REXX, PERL, AND VISUAL BASIC

Bebo White
Stanford Linear Accelerator Center
and (not vs.)

Perl
Programming

perl

Larry Wall and Randal L. Schwartz

O'Reilly & Associates, Inc.
Caveats

- I am a REXX bigot, but the cards weren't stacked against Perl; I am not a Perl expert (much less bigot);

- the most important thing about comparing these languages is determining how well they support their environment; this is largely implementation-dependent;

- I have never used REXX and Perl on the same system;

- this talk started out as "REXX vs. Perl" - but they really aren't competitors;

- I like Perl; it makes Unix far more "approachable" for me;

- I think that some of the features of Perl can contribute to the development of REXX;
REXX and Perl Have a Similar Background

BOTH-

° were developed largely by an single individual;

° were developed for a particular operating system and strongly utilize features of that system;

° have their roots in a "popular" high level programming language;

° have "natural typing";

° emphasize string processing;

° provide a strong built-in function library;

° emphasize readability and an understandable block structure;

° have useful debugging capabilities;
Perl Names

BLATZ - a filename or directory "handle"
$BLATZ - a scalar variable
@BLATZ - a normal array
%BLATZ - an associative array
&BLATZ - a subprogram
*BLATZ - everything named BLATZ

° does not harken back to EXEC, EXEC2 or Batch;

° does increase the readability /understandability of a program;

° allows program entities to be associated in a subtle way;

° eliminates part of a "style controversy";
Perl Lists

- an ordered list of scalars;
- can be like an array, or "user-defined types";
- can be fully dynamic;
- incorporates some of the capabilities of Parse; for example -

  - @ARGV consists of
    $ARGV[0] to $ARGV[$#ARGV]

  - ($name,$address) =
    split(\/\//,<NAMES>)
Perl "Gotchas"
(for REXX users)

- the default value of a variable is the null string;
- a value is TRUE if it isn't the null string, 0 or "0";
- there are different comparison operators for numerics and strings;
- some operators are borrowed from sed, awk, and various Unix utilities;
Some General Conclusions

- REXX is easier to learn and more readable; REXX is more accessible to a greater audience;

- Perl's syntax is harder to learn and read (unless you're a big C fan); appeals to "hackers";

- Perl is an excellent interpreted shell script/systems language, but not a common embedded macro language for Unix;

- Perl is more consistent with a "Unix mindset" than REXX;

- Some Perl operations are very arcane (e.g., ++i, i++);

- Perl has many more redundancies than REXX;

- Perl has better support for aggregate types than REXX; both languages lack support for non-trivial datatypes;

- Perl is more compact for some things (e.g., string processing); compactness <----> safety?
Perl has an extensive collection of pattern matching operators; REXX relies more heavily on PARSE;

Perl has built-in file feature operators; where REXX relies on OS;

Perl has a package mechanism which REXX lacks;

REXX is more extensible than Perl;
Can REXX Learn From Perl?

- Associative arrays are very "CMS-like"; can be weakly implemented by the REXX ABBREV;

- Perl lists allow for a *for each* construct;

- Perl makes extensive use of the `<STDIN>, <STDOUT>, <STDERR>` streams; REXX LINEIN, LINEOUT capabilities not always implemented;

- PIPELINES can add some Perl capabilities to REXX;