Programming languages use floating point numbers that can behave weirdly, and users also provide inaccurate inputs...

How to trust the outcomes of numerical software?

**Design Elements**
- decimal rationals 0.0001123
- decimal repetents 1 / 3 = 0.(3)
- midpoint radiuses 5 ± 0.1 == [5-0.1, 5+0.1]
- precision literals ±5.0 == 5±0.05
- error obliviousness
- algebraic laws

**Results**
- axiomatized midpoint radius algebra based on rational numbers (fractions)
- readable (in)exact outputs 0.(3) ± 0.1
- unlike floats, RadCal behaves well with proven associativity and commutativity
- unlike intervals, RadCal has distributivity and (weak) inversion
- fully automatic accuracy tracking with “reasonably tight” bounds (!)
- error refactoring, static error analysis, dynamic error-guided optimizations, all enabled by “error obliviousness” (midpoints are independent of the error estimates)

**TODO:** efficient implementation for the Rascal metaprogramming language on the JVM using fractions and automatically scaled big integers

**TODO:** evaluation of automated precision tracking on common statistical methods such as Pearson correlation

**Disclaimer:** RadCal is probably prohibitively slow for supercomputing purposes, and prohibitively expensive for optimally green computing.