LING/C SC/PSYC 438/538

Lecture 8

Sandiway Fong

Today's Topics

- Perl hash (*recap*) and Python dict comparison
- Perl anonymous array […]
- Part of Speech tagset example
- Homework 6
  - has two themes: part 1 and part 2

Perl Hashes

perlintro
Perl 5 version 12.1 documentation

Go to top • Download PDF
Show page index • Show recent pages
Search

Home > Overview > perlintro

- **Hashes**

A hash represents a set of key/value pairs:

```
1.  my %fruit_color = ("apple", "red", "banana", "yellow");
```

fat comma

You can use whitespace and the => operator to lay them out more nicely:

```
1.  my %fruit_color = (
2.      apple  => "red",
3.      banana => "yellow",
4.  );
```

To get at hash elements:

```
1.  $fruit_color{"apple"};          # gives "red"
```

You can get at lists of keys and values with **keys()** and **values()**.

```
1.  my @fruits = keys %fruit_colors;
2.  my @colors = values %fruit_colors;
```

Hashes have no particular internal order, though you can sort the keys and loop through them.

Just like special scalars and arrays, there are also special hashes. The most well known of these is %ENV which contains environment variables. Read all about it (and other special variables) in perlvar.

- aka `dict` in Python

Notation:

1. Perl array: @ […]
2. Perl hash: % {…}

Perl Hashes

- Compare (@) arrays and (%) hashes
  - **arrays** are indexed using integers (0, 1, 2, 3,…) as **keys**
  - **hashes** are like arrays with *user-defined* indexing (*aka* keys)
    aka associative array or hash table or dict (Python)

  - Initialization (use list notation (or *shortcut*): ***round brackets and commas***)
    - `@a = ("zero", "one", "two", "three", "four");`
    - `%h = ("zero", 0, "one", 1, "two", 2, "three",  3, "four", 4);`
      (*key/value pairs*)

  - Access to individual elements
    (*square brackets vs. curly braces*)
    - `$a[1]` "one"
    - `$h{zero}` 0

no quotes needed for strings
'zero' "zero" cf. Python

Shortcut (*no quotes, no commas needed*):

```
1.    qw(foo bar baz)
```

is semantically equivalent to the list:

```
1.    "foo", "bar", "baz"
```

Perl Hashes

- Compare (@) arrays and (%) hashes
  - `@a = ("zero", "one", "two", "three", "four");`
  - `%h = (zero => 0, one => 1, two => 2, three => 3, four => 4);`
  - `h = {'zero':0, 'one':1, 'two':2, 'three':3, 'four':4}` (*Python*)
- **output**
  - `print @a` `zeroonetwothreefour`
  - `print "@a"` `zero one two three four`
  - `print %h` `two2three3one1zero0four4` (note: *order*)
  - `print "%h"` `%h` (*no interpolation done*)
  - `h` (*Python*)
  - `{'zero': 0, 'one': 1, 'two': 2, 'three': 3, 'four': 4}`

More on Hash tables

- Looping over the **array** of keys:

```
1%fruitColor = qw(apple red banana yellow kiwi green);¶
2for (keys %fruitColor) {¶
3  print "$_ => $fruitColor{$_}\n"¶
4}¶
```

www.freeimages.com

- Output:

```
$ perl –e '%fruitColor = qw(apple red banana yellow kiwi green);
for (keys %fruitColor) {print "$_ => $fruitColor{$_}\n"}'
kiwi => green
banana => yellow
apple => red
```

- Notice: order is not guaranteed! (*Implementation dependent*)

More on Hash tables

- Unique key constraint:

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

```
perl -e '%h = qw(apple red banana yellow
apple green); print $h{apple}, "\n"'
green
```


```
>>> fruitColor = {'apple':'red', 'banana':'yellow', 'kiwi':'green',
'apple':'green'}
>>> fruitColor['apple']
'green'
```

Perl `each` and Python `.items()`

- Use a for-loop and method `.items()`: k,v is a tuple

```
>>> knights = {'gallahad': 'the pure', 'robin': 'the brave'}
>>> for k, v in knights.items():
...     print(k, v)
...
gallahad the pure
robin the brave
```

```
perl -e '%knights = (gallahad => "the pure", robin => "the brave");
while (($key, $val) = each %knights) { print "$key $val\n"}'
robin the brave
gallahad the pure
```

More on Hash tables

```
%fruitColor = qw(apple red banana
yellow kiwi green)
```

- Delete key/value entry:
  - `delete $fruitColor{apple}`
- Exists (condition):
  - `exists $fruitColor{kiwi}`
  `1 (true)`
  - `exists $fruitColor{orange}`
  `"" (false)`
- lookup:
  - `$fruitColor{orange}`
  `""`

```
fruitColor = {'apple':'red',
'banana':'yellow', 'kiwi':'green'}
```

- Python:
  - `del fruitColor['apple']`
- Python:
  - `'kiwi' in fruitColor`
  `True`
  - `'orange' in fruitColor`
  `False`
  - `fruitColor['orange']`

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'orange'
```

Counter objects have a dictionary interface except that they return a zero count for missing items instead of raising a `KeyError`:

Python `dict`

- Dictionary order preservation depends on the version of Python used …

New dict implementation

Python 3.6 onwards

The dict type now uses a "compact" representation based on a proposal by Raymond Hettinger which was first implemented by PyPy. The memory usage of the new `dict()` is between 20% and 25% smaller compared to Python 3.5.

The order-preserving aspect of this new implementation is considered an implementation detail and should not be relied upon (this may change in the future, but it is desired to have this new dict implementation in the language for a few releases before changing the language spec to mandate order-preserving semantics for all current and future Python implementations; this also helps preserve backwards-compatibility with older versions of the language where random iteration order is still in effect, e.g. Python 3.5).

More on Hash tables

- Hash slice:
  - `perl -e '%h = qw(mon Monday tue Tuesday wed Wednesday thu Thursday fri Friday sat Saturday sun Sunday); print "@h{(sat, sun)}\n"'`
  - `Saturday Sunday`
- Delete hash:
  - `%h = ()`
  - `h = {}` (Python)

Python `dict`

The `dict()` constructor builds dictionaries directly from sequences of key-value pairs:

```
>>> dict([('sape', 4139), ('guido', 4127), ('jack', 4098)])
{'sape': 4139, 'guido': 4127, 'jack': 4098}
```

In addition, dict comprehensions can be used to create dictionaries from arbitrary key and value expressions:

```
>>> {x: x**2 for x in (2, 4, 6)}
{2: 4, 4: 16, 6: 36}
```

similar syntax to a list comprehension
[… for *var* in *ITERABLE*]

When the keys are simple strings, it is sometimes easier to specify pairs using keyword arguments:

```
>>> dict(sape=4139, guido=4127, jack=4098)
{'sape': 4139, 'guido': 4127, 'jack': 4098}
```

Python Part-of-Speech `dict`

- likes is both a verb (3.sg.present) and a noun (plural)

  He has his *likes* and *dislikes* as well as we. (OED: 1735)

Facebook: 2004

4. In the context of social media: an expression of approval or support made by clicking on a particular icon. **2009–**

2009 FriendFeed should implement a notifications tab ala Facebook at the bottom for any 'likes' or new comments.
@RuleOfThum 21 January in *twitter.com* (O.E.D. Archive)

2011 The Refresh campaign..got three and a half million 'likes' on Pepsi's Facebook page.
New Yorker 16 May 68/1 ...

2014 Your likes on Facebook say so much about you.
T. Payton & T. Claypoole, *Privacy in Age of Big Data* 85 ...

Show fewer quotations

Python Part-of-Speech `dict`

```
>>> pos
{'likes': ['n', 'v'], 'car': ['n'], 'apple': ['n'], 'smiled': ['v']}
>>> for word, tag in pos.items():
...     if 'n' in tag:
...             print(word)
...
likes
car
apple
```

All values are lists
Advantage: simplifies the code

Trade-off

```
>>> pos
{'car': 'n', 'likes': ['n', 'v'], 'apple': 'n', 'smiled': 'v'}
>>> for word, tag in pos.items():
...     if tag == 'n' or 'n' in tag:
...             print(word)
...
car
likes
apple
```

Values are lists or a string
Advantage: simpler-looking dict

can be simplified because

```
>>> 'n' in 'n'
True
>>>
```

Python Part-of-Speech `dict`

- Works too!

```
{'car': 'n', 'likes': ['n', 'v'], 'apple': 'n', 'smiled': 'v'}
>>> for word, tag in pos.items():
...     if 'n' in tag:
...             print(word)
...
car
likes
apple
>>>
```

Less transparent:

- relies on a Python-particular quirk …
- and doesn't always work!

```
>>> 'abc' in 'abc'
True
>>> 1 in 1
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: argument of type 'int' is not iterable
```

Perl Part-of-Speech hash

- `[…]` = reference to an array (scalar)
- `@$refname` dereferences the scalar reference (gets you the array)
- in scalar context `grep` *`block`* `array` returns number of matches:
  - `perl -e '%pos=(likes => ["n","v"], car=>["n"], smiled=>["v"]); while(($word,$tagset)=each %pos){print "$word\n" if grep {$_ eq "n"} @$tagset}'`

    ```
    car
    likes
    ```
- in array context, `grep` *`block`* `array` returns the array of matches:
  - `perl -e '@a = qw(1 3 5 7 9 11 13 15 17); @lt10 = grep {$_ < 10} @a; print "@lt10\n"'`

    ```
    1 3 5 7 9
    ```

Python `dict`

- function zip() pairs up elements from two lists into an iterable

```
>>> questions = ['name', 'quest', 'favorite color']
>>> answers = ['lancelot', 'the holy grail', 'blue']
>>> for q, a in zip(questions, answers):
...     print('What is your {0}?  It is {1}.'.format(q, a))
...
What is your name?  It is lancelot.
What is your quest?  It is the holy grail.
What is your favorite color?  It is blue.
```

Homework 6: Part 1

- Well-known English spelling rule:
  - I before E except after C
- Orthography:
  - *receive* (digraph *ie*)
  - science (I before E after C: exception to the rule, type 1)
  - foreign (E before I not after C: exception to the rule type 2)

Homework 6: Part 1

- Adapt the code from the Homework 5 review to find exceptions in the *n*`letters.txt` files for n ≥ 3.
  - type 1: make use of the Perl index function (*see next slide*)
  - type 2: use a regex condition match `if ($_ =~ /[^C]EI/)`
    - =~ means match pattern **/…/**
    - `[^C]` means not a C character, so
    - `[^C]EI` means not a C followed by EI
- How many exceptions did you find for types 1 and 2 for the *n*`letters.txt` files?

index STR,SUBSTR,POSITION

index STR,SUBSTR

```
perl -e '$s="license"; print index($s, "ice"), "\n"'
1
```

The index function searches for one string within another, but without the wildcard-like behavior of a full regular-expression pattern match. It returns the position of the first occurrence of SUBSTR in STR at or after POSITION. If POSITION is omitted, starts searching from the beginning of the string. POSITION before the beginning of the string or after its end is treated as if it were the beginning or the end, respectively. POSITION and the return value are based at zero. If the substring is not found, `index` returns -1.

Find characters or strings:

```
index("Perl is great", "P");     # Returns 0
index("Perl is great", "g");     # Returns 8
index("Perl is great", "great"); # Also returns 8
```

Attempting to find something not there:

```
index("Perl is great", "Z");     # Returns -1 (not found)
```

Using an offset to find the *second* occurrence:

```
index("Perl is great", "e", 5);  # Returns 10
```

Homework 6: Part 2

Disemvoweling, **disemvowelling** (see doubled *L*), or **disemvowelment** of a piece of alphabetic text is rewriting it with all the vowel letters removed.[1][*full citation needed*] This original sentence:

> The quick brown fox jumps over the lazy dog

would, after being disemvowelled, look like this:

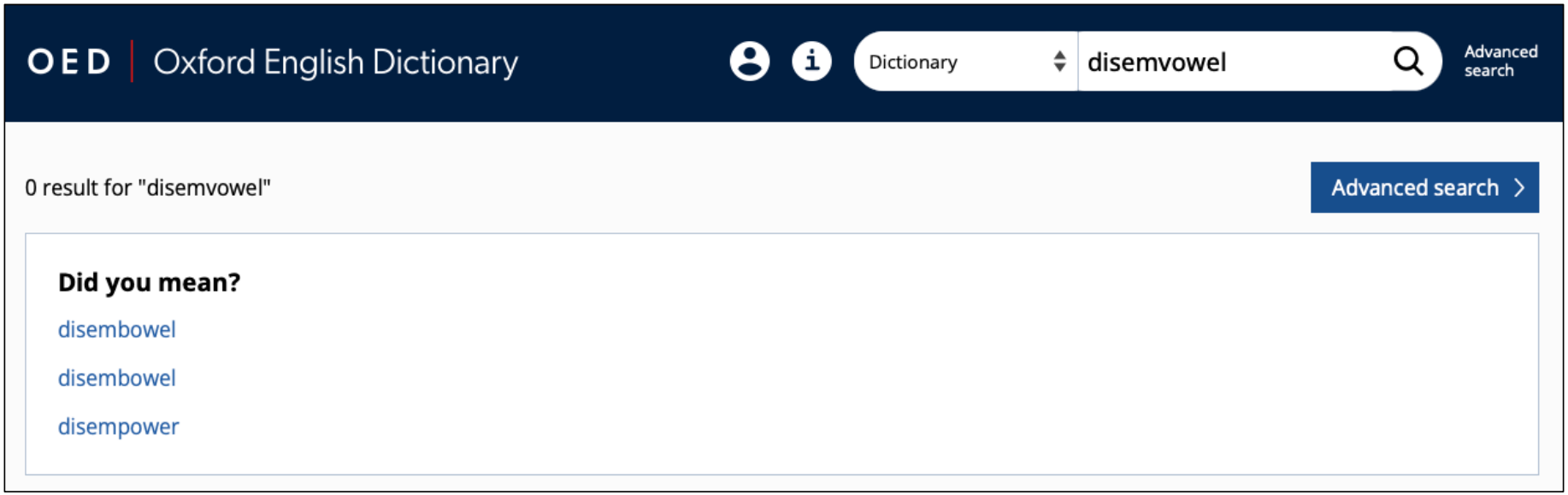
> Th qck brwn fx jmps vr th lzy dg

Disemvoweling is a common feature of SMS language[1] as disemvoweling requires little cognitive effort[*citation needed*] to read, so it is often used where space is costly. The word *disemvoweling* is a portmanteau combining *vowel* and *disembowel*.[1]

The word was used with precisely this meaning in the 1939 novel *Finnegans Wake*

https://en.wikipedia.org/wiki/Disemvoweling

Homework 6: Part 2



Background

—No more than Richman's periwhelker.

—Nnn ttt wrd?

—Dmn ttt thg.

—A gael galled by scheme of scorn? Nock?

—Sangnifying nothing. Mock!

—*Fortitudo eius rhodammum tenuit?*

—Five maim! Or something very similar.

—I should like to euphonise that. It sounds an isochronism. Secret speech Hazelton and obviously disemvowelled. But it is good laylaw too. We may take those wellmeant kicks for free granted, though *ultra vires*, void and, in fact, unnecessarily so. Happily you were not quite so successful in the process verbal whereby you would sublimate your blepharospasmockical suppressions, it seems?

—What was that? First I heard about it.

Finnegan's Wake by James Joyce
Book 3: Episode 3

disemvowelled

- wordplay: *periwhelker* = periwinkle
- how about?
  - Nnn ttt wrd? (*Not a word*)
  - Dmn ttt thg. (*Damn that thing*)

clemsonhgic.wpengine.com/

Background

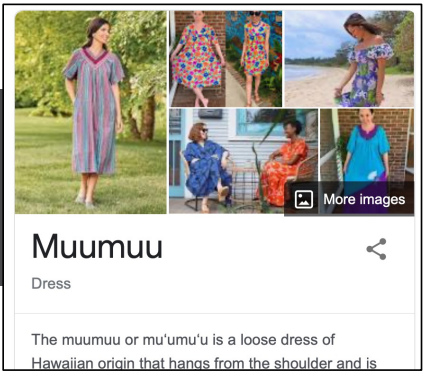
Where Have All the Vowels Gone?

Consider the muumuu.

By John Williams • Dec. 29, 2018

- https://www.nytimes.com/2018/12/29/style/vowels-no-more.html
  - tech companies like **Tumblr** and **Flickr** arrived on the scene, dropping e's both for distinctiveness and because the altered names made it easier to trademark, claim domain names on the internet and conduct other practical business.

After all, there are words that can hardly do without them: muumuu, audio and oboe, just to queue up a few. One cannot text someone "b" and expect them to know one is referring to an oboe.



Homework 6: Part 2

- Read https://en.wikipedia.org/wiki/Disemvoweling
- Part A:
  - write a Perl program to remove vowels *a, e, i, o, u* from words typed into the command line. (Don't worry about *y*.)
- Hint: use `split`, read the documentation (*see next slide*)
- Possible template for your program (*see slide ahead*)
- **Example:**

```
perl hw6.perl quick brown fox
qck brwn fx
```

Homework 6: Part 2

split /PATTERN/,EXPR,LIMIT

split /PATTERN/,EXPR

split /PATTERN/

split

Splits the string EXPR into a list of strings and returns the list in list context, or the size of the list in scalar context. (Prior to Perl 5.11, it also overwrote `@_` with the list in void and scalar context. If you target old perls, beware.)

If only PATTERN is given, EXPR defaults to `$_`.

https://perldoc.perl.org/functions/split

uses the `b` in `'abc'` as a separator to produce the list ("a", "c"). However, this:

```
my @x = split(//, "abc"); # ("a", "b", "c")
```

uses empty string matches as separators; thus, the empty string may be used to split EXPR into a list of its component characters.

Homework 6: Part 2

File: `hw6.perl`

```
1my  %vowel = qw(a 1 e 1 i 1 o 1 u 1 A 1 E 1 I 1 O 1 U 1);
2foreach $word (@ARGV) {
3
4
5
6
7
8
9}
10print "\n"
```

- Simple Recipe (*but not resulting in the shortest program*)
  - Take each word from the command line, e.g. *rabbit*
    - Split it into characters, e.g. *r a b b i t*
    - for each character, print it if it's not a vowel
  - at the end of a word, print a space
- at the end of all the words, print a newline
- Could make use of hash `%vowel`:
  - suppose `$char` is a consonant or a vowel, what happens with `$vowel{$char}` ?

Homework 6: Part 2

- Part B:
  - Modify your program for part A to **not** delete leading vowels
  - Example:
    - `If a sentence is unreadable`
    - `f sntnc s nrdbl` (*leading vowels deleted*)
    - `If a sntnc is unrdbl` (*more legible*)

Homework 6

- Email to me: sandiway@arizona.edu
- Subject: 438/538 Homework 8: *YOUR NAME*
- Submit code and example runs in one PDF file
  - *submit partial code if you cannot complete it*
- Due Sunday midnight: reviewed next Tuesday