

Functional Programming Workout

Code4Lib North, May 2011

William J. Turkel
william.j.turkel@gmail.com
<http://williamjturkel.net>



Fri 13 May 2011 15:52:39 (GMT -4.)

Three audiences

1. Non-programmers
You can accomplish difficult things with a few commands
2. Programmers who aren't familiar with functional programming
Why you might be interested in exploring these techniques
3. Functional programmers
Some possibilities in Mathematica

Imperative programming

Give the computer a sequence of commands that change state

```
var = 1
```

```
1
```

```
exampleList = {a, b, c, d}
```

```
{a, b, c, d}
```

Can refer to pieces of a list using an index (in Mathematica count begins with 1)

```
exampleList[[2]]
```

```
b
```

The For loop is a classic imperative statement

```
For[i = 1, i < 4, i ++,  
  Print[exampleList[[i]]]]
```

```
a
```

```
b
```

```
c
```

Imperative programming versus functional programming

Imperative programs change state and thus have side effects. The variable `i`, which we used in our For loop, still has a value...

`i`

`4`

Functional programming avoids changing state, and treats computation as the evaluation of functions

Simon Peyton-Jones: functional programming is "a radical attack on the whole business of writing programs"

Map

One functional idiom for replacing the For statement is to use Map

```
exampleList
```

```
{a, b, c, d}
```

```
Map[Print, exampleList]
```

```
a
```

```
b
```

```
c
```

```
d
```

```
{Null, Null, Null, Null}
```

Note that functions are first-class objects. You can pass them to other functions as arguments and return them as values.

Note that each time the print statement is evaluated it creates a side effect and returns a null value. The map function collects these values in a list and returns that.

Another example of Map

exampleList

{a, b, c, d}

Framed[exampleList]

{a, b, c, d}

Map[Framed, exampleList]

{a, b, c, d}

Functions can be named or anonymous

```
plus2[x_] :=  
  Return[x + 2]  
Map[plus2, {1, 2, 3}]  
{3, 4, 5}
```

One way of writing an anonymous function in Mathematica is to use a slot in place of a variable

```
# + 2 &
```

So we don't have to define our function in advance, we can just write it where we need it

```
Map[# + 2 &, {1, 2, 3}]  
{3, 4, 5}
```

We can apply an anonymous function to an argument like this

```
(# + 2 &) [40]
```

```
42
```

A named function like `plus2` is still sitting there when we're done with it. An anonymous function disappears after use.

A sample text

As a sample text, we will use the US Declaration of Independence

```
sample = ExampleData[{"Text", "DeclarationOfIndependence"}];  
Short[sample, 2]
```

```
When in the Course of human  
  events, it becomes necessary for one  
  ...el Huntington; William Williams; Oliver  
  Wolcott; Matthew Thornton
```

We convert a string into a list with the `StringSplit` command. In this case I am saying I want to get rid of anything that is not a word character (to eliminate punctuation)

```
sampleList = StringSplit[sample, Except[WordCharacter] ..];  
Short[sampleList, 2]
```

```
{When, in, the, Course, of, human,  
  events, <<1431>>, Huntington, William,  
  Williams, Oliver, Wolcott, Matthew, Thornton}
```


Selecting pieces of lists

```
shortSampleList = Take[sampleList, 40]
{When, in, the, Course, of, human, events, it, becomes,
 necessary, for, one, people, to, dissolve, the,
 political, bands, which, have, connected, them, with,
 another, and, to, assume, among, the, Powers, of, the,
 earth, the, separate, and, equal, station, to, which}
```

```
First[shortSampleList]
```

```
When
```

```
Last[shortSampleList]
```

```
which
```

```
Rest[shortSampleList]
```

```
{in, the, Course, of, human, events, it, becomes,
 necessary, for, one, people, to, dissolve, the,
 political, bands, which, have, connected, them, with,
 another, and, to, assume, among, the, Powers, of, the,
 earth, the, separate, and, equal, station, to, which}
```

We can also use an index to pull out list elements

```
shortSampleList[[4]]
```

```
Course
```

We test membership in a list with MemberQ

```
MemberQ[shortSampleList, "human"]
```

```
True
```

```
MemberQ[shortSampleList, "alien"]
```

```
False
```

Map lets us process each element of our list

Map[ToUpperCase, shortSampleList]

```
{WHEN, IN, THE, COURSE, OF, HUMAN, EVENTS, IT, BECOMES,  
NECESSARY, FOR, ONE, PEOPLE, TO, DISSOLVE, THE,  
POLITICAL, BANDS, WHICH, HAVE, CONNECTED, THEM, WITH,  
ANOTHER, AND, TO, ASSUME, AMONG, THE, POWERS, OF, THE,  
EARTH, THE, SEPARATE, AND, EQUAL, STATION, TO, WHICH}
```

Map[ToLowerCase, shortSampleList]

```
{when, in, the, course, of, human, events, it, becomes,  
necessary, for, one, people, to, dissolve, the,  
political, bands, which, have, connected, them, with,  
another, and, to, assume, among, the, powers, of, the,  
earth, the, separate, and, equal, station, to, which}
```

Map[StringLength, shortSampleList]

```
{4, 2, 3, 6, 2, 5, 6, 2, 7, 9, 3, 3, 6, 2, 8, 3, 9, 5, 5, 4,  
9, 4, 4, 7, 3, 2, 6, 5, 3, 6, 2, 3, 5, 3, 8, 3, 5, 7, 2, 5}
```

Computing word frequencies

```
lowerSampleList = Map[ToLowerCase, sampleList];
```

Sort does what you'd expect

```
sortedSampleList = Sort[lowerSampleList];
```

```
Short[sortedSampleList]
```

```
{a, a, a, a, a, a, <<1433>>,
 world, world, world, would, would, wythe}
```

Tally lets us count how often each element appears

```
wordFreq = Tally[sortedSampleList];
```

```
Short[wordFreq]
```

```
{{a, 16}, {abdicated, 1}, {abolish, 1},
 <<613>>, {world, 3}, {would, 2}, {wythe, 1}}
```

We can sort the list by the frequency. We have to pass an anonymous function to Sort to get the order right

```
sortedFrequencyList = Sort[wordFreq, #1[[2]] > #2[[2]] &];
```

```
Short[sortedFrequencyList]
```

```
{{of, 79}, {the, 77}, {to, 65}, <<613>>,
 {abraham, 1}, {abolish, 1}, {abdicated, 1}}
```

Getting word frequencies

The twenty most frequent words

```
Take[sortedFrequencyList, 20]
```

```
{{of, 79}, {the, 77}, {to, 65}, {and, 56}, {for, 28},  
 {our, 26}, {their, 20}, {has, 20}, {in, 19}, {he, 19},  
 {a, 16}, {them, 15}, {these, 13}, {that, 13}, {by, 13},  
 {we, 11}, {us, 11}, {have, 11}, {which, 10}, {people, 10}}
```

The Cases statement pulls every item from a list that matches a pattern. In this case, we are looking to see how often the word "powers" appears

```
Cases[wordFreq, {"powers", _}]
```

```
{{powers, 5}}
```

nGrams

The Partition command can be used to create n-grams. This tells Mathematica to give us all of the partitions of a list that are two elements long and that are offset by 1

```
bigrams = Partition[lowerSampleList, 2, 1];
```

```
Short[bigrams, 3]
```

```
{{when, in}, {in, the}, {the, course}, {course, of},  
 {of, human}, <<1434>>, {william, williams},  
 {williams, oliver}, {oliver, wolcott},  
 {wolcott, matthew}, {matthew, thornton}}
```

We can tally those, too. Here we pass an anonymous function to Sort again so the most frequent bigrams are listed first

```
sortedBigrams = Sort[Tally[bigrams], #1[[2]] > #2[[2]] &];
```

```
Short[sortedBigrams, 5]
```

```
{{{he, has}, 18}, {{of, the}, 12}, {{of, our}, 7},  
 {{to, the}, 7}, {{in, the}, 7}, {{for, the}, 6},  
 {{of, these}, 6}, {{to, be}, 6}, <<1245>>,  
 {{becomes, necessary}, 1}, {{it, becomes}, 1},  
 {{events, it}, 1}, {{human, events}, 1}, {{of, human}, 1},  
 {{course, of}, 1}, {{the, course}, 1}, {{when, in}, 1}}
```

Concordance

A concordance shows keywords in the context of surrounding words. We can make one of these quite easily if we starting by generating n-grams.

```
sevemgrams = Partition[lowerSampleList, 7, 1];
```

Here we use Cases to pull out all of the 7-grams that have "powers" as the middle word

The TableForm command formats things nicely

```
TableForm[Cases[sevemgrams, {_, _, _, "powers", _, _, _}]]
```

assume	among	the	powers	of
deriving	their	just	powers	from
and	organizing	its	powers	in
whereby	the	legislative	powers	incapable
for	establishing	judiciary	powers	he

Removing stop words

Mathematica has access to a lot of built-in, curated data. Here we grab a list of English stopwords.

```
stopWords = WordData[All, "Stopwords"];  
Short[stopWords, 2]  
{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, a, A, about, <<237>>,   
  without, would, x, X, y, Y, yet, you, your, yours, z, Z}
```

The Select command allows us to use a function to pull items from a list. We want everything that is not a member of the list of stop words.

```
Short[lowerSampleList, 3]  
{when, in, the, course, of, human, events,   
  it, becomes, necessary, for, <<1423>>, ellery,   
  roger, sherman, samuel, huntington, william,   
  williams, oliver, wolcott, matthew, thornton}  
  
lowerSampleNoStopwords =  
  Select[lowerSampleList, Not[MemberQ[stopWords, #]] &];  
Short[lowerSampleNoStopwords, 3]  
{course, human, events, necessary, people,   
  dissolve, political, bands, connected, assume,   
  <<700>>, sherman, samuel, huntington, william,   
  williams, oliver, wolcott, matthew, thornton}
```

Bigrams containing most frequent words

A more complicated example built mostly from functions we've already seen.

Find the most frequently occurring words

```
freqWordCounts =
  Take[Sort[Tally[Take[lowerSampleNoStopwords, {1, -120}]]],
    #1[[2]] > #2[[2]] &], 26]
{{people, 10}, {laws, 9}, {states, 7}, {right, 7},
 {government, 6}, {time, 5}, {powers, 5}, {free, 4},
 {independent, 4}, {large, 4}, {assent, 4},
 {colonies, 4}, {new, 4}, {war, 3}, {seas, 3},
 {power, 3}, {peace, 3}, {justice, 3}, {pass, 3},
 {refused, 3}, {world, 3}, {repeated, 3}, {absolute, 3},
 {usurpations, 3}, {abolishing, 3}, {themselves, 3}}
```

```
freqWords = Map[First, freqWordCounts];
```

Rewrite bigrams as list of graph edges

```
edgeList = Map[#[[1]] → #[[2]] &,
  Partition[lowerSampleNoStopwords, 2, 1]];
Short[edgeList]
```

```
{course → human, human → events,
 <<714>>, wolcott → matthew, matthew → thornton}
```

Grab the most frequent ones

```
freqBigrams =
  Union[Select[edgeList, MemberQ[freqWords, #[[1]] &],
    Select[edgeList, MemberQ[freqWords, #[[2]] &]];
Short[freqBigrams]
{abdicated → government, abolishing → forms,
 <<188>>, world → rectitude, world → refused}
```


Import can be used to scrape webpages

```

Import["http://williamjturkel.net", "Hyperlinks"]
{http://williamjturkel.net/,
 http://williamjturkel.net/updates/,
 http://williamjturkel.net/fabrication/,
 http://williamjturkel.net/how-to/,
 http://williamjturkel.net/,
 http://williamjturkel.files.wordpress.com/2011/02/ob033-
   histogram3d-log-log.png,
 http://williamjturkel.net/2011/02/21/stealth-mode/,
 http://reference.wolfram.com/mathematica/guide/Mathematica
   .html, http://criminalintent.org/,
 http://digitalhistoryhacks.blogspot.com/,
 https://github.com/williamjturkel,
 http://niche-canada.org/,
 http://niche-canada.org/programming-historian,
 http://history.uwo.ca/faculty/turkel,
 http://creativecommons.org/licenses/by-nc-sa/3.0/,
 http://williamjturkel.net/2011/05/03/what-is-the-new-
   manufactory/,
 http://williamjturkel.net/2011/04/19/bits-from-bytes-
   darwin-reprap/,
 http://williamjturkel.net/2011/04/18/building-makerbot-
   00018/,
 http://williamjturkel.net/2011/04/05/measure-refactor/,
 http://williamjturkel.net/2011/04/04/write-and-cluster/,
 http://williamjturkel.net/2011/03/27/burst-documents/,
 http://williamjturkel.net/2011/03/22/spider-to-collect-
   sources/,
 http://williamjturkel.net/2011/03/15/going-digital/,
 http://williamjturkel.net/category/making/,
 http://williamjturkel.net/category/method/,
 http://wordpress.com/?ref=footer,
 http://theme.wordpress.com/themes/wu-wei/,
 http://equivocality.com}

```

Web crawler in a few lines of code

This example comes from Mathematica 8 documentation

<http://www.wolfram.com/mathematica/new-in-8/graph-and-network-modeling/structure-of-the-web.html>

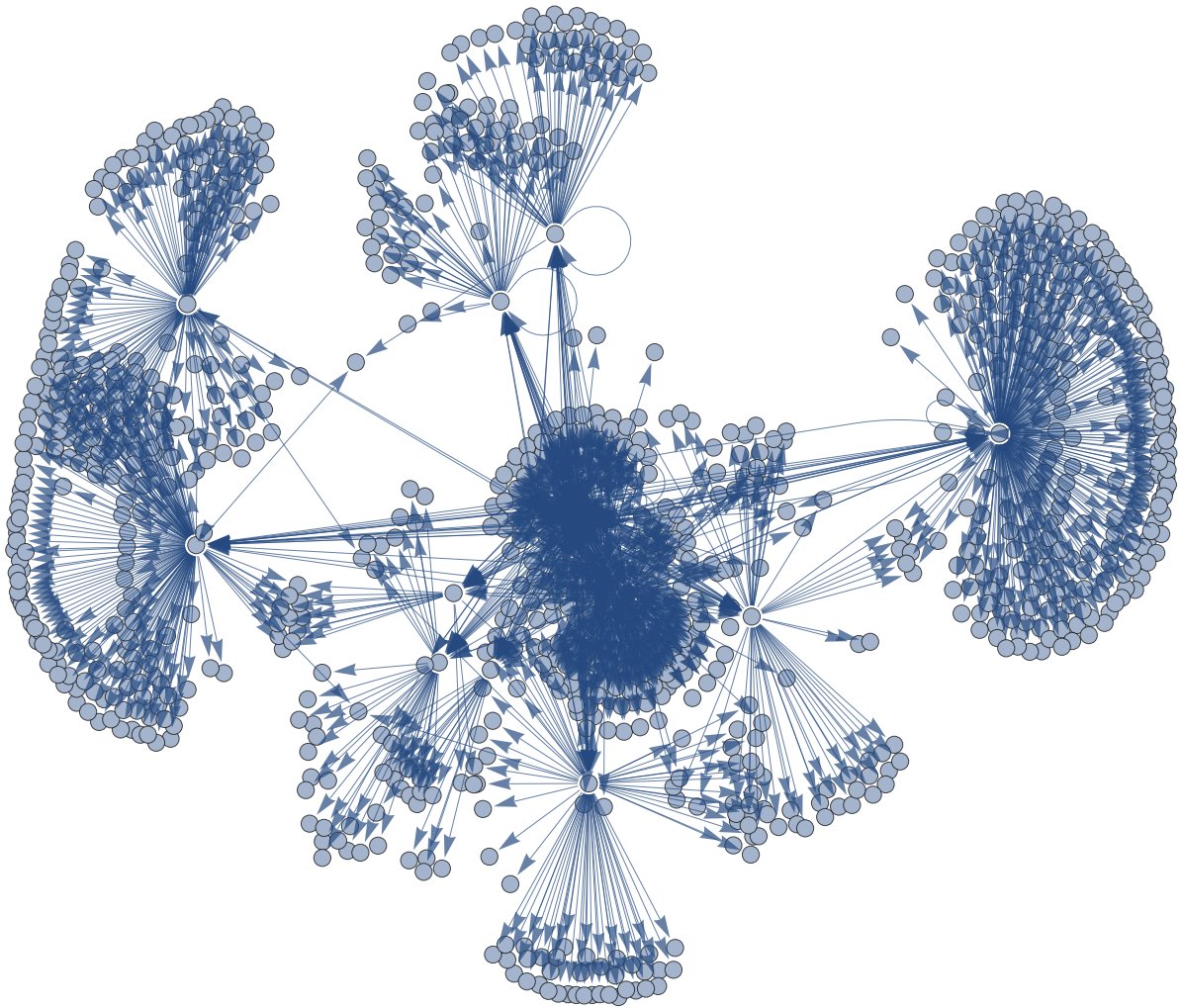
```
webcrawler[rooturl_, depth_] :=  
  Flatten[Rest[NestList[Union[Flatten[  
    Map[Thread[# → Import[#, "Hyperlinks"]]&,  
    Map[Last, #]]]]&, {"" → rooturl}, depth]]];
```

Visualize the network

```
Graph[webcrawler["http://williamjturkel.net", 2],  
      ImageSize -> Full]
```

Import::noelem :

The Import element "Hyperlinks" is not present when importing as PNG. >>



Learn more about functional programming

Learn more about Mathematica

<http://wolfram.com>

Haskell is a free functional programming language

<http://haskell.org>

Scheme and LISP support functional programming

<http://schemers.org>

Other languages have some functional programming constructs

Perl <http://perl.org>

Python <http://python.org>

R <http://r-project.org>