Power serious: power series in ten one-liners

These Haskell functions implement operations on power series.
Series are represented as lists of numeric coefficients
and are understood formally; convergence is not an issue.
Because the lists are unbounded in length, lazy evaluation is essential.

Extra rules for empty lists will make the operations work also on
finite-length inputs (polynomials) and allow coerced scalars to be finite series.

Reference: M. D. McIlroy, The music of streams (.ps.gz),
Information Processing Letters 77 (2001) 189-195.

Function definitions

A series variable has suffix `s`, or `t` when it's a tail.
An explanation page elucidates some details of formulas marked as links.

Coerce scalar to series

```
series f = f : repeat 0
fromInteger c = series(fromInteger c)
```

Negation

```
negate (f:ft) = -f : -ft
```

Addition

```
(f:ft) + (g:gt) = f+g : ft+gt
```

Multiplication

```
(f:ft) * gs@(g:gt) = f*g : ft*gs + series(f)*gt
```

Division

```
(f:ft) / (g:gt) = qs where qs = f/g : series(1/g)*(ft-qs*gt)
```

Subtraction, integer power

For these operations we rely on Haskell's default definitions of
subtraction in terms of addition and negation,
reciprocation (`recip`) in terms of division,
nonnegative integer power (^) in terms of multiplication, and
general integer power (^^) in terms of (^) and reciprocation.

Composition (#)

```
(f:ft) # gs@(0:gt) = f : gt*(ft#gs)
```

Reversion (compositional inverse)

```
revert (0:ft) = rs where rs = 0 : 1/(ft#rs)
```

Integration

```
int fs = 0 : zipWith (/) fs [1..]    -- integral from 0 to x
```

Differentiation

```
diff (_:ft) = zipWith (*) ft [1..]   -- type (Num a,Enum a)=>[a]->[a]
```

Examples

Coercion allows concisely written polynomials to be treated as power series.
Thus $1+x^2$ may be written `1+(0:1)^2` or `1:0:1`, and is deployed below to define the series for tan x in terms of the series for its functional inverse, arctan $x = \int dx/(1+x^2)$.

```
tans = revert(int(1/(1:0:1)))
```

From the usual differential relations between sine and cosine follows code to compute their power series. Lazy evaluation enables the mutual recursion.

```
sins = int coss
coss = 1 - int sins
```

When the operations are generalized to keep polynomials finite, the coefficients of power series can themselves be (finite) power series. Then the identity $1/(1-(1+x)z) = \Sigma\ (1+x)^n z^n$ leads to a generator of Pascal's triangle:

```
pascal = 1/[1, -[1,1]]
```

This formula expands to a list of lists:

```
[[1], [1,1], [1,2,1], [1,3,3,1], [1,4,6,4,1], ...]::[[Rational]]
```

Complete packages

The code above plus a few lines of declaration make a working bare-bones package. For a quick test, try `take 10 tans`. This exercises every operation but `diff`.

(Sorry, package file extensions are .txt to placate some browsers.
And some browsers render minus signs in these programs as hyphens on the screen.)

Extensions to handle polynomials make a practical package, doubled in size, not as pretty, but much faster and capable of feats like `pascal`.
To see the dramatic speedup, try a bigger test like `take 20 tans`.

Why is finite more complicated than infinite? The end must be detected, if nothing else.

Colophon

Written by Doug McIlroy
doug@cs.dartmouth.edu

July 2007

Aug 2007. Text, but not code, trivially modified.

Sep 2007. Misprint in definition of `sins` corrected; OK in the complete packages.

Explanation page and Pascal's triangle added. Minor text modifications.

Mar 2008. Tweak the introduction.

Apr 2008. Link code to explanations.

Change certain hyphens to minus signs. Explain and fix Enum nuisance.

Jul 2008. Trivial text edits.

Sep 2009. Mention operator (^^). Shortcut multiplication by 0 in practical package.

Oct 2009. Mention `recip`.

Apr 2012. Page translated to Romanian by Alexandra Seremina. No longer available.

Mar 2013. Correct a typo in explanations.

Nov 2013. Simplified definition of `pascal`. Tweaked typesetting of formulas in explanations.

Dec 2013. Replace package file extensions .hs by .txt to mollify browsers.

Feb 2014. Show the expansion of `pascal`.

Sep 2016. One-word text edit.

Jan 2017. Ditto. Placate GHC default prelude by adding (Eq a) to contexts.

Cleverer `int` and `diff` in practical package; reflected in revised explanation.

May 2017. Rework explanations of coercion and the `pascal` example.

Trivial changes in punctuation characters.

Oct 2017. Rework coercion explanation futher; delete related `--` comment

Mar 2018. Disambiguate explanation of multiplication: replace $F'(g+xG)$ with $(F')(g+xG)$.

Oct 2018. Correct spelling errors in explanations.