Today’s Topics

• Sorting:
  • fc (fold case), hash sort

• Homework 4 Review

• More Perl:
  • Implicit Coercion
  • Conditionals
  • Loops

• On Thursday: Homework 5
  • first programming homework
Last Time:

• Arrays:
  • insertion and deletion from the ends

<table>
<thead>
<tr>
<th>shift/unshift</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>n-2</th>
<th>n-1</th>
</tr>
</thead>
</table>

Generalized form:

• splice ARRAY,OFFSET,LENGTH,LIST

Perl functions may have **side effects** and also **return values**

- **shift**
  Shifts the first value of the array off and returns it, shortening the array by 1 and moving everything down.
- **unshift**
  Does the opposite of a **shift**. Or the opposite of a **push**, depending on how you look at it. Prepends list to the front of the array, and returns the new number of elements in the array.
- **push**
  Treats ARRAY as a stack, and pushes the values of LIST onto the end of ARRAY.
- **pop**
  Pops and returns the last value of the array, shortening the array by one element.
Perl Arrays

Sorting:

• \{a cmp b\}
  • default (ASCII table order)

• \{a <=> b\}
  • numeric

• flip $a $b for reverse order
• enable function fc for Unicode casefolding

```perl
use feature 'fc';
@a = ("This", "is", "a", "test", "sentence");
@s = sort {fc($a) cmp fc($b)} @a;
print "@s\n";
```
Handling characters beyond ASCII ...

• Unicode encoding (utf-8)

UTF-8 - Wikipedia, the free encyclopedia
en.wikipedia.org/wiki/UTF-8

UTF-8 (UCS Transformation Format—8-bit) is a variable-width encoding that can
represent every character in the Unicode character set. It was designed for ...

use feature 'fc';
use utf8;
binmode STDOUT, "utf8";
my $a = "AbCdéÉ";
print uc($a), "\n";
print lc($a), "\n";
print fc($a), "\n";
bash-3.2$ perl case.perl
ABCDÉÉ
abcdéé
Handling characters beyond ASCII ...

• Note:

```perl
use feature 'fc';
my $a = "AbCdé";
print uc($a), "\n";
print lc($a), "\n";
print fc($a), "\n";
```

```bash
bash-3.2$ perl case2.perl
ABCDé
abcdé
abcdé
```

```perl
use feature 'fc';
use utf8;
use open qw(:std :utf8);
my $a = "AbCdé";
print uc($a), "\n";
print lc($a), "\n";
print fc($a), "\n";
```

```bash
bash-3.2$ perl case2.perl
ABCD\311\311
abcd\351\351
abcd\351\351
```

`open pragma`

`most general solution`

[https://perldoc.perl.org/open.html](https://perldoc.perl.org/open.html)
Perl Hashes

• Sorting with a hash

```
# this sorts the %age hash by value instead of key
# using an in-line function
@eldest = sort { $age{$b} <=> $age{$a} } keys %age;

# sort using explicit subroutine name
sub byage {
    $age{$a} <=> $age{$b};  # presuming numeric
}
```

Usage:
```
sort byage keys %age
```

```
use strict;
my %fruitColor = ("apple", "red", "banana", "yellow", "kiwi", "green");

Let's sort fruits by color, ascending, descending...
```
Homework 4 Review

Stanford Parser: I made her duck

Parse

(ROOT
  (S
    (NP (PRP I))
    (VP (VBD made)
      (NP (PRP$ her) (NN duck)))
    ( .  . )))

Universal dependencies

  nsubj(made-2, I-1)
  root(ROOT-0, made-2)
  nmod:poss(duck-4, her-3)
  dobj(made-2, duck-4)

Relevant tags:

• Phrasal:
  • S Sentence; NP Noun Phrase; VP Verb Phrase

• Part of Speech (POS):
  • PRP Personal Pronoun; VBD Verb Past; PRP$ Possessive Personal Pronoun; NN Common Noun; . .

• Dependency links:
  • nsubj; nmod:poss; dobj
Homework 4 Review

Lisp S-EXP syntax:

```
(ROOT
  (S
    (NP (PRP I))
    (VP (VBD made)
      (NP (PRP$ her) (NN duck))
      (.)
  ))
```

Tree representation:
Homework 4 Review

(1.5) I cooked waterfowl for her.  
(1.6) I cooked waterfowl belonging to her.  
(1.7) I created the (plaster?) duck she owns.  
(1.8) I caused her to quickly lower her head or body.  
(1.9) I waved my magic wand and turned her into undifferentiated waterfowl.

❌ Verb: duck  
❌ Ditransitive verb: duck  
❌ cf. I baked [John] [a cake]
Homework 4 Review

Winograd’s SHRDLU system:
Move the red block on top of the smaller green one

Tree Representation:
PP modifies VP (not NP)
Homework 4 Review

Another interpretation:

Stanford parse:

Universal dependencies, enhanced

root(ROOT-0, Move-1)
det(block-4, the-2)
amod(block-4, red-3)
dobj(Move-1, block-4)
case(one-11, on-5)
mwe(on-5, top-6)
mwe(on-5, of-7)
det(one-11, the-8)
dep(green-10, smaller-9)
amod(one-11, green-10)
nmod:on_top_of(Move-1, one-11)
Implicit Coercion

Perl features implicit coercion of data types

**Example:**
- the following program prints *Equal!*
- `==` is the numeric equality comparison operator

```perl
my $one_string = "1";
my $one_number = 1;
if ($one_string == $one_number) {
    print "Equal!\n"
} else {
    print "Different!\n"
}
```

**Example:**
- the following program prints *3 is my number*
- `'` is the string concatenation operator

```perl
my @a = qw(one, two, three);
my $string = @a." is my number";
print "$string\n";
```
Implicit Coercion

- print "4" * 4
  16

- print "4" x 4
  4444

- @a = (4) x 4
  (4, 4, 4, 4)

("x" is the repetition operator)

(in list context)
Conditionals and Looping

• **Conditionals**
  
  • if ( @a < 10 ) { print “Small array\n” } else {print “Big array\n” }
  
  • **Note**: @a here is a scalar = size of array
  
  • unless (@a > 10) { print “@a\n” }
  
  • **Note**: if size of array $a$ is $\leq 10$, it prints the contents of array $a$
Conditionals and Looping

- **Numeric comparison**
  1. `==` equality
  2. `!=` inequality
  3. `<` less than
  4. `>` greater than
  5. `<=` less than or equal
  6. `>=` greater than or equal

- **String comparison**
  1. `eq` equality
  2. `ne` inequality
  3. `lt` less than
  4. `gt` greater than
  5. `le` less than or equal
  6. `ge` greater than or equal

(Why do we have separate numeric and string comparisons? Because we don’t have special variable types, and Perl needs to know whether to sort numerically (where 99 is less than 100) or alphabetically (where 100 comes before 99).

- **Boolean logic**
  1. `&&` and
  2. `||` or
  3. `!` not
Conditionals and Looping

• Looping over arrays
  • %fruits = qw(apple green orange orange lemon yellow);
  • foreach $f (keys %fruits) { print $f, " => ",
    $fruits{$f}, "\n" }

gives output:
  • lemon => yellow
  • apple => green
  • orange => orange
  • Note: keys %fruits = ("lemon", "apple", "orange") is an array
ARGV

1 print "Index of last argument: $#ARGV\n";
2 foreach $a (@ARGV) {
3 print "$a\n";
4}

Command line:
perl argv.perl 1 2 3
General Looping

- **while**
  ```
  1. while ( condition ) {
  2.     ...
  3. }
  ```

  There's also a negated version, for the same reason we have **unless**:
  ```
  1. until ( condition ) {
  2.     ...
  3. }
  ```

- **for**
  Exactly like C:
  ```
  1. for ($i = 0; $i <= $max; $i++) {
  2.     ...
  3. }
  ```

  The C style for loop is rarely needed in Perl since Perl provides the more friendly list scanning **foreach** loop.