

LING/C SC/PSYC 438/538

Lecture 6

Prof. Sandiway Fong

Today's Topics

- `perlintro`:
 - `@ARGV` array and other array operations
 - equality
 - coercion
 - useful functions on strings: e.g. `reverse`
 - `file input/output` (I/O)
- Homework 5 (**due Sunday midnight**)
 - *everything you need is in Lectures 5 & 6*

Perl Arrays

- Built-in arrays:
 - @ARGV (command line arguments; coercion possible)
 - \$ARGV[0] (1st argument)
 - \$0 (program name)
 - @_ (sub(routine) arguments)
- **Example:** myprog.perl

```
1 print "\$0:$0\n";  
2 print "$ARGV[0]\n";  
3 print $ARGV[0]+$ARGV[0], "\n";  
4 print $ARGV[0].$ARGV[0], "\n";
```

```
[ling538-26$ perl myprog.perl  
$0:myprog.perl  
  
0  
  
[ling538-26$ perl myprog.perl 1  
$0:myprog.perl  
1  
2  
11  
[ling538-26$ perl myprog.perl 1 2  
$0:myprog.perl  
1  
2  
11  
ling538-26$
```

From C (originally)

The C Programming Language, Kernighan & Ritchie (1978; 1988)

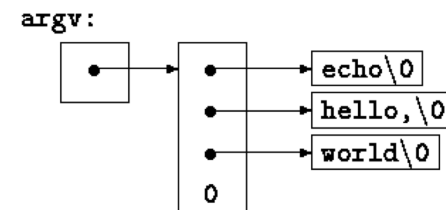
5.10 Command-line Arguments

In environments that support C, there is a way to pass command-line arguments or parameters to a program when it begins executing. When `main` is called, it is called with two arguments. The first (conventionally called `argc`, for argument count) is the number of command-line arguments the program was invoked with; the second (`argv`, for argument vector) is a pointer to an array of character strings that contain the arguments, one per string. We customarily use multiple levels of pointers to manipulate these character strings.

The simplest illustration is the program `echo`, which echoes its command-line arguments on a single line, separated by blanks. That is, the command

```
echo hello, world
prints the output
hello, world
```

By convention, `argv[0]` is the name by which the program was invoked, so `argc` is at least 1. If `argc` is 1, there are no command-line arguments after the program name. In the example above, `argc` is 3, and `argv[0]`, `argv[1]`, and `argv[2]` are "echo", "hello,", and "world" respectively. The first optional argument is `argv[1]` and the last is `argv[argc-1]`; additionally, the standard requires that `argv[argc]` be a null pointer.



```
#include <stdio.h>

/* echo command-line arguments; 1st version */
main(int argc, char *argv[])
{
    int i;

    for (i = 1; i < argc; i++)
        printf("%s%s", argv[i], (i < argc-1) ? " " : "");
    printf("\n");
    return 0;
```

Python argv

- **Essential details:**

- `import sys`
- `sys.argv` (list of command arguments as strings)
- `sys.argv[0]` (Python script name)
- `sys.argv[1]` (1st argument)

- **Example:**

`myprog.py`

```
1import sys
2print("argv[0]:" + sys.argv[0])
3print(sys.argv[1] + sys.argv[1])
```

```
[Machine$ python3 myprog.py 1
argv[0]:myprog.py
11
Machine$
```

- use `int(sys.argv[1])` to convert string into an integer

Perl Arrays

- Built-in functions:

- `sort @ARRAY; reverse @ARRAY;`
- `push @ARRAY, $ELEMENT; pop @ARRAY;` (operates at right end of array)
- `shift @ARRAY; unshift @ARRAY, $ELEMENT;` (left end of array)
- `splice @ARRAY, $OFFSET, $LENGTH, $ELEMENT`
 - *\$ELEMENT above can be @ARRAY*

- **Python:**

- `sorted(array)` (new array), `array.sort()` (modify *array*), `array.reverse()`
- **NO push** (use `array.append()` instead), `array.pop()`
- **No shift/unshift** etc... (but can use slicing and concatenation)

perlintro: Perl Arrays

- **shift ARRAY** Similar to pop/push, but operates at the left end of the array
- **shift**
Shifts the first value of the array off and returns it, shortening the array by 1 and moving everything down. If there are no elements in the array, returns the undefined value. If ARRAY is omitted, shifts the `@_` array within
- **unshift ARRAY,LIST**
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.

Python doesn't have these defined but can be simulated via slicing and concatenation:

```
array[1:]  
list + array
```

perlintro: Perl Arrays

- **splice ARRAY,OFFSET,LENGTH,LIST**

- **splice ARRAY,OFFSET,LENGTH**
- **splice ARRAY,OFFSET**
- **splice ARRAY**

Removes the elements designated by OFFSET and LENGTH from an array, and replaces them with the elements of LIST, if any. In list context, returns the elements removed from the array. In scalar context, returns the last element removed, or undef if no elements are removed. The array grows or shrinks as necessary. If OFFSET is negative then it starts that far from the end of the array. If LENGTH is omitted, removes everything from OFFSET onward. If LENGTH is negative, removes the elements from OFFSET onward except for -LENGTH elements at the end of the array. If both OFFSET and LENGTH are omitted, removes everything. If OFFSET is past the end of the array and a LENGTH was provided, Perl issues a warning, and splices at the end of the array.

The following equivalences hold (assuming `$#a >= $i`)

1.	<code>push(@a,\$x,\$y)</code>	<code>splice(@a,@a,0,\$x,\$y)</code>
2.	<code>pop(@a)</code>	<code>splice(@a,-1)</code>
3.	<code>shift(@a)</code>	<code>splice(@a,0,1)</code>
4.	<code>unshift(@a,\$x,\$y)</code>	<code>splice(@a,0,0,\$x,\$y)</code>
5.	<code>\$a[\$i] = \$y</code>	<code>splice(@a,\$i,1,\$y)</code>

Equality

Equality	Numeric	String
Equal	<code>==</code>	<code>eq</code>
Not Equal	<code>!=</code>	<code>ne</code>
Comparison	<code><=></code>	<code>cmp</code>
Relational	Numeric	String
Less than	<code><</code>	<code>lt</code>
Greater than	<code>></code>	<code>gt</code>
Less than or equal	<code><=</code>	<code>le</code>
Greater than or equal	<code>>=</code>	<code>ge</code>

returns 1 for true and "" for false

- **Conditionals:**

```
if ( comparison ) { statement }  
  else { statement }  
  elsif ( comparison ) { statement }
```

- **Example:**

- `if ( @a < 10 ) { print "Small array\n" } else { print "Big array\n" }`
- **Note:** @a here is in a **scalar context** = size of array

- **Example:**

- `unless ( @a > 10 ) { print "@a\n" }`
- if size of array *a* is ≤ 10 , it prints the contents of array *a*

Equality

- **Boolean logic**

1.	&&	<u>and</u>
2.		<u>or</u>
3.	!	<u>not</u>

Perl equality (numeric and string)

- What does **eq** do?

returns 1 for true and "" for false

- `perl -e '$x = "windy" eq "Windy"; print "$x\n"'`
- `perl -e 'use feature fc; $x = "windy" eq fc("Windy"); print "$x\n"'`
 - 1
- `perl -e '$x = "windy" eq "windy"; print "$x\n"'`
 - 1
- `perl -e '$x = 0.0 eq 1; print "$x\n"'`
- `perl -e '$x = 0.0 eq 0; print "$x\n"'` ← implicit coercion
 - 1
- `perl -e '$x = 0.0 eq "0"; print "$x\n"'`
 - 1
- `perl -e '$x = 0.0 eq "windy"; print "$x\n"'`

Equality	Numeric	String
Equal	<code>==</code>	<code>eq</code>
Not Equal	<code>!=</code>	<code>ne</code>
Comparison	<code><=></code>	<code>cmp</code>
Relational	Numeric	String
Less than	<code><</code>	<code>lt</code>
Greater than	<code>></code>	<code>gt</code>
Less than or equal	<code><=</code>	<code>le</code>
Greater than or equal	<code>>=</code>	<code>ge</code>

Perl equality (numeric and string)

- What does **==** do?

- perl -e '\$x = 0.0 == 1; print "\$x\n"'
- perl -e '\$x = 0.0 == 0; print "\$x\n"'
- 1
- perl -e '\$x = 0.0 == "0"; print "\$x\n"'
- 1
- perl -e '\$x = 0.0 == "windy"; print "\$x\n"'
- 1

Equality	Numeric	String
Equal	==	eq
Not Equal	!=	ne
Comparison	<=>	cmp
Relational	Numeric	String
Less than	<	lt
Greater than	>	gt
Less than or equal	<=	le
Greater than or equal	>=	ge

Perl equality (numeric and string)

- Turn on warnings:

```
$ perl -e 'use warnings; "windy" == 0'
```

```
Useless use of numeric eq (==) in void context at -e line 1.
```

```
Argument "windy" isn't numeric in numeric eq (==) at -e line 1.
```

- Data type testing:

- use Scalar::Util "looks_like_number";

```
$ perl -e 'use Scalar::Util "looks_like_number"; print  
looks_like_number($ARGV[0]), "\n"' 1.23
```

```
1
```

```
$ perl -e 'use Scalar::Util "looks_like_number"; print  
looks_like_number($ARGV[0]), "\n"' windy
```

```
$
```

More Implicit Coercion

- Example:

- the following program

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

- prints

3 is my number

- Note: . (*period*) is the string concatenation operator (*scalar context*)

Note: qw = quoted word, cf.
("one", "two", "three")

Implicit Coercion

- perl -e 'print "4" * 4, "\n"'
16

```
$ python3
>>> print("4"*4)
4444
>>>
```

- perl -e 'print "4" x 4, "\n"'
4444

("x" is the repetition operator)

- perl -e '@a = (4) x 4; print "@a\n"'
4 4 4 4

(list context)

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

General Looping: while and for

- **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:

Python: use `range(start, end, step)` instead. Or Numpy `arange()`

```
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.

General Looping: foreach

The `foreach` keyword is actually a synonym for the `for` keyword, so you can use either. If VAR is omitted, `$_` is set to each value.

```
1.     for (@ary) { s/foo/bar/ }
2.
3.     for my $elem (@elements) {
4.         $elem *= 2;
5.     }
6.
7.     for $count (reverse(1..10), "BOOM") {
8.         print $count, "\n";
9.         sleep(1);
10.    }
11.
12.    for (1..15) { print "Merry Christmas\n"; }
```

`$_` implicit variable

General Looping

- `perl -e 'for (@ARGV) {print $_ * $_, " "}' 1 2 3 4 5`
`1 4 9 16 25`

Perl list ranges

```
1.  for (1 .. 1_000_000) {  
2.      # code  
3.  }
```

iterates setting `$_` (*the default variable*)
from 1, 2, .., 1000000

```
[ling538-19$ perl -le 'for (1..10) {print}'  
1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
ling538-19$ █
```

Python equivalent:

- `for i in range(1,1000001):`
 # code

```
1.  @alphabet = ("A" .. "Z");
```

to get all normal letters of the English alphabet, or

```
1.  $hexdigit = (0 .. 9, "a" .. "f")[$num & 15];
```

```
[ling538-19$ perl -le '@a = (1..10); print "@a"  
1 2 3 4 5 6 7 8 9 10  
ling538-19$ █
```

split

- <https://perldoc.perl.org/functions/split.html>
- Compare:
 - `@a = split " ", "this is a sentence."`
 - `@a = split //, "this is a sentence."`
- **Exercise:** what is the size of array @a?

Perl: useful string functions

Functions for SCALARs or strings

- chomp - remove a trailing record separator from a string
- chop - remove the last character from a string
- chr - get character this number represents
- crypt - one-way passwd-style encryption
- hex - convert a string to a hexadecimal number
- index - find a substring within a string
- lc - return lower-case version of a string
- lcfirst - return a string with just the next letter in lower case
- length - return the number of bytes in a string
- oct - convert a string to an octal number
- ord - find a character's numeric representation
- pack - convert a list into a binary representation
- q/STRING/ - singly quote a string
- qq/STRING/ - doubly quote a string
- reverse - flip a string or a list
- rindex - right-to-left substring search
- sprintf - formatted print into a string
- substr - get or alter a portion of a string
- tr/// - transliterate a string
- uc - return upper-case version of a string
- ucfirst - return a string with just the next letter in upper case
- y/// - transliterate a string

- chomp (useful with file I/O) vs. chop

```
@a = split " ", $line;
```

Note: multiple spaces ok with " " variant

Perl: useful string functions

Transliterate:

- **destructive operation!**
- *tr/matchingcharacters/replacementcharacters/modifiers*
- modifiers are optional:

```
1.  c  Complement the SEARCHLIST.
2.  d  Delete found but unreplaced characters.
3.  s  Squash duplicate replaced characters.
4.  r  Return the modified string and leave the original string
5.      untouched.
```

```
1$s = "A Big Cat";
2$s =~ tr/ABC/abc/;
3print "$s\n";
```

```
1$s = "39,250,017";
2$s =~ tr/,//d;
3print "$s\n";
```

Perl: useful string functions

- Perl doesn't have a built-in *trim-whitespace-from-both-ends-of-a-string* function.
- Can be mimicked using regex (*more later*)

```
1$s = " This is a sentence. ";  
2$s =~ s/^\s+|\s+$//g;  
3print "<$s>\n";
```

Python:

`str.strip([chars])`

Return a copy of the string with the leading and trailing characters removed. The *chars* argument is a string specifying the set of characters to be removed. If omitted or `None`, the *chars* argument defaults to removing whitespace. The *chars* argument is not a prefix or suffix; rather, all combinations of its values are stripped:

```
>>> '   spacious   '.strip()  
'spacious'  
>>> 'www.example.com'.strip('cmowz.')  
'example'
```

The outermost leading and trailing *chars* argument values are stripped from the string. Characters are removed from the leading end until reaching a string character that is not contained in the set of characters in *chars*. A similar action takes place on the trailing end. For example:

Terminal: newline ("`\n`") is
of length 1 in Perl.

A note on string length



```
[~$ perl -le '$l = "été"; print length($l)'  
5  
[~$ perl -le '$l = "\n"; print length($l)'  
1  
[~$ perl -le '$l = "l'\''été"; print length($l)'  
7  
[~$ perl -le '$l = "侍"; print length($l)'  
3  
[~$ perl -le '$l = "l'\''été"; print length($l)'  
7  
[~$ perl -le '$l = "samurai"; print length($l)'  
7  
~$ █
```

Python: a note on string length

```
[~$ python3 -c 'print(len("侍"))'  
1  
[~$ python3 -c 'print(len("été"))'  
3  
[~$ python3 -c 'print(len("'l\''été"))'  
5  
~$ █
```

```
[~$ perl -le 'use utf8; $l = "été"; print length($l)'  
3  
[~$ perl -le 'use utf8; $l = "侍"; print length($l)'  
1  
~$ █
```

Python: bytes vs. characters

- (len) number of characters vs. number of bytes

```
[~$ python3 -c 'print(len("侍"))'  
1  
[~$ python3 -c 'print(len("été"))'  
3  
[~$ python3 -c 'print(len("'1\"'été'))'  
5  
~$ █
```

```
[$ python3 -c 'print(len("a".encode("utf-8")))'  
1  
[$ python3 -c 'print(len("ba".encode("utf-8")))'  
2  
[$ python3 -c 'print(len("侍".encode("utf-8")))'  
3  
[$ python3 -c 'print(len("é".encode("utf-8")))'  
2  
$ █
```

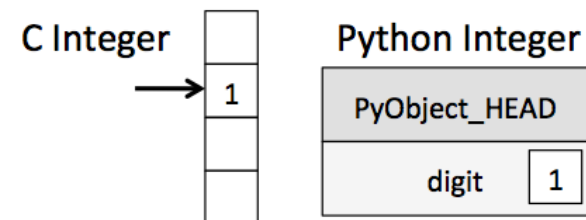
Python: size of object

`sys.getsizeof(object[, default])`

Return the size of an object in bytes. The object can be any type of object. All built-in objects will return correct results, but this does not have to hold true for third-party extensions as it is implementation specific.

```
[$ python3 -c 'import sys;print(sys.getsizeof("侍"))'
76
[$ python3 -c 'import sys;print(sys.getsizeof("a"))'
50
[$ python3 -c 'import sys;print(sys.getsizeof("ab"))'
51
[$ python3 -c 'import sys;print(sys.getsizeof("a侍"))'
78
```

- Python: inefficient in data storage
 - **overhead**



Python: strings

```
5
>>> len(s[0])
1
>>> s = ''
>>> len(s)
0
>>> s = ' '
>>> len(s)
1
>>> s = '日'
>>> len(s)
1
>>> s1 = 'white'
>>> s2 = 'board'
>>> s1 + s2
'whiteboard'
>>> s1 + '-' + s2
'white-board'
>>> s2 + s1
'boardwhite'
>>> s = s1 + '-' + s2
>>> s
'white-board'
>>> █
```

- Many methods that work on lists also work on strings

str.capitalize()

Return a copy of the string with its first character capitalized and the rest lowercased.

str.casefold()

Return a casefolded copy of the string. Casefolded strings may be used for caseless matching.

Casefolding is similar to lowercasing but more aggressive because it is intended to remove all case distinctions in a string. For example, the German lowercase letter 'ß' is equivalent to "ss". Since it is already lowercase, `lower()` would do nothing to 'ß'; `casefold()` converts it to "ss".

The casefolding algorithm is described in section 3.13 of the Unicode Standard.

New in version 3.3.

str.center(width[, fillchar])

Return centered in a string of length *width*. Padding is done using the specified *fillchar* (default is an ASCII space). The original string is returned if *width* is less than or equal to `len(s)`.

str.count(sub[, start[, end]])

Return the number of non-overlapping occurrences of substring *sub* in the range [*start*, *end*]. Optional arguments *start* and *end* are interpreted as in slice notation.

str.endswith(suffix[, start[, end]])

Return `True` if the string ends with the specified *suffix*, otherwise return `False`. *suffix* can also be a tuple of suffixes to look for. With optional *start*, test beginning at that position. With optional *end*, stop comparing at that position.

Perl: file I/O

- Step 1: call `open()`

Files and I/O

You can open a file for input or output using the `open()` function. It's documented in extravagant detail in [perlfunc](#) and [perlopentut](#), but in short:

```
1. open(my $in, "<", "input.txt") or die "Can't open input.txt: $!";  
2. open(my $out, ">", "output.txt") or die "Can't open output.txt: $!";  
3. open(my $log, ">>", "my.log") or die "Can't open my.log: $!";
```

Files: must be opened for reading “<” or writing “>”
(overwrite or append mode “>>”)

Shell syntax: **I/O redirection** “<” “>”

Opening a file creates a **file handle** (Perl variable)

– not to be confused with filename

Supply the **file handle as an argument** for read/write

Perl: file I/O

- Step 2: use the <> operator:

You can read from an open filehandle using the `<>` operator. In scalar context it reads a single line from the filehandle, and in list context it reads the whole file in, assigning each line to an element of the list:

```
1. my $line = <$in>;  
2. my @lines = <$in>;
```

Reading in the whole file at one time is called slurping. It can be useful but it may be a memory hog. Most text file processing can be done a line at a time with Perl's looping constructs.

Note: `$in` is the file handle instantiated by the `open( )` call, e.g.
`open($in, "<", "inputfile.txt");`
or
`open($in, "inputfile.txt");`

Perl: file I/O

- Line by line:

The `<>` operator is most often seen in a `while` loop:

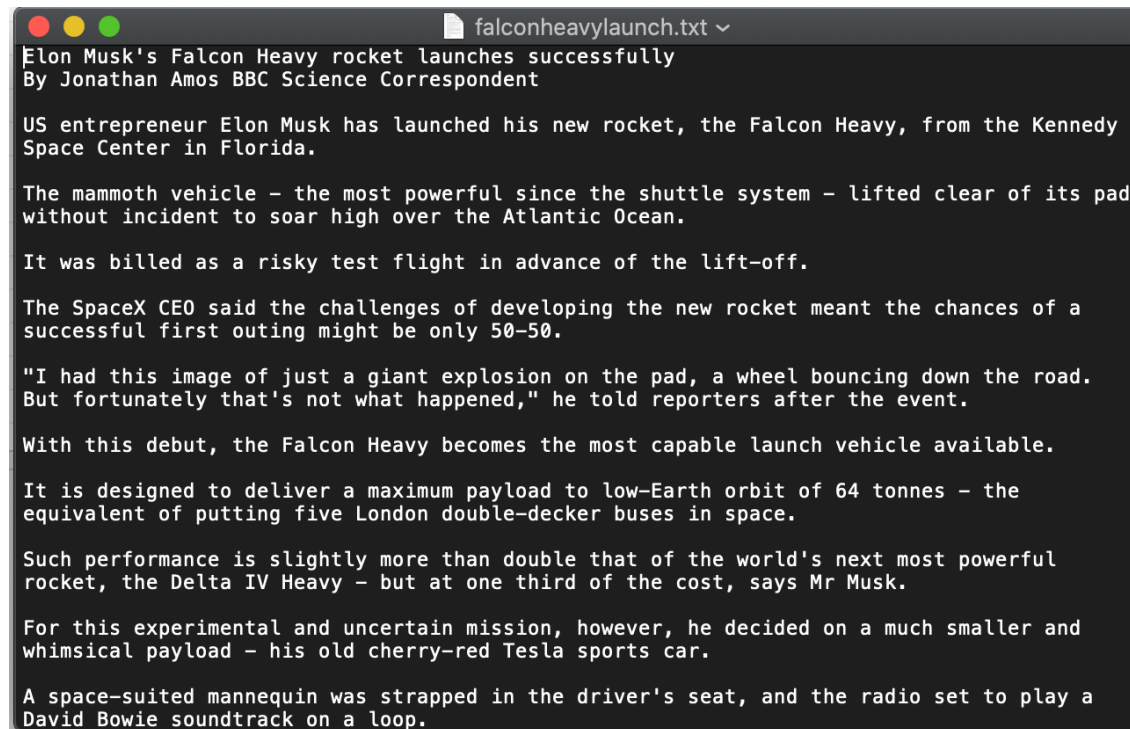
```
1. while (<$in>) {      # assigns each line in turn to $_
2.     print "Just read in this line: $_";
3. }
```

```
read.perl open($txtfile, $ARGV[0]) or die "File $ARGV[0] not found!\n";
while ($line = <$txtfile>) {
    print "$line";
}
close($txtfile)
```

Notes:

1. command `$line = <$txtfile>` inside the while condition reads in a line from the file referenced by the *file handle* `$txtfile`
2. and places that line into the variable `$line` (*including the newline at the end of the line*)
3. At the end of the file, `$line` is just an empty string (equivalent to false).
4. the filename is the first argument to the Perl program (arguments go in `@ARGV`).

Perl: file I/O

A screenshot of a terminal window with a dark background. The title bar at the top shows three colored window control buttons (red, yellow, green) on the left and a file name 'falconheavylaunch.txt' with a dropdown arrow on the right. The terminal displays the output of a Perl script, which is a news article about the Falcon Heavy rocket launch. The text is formatted with a title, author, and several paragraphs of text.

```
falconheavylaunch.txt ~
Elon Musk's Falcon Heavy rocket launches successfully
By Jonathan Amos BBC Science Correspondent

US entrepreneur Elon Musk has launched his new rocket, the Falcon Heavy, from the Kennedy
Space Center in Florida.

The mammoth vehicle – the most powerful since the shuttle system – lifted clear of its pad
without incident to soar high over the Atlantic Ocean.

It was billed as a risky test flight in advance of the lift-off.

The SpaceX CEO said the challenges of developing the new rocket meant the chances of a
successful first outing might be only 50-50.

"I had this image of just a giant explosion on the pad, a wheel bouncing down the road.
But fortunately that's not what happened," he told reporters after the event.

With this debut, the Falcon Heavy becomes the most capable launch vehicle available.

It is designed to deliver a maximum payload to low-Earth orbit of 64 tonnes – the
equivalent of putting five London double-decker buses in space.

Such performance is slightly more than double that of the world's next most powerful
rocket, the Delta IV Heavy – but at one third of the cost, says Mr Musk.

For this experimental and uncertain mission, however, he decided on a much smaller and
whimsical payload – his old cherry-red Tesla sports car.

A space-suited mannequin was strapped in the driver's seat, and the radio set to play a
David Bowie soundtrack on a loop.
```

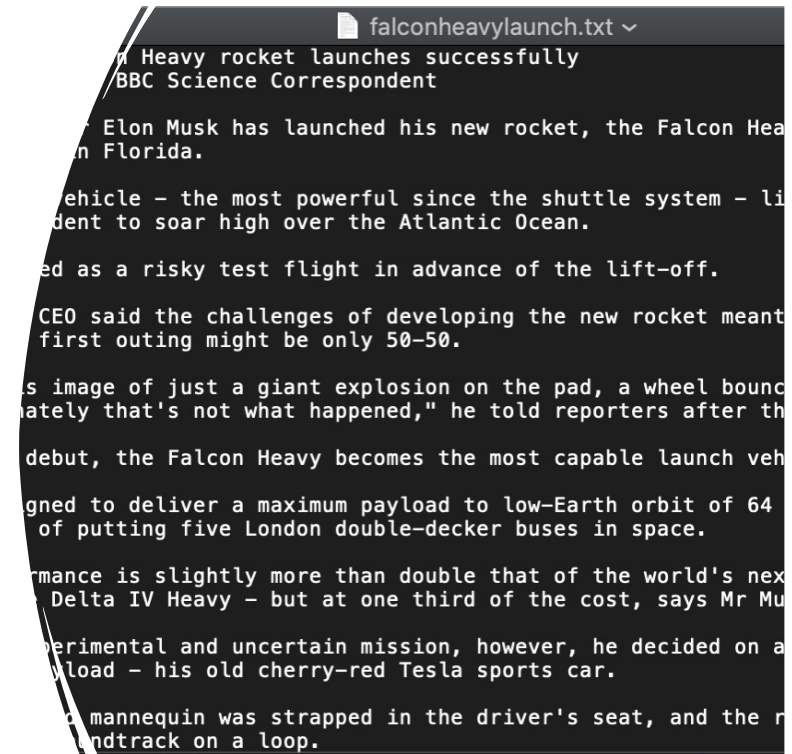
Perl: file I/O

- **What does this code do?**

- `perl -e 'open $f, "falconheavylaunch.txt";
while (<$f>) {print((split " ")[0],"\n"))}'`

- `perl -e 'open $f, "falconheavylaunch.txt";
while (<$f>) {print((split "
")[0],"\n"))}' | wc -l`

- *reports 49 lines*



```
falconheavylaunch.txt ✓
Heavy rocket launches successfully
BBC Science Correspondent

Elon Musk has launched his new rocket, the Falcon Heavy, from Cape Canaveral, Florida.

The vehicle - the most powerful since the shuttle system - lifted off at 11.05pm and is expected to soar high over the Atlantic Ocean.

It was described as a risky test flight in advance of the lift-off.

SpaceX CEO said the challenges of developing the new rocket meant the first outing might be only 50-50.

Instead of the image of just a giant explosion on the pad, a wheel-bounce was not what happened," he told reporters after the launch.

On debut, the Falcon Heavy becomes the most capable launch vehicle designed to deliver a maximum payload to low-Earth orbit of 64,000kg - of putting five London double-decker buses in space.

Its performance is slightly more than double that of the world's next best, the Delta IV Heavy - but at one third of the cost, says Mr Musk.

For the experimental and uncertain mission, however, he decided on a small payload - his old cherry-red Tesla sports car.

A crash test dummy mannequin was strapped in the driver's seat, and the rocket landed on a loop.
```

Perl: file I/O

- A bit more:

- `perl -e 'open $f, "falconheavylaunch.txt"; while (<$f>) {@words = split " "; $sum+=@words}; print $sum'`

- Compare with:

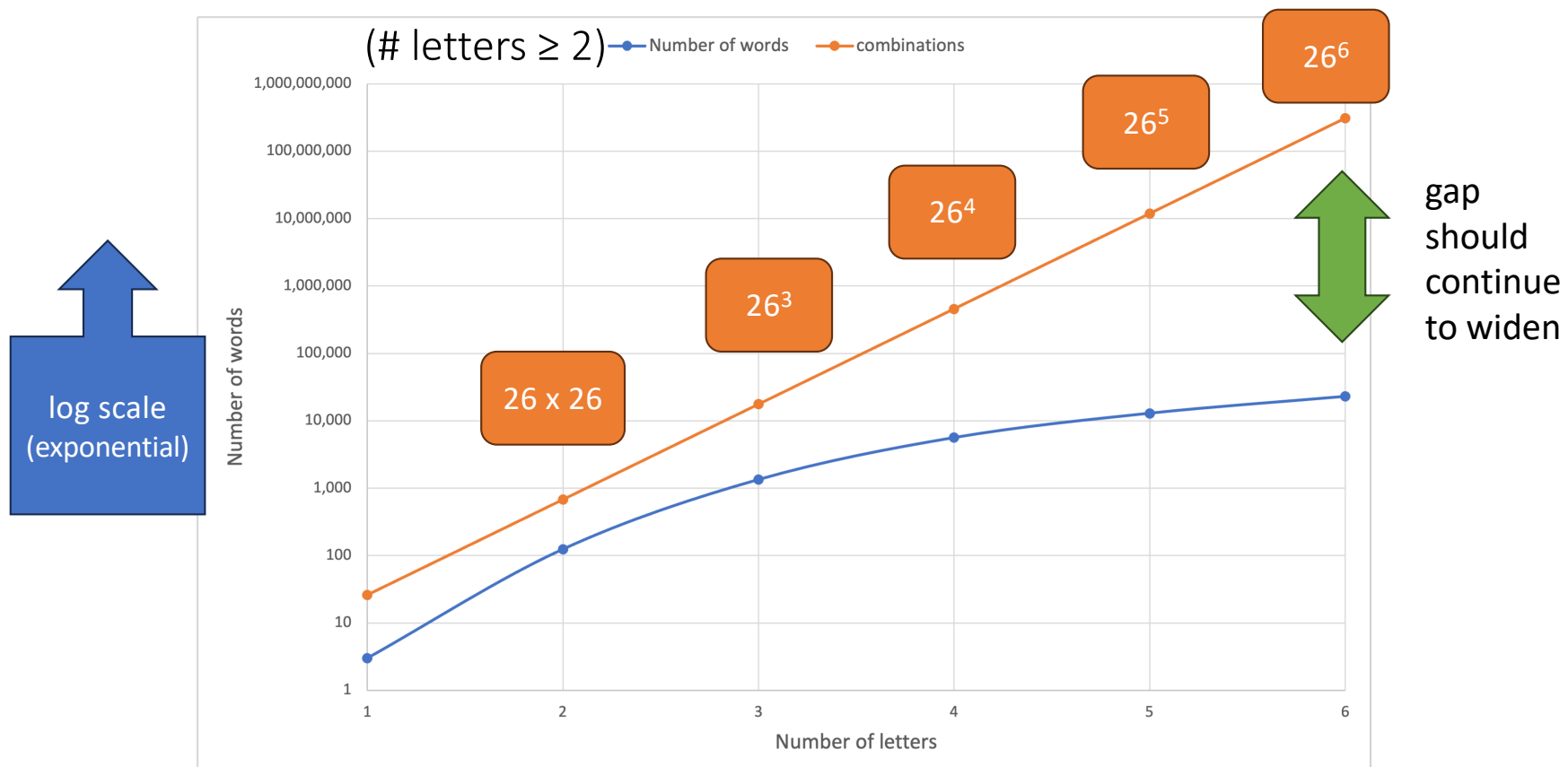
```
wc falconheavylaunch.txt
```

```
49      669      3973 falconheavylaunch.txt
```

Homework 5

- There are 4 files on the course website:
 - 5letters.txt words with 5 letters *etc.*
 - 6letters.txt
 - 7letters.txt

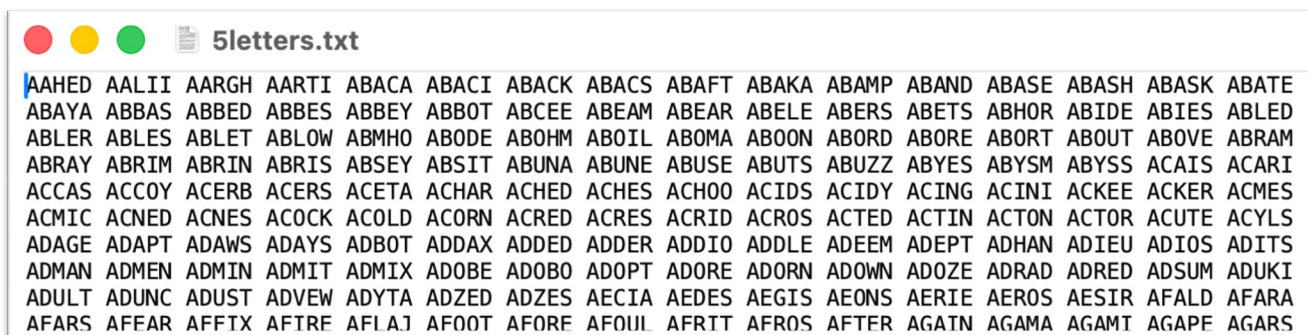
A particular set of Scrabble Words



Homework 5

- **Question 1:**

- write a Perl one-liner or a Perl program that reads in one of the `nletter.txt` files and prints out the number of words.
- Show your program working on each `nletter.txt` for $n = 5, 6$ and 7 at the terminal, e.g. screenshot.
- What numbers did you get for $n = 5, 6$ and 7?



A screenshot of a text editor window titled "5letters.txt". The window displays a list of 5-letter words arranged in 10 rows and 15 columns. The words are: AAHED, AALII, AARGH, AARTI, ABACA, ABACI, ABACK, ABACS, ABAFT, ABAKA, ABAMP, ABAND, ABASE, ABASH, ABASK, ABATE, ABAYA, ABBAS, ABBED, ABBES, ABBEY, ABBOT, ABCEE, ABEAM, ABEAR, ABELE, ABERS, ABETS, ABHOR, ABIDE, ABIES, ABLED, ABLER, ABLES, ABLET, ABLOW, ABMHO, ABODE, ABOHM, ABOIL, ABOMA, ABOON, ABORD, ABORE, ABORT, ABOUT, ABOVE, ABRAM, ABRAY, ABRIM, ABRIN, ABRIS, ABSEY, ABSIT, ABUNA, ABUNE, ABUSE, ABUTS, ABUZZ, ABYES, ABYSM, ABYSS, ACAIS, ACARI, ACCAS, ACCOY, ACERB, ACERS, ACETA, ACHAR, ACHED, ACHES, ACHOO, ACIDS, ACIDY, ACING, ACINI, ACKEE, ACKER, ACMES, ACMIC, ACNED, ACNES, ACOCK, ACOLD, ACORN, ACRED, ACRES, ACRID, ACROS, ACTED, ACTIN, ACTON, ACTOR, ACUTE, ACYLS, ADAGE, ADAPT, ADAWS, ADAYS, ADBOT, ADDAX, ADDED, ADDER, ADDIO, ADDLE, ADEEM, ADEPT, ADHAN, ADIEU, ADIOS, ADITS, ADMAN, ADMEN, ADMIN, ADMIT, ADMIX, ADOBE, ADOBO, ADOPT, ADORE, ADORN, ADOWN, ADOZE, ADRAD, ADRED, ADSUM, ADUKI, ADULT, ADUNC, ADUST, ADVEW, ADYTA, ADZED, ADZES, AECIA, AEDES, AEGIS, AEONS, AERIE, AEROS, AESIR, AFALD, AFARA, AFARS, AFFAR, AFFIX, AFTRE, AFLAJ, AFOOT, AFORE, AFOUL, AFRTT, AFROS, AFTER, AGATN, AGAMA, AGAMT, AGAPE, AGARS.

Homework 5

- **Question 1:**

- HINT *for new programmers:*

1. look at the file format (plain text)
2. use `$ARGV[0]` (filename)
3. use `open` and `<...>` to read the contents
4. use `split`
5. Store the words in an array
6. Print the size of the array

Homework 5

- **Question 2:**

- Which words in the `nletter.txt` files spell the same forwards as backwards (*palindrome*)?
- **Examples:** YAY, WOW, DEED, NAAN (*these are 3 and 4 letter ones*)
- write a Perl one-liner or a Perl program that reads in one of the `nletter.txt` files and prints out the words that satisfy this condition.
- Show your program working on each `nletter.txt` for $n = 5, 6$ and 7 at the terminal, e.g. screenshot.
- Report how many words you got for $n = 5, 6$ and 7?

Homework 5

- **Question 2:**

- HINT *for new programmers:*

- extend your answer from Question 1
 - 1. use foreach to loop over your array (for each word),
 - 2. use reverse and eq to test if each word is a palindrome
 - 3. print it if it satisfies the condition

Homework 5

- Email to sandiway@arizona.edu
- Due by **Sunday** midnight (*no late homeworks, please*)
 - Will be reviewed in class on **Tuesday**
- Email **SUBJECT**: 438/538 Homework 5: **YOUR NAME**
- Either Plain Text or PDF accepted (*no Word .docx files please!*)
- Submit an actual file, no links to files.