applied math.
Math 420,

(a) the model – its construction usually involves simplifications and omissions;
(b) the data – there may be errors in measuring or estimating values;
(c) the numerical method – generally based on some approximation;
(d) the representation of numbers – for example, π cannot be represented exactly by a finite number of digits;
(e) the arithmetic – frequently errors are introduced in carrying out operations such as addition (+) and multiplication ($\times$).

numerical methods.

need to know what errors are in the inputs, but we don't focus on designing measuring equipment that's more accurate.

Science is about quantifying and bounding the errors in the answer...

- Answer is an approximation of truth.
- Science is bounding the error in the answer.

```
$ julia
               _
   _       _ _(_)_     |  Documentation: https://docs.julialang.org
  (_)     | (_) (_)    |
   _ _   _| |_  __ _   |  Type "?" for help, "]?" for Pkg help.
  | | | | | | |/ _` |  |
  | | |_| | | | (_| |  |  Version 1.6.7 (2022-07-19)
 _/ |\__'_|_|_|\__'_|  |  Official https://julialang.org/ release
|__/                   |
```

```
julia> 1+1
2

julia> 1+1-2
0

julia> typeof(1)
Int64
```

exact arithmetic in Julia.

64 bit integers...

```
julia> 1/2+1/2
1.0

julia> typeof(1.0)
Float64

julia> typeof(1/2+1/2)
Float64

julia> 1/2+1/2-1.0
0.0
```

floating point arithmetic...

sometimes it's exact

```
julia> x=1/10
0.1

julia> typeof(x)
Float64

julia> y=x+x+x+x+x+x+x+x+x+x
0.9999999999999999
```

sometimes there is rounding error...

https://docs.julialang.org/en/v1/manual/integers-and-floating-point-numbers/

Manual / Integers and Floating-Point Numbers

Please read this page...a little...to see the different types of numbers available in Julia.

```
julia> x=big(1)/big(10)
0.1000000000000000000000000000000000000000000000000000000000000000
2
```

still rounding error in arb. precision

```
julia> typeof(x)
BigFloat

julia> x=1//10
1//10
```

exact arith. w/ fractions.

```
julia> y=x+x+x+x+x+x+x+x+x+x
1//1
```

```
julia> a=log(2)
0.6931471805599453

julia> exp(a)
2.0

julia> exp(a)-2.0
0.0
```

```
julia> a=log(3)
1.0986122886681098

julia> exp(a)
3.0000000000000004

julia> exp(a)-3.0
4.440892098500626e-16
```

```
julia> a=log(big(3))
1.098612288668109691395245236922525704647490557822749451734694333637494293218603

julia> exp(a)-3.0
0.0
```

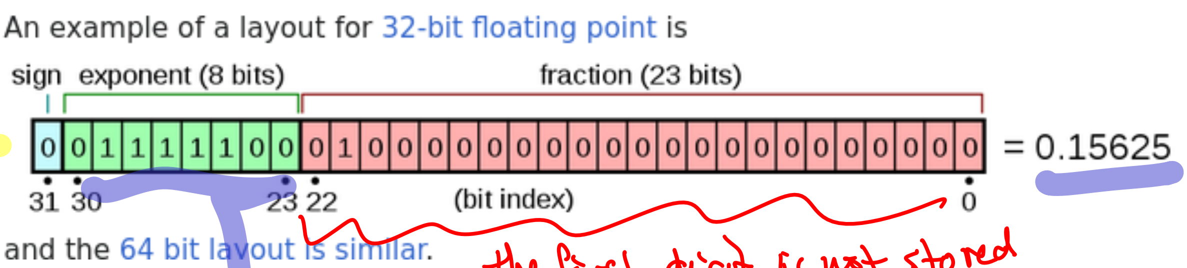
Examples of using Julia and rounding errors...

```
julia> a=log(big(10))
2.3025850929940456840179914546843642076011014886287729760333279009675726096773

julia> exp(a)
10.000000000000000000000000000000000000000000000000000000000000000000000000000

julia> exp(a)-10.0
1.3817869688151111400618162980480639313785600583098050216037925552269746885059
e-76
```

For Friday read the rest of Step 1...about the pendulum...and pay attention to the types of errors in the story.

The IEEE 754 standard for Floating point...

https://en.wikipedia.org/wiki/IEEE_754



the first digit is not stored

Scientific notation:

0.00476 $\qquad 4.76 \times 10^{-3}$

first non-zero digit in the number.

Floating point. Same as Scientific notation except in base 2. (only digits are 0 and 1)

0.00011011 $\qquad 1.1011 \times 2^{-4}$

first non-zero digit is always going to be 1.

$$1.01 \times 2^{e-127} = 1.01 \times 2^{124-127} = 1.01 \times 2^{-3}$$

exponent

128 64 32 16 8 4 2 1

e = 1111100 = 4 + 8 + 16 + 32 + 64 = 124

$101 = 5$

$1.01 \times 2^{-3} = 101 \times 2^{-5} = \frac{5}{2^5} = 0.15625$