Notes for CSC 254, October 12 and 14, 2020

==========

Scripting languages

What ***is*** a scripting language?
- glue
- extension
- text processing
- web (CGI, server-side, client-side, XSLT)

Common characteristics
- economy of expression
- usually interpreted; for both batch and interactive use
- often a single canonical implementation
- lack of definitions; simple scoping rules
- dynamic typing, sometimes lots of coercion
- high-level types: sets, bags, dictionaries, lists, tuples, objects
- easy access to other programs
- pattern matching and string manipulation

Perhaps more accurately termed ***dynamic languages***, in reference to the use of dynamic typing.

Ancestors
- shells
  - JCL
  - sh/ksh/bash
  - csh/tcsh
  - DOS shell
- text processing
  - RPG
  - sed
  - awk

Modern categories
- general-purpose
  - Rexx (old but still used on IBM platforms) late '70s
  - Perl (long the most widely used) late '80s
  - Tcl (now on the downswing, except for Tk) late '80s
  - Python (has probably passed Perl) early '90s
  - Ruby (on the upswing) early '90s*
    - * didn't really catch on in the West until good English documentation came out in 2001)
  - AppleScript (Mac platform only)
  - PowerShell (and, once, Visual Basic — Windows platform only)
- extension
  - most of the general-purpose ones
    - Python at Disney and ILM
    - Tcl in AOLServer
  - Lua
    - esp. in gaming
  - Scheme
    - Elk
    - SIOD — leading extension language for GIMP
      - (Tcl, Python, Perl also supported)
    - Guile
  - Emacs Lisp
  - proprietary
    - Maya
    - Cold Fusion
    - AutoCAD
    - Macromedia Director, Flash
    - Adobe tools w/ JavaScript, AppleScript, or VBScript
    - many, many others
- math
  - APL
  - S, R
  - Mathematica, Matlab, Maple
- web
  - CGI -- all the GP options
  - PHP -- leading server-side option; also ASP
  - JavaScript -- leading client side option (VB used w/in some orgs.)
  - Dart -- leader among several languages designed to compile to JavaScript as a way to get around the not-on-all-browsers problem
  - XSLT -- for processing XML

---

names & scopes
- what is the scope of an undeclared variable?
  - Perl: global unless declared otherwise
  - PHP: local unless explicitly imported
  - Ruby: foo is local; $foo is global; @foo is instance; @@foo is class

  - Python and R: local if written; global otherwise
    - Fig. 14.16, p. 740:

```
i = 1; j = 3
def outer():
   def middle(k):
      def inner():
         global i     # from main program, not outer
         i = 4
      inner()
      return i, j, k     # 3-element tuple
   i = 2              # new local i
   return middle(j)

print(outer())
print(i, j)
```

    - prints
      ```
      (2, 3, 3)
      4 3
      ```

    - No way in Python to write an intermediate-level (non-global, non-local) var
    - In R, use `foo <<- val` (rather than `foo <- val`) avoids creating a new local R; accesses closest existing

  - Both static and dynamic scope in Perl:
    - Fig. 14.17, p. 741:

```
sub outer($) {                # must be called with scalar arg
   $sub_A = sub {
      print "sub_A  $lex, $dyn\n";
   };
   my $lex = $_[0];           # static local initialized to first arg
   local $dyn = $_[0];        # dynamic local initialized to first arg
   $sub_B = sub {
      print "sub_B  $lex, $dyn\n";
   };
   print "outer  $lex, $dyn\n";
   $sub_A->();
   $sub_B->();
}

$lex = 1; $dyn = 1;
print "main    $lex, $dyn\n";
outer(2);
print "main    $lex, $dyn\n";
```

    - prints
      ```
      main 1, 1
      outer 2, 2
      sub_A 1, 2
      sub_B 2, 2
      main 1, 1
      ```

---

strings & pattern matching

- grep and sed: "basic" REs

- awk, egrep, C regex library: "extended" (Posix) REs
  - quantifiers (generalizations of Kleene closure)
  - character sets
  - ^ and $, .
  - backslash

- Perl, Python, Ruby, JavaScript, elisp, Java, C#: "advanced" REs
  - trailing modifiers:
    - g global (all matches)
    - i case insensitive
    - s allow dot to match an embedded newline
    - m allow $ and ^ to match before / after embedded newline
    - x ignore comments and white space in pattern

  - capture:
    ```
    $_ = "-3.14e+5";    # default subject of match if =~ not used
    if (/^([+-]?)((\d+)\.|(\d*)\.(\d+))(e([+-]?\d+))?$/) {
       # floating point number
       print "sign:  ", $1, "\n";
       print "integer:  ", $3, $4, "\n";  # only one nonempty
       print "fraction: ", $5, "\n";
       print "mantissa: ", $2, "\n";
       print "exponent: ", $7, "\n";
       }
    ```

    - This prints
      ```
      sign:   -
      integer:  3
      fraction: 14
      mantissa: 3.14
      exponent: +5
      ```

  - greedy (default *) and minimal (*?) matches
    - +? matches at least one but no more than necessary
    - ?? matches zero or one, with a preference for zero

  - special escape sequences
    - ***lots*** of these. E.g.,
      - \n, \r, \t, ...
      - \d digit
      - \s white space
      - \w word character (letter, digit, underscore)
      - ...

- Implementation
  - NFA v. DFA
    - tradeoff: DFA requires compilation: good if repeated; may also be necessary if there is capture.
      NFA can be emulated immediately.

  - compilation?
    - qr operator forces (one-time) compilation:

```
for (@patterns) {          # iterate over patterns
   my $pat = qr($_);       # compile to automaton
   for (@strings) {        # iterate over strings
      if (/$pat/) {        # no recompilation required
         print;            # print all strings that match
         print "\n"; }
      }
   print "\n";
}
```

data types
- Perl goes crazy with coercion
  ```
  $a = "4";              # string
  print $a . 3 . "\n";  # concatenation    ==> "43"
  print $a + 3 . "\n";  # addition         ==> 7
  ```

- notion of context in Perl
  - numeric, string, scalar/array, ...

- considerable variety in numeric types
  - always doubles in JavaScript; doubles by default in Lua
  - always strings in Tcl (!)
  - PHP: ints & doubles
  - Perl, Ruby: ints, doubles, and bignums
  - Scheme: ints, doubles, bignums, rationals

- composites:
  - where static languages tend to emphasize arrays & structs, scripting languages typically emphasize mappings (aka hashes, dictionaries, associative arrays)

  - Perl, Python, Ruby:
    - arrays and hashes — both self-expanding (syntax varies)
  - Python: also tuples & sets
    - tuples are immutable (and thus faster than arrays)
    - sets support union, intersection, difference, xor
  - PHP & Tcl: arrays == hashes
    - array is just a hash w/ numeric keys
  - JavaScript: arrays == hashes == objects
  - multidimensional arrays via tuple keys
    - not very efficient
    - much better support in the 3Ms

objects
- hack in Perl 5; supposed to be real in Perl 6
- "Object-based" approach in JavaScript
  - classes added in ECMAScript 6 (backward compatible)
- pure object orientation in Ruby, ala Smalltalk
- executable class declarations
  - mentioned under "elaboration" in Chap. 3 lecture
  - can be used, e.g., to give the effect of conditional compilation