Useful Scripting for Biologists

Brad Chapman

23 Jan 2003

Objectives

- Explain what scripting languages are
- Describe some of the things you can do with a scripting language
- Show some tools to use once you pick a language
- Show an example of how I've used scripting languages
- Overall, convince you it's worthwhile to learn to program in a scripting language

Who am I?

- Biologist – all of my background is in lab work
- Took two programming classes as an undergrad (C++)
- Needed programming for my work when I came to UGa
- Taught myself to program in the scripting language python

What is a scripting language?

In Computer Science terms

- Dynamically typed (do not have to declare variables)
- Are not compiled

In Practical Terms

- Faster to learn to program in
- Easier to debug and fix problems
- Programs run slower, however, because of the niceties

Examples of scripting languages

- Perl – http://www.perl.com
- Python – http://www.python.org
- Tcl – http://tcl.sourceforge.net/
- Ruby – http://www.ruby-lang.org/en/
- "Web" languages – JavaScript, PHP
- Others – Lisp, Scheme, Ocaml

Some things scripting languages make easier

- Dealing with text files
- Interacting with the web
- Making graphical user interfaces
- Creating dynamic web pages
- Tieing together multiple programs
- Parsing information from files or web pages
- Retrieving information from Databases
- Reorganizing data (ie. into a spreadsheet ready format)
- Generating figures
- Automating repetitive daily tasks
- Writing one-time programs

Other Practical Advantages of Scripting Languages

- All those mentioned are freely available and most have very non-restrictive licenses
- Lots of free documentation on the web for learning how to program in the language
  - Python – http://www.python.org/doc/current/tut/tut.html
  - Ruby – http://www.rubycentral.com/book/
- Really relieves your frustration level in programming compared to compiled languages like C++
- Programs can often be written quicker because you don't have to worry about compiling woes, variable declaration and so on

How do you choose a scripting language?

- Want to pick something easy to learn and get started with
- Make a list of some things you want to do with the language and look for languages that have ready made packages for what you'd like to do
  - Python – Interact with Excel on a Windows computer: Python Windows Extensions
  - Perl – Create charts for display on the web: GDGraph
  - Tcl – write a graphical user interface : Tk
- Looking at the code in different languages to see what appeals to you
- Try them out

Perl code

```
$nBottles = $ARGV[0];
$nBottles = 100 if $nBottles eq '' || $nBottles < 0;

foreach (reverse(1 .. $nBottles)) {
    $s = ($_ == 1) ? "" : "s";
    $oneLessS = ($_ == 2) ? "" : "s";
    print "\n$_ bottle$s of beer on the wall,\n";
    print "$_ bottle$s of beer,\n";
    print "Take one down, pass it around,\n";
    print $_ - 1, " bottle$oneLessS of beer on the wall\n";
}
print "\n*burp*\n";
```

Python code

```
def bottle(n):
    try:
        return { 0: "no more bottles",
                 1: "1 bottle"} [n] + " of beer"
    except KeyError: return "%d bottles of beer" % n

for i in range(99, 0, -1):
    b1, b0 = bottle(i), bottle(i-1)
    print "%(b1)s on the wall, %(b1)s,\n"\
      "take one down, pass it around,\n"\
      "%(b0)s on the wall." % locals()
```

Tcl code

```
proc bottles {i} {
    return "$i bottle[expr $i!=1?"s":""] of beer"
}

proc line123 {i} {
    puts "[bottles $i] on the wall,"
    puts "[bottles $i],"
    puts "take one down, pass it around,"
}

proc line4 {i} {
    puts "[bottles $i] on the wall.\n"
}

for {set i 99} {$i>0} {} {
    line123 $i
    incr i -1
    line4 $i
}
```

Biological Tools for Scripting Languages

- Because the advantages of scripting languages help ease a lot of biological problems (ie. file parsing) there are good resources for code for scripting language

- Open-Bio set of projects
  - BioPerl – http://www.bioperl.org
  - Biopython – http://www.biopython.org
  - BioRuby – http://www.bioruby.org/

- Projects contain code for representing sequences, parsing biological file formats, running bioinformatics programs and tons more

More Resources for Scripting Languages

- Most scripting languages come out of the open-source community
- Community built up around them that is encouraged to give away their code
- Can easily find code other people have written that may be useful to you:

  Perl Comprehensive Perl Archive Network – `http://www.cpan.org/`
  Python Vaults of Parnassus – `http://www.vex.net/parnassus/`
  Ruby Ruby Application Archive – `http://www.ruby-lang.org/raa/`

- In many cases you can find code to solve your problem or close to your problem without having to start from scratch

Solving problems

- The ultimate advantage of scripting languages is that they let you solve your problems faster

- Most of my problems in biology are not huge programs to write, but small tasks that need to get finished.
  - Retrieve information from a web page
  - Parse a ton of BLAST outputs
  - Rewrite files into different phylogenetic formats
  - Generate files for GenBank submission
  - Retrieve information from ABI trace files

Example: batch primer design

- Simplified from a real example where we needed to generate primers for several hundred sequences

- Our goals:
  - Start with a file full of sequences
  - Design primers for each sequence with specific criterion (have to span a central region)
  - Write the files to a format that can be loaded into Excel

- Accomplished in python with the help of the Biopython libraries

Our starting sequences

- Begin with files in Fasta format, a standard sequence format

```
>CL031541.94_2268.amyltp
CGCGGCCGCSGACCTYGCCGCAGCCTCCCCTTCMATCCTCCTCCCGCTCC
TCCTACGCGACGCCGGTGACCGTGATGAGGCCGTCGCCGCYTCCGCGCTC
CCTCAAGGCNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
AAGTGCCTTGGCTTCACGCTCTACCATCCGGTCCTCGTCTCCACCGTCTC
AGGTAGAATGGCTCCCCTGTATCCACAACTCGTCAGTGAAATGTGCAGTT
```

- Biopython has a module (`Bio.Fasta`) which can read these Fasta formatted files.

Our primer design program

- Primer3, from the Whitehead Institute – http://www-genome.wi.mit.edu/genome_software/other/primer3.html
- EMBOSS, a free set of bioinformatics programs, has an interface (eprimer3) to make the program easier to use – http://www.emboss.org
- Biopython has code to work with eprimer3
  - `Primer3Commandline` – run the program
  - `Primer3Parser` – parse the output from the program

Program Step 1 – read the Fasta File

Load the biopython module that deals with Fasta

```
from Bio import Fasta
```

Open the Fasta file from python

```
open_fasta = open(fasta_file, "r")
```

Initialize an Iterator that lets us step through the file

```
parser = Fasta.RecordParser()
iterator = Fasta.Iterator(open_fasta, parser)
```

Program Step 2 – Read through the file

We want to go through the file one record at a time.

Generate a loop and use the iterator to get the next record.

We stop if we reach the end of the file (get a record that is None).

```
while 1:
    cur_record = iterator.next()
    if not(cur_record):
        break
```

Program Step 3 – Set up the primer3 program

We want to create a run of the primer3 program with our parameters

```
from Bio.Emboss.Applications import Primer3Commandline

primer_cl = Primer3Commandline()
primer_cl.set_parameter("-sequence", "in.pr3")
primer_cl.set_parameter("-outfile", "out.pr3")
primer_cl.set_parameter("-productsizerange", "350,10000")
primer_cl.set_parameter("-target", "%s,%s" % (start, end))
```

start and end are the middle region we want to design primers around

Program Step 4 – Run the primer3 program

Biopython has a single way to run a program and get the output.

Python interacts readily with the operating system,
making it easy to run other programs from it.

```
from Bio.Application import generic_run

result, messages, errors = generic_run(primer_cl)
```

The Primer3 output file

Primer3 generates an output file that looks like:

```
# PRIMER3 RESULTS FOR CLB11512.1_789.buhui

#                Start Len  Tm     GC%   Sequence

1 PRODUCT SIZE: 227
FORWARD PRIMER  728 20  59.91  50.00  TTCACCTACTGCAAACGCAC

REVERSE PRIMER  935 20  59.57  50.00  TTGGTACGTTGTCCATCTCG
```

A ton of these files are not easy to deal with.

Program Step 5 – parse the output file

We use the Biopython parser to parse these files.

```
open_outfile = open("out.pr3", "r")

from Bio.Emboss.Primer import Primer3Parser

parser = Primer3Parser()
primer_record = parser.parse(open_outfile)
```

The result is that we get the information into a python ready format that we can readily output.

Program Step 6 – Write the output

We want to write the output to an Excel ready format

We write the forward and reverse sequences along with the sequence name to a comma separated value file.

```
open_excelfile = open(excel_file, "w")

primer = primer_record.primers[0]

open_excelfile.write("%s,%s,%s" % (
  sequence_name, primer.forward_seq, primer.reverse_seq))
```

The result is a file full of primers that you can then deal with.

Conclusions from all of this

- Scripting languages can be used to automate repetitive tasks (designing 500 primers)
- Scripting languages readily interact with the things biologists need to do (parse files, run programs)
- Scripting languages allow you to tackle the problem in small simpler steps (parse a file, run a program, write a file)
- Scripting languages aren't too hard to learn (see that code was easy)
- You should run right out and learn a scripting language