Perl's Standard Library

Suggested reading:
Programming Perl, 3rd
Chp 29. Functions
Read up to "29.1 Perl Functions by Category", then skim the rest
Rich standard functionality

• Some built in, some available as modules you can request
• The distinction between the two is minor and sometimes arbitrary.
• eg Regular expressions
  – Built into Perl and Ruby
  – Standard module ("re") in Python
Perl specific

- When is doubt, look to C
- perldoc -f functionname
String manipulation

- chomp, chop, chr, crypt, hex, index, lc, lcfirst, length, oct, ord, pack, reverse, rindex, sprintf, substr, uc, ucfirst
Regular expressions

- pos, quotemeta, split, study
Numeric functions

• abs, atan2, cos, exp, hex, int, log, oct, rand, sin, sqrt, srand
Array and hash manipulation

• Arrays: grep, join, map, pop, push, reverse, shift, sort, splice, unpack, unshift
• Hashes: delete, each, exists, keys, values
Input and output

- binmode, close, closedir, die, eof, fileno, flock, format, getc, print, printf, read, readdir, rewinddir, say, seek, seekdir, select, syscall, sysread, sysseek, syswrite, tell, telldir, truncate, warn, write
Files and directories

- `-X`, `chdir`, `chmod`, `chown`, `chroot`, `fcntl`, `glob`, `ioctl`, `link`, `lstat`, `mkdir`, `open`, `opendir`, `readlink`, `rename`, `rmdir`, `stat`, `symlink`, `sysopen`, `umask`, `unlink`, `utime`

  - `-X` is a shorthand for: `-A -B -C -M -O -R -S -T -W -X -b -c -d -e -f -g -k -l -o -p -r -s -t -u -w -x -z
Processes

- alarm, exec, fork, getpgrp, getppid, getpriority, kill, pipe, qx// (AKA `backtick`), setpgrp, setpriority, sleep, system, times, wait, waitpid
Control flow

- caller, continue, die, do, dump, eval, exit, goto, last, next, redo, return, sub, wantarray
Time

• gmtime, localtime, time, times
Modules

- do, import, no, package, require, use
Objects

- bless, package, ref, tie, tied, untie, use
Low-level socket

- accept, bind, connect, getpeername, getsockname, getsockopt, listen, recv, send, setsockopt, shutdown, socket, socketpair
Networking info

- endprotoent, endservent, gethostbyaddr, gethostbyname, gethostent, getnetbyaddr, getnetbyname, getnetent, getprotobynumber, getprotoent, getservbyname, getservbyport, getservent, sethostent, setnetent, setprotoent, setservent
Fetching user and group info

- endgrent, endhostent, endnetent, endpwent, getgrent, getgrgid, getgrnam, getlogin, getpwent, getpwnam, getpwuid, setgrent, setpwent
sort: Sort an array

- Sorts by value of each character:
  - a-z, A-Z, 0-9
- my @sorted_array = sort @array;
Custom sort

• $a$ and $b$ are automatic

```perl
sub compare_numbers {
    if($a < $b) { return -1; }
    if($a > $b) { return 1; }
    return 0;
}
my @sorted_array = sort compare_numbers @array;
```
Custom sort

- Abbreviate for above:
- \( \leq \) compares numbers
- `cmp` compares strings

```perl
sub compare_numbers {
    { return $a <=> $b; }
}
my @sorted_array = sort compare_numbers @array;
```
Custom sort

• Shove the function inline

```perl
my @sorted_array = 
    sort { return $a <=> $b; } 
    compare_numbers @array;
```
Custom sort

• "return" is optional; the last result is automatically returned. And the last semicolon in a block is also optional:

```
my @sorted_array =
    sort { $a <=> $b }
    compare_numbers @array;
```
Sorting a hash

my @sorted_keys = sort keys %hash;

foreach my $key (sort keys %hash) {
    my $value = $hash{$key};
    # do something interesting here
}
Sorting a hash by value

my @students_by_score = sort

{ $students{$a} <=> $students{$b} }

keys %students;
grep

• Common problem: I only want some of this array.

my @temp_files;
foreach my $file (@files) {
    if($file =~ /\.*tmp\$/) {
        push @temp_files, $file;
    }
}
grep

- grep implements this more tersely. Sets $_ equal to one element at a time.

my @output = grep boolean-expression, @array;

my @output = grep /regular expression/, @array;

my @output = grep { function } @array;
grep

my @temp_files = grep /^[.tmp$]/, @files;
my @odd_numbers = grep $_ % 2, @numbers;
my @usable_files = grep {
  if( ! readable_file($_)) { return 0; } 
  if( ! writeable_file($_)) { return 0; } 
  return 1; }, @files;
map

• Common problem: walk an array, doing "something" to each item.
• eg Build new array of words in @words, but all lower case

```perl
my @words_lc;
foreach my $word (@words) {
    push @words_lc, lc($word);
}
```
map

• map implements this more tersely. Sets \$_ equal to one element at a time.
• Return the new, modified element
  – Return nothing to delete
  – Can return many to expand

```perl
my @words_lc = map { lc($_) } @words;
```
map

- Quick trick for turning an array into a set:

```perl
my %set = map { @$_ => 1 } @array;
```
More philosophy
Variable (and function)
names
Before

```perl
%hits;
while(my $line = <IN>) {
    my @fields = split(/ /, $line);
    $hits{$fields[6]}{$fields[10]}++;
}
```
After

```perl
%hits;
while(my $line = <IN>) {
    my @fields = split(/ /, $line);
    my($local_url, $refer_url) = ($fields[6], $fields[10]);
    $hits{$local_url}{$refer_url}++;
}
```
Before

```perl
foreach my $key (keys %hits) {
    print "$key\n";

    foreach my $key1 (keys %{$hits{$key}}) {
        print "  \$hits{$key}{$key1} ";
        print "$key1\n";
    }
}
```
After

```perl
foreach my $local_url (keys %hits) {
    print "$local_url\n";

    foreach my $refer_url (keys %{$hits{$local_url}}) {
        print "  \$hits{$local_url}{$refer_url} ";
        print "$local_url\n";
    }
}
```
A good variable or function name is frequently better than an explanatory comment
"If your code needs a comment to be understood, it would be better to rewrite it so it's easier to understand."

— "Notes on Programming in C" by Rob Pike

(But don't tell your TAs I told you this.)