Announcements

- Email me if you’re not on the classlist or haven’t gotten the piazza invite (bsnyder@cs.wisc.edu)
- Questions about thursday’s lecture?
- HW 1 questions?

Why Perl and Python?

- There are many reasons to write code.
- To learn to be a better programmer
- To make or improve a software product
- To answer a question
- To test an idea (prototype)

- Some code gets used once or twice, then tossed.
- Sometimes production-level quality is not necessary.
- Every engineer needs some version of duct tape.
Two Programmers

Key: 
- writing code 
- running code 
- death of code

1.

2.

The Danger of Scripting Languages

- It can be very easy to hack things together this way
- You could do **everything** in Perl or Python
- Many NLP researchers do
- You might live to regret it
- Knowing when to use a quick and dirty solution is key.

Perl Programs
Anatomy of a Perl Program (Script)

- Name it `something.pl`
- Make it executable:
  ```
  > chmod 700 something.pl
  ```
- Run it:
  ```
  > ./something.pl
  ```
- Perl is an interpreted language (no compilation).

```
#!/usr/local/bin/perl
use mymodule;
# comments
# code goes here
sub myfunction {
# function code
}
```

"Hello, world!"

```perl
print "Hello, world!\n";
```

- Or
  ```perl
  print STDOUT "Hello, world!\n";
  ```

- Similar:
  ```perl
  print STDERR "Hello, alternate world!\n";
  open(FOO, ">file.txt");
  print FOO "Hello, disk world!\n";
  ```

Important data types in Perl

- Filehandle: `FOO`
- Scalar (bools, strings, ints, floats, references): `$bar`
  - Important one: `$0` (the current command)
- Array: `@things`
  - ordered list of scalars (for now)
  - important one: `@ARGV`
- Hash: `%mapping`
  - unordered set of (unique key, value) pairs
Perl Syntax and Control Structures

• Similar to C, but a little less strict.
  
  ```perl
  print $bar if($foo);
  if($fish) { print "fish"; }
  elsif($beast) { print "beast"; }
  else { die "no feast!" }
  foreach $x (@array) { ... }
  • also have: while, until, for
  ```

Basics of Strings

• When you refer to it, it commences to exist. Default value is "", the null string.

• Three kinds of quoting:
  
  ```perl
  $winner = 'calculate_winner.pl';
  $suspense = "The winner is: $winner!
  $s = "Just kidding! It's $winner!"
  ```

• Concatenation is easy:
  
  ```perl
  $c = $a . $b;
  ```

• Watch out for comparison:
  
  ```perl
  if($a eq $b) { ... } # ne, lt, gt
  ```

Basics of Numbers

• When you refer to it, it commences to exist. Default value is 0, the null value.

• Arithmetic is just like C, but Perl doesn’t worry about integers versus floats. Yes, you have ++ and friends.

• Perl generally knows what to do:
  
  ```perl
  $sleepers = '3'; # it’s a string
  print $sleepers + 1; # int, int, string!
  ```

• Comparison: ==, !=, <, >
Basics of Arrays

- When you refer to it, it commences to exist. Default value is an empty array (length 0).

```perl
@L = ("shirt", "pants", "trousers");
($first, $second, $third) = @L;
die unless($L[0] == $first);
for($i = 0; $i < scalar(@L); ++$i) {
    print $i, ": ", $L[$i];
}
($cat, $man) = ($man, $cat);  # swap!

- Also: push, pop, shift, unshift
- @ARGV
```

Basics of Hashes

- When you refer to it, it commences to exist. Default value is an empty array (length 0).

- It doesn’t matter to you that it’s a hash table. Think of it as an associative array.

```perl
%funeral="ashes", '@$HE$', "dust", 'DU$+");
$funeral{"song"} = "Swing Low";
foreach $x (@clothes) {
    $funeral{"color of $x"} = "black";
}
@k = keys %funeral;
```

Truth in Perl

- Things that are true:
  1. Strings, except for "" and "0".
  2. Numbers, except for 0.
  3. Lists that are not empty.
  4. Hashes that are not empty.
  5. Any references (I’m not going to talk about references today).

- Logical operators: &&  ||  ! (like C), also and  or  not  xor (like English)
Basics of File Output

- Open a file for writing:
  ```perl
  open(FOO, ">foo.txt")
  or die "couldn't open foo.txt!";
  ```

- Print stuff to the file:
  ```perl
  print FOO $fish[1], $fish[2], $fish[$red], $fish[$blue];
  ```

- Close the file:
  ```perl
  close(FOO);
  ```

Basics of File Input

- Most of your Perl programs will have this somewhere:
  ```perl
  while($line = <>) { ... }
  ```

- Or:
  ```perl
  open(FOO, "<input.txt");
  while($line = <FOO>) { ... }
  ```

- Also okay:
  ```perl
  while(<FOO>) { print $_; }
  ```

- Useful:
  ```perl
  while(chomp($line = <>)) { ... }
  ```

Calling Other Programs

- The simplest way to call another program:
  ```perl
  system("wc $file");
  ```

- Careful about paths!
  ```perl
  chomp($wordcount = `wc -w $file");
  ```

- Piping ...
Piping

```perl
open(PIPE, "| sort");
while(<>) {
    print PIPE rand, "\t", $_;
}
close PIPE;

open(PIPE, "sort $file |");
while(<PIPE>) {...}
```

• Pipes you can read and write to? Look up `IPC::Open2`.

Regexp

• A filter program:
  ```perl
  while($line = <FOO>) {
      if($line =~ m/http:/) {
          print $line; }
  }
  ```
  ```perl
  More terse:
  while(<FOO>) { print if m/http:/; }
  ```
  • Note that the match could be anywhere in the string.
  • To the extent that you master regexps, your Perl programs will become shorter and harder to read.

Regexp

• `m/a/` matches character a
• `m/abc/` matches the string abc
• `m/[abc]/` matches a, b, or c
• `m/[a-y]/` matches a, b, ..., x, or y
• `m/[a-zA-Z0-9]/` matches an alphanumeric character
• `m/\d/` matches a digit
• `m/\S/` matches a non-whitespace character
• `m/\s/` matches a whitespace character
Regexp examples:

- \bfoo\b matches foo or bar
- \bf\/ matches fo, foo, fooo, etc.
- \bf\+ matches fo, foo, fooo, etc.
- \bf\+ matches fo, foofoo, foofoofoo, etc.
- \bf\+? matches fo in its first, shortest occurrence

Regexp: Parentheses

\$
\text{\textasciitilde m/\((\d+)\s+(A-Z)\S+\S+\s+\+.\)/;}
$

- If there's a match, the above returns:
  - 1 (in a scalar context)
  - A list of the substrings corresponding to the (...) parts (in a list context)
- Also, the (...) parts go into \$1, \$2, etc.
- I like:
  \$(\$num, \$street) = m/\((\d+)\s+(\S+\s+\S+\s+.\)/;
- (What string is getting matched?)

Regexp: Substitution

- Manipulating strings is not fun in most languages, but it is in Perl.
- \$/serious matter/Serious Matter/g;
- \$/\$/g;
- \$/Prof\ . Smith/Superman/g;
- \$/(pepper|white) and (salt|black)/\2 and \1/g;

© Noah A. Smith 2011
Other Things to Say

• You can write functions, but then you have to think about variable scope. Unlike C, pretty much everything means something, and unintended side effects are common if you don’t know what you’re doing.

• Use descriptive variable names; you’ll be glad later.

• Functions I like:
  
  ```perl
  @L = split '/\s+/', $line;
  @L = sort @L;
  print join " ", @L;
  ```

Command-Line Perl

1. Evaluate some Perl.

• Use `-e` to type a small program on the command line.
  
  ```bash
  > perl -e 'print "hello, world!\n"';
  > perl -e 'print 8487 / 2398, "\n"';
  > perl -e 'for($i=1; $i <= 100; ++$i) { print "$i\n"; }' > numbers.txt
  > perl -e 'for($i=1; $i <= 100; ++$i) { print rand, "\n"; }' > random_numbers.txt
  ```

• Most often, I use this instead of the `calc` program.
2. Process data and then say something.

- The `-n` option says, “expect line-by-line input.”
  ```bash
  perl -ne 'print length, 
  
  • You can specify `BEGIN` and `END` blocks:
    ```bash
    perl -ne '$L += length; END {print "$L\n";}'
  ```
  - The `--a` option automatically splits on whitespace and puts the result in `@F`:
    ```bash
    perl -ane '$x += $F[3]; END {print "$x\n";}'
    perl -ne 'print unless ($. % 4 == 2);'
  ```
  - (Perl sugar: `$.` is the line number.)

3. Read and write lines.

- The `-p` option is just like `-n`, but says “print after each line.”
  ```bash
  perl -pe 's/\s+/ /g;'
  ```
- Really useful for small reformatting tasks:
  ```bash
  perl -pe 's/\s+//; s/\s+/ /g; s/\s+$/\n/;'
  ```
- Often useful with pipes:
  ```bash
  cat foo.txt | perl -pe '...' | sort | ...
  ```

-n, -a, and -p

```perl
# BEGIN block here
while(<>) {
    @F = split /\s+/; # if you use -a
    # your code here
    print; # if you use -p instead of -n
}  # END block here
```
Perl Summary

- I don’t know everything about Perl, but I know enough.
- What you don’t know won’t hurt you.
- There are more than one ways to do everything.
- Perl is great for:
  - Data munging (especially text)
  - Putting stuff in the right format
  - Tokenizing data
  - Regular expressions
- Perl is (in my opinion) not great for:
  - Complex data structures
  - Hard-core, bug-prone algorithms

Python Programs

- Name it something.py
- Make it executable:
  > chmod 700 something.py
- Run it:
  > ./something.py
- Python is an interpreted language (no compilation).
“Hello, world!”

```
print "Hello, world!"  # newline implied
print 'Hello, world!
print '''Hello, world!'''
```

---

**Important data types in Python**

- file
- bool
- int
- float
- str (character string)
- set
- list
- dict

---

**Python Syntax and Control Structures**

- No curly braces for code blocks; instead, indentation:

  ```python
  if True:
      print "yes, sir"
  elif False:
      print "no, way"
  else:
      print "no way, Jose"
  while x < 10:
      x += 1
  ```
Python's for ... in ...

```python
word = "foobar"
list = []
for c in word:
    list.append(c)
print list
# output: ['f', 'o', 'o', 'b', 'a', 'r']
for i in range(len(list)):
    print str(i) + "th character is", list[i]```

Basics of Strings

• Three kinds of quoting, all the same: single, double, triple
• Triple quotes let you run across multiple lines.
• Concatenation is easy:
  ```python
a = "foo"
b = "bar"
ab = a + b
```
• Comparison:
  ```python
  if a == b:
  ...
  elif a <= b:
  ...
  ```

Code in Strings

```python
classes = ['sophomore', 'junior', 'senior']
code = "for c in classes:
    print "We have some " + c + "s here."
"
exec(code)

force = "mass*acceleration"
print eval(force, {"mass":4, "acceleration": 12})```
Useful String Functions

```python
wlist = ['I', 'saw', 'her', 'duck']
print ' like '.join(wlist)

zlist = sentence.split(': ')
wlist = sentence.split()  # split on whitespace
d = data.splitlines(1)  # 1 means "keep the newlines"
```

Basics of Lists

- Lists are referred to using square brackets.
- The elements of a list can be of different types.
- 0-indexing like Perl, C, Java, etc.
  - But `mylist[-1]` means the last element!
- Slicing:
  - `mylist[:10]` (first 10, indexed from 0 to 9)
  - `mylist[10:]` (indices from 10 to the end)
  - `mylist[4:8]` (fifth through eighth elements)
  - `mylist[-5:-1]` (fifth-to-last through second to last)

More List Operations

```python
mylist.append(4)
mylist.extend(otherlist)
mylist.append(otherlist)  # watch out!
(mylist by only one element, which is a list)
mylist.insert(4, "newt")  # nothing is erased
mylist.remove("newt")
mylist.pop(0)  # default is -1
```
Basics of Dictionaries

funeral={"ashes":"@$HE$", "dust":"DU$+"}
funeral["song"] = "Swing Low"
for x in clothes:
    funeral["color of " + x] = "black"

k = funeral.keys();

Truth in Python

- Things that are true:
  1. Strings, except for "".
  2. Numbers, except for 0.
  3. Lists that are not empty.
  4. Dictionaries that are not empty.

- Logical operators: and or not (like English)

Basics of File Output

- Open a file for writing:
  foo = open("foo.txt", "w")
  if not foo:
      print "error opening foo.txt"

- Print stuff to the file:
  foo.write(mystring)
  foo.writelines(mylist)
  print >>foo, mystring

- Close the file:
  foo.close()
Basics of File Input

- Open the file for reading:
  ```python
  foo = open("input.txt", "r")
  ```
- Read the whole thing in:
  ```python
  everything = foo.read()
  ```
- My favorite:
  ```python
  for line in open("input.txt"):
  ```
- Alternately, split into lines:
  ```python
  mylist = foo.readlines()
  ```

Regexp

- They're less elegant than Perl, but still better (?) than Java.
  ```python
  import re
  myregexp = re.compile("http:"
  for line in foo:
    m = myregexp.match(line)
    if m:
      print line
  ```

Functions

- Python is a bit more function-friendly than Perl.
  ```python
  def obit(n, t, y=1921):
    print n, ", born in", t, "in", y ...
  ```
- You can call this lots of ways:
  ```python
  obit(\"Smith\", \"Washington, D.C.\", longago)
  obit(y=longago, n=\"Smith\", t=\"D.C.\")
  d = {\"n\":\"Smith\", \"t\":\"D.C.\", \"y\":longago}
  obit(**d)
  ```
More on Functions

• Cool lambda syntax: useful for map, filter
  
  def myfun(x, y):
      return 2 * x + y

• Scoping:
  • Global namespace, and one for each nested function you’re in.
  • Python starts most local and looks iteratively farther out to resolve names.
  • But: within a function, you can only modify local and global objects.

Python Summary

• I don’t know everything about Python, but I can read what I need to.
• What you don’t know won’t hurt you.
• Python is great for:
  • Text algorithms that have a little meat on them
  • Putting stuff in the right format
  • Tokenizing data
  • Building code libraries (e.g., NLTK)
  • XML
• Python is (in my opinion) not great for:
  • Super-fast code (though: pypy and C code integration)

Note to Object-Oriented People

• You can use Perl and Python as OO languages.
• I don’t.