Quantifying Dinosaur Pee:
Expressing Probabilities as Floating-Point Values

CppCon 2022
September 16, 2022

John Lakos
Senior Architect

TechAtBloomberg.com

Bloomberg Engineering

first things first

Welcome to my talk

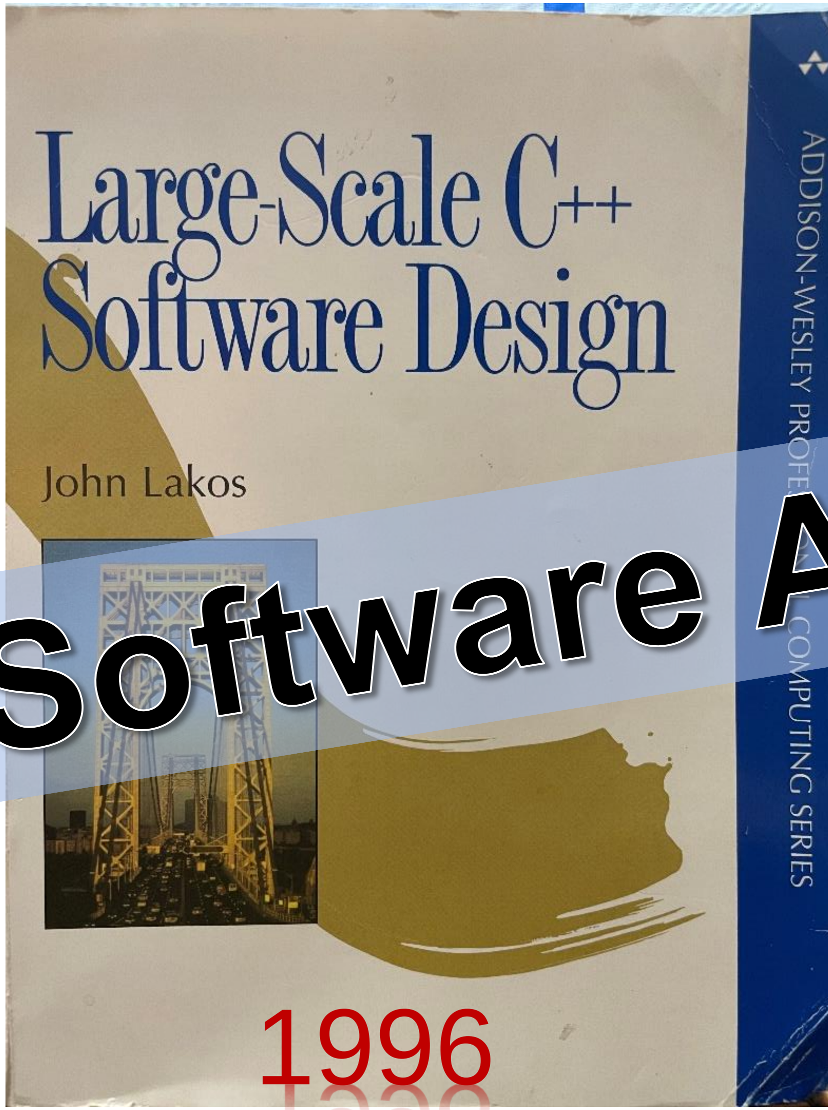
Outline

1. Introduction
2. Reprise “Birthday Problem”
3. Quantify Dinosaur Pee
4. Discuss floating-point values
5. Answer four questions

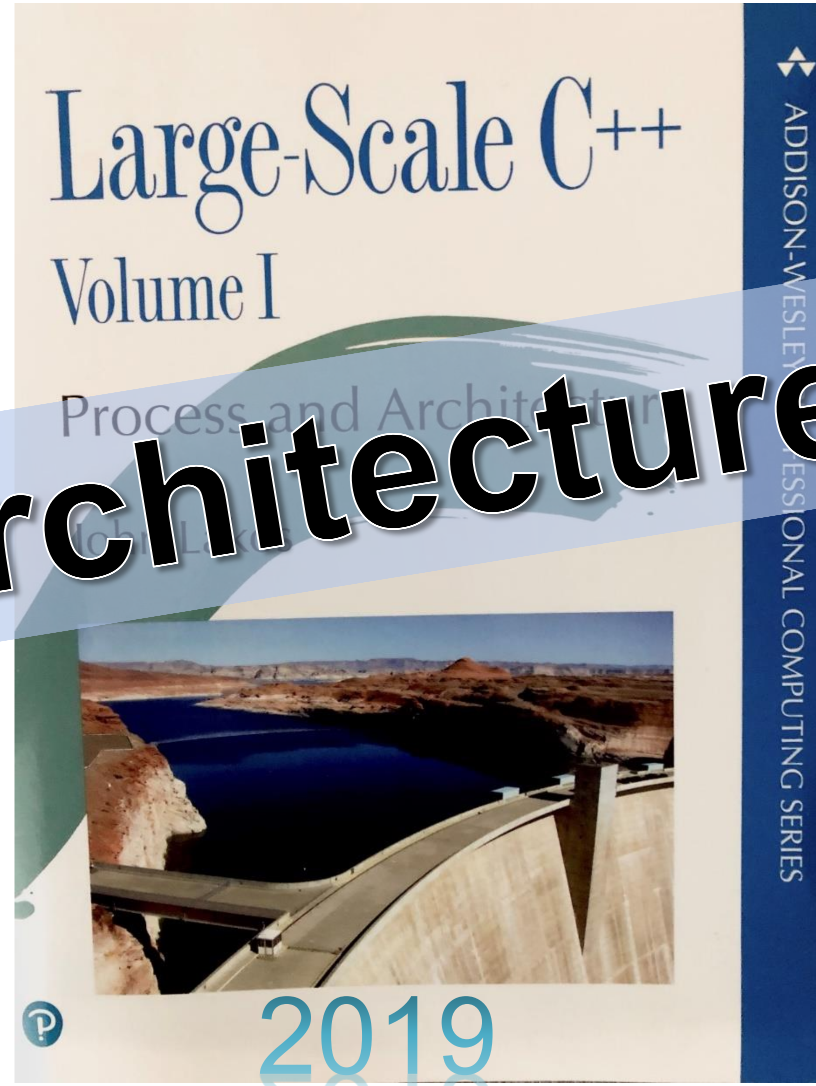
Who am I?

John Lakos

- **I work at Bloomberg in New York City** (c. 2001)
- **I'm active with the C++ Standards Committee:**
  - Initial engagement (c. 2005)
  - Voting member (c. 2007)
- **I've written a few books:**
  - Large-Scale C++ Software Design (c. 1996)
  - Large-Scale C++ (Volume I):
    Process and Architecture (c. 2019)
  - Embracing Modern C++ *Safely* (c. 2021)



1996



2019

Software Architecture

noexcept: CppCon`21

Working on some more books…

- C++ Allocators for the Working Programmer **(CAWP) – 2023** **ISBN:** 9780138060725
- Practical Contract-Driven Programming **(PCDP) – 2024**
- Embracing Modern C++ *Safely* (2^{nd} Ed.) **(EMC++S2ed) – 2025**
- Large-Scale C++ Volume III: Verification and Testing **(LSC++V3:V&T) – TBD**

Large-Scale C++ Volume III:

Verification & Testing

Topics Covered

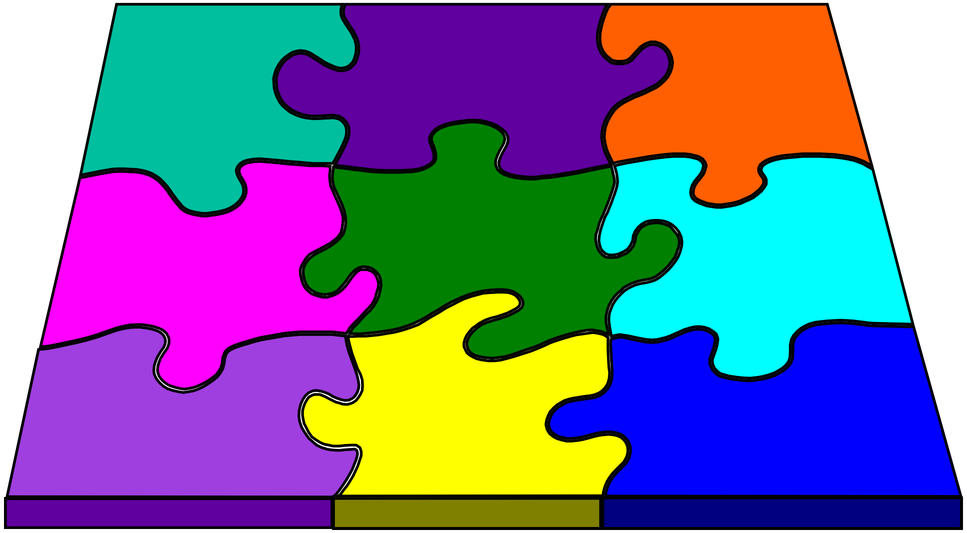
- Ch. 7: Component-Level Testing
  - The absolute need for unit testing
- Ch. 8: Test-Data Selection Methods
  - Identifying test data systematically
- Ch. 9: Test-Case Implementation Techniques
  - Rendering individual test cases effectively
- Ch. 10: Test-Case Ordering
  - Exploiting previously tested, more primitive functionality

Ch. 8: Test-Data Selection Methods

How do we choose our test inputs (“test vectors”)?

- **Ad hoc** – whatever makes sense
- **Boundary conditions** – look at the edges of our algorithms
- **Area testing** – try everything in an area or region
- **Orthogonal dimensions** – choosing a canonical value and then varying each dimension independently
- **Random/statistical** – relying on chance to detect unanticipated problems
- **Depth-Ordered Enumeration** – systematic testing around the origin of a design space
- **Category partitioning** – systematic testing based on equivalence classes

Data Selection: Boundary Conditions



Boundary Conditions

There are at least ***three*** kinds of boundaries to consider:

Those…

1. *Defined* by the **Interface**
2. *Created* by the **Implementation**
3. *Imposed* by the **Platform**

Boundary Conditions

Recall the birthday problem:

What is the probability that two or more people in a room of *n* people have the same birthday?

Boundary Conditions

Birthday Problem

- What is the *minimum number* of people we would need in a room for the probability that "*at least two* of them have the same birthday" is *greater than 50%?*
- Simplifying Assumptions:
  - Only the day of the year matters.
  - All years have 365 days (no leap years)
  - Birthdays are uniformly distributed over the days of the year.
- How does the probability vary with the number of people in the room?

Boundary Conditions

Birthday Function

- Let's design a function that provides a probability value in the range [0.0 .. 1.0] as a function of the number of people in a room.
- What is the interface?
  - Function Name? Return Type? Parameter Name & Type?
- What is the contract?
  - What does it do? How wide should we make it?
  - *Essential Behavior*? (What must happen on valid input?)
  - *Undefined Behavior*? (What input values are not allowed?)

Boundary Conditions

Birthday Function

```
double sameBirthday(int numPeople);
    // Return the probability that at least
    // two of the specified (randomly-
    // chosen) 'numPeople' were born on the
    // same day of the same month.  People
    // born on February 29th are excluded.
    // The behavior is undefined unless
    // '0 <= numPeople'.
```

Designing Component-Level Function Tests

Boundary Conditions

sameBirthday

```
P(1) = 0
P(2) = 0/365 +  365/365    * 1/365
P(3) = 1/365 +  364/365    * 2/365
P(4) = P(3)  + (1 - P(3)) * 3/365
P(5) = P(4)  + (1 – P(4)) * 4/365
. . .
P(363) = P(362) + (1 – P(362)) * 362/365
P(364) = P(363) + (1 – P(363)) * 363/365
P(365) = P(364) + (1 – P(364)) * 364/365
P(366) = P(365) + (1 – P(365)) * 365/365 = 1
P(367) = 1
```

Designing Component-Level Function Tests

Boundary Conditions

sameBirthday

```
Probability ^
            |
       1.00 + - - - - - - - - - - - - - - - - - -*- -*- - - -
            |                                *    \
            |                                      \
            |                                      /
            |                                     /
            .                                    /
            :    Singular Point    Singular Point
                /
            |  /
            | /                                 (not to scale)
            | \
            |  \    *                                 numPeople
       0.00 *---*---'---------- ... ---------'---'---'------>
            0   1   2                       365 366 367
```

Designing Component-Level Function Tests

Boundary Conditions

sameBirthday

```
ASSERT(0.0 == sameBirthday(0)); // external interface boundary
ASSERT(0.0 == sameBirthday(1)); // internal singular point
ASSERT(0.0  < sameBirthday(2)); // internal interface boundary


ASSERT(1.0  > sameBirthday(365));// internal interface boundary
ASSERT(1.0 == sameBirthday(366));// internal singular point
ASSERT(1.0 == sameBirthday(367));// internal interface boundary

ASSERT(1.0 == sameBirthday(INT_MAX);
                                 // external interface boundary
                                 //     also platform boundary
```

Designing Component-Level Function Tests

Boundary Conditions

sameBirthday

```
ASSERT(0.0 == sameBirthday(0)); // external interface boundary
ASSERT(0.0 == sameBirthday(1)); // internal singular point
ASSERT(0.0  < sameBirthday(2)); // internal interface boundary


ASSERT(1.0  > sameBirthday(365));// internal interface boundary
ASSERT(1.0 == sameBirthday(366));// internal singular point
ASSERT(1.0 == sameBirthday(367));// internal i... ...ce boundary

ASSERT(1.0 == s...
                                    ...erface boundary
                            //      also platform boundary
```

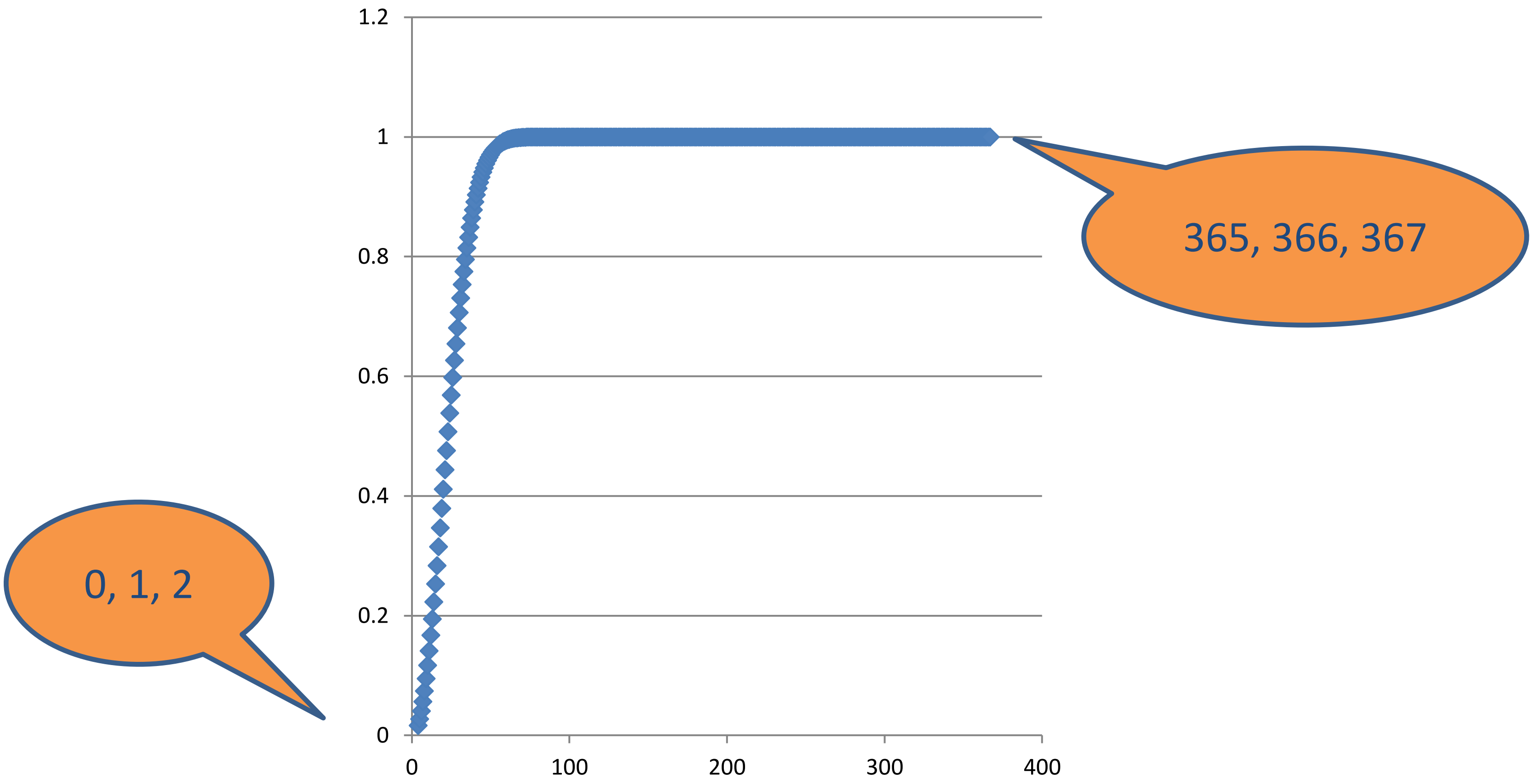
Assertion Failure!

WHY?

Designing Component-Level Function Tests

Boundary Conditions

sameBirthday

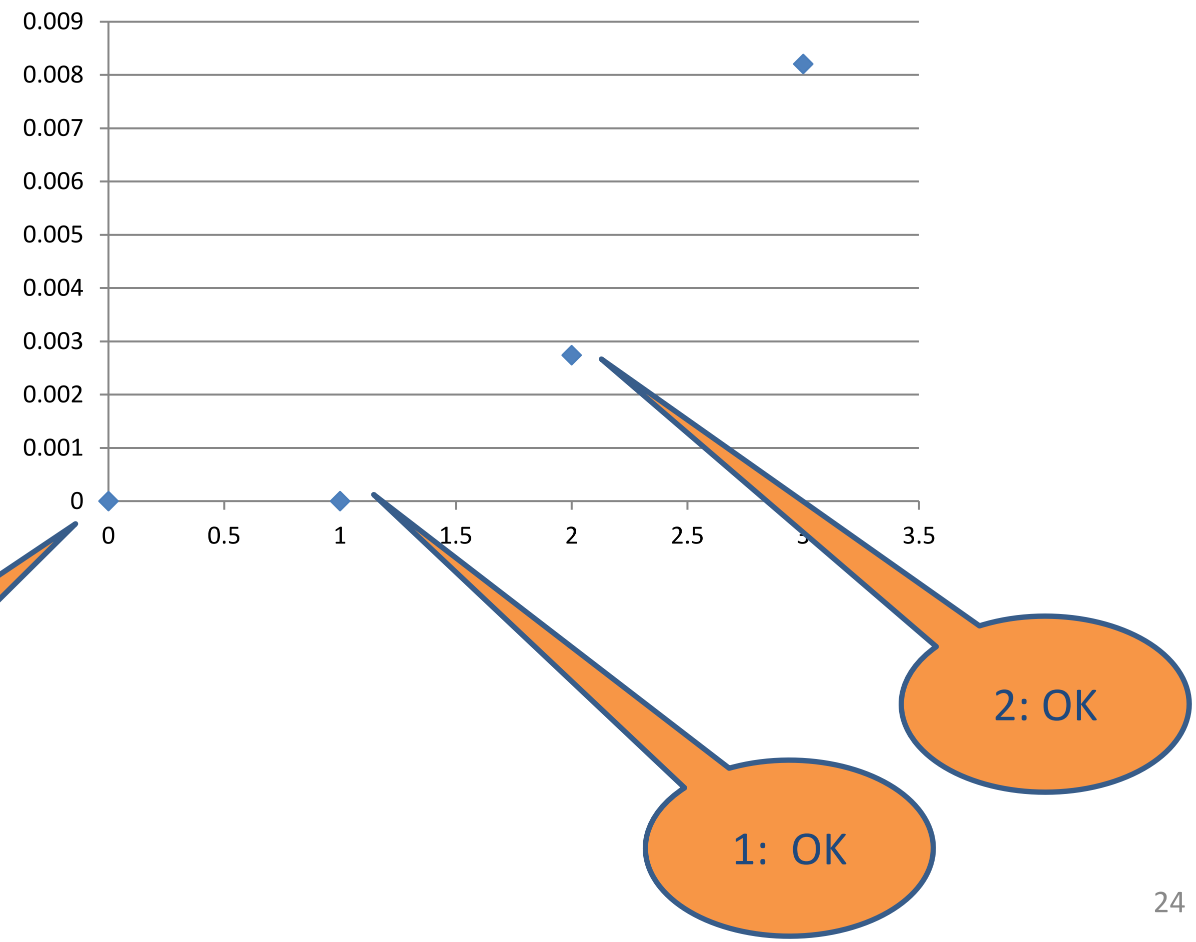


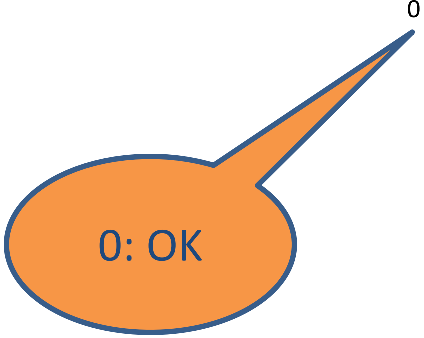
Designing Component-Level Function Tests

Boundary Conditions

sameBirthday

0	0
1	0
2	0.00274
3	0.008204
4	0.016356



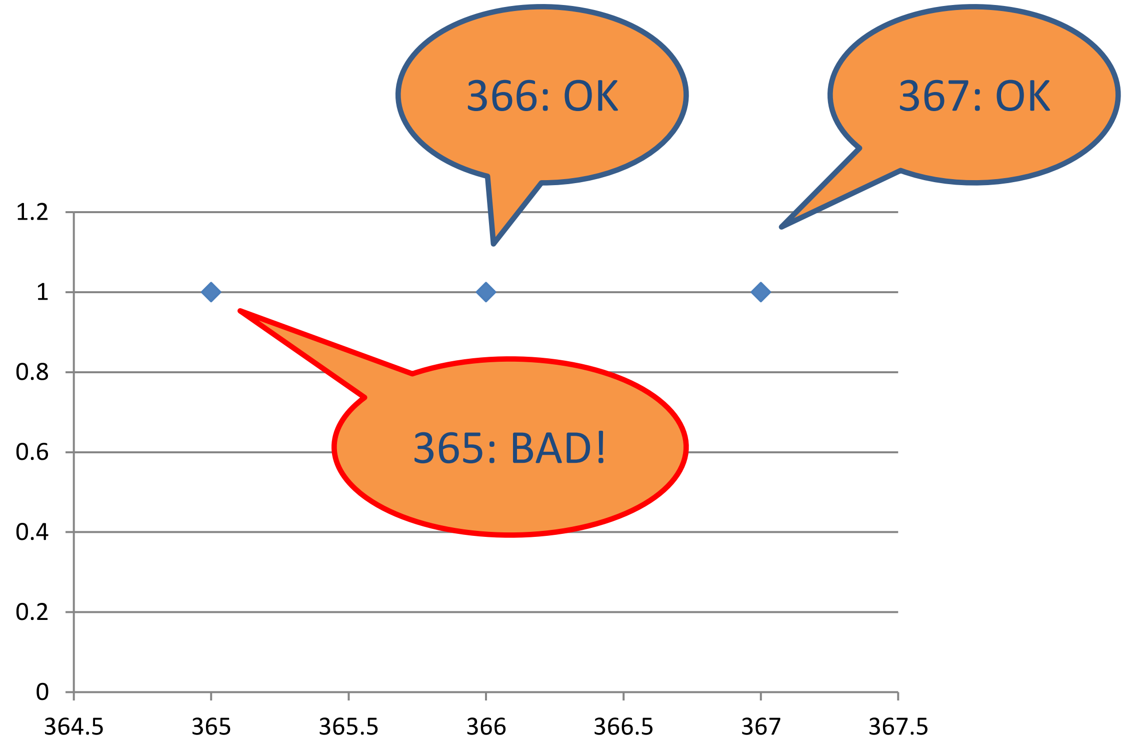


Designing Component-Level Function Tests

Boundary Conditions

sameBirthday

365	1
366	1
367	1

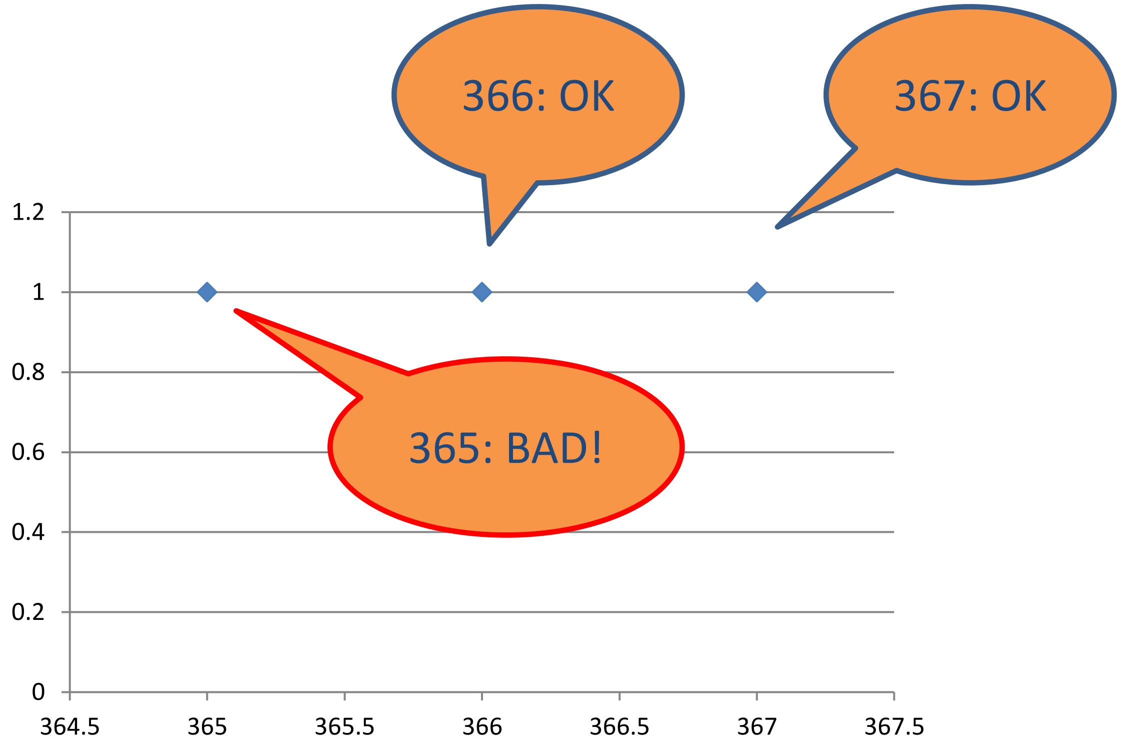


ASSERT(1.0 > sameBirthday(365)); // FAIL

Boundary Conditions

sameBirthday

365	1
366	1
367	1

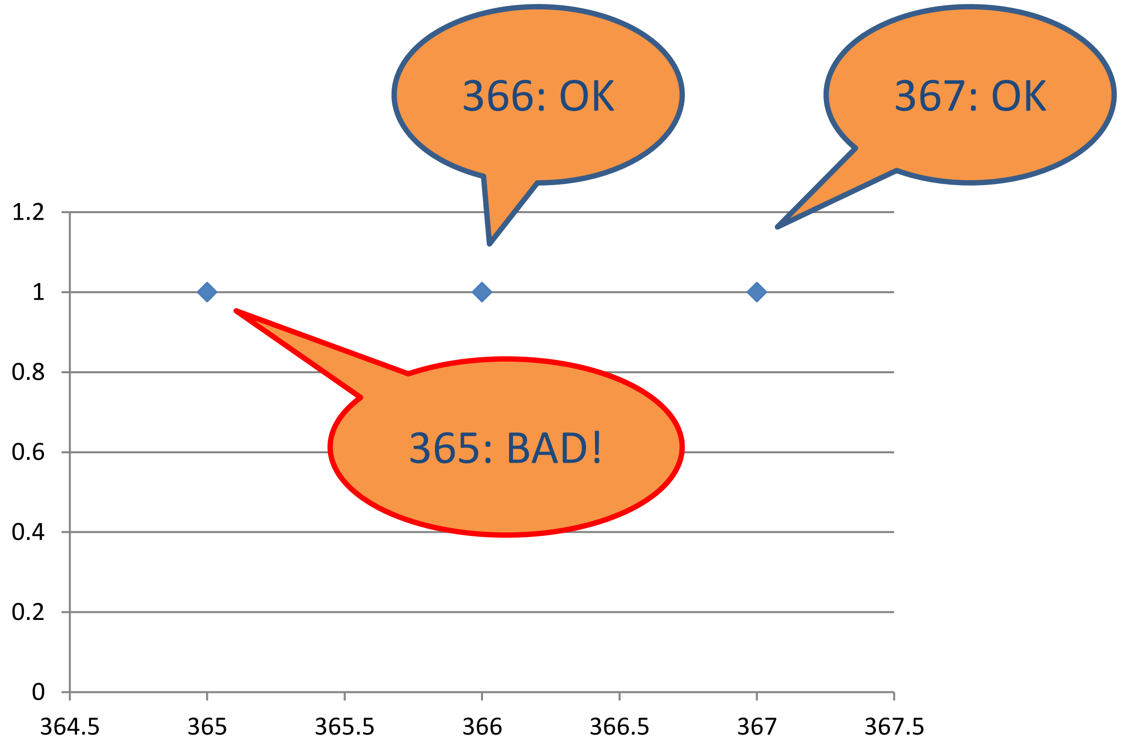



```
ASSERT(1.0 > sameBirthday(364)); // FAIL
```

Boundary Conditions

sameBirthday

365	1
366	1
367	1

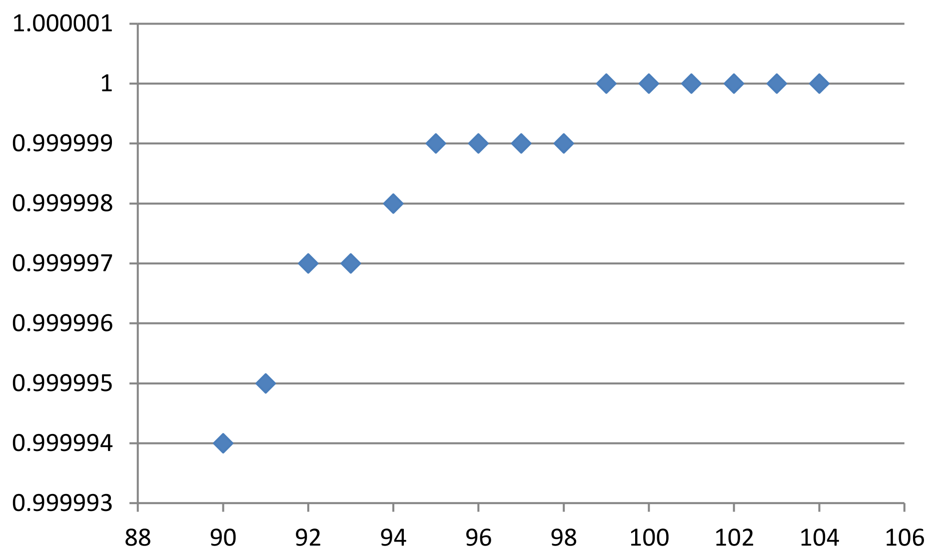



```
ASSERT(1.0 > sameBirthday(363)); // FAIL
```

Boundary Conditions

sameBirthday

90	0.999994
91	0.999995
92	0.999997
93	0.999997
94	0.999998
95	0.999999
96	0.999999
97	0.999999
98	0.999999
99	1
100	1
101	1
102	1
103	1
104	1
105	1

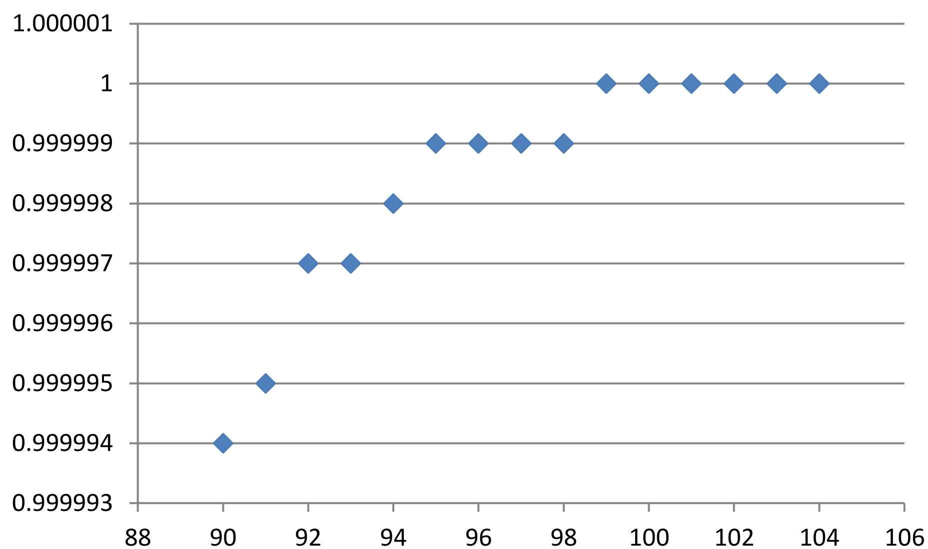



```
ASSERT(1.0 > sameBirthday(98)); // SUCCESS
```

Boundary Conditions

sameBirthday

90	0.999994
91	0.999995
92	0.999997
93	0.999997
94	0.999998
95	0.999999
96	0.999999
97	0.999999
98	0.999999
99	1
100	1
101	1
102	1
103	1
104	1
105	1



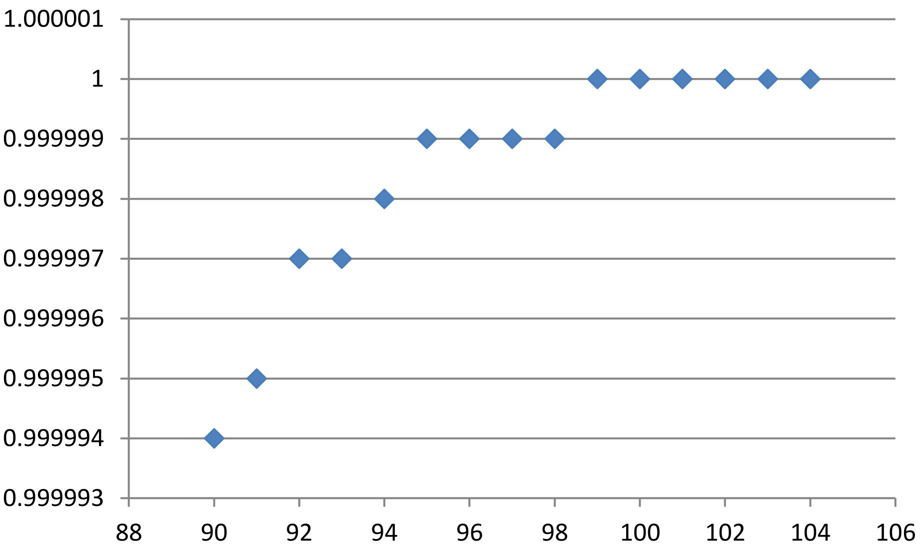

```
ASSERT(1.0 > sameBirthday(99)); // SUCCESS
```

Designing Component-Level Function Tests

Boundary Conditions

sameBirthday

90	0.999994
91	0.999995
92	0.999997
93	0.999997
94	0.999998
95	0.999999
96	0.999999
97	0.999999
98	0.999999
99	1
100	1
101	1
102	1
103	1
104	1
105	1



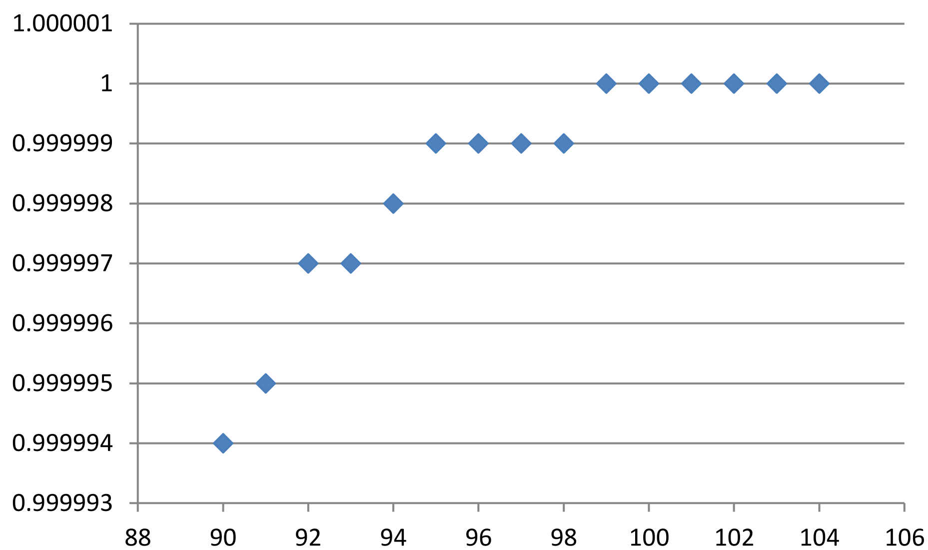

```
ASSERT(1.0 > sameBirthday(100)); // SUCCESS
```

Designing Component-Level Function Tests

Boundary Conditions

sameBirthday

90	0.999994
91	0.999995
92	0.999997
93	0.999997
94	0.999998
95	0.999999
96	0.999999
97	0.999999
98	0.999999
99	1
100	1
101	1
102	1
103	1
104	1
105	1

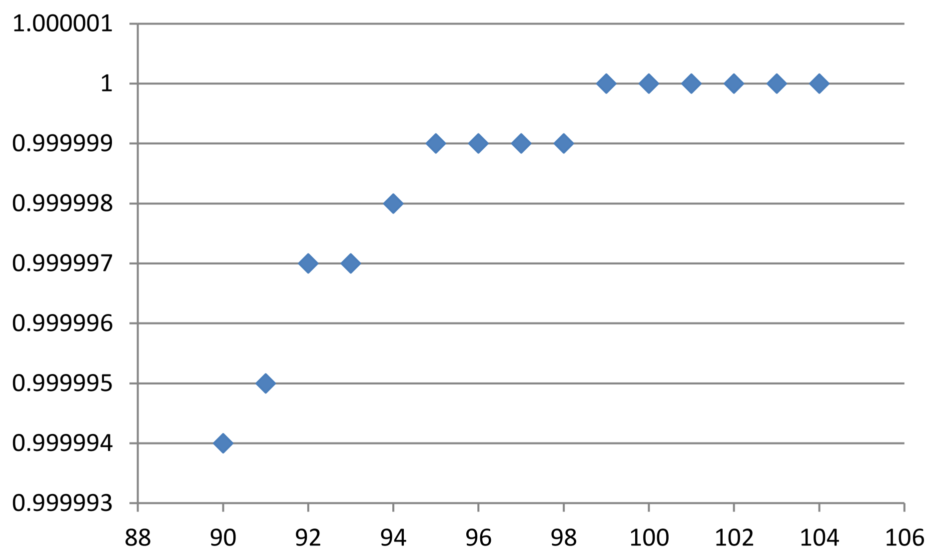



```
ASSERT(1.0 > sameBirthday(101)); // SUCCESS
```

Boundary Conditions

sameBirthday

90	0.999994
91	0.999995
92	0.999997
93	0.999997
94	0.999998
95	0.999999
96	0.999999
97	0.999999
98	0.999999
99	1
100	1
101	1
102	1
103	1
104	1
105	1

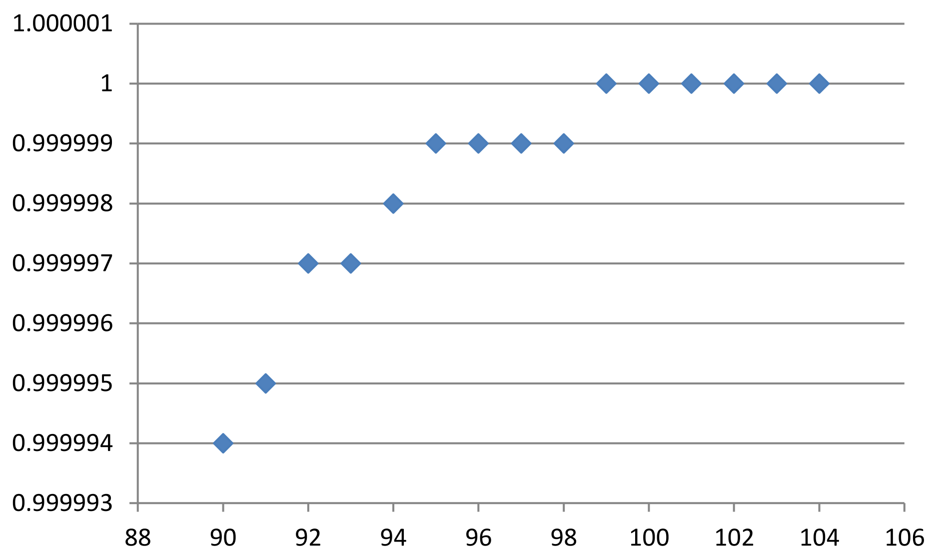



```
ASSERT(1.0 > sameBirthday(102)); // SUCCESS
```

Boundary Conditions

sameBirthday

90	0.999994
91	0.999995
92	0.999997
93	0.999997
94	0.999998
95	0.999999
96	0.999999
97	0.999999
98	0.999999
99	1
100	1
101	1
102	1
103	1
104	1
105	1

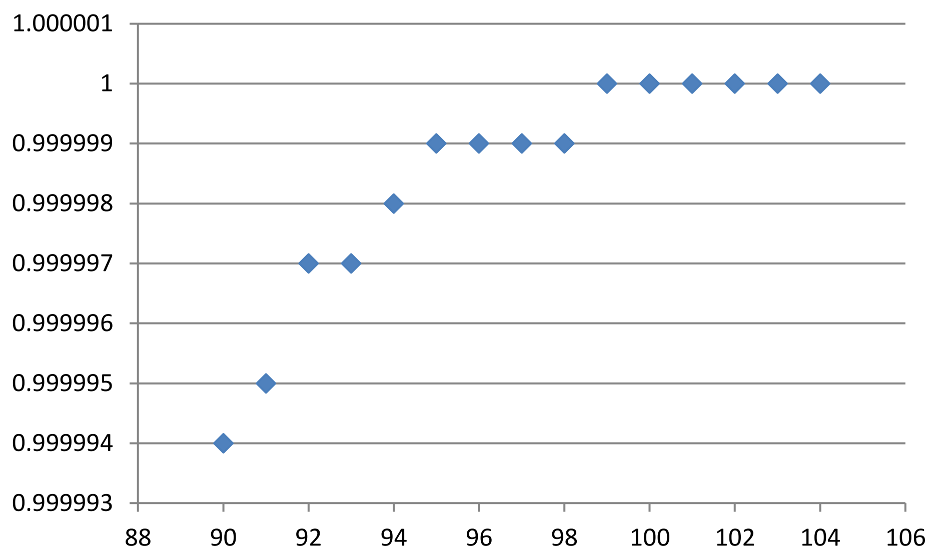



```
ASSERT(1.0 > sameBirthday(103)); // SUCCESS
```

Boundary Conditions

sameBirthday

90	0.999994
91	0.999995
92	0.999997
93	0.999997
94	0.999998
95	0.999999
96	0.999999
97	0.999999
98	0.999999
99	1
100	1
101	1
102	1
103	1
104	1
105	1



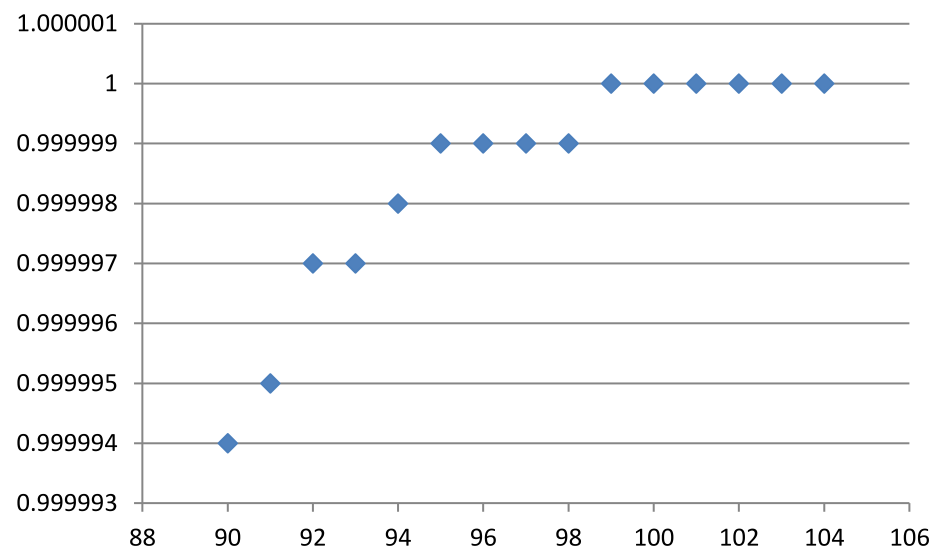

```
ASSERT(1.0 > sameBirthday(110)); // SUCCESS
```

Designing Component-Level Function Tests

Boundary Conditions

sameBirthday

90	0.999994
91	0.999995
92	0.999997
93	0.999997
94	0.999998
95	0.999999
96	0.999999
97	0.999999
98	0.999999
99	1
100	1
101	1
102	1
103	1
104	1
105	1



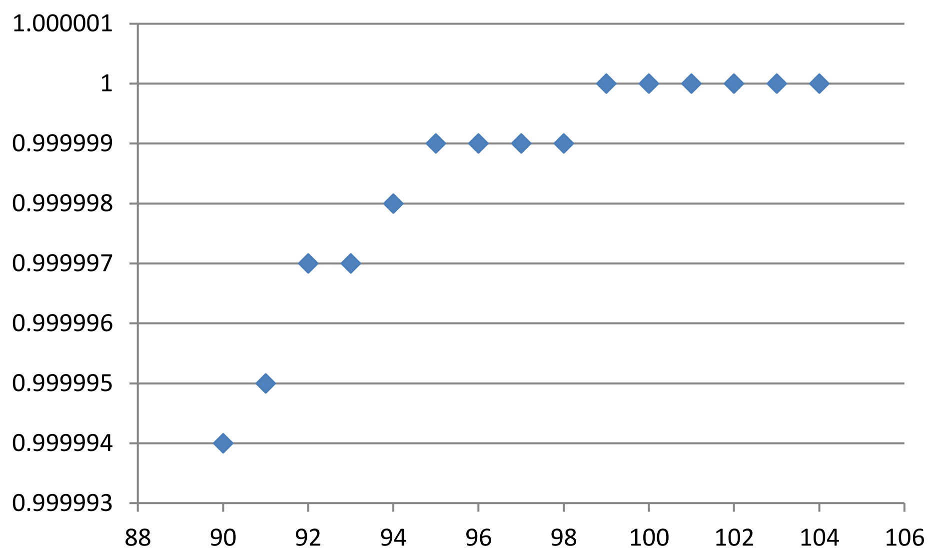

```
ASSERT(1.0 > sameBirthday(120)); // SUCCESS
```

Designing Component-Level Function Tests

Boundary Conditions

sameBirthday

90	0.999994
91	0.999995
92	0.999997
93	0.999997
94	0.999998
95	0.999999
96	0.999999
97	0.999999
98	0.999999
99	1
100	1
101	1
102	1
103	1
104	1
105	1



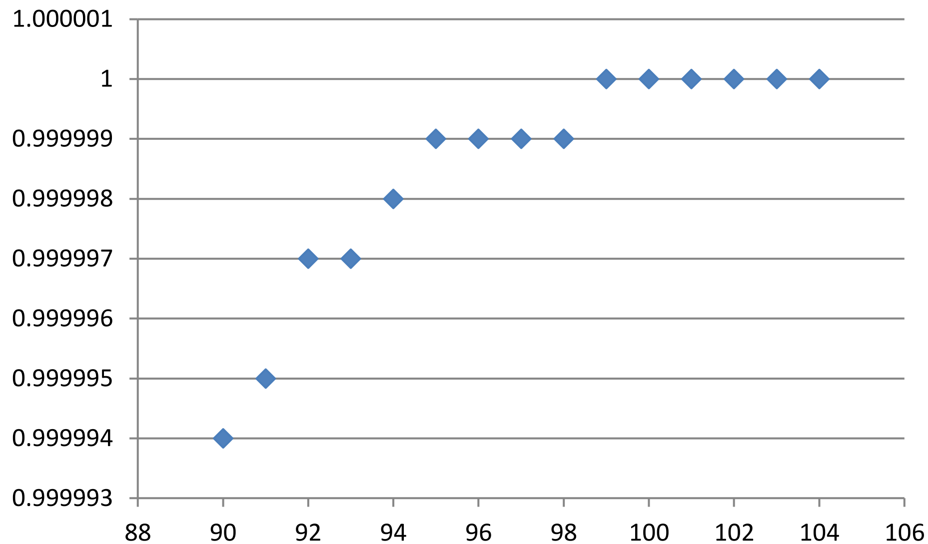

```
ASSERT(1.0 > sameBirthday(150)); // SUCCESS
```

Designing Component-Level Function Tests

Boundary Conditions

sameBirthday

90	0.999994
91	0.999995
92	0.999997
93	0.999997
94	0.999998
95	0.999999
96	0.999999
97	0.999999
98	0.999999
99	1
100	1
101	1
102	1
103	1
104	1
105	1

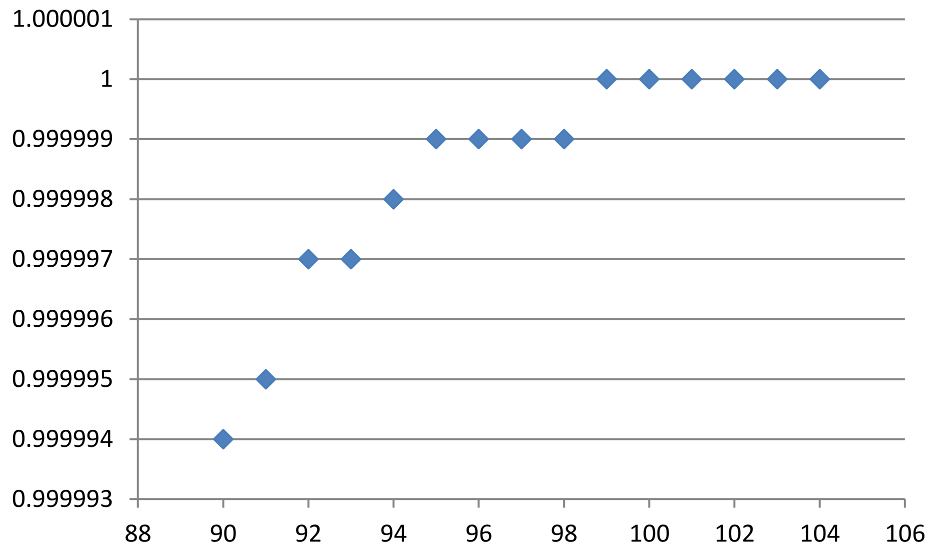



```
ASSERT(1.0 > sameBirthday(200)); // FAIL
```

Boundary Conditions

sameBirthday

90	0.999994
91	0.999995
92	0.999997
93	0.999997
94	0.999998
95	0.999999
96	0.999999
97	0.999999
98	0.999999
99	1
100	1
101	1
102	1
103	1
104	1
105	1



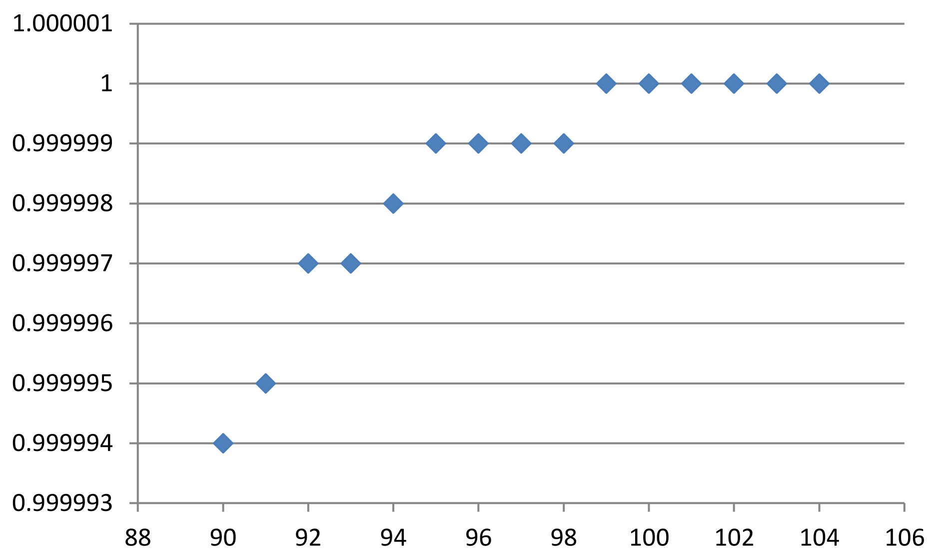

```
ASSERT(1.0 > sameBirthday(175)); // SUCCESS
```

Designing Component-Level Function Tests

Boundary Conditions

sameBirthday

90	0.999994
91	0.999995
92	0.999997
93	0.999997
94	0.999998
95	0.999999
96	0.999999
97	0.999999
98	0.999999
99	1
100	1
101	1
102	1
103	1
104	1
105	1



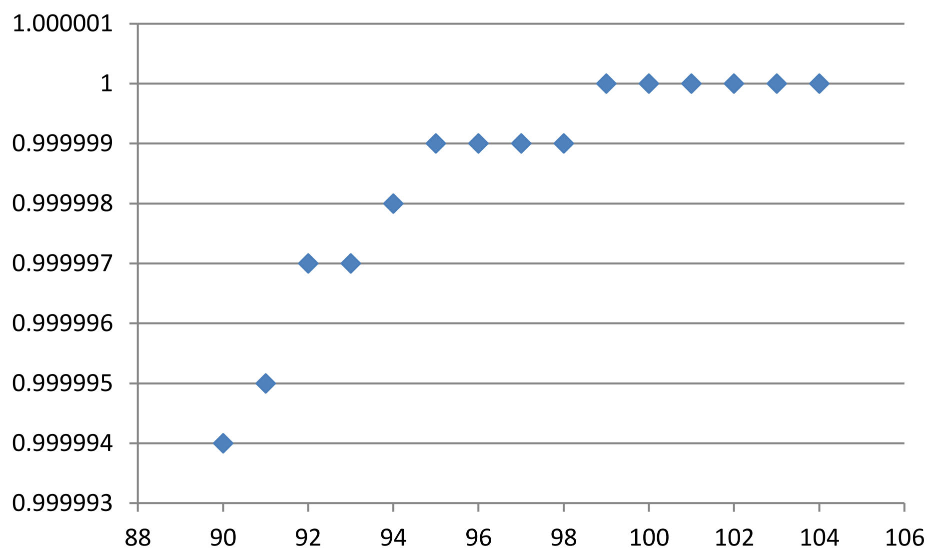

```
ASSERT(1.0 > sameBirthday(185)); // FAIL
```

Designing Component-Level Function Tests

Boundary Conditions

sameBirthday

90	0.999994
91	0.999995
92	0.999997
93	0.999997
94	0.999998
95	0.999999
96	0.999999
97	0.999999
98	0.999999
99	1
100	1
101	1
102	1
103	1
104	1
105	1

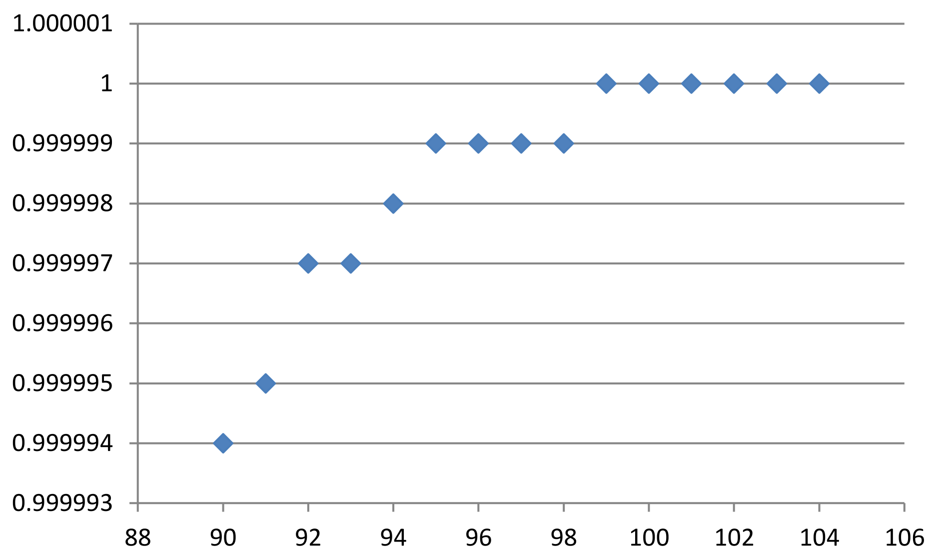



```
ASSERT(1.0 > sameBirthday(180)); // SUCCESS
```

Boundary Conditions

sameBirthday

90	0.999994
91	0.999995
92	0.999997
93	0.999997
94	0.999998
95	0.999999
96	0.999999
97	0.999999
98	0.999999
99	1
100	1
101	1
102	1
103	1
104	1
105	1

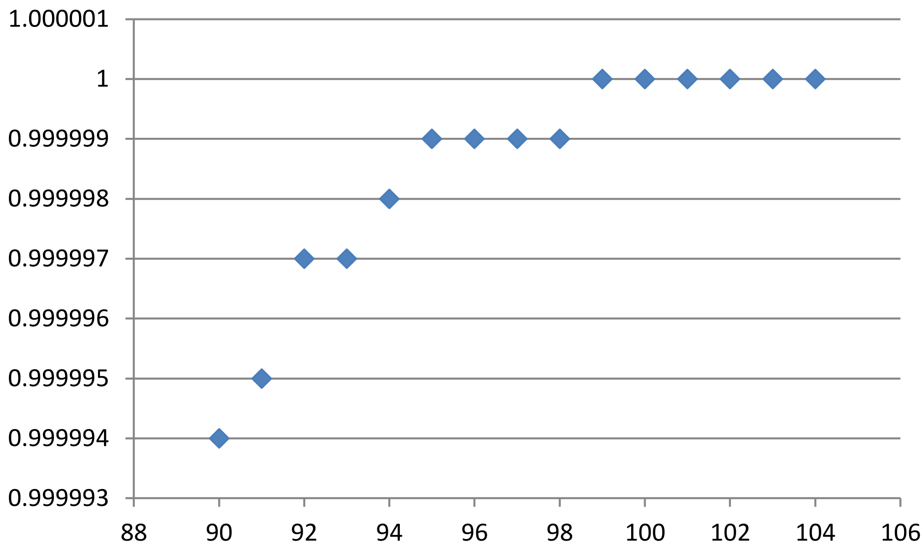



```
ASSERT(1.0 > sameBirthday(182)); // SUCCESS
```

Boundary Conditions

sameBirthday

90	0.999994
91	0.999995
92	0.999997
93	0.999997
94	0.999998
95	0.999999
96	0.999999
97	0.999999
98	0.999999
99	1
100	1
101	1
102	1
103	1
104	1
105	1



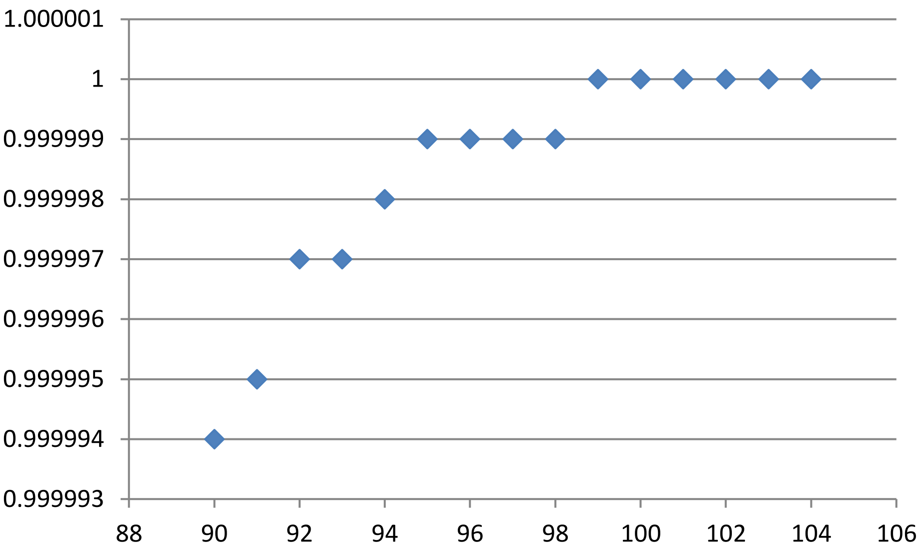

```
ASSERT(1.0 > sameBirthday(184)); // FAIL
```

Designing Component-Level Function Tests

Boundary Conditions

sameBirthday

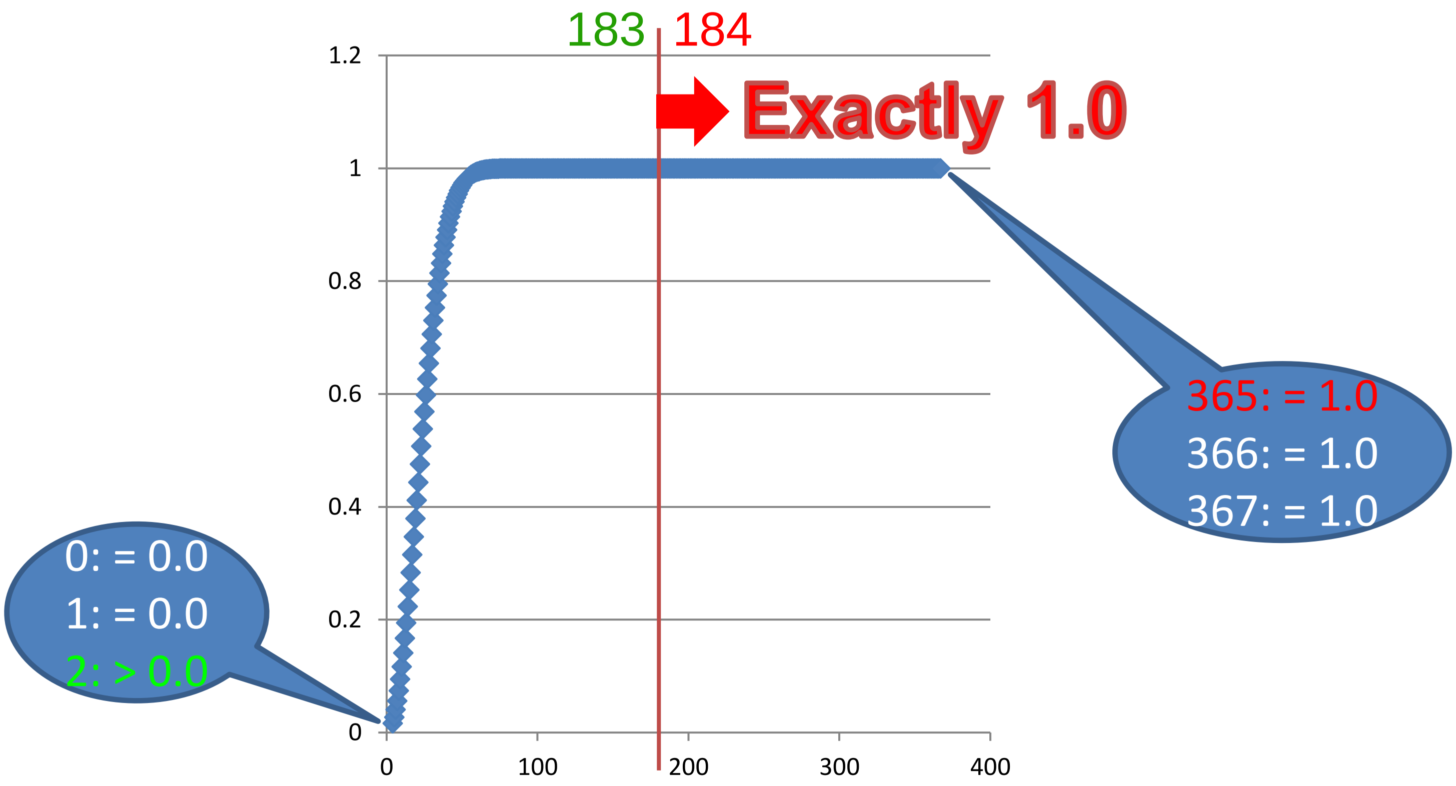
90	0.999994
91	0.999995
92	0.999997
93	0.999997
94	0.999998
95	0.999999
96	0.999999
97	0.999999
98	0.999999
99	1
100	1
101	1
102	1
103	1
104	1
105	1




```
ASSERT(1.0 > sameBirthday(183)); // SUCCESS
```

Boundary Conditions

sameBirthday



Designing Component-Level Function Tests

Boundary Conditions

sameBirthday

So, what’s *wrong* with our implementation?

Designing Component-Level Function Tests

Boundary Conditions

sameBirthday

```
#include <assert.h>

double sameBirthday(int numPeople)
{
    assert(0 <= numPeople);

    if (numPeople > 365) {
        return 1.0;
    }

    double probability = 0.0;

    for (int i = 1; i < numPeople; ++i) {
        probability += (1.0 - probability) * i / 365.0;
    }

    return probability;
}
```

Boundary Conditions

sameBirthday

We have encountered a ***platform-imposed*** boundary condition between 183 and 184 people!

Boundary Conditions

Using a `double` to represent "*Almost* One"

IEEE-754

```
sign-bit  11-bit exponent                  52-bit mantissa
 /       /                                /
0 01111111110 1111111111111111111111111111111111111111111111111111

+ 2^-1 * [ 1 + 1/2 + 1/4 + 1/8 + 1/16 + ...+ 1 * 2^-51 + 1 * 2^-52 ]

      Almost One: 1 - 2^-52 = 0.9999 9999 9999 9999


0 01111111111 0000000000000000000000000000000000000000000000000000

+ 2^0 * [ 1 + 0/2 + 0/4 + 0/8 + 0/16 + ... + 0 * 2^-51 + 0 * 2^-52 ]

      One: 1 = 1.0000 0000 0000 0000
```

Boundary Conditions

Using a `double` to represent "*Almost Zero*"

IEEE-754

`sign-bit  11-bit exponent                          52-bit mantissa`

`0 00000000000 0000000000000000000000000000000000000000000000000001`

`+ 2^-1022 * [ 0/2 + 0/4 + 0/8 + 0/16 + ... + 0 * 2^-51 + 1 * 2^-52 ]`

Almost Zero: 2^-1074 = 4.9406 5645 8412 466 * 10^-324

~~0~~ `00000000000 0000000000000000000000000000000000000000000000000000`

→ ~~1~~ `00000000000 0000000000000000000000000000000000000000000000000000`

`+ 2^-1022 * [ 0/2 + 0/4 + 0/8 + 0/16 + ... + 0 * 2^-51 + 0 * 2^-52 ]`

Zero: 0 = 0.0 ← zero exponent & mantissa (**sign bit is ignored**)

Boundary Conditions

~~sameBirthday~~

- How can we fix our implementation for `sameBirthday`?
  - ➢ We can't: No viable implementation exists!
- What should we do?
  - ➢ Rework our interface & contract.
- How?
  - ➢ Use what we've learned about non-uniformity in the dynamic range of IEEE-754 `double` values.

Designing Component-Level Function Tests

Boundary Conditions

uniqueBirthday

- We can express the probability $\mathrm{P(N)}$ that no two of N people have the same birthday:

$$\mathrm{P(N)} = \frac{365}{365} \times \frac{364}{365} \times \frac{363}{365} \times \frac{362}{365} \times \cdots \times \frac{366-N}{365} = \frac{365!}{(365-N)! \times 365^N}$$

$$\mathrm{P(365)} = \frac{\mathbf{365!}}{\mathbf{365^{365}}} \cong \boldsymbol{e^{-365}} \times \sqrt{\mathbf{2\pi \times 365}}$$

Stirling's approx.*

$$\cong \mathbf{1.45 \times 10^{-157}}$$

$$\gg 4.9406\,5645\,8412\,466 \ * \ 10^{-324}$$

- With `double uniqueBirthday(int numPeople)`, we can at least ***REPRESENT*** the results for the entire range of valid inputs.

*Stirling's approximation: $\boldsymbol{n! = n^n e^{-n} \sqrt{2\pi n}}\,(1 + \ldots)$

Designing Component-Level Function Tests

Boundary Conditions

uniqueBirthday

```
double uniqueBirthday(int numPeople);
    // Return the probability that at least
    // no two of the specified (randomly-
    // chosen)'numPeople' were born on the
    // same day of the same month.  People
    // born on February 29th are excluded.
    // The behavior is undefined unless
    // '0 <= numPeople'.
```

Boundary Conditions

sameBirthday

```
#include <assert.h>

double sameBirthday(int numPeople)
{
    assert(0 <= numPeople);

    if (numPeople > 365) {
        return 1.0;
    }

    double probability = 0.0;

    for (int i = 1; i < numPeople; ++i) {
        probability += (1.0 - probability) * i / 365.0;
    }

    return probability;
}
```

Designing Component-Level Function Tests

Boundary Conditions

uniqueBirthday

```
#include <assert.h>

double uniqueBirthday(int numPeople)
{
    assert(0 <= numPeople);

    if (numPeople > 365) {
        return 0.0;
    }

    double probability = 1.0;

    for (int i = 364; i >= 366 - numPeople; --i) {
        probability *= (1.0 - probability) * i/365.0;
    }

    return probability;
}
```

Designing Component-Level Function Tests

Boundary Conditions

uniqueBirthday

```
#include <assert.h>

double uniqueBirthday(int numPeople)
{
    assert(0 <= numPeople);

    if (numPeople > 365) {
        return 0.0;
    }

    double probability = 1.0;

    for (int i = 364; i >= 366 - numPeople; --i) {
        probability *= i/365.0;
    }

    return probability;
}
```

Designing Component-Level Function Tests

Boundary Conditions

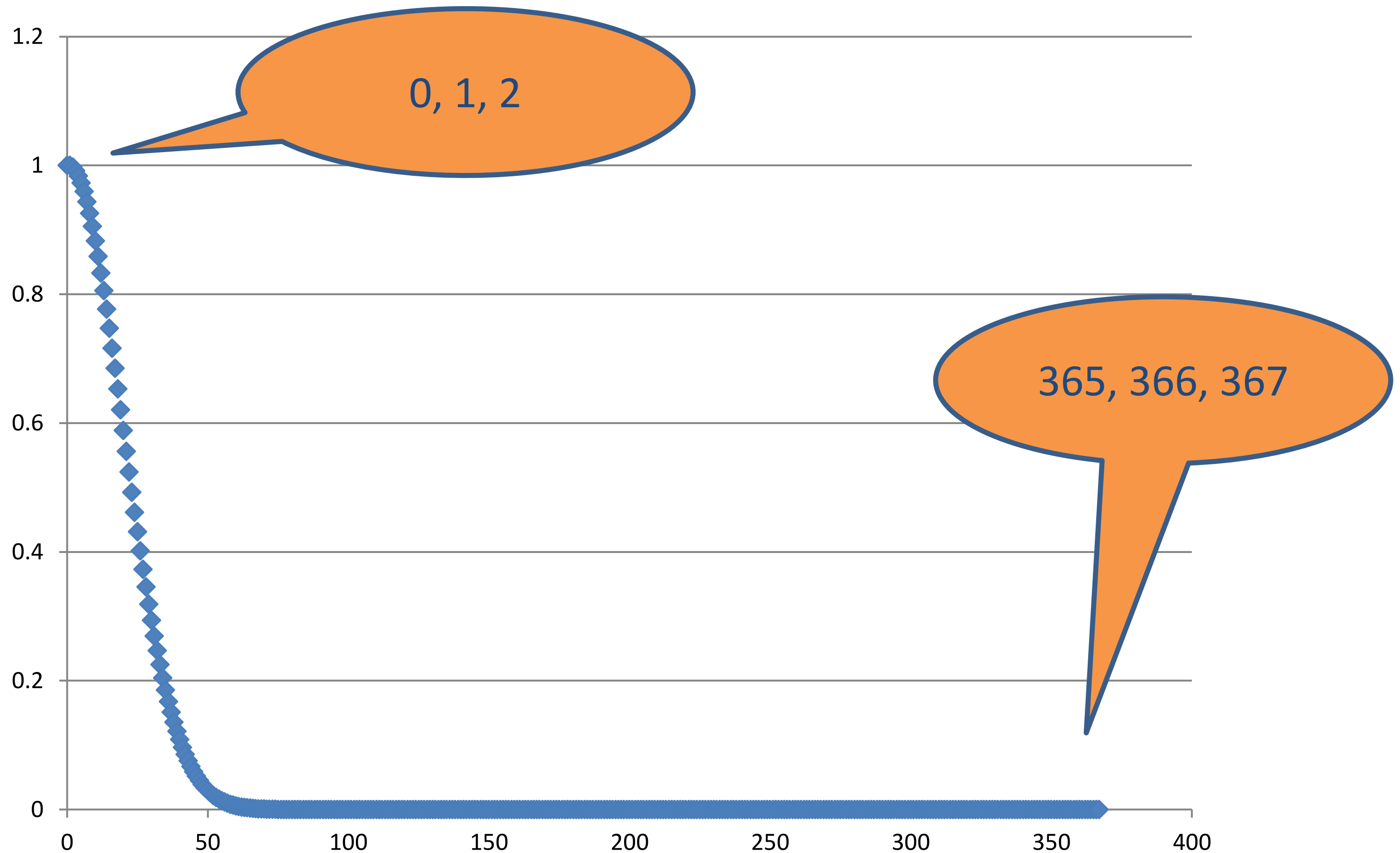
uniqueBirthday

```
#include <assert.h>

double uniqueBirthday(int numPeople)
{
    assert(0 <= numPeople);

    if (numPeople > 365) {
        return 0.0;
    }

    double probability = 1.0;

    for (int i = 364; i >= 366 - numPeople; --i) {
        probability *= i/365.0;
    }

    return probability;
}
```

Designing Component-Level Function Tests

Boundary Conditions

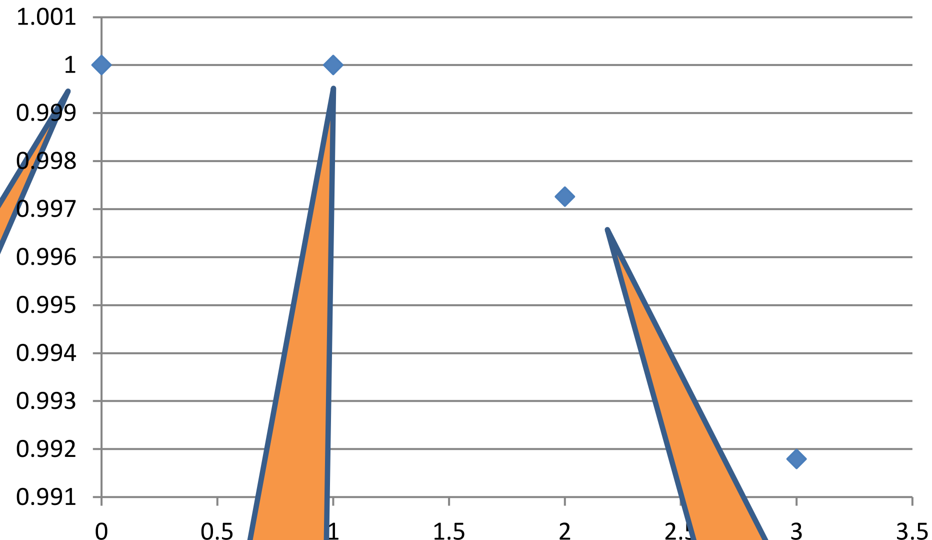
uniqueBirthday



Boundary Conditions

uniqueBirthday

0	1
1	1
2	0.99726
3	0.991796

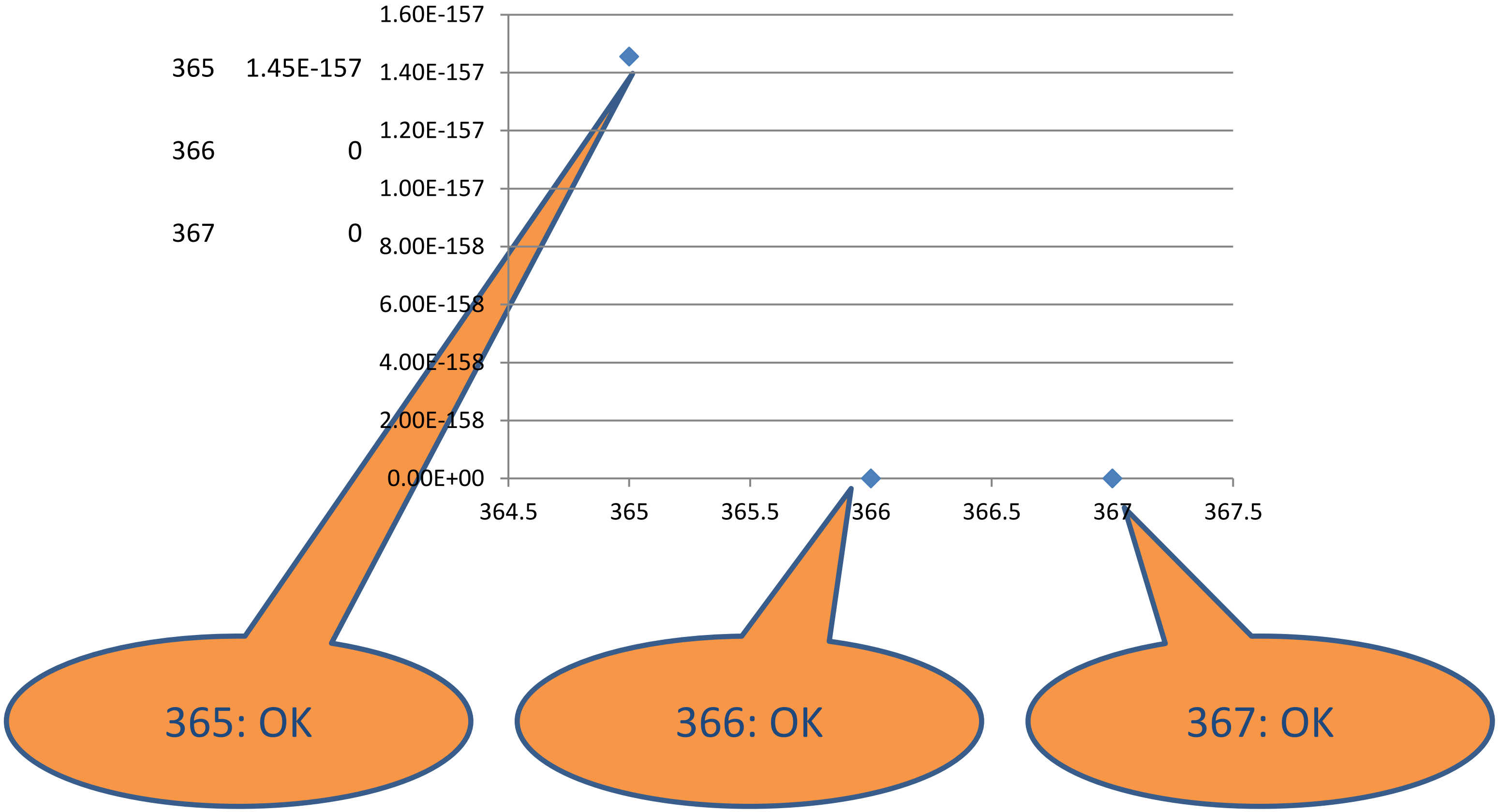


Designing Component-Level Function Tests

Boundary Conditions

uniqueBirthday

365	1.45E-157
366	0
367	0

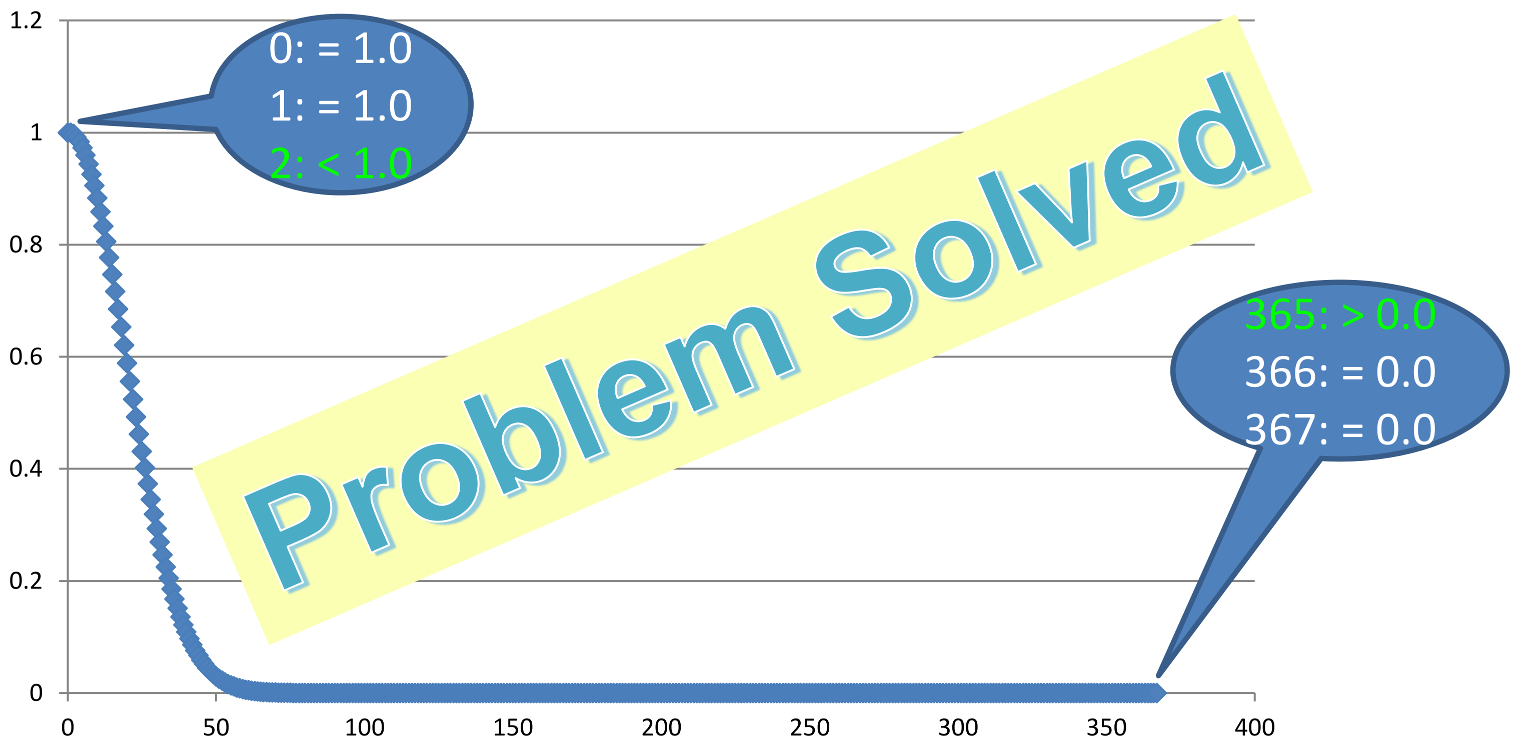


Designing Component-Level Function Tests

Boundary Conditions

uniqueBirthday

```
double uniqueBirthday(int n);
```



Boundary Conditions

Observations

- We cannot expect to know everything ***a priori***.
- The purpose of our initial testing here was to verify certain basic assumptions.
- Creating just a few thoughtful tests at critical ***interface*** *boundaries* helped us to discover important ***platform*** *boundaries*, which, in turn, led us to redesign our function's interface.
- Thorough testing does not necessarily tell us how to solve a problem; it does, however, alert us to when we haven't done so yet.

Photo by [Fausto García-Menéndez](https://unsplash.com) on [Unsplash](https://unsplash.com)

Combinatorics is fun!

Today's main topic:

How much of the world's water is dinosaur pee?

How did this topic come up?

Meet my stepdaughter Ava

Quantifying Dinosaur Pee

How did this topic come up?

Considerations

Many dinosaurs peeing

Many liters of water on earth

Many millions of years of peeing

Many molecules in a liter of water

Could it be that every molecule of water on earth is tainted?

What is the probability of that?

What's your intuition?

I came up with four questions

Assume that *all* dinosaurs have **equal access** to *all* water (e.g., no polar ice caps).

Remember your answers and score yourself at the end of the talk.

Question 1

What *proportion* of the Earth's water is dinosaur pee today?

(a) 0

(b) almost 0

(c) some

(d) almost 1

(e) 1

Quantifying Dinosaur Pee

Question 2

What is the probability that *ALL* molecules of water on earth were once inside a dinosaur?

(a) 0

(b) almost 0

(c) somewhere in the middle

(d) almost 1

(e) 1

Question 3

Suppose the *expected* amount of untainted water turns out to be on the order of a single molecule; what would then be the probability that *ALL* the molecules were tainted?

(a) 0

(b) almost 0

(c) somewhere in the middle

(d) almost 1

(e) 1

Quantifying Dinosaur Pee

Question 4

If I have an 8-ounce (236.5 ml) glass of water today, what is the probability that it has *no* dinosaur pee in it?

(a) 0

(b) almost 0

(c) somewhere in the middle

(d) almost 1

(e) 1

Quantifying Dinosaur Pee

Question Recap

Record your answers now please

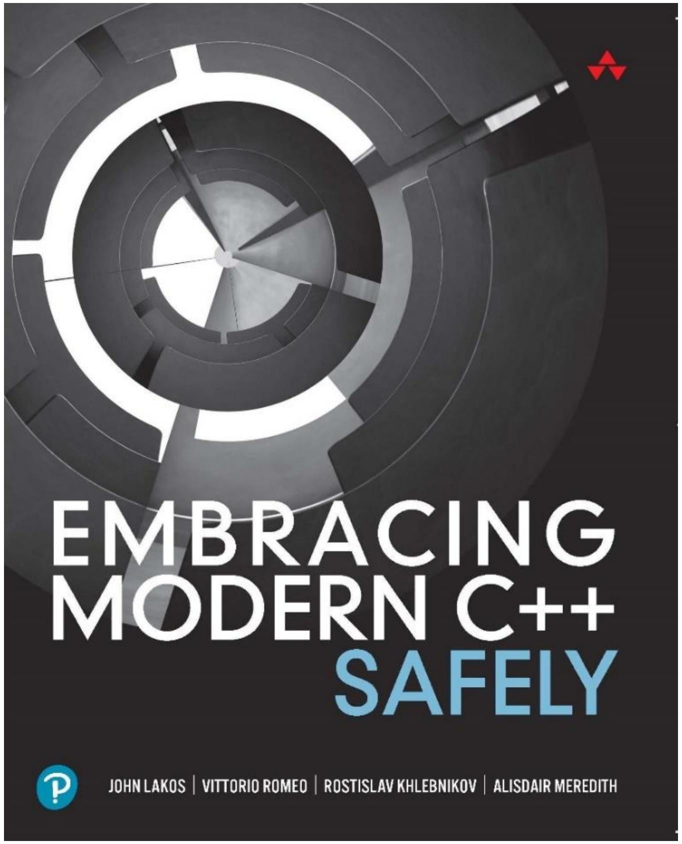
1. What fraction of all water is pee?
2. What's the chance every molecule is pee?
3. What would ans. **2.** be if ans. **1.** were "*all but 1 molecule*"?
4. What's the chance my cup of water is pure?

Answers: (a) **0** (b) **0+** (c) ~**0.5** (d) **1-** (e) **1**

Quantifying Dinosaur Pee

The answers would have to wait!

I was busy writing EMC++S!



Quantifying Dinosaur Pee

The answers would have to wait!

Over time, I accumulated some facts:

- Dinosaurs lived for about **165 million years**.
- Earth has about **1.26e21 liters (kg*)** of water.

1,260,000,000,000,000,000,000 liters

(1,260 Billion Billion liters)

Open question

How much water did dinosaurs drink per year?

*Note that a kg and a liter of H_2O are the same at 1 ATM and 4º C.

Finally, I was done with EMC++S

I used the simplifying assumption:

- All dinosaurs have access to all water

I also made a WAG (wild-ass guess):

- Dinosaur mass = animal mass today

More Facts:

- All animals today = ~4e9 tons = 3.62874e12 kg
- A reptile pees between 10-30 ml per kg per day.

Let's call it 20 ml/kg/day of dino pee

Quantifying Dinosaur Pee

Q1. How much water is pee?

Deriving the *annual* rate of dino pee

total dinosaur mass: **3.62874e12 kg**

daily rate of dino pee per kg: **20 ml/day/kg**

total *daily* dino pee:

0.02liter/day/kg * 3.62874e12 kg = **7.25748e10 liter/day**

total *annual* dino pee:

365.25 day/year * 7.25748e10 liter/day

= **2.6507946e13 liter/year**

Quantifying Dinosaur Pee

Q1. How much water is pee?

Recall differential equations

Let $y(t)$ = untainted water (liters) at time t.

Let $\boldsymbol{R}$ = initial *fractional* rate (time^{-1}) at which water is consumed…

(…and peed).

$$\frac{dy}{dt} = -Ry \qquad \frac{dy}{y} = -R\,dt$$

$$\int \frac{1}{y}\,dy = \int -R\,dt$$

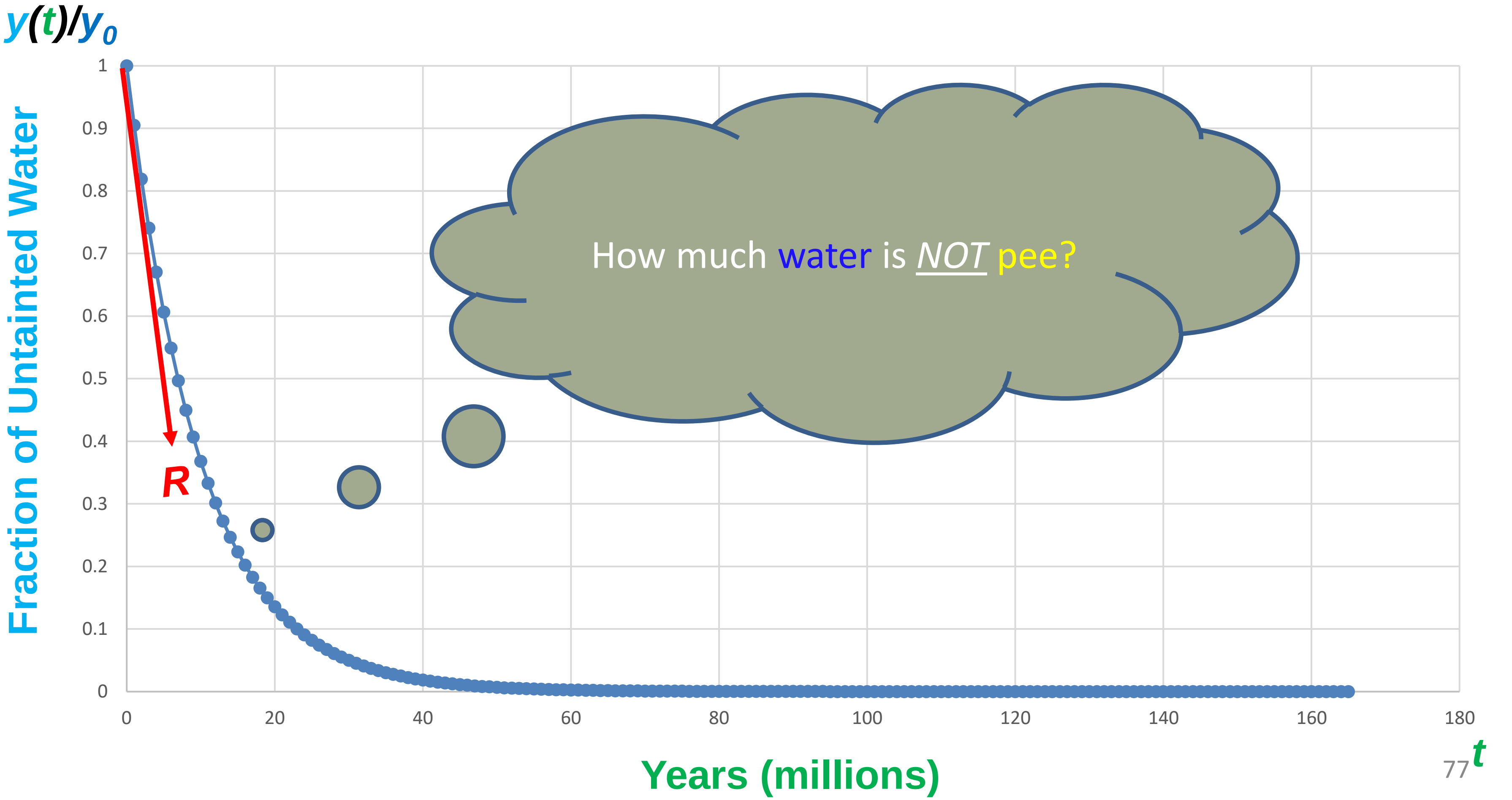
$$\ln y = -Rt + Constant$$

$$y(t) = y_0 e^{-Rt}$$

Total Water

Q1. How much water is pee?

Modeling the problem with e^{-Rt}



Q1. How much water is pee?

Deriving the annual rate of decay, *R*

$$\boldsymbol{R} = \frac{AnnualPee}{y_0 = TotalWater}$$

$$= \frac{2.6507946\text{e}13 \text{ liters/year}}{1.26\text{e}21 \text{ liters}}$$

$$= \quad 2.10381\text{e-}08/\text{year}$$

Note: *R* = –ln(1 – *AnnualPee*/*TotalWater*) but, for sufficiently small x, e^{-x} ~= 1 – x

Q1. How much water is pee?

Plug in the *R* and *t* and get the answer

$$y(t) = y_0 e^{-Rt}$$

years

years^{-1}

years

$$\frac{y(165,000,000)}{y_0} = e^{-2.10381e-08 * 165,000,000}$$

unit-less

$$= e^{-3.465}$$

$$= 0.031077024$$

Answer ➡ $= 3.11\%$ (untainted) (D?)

(Almost all?)

Quantifying Dinosaur Pee

Q1. How much water is pee?

What if we use *daily* *R* and *t*?

$$R_{daily} = \frac{R}{365.25} \qquad t_{daily} = 365.25t$$

$$\begin{aligned} y_{daily}\left(t_{daily}\right) &= y_0 e^{-R_{daily}t_{daily}} \\ &= y_0 e^{-365.25\frac{Rt}{365.25}} \\ &= y_0 e^{-Rt} = y(t) \end{aligned}$$

No Difference!

It all cancels out.

Quantifying Dinosaur Pee

Q1. How much water is pee?

Another Model: discrete probability

Let t be time (in units of years)— i.e., Δ_t = 1 year

Let R be the fractional decay rate (in units of time^{-1})

Let y be the expected quantity of pure water:

$$y(t) \cong y_0(1 - R\Delta_t)^{t/\Delta_t}$$

Why?

Q1. How much water is pee?

Another Model: discrete probability

Because…

$$e^{x} = \lim_{\mu \to \infty} \left(1 + \frac{x}{\mu}\right)^{\mu}$$

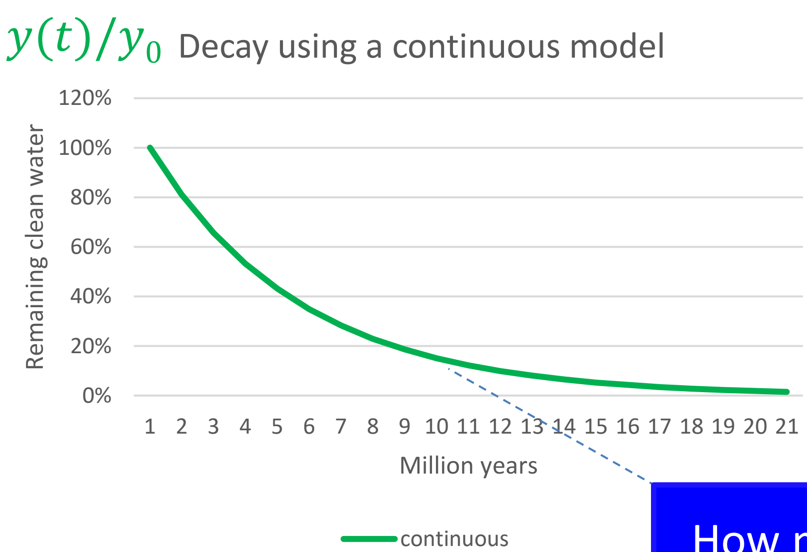
(We'll talk about compounding shortly)

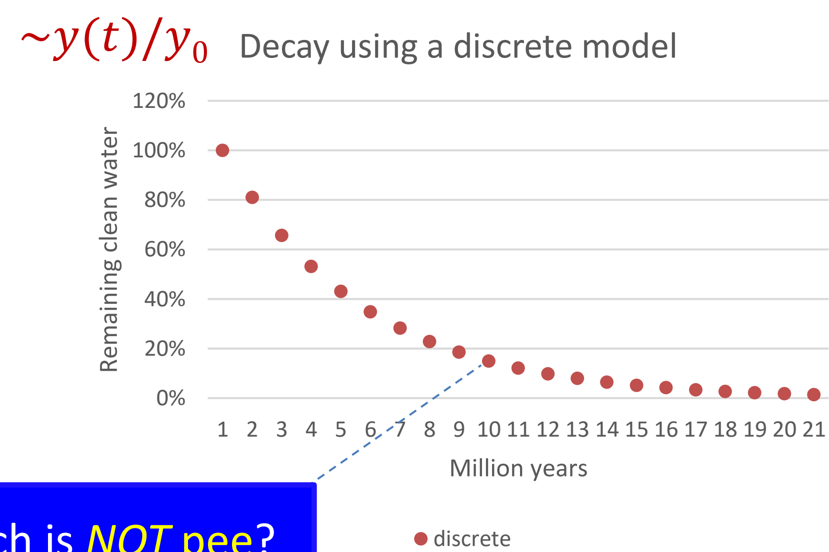
w

Q1. How much water is pee?

Another Model: discrete probability

$$y(t)/y_0 = e^{-Rt} \qquad y(t)/y_0 \cong (1 - R\Delta_t)^{t/\Delta_t}$$





How much is ***NOT*** pee?

Quantifying Dinosaur Pee

Q1. How much water is pee?

Does discrete probability make a difference?

million years	continuous	discrete (annual)	difference*
0	1.00000000000000	1.00000000000000	0.00000000000000
25	0.59099217614322	0.59099217287355	0.00000000378656
50	0.34927175226250	0.34927174839780	0.00000000447565
75	0.20641687293497	0.20641686950896	0.00000000396761
100	0.12199075692852	0.12199075422885	0.00000000312644
125	0.07209558290654	0.07209558091219	0.00000000230962
150	0.04260792543225	0.04260792401787	0.00000000163796
165	0.03107702434069	0.03107702320593	0.00000000131415

Data:
t <= 165M years (ours)
R = 2.10381e-8/year (ours)

How much is *pure*?

*difference = abs(discrete – continuous)

Compound Interest

$$A = P(1 + I)^{N}$$

A = final amount P = principal

I = interest rate (annual) N = time periods (years)

$$A_k = P(1 + I/k)^{kN}$$

k = compounding factor

For daily compounding k ≅ 365.25

Note, financial products that have daily compounding are often marketed with a different annual percentage rate (APR). For example, a 2% APR results from a 1.98% rate compounded daily.

Quantifying Dinosaur Pee

Compounding

Let t be time (in units of years)— i.e., Δ_t = 1 year

Let R be the fractional decay rate (in units of time^{-1})

Let y be the expected quantity of pure water (liters)

$$y(t) \cong y_0(1 - R\Delta_t)^{t/\Delta_t}$$

Compounding

Let t be time (in units of years)— i.e., Δ_t = 1 year

Let R be the fractional decay rate (in units of time^{-1})

Let y be the expected quantity of pure water (liters)

Let k be the compounding factor:

$$\boldsymbol{y(t) \cong y_0 \left(1 - \frac{R}{k}\Delta_t\right)^{kt/\Delta_t}}$$

Quantifying Dinosaur Pee

How much water is pee?

Using *infinite precision* with our data

compounding cycle	k	$y(t)/y_0$	error* (%)
continuous	$k \to \infty$	0.03107702434070	0
per day	365.25	0.03107702433759	0.00000000999716
per year	1	0.03107702320593	0.00000365146361
per 10 years	10^{-1}	0.03107701299304	0.00003651463471
per 100 years	10^{-2}	0.03107691086412	0.00036514620805
per 1,000 years	10^{-3}	0.03107588957926	0.00365144817284
per 10,000 years	10^{-4}	0.03106567715860	0.03651309073776
per 100,000 years	10^{-5}	0.03096359581703	0.36499158486497
per 1,000,000 years	10^{-6}	0.02994713891235	3.63575809563761

$$* \text{ error} = 100 \times \frac{|\text{discrete} - \text{continuous}|}{\text{continuous}}$$

Data:
t = 165M years (ours)
R = 2.10381e-8/year (ours)

Q1. How much water is pee?

Other attempts at this problem

I watched a great video on this subject by *In-que-ri-ty*.

Inquerity19: https://youtu.be/i67uCBegsyU

Quantifying Dinosaur Pee

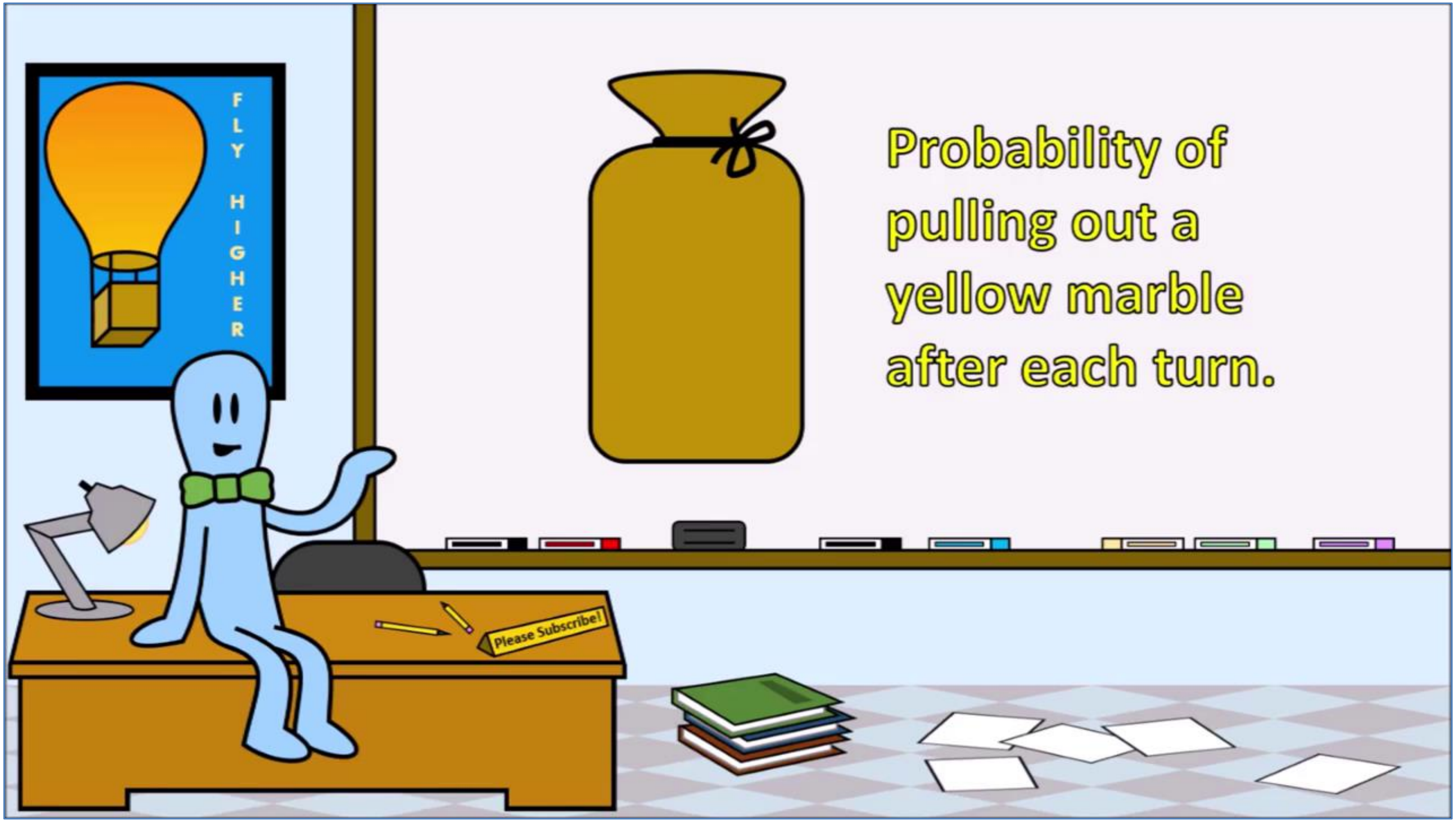
Inquerity's Approach

Reprinted with permission of Inquerity Studios: **Inquerity19** @ 0:0

Inquerity's Approach

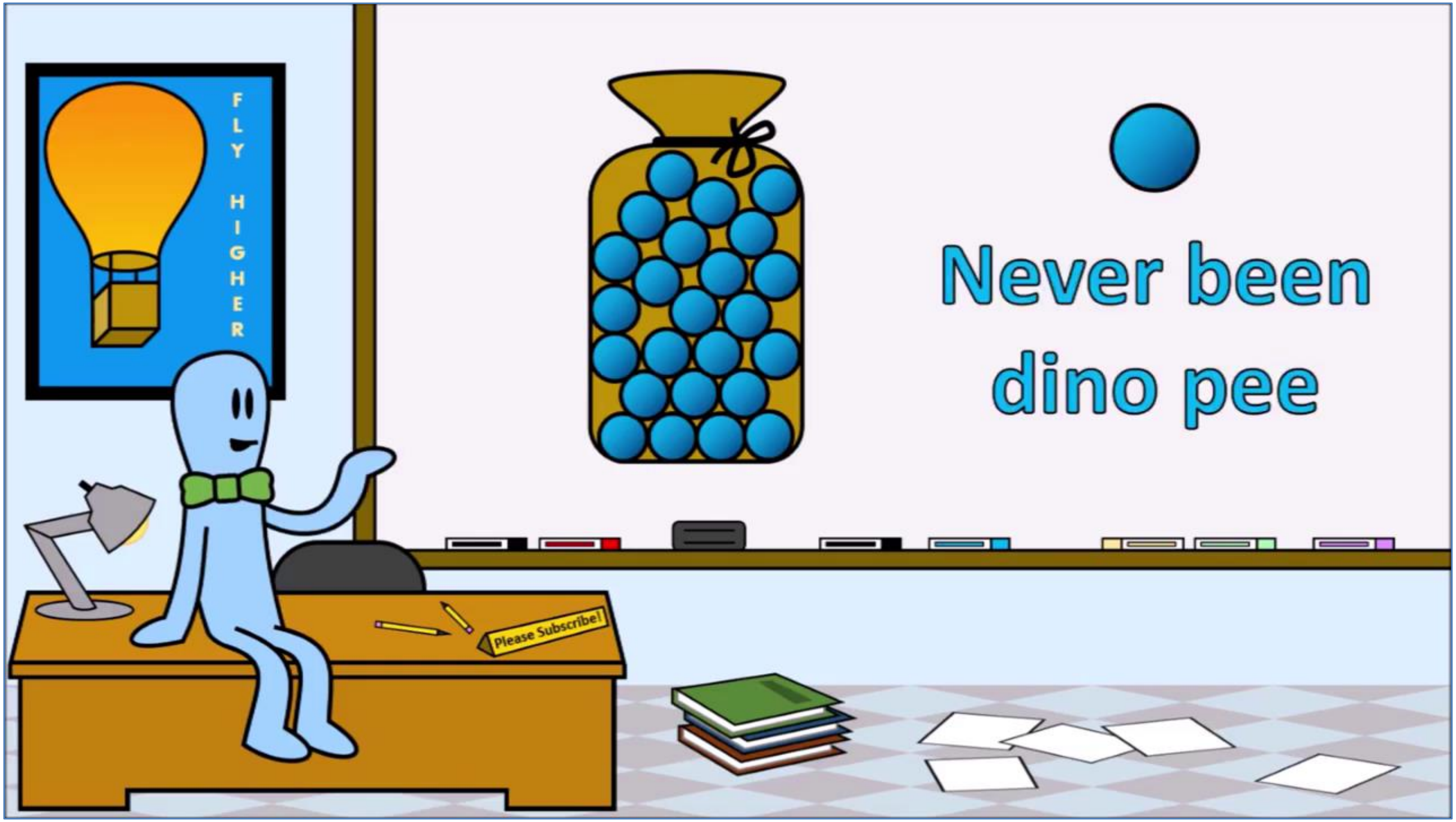
Reprinted with permission of Inquerity Studios: **Inquerity19** @ 0:0

Inquerity's Approach



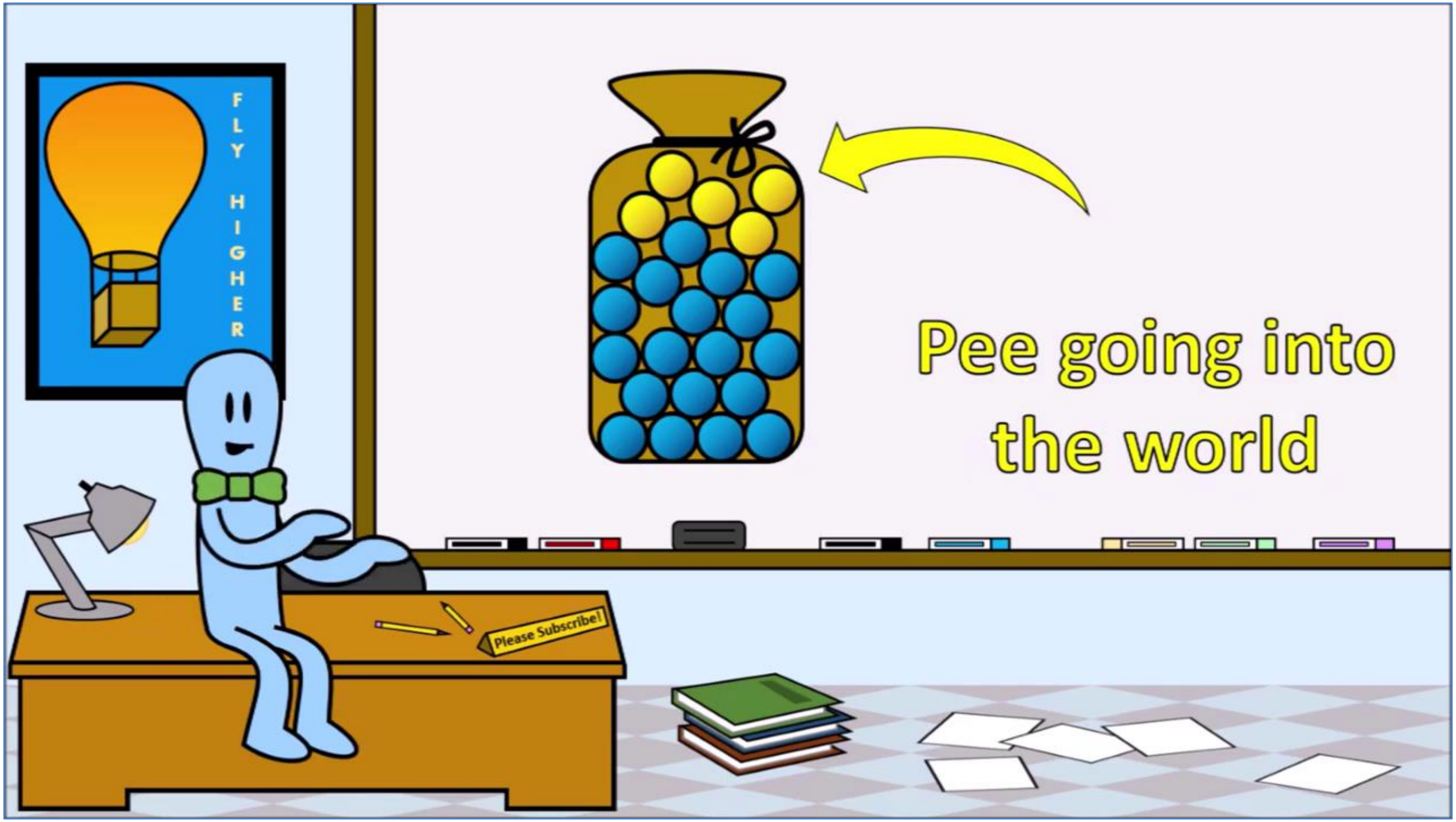
Reprinted with permission of Inquerity Studios: **Inquerity19** @ 2:23

Inquerity's Approach



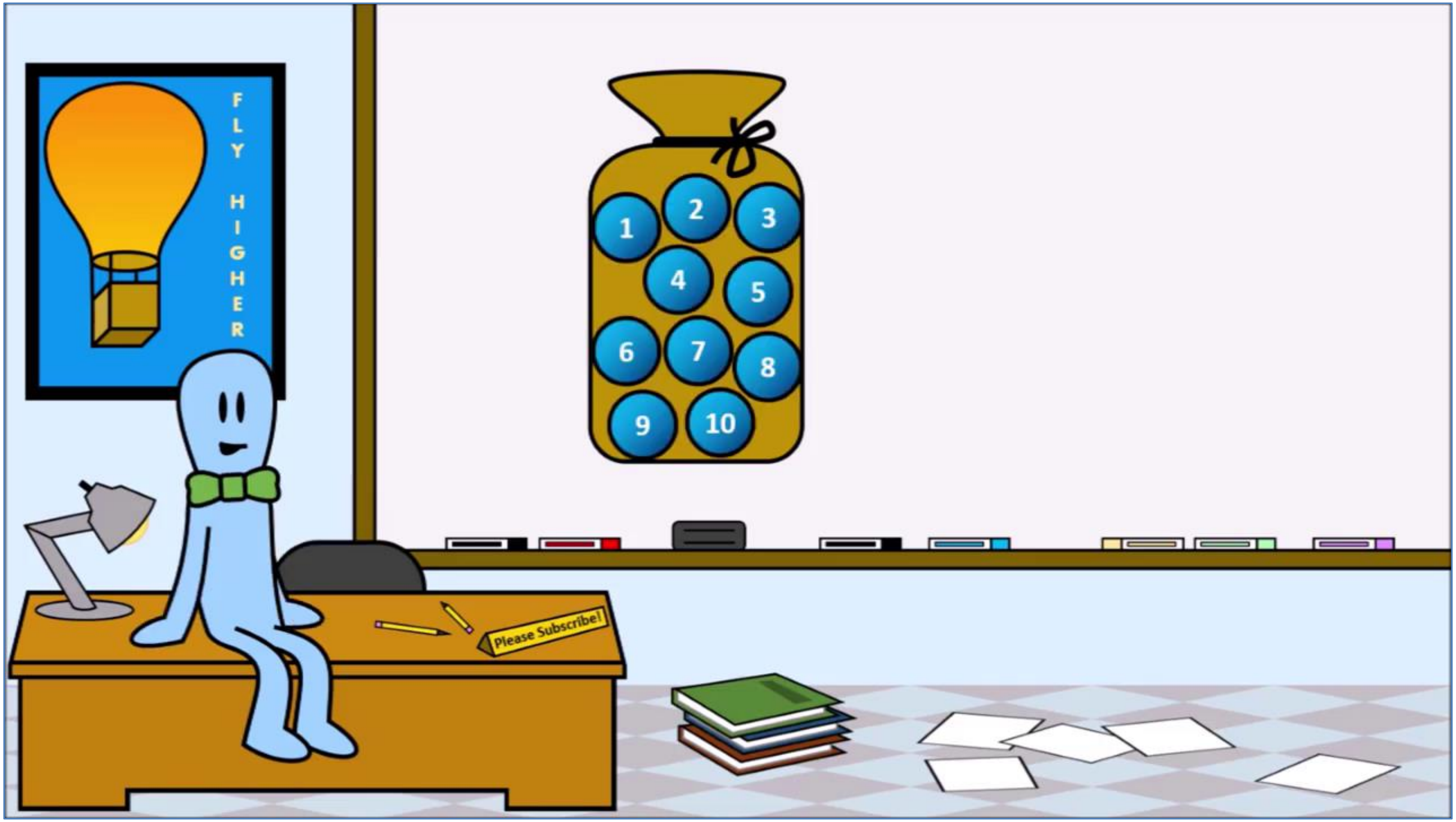
Reprinted with permission of Inquerity Studios: **Inquerity19** @ 1:16

Inquerity's Approach



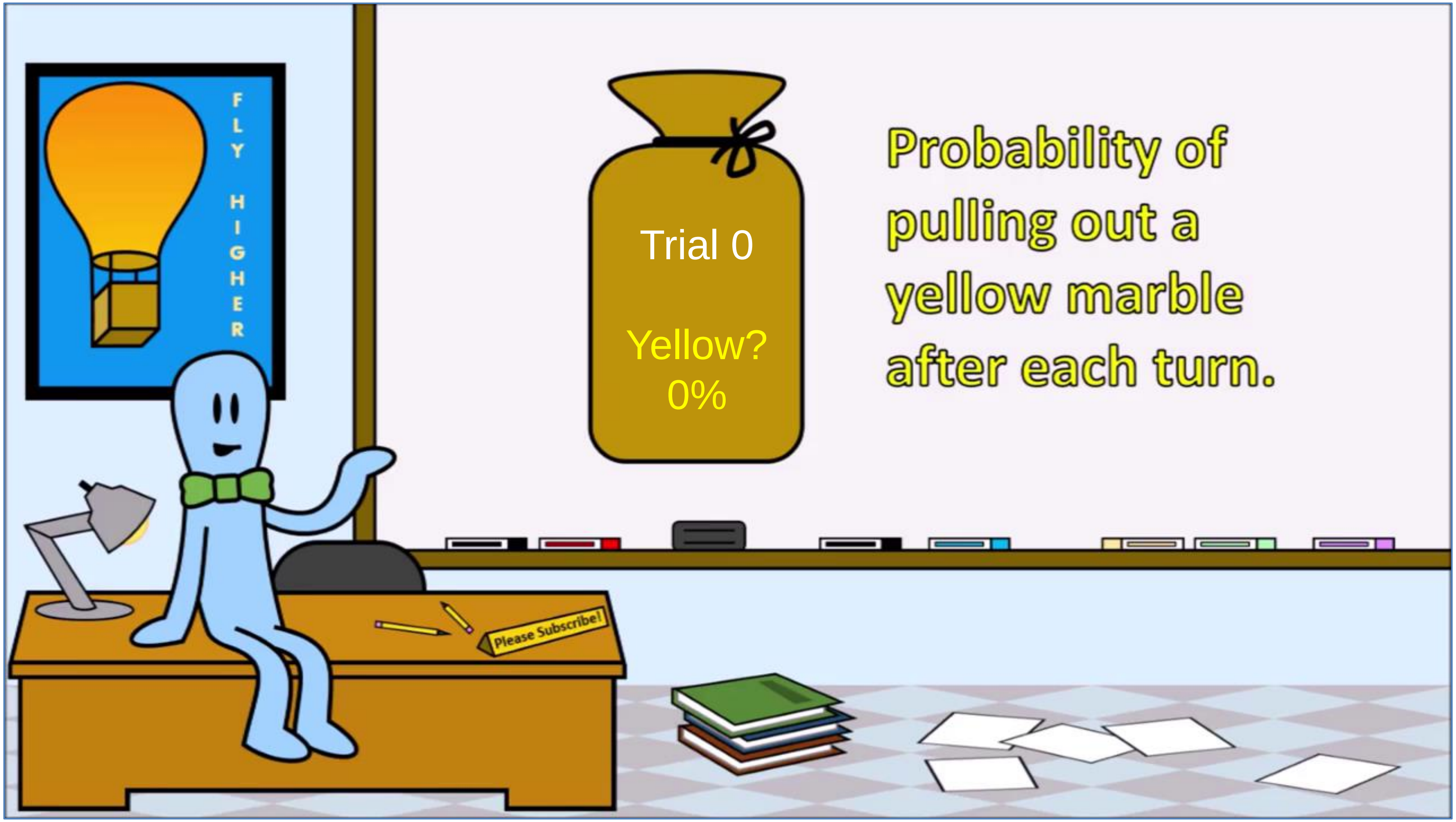
Reprinted with permission of Inquerity Studios: **Inquerity19** @ 1:38

Inquerity's Approach



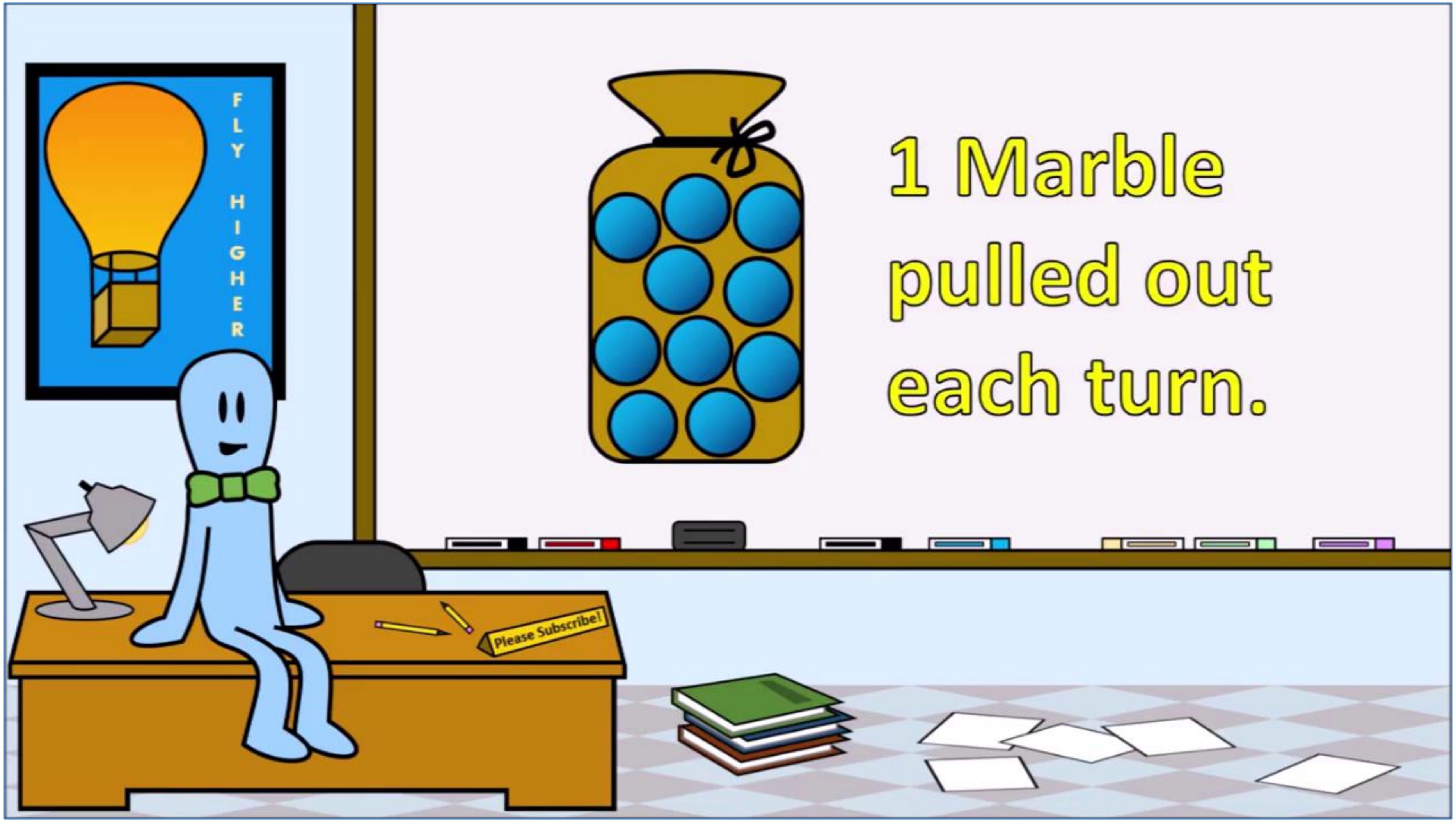
Reprinted with permission of Inquerity Studios: **Inquerity19** @ 2:30

Inquerity's Approach



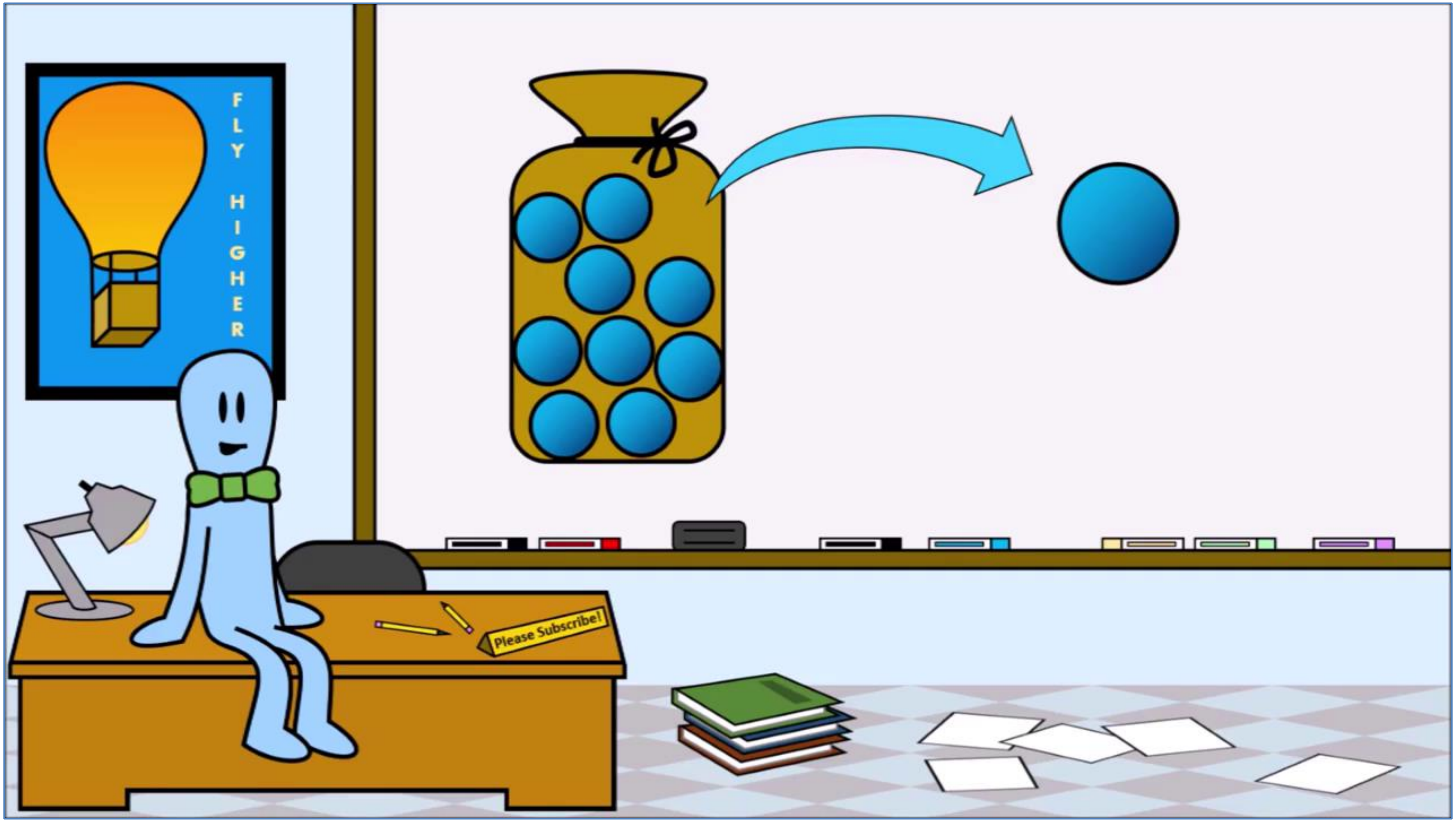
Reprinted with permission of Inquerity Studios: **Inquerity19** @ 2:23

Inquerity's Approach



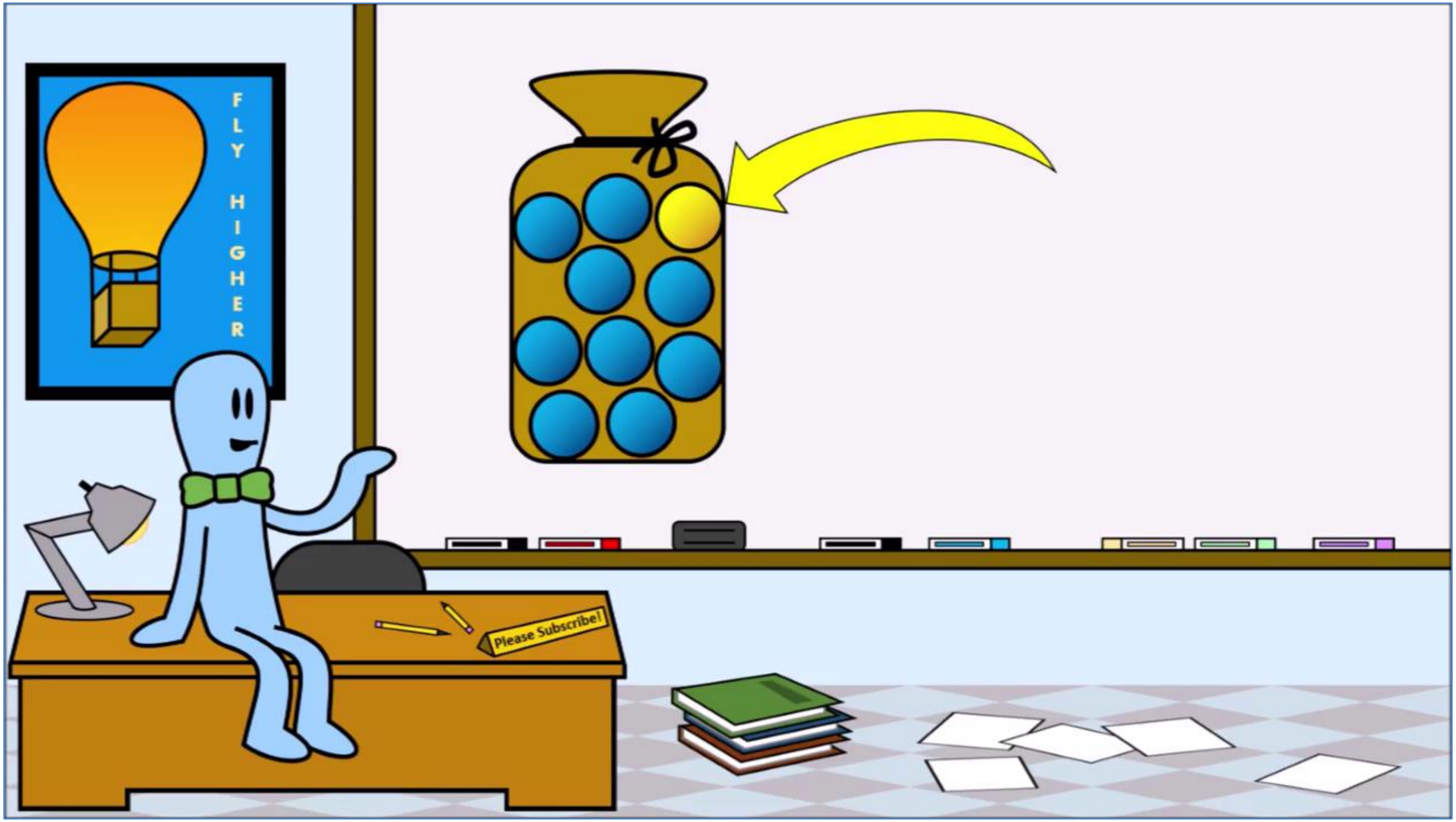
Reprinted with permission of Inquerity Studios: **Inquerity19** @ 2:31

Inquerity's Approach



Reprinted with permission of Inquerity Studios: **Inquerity19** @ 2:42

Inquerity's Approach

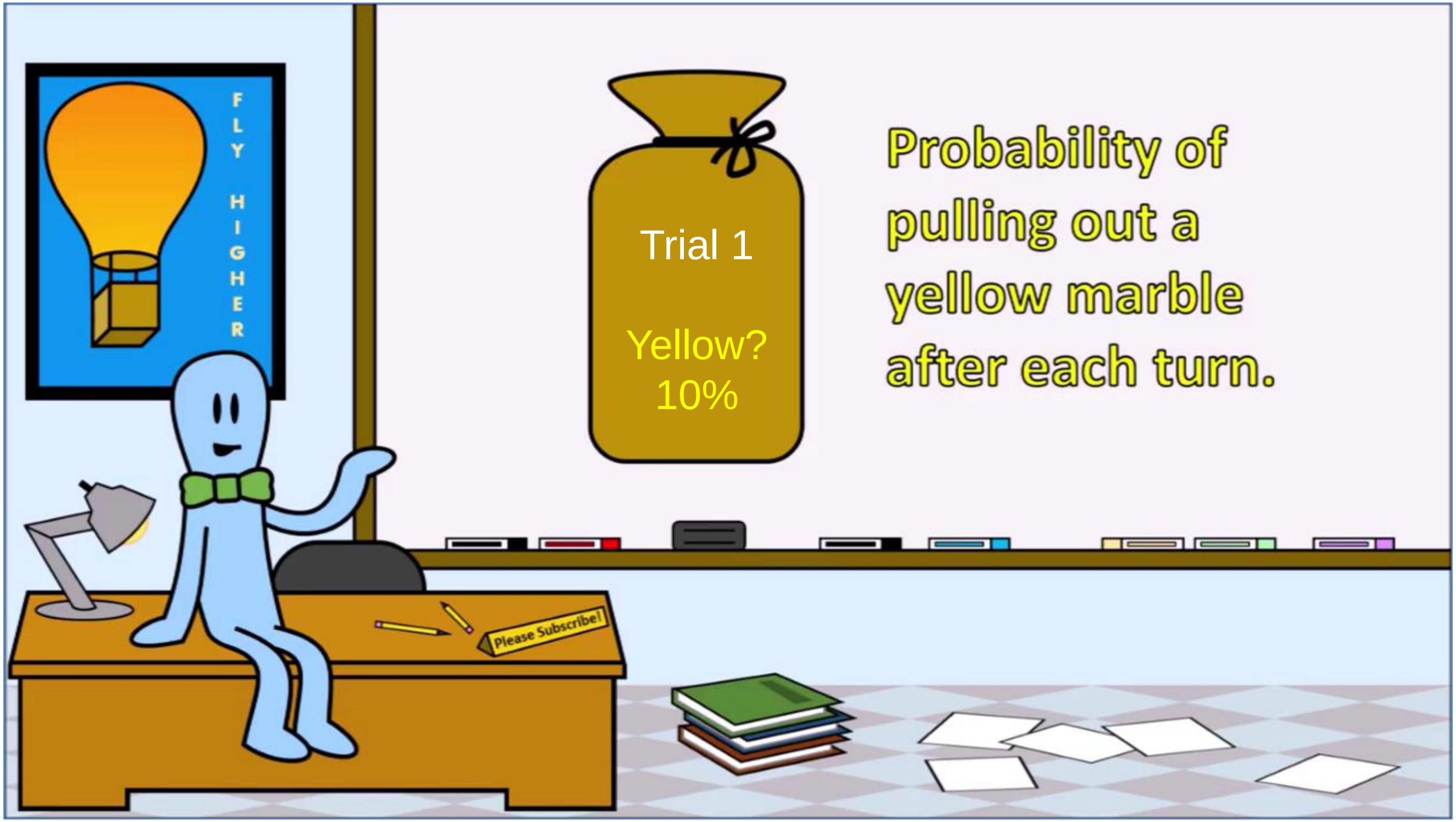


Reprinted with permission of Inquerity Studios: **Inquerity19** @ 2:45

Inquerity's Approach

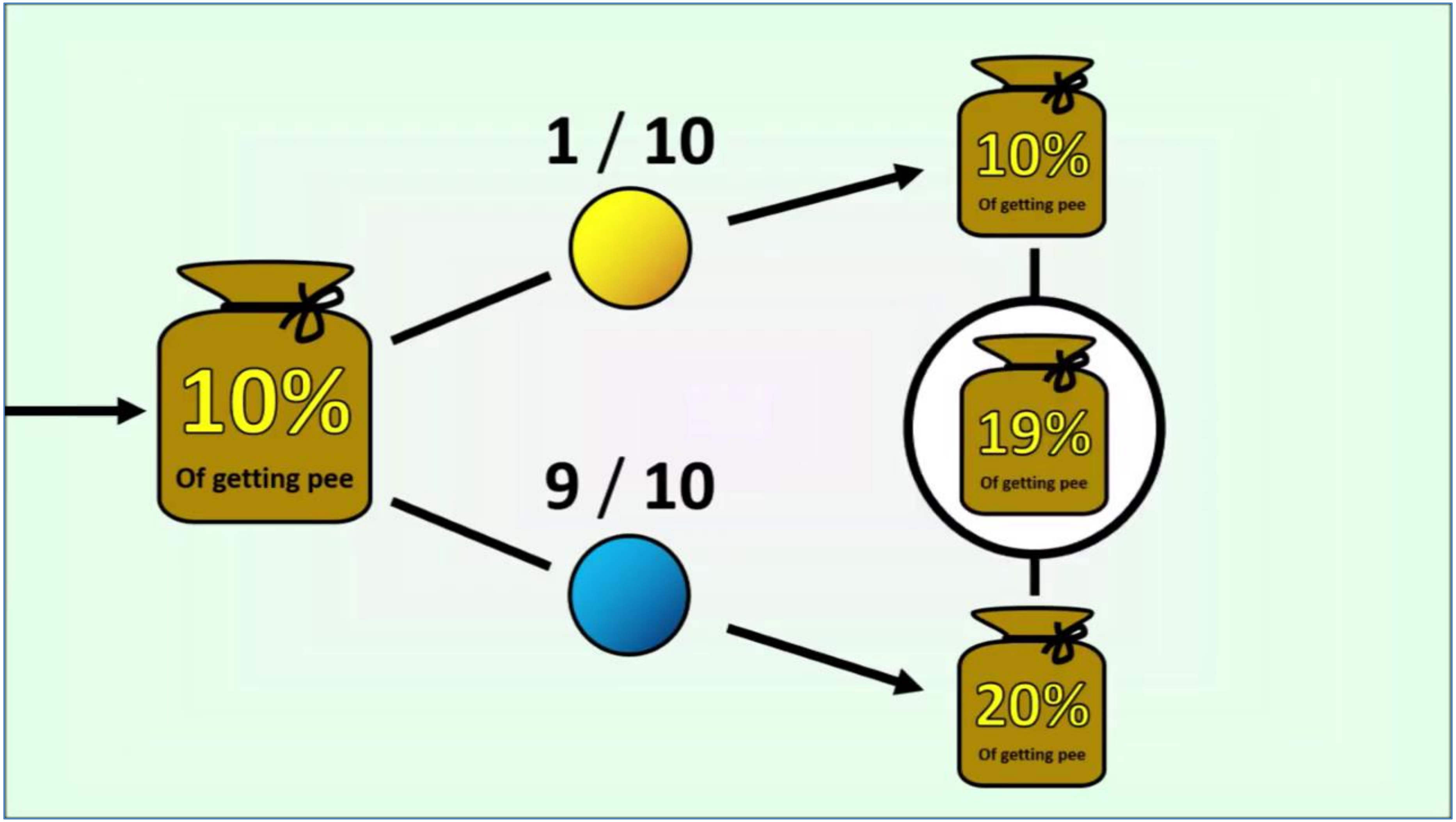
Reprinted with permission of Inquerity Studios: **Inquerity19** @ 2:49

Inquerity's Approach



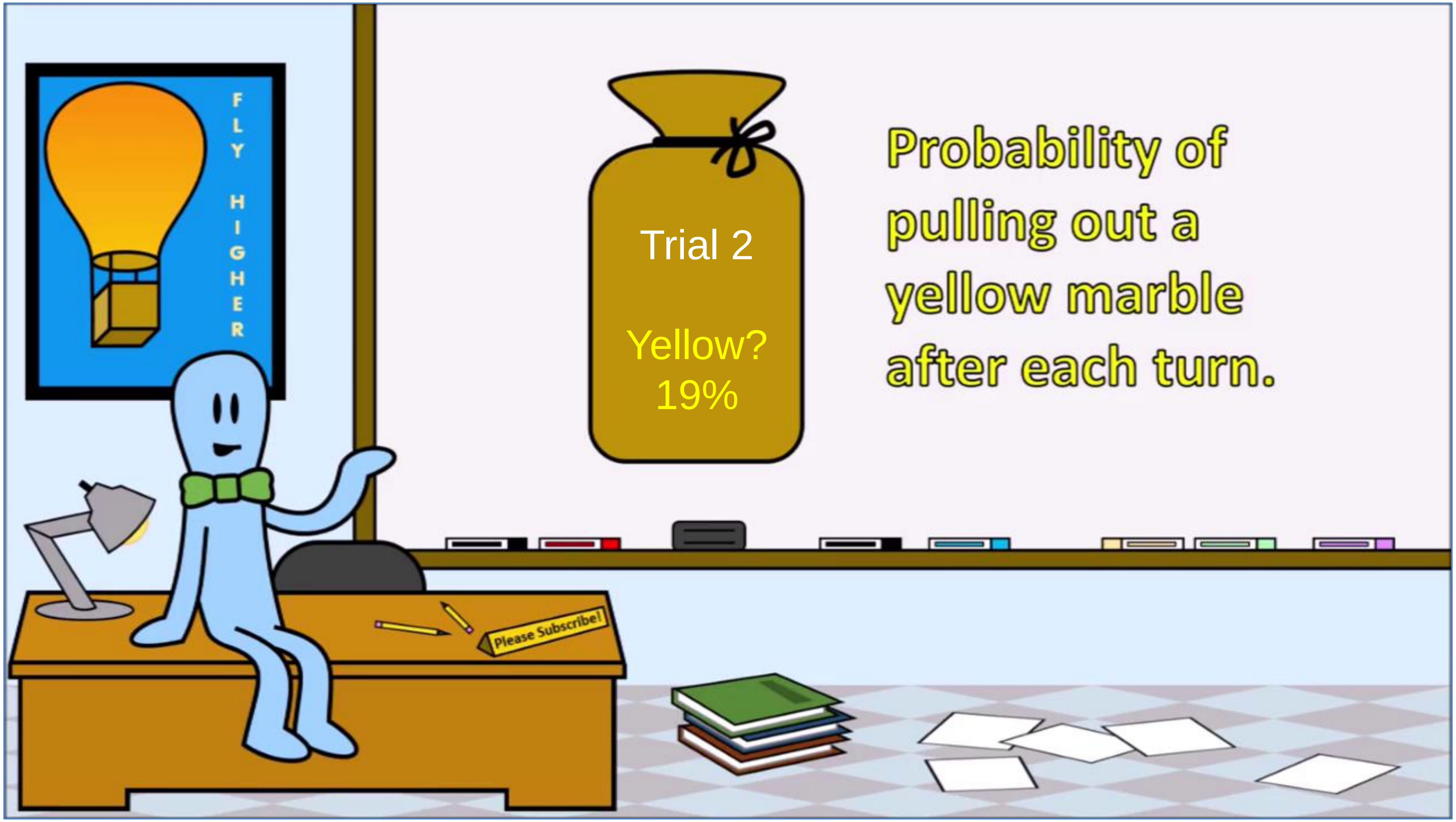
Reprinted with permission of Inquerity Studios: **Inquerity19** @ 2:23

Inquerity's Approach



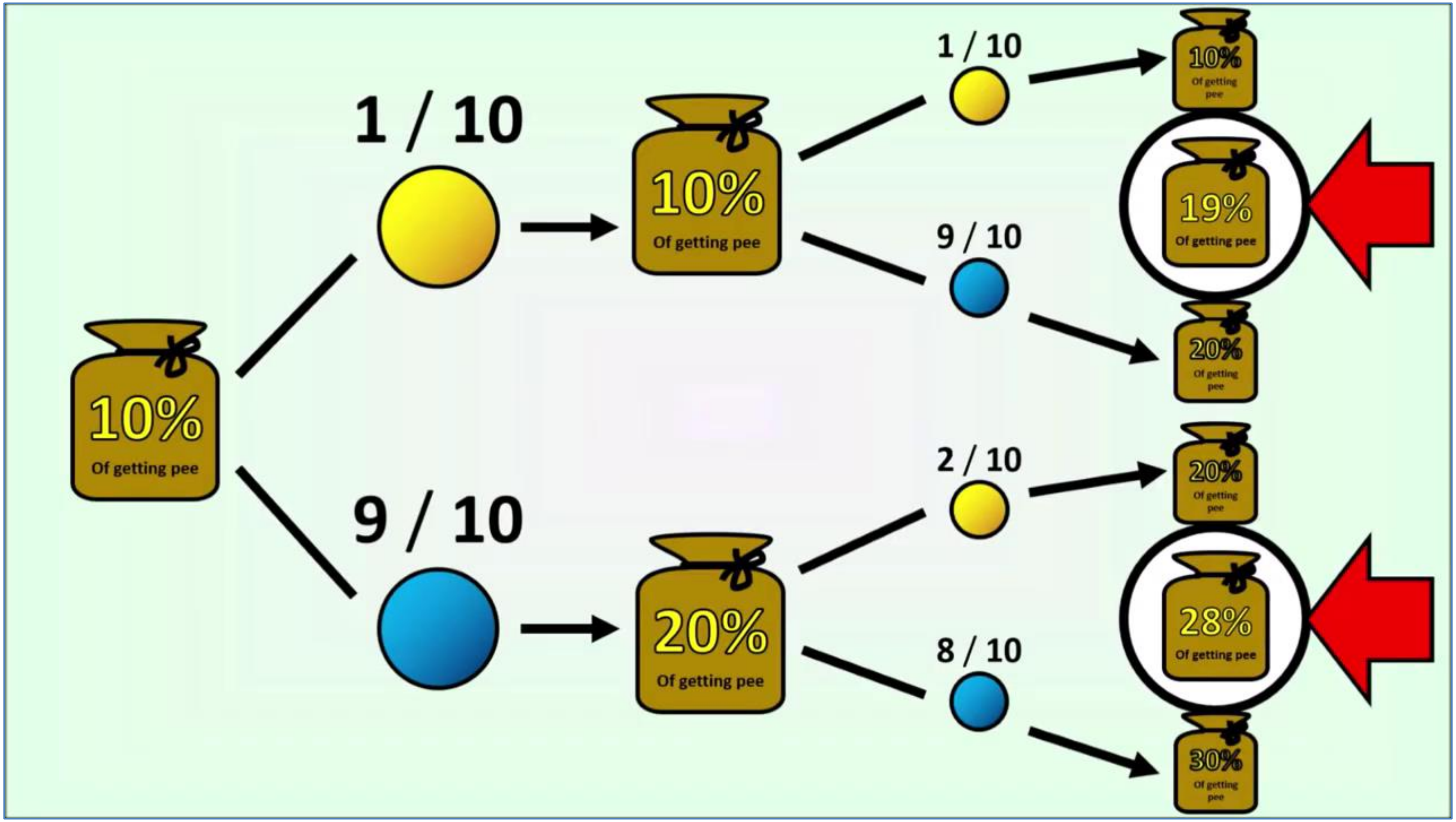
Reprinted with permission of Inquerity Studios: **Inquerity19** @ 4:12

Inquerity's Approach



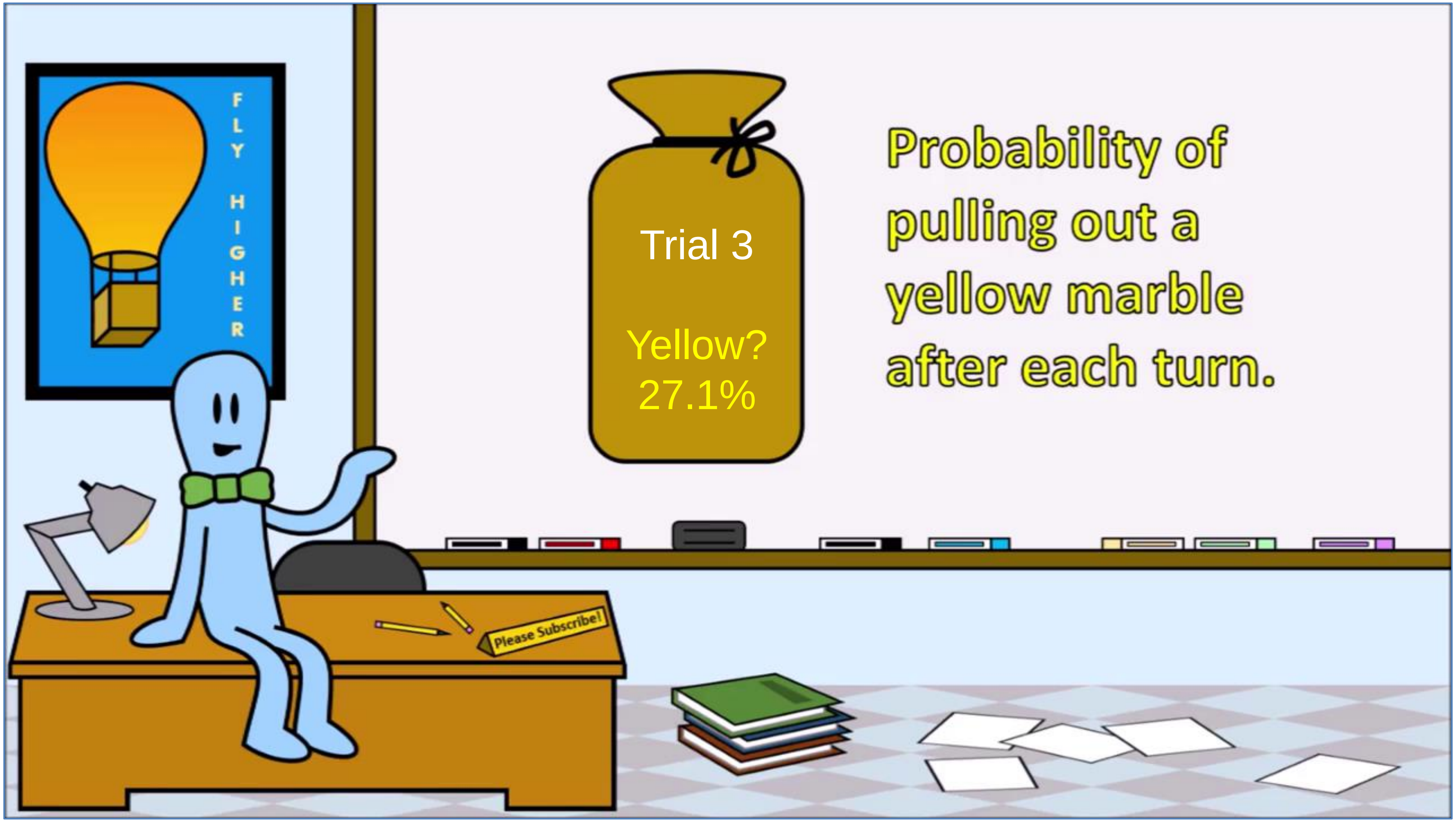
Reprinted with permission of Inquerity Studios: **Inquerity19** @ 2:23

Inquerity's Approach



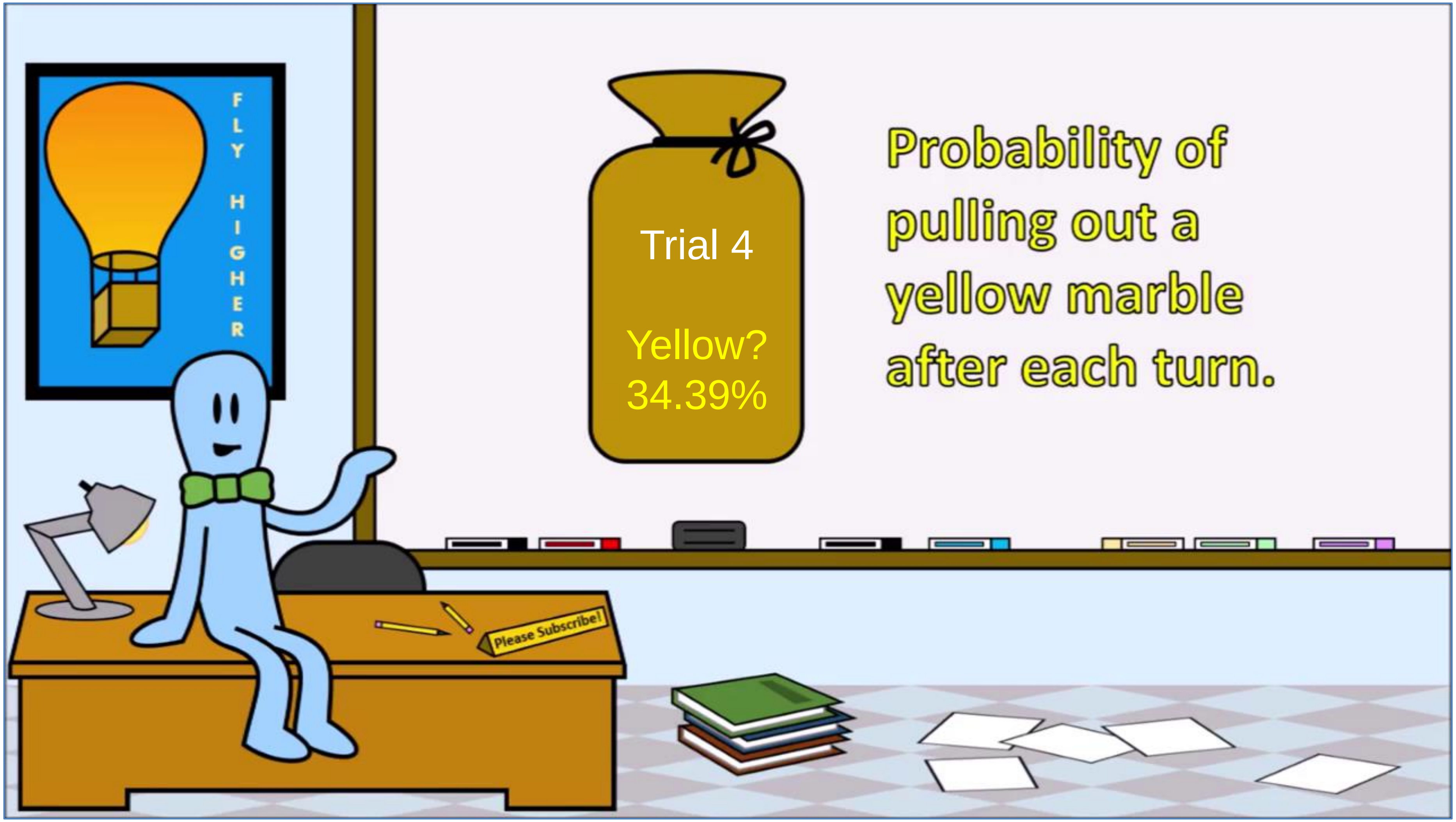
Reprinted with permission of Inquerity Studios: **Inquerity19** @ 4:15

Inquerity's Approach



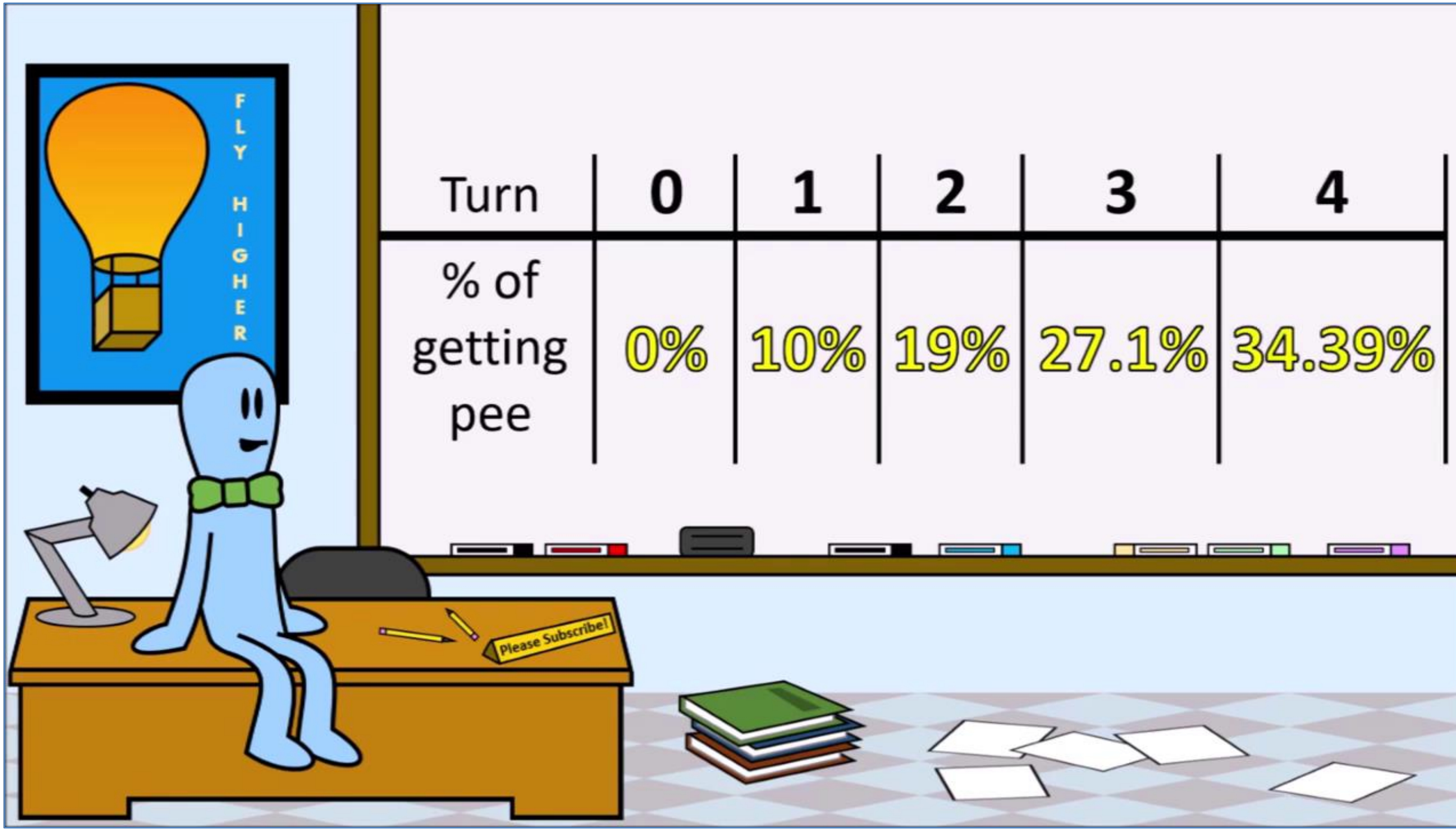
Reprinted with permission of Inquerity Studios: **Inquerity19** @ 2:23

Inquerity's Approach



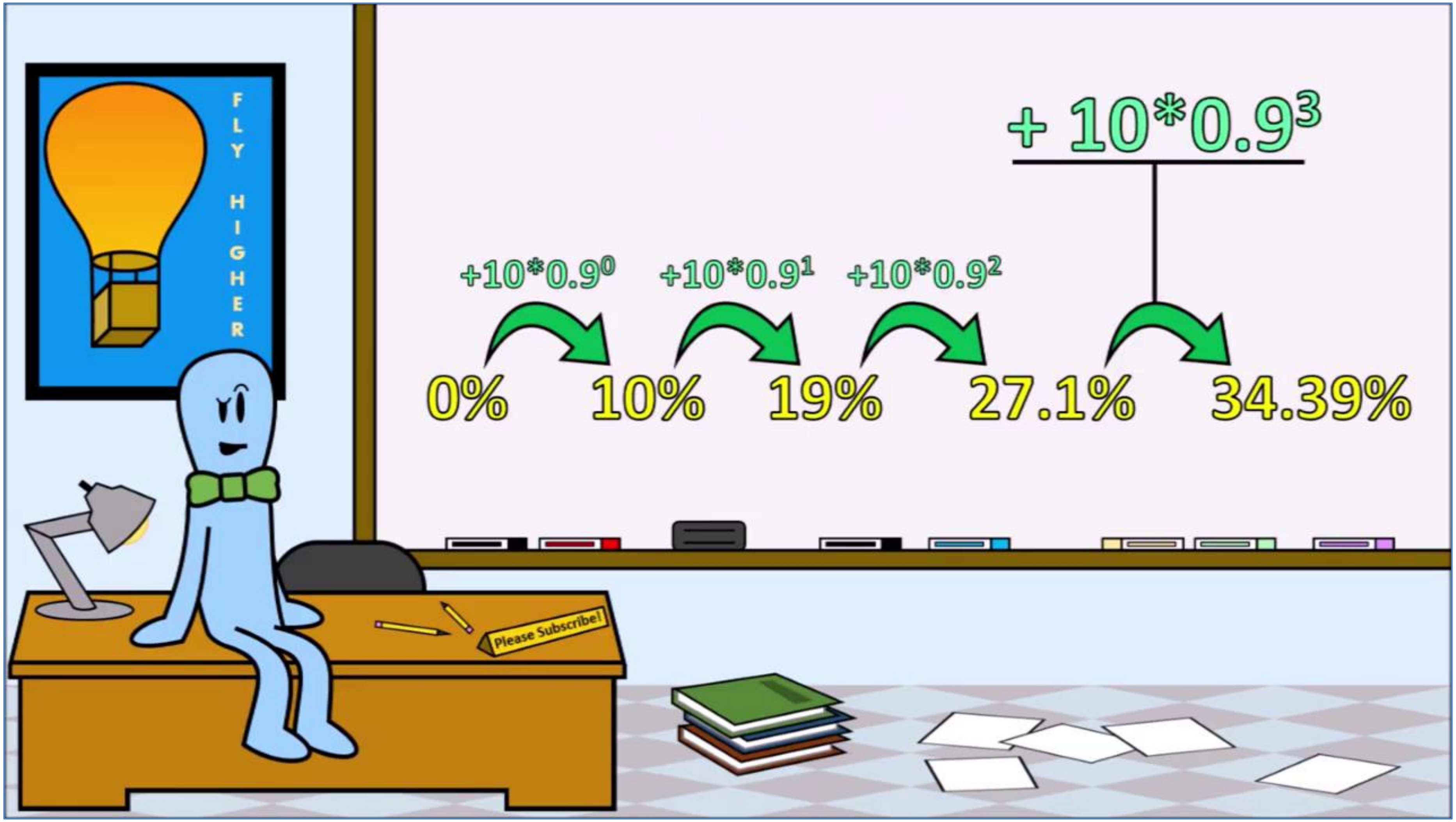
Reprinted with permission of Inquerity Studios: **Inquerity19** @ 2:23

Inquerity's Approach



Reprinted with permission of Inquerity Studios: **Inquerity19** @ 5:11

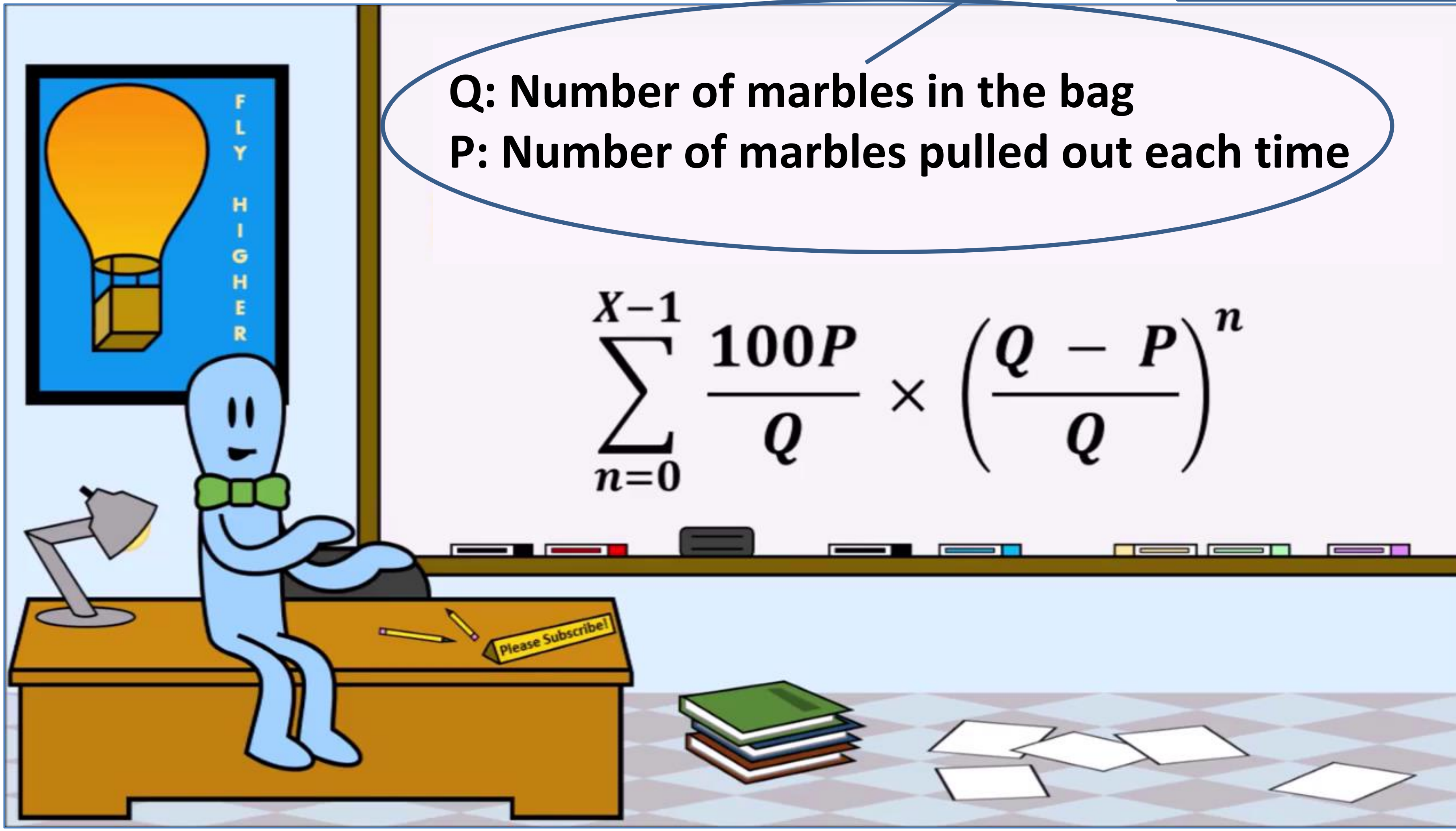
Inquerity's Approach



Reprinted with permission of Inquerity Studios: **Inquerity19** @ 5:30

Inquerity's Approach

I added these two annotations for this presentation.



Reprinted with permission of Inquerity Studios: **Inquerity19** @ 12:09

Deriving the *geometric sum* formula

$$\%Pee = \sum_{n=0}^{X-1} \frac{100P}{Q} \times \left(\frac{Q-P}{Q}\right)^n$$

Not like *our* **R**

Let $R = 1 - \frac{P}{Q}$

$$= \sum_{n=0}^{X-1} 100 \times (1-R) \times R^n$$

Let $A = 100 \times \frac{P}{Q}$

$= 100 \times (1-R)$

$$= \sum_{n=0}^{X-1} AR^n = A + AR + AR^2 + \cdots + AR^{X-1}$$

Deriving the *geometric sum* formula

$$S = \sum_{n=0}^{X-1} AR^n = AR^0 + AR^1 + AR^2 + \cdots + AR^{X-2} + AR^{X-1}$$

$$RS = AR^1 + AR^2 + \cdots + AR^{X-2} + AR^{X-1} + AR^X$$

$$S - RS = AR^0 + \cancel{AR^1} + \cancel{AR^2} + \cdots + \cancel{AR^{X-2}} + \cancel{AR^{X-1}}$$
$$- \cancel{AR^1} - \cancel{AR^2} - \cdots - \cancel{AR^{X-2}} - \cancel{AR^{X-1}} - AR^X$$

$$S(1-R) = AR^0 - AR^X$$

$$S = \frac{AR^0 - AR^X}{1-R}$$

$$= \frac{A(1-R^X)}{1-R}$$

Deriving the *geometric sum* formula

Now plugging Inquerity's symbols

$$R = 1 - \frac{P}{Q}$$

$$A = 100 \times \frac{P}{Q} = 100 \times (1 - R)$$

into *geometric sum* formula

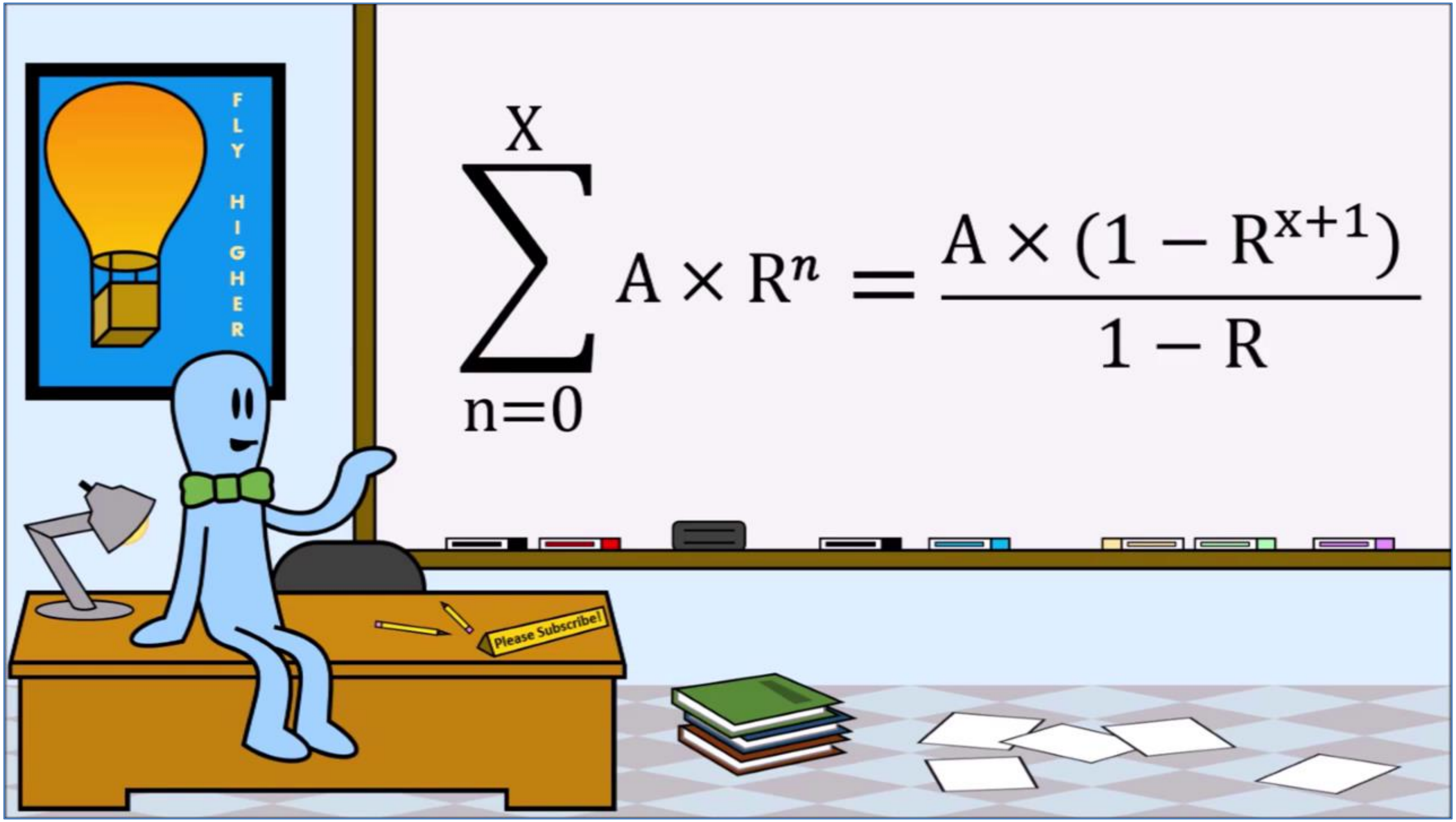
$$\sum_{n=0}^{X-1} AR^n = \frac{A(1 - R^X)}{1 - R}$$

We'll meet again

we get

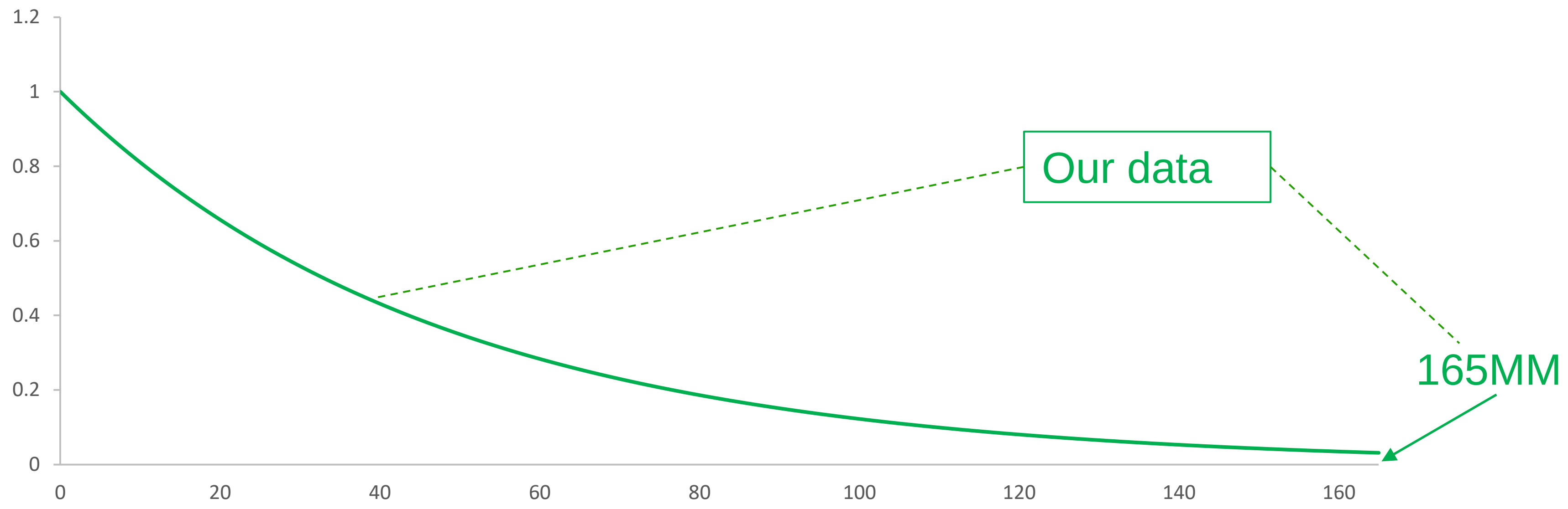
$$\%Pee = \sum_{n=0}^{X-1} \frac{100P}{Q} \times \left(\frac{Q-P}{Q}\right)^n = \boxed{100 \times \frac{P}{Q} \times \frac{\left(1 - \left(1 - \frac{P}{Q}\right)^X\right)}{1 - \left(1 - \frac{P}{Q}\right)}}$$

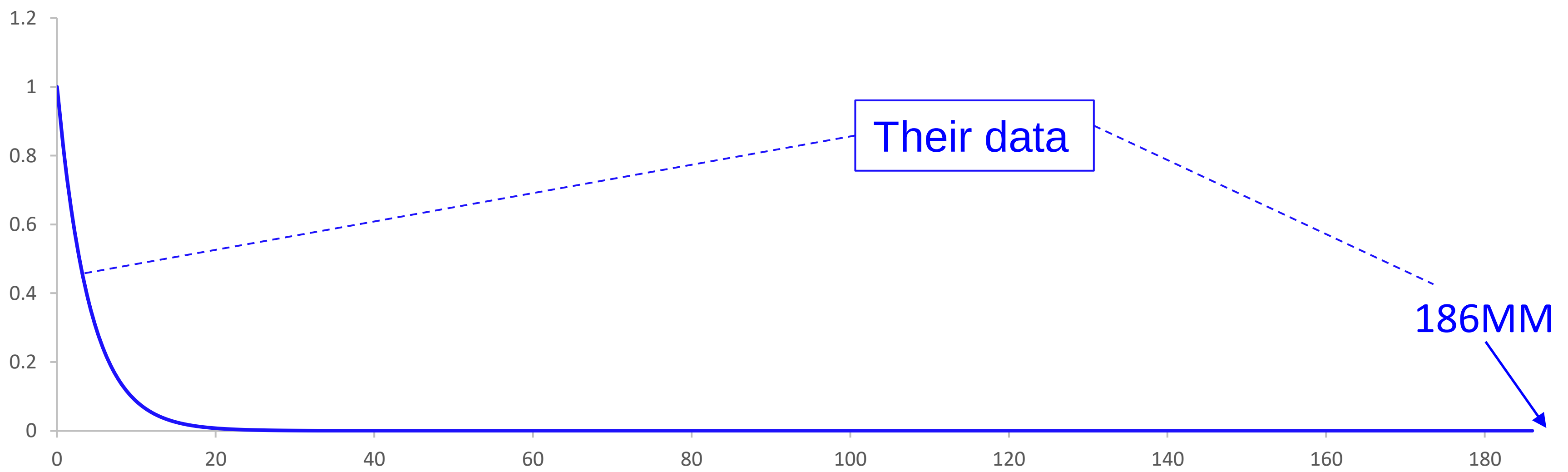
Inquerity's Approach



Reprinted with permission of Inquerity Studios: **Inquerity19** @ 12:55

Inquerity's Approach





Inquerity used "Alternative Facts"

1. Time scale:
   a. We used 165,000,000 years.
   b. They used 186,000,000 years.

1.

(~10% more)

Inquerity's Approach

Assuming *our* pee rate

$$y(t) = y_0 e^{-Rt}$$

Our time scale:

$$y(165,000,000)/y_0 = e^{-2.10381e-08*165{,}000{,}000}$$

= 0.0311 = 3.11% untainted

Their time scale:

$$y(186,000,000)/y_0 = e^{-2.10381e-08*186{,}000{,}000}$$

= 0.0200 = 2.00% untainted

Inquerity's Approach

Pure water remaining after 186M years

Using *infinite precision* with ***our*** **rate and Inquerity's time scale**

compounding cycle	k	$y(t)/y_0$	error* (%)
continuous	$k \to \infty$	0.0199787394168875	0
per day	365.25	0.0199787394146360	0.0000000112695
per year	1	0.0199787385945236	0.0000041161953
per 10 years	10^{-1}	0.0199787311932486	0.0000411619509
per 100 years	10^{-2}	0.0199786571805470	0.0004116192662
per 1,000 years	10^{-3}	0.0199779170583381	0.0041161683756
per 10,000 years	10^{-4}	0.0199705163166065	0.0411592548932
per 100,000 years	10^{-5}	0.0198965569936879	0.4113493923953
per 1,000,000 years	10^{-6}	0.0191618274674012	4.0889063741219

$$* \text{ error} = 100 \times \frac{|\text{discrete} - \text{continuous}|}{\text{continuous}}$$

Data:
t = 186M years (theirs)
R = 2.10381e-8/year (ours)

Inquerity's Approach

Assuming *our* pee rate

Does discrete probability make a difference?

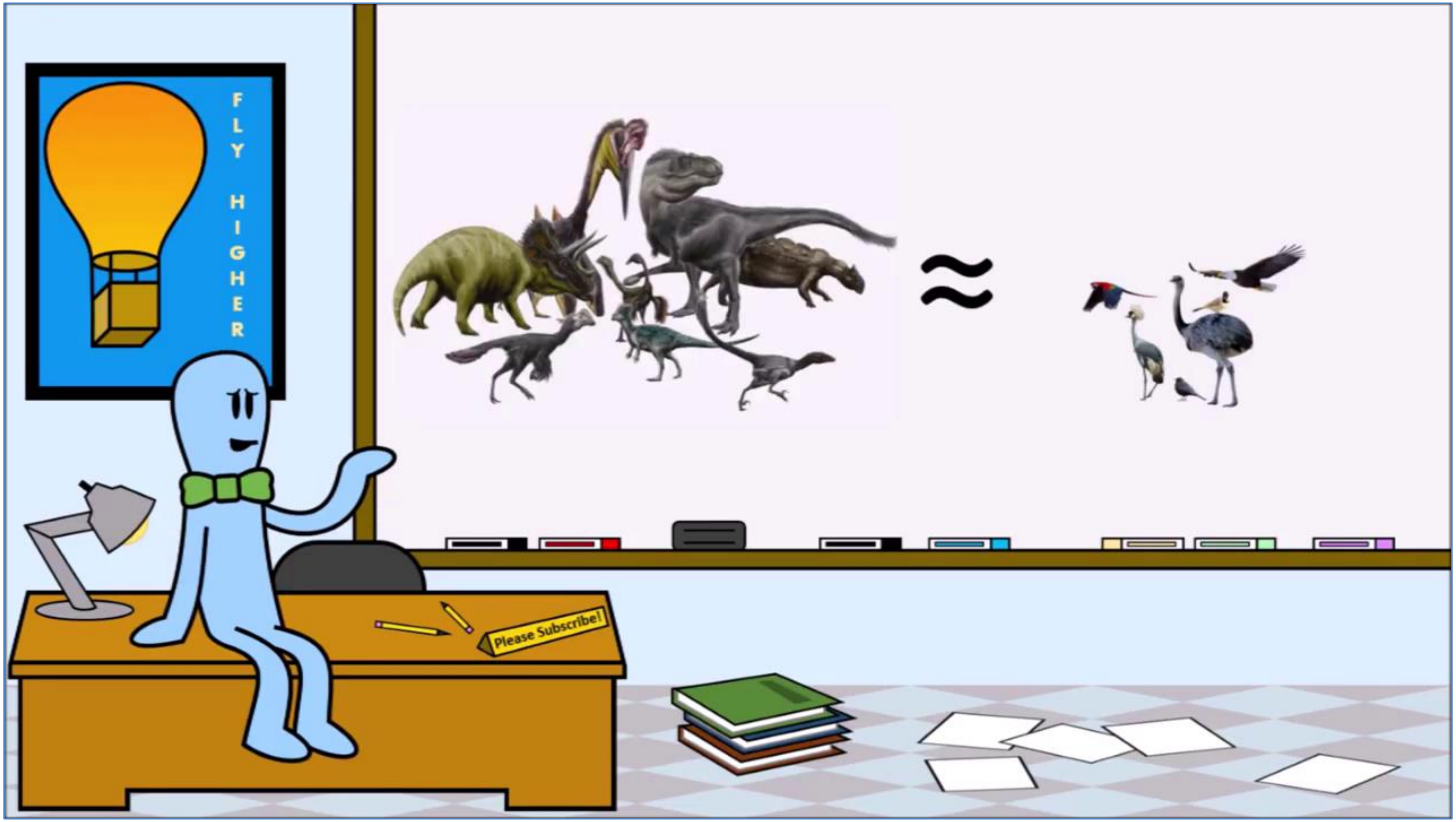
million years	continuous	discrete (annual)	difference*
0	1.00000000000000	1.00000000000000	0.00000000000000
25	0.59099217614322	0.59099217287355	0.00000000378656
50	0.34927175226250	0.34927174839780	0.00000000447565
75	0.20641687293497	0.20641686950896	0.00000000396761
100	0.12199075692852	0.12199075422885	0.00000000312644
125	0.07209558290654	0.07209558091219	0.00000000230962
150	0.04260792543225	0.04260792401787	0.00000000163796
165	0.03107702434069	0.03107702320593	0.00000000131415
175	0.02518095057216	0.02518094959696	0.00000000097520
186	0.01997873941689	0.01997873859452	0.00000000082237

*difference = abs(discrete – continuous)

Data:

R = 2.10381e-8/year (ours)

Inquerity's Approach



Inquerity used birds to estimate dinosaur pee

Reprinted with permission of Inquerity Studios: **Inquerity19** @ 12:55

Inquerity used "Alternative Facts"

- Inquerity's pee rate was based on a *bird model* and *domestic turkey* water consumption.

2. Pee rate:

    a. We estimated 2.650e13 liters/year

    b. They estimated 3.097e14 liters/year

1.

(~10x more)

Inquerity's Approach

Deriving their daily rate of decay, R_{daily}

$$R_{daily} = \frac{DailyPee}{TotalWater}$$

$$= \frac{8.48e11\ \text{kg/day}}{1.26e21\ \text{kg}}$$

$=$ **6.73015873e-10/day**

Multiply by 365.25 days/year

$R = R_{annual} =$ **2.45819048e-8/year**

Inquerity's Approach

Pure water remaining after 186M years

Using *infinite precision* with *Inquerity's* data

compounding cycle	k	$y(t)/y_0$	error* (%)
continuous	$k \to \infty$	$1.3900648155 \times 10^{-20}$	0
per day	365.25	$1.3900647941 \times 10^{-20}$	0.0000015385932
per year	1	$1.3900570037 \times 10^{-20}$	0.0005619698639
per 10 years	10^{-1}	$1.3899866999 \times 10^{-20}$	0.0056195648148
per 100 years	10^{-2}	$1.3892838455 \times 10^{-20}$	0.0561822679091
per 1,000 years	10^{-3}	$1.3822736851 \times 10^{-20}$	0.5604868419451
per 10,000 years	10^{-4}	$1.3139803625 \times 10^{-20}$	5.4734464241540
per 100,000 years	10^{-5}	$7.8505325107 \times 10^{-21}$	43.523982310550
per 1,000,000 years	10^{-6}	$1.6234881670 \times 10^{-23}$	99.883207736146

$$* \text{ error} = 100 \times \frac{|\text{discrete} - \text{continuous}|}{\text{continuous}}$$

Data:
t = 186M years (theirs)
R = 6.73016e-10/day (theirs)

Inquerity's Approach

Assuming *their* pee rate

Does discrete probability make a difference?

million years	continuous	discrete (annual)	difference*
0	1.00000000000000	1.00000000000000	0.0000000000000
25	0.00214315261380	0.00214315099499	$1.61880391399\times10^{-9}$
50	$4.593103126 \times 10^{-6}$	$4.593096187 \times 10^{-6}$	$6.9386850585\times10^{-12}$
75	$9.843720970 \times 10^{-9}$	$9.843698664 \times 10^{-9}$	$2.230598310\times10^{-14}$
100	$2.109659632 \times 10^{-11}$	$2.109653258 \times 10^{-11}$	$6.374014392\times10^{-17}$
125	$4.521322555 \times 10^{-14}$	$4.521305480 \times 10^{-14}$	$1.707560056\times10^{-19}$
150	$9.689884253 \times 10^{-17}$	$9.689840338 \times 10^{-17}$	$4.391472497\times10^{-22}$
165	$2.426333242 \times 10^{-18}$	$2.426321146 \times 10^{-18}$	$1.209580044\times10^{-23}$
175	$2.076690076 \times 10^{-19}$	$2.076679096 \times 10^{-19}$	$1.098019091\times10^{-24}$
186	$1.390064815 \times 10^{-20}$	$1.390057003 \times 10^{-20}$	$7.811745352\times10^{-26}$

*difference = abs(discrete – continuous)

Data:

R = 6.73016e-10/day (theirs)

Inquerity's Approach

Pure water remaining after 165M years

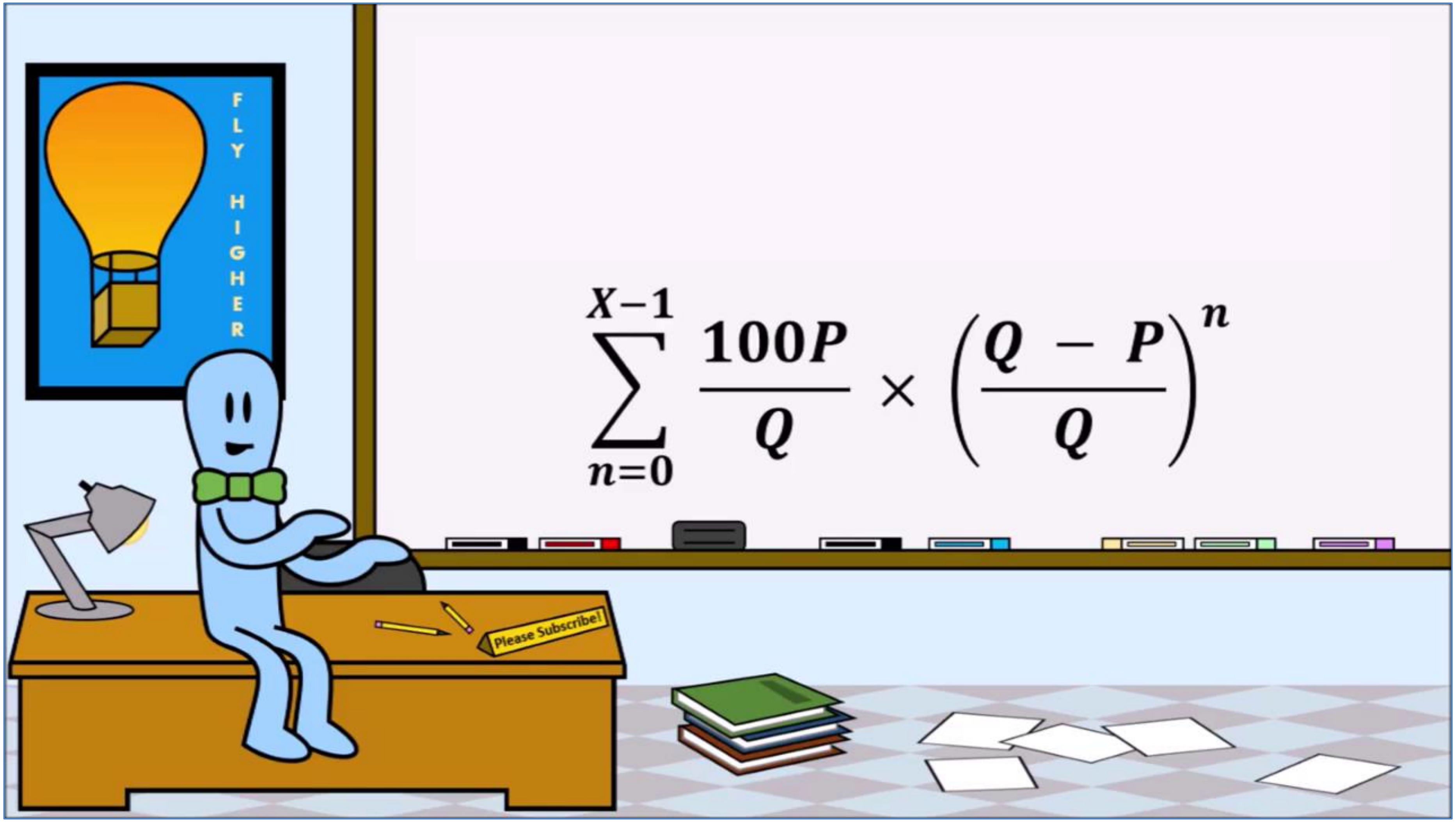
Using *infinite precision* with *Inquerity's* rate

compounding cycle	k	$y(t)/y_0$	error* (%)
continuous	$k \to \infty$	$2.4263348435\times10^{-18}$	0
per day	365.25	$2.4263348104\times10^{-18}$	0.0000013648815
per year	1	$2.4263227477\times10^{-18}$	0.0004985217954
per 10 years	10^{-1}	$2.4262138880\times10^{-18}$	0.0049851134721
per 100 years	10^{-2}	$2.4251255416\times10^{-18}$	0.0498406880240
per 1,000 years	10^{-3}	$2.4142671348\times10^{-18}$	0.4973636993121
per 10,000 years	10^{-4}	$2.3081530972\times10^{-18}$	4.8707929451980
per 100,000 years	10^{-5}	$1.4616054550\times10^{-18}$	39.7607688451025
per 1,000,000 years	10^{-6}	$6.0738395589\times10^{-21}$	99.7496701836063

$$* \ \mathrm{error} = 100 \times \frac{|\mathrm{discrete} - \mathrm{continuous}|}{\mathrm{continuous}}$$

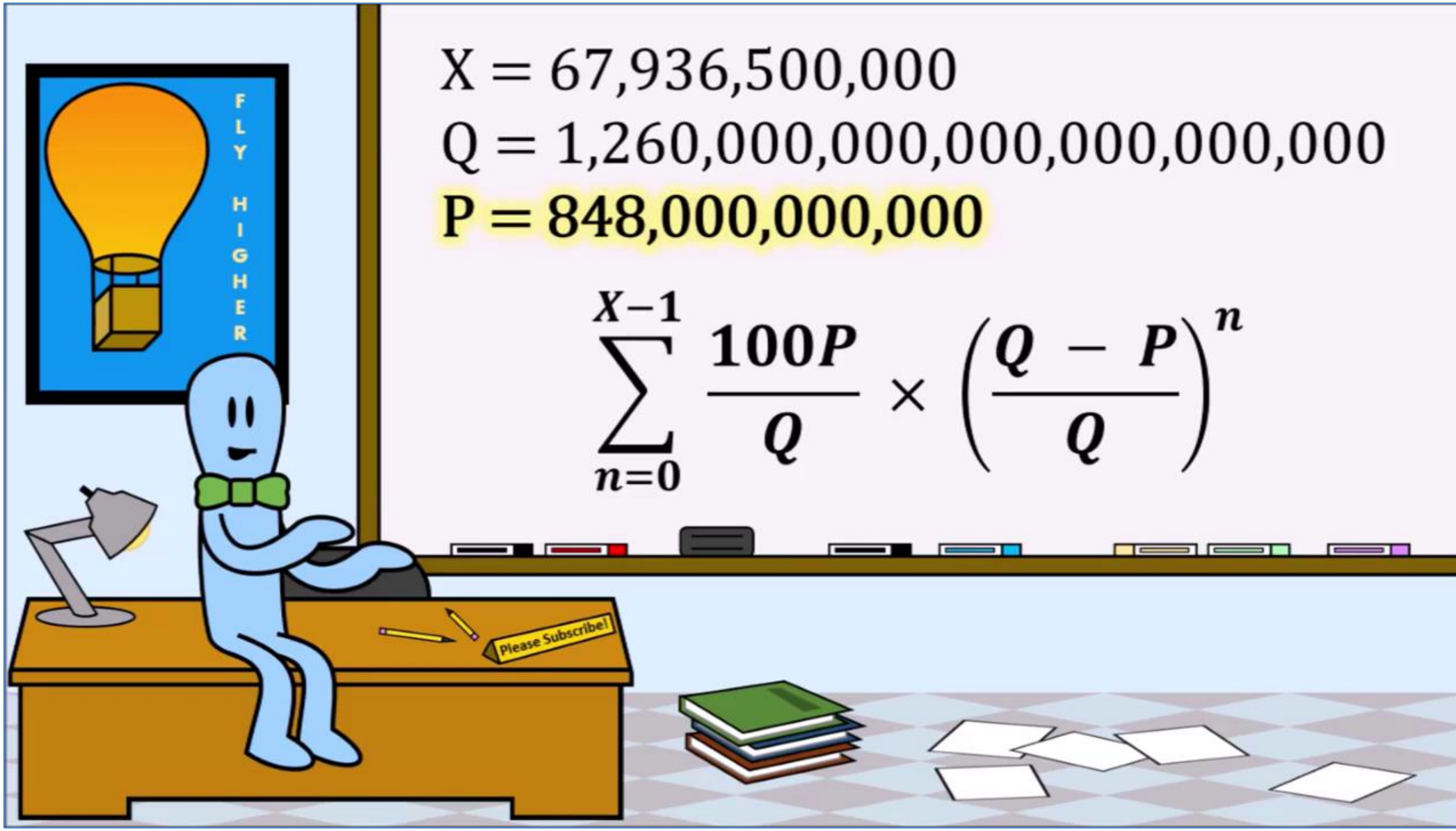
Data:
t = 165M years (ours)
R = 6.73016e-10/day (theirs)

Inquerity's Approach



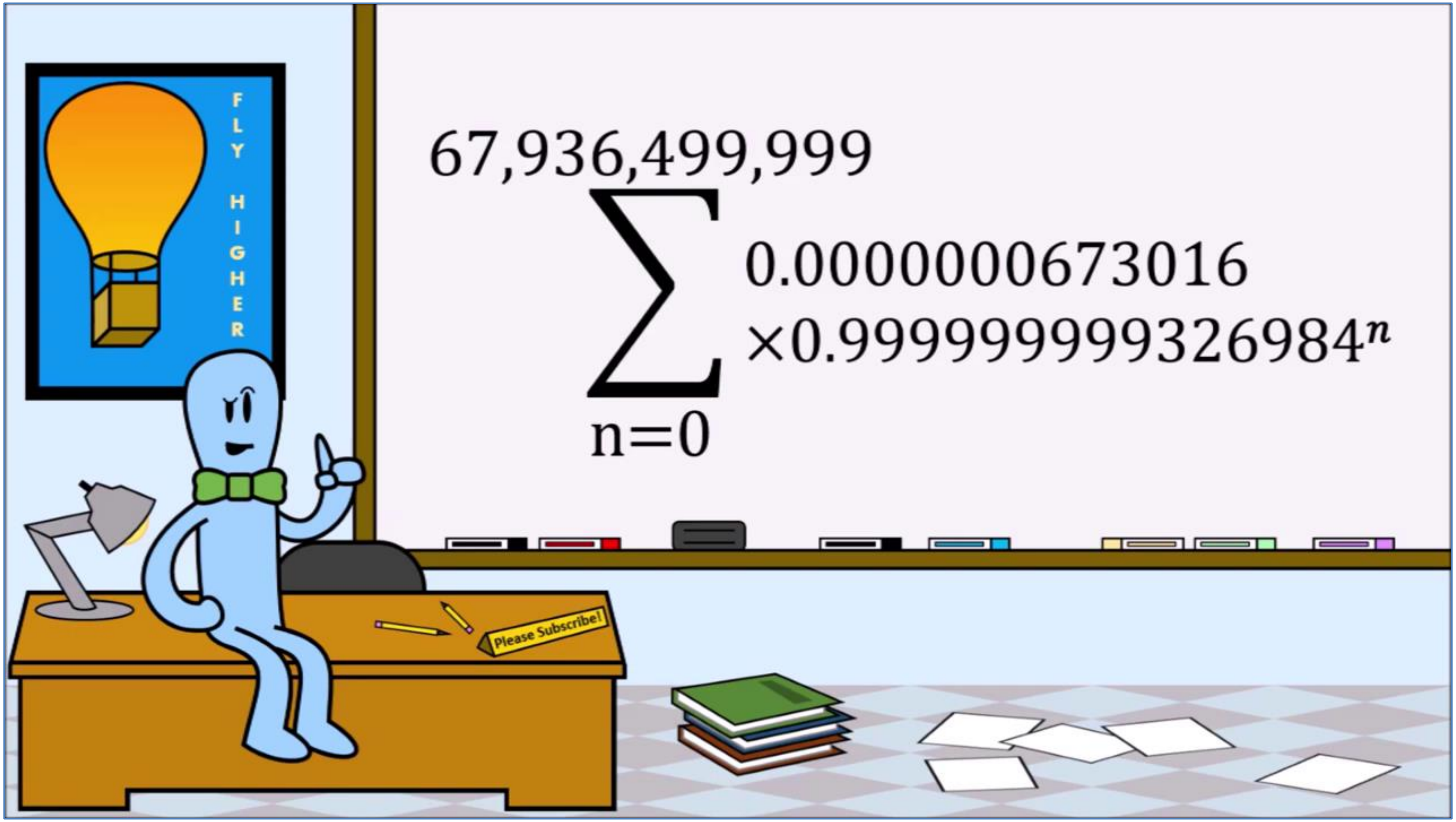
Reprinted with permission of Inquerity Studios: **Inquerity19** @ 12:09

Inquerity's Approach



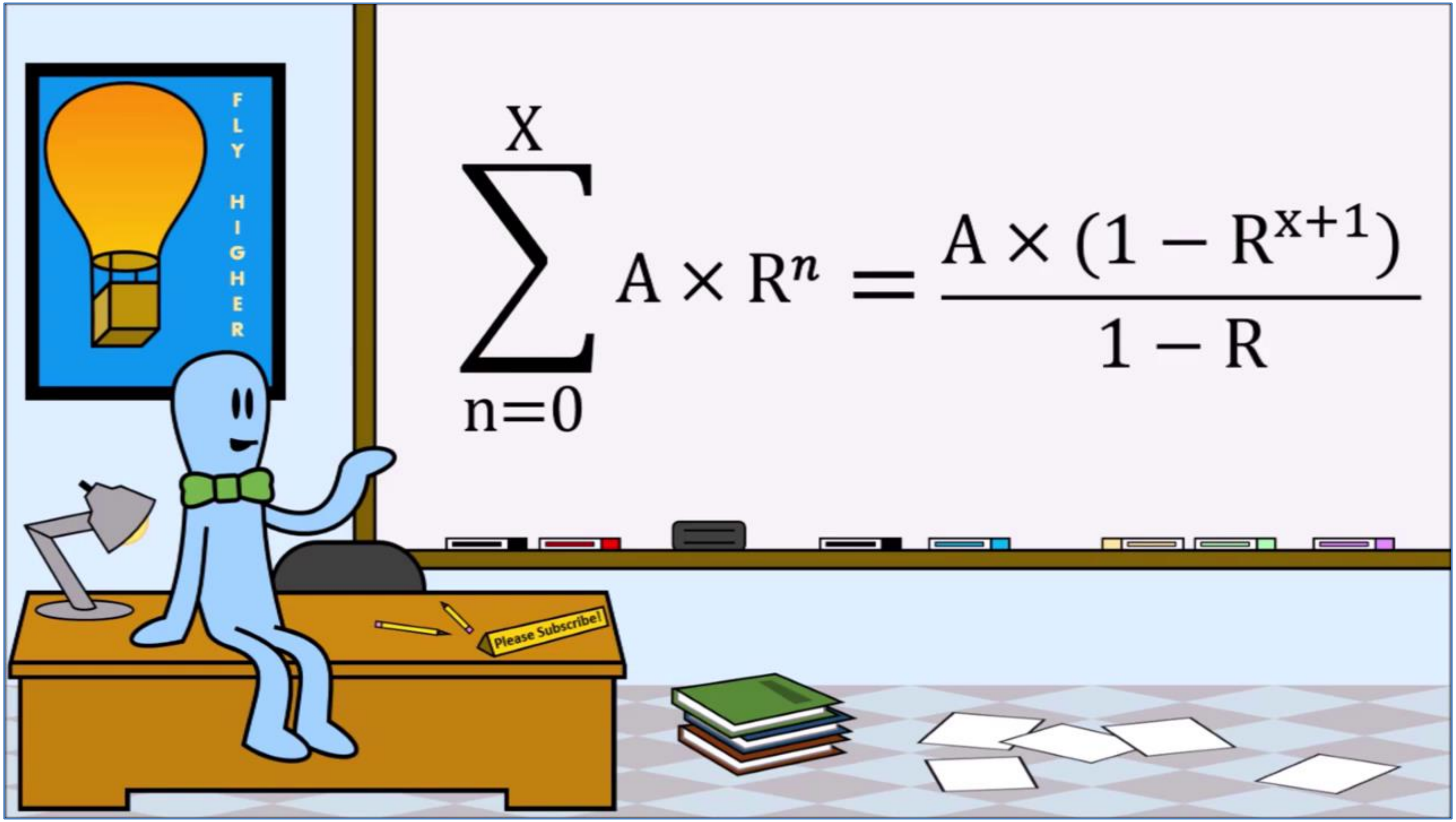
Reprinted with permission of Inquerity Studios: **Inquerity19** @ 12:09

Inquerity's Approach



Reprinted with permission of Inquerity Studios: **Inquerity19** @ 12:34

Inquerity's Approach



Inquerity could have canceled out $(1 - R)$

Reprinted with permission of Inquerity Studios: **Inquerity19** @ 12:54

Inquerity's Approach

So, what did they type into Google?

R = 1 – P/Q = 0.999999999326984

A = 100 * P/Q = 0.0000000673016

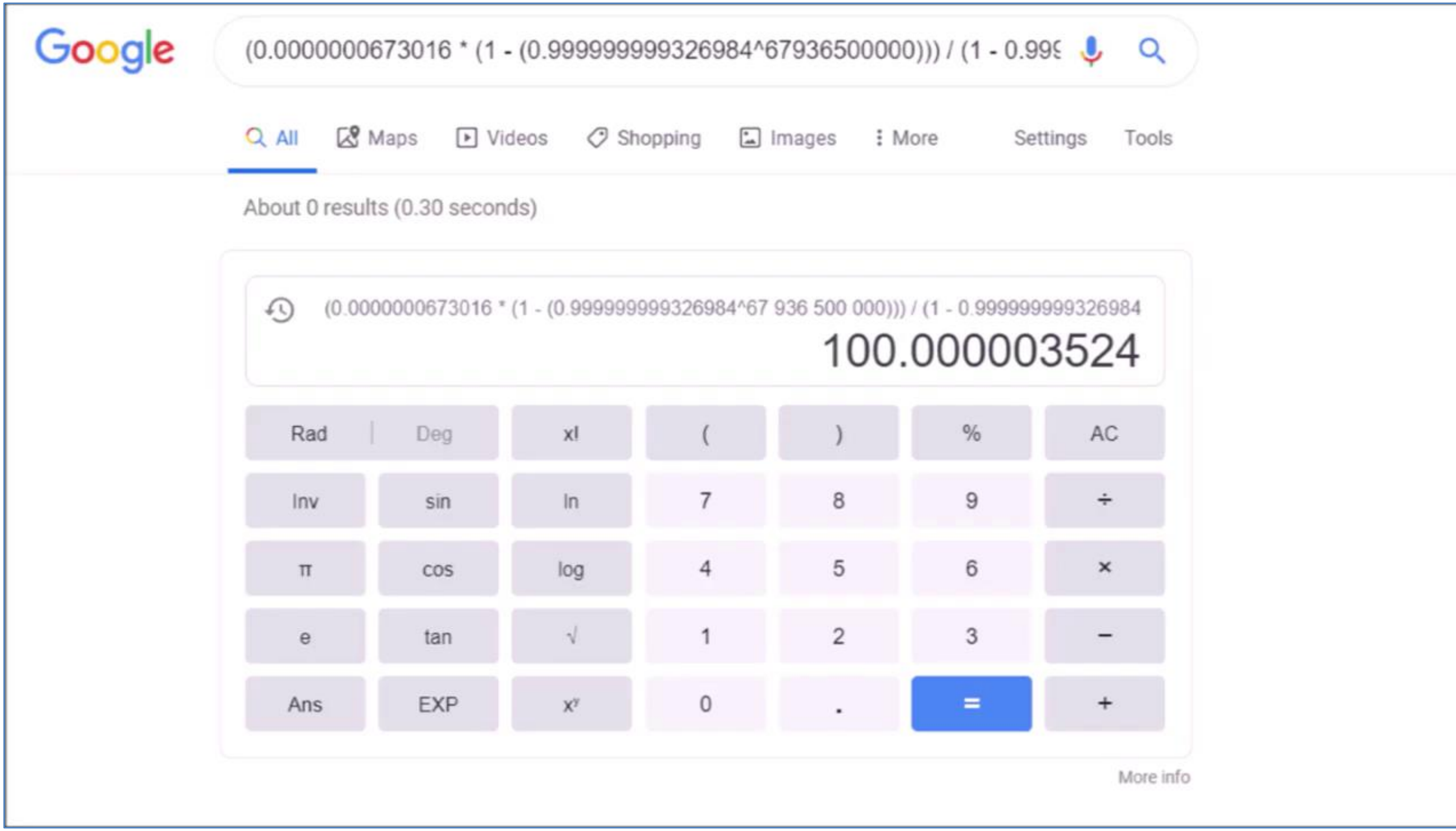
X+1 = 67936500000

$$\sum_{n=0}^{X} AR^n = \frac{A(1-R^{X+1})}{1-R}$$

$$\frac{0.0000000673016 \times (1-0.999999999326984^{67936500000})}{1-0.999999999326984}$$

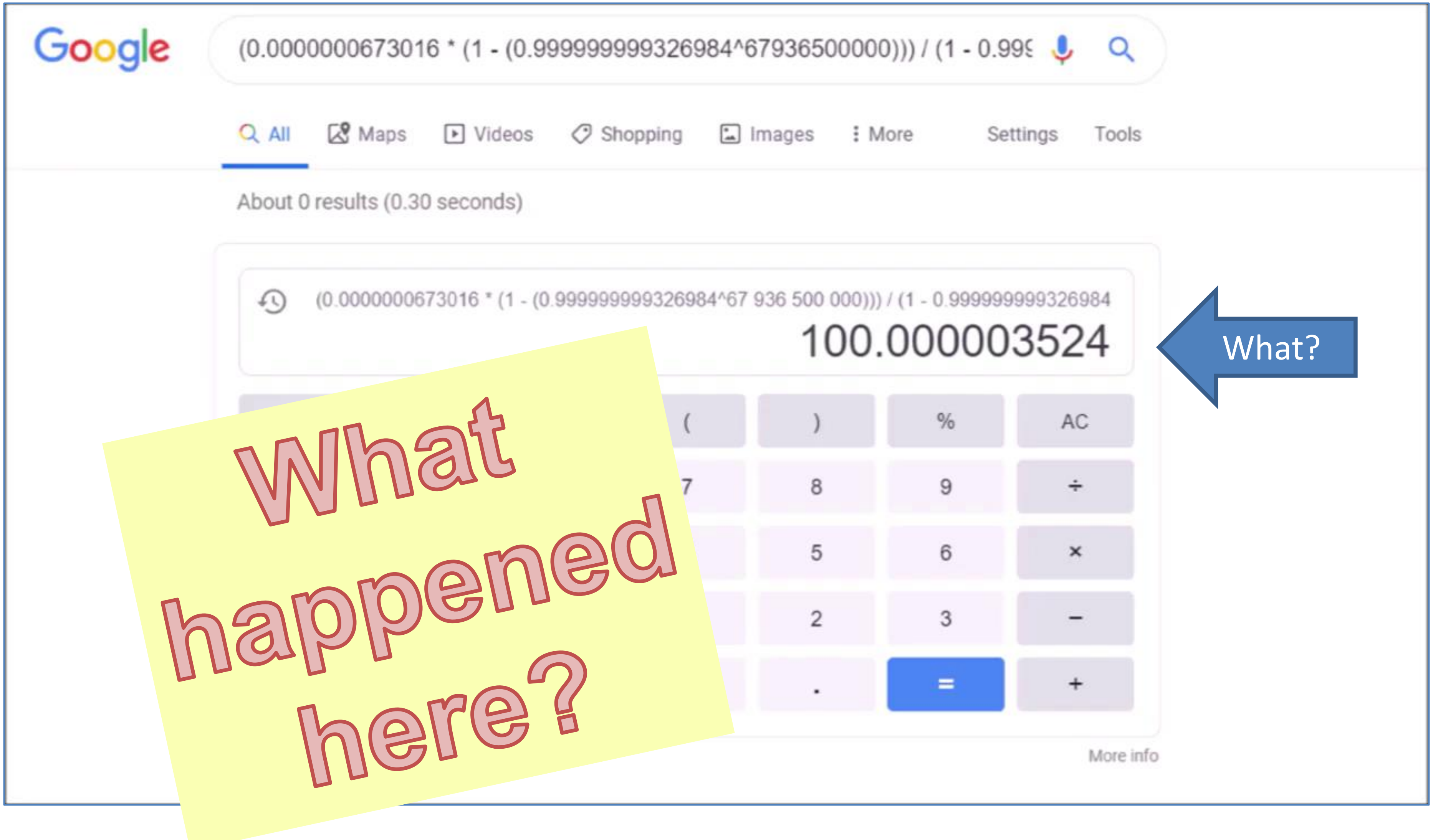
(0.0000000673016*(1-(0.999999999326984^67936500000)))/(1-0.999999999326984)

Inquerity's Approach



Reprinted with permission of Inquerity Studios: **Inquerity19** @ 13:01

Inquerity's Approach



Reprinted with permission of Inquerity Studios: **Inquerity19** @ 13:01

What went wrong?

$$\sum_{n=0}^{X} AR^{n} = \frac{A(1-R^{X+1})}{1-R}$$

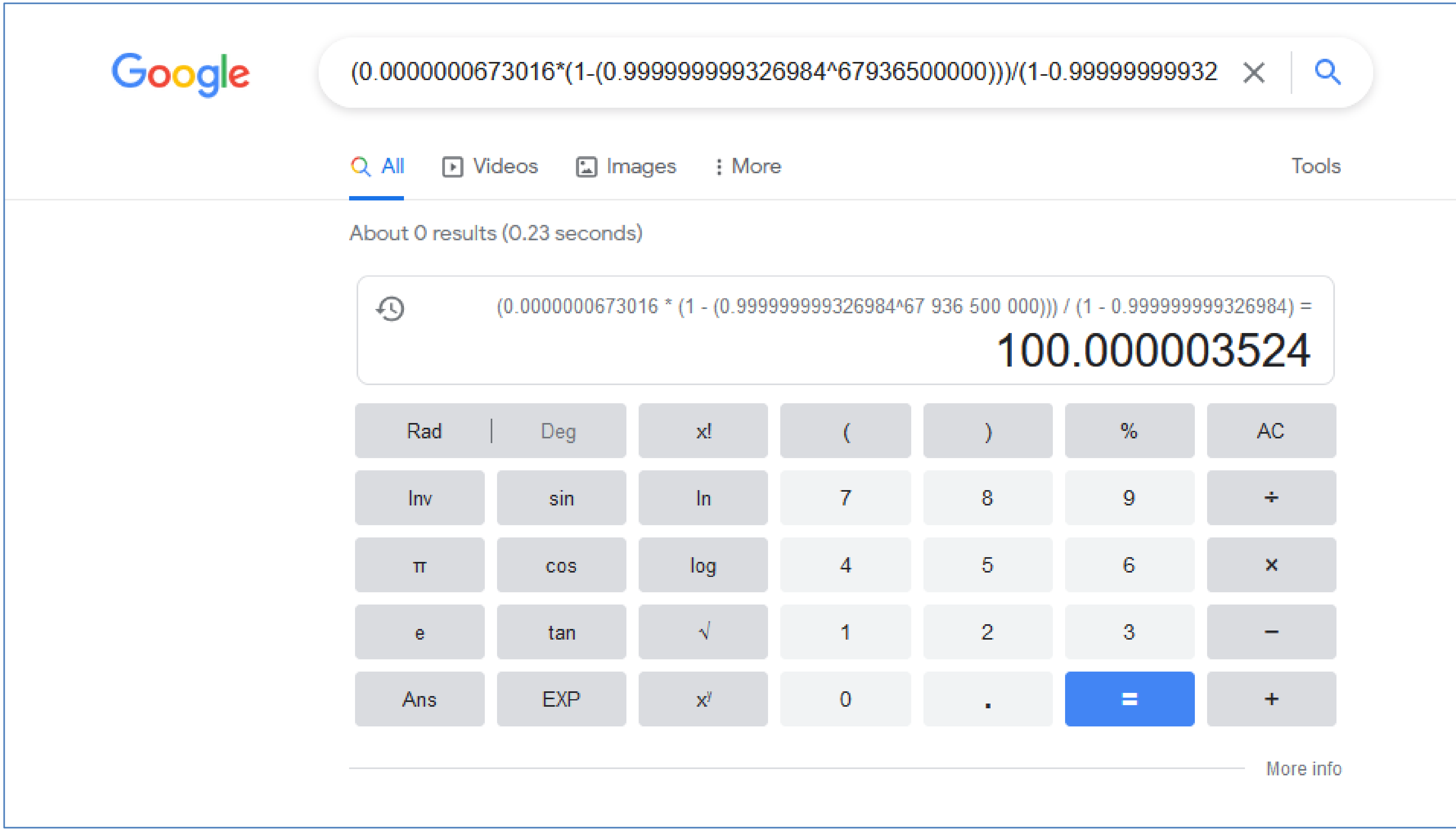
R = 1 – P/Q = 0.999999999326984

A = 100 * P/Q = 0.0000000673016

X+1 = 67936500000

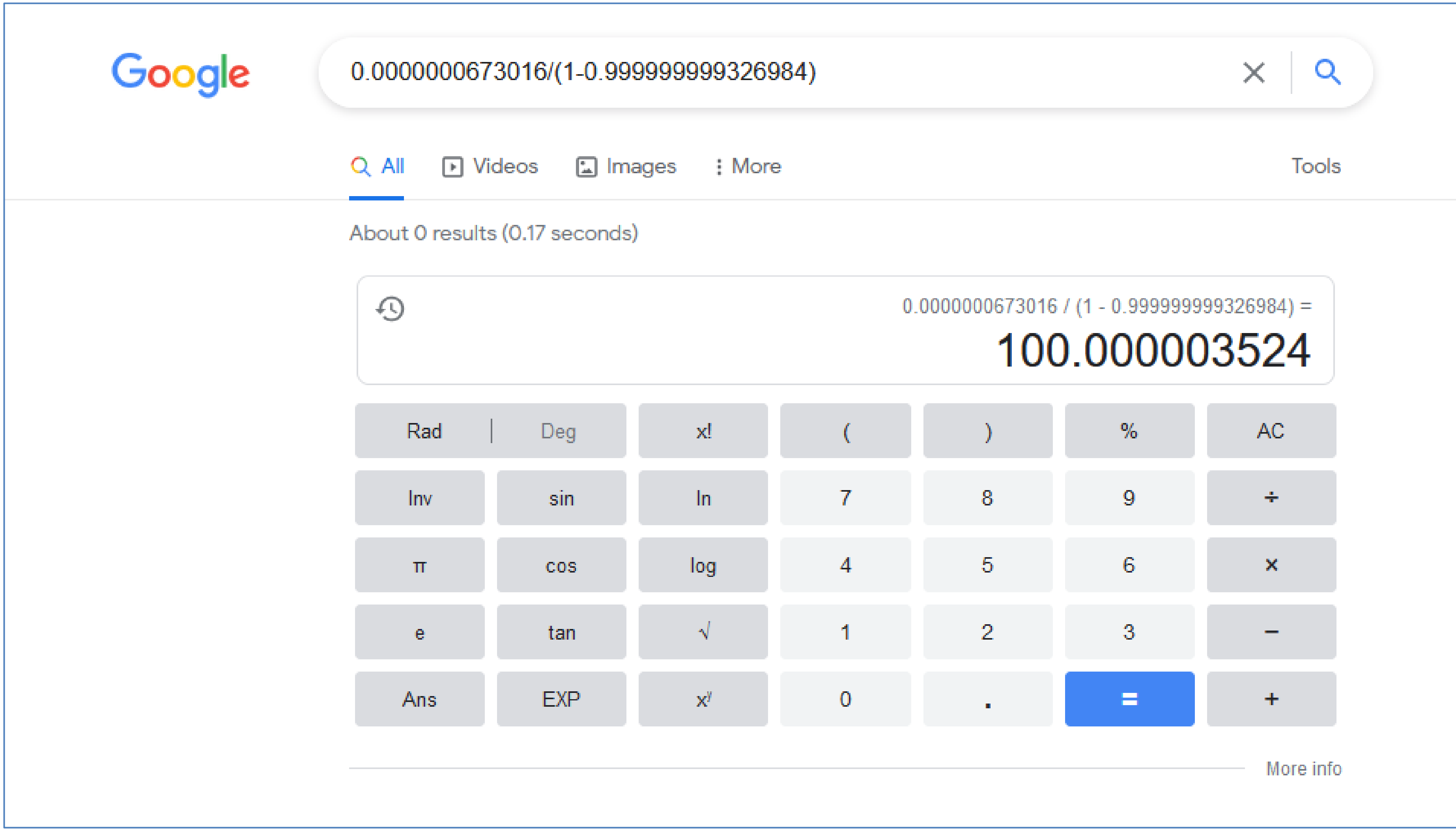
- ❑ General rounding error?
- ❑ Problem with $\frac{A}{1-R}$?
- ❑ Problem with $\frac{A/100}{1-R}$?
- ❑ Problem with $1-R^{X+1}$?
- ❑ Problem with R^{X+1}?

Inquerity's Approach



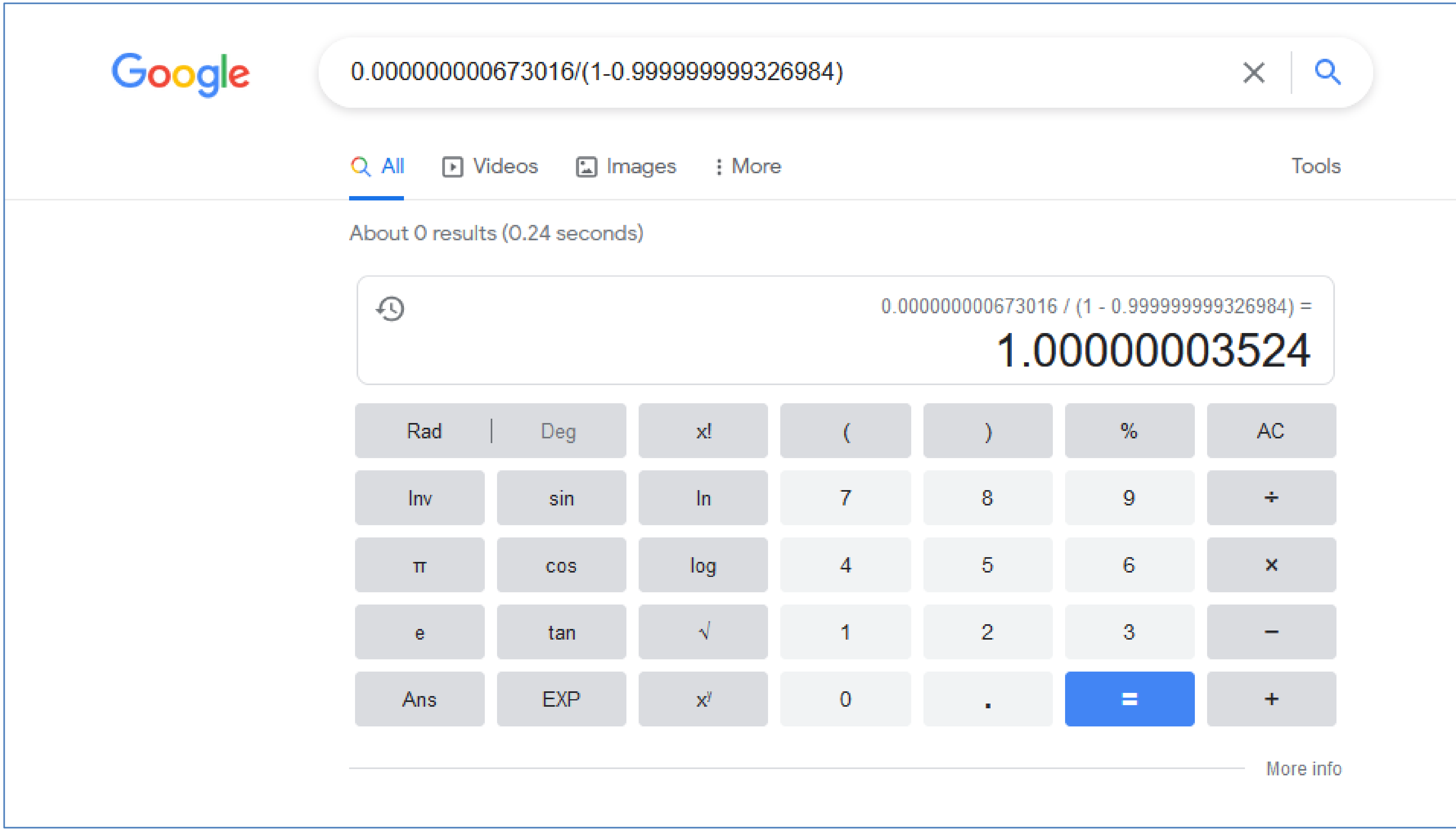
$$\frac{A(1-R^{X+1})}{1-R} = \frac{0.0000000673016 \times (1 - 0.999999999326984^{67936500000})}{1 - 0.999999999326984}$$

Inquerity's Approach



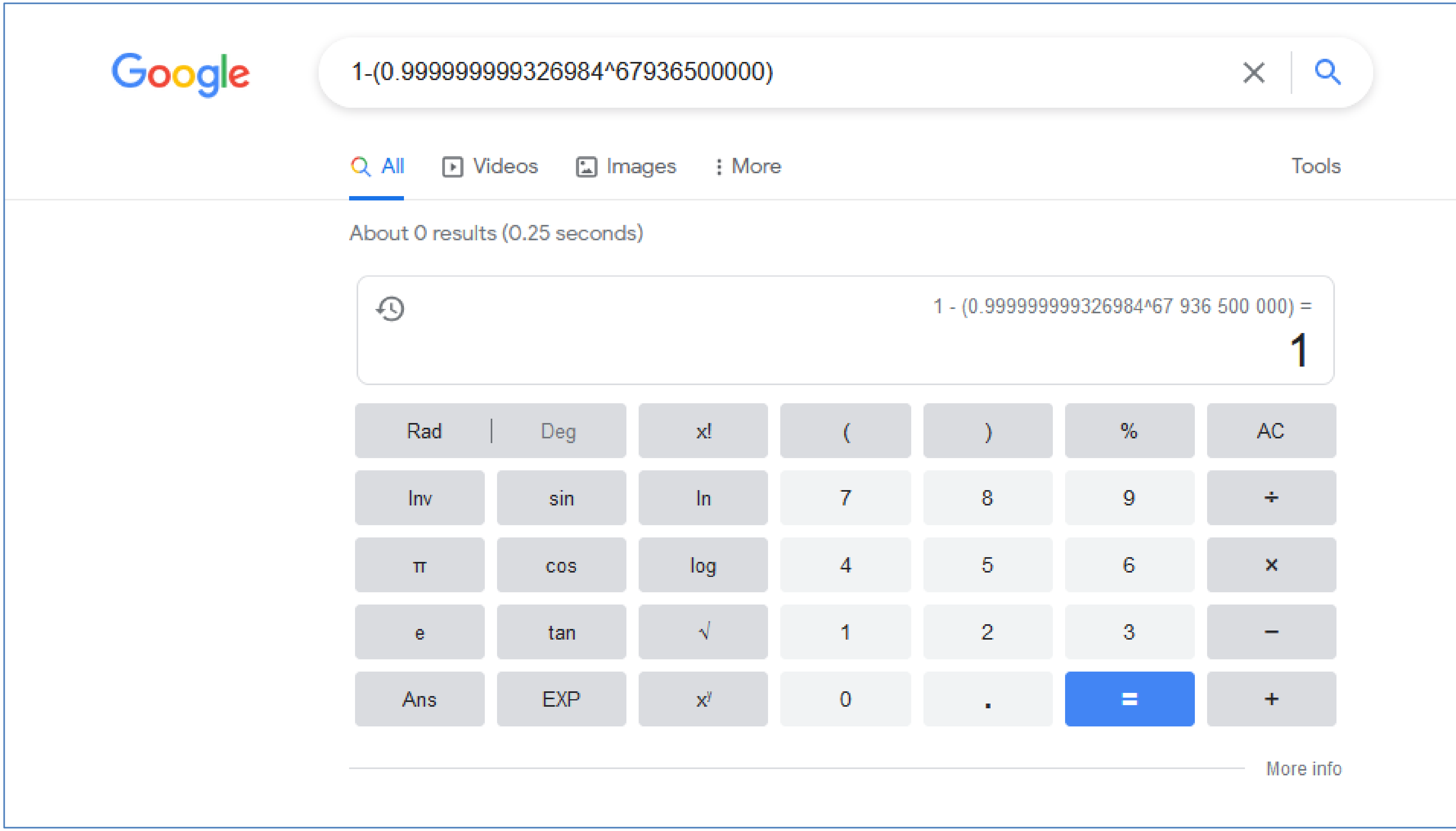
$$\frac{A}{1-R} = \frac{0.0000000673016}{1-0.999999999326984}$$

Inquerity’s Approach



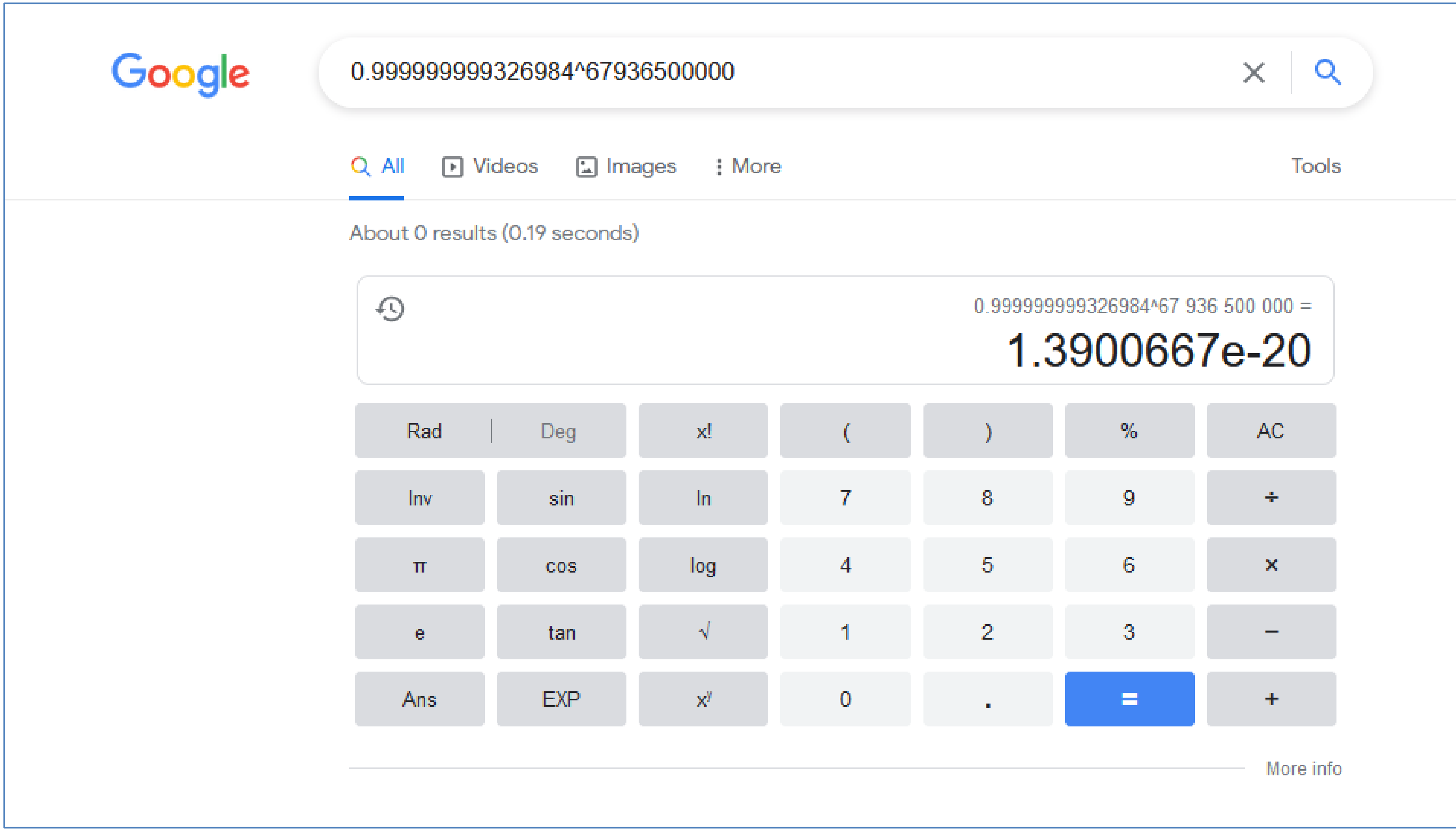
$$\frac{A/100}{1-R} = \frac{0.000000000673016}{1-0.999999999326984}$$

Inquerity’s Approach



$$\frac{R^{X+1}}{} = \frac{1 - 0.999999999326984^{67936500000}}{}$$

Inquerity's Approach



$$\underline{R^{X+1}} = \underline{0.999999999326984^{67936500000}}$$

Inquerity's Approach

99.99999999999999999860993520 59…

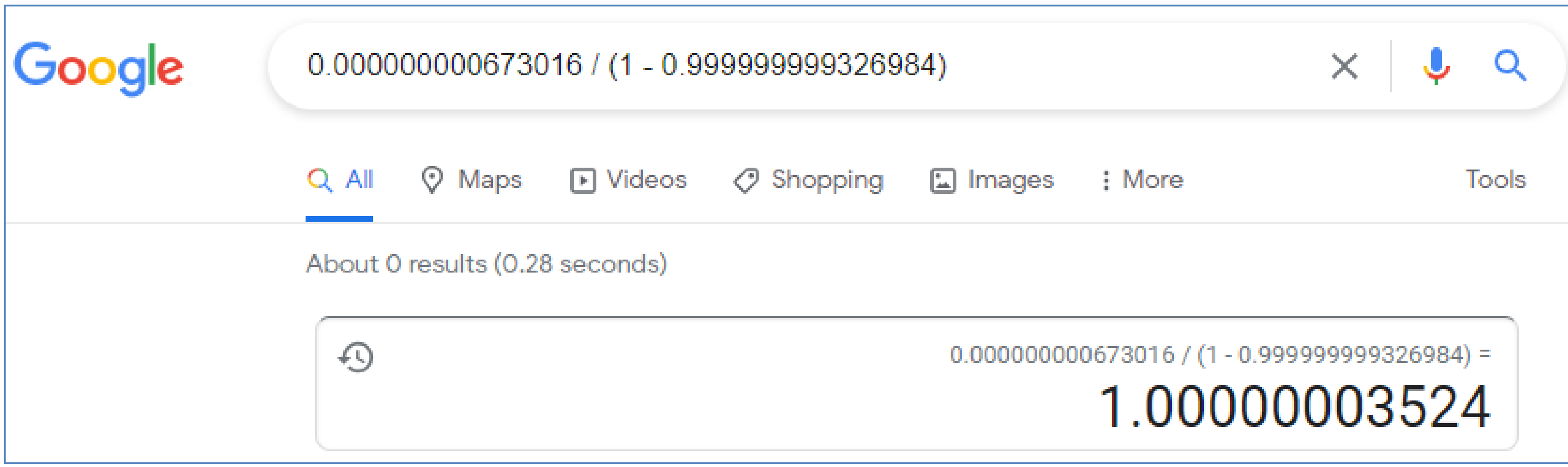
$\underbrace{99.99999999999999999}_{19\ 9\text{s}}$

19 9s

Inquerity's Approach

What went wrong?

Google Calculator employs a `double`.



double x = $\overbrace{0.000000000673016}^{A}$; // 6.7301599999999984E-10

double y = 1 – x; // 9.99999999326984024E-1

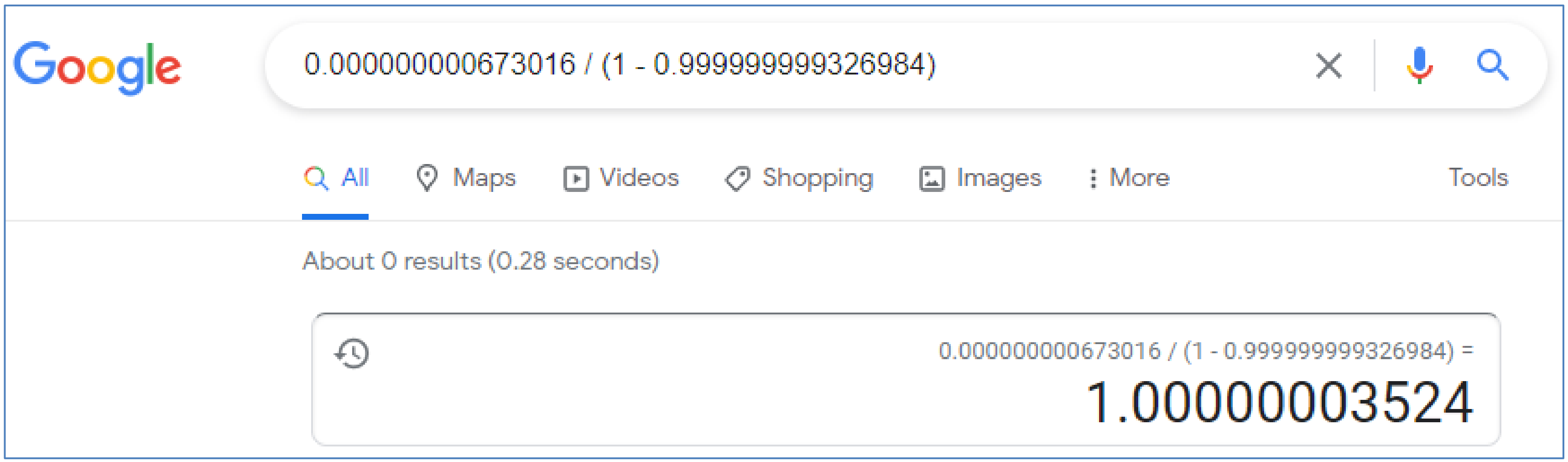
double z = 1 – y; // 6.73015976282442807E-10

double w = x / z; // 100.00000352407045 > 1

Inquerity’s Approach

What went wrong?

Google Calculator employs a `double`.



$$\frac{A}{1-R} = \frac{0.000000000673016}{1 - 0.999999999326984}$$

`float:` nan

`double:` 100.0000352407045 – same as Google

`__float80:` 99.9999999606698857

`__float128:` 99.9999999999999999986099329661338

Quantifying Dinosaur Pee

Is $v \approx 1 - (1 - v)$?

$$v = 10^{-1}$$

Equation	Decimal	Binary
$v = 10^{-1}$	0.1000000015	001111011**10011001100110011001101**
$1 - v$	0.8999999762	001111110**11001100110011001100110**
$1 - (1 - v)$	0.1000000238	001111011**10011001100110011010000**
$1 - (1 - (1 - v))$	0.8999999762	001111110**11001100110011001100110**

Quantifying Dinosaur Pee

Is $v \approx 1 - (1 - v)$?

$$v = 10^{-2}$$

Equation	Decimal	Binary
$v = 10^{-2}$	0.0099999998	001111000**01000111101011100001010**
$1 - v$	0.9900000095	001111110**11110101110000101000100**
$1 - (1 - v)$	0.0099999905	001111000**0100011110101110000**0000
$1 - (1 - (1 - v))$	0.9900000095	001111110**11110101110000101000100**

Quantifying Dinosaur Pee

Is $v \approx 1 - (1 - v)$?

$$v = 10^{-3}$$

Equation	Decimal	Binary
$v = 10^{-3}$	0.0010000000	001110101**00000110001001001101111**
$1 - v$	0.9990000129	001111110**11111110111110011101111**
$1 - (1 - v)$	0.0009999871	001110101**000001100010010000000000**
$1 - (1 - (1 - v))$	0.9990000129	001111110**11111110111110011101111**

Quantifying Dinosaur Pee

Is $v \approx 1 - (1 - v)$?

$$v = 10^{-4}$$

Equation	Decimal	Binary
$v = 10^{-4}$	0.0001000000	001110001**10100011011011100010111**
$1 - v$	0.9998999834	001111110**11111111111100101110010**
$1 - (1 - v)$	0.0001000166	001110001**10100011100000000000000**
$1 - (1 - (1 - v))$	0.9998999834	001111110**11111111111100101110010**

Quantifying Dinosaur Pee

Is $v \approx 1 - (1 - v)$?

$$v = 10^{-5}$$

Equation	Decimal	Binary
$v = 10^{-5}$	0.0000100000	001101110**01001111100010110101100**
$1 - v$	0.9999899864	001111110**11111111111111101011000**
$1 - (1 - v)$	0.0000100136	001101110**010**10000000000000000000
$1 - (1 - (1 - v))$	0.9999899864	001111110**11111111111111101011000**

Quantifying Dinosaur Pee

Is $v \approx 1 - (1 - v)$?

$$v = 10^{-6}$$

Equation	Decimal	Binary
$v = 10^{-6}$	0.0000010000	001101011**00001100011011110111101**
$1 - v$	0.9999989867	001111110**11111111111111111101111**
$1 - (1 - v)$	0.0000010133	001101011**00010000000000000000000**
$1 - (1 - (1 - v))$	0.9999989867	001111110**11111111111111111101111**

Quantifying Dinosaur Pee

Is $v \approx 1 - (1 - v)$?

$$v = 10^{-7}$$

Equation	Decimal	Binary
$v = 10^{-7}$	0.0000001000	001100111**10101101011111110010101**
$1 - v$	0.9999998808	001111110**11111111111111111111110**
$1 - (1 - v)$	0.0000001192	00110100000000000000000000000000
$1 - (1 - (1 - v))$	0.9999998808	001111110**11111111111111111111110**

Quantifying Dinosaur Pee

Is $v \approx 1 - (1 - v)$?

$$v = 10^{-8}$$

Equation	Decimal	Binary
$v = 10^{-8}$	0.0000000100	001100100**01010111100110001110111**
$1 - v$	1.0000000000	001111111**00000000000000000000000**
$1 - (1 - v)$	0.0000000000	000000000**00000000000000000000000**
$1 - (1 - (1 - v))$	1.0000000000	001111111**00000000000000000000000**

Quantifying Dinosaur Pee

Floating-Point Subtraction Algorithm

==

Subtracting floating point numbers

==

***** Express the two numbers in IEEE 754 format (float):**

***** Increase the smaller exponent so it matches the larger, shifting the mantissa to the right.**

The FPU keeps track of excess bits.

***** Now do the subtraction:**

***** Now we need to normalize (make the "hidden bit" equal to 1):**

***** Round rule:**

```
if LSB+1=0, do nothing.
if LSB+1=1 {
        round;
        if LSB+2=1 and no bits afterwards, apply nearest (even) rule (select even one from 2 nearest ones;)
        otherwise, just do rounding (select closest one).
}
```

Note that if there is overflow during round-up (i.e., if all mantissa bits are 1, and the LSB+ bits lead to a round up, an extra left shift is performed and the mantissa bits all become 0). See Intel Architecture Software Developer's Manual - Floating-Point Unit – Page 17.

***** Normalize again if necessary**

(due to rounding carry)

==

Quantifying Dinosaur Pee

Example: $1 - 10^{-1}$

```
==============================================================
Subtracting 0.1 from 1.0
==============================================================
*** Express the two numbers in IEEE 754 format (float):
1.0000000000e+00: : [ 0 01111111 (1) 00000000000000000000000 ]
1.0000000149e-01: : [ 0 01111011 (1) 10011001100110011001101 ]
*** Increase the smaller exponent so it matches the larger,
shifting the mantissa to the right. In this case, we do it four times:
1.0000000149e-01: : [ 0 01111011 (1) 10011001100110011001101 ]
                    [ 0 01111100 (0) 110011001100110011001101 ]
                    [ 0 01111101 (0) 0110011001100110011001101 ]
                    [ 0 01111110 (0) 00110011001100110011001101 ]
                    [ 0 01111111 (0) 000110011001100110011001101 ]
*** Now do the subtraction:
1.0000000000e+00: : [ 0 01111111 (1) 000000000000000000000000000 ]
-
                    [ 0 01111111 (0) 000110011001100110011001101 ]
=
                    [ 0 01111111 (0) 111001100110011001100110011 ]
*** Now we need to normalize (make the "hidden bit" equal to 1):
                    [ 0 01111111 (0) 111001100110011001100110011 ]
                    [ 0 01111110 (1) 11001100110011001100110011 ]
*** Round
                    [ 0 01111110 (1) 11001100110011001100110011 ]
8.9999997616e-01: : [ 0 01111110 (1) 11001100110011001100110 ]
*** Now need to re-normalize (if required):
8.9999997616e-01: : [ 0 01111110 (1) 11001100110011001100110 ]
*** Compare with FPU result:
8.9999997616e-01: : [ 0 01111110     11001100110011001100110 ]
==============================================================
```

Quantifying Dinosaur Pee

Example: $1 - 10^{-2}$

```
================================================================
Subtracting 0.01 from 1.0
================================================================
```

***** Express the two numbers in IEEE 754 format (float):**

```
1.0000000000e+00: : [ 0 01111111 (1) 00000000000000000000000 ]
9.9999997765e-03: : [ 0 01111000 (1) 01000111101011100001010 ]
```

***** Increase the smaller exponent so it matches the larger,**
shifting the mantissa to the right. In this case, we do it seven times:

```
9.9999997765e-03: : [ 0 01111000 (1) 01000111101011100001010 ]
                    [ 0 01111001 (0) 10100011110101110000101 0 ]
                    [ 0 01111010 (0) 01010001111010111000010 10 ]
                    [ 0 01111011 (0) 00101000111101011100001 010 ]
                    [ 0 01111100 (0) 00010100011110101110000 1010 ]
                    [ 0 01111101 (0) 00001010001111010111000 01010 ]
                    [ 0 01111110 (0) 00000101000111101011100 001010 ]
                    [ 0 01111111 (0) 00000010100011110101110 0001010 ]
```

***** Now do the subtraction:**

```
1.0000000000e+00: : [ 0 01111111 (1) 00000000000000000000000 0000000 ]
-
                    [ 0 01111111 (0) 00000010100011110101110 0001010 ]
=
                    [ 0 01111111 (0) 11111101011100001010001 1110110 ]
```

***** Now we need to normalize (left-shift and decrease exponent):**

```
                    [ 0 01111111 (0) 11111101011100001010001 1110110 ]
                    [ 0 01111110 (1) 11111010111000010100011 110110 ]
```

***** Round: nearest (even)**

```
                    [ 0 01111110 (1) 11111010111000010100011 110110 ]
9.9000000954e-01: : [ 0 01111110 (1) 11111010111000010100100 ]
```

***** Now need to re-normalize (if required):**

```
9.9000000954e-01: : [ 0 01111110 (1) 11111010111000010100100 ]
```

***** Compare with FPU result:**

```
9.9000000954e-01: : [ 0 01111110     11111010111000010100100 ]
================================================================
```

Quantifying Dinosaur Pee

Example: $1 - 10^{-3}$

```
==============================================================
Subtracting 0.001 from 1.0
==============================================================
```

***** Express the two numbers in IEEE 754 format (float):**

```
1.0000000000e+00: : [ 0 01111111 (1) 00000000000000000000000 ]
1.0000000475e-03: : [ 0 01110101 (1) 00000110001001001101111 ]
```

***** Increase the smaller exponent so it matches the larger,**
shifting the mantissa to the right. In this case, we do it ten times:

```
1.0000000475e-03: : [ 0 01110101 (1) 00000110001001001101111 ]
                    [ 0 01110110 (0) 10000011000100100110111 1 ]
                    [ 0 01110111 (0) 01000001100010010011011 11 ]
                    [ 0 01111000 (0) 00100000110001001001101 111 ]
                    [ 0 01111001 (0) 00010000011000100100110 1111 ]
                    [ 0 01111010 (0) 00001000001100010010011 01111 ]
                    [ 0 01111011 (0) 00000100000110001001001 101111 ]
                    [ 0 01111100 (0) 00000010000011000100100 1101111 ]
                    [ 0 01111101 (0) 00000001000001100010010 01101111 ]
                    [ 0 01111110 (0) 00000000100000110001001 001101111 ]
                    [ 0 01111111 (0) 00000000010000011000100 1001101111 ]
```

***** Now do the subtraction:**

```
1.0000000000e+00: : [ 0 01111111 (1) 00000000000000000000000 0000000000 ]
-
                    [ 0 01111111 (0) 00000000010000011000100 1001101111 ]
=
                    [ 0 01111111 (0) 11111111101111100111011 0110010001 ]
```

***** Now we need to normalize:**

```
                    [ 0 01111111 (0) 11111111101111100111011 0110010001 ]
                    [ 0 01111110 (1) 11111111011111001110110 110010001 ]
```

***** Round:**

```
                    [ 0 01111110 (1) 11111111011111001110110 110010001 ]
9.9900001287e-01: : [ 0 01111110 (1) 11111111011111001110111 ]
```

***** Now need to re-normalize (if required):**

```
9.9900001287e-01: : [ 0 01111110 (1) 11111111011111001110111 ]
```

***** Compare with FPU result:**

```
9.9900001287e-01: : [ 0 01111110     11111111011111001110111 ]
==============================================================
```

Quantifying Dinosaur Pee

Example: $1 - 10^{-7}$

===

Subtracting 0.0000001 from 1.0

===

***** Express the two numbers in IEEE 754 format (float):**

1.0000000000e+00: : [0 01111111 (1) 00000000000000000000000]
1.0000000117e-07: : [0 01100111 (1) 10101101011111110010101]

***** Increase the smaller exponent so it matches the larger,**
shifting the mantissa to the right. In this case, we do it 24 times:

1.0000000117e-07: : [0 01100111 (1) 10101101011111110010101]
[0 01101000 (0) 11010110101111111001010**1**]
[0 01101001 (0) 01101011010111111100101**01**]
[0 01101010 (0) 00110101101011111110010**101**]
[0 01101011 (0) 00011010110101111111001**0101**]
[0 01101100 (0) 00001101011010111111100**10101**]
[0 01101101 (0) 00000110101101011111110**010101**]
[0 01101110 (0) 00000011010110101111111**0010101**]
[0 01101111 (0) 00000001101011010111111**10010101**]
[0 01110000 (0) 00000000110101101011111**110010101**]
[0 01110001 (0) 00000000011010110101111**1110010101**]
[0 01110010 (0) 00000000001101011010111**11110010101**]
[0 01110011 (0) 00000000000110101101011**111110010101**]
[0 01110100 (0) 00000000000011010110101**1111110010101**]
[0 01110101 (0) 00000000000001101011010**11111110010101**]
[0 01110110 (0) 00000000000000110101101**011111110010101**]
[0 01110111 (0) 00000000000000011010110**1011111110010101**]
[0 01111000 (0) 00000000000000001101011**01011111110010101**]
[0 01111001 (0) 00000000000000000110101**101011111110010101**]
[0 01111010 (0) 00000000000000000011010**1101011111110010101**]
[0 01111011 (0) 00000000000000000001101**01101011111110010101**]
[0 01111100 (0) 00000000000000000000110**101101011111110010101**]
[0 01111101 (0) 00000000000000000000011**0101101011111110010101**]
[0 01111110 (0) 00000000000000000000001**10101101011111110010101**]
[0 01111111 (0) 00000000000000000000000**110101101011111110010101**]

Quantifying Dinosaur Pee

Example: $1 - 10^{-7}$

```
=============================================================
Subtracting 0.0000001 from 1.0 continued
=============================================================
*** Now do the subtraction:
1.0000000000e+00: : [ 0 01111111 (1) 00000000000000000000000000000000000000000000000 ]
-
                    [ 0 01111111 (0) 00000000000000000000000110101101011111110010101 ]
=
                    [ 0 01111111 (0) 11111111111111111111111001010010100000001101011 ]
*** Now we need to normalize (left-shift and decrease exponent):
                    [ 0 01111111 (0) 11111111111111111111111001010010100000001101011 ]
                    [ 0 01111110 (1) 1111111111111111111111001010010100000001101011 ]
*** Round: no rounding according to the rule
                    [ 0 01111110 (1) 1111111111111111111111001010010100000001101011 ]
9.9999988079e-01: : [ 0 01111110 (1) 11111111111111111111110 ]
*** Now need to re-normalize (if required):
9.9999988079e-01: : [ 0 01111110 (1) 11111111111111111111110 ]
*** Compare with FPU result:
9.9999988079e-01: : [ 0 01111110     11111111111111111111110 ]

=============================================================
```

Quantifying Dinosaur Pee

Example: $1 - 10^{-8}$

```
=====================================================
Subtracting 0.00000001 from 1.0
=====================================================
```

***** Express the two numbers in IEEE 754 format (float):**

```
1.00000000000000000000e+00: [ 0 01111111 (1) 00000000000000000000000 ]
9.99999993922529029078e-09: [ 0 01100100 (1) 01010111100110001110111 ]
```

***** Increase the smaller exponent so it matches the larger,**
shifting the mantissa to the right. In this case, we do it 27 times:

```
9.99999993922529029078e-09: [ 0 01100100 (1) 01010111100110001110111 ]
                            [ 0 01100101 (0) 101010111100110001110111 ]
                            [ 0 01100110 (0) 0101010111100110001110111 ]
                            [ 0 01100111 (0) 00101010111100110001110111 ]
                            [ 0 01101000 (0) 000101010111100110001110111 ]
                            [ 0 01101001 (0) 0000101010111100110001110111 ]
                            [ 0 01101010 (0) 00000101010111100110001110111 ]
                            [ 0 01101011 (0) 000000101010111100110001110111 ]
                            [ 0 01101100 (0) 0000000101010111100110001110111 ]
                            [ 0 01101101 (0) 00000000101010111100110001110111 ]
                            [ 0 01101110 (0) 000000000101010111100110001110111 ]
                            [ 0 01101111 (0) 0000000000101010111100110001110111 ]
                            [ 0 01110000 (0) 00000000000101010111100110001110111 ]
                            [ 0 01110001 (0) 000000000000101010111100110001110111 ]
                            [ 0 01110010 (0) 0000000000000101010111100110001110111 ]
                            [ 0 01110011 (0) 00000000000000101010111100110001110111 ]
                            [ 0 01110100 (0) 000000000000000101010111100110001110111 ]
                            [ 0 01110101 (0) 0000000000000000101010111100110001110111 ]
                            [ 0 01110110 (0) 00000000000000000101010111100110001110111 ]
                            [ 0 01110111 (0) 000000000000000000101010111100110001110111 ]
                            [ 0 01111000 (0) 0000000000000000000101010111100110001110111 ]
                            [ 0 01111001 (0) 00000000000000000000101010111100110001110111 ]
                            [ 0 01111010 (0) 000000000000000000000101010111100110001110111 ]
                            [ 0 01111011 (0) 0000000000000000000000101010111100110001110111 ]
                            [ 0 01111100 (0) 00000000000000000000000101010111100110001110111 ]
                            [ 0 01111101 (0) 000000000000000000000000101010111100110001110111 ]
                            [ 0 01111110 (0) 0000000000000000000000000101010111100110001110111 ]
                            [ 0 01111111 (0) 00000000000000000000000000101010111100110001110111 ]
```

Quantifying Dinosaur Pee

Example: $1 - 10^{-8}$

```
==============================================================
```

Subtracting 0.00000001 from 1.0 continued

```
==============================================================
```

***** Now do the subtraction:**

```
1.00000000000000000000e+00: [ 0 01111111 (1)   00000000000000000000000000000000000000000000000000 ]
-
                            [ 0 01111111 (0)   00000000000000000000000000101010111100110001110111 ]
=
                            [ 0 01111111 (0)   11111111111111111111111111010101000011001110001001 ]
```

***** Now we need to normalize (left-shift and decrease exponent):**

```
                            [ 0 01111111 (0)   11111111111111111111111111010101000011001110001001 ]
                            [ 0 01111110 (1)   1111111111111111111111111010101000011001110001001 ]
```

***** Round**

```
                            [ 0 01111110 (1)   1111111111111111111111111010101000011001110001001 ]
                            [ 0 01111110 (10)  00000000000000000000000 ]
```

***** Now need to re-normalize:**

```
                            [ 0 01111110 (10)  00000000000000000000000 ]
                            [ 0 01111111 (1)   000000000000000000000000 ]
1.00000000000000000000e+00: [ 0 01111111 (1)   00000000000000000000000 ]
```

***** Compare with FPU result:**

```
1.00000000000000000000e+00: [ 0 01111111 (1)   00000000000000000000000 ]
====================================================
```

Quantifying Dinosaur Pee

Floating-Point Type Accuracy

	Mantissa bits	# of decimal digits around 1 precision	Smallest normal > 0	# of decimal digits around 0
float	23	7.2	2^{-126}	38.2
double	52	15.9	2^{-1022}	307.9
long double	64	19.6	2^{-16382}	4931.8
__float128	112	34.0	2^{-16382}	4931.8

Inquerity's Approach

Pure water remaining after 165M years

Using `__float128` with *our* data

compounding cycle	k	$y(t)/y_0$	error* (%)
continuous	$k \to \infty$	0.03107702434070	0.00000000000003
per day	365.25	0.03107702436966	0.00000009318755
per year	1	0.03107702320596	0.00000365136619
per 10 years	10^{-1}	0.03107701299304	0.00003651462674
per 100 years	10^{-2}	0.03107691086412	0.00036514620902
per 1,000 years	10^{-3}	0.03107588957926	0.00365144817291
per 10,000 years	10^{-4}	0.03106567715860	0.03651309073774
per 100,000 years	10^{-5}	0.03096359581703	0.36499158486500
per 1,000,000 years	10^{-6}	0.02994713891235	3.63575809563764

$$* \text{ error} = 100 \times \frac{|\text{discrete} - \text{continuous}|}{\text{continuous}}$$

Data:
t = 165M years (ours)
R = 2.10381e-8/year (ours)

Inquerity's Approach

Pure water remaining after 165M years

Using `__float80` with *our* data

compounding cycle	k	$y(t)/y_0$	error* (%)
continuous	$k \to \infty$	0.03107702434070	0.00000000000003
per day	365.25	0.03107702436966	0.00000009318755
per year	1	0.03107702320596	0.00000365136619
per 10 years	10^{-1}	0.03107701299304	0.00003651462674
per 100 years	10^{-2}	0.03107691086412	0.00036514620902
per 1,000 years	10^{-3}	0.03107588957926	0.00365144817291
per 10,000 years	10^{-4}	0.03106567715860	0.03651309073774
per 100,000 years	10^{-5}	0.03096359581703	0.36499158486500
per 1,000,000 years	10^{-6}	0.02994713891235	3.63575809563764

$$* \text{ error} = 100 \times \frac{|\text{discrete} - \text{continuous}|}{\text{continuous}}$$

Data:
t = 165M years (ours)
R = 2.10381e-8/year (ours)

Inquerity's Approach

Pure water remaining after 165M years

Using `double` with *our* data

compounding cycle	k	$y(t)/y_0$	error* (%)
continuous	$k \to \infty$	0.03107702434070	0.00000000000002
per day	365.25	0.03107708234327	0.00018664134340
per year	1	0.03107702300165	0.00000430879926
per 10 years	10^{-1}	0.03107701301647	0.00003643922319
per 100 years	10^{-2}	0.03107691086479	0.00036514407125
per 1,000 years	10^{-3}	0.03107588957936	0.00365144786791
per 10,000 years	10^{-4}	0.03106567715858	0.03651309079918
per 100,000 years	10^{-5}	0.03096359581703	0.36499158487144
per 1,000,000 years	10^{-6}	0.02994713891235	3.63575809563677

$$* \text{ error} = 100 \times \frac{|\text{discrete} - \text{continuous}|}{\text{continuous}}$$

Data:
t = 165M years (ours)
R = 2.10381e-8/year (ours)

Inquerity's Approach

Pure water remaining after 165M years

Using `float` with *our* data

compounding cycle	k	$y(t)/y_0$	error* (%)
continuous	$k \to \infty$	0.03107703104615	0.00002157688344
per day	365.25	1.0	3117.81129697925906
per year	1	1.0	3117.81129697925906
per 10 years	10^{-1}	0.01956707425416	37.03684741614751
per 100 years	10^{-2}	0.03199512138963	2.95426305576596
per 1,000 years	10^{-3}	0.03107588957936	0.00365144786791
per 10,000 years	10^{-4}	0.03105368465185	0.07510271444032
per 100,000 years	10^{-5}	0.03096382319927	0.36425991171359
per 1,000,000 years	10^{-6}	0.02994706295431	3.63600251428315

$$* \text{ error} = 100 \times \frac{|\text{discrete} - \text{continuous}|}{\text{continuous}}$$

Data:
t = 165M years (ours)
R = 2.10381e-8/year (ours)

Inquerity's Approach

Pure water remaining after 186M years

Using `__float128` with *Inquerity's* data

compounding cycle	k	$y(t)/y_0$	error* (%)
continuous	$k \to \infty$	$1.3900648155\times10^{-20}$	0.0000000000002
per day	365.25	$1.3900647916\times10^{-20}$	0.0000017184243
per year	1	$1.3900570037\times10^{-20}$	0.0005619697115
per 10 years	10^{-1}	$1.3899866999\times10^{-20}$	0.0056195648640
per 100 years	10^{-2}	$1.3892838455\times10^{-20}$	0.0561822679079
per 1,000 years	10^{-3}	$1.3822736850\times10^{-20}$	0.5604868419449
per 10,000 years	10^{-4}	$1.3139803625\times10^{-20}$	5.4734464241542
per 100,000 years	10^{-5}	$7.8505325107\times10^{-21}$	43.5239823105510
per 1,000,000 years	10^{-6}	$1.6234881670\times10^{-23}$	99.8832077361464

$$* \text{ error} = 100 \times \frac{|\text{discrete} - \text{continuous}|}{\text{continuous}}$$

Data:
t = 186M years (theirs)
R = 6.73016e-10/day (theirs)

Inquerity's Approach

Pure water remaining after 186M years

Using `__float80` with *Inquerity's* data

compounding cycle	k	$y(t)/y_0$	error* (%)
continuous	$k \to \infty$	$1.3900648155\times10^{-20}$	0.0000000000002
per day	365.25	$1.3900647916\times10^{-20}$	0.0000017184243
per year	1	$1.3900570037\times10^{-20}$	0.0005619697115
per 10 years	10^{-1}	$1.3899866999\times10^{-20}$	0.0056195648640
per 100 years	10^{-2}	$1.3892838455\times10^{-20}$	0.0561822679079
per 1,000 years	10^{-3}	$1.3822736850\times10^{-20}$	0.5604868419449
per 10,000 years	10^{-4}	$1.3139803625\times10^{-20}$	5.4734464241542
per 100,000 years	10^{-5}	$7.8505325107\times10^{-21}$	43.523982310551O
per 1,000,000 years	10^{-6}	$1.6234881670\times10^{-23}$	99.8832077361464

$$* \text{ error} = 100 \times \frac{|\text{discrete} - \text{continuous}|}{\text{continuous}}$$

Data:
t = 186M years (theirs)
R = 6.73016e-10/day (theirs)

Inquerity's Approach

Pure water remaining after 186M years

Using `double` with *Inquerity's* data

compounding cycle	k	$y(t)/y_0$	error* (%)
continuous	$k \to \infty$	$1.3900648155 \times 10^{-20}$	0.0000000000001
per day	365.25	$1.3900670339 \times 10^{-20}$	0.0001595903162
per year	1	$1.3900570117 \times 10^{-20}$	0.0005613959873
per 10 years	10^{-1}	$1.3899866992 \times 10^{-20}$	0.0056196104371
per 100 years	10^{-2}	$1.3892838455 \times 10^{-20}$	0.0561822722312
per 1,000 years	10^{-3}	$1.3822736851 \times 10^{-20}$	0.5604868421384
per 10,000 years	10^{-4}	$1.3139803625 \times 10^{-20}$	5.4734464241426
per 100,000 years	10^{-5}	$7.8505325107 \times 10^{-21}$	43.5239823105558
per 1,000,000 years	10^{-6}	$1.6234881670 \times 10^{-23}$	99.8832077361464

$$* \text{ error} = 100 \times \frac{|\text{discrete} - \text{continuous}|}{\text{continuous}}$$

Data:
t = 186M years (theirs)
R = 6.73016e-10/day (theirs)

Inquerity's Approach

Pure water remaining after 186M years

Using `float` with *Inquerity's* data

compounding cycle	k	$y(t)/y_0$	error* (factor)
continuous	$k \to \infty$	$1.3900707186\times10^{-20}$	0.000004246636
per day	365.25	1.0	$7.19390915 \times 10^{19}$
per year	1	$5.5060133639\times10^{-20}$	2.960975993712
per 10 years	10^{-1}	$1.8169225719\times10^{-20}$	0.307077592095
per 100 years	10^{-2}	$1.4548599632\times10^{-20}$	0.046613040613
per 1,000 years	10^{-3}	$1.3847289410\times10^{-20}$	0.003853371112
per 10,000 years	10^{-4}	$1.3133968919\times10^{-20}$	0.058373766584
per 100,000 years	10^{-5}	$7.8505361258\times10^{-21}$	0.77066227481
per 1,000,000 years	10^{-6}	$1.6234889958\times10^{-23}$	855.22065754416

$$* \text{ error} = \frac{\max(discrete, continuous)}{\min(discrete, continuous)} - 1$$

Data:
t = 186M years (theirs)
R = 6.73016e-10/day (theirs)

Inquerity's Approach

Inquerity's *geometric sum* formula

Recall Inquerity's formula:

$$\%Pee = 100 \times \frac{P}{Q} \times \frac{(1-(1-\frac{P}{Q})^X)}{1-(1-\frac{P}{Q})}$$

$1-(1-x)=x$ (usually 😊) :

$$\%Pee = 100 \times \frac{P}{Q} \times \frac{(1-(1-\frac{P}{Q})^X)}{\frac{P}{Q}}$$

Cancel out the $\frac{P}{Q}$:

$$\%Pee = 100 \times (1-(1-\frac{P}{Q})^X)$$

Or, alternatively:

$$\%Pure = 100 \times (1-\frac{P}{Q})^X$$

Inquerity's Approach

Q1. How much water is pee?

They employed discrete probability

And arrived at a "correct" formula:

Let t be the number of time units

Let R be the decay rate per unit time

Let z be the *expected* fraction of tainted water

$$z(t) = 1 - \overbrace{(1 - \underbrace{R}_{\text{How small is this?}})^{t}}^{\text{How small is this?}}$$

Inquerity's Approach

Q1. How much water is pee?

Houston, *they had a problem !*

- Recall `sameBirthday(n)` cannot be represented with a `double` when `n` is greater than 183.

1 – *very small*
= BAD IDEA

- They could not represent $1-(1-\boldsymbol{R})^{\boldsymbol{t}}$ as a `double` or even a `long double`.
- They could, however, represent $1-\boldsymbol{R}$ as a `double`, as we have demonstrated.

Q1. How much water is pee?

They employed discrete probability

And arrived at a "correct" formula:

Let t be the number of time units

Let R be the decay rate per unit time

Let z be the *expected* fraction of tainted water

Let w be the *expected* fraction of pure water, 1-z

$$\boldsymbol{w(t) = (1 - R)^t}$$

Quantifying Dinosaur Pee

Q1. How much water is pee?

Untainted Water		Time (years)	
		$165{,}000{,}000$	$186{,}000{,}000$
Rate of Decay (per year)	R = $2.10381e-08$ (My Rate)	3.11% 3.94e19 Liters	2.00% 2.54e19 Liters
	R = $2.45794e-07$ (~10X My Rate)	2.44e-16% ~3070 Liters	1.40e-18% ~17.6 Liters

Final Answer: (d) Almost 100%

Quantifying Dinosaur Pee

Range of Numbers

	$largest < 1$	
float	0.9999 9994	7.2
double	0.9999 9999 9999 9999	15.9
long double (80 bits)	0.999 999 999 999 999 999 94	19.6
__float128	0.999 999 999 999 999 999 999 999 999 999 999 9	34.0

Quantifying Dinosaur Pee

Range of Numbers

	$smallest > 0$
float	0.00000000000000000000000000000000000001
double	0.000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000002
long double (80 bits)	0.000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 000000000000000000000000000000000000000000000000000000000000000000000000000000000003
__float128	0.000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 000000000000000000000000000000000000000000000000000000000000000000000000000000000000003

Quantifying Dinosaur Pee

Range of Denormalized Numbers

	$smallest > 0$
float	0.00000000000000000000000000000000000000000000001
double	0.0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 000000000000000000000002
long double (80 bits)	0.0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 000000000000000000000000000000000000000000000000003
__float128	0.0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 000000000000000000000000000000000000000000000000000000000000000003

Q2. What's the chance it's ***all*** pee?

What is Expected Value?

Expected Value (E) vs. Probability (P)

$$\boldsymbol{E}[\boldsymbol{X}] = \sum_{i} \boldsymbol{x}_i \boldsymbol{P}(\boldsymbol{X} = \boldsymbol{x}_i)$$

Flipping Coins

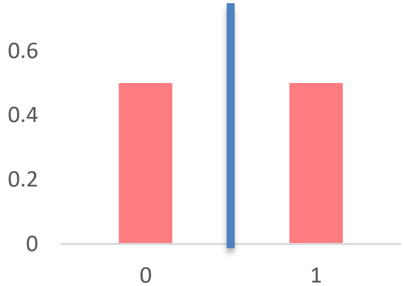
1 Coin

Expected number of heads?

0.5

Probability of exactly 1 head?

$$\frac{1}{2}$$



Quantifying Dinosaur Pee

Flipping Coins

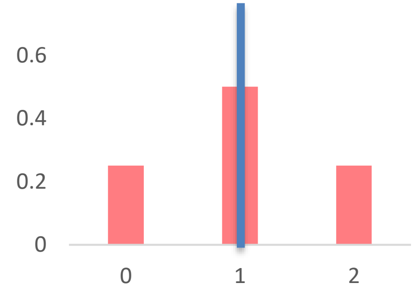
2 Coins

Expected number of heads?

1.0

Probability of exactly 1 head?

$$\frac{2}{4}$$



Flipping Coins

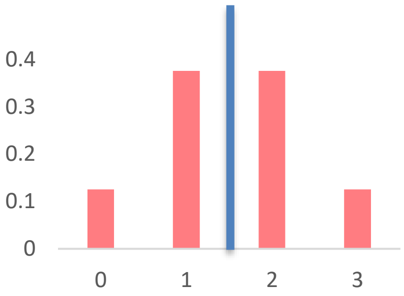
3 Coins

Expected number of heads?

1.5

Probability of exactly 1 head?

$$\frac{3}{8}$$



Flipping Coins

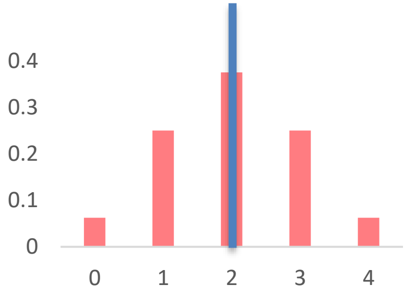
4 Coins

Expected number of heads?

2

Probability of exactly 1 head?

$$\frac{4}{16}$$



Quantifying Dinosaur Pee

Flipping Coins

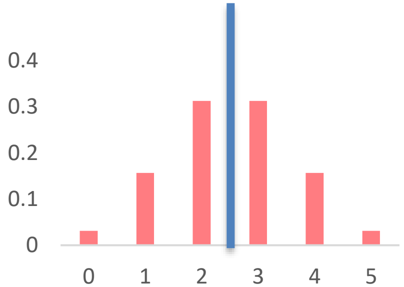
5 Coins

Expected number of heads?

2.5

Probability of exactly 1 head?

$$\frac{5}{32}$$



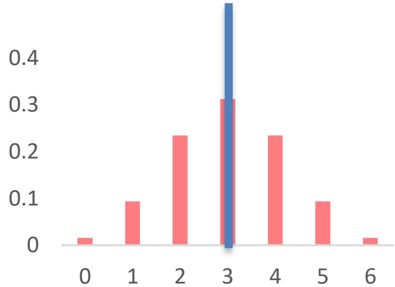
Flipping Coins

6 Coins

Expected number of heads?
3

Probability of exactly 1 head?

$$\frac{6}{64}$$



Flipping Coins

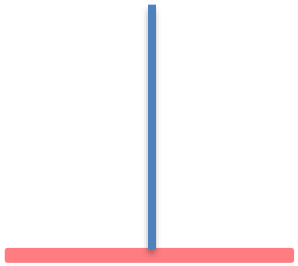
0 Coins

Expected number of heads?

0

Probability of exactly 1 head?

0

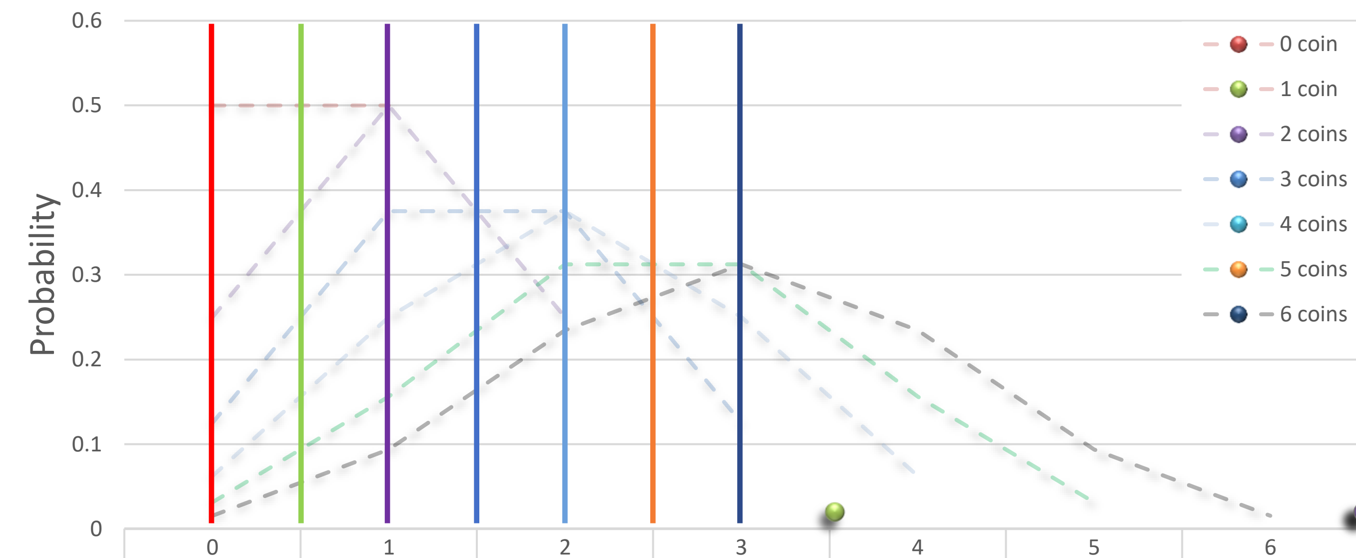


Quantifying Dinosaur Pee

Summary: Expected Value vs. Probability

$$E[X] = \sum_i x_i P(X = x_i)$$

Coin Tosses	Expected Heads
0	0
1	0.5
2	1.0
3	1.5
4	2.0
5	2.5
6	3.0



	0	1	2	3	4	5	6
0 coin	0						
1 coin	0.5	0.5					
2 coins	0.25	0.5	0.25				
3 coins	0.125	0.375	0.375	0.125			
4 coins	0.0625	0.25	0.375	0.25	0.0625		
5 coins	0.03125	0.15625	0.3125	0.3125	0.15625	0.03125	
6 coins	0.015625	0.09375	0.234375	0.3125	0.234375	0.09375	0.015625

Expected number of zero bits
Random Large Unsigned Integer

0	0	0	1	X						…

$$\underbrace{0 \quad 0 \quad 0}_{\text{First N bits}}$$

Probability first N are all 0 bits

Expected number of zero bits
Random Large Unsigned Integer

0	1	X								…

N = 1

P(exactly N leading 0 bits) = 0.25

Expected number of zero bits
Random Large Unsigned Integer

0	0	1	X						

N = 2

P(exactly N leading 0 bits) = 0.125

Expected number of zero bits
Random Large Unsigned Integer

0	0	0	1	X					

N = 3

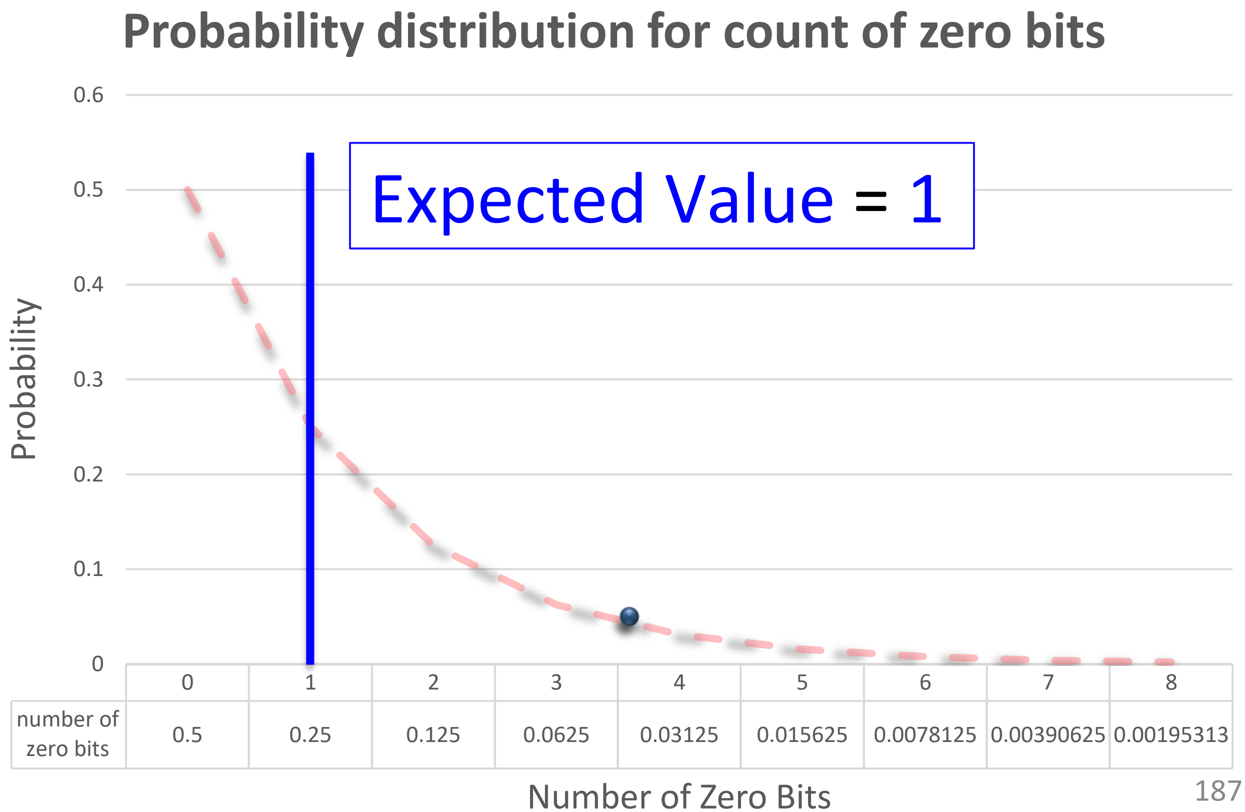
P(exactly N leading 0 bits) = 0.0625

Quantifying Dinosaur Pee

Expected number of zero bits

$$E[\#\ zero\ bits] = \sum_i i * P(zeros = i\,) = \sum_i i * 2^{-(i+1)} = 1$$

Number of zero bits	Probability
0	0.5
1	0.25
2	0.125
3	0.0625
…	…



Q2. What's the chance it's ***all*** pee?

Fact

Molecules per liter (kg) of water = 3.334e25

M = Expected # tainted water molecules:
0.97 * 3.334e25 molecules/kg * 1.26e21 kg =
4.07e46 molecules

N = total # water molecules:
3.334e25 molecules/kg * 1.26e21 kg =
4.2e46 molecules

Q2. Expected Value vs. Probability

Recall: $E[X] = \sum_i x_i P(X = x_i)$

$$P = e^{-Rt}$$

$$E = P * N = P * \underbrace{(1 + 1 + 1 + \ldots + 1 + 1 + 1)}_{N}$$

$$E = P * N = \underbrace{P*1 + P*1 + \ldots + P*1 + P*1}_{N}$$

Probability that a specific molecule is pure

Q2. What’s the chance it’s ***all*** pee?

Probability any specific molecule is:

$$\frac{M}{N} = 0.97$$

Probability all *N* molecules are tainted:

$$\left(\frac{M}{N}\right)^N = 10^{-5.76e44}$$

Q2. What's the probability all is tainted?

The world contains 4.2e46 molecules

$$P(all\ clean) = P(molecule\ tainted)^{4.2E46}$$

$$= 0.97^{4.2E46}$$

$$= e^{ln(0.97^{4.2E46})}$$

$$= 10^{4.2E46 * ln(0.97) * log_{10}(e)}$$

$$= 10^{4.2E46 * (-0.031593) * 0.43429}$$

$$= 10^{4.2E46 * (-0.031593) * 0.43429}$$

$$= 10^{-5.7628E44}$$

$$= 0.\underbrace{0\ 0\ 0\ 0\ 0\ 0\ 0\ 0 \ldots\ldots\ldots\ldots\ldots\ldots\ldots 0\ 0}_{5.76 * 10^{44}\ zeros}\ 1$$

Quantifying Dinosaur Pee

Q2. Chance every molecule is pee?

Probability All Pee		Time (years) 165,000,000	186,000,000
Rate of Decay (per year)	R = 2.10381e − 08 (My Rate)	10^{-5e44} ($5 * 10^{44}$ zeros)	10^{-4e44} ($4 * 10^{44}$ zeros)
	R = 2.45794e − 07 (~10X My Rate)	10^{-4e28} ($4 * 10^{28}$ zeros)	10^{-2e26} ($2 * 10^{26}$ zeros)

Final Answer: (b) Almost 0

Q3. What is Q2 if Q1 is ‘all but 1’?

A single untainted molecule is *expected:*

We rely on the well-known result that

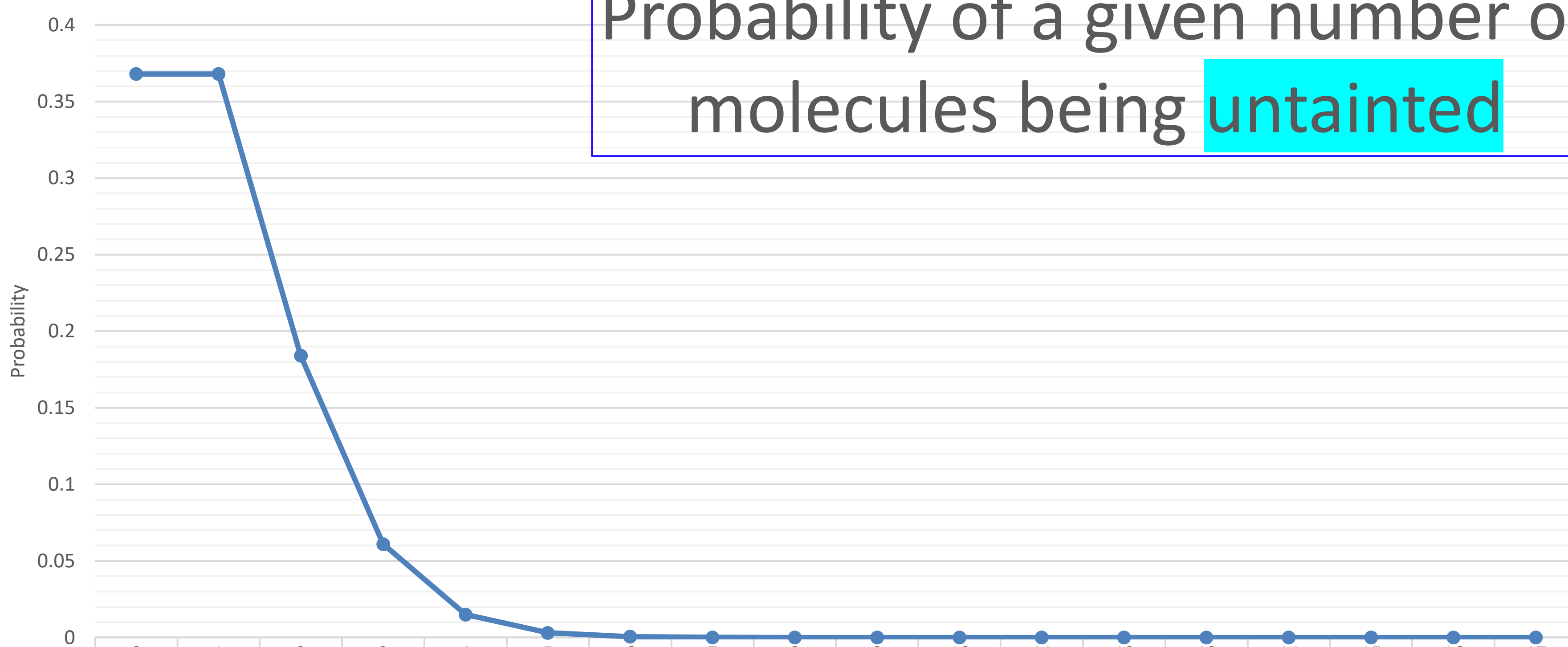
$$\lim_{n\to\infty}\left(1-\frac{1}{n}\right)^n = \frac{1}{e}$$

Quantifying Dinosaur Pee

Q3. What is Q2 if Q1 is ‘all but 1’?

Probability Density Function and Expected Value

Probability of a given number of molecules being untainted



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Probability	0.368	0.368	0.184	0.061	0.015	0.003	0.0005	7.1E-05	8.9E-06	9.9E-07	9.9E-08	9E-09	7.5E-10	5.8E-11	4.1E-12	2.8E-13	1.7E-14	1E-15

Number of pee molecules

Q3. Q2 if Q1 answer were 'all but 1'

Probability **All Pee if E = 1/N**		Time (years)	
		$165,000,000$	$186,000,000$
Rate of Decay (per year)	**R** = $2.10381\text{e}-08$ (My Rate)	$\frac{1}{e} = 36.7879\%$	
	R = $2.45794\text{e}-07$ (~10X My Rate)		

Final Answer: (c) near middle

Q4. Chance my cup is pure?

1 cup = 250 ml water = 8.3e24 molecules

$$P(all\ clean) = P(molecule\ clean)^{8.3E24}$$

$$= 0.03^{8.3E24}$$

$$= e^{ln(0.03^{8.3E24})}$$

$$= 10^{8.3E24*ln(0.03)*log_{10}(e)}$$

$$= 10^{8.3E24*(-3.47054)*0.43429}$$

$$= 10^{8.3E24*(-3.47054)*0.43429}$$

$$= 10^{-1.2501E25}$$

$$= 0.\underbrace{0\ 0\ 0\ 0\ 0\ 0\ 0\ 0 \ldots\ldots\ldots\ldots\ldots\ldots\ldots 0\ 0}_{1.25\ *\ 10^{25}\ zeros}\ 1$$

Quantifying Dinosaur Pee

Q4. Chance my cup is pure water?

Probability: All Pure		Time (years): 165,000,000	Time (years): 186,000,000
Rate of Decay (per year)	R = 2.10381e – 08 (My Rate)	$10^{-1.2e25}$ (10^{25} zeros!)	$10^{-1.4e25}$ (10^{25} zeros!)
Rate of Decay (per year)	R = 2.45794e – 07 (~10X My Rate)	10^{-1e26} (10^{26} zeros!)	10^{-2e26} ($2 * 10^{26}$ zeros!)

Final Answer: (b) Almost 0

Quantifying Dinosaur Pee

Question Recap

How did you do?

1. What fraction of all water is pee?

   (a) ~~0~~ (b) 0+ (c) ~0.5 (d) 1- (e) ~~1~~

2. What's the chance every molecule is pee?

   (a) ~~0~~ (b) 0+ (c) ~0.5 (d) 1- (e) ~~1~~

3. What would ans. **2.** be if ans. **1.** were "*all but 1 molecule*"?

   (a) ~~0~~ (b) 0+ (c) ~0.5 (d) 1- (e) ~~1~~

4. What's the chance my cup of water is pure?

   (a) ~~0~~ (b) 0+ (c) ~0.5 (d) 1- (e) ~~1~~

Quantifying Dinosaur Pee

Takeaways

1. Dinosaurs created a lot of pee.
2. Molecules are *very* small.
3. Even if there's almost no *expected* water left, the probability that ***all*** the remaining water is pee is *very* small indeed.
4. Just like the *birthday problem*, we need to think about how we can avoid representing values *very*, *very* close to 1 — as even a `long double` might not get the job done.

The Takeaway

To X,

or $1 - X$,

that is the question.

Dino-steak

Thank you!

Visit www.TechAtBloomberg.com/cplusplus to learn more about Bloomberg's involvement with the C++ community.

Bloomberg Engineering

TechAtBloomberg.com