange form	73
8.1.4	Interpolation	74
8.1.5	Bivariate linear polynomials	74
8.2	Piecewise Polynomial Approximation	75
8.2.1	Piecewise polynomials	76
8.2.2	Multivariate piecewise linear polynomials	76
8.3	Trigonometric Approximation	77

9	Matrices	80
9.1	Matrix Computations	80
9.1.1	Partitioned matrices	80
9.1.2	Structured matrices	81
9.1.3	NumPy matrix operations	81
9.1.4	Efficient matrix operations	82
9.2	Triangular Factorizations	82
9.2.1	Division by a triangular matrix	83
9.2.2	LU factorization	84
9.2.3	Symmetric matrices	86
9.2.4	NumPy and LAPACK functions	86
9.3	Orthogonal Factorizations	88
9.3.1	Full and reduced QR factorizations	88
9.3.2	Singular value decomposition	89
9.3.3	NumPy and LAPACK functions	90
9.4	Spectral Factorizations	91
9.4.1	General matrices	92
9.4.2	Symmetric matrices	93
9.4.3	NumPy and LAPACK functions	94
9.5	Parallel Algorithms	94
9.5.1	Divide and conquer	94
9.5.2	Relaxation methods and reordering	95

Preface

The purpose of the course is to expose students to computational concepts, tools, and skills useful for research in computational science and engineering, beyond what is learned in a first programming course (and basic mathematics courses).

Ideally computational science and engineering (CS&E) is practiced by using convenient input devices to define (a model of) physical reality and the questions that are being asked. Results are displayed in a way that most effectively conveys the information—usually graphically.

Even in the best cases, application-oriented software and environments require computational skills from the user. The user will have to write scripts for the application and for job control. He(/she) will have to choose algorithm parameters and make judgments about the accuracy of the results.

In less than best cases, the computational scientist will have to do some software development, even if “only” to modify the code. In the worst case, he(/she) may have to develop algorithms.

Prerequisites for this course are

- mathematical knowledge and maturity of an MS student in the physical sciences. (It is not assumed that the student is in the physical sciences, only that he(/she) has this level of mathematics.) In particular, some familiarity with matrix algebra and elementary probability is expected.
- programming experience in C, C++, Java, or Fortran or extensive scripting experience; also, commensurate computer skills.

There is no suitably comprehensive textbook; hence these notes with their links to web resources. A good reference is [Software Carpentry](#). Some useful books:

- Python in a Nutshell, by Alex Martelli, O’Reilly, March 2003, ISBN: 0-596-00188-6,
- Learning Python, 2nd Edition, by Mark Lutz, and David Ascher, O’Reilly, December 2003, ISBN: 0-596-00281-5,
- Python Scripting for Computational Science, 3rd Edition, 2nd Printing
by Hans Petter Langtangen, Springer, 2008,
- The C Programming Language, 2nd Edition, by Brian W. Kernighan and Dennis M. Ritchie, Prentice-Hall, 1988, ISBN 0-13-110362-8,

- An Introduction to Parallel Computing: Design and Analysis of Algorithms, 2nd Edition, by Ananth Grama, George Karypis, Vipin Kumar, and Anshul Gupta, Addison-Wesley, 2003, ISBN: 0-201-64865-2.

The course outline is

Theme I: scripting to get things done

1. introduction to CS&E, Unix, software installation
2. files and text processing in Python, including regular expressions
3. mathematical computing in Python, including floating-point arithmetic, graphics, Numpy, computer algebra

Theme II: programming for performance

4. C programming, including data structures, interfacing to Python
5. parallel programming, including computer organization, PBS, MPI, OpenMP

Theme III: developing software

6. developing software: object-oriented programming, GUIs, version control, automated builds

Theme IV: numerical computing

7. error: data error, error propagation, computational error, algorithms, numerical stability
8. functions: polynomials, piecewise polynomials, trigonometric polynomials
9. matrices: operations, factorizations, parallelism

Theme IV is to be covered concurrently with Themes I–III as a source of applications.

Chapter 1

Introduction

COPYRIGHT ©2009 Robert D. Skeel. All Rights Reserved.

1.1 Nature of Computational Science and Engineering

Many say that computation has joined experiment and theory to become the third pillar of science; an alternative view is that computation is an extension of theory.

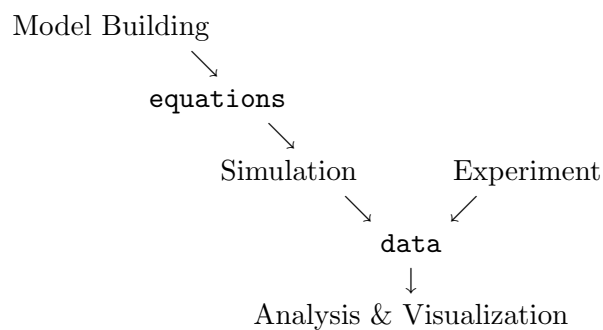
Three stages of scientific computing might be identified:

model building , which produces equations

simulation , which produces data, and

analysis

Experiment also produces data (for analysis):



Mathematics is central to science: scientific claims must be falsifiable, and only precise—i.e., mathematical—statements are falsifiable. Mathematics is central to computation: only precisely formulated instructions can be executed.

1.2 Using Unix

References: Langtangen(2006), Chapter 1 and Appendix A; [Software Carpentry](#), Basic Unix Shell, More Unix Shell; [http://en.wikipedia.org/wiki/Shell_\(computing\)](http://en.wikipedia.org/wiki/Shell_(computing)).

The operating system handles the logistics of computing. Application programs interact with the operating system through function calls as specified by an *application program interface* (API). Users interact through a program called a *shell*, either a *command line interface* or a *graphical user interface* (GUI). Examples of the latter include Windows Explorer and Mac OS Finder.

There are three main flavors of Unix: Solaris, BSD, and Linux. Max OS X is partly based on BSD. Quoting from *Unix Power Tools* p. 1,

UNIX ... wasn't designed as a commercial operating system meant to run application programs, but as a hacker's toolset, by and for programmers.

(The word “hacker” is used here in its original benign sense to mean a computer enthusiast.) Unix tools are freely available for Windows in a collection called Cygwin.

POSIX is a standard for Unix operating systems. Solaris and Mac OS 10.5 are POSIX compliant; most versions of Linux are not.

Using [virtualization software](#), two (or more?) operating systems can be run simultaneously.

Shells The most popular (command line) shell is bash, which is the default for Linux and Mac OS X. Also popular is csh/tsh. The bash prompt is “\$”, and the csh prompt is “%”. To change your login shell, run the `chsh` command. (The command line shell for Windows is `cmd.exe`.)

Navigating the file system A file is the basic unit of storage on a file system, e.g., a disk drive. These are hierarchically grouped into directories (or folders). Especially useful commands are

```
pwd
ls or ls -l or ls -a or ls -al
cd <name>
cd ..
cd / # the root
cd ~ # home directory
cd ~myfriend
```

An *absolute path* starts at the root; a *relative path* starts at the current working directory, denoted by a single dot. Navigating becomes more complicated for networked file systems, e.g., NFS, that span several machines. Be aware that some files and directories are merely links (or aliases) of actual files and directories. The command `ls -F` distinguishes among files, directories, links, executables, etc.

Command syntax A command begins with the name of the command followed by a list of zero or more arguments separated by spaces. One can use a limited set of emacs commands to edit the line before hitting `return`: arrow keys, `esc-B`, `esc-F`, `control-A`, `control-E`, `control-T`, `esc-D`, `esc-delete`, `control-D`, `delete`, `ESC-L`, `ESC-C`, `ESC-U`. The entire line can be deleted using `control-U`.

Command completion uses the tab. Most commands have optional “keyword” arguments known as *flags*. The keyword (usually a single letter) is preceded by a hyphen and followed by the argument value, if any. Sometimes the position of the flag matters. Some generally useful flags for running programs are

- help, e.g., `python -h` , `grep --help` ,
- version, e.g., `python -V` , `gcc -v` , `mpirun -version` , `ipython -Version` ,
- verbose, e.g., `gcc -v` .

The last of these is useful for determining location of files and directories used by the program.

Processing files The simplest way to create a short file is to enter

```
cat > myfile
```

and then sequentially enter the lines of the file terminated by control-D. To allow corrections, you need to use a text editor like

```
vi myfile or emacs myfile
```

While in emacs, you use the arrow keys to navigate and you do insertions simply by typing them in. To save updates, enter control-X, control-S, and to quit, enter control-X, control-C. *Renaming* a file is done via

```
mv file1 file2
```

removing a file via

```
rm file1
```

and *copying* a file via

```
cp file1 file2
```

The command

```
find $HOME/ -name "*python*"
```

prints the path of each file in your home direct and its subdirectories whose name contains the string “python”. The wildcard character “*” matches any string including an empty string. The character “?” matches any single character.

Variables and scope Variables can be defined to have strings as their values, e.g.,

```
x=one+2
```

```
export y="three, 4"
```

If we then run a new shell, only those variables that are exported will be known. Try

```
bash
```

```
echo $x
```

```
echo $y
```

```
x=five,6
```

If we exit the new shell,

```
exit
```

the first value of `x` is restored.

Commands The user can create commands. One way is to create an executable, e.g.,

```
gcc myprog.c
```

and then execute it by entering

```
a.out or ./a.out
```

Existing programs can be executed if their path is known, e.g.,

```
whereis emacs
```

reveals that we need to enter

```
/usr/bin/emacs
```

However, the `whereis` command looks only in a few likely places. If, in fact, `emacs` is known without giving a path and there is more than one program with that name, you can learn which version is being used, by entering

`which emacs` To interrupt a running program, enter control-Z. To resume execution, enter `fg` (foreground). To abort a computation, control-C often works.

Environment variables Their values can be seen by entering

```
printenv
```

An important variable is `PATH`, whose value is a colon-separated list of directory paths, which are searched for command names.

Job control The `jobs` command shows those processes initiated by the user from the command line. To refer to a process by its number in the jobs listing, precede that number by `%`, e.g., `kill %2` terminates the second job listed.

Advanced commands To pack a set of files and directories into a single file, use

```
tar -cf - <list> > name.tar
```

and to unpack

```
tar -xf name.tar
```

An alternative is to use

```
zip <name>.zip <list>
```

for packing

```
unzip <name>.zip
```

for unpacking. To locate occurrences of a string of characters in a set of files, use, e.g.,

```
grep "site-packages" *.py
```

1.2.1 Shell scripts

Creating commands You may create a command from a shell script. The simplest way is to put commands into some file `myscript` and enter “`source myscript.`” Using the `source` command is unnecessary if the first line is

```
#!/bin/bash
```

and the file is made executable by entering the command

```
chmod 700 myscript
```

at the shell prompt. The first digit specifies your permissions and the other two digits the permissions of other users. Each digit is a weighted sum of 4, 2, and 1 where

```
4 read  2 write  1 execute
```

To make the file executable (and readable) by others, use 755 instead.

The .profile and .bashrc files These file contains commands that are executed when you open a new shell. If it is a “login shell”, the `.profile` file is executed; otherwise the `.bashrc` file is. The distinction is not simple to describe and may be platform dependent, so one solution is to put the line `source .bashrc` at the end of the `.profile` file (and place most of the commands in the `bashrc` file). These files may contain commands like

```
export PATH=~ /bin:${PATH}:/usr/local/bin
export EDITOR=emacs
alias mv="mv -i"
```

To execute the file after modifying it, enter

```
source .profile
```

In `csh` the `.login` and `.cshrc` files serve the same purpose.

1.2.2 Unix tools

As an example, the `awk` command

```
awk '{if ($1=="ENERGY:") print $2 " " $11}' < all.data > select.data
```

reads from `all.data` and for each line having “ENERGY:” as its first item it prints the second and eleventh item of that line to `select.data`.

Review questions

1. What is the Unix shell command for changing to the parent of the current directory?
2. What is the effect of the command

```
mv file1 file2
```

if `file2` does not exist? if `file2` does exist?
3. The default name for the executable produced by the C compiler is `a.out`. If the command `a.out` does not work, what should one try next?
4. What is the name of the environment variable that lists directories which are searched to find a command name?
5. What command should you execute after modifying the `.profile` file?
6. Give a Unix command for creating a `tar` file consisting of all files in the current directory with extensions `.tex`, `.eps`, and `.bbl`.
7. How should you set the permissions to make a file executable by any user but readable and writeable only by the owner?

1.3 Installation of Software

Installation of software is done in two ways:

1. from a binary. This is typically performed by point-and-click and generally seems to require that you have superuser/administrator privileges.
2. from source. This done at the command line. If you have superuser privileges, you can preface the install command with `sudo`. If not, you can install in `$HOME/local` using an option like `--prefix=$HOME/local` or `--home=$HOME/local`. In such case, make sure that `$HOME/local/bin` is in your path.

The installation process includes

download, unpack, configure, build, install, test.

Much of what is downloaded can be discarded after you are satisfied with the installation.

1.3.1 Remote access and file transfer

You may choose to use the Rosen Center Red Hat Linux cluster `radon`. Documentation is [here](#). In particular, there is information on [remote access and file transfer](#). For example, to copy between machines, use something like

```
scp -rpc me@myothercomputer.purdue.edu:file1 file2
```

Also given there are links to GUIs for Windows. There is a GUI for Macs called [fugu](#).

1.3.2 Unix, C compiler, OpenMP, and MPI

Some of this software might already be installed.

1. Needed is a C compiler that supports OpenMP, e.g., [version 4.2 or later](#) from [www.gnu.org](#). Test with the program `ohello.c`, adapted from [Wikipedia](#):

```
#include <omp.h>
#include <stdio.h>
int main (int argc, char *argv[]) {
    #pragma omp parallel private(id)
    {
        int id = omp_get_thread_num();
        printf("Hello World from thread %d\n", id);
        #pragma omp barrier
        if ( id == 0 ) {
            int nthreads = omp_get_num_threads();
            printf("There are %d threads\n",nthreads);
        }
    }
    return 0;
}
```

as follows:

```
$ export OMP_NUM_THREADS=4
$ gcc -fopenmp -lgomp ohello.c
$ ./a.out
```

2. MPI. Test with the program hello.c

```
#include <mpi.h>
#include <stdio.h>
int main(int argc, char *argv[]) {
    MPI_Init(&argc, &argv);
    int npes; MPI_Comm_size(MPI_COMM_WORLD, &npes);
    int myrank; MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
    printf("Hi there from %d of %d\n", myrank, npes);
    MPI_Finalize();
}
```

as follows:

```
$ mpicc hello.c
$ mpirun -np 4 ./a.out
```

Platform specific notes follow:

Windows

For Windows users, a probably better choice than Cygwin is to install Ubuntu. (A disadvantage is the need to restart when switching between Ubuntu and Windows.) Installation is very easy and straightforward. The link for downloading windows installer of ubuntu is <http://wubi-installer.org/> . The default installation will create a dual boot menu at startup time.

Linux

For Ubuntu you might start with <https://help.ubuntu.com/> .

Mac OS X

Install “Xcode Tools” either from the install DVD or from <http://developer.apple.com/tools/download/>.

Install Xwindows from the “Optional Installs” on the install DVD and select X11 from among the Applications and run Software Update. (Reference: <http://developer.apple.com/opensource/tools/runningx11.html>)

See <http://hpc.sourceforge.net/> for many useful binaries.

1.3.3 Python, NumPy, and MPI for Python

Consider installing the comprehensive collection of tools and libraries from the “[Enthought Python Distribution](#)”. In particular, it includes Python 2.5.4, NumPy 1.3.0, scipy, matplotlib 0.98.5.2, wxpython, VTK, mpi4py 0.6.0, IPython 0.9.1, and sympy. Go to <http://www.enthought.com/products/getepd.php> and just click on “Educational Use.”

The Enthought distribution is available for only Windows, Mac OS X, Red Hat Linux, and Solaris. For Ubuntu Linux, try <http://packages.ubuntu.com/hardy/python/>. Also, see <http://www.scipy.org/Download> for NumPy. There are package systems for other Unix distributions, including Debian, Fedora, SUSE, fink, FreeBSD, and OpenBSD.

Version 2.5.4 of Python provided by Enthought is not the latest version. It contains obsolete features not present in Python 3.1, which is the most recent version. However, many packages have not upgraded yet, so there is a version Python 2.6.2 that includes features of both. If you want to install this version, you will have to do separate installations of the desired packages. Proceed carefully and search the web for advice. Chapter 1, Appendix A, and Appendix B.1.3 of Langtangen(2008) may be helpful.

Download Python 2.6.2, from <http://www.python.org/download/>. Using `which python`, determine whether 2.6 is the default or whether `python` references a pre-existing installation, in which case `python2.6` might work. For installation and use of some software, you need the location of the actual python binary:

```
python -c "import sys;print sys.prefix"
```

Installation of packages is facilitated by [EasyInstall](#). For upgrading an existing installation, use the `--upgrade` flag.

Obtain NumPy 1.3.0 from <http://sourceforge.net/projects/numpy/files/NumPy/>. Afterwards, execute “`import numpy; numpy.test()`”. You may wish also to install SciPy, available from <http://sourceforge.net/projects/scipy/files/>.

For a plotting program, you might use [Matplotlib](#). Read <http://matplotlib.sourceforge.net/users/installing.html> and install from <http://sourceforge.net/projects/matplotlib>. If you do not have administrator permissions for your machine, use `python setup.py install --home $HOME/local`.

Installation can be tested with

```
import matplotlib
from pylab import *
ion()
plot([1, 2, 3])
```

Quite useful for interactive use and code development is [IPython](#). You can install it and other packages such as `mpi4py` using

```
sudo easy_install ipython[kernel,security,test]
```

After installation open a new Terminal window and type

```
ipython
```

Alternatively, obtain `mpi4py` from <http://mpi4py.scipy.org/>. The latest version is 1.1.0. Note that `mpi4py` may not work with all Python distributions. Installation instructions for IPython are at <http://ipython.scipy.org/doc/rel-0.10/html/install/install.html> The latest version

is 0.10.

Review questions

1. Suppose

```
$ which python
/usr/bin/python
$ ls -l /usr/bin/python
lrwxr-xr-x 1 root  wheel  72 Nov 16  2007 /usr/bin/python@ -> ../System/Li
brary/Frameworks/Python.framework/Versions/2.5/bin/python
```

What would be the absolute path for the python binary?

2. In order for a command `mycmd` in the current working directory to execute in Unix, what two requirements must be met?

Chapter 2

Files and Text Processing

COPYRIGHT ©2009 Robert D. Skeel. All Rights Reserved.

Python is a suitable scripting language for file and text processing.

2.1 Scripting Languages

2.1.1 Python

Python is a full-featured programming language which together with extensions offers the features of both Matlab and Perl with only modest reduction in convenience. It is a scripting language suitable for a variety of high-level tasks. An example is given in Langtangen(2006), Section 2.4, of using Python to conduct a battery of numerical experiments and present the results on web pages. It is also suitable for rapid prototyping of algorithms later to be translated into a more efficient programming language. And it is useful for gluing together functions written in several programming languages.

Python was created by Guido van Rossum, Benevolent Dictator For Life, at CWI in Amsterdam in 1991. It is free and has an unrestrictive license.

Python includes a standard library. In addition, numerous extensions have been developed, most important of which is Numpy, which permits efficient array operations, essential for scientific computing.

2.1.2 Other scripting languages

Tcl/Tk Tcl is a scripting language and Tk is a toolkit for building GUIs. It is free, stable, and lightweight. The Python standard library contains is an interface to Tk called Tkinter.

Perl Perl is a popular scripting language with syntax from C and Unix shell scripting. It is now losing marketshare to Ruby.

Ruby Ruby is described as a combination of Perl syntax and SmallTalk semantics. SmallTalk is a good example of an object-oriented programming language.

Lua Lua is a light-weight scripting language, which is said to interface easily to C.

2.2 Using Python

References: Langtangen(2006), Sections 2.1 and 3.4; [Software Carpentry: Python Basics](#); [Software Carpentry: Python Functions and Modules](#); and [Python Tutorial](#). The tutorial has links to the documentation, and starting from the tutorial is an excellent way to access the documentation.

2.2.1 Doing computations

Python programs can be run noninteractively from the operating system command line. They can be run interactively in various ways:

best by typing “ipython” from the command line.

better by launching IDLE, which is an integrated development environment GUI. Either double-click its icon or enter “idle -n &” (or equivalent) from the command line. The “-n” flag is needed if using Matplotlib.

good by typing “python” from the command line.

Interaction with Python at the most basic level is illustrated below:

```
$ python
>>> import math
>>> math.sin(3.141592)
6.53589793076e-07
```

The Python `import` command can be used in various ways:

with `import math` you can use `math.sin`, `math.cos`, etc.

with `from math import sin` you can use `sin` but `math.cos`, etc. are unavailable,

with `from math import *` you can use `sin`, `cos`, etc.

The last of these should be used sparingly. It may be convenient to put such commands into an [interactive startup file](#).

To continue a statement to the next line put a backslash at the end of the line.

2.2.2 Developing a script

An example from Langtangen(2006): file `hw.py` containing

```
#!/usr/bin/env python
import sys, math
r = float(sys.argv[1])
s = math.sin(r)
print "Hello, World! sin(" + str(r) + ")=" + str(s)
```

can be executed by entering

```
./hw.py 1.4
```

Scripts can be developed using your favorite editor and running them either at the command line—or inside IPython:

```
$ ipython
In [1]: run hw.py 1.4
Hello, World! sin(1.4)=0.985449729988
In [2]: edit hw.py
```

The “run” and “edit” commands are IPython extensions to Python. Also very useful is “history.” Here are [tips for using IPython effectively](#).

2.2.3 Developing a module

For example, if file `hwm.py` contains

```
def hw(r):
    import math
    s = math.sin(r)
    print "Hello, World! sin(" + str(r) + ")=" + str(s)
```

the following would result:

```
$ python
>>> import hwm; hwm.hw(3.141592)
Hello, World! sin(3.141592)=6.53589793076e-07
```

Suppose we want to edit `hwm.py` to change `sin` to `cos`.

best Use the IPython `edit` command. Exiting the editor causes the contents of the file to be executed, including definitions:

```
Hello, World! cos(3.141592)=-1.0
(However, the name of the module ‘hwm’ will not be defined by this action.)
```

better Edit ‘`hwm.py`’ in another window, and enter

```
execfile('hwm.py')
Hello, World! cos(3.141592)=-1.0
(Again, the name of the module ‘hwm’ will not be defined by this action.) In IPython, it is
enough to enter execfile 'hwm.py' .
```

good Edit ‘`hwm.py`’ in another window, and enter

```
>>> reload(hwm)
Hello, World! cos(3.141592)=-1.0
The import command defines the hwm module only if it has not already been defined. A
reload is needed to overwrite the existing definition. A word of caution. If, instead you had
initially entered simply
    from hwm import hw
you have to do the following steps after editing the function
    import hwm
    reload(hwm)
```

```
from hwm import hw
```

(An additional limitation of `import` is noted in Section 3.8.1.)

2.2.4 Documentation

There is online [Python Documentation](#). At the operating system command line, type `pydoc`; at the Python command line, `help`.

2.2.5 The `os` module

Operating system commands can be executed using generic Python commands:

```
os.remove('junk')
os.listdir(os.getcwd())
import glob
glob.glob('*') + glob.glob('.*')
```

2.2.6 The `subprocess` module

Alternatively, operating system commands can be executed using shell commands:

```
subprocess.call('ls', shell=True)
subprocess.call('gnuplot sine.gplt', shell=True)
```

An alternative to `subprocess.call` that works with IDLE is given below:

```
def bang(cmd):
    import subprocess
    proc = subprocess.Popen(cmd, shell=True, stdout=subprocess.PIPE)
    for line in proc.communicate(): print line
bang('ls')
```

Review questions

1. What is the current name of the package necessary for efficient scientific computing with Python?
2. What is the value of `sys.argv`, in the case where `merge.py 1 '2 3'` has been entered.
3. What module contains functions for navigating the file hierarchy and otherwise interacting with the operating system?
4. Assuming that `glob` has been imported, what does `glob.glob('*.*bib')` return?


```
>>> heaviside(0.)
0.5
```

Creating a list with embedded for loop:

```
>>> [float(n) for n in range(1, 11)]
[1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 10.0]
```

Strings

```
str.split('12 34')
str.join(' ', ['12', '34'])
str.replace('102 -100 = 0102', '102', '___', 1)
```

Note that in strings “\” is an escape character, which changes the meaning of what follows, e.g. \a is bell, \b is backspace, \n is newline, \r is carriage return, and \t is tab. Use \\ to denote a backslash.

Reading and writing Reading from a file and writing to a file:

```
ifile = open('data', 'r')
ofile = open('results', 'w')
for line in ifile:
    items = str.split(line)
    file.write(ofile, items[2] + '\n')
file.close(ifile)
file.close(ofile)
```

The open command returns a file object, which is a representation of the file in computer memory. Lines in Windows files end with “\r\n”. Go [here](#) for conversion utilities.

Output formatting For information see [Software Carpentry: Python Basics](#) or “6.6.2 String Formatting Operations”

Input Following are functions for reading: `file.readline`, `file.readlines`, `file.read`. Input from the terminal is from file object `sys.stdin`. For more information see “7.2 Reading and Writing Files”

Objects An *object* is a unit of data residing in memory. Objects come in different types, e.g., `int`, `float`, `str`, `list`, `tuple`, `dict`, `file`, `bool`, `long`, `NoneType`. Each type of object has a *constructor* for creating it, the name of the constructor being the same as the name of the type, e.g.,

```
ifile = file('data', 'r')
```

which was written above as

```
ifile = open('data', 'r')
```

because `open` happens to be an alias for the `file` constructor. Associated with a type or *class* of objects are *methods* for operating on them, e.g., `file.write` and `file.close`. The first argument of such a method must be an object from the designated class.

Object-oriented style The following pairs are equivalent:

<code>file.write(ofile, line)</code>	<code>ofile.write(line)</code>
<code>file.close(ifile)</code>	<code>ifile.close()</code>
<code>str.split('12 34')</code>	<code>'12 34'.split()</code>
<code>str.join(' ', ['12', '34'])</code>	<code>' '.join(['12', '34'])</code>
<code>list.append(names, name)</code>	<code>names.append(name)</code>

Since the type of an object is known, the name of the type can be replaced by the object.

Example of a filter With this much knowledge of Python, we can write a script equivalent to the awk script of Section 1.2.2:

```
#!/usr/bin/env python
import sys
for line in sys.stdin:
    items = line.split()
    if items != [] and items[0] == 'ENERGY:':
        print items[1], items[-4]
```

A program such as this which reads from `stdin` and writes to `stdout` is called a *filter*.

Review questions

1. Which file object does `print` write to? and what extra things does `print` do to the output that `write` does not do?
2. What is the alternative objected-oriented syntax for `file.close(myfile)` ?
3. Write a Python code segment that constructs a list of (double-precision) floats with integer values from 0 to `n`.

2.4 Regular Expressions

References: Langtangen(2006), sections 8.2.0–8.2.3, [Software Carpentry: Regular Expressions](#), and [Python Library Reference, 4.2](#).

A *regular expression* (regex) is a language for specifying string patterns. A string *pattern* is a set of strings. Regular expressions can be used in text editors like VIM/vi and emacs, in programs like grep, and in languages like awk, perl, and Tcl.

There are three levels of regular expressions: basic, extended, and Perl. Python and `grep -P` use Perl regexes; emacs regexes are somewhat different from these. (An emacs forward regex search begins with `esc control-S`.)

Here are some basic rules:

- Most characters represent themselves. To represent a special character precede it with a backslash.
- A dot represents any character.

- Quantifiers, `*`, `+`, `?` follow the pattern they quantify.
- Parentheses are for grouping.
- Alternatives are enclosed in parentheses and separated by a vertical line.
- An enumerated set of single characters is enclosed in square brackets. In this context an initial caret means “not.”
- At the beginning of a regular expression, a caret means “beginning of string” and at the end a dollar sign means “end of string.”
- An abbreviation for `[0-9]` is `\d`.

Within a Python string, a backslash `\` is represented by `\\`, e.g., `'\\d+(\\.\\d*)?'`. Alternatively, use a raw string `r'\d+(\\.d*)?'`, which is recommended for regular expressions.

As an example, one might use

```
integer = r'(0|-?[1-9]\d*)'
```

to locate a properly formatted integer in a string. The Python code

```
import re
match = re.search(integer, '102 -100 = 0102')
if match: print match.group(), match.start(), match.end()
```

prints `102 0 3`. To find all nonoverlapping substrings that match, use

```
re.findall(integer, '102 -100 = 0102')
```

which returns `['102', '-100', '0', '102']`. Note that only longest possible matches are returned. This result, however, is not what is intended. A better pattern is

```
integer = r'(^|\d)(0|-?[1-9]\d*)($|^\.\d)'
```

This requires that the integer be either at the beginning of the string or be preceded by other than a digit, and it requires that it either be at the end of the string or be followed by other than a digit or decimal point. However, the substring we want should exclude a character that precedes or follows the integer. This is denoted by `match.group(2)` in this example; the 2 refers to the sub-regex between the 2nd left parenthesis and its matching right parenthesis. Text matching the entire regex itself is `match.group()`.

As a second example, consider those strings that represent a `float` value in Python. To be more specific, consider those strings `s` for which (i) `type(eval(s))` is `float` and (ii) `float(s)` does not raise a `ValueError`. A suitable regex will encompass both “F” format and “E” format. An F-formatted `float` will have exactly one decimal point and at least one digit:

```
real_f = r'[+-]?(\d+\.\d*|\.\d+)'
```

An E-formatted `float` need not have a decimal point:

```
real_e = r'[+-]?(\d+(\.\d*)?|\.\d+)[Ee][+-]?\d+'
```

Combining the two:

```
real = '(' + real_f + '|' + real_e + ')'
```

Review questions

1. What is the result of

```
import re
re.findall(r'(0|-?[1-9]\d*)', '102 -100 = 0102')
```

2. Write a regular expression for a string of zeros and ones which begins with a one. The regular expression must match the entire string, not just a substring.
3. Write a regular expression for a dollar amount. The string must begin with “\$” and have a decimal point preceded by one or more digits and followed by two digits. The first digit may not be a zero unless the amount is less than one dollar.
4. Consider the two tests (i) `type(eval(s)) is float` and (ii) `float(s)` does raise not a `ValueError`. Give an example of a string that passes the first test but not the second, as well as one that passes the second test but not the first.

2.5 Data

The `cPickle` module of the Python standard library can be used to store a Python object onto disk for later use.

2.5.1 XML

Reference: [Software Carpentry: XML](#)

XML is a (verbose) language with syntax like HTML for representing data in a standardized hierarchical fashion so that it can be readily processed by computer. It is customizable to specific applications.

2.5.2 Relational databases

Reference: [Software Carpentry: Databases](#)

Data bases refer to the organization of data stored permanently, e.g., on hard disks. (Data structures on the other hand refers to the temporary representation of data in memory.)

It is popular to use relational data bases and the Structured Query Language SQL for accessing them. A well known free database management system (DBMS) is MySQL.

Review questions

1. What is the HTML-like language for describing data?
2. What is the popular standardized language for using relational data bases?