Perl Weekly Challenge - 004

2019-04-22 14:08 +00 • Mark Senn

## Challenge #1

From Perl Weekly Challenge - 004 retrieved on 2019-04-17 at 02:37 +00:

> Write a script to output the same number of PI digits as the size of your script. Say, if your script size is 10, it should print 3.141592653.

See Leibniz's Formula for Pi (retrieved on 2019-04-20 at 16:27 +00) for a proof that

$$\pi = 4\sum_{k\geq 0}(-1)^k\frac{1}{2k+1}.$$

Or rewriting slightly,

$$\pi = 4\sum_{k=0}^{\infty}\frac{(-1)^k}{2k+1}.$$

I experimented with this formula and was impressed with its simplicity but not with its rate of convergence.

Laurent Rosenfeld, Tom Browder, and Fernando Santagata sent messages to the perl6-users mailing list beginning on 2019-04-19 describing the BPP formula and "FatRat's falling back to Num's". Laurent Rosenfeld wrote

> for example, let's print the first 200 digits of pi.
>
> Since I am getting about 1.2 * $n correct digits for a 0..$n range when calling the plouffe subroutine, it is sufficient to use the 0..200 range to get (more than) 200 correct digits.
>
> ```
> sub plouffe (Int $k) {
>     my $result = (1.FatRat / 16 ** $k) *
>     (
>         (4 / (8 * $k + 1))
>       - (2 / (8 * $k + 4))
>       - (1 / (8 * $k + 5))
>       - (1 / (8 * $k + 6))
>     );
> }
> # printing 200 digits of pi
> my $pi = [+] (plouffe $_ for  0..200);
> print substr $pi, 0, 201;
> ```
>
> Result (reformatted): 3.141 592 653 589 793 238 462 643 383 279 502 884 197 169 399 375 105 820 974 944 592 307 816 406 286 208 998 628 034 825 342 117 067 982 148 086 513 282 306 647 093 844 609 550 582 231 725 359 408 128 481 117 450 284 102 701 938 521 105 559 644 622 948 954 930 381 9
>
> Note that $pi contains a lot more digits, many of which are not correct, but the first 201 digits are correct, so I only need to keep the number that are required.

The Perl 6 program:

```
# Perl Weekly Challenge - 004
# Challenge #1
#
# See
#     engineering.purdue.edu/~mark/pwc-004.pdf
# for more information.

# Run using Perl 6.
use v6;

sub plouffe (Int $k)
{
    (1.FatRat/16**$k) *
    (
          (4 / (8*$k + 1))
        - (2 / (8*$k + 4))
        - (1 / (8*$k + 5))
        - (1 / (8*$k + 6))
    )
}

# Get size of the script.
my $size = $*PROGRAM-NAME.IO.s;

# The value of pi has only been double
# checked up to 1000 characters.
if ($size > 1_000)
{
    die "program not tested for a "
        ~ "script size of more than "
        ~ "1000 characters";
}

my $pi = [+] (plouffe $_ for 0..$size);

say substr($pi, 0, $size);
```

Challenge #2

From Perl Weekly Challenge - 004 retrieved on 2019-04-15 at 08:50 +00:

> You are given a file containing a list of words (case insensitive 1 word per line) and a list of letters. Print each word from the file than can be made using only letters from the list. You can use each letter only once (though there can be duplicates and you can use each of them once), you dont have to use all the letters. (Disclaimer: The challenge was proposed by Scimon Proctor)

This would be easy if Perl 6 could compare the set of letters in the word to see if it is a subset of the letters given.

Perl 6 has sets. Sets only keep track of if an element exists and not the number of times it exists. The example word “example” won’t work because it contains the letter “e” twice.

Perl 6 has bags. Bags keep track of if an element exists and the number of times it exists. This is exactly what we need. If the word bag is a subset of the letter bag then all the letters in the word are in the list of letters. The syntax “$word (<=) $letters” does that comparison. Comparing bags will be much easier than writing code to compare hashes.

The Perl 6 program:

```
# Perl Weekly Challenge - 004
# Challenge #2
#
# See
#     engineering.purdue.edu/~mark/pwc-004.pdf
# for more information.

# Run using Perl 6.
use v6;


# Read the list of letters from the
# "letters.txt" file.
# Each letter is on a separate line.
# Words and letters are matched in a
# case-insensitive way.

# Get the list of letters from letters.txt.
my @let = slurp('letters.txt').comb(/<[a..z]>/);
# Make all letters lowercase.
@let = map({.trans('A..Z' => 'a..z')}, @let);
# Make a Perl 6 bag of letters so letters and
# words can be compared easily.
my $let = bag @let;


# Read the words from the "words.txt" file
# one at a time.
# Each word is on a separate line.
# Words and letters are matched in a
# case-insensitive way.

my $fn = 'words.txt';
my $fh = open $fn;

for $fh.lines
{
    # Make all word letters lowercase.
    .trans('A..Z' => 'a..z');
    # Make a Perl 6 bag of word letters
    # so letters and word letters can
    # be compared easily.
    my $word = bag $_.comb(/<[a..z]>/);
    # If the word letters are a subset
    # of the letters specified, print the word.
    ($word (<=) $let)  and  say "$_";
}
close $fh;
```