Sequences and Summation
Sequences

Informally, a sequence is an infinite progression of objects (usually numbers), consisting of a first, a second, a third, and so on. The members of a sequence are called elements or terms.

Example sequence: 2,4,6,8,10, ...

It is customary to denote sequences with the letters $a, b, c$ and to use subscript notation to refer to individual terms: $a_n$ is the $n$th term of the sequence $a$. The notation $\{a_n\}$ refers to the entire sequence, not to the set of the terms. A sequence is an ordered list, whereas a set is an unordered collection of objects.

Example: if $\{a_n\} = 2,4,6,8,10, ...$ then $a_0 = 2, a_1 = 4$.

To simplify some of the formulas, the index $n$ will always start with $n = 0$ in this presentation. So for us, the initial term of the sequence is $a_0$ - the zeroth term. This choice comes at a price. Speaking of the “first” term of the sequence is now ambiguous. Is it $a_0$, or is it $a_1$? In the rigorous sense defined above - $a_n$ is the $n$th term of the sequence – the first term is $a_1$. In a colloquial sense, where “first” is taken as a synonym for initial – the first term is $a_0$.

[Because of these difficulties, many mathematicians prefer the more natural one-based indexing or numbering where the initial element is $a_1$ and there is never any confusion what is meant by the “first” term of a sequence.]
The disagreement among mathematicians over whether the natural numbers should include zero or not – and whether the ‘first’ element of a sequence is the “first” or the “zeroth” - is reflected in a corresponding disagreement among programming languages concerning the proper indexing of arrays.

In the C family of programming languages, as well as in Python, Javascript, Ruby and many others, array indices start at zero. Therefore, if you have an array declared by

```c
int numbers[10];
```

the proper way of looping through these elements is

```c
for(int i=0; i < 10; i++)
    numbers[i] ....
```

Trying to access or assign a value to `number[10]` is an illegal operation.

Other programming languages use one-based indexing. This includes prehistoric ones such as COBOL and Fortran, but also newer and almost popular ones like Lua. The Wolfram and MATLAB languages are also one-based.
How not to define a sequence

On a previous slide, we defined a sequence by giving the first 5 terms and expected that a reasonable reader would understand that we mean the sequence of positive even numbers.

This way of defining a sequence, by giving finitely many terms of it and expecting the reader to recognize a pattern in them is mathematically indefensible because there is always more than one conceivable pattern to continue a sequence, and, more importantly, a sequence does not have to fit any pattern in the first place. Each term is independent from all other terms and can assume any value.

Given \( \{a_n\} = 1,2,3, \ldots \) \( \{a_n\} \) could be the sequence that repeats these 3 numbers in perpetuity: \( \{a_n\} = 1,2,3,1,2,3, \ldots \) or \( \{a_n\} \) could be constant after the third term: \( \{a_n\} = 1,2,3,7,7,7,7, \ldots \)

If you think that these examples are far-fetched and exaggerate the issue of misunderstanding, consider the following example: \( \{a_n\} = 1,2,4, \ldots \) could represent \( \{a_n\} = 1,2,4,8,16,32,64, \ldots \) (each term is double the previous) but also \( \{a_n\} = 1,2,4,7,11,16, \ldots \) where the \( n \)th term plus \( n \) produces the next term, for all \( n \).
Defining a sequence properly

A proper definition of a sequence requires us to define all terms, not just finitely many of them. This can be done in two ways, directly and recursively. (We will discuss recursive definition later in this presentation).

A direct definition gives each $a_n$ as a function of $n$. We often just give an equation for $a_n$ without bothering to quantify the “for all $n \in \mathbb{N}_0$“.

Examples:

$a_n = 2n$ is the sequence of nonnegative even numbers.

$a_n = n^2$ is the sequences of squares.

$a_n = 2^n$ is the sequence of powers of 2 that are integers.
Sequences as Functions

Technically, a sequence is a special kind of function, namely a function whose domain is $\mathbb{N}_0$. Therefore, we could use standard function notation to represent sequences and write $f(n)$ instead $a_n$, but we use the latter for reasons of tradition.
Arithmetic sequences

A sequence that has a constant difference between successive terms is called arithmetic. An arithmetic sequence has the form

\[ a, a + d, a + 2d, a + 3d, a + 4d, \ldots \]

Where \( a \) is the first term and \( d \) is the common difference between successive terms. The general formula is

\[ a_n = a + nd. \]

(Important: that’s the general formula assuming zero-based numbering. Think about what the correct formula is for one-based numbering.)

An arithmetic sequence is just a linear function with a domain restricted to the natural numbers.

Example: \( a_n = 1 + 2n \) is arithmetic with \( a = 1 \) and \( d = 2 \).
A sequence that has a constant quotient between successive terms is called geometric. A geometric sequence has the form

\[ a, aq, aq^2, aq^3, \ldots \]

Where \( a \) is the first term and \( q \) is the common quotient between successive terms. The general formula is \( a_n = aq^n \).

Again, that formula is only correct for zero-based numbering. Think about what the formula is for one-based numbering.

A geometric sequence is just an exponential function with a domain restricted to the natural numbers.

Example: \( a_n = 3 \cdot 2^n \) is arithmetic with \( a = 3 \) and \( q = 2 \).
Recursive Definition

A recursive definition gives each term of a sequence as a function of previous sequence terms:

\[ a_n = f(a_{n-1}, a_{n-2}, \ldots, a_{n-k}) \]

This equation is called a recurrence relation, or more precisely, a k-step recurrence relation. A recursive definition involving a k-step recurrence relation requires the values of the first k terms: \( a_0, a_1, \ldots, a_k \). These values are called the initial conditions.

For example, \( a_n = a_{n-1} + 2 \) and \( a_0 = 0 \) defines the sequence of non-negative even numbers recursively. The equation \( a_n = a_{n-1} + 2 \) is the recurrence relation.

Each arithmetic sequence \( a_n = a + nd \) has the recursive definition \( a_n = a_{n-1} + d, a_0 = a \).

Each geometric sequence \( a_n = aq^n \) has the recursive definition \( a_n = qa_{n-1}, a_0 = a \).

These three recursive definitions all involve 1-step relations.
Efficiently Computing Arithmetic Sequences (1)

Suppose you need to sample $[1,3] \times [2,5]$, i.e. the rectangle in $\mathbb{R}^2$ consisting of the points $(x,y)$ where $x$ goes from 1 to 3 and $y$ goes from 2 to 5.

You decide to create a regular grid by subdividing each interval into $N$ smaller parts, and then you loop over all the grid points as follows:

```plaintext
for(n=0;n<=N;n++)
  for(m=0;m<=N;m++) {
    x = 1 + (3-1)/N*n;
    y = 2 + (5-2)/N*m;
    procedure(x,y);
  }
```

If $N$ is large and the procedure to be carried out is not particularly expensive computationally, then the total running time of these nested loops may well be dominated by the repetitive linear transformations that compute $x$ from $n$ and $y$ from $m$.

A first speedup is gained by the realization that $x$ does not change in the inner loop because it only depends on the outer looping variable $n$. Additionally, there is no reason to re-compute the slopes of the two linear functions

$$x(n) = 1 + (3-1)/N \times n$$
$$y(n) = 2 + (5-2)/N \times m$$

again and again. It is more efficient to compute the slopes once and for all at the beginning of the program. Based on this, we optimize our program as follows:
Efficiently Computing Arithmetic Sequences (2)

Optimized program:

\[ S_x = \frac{2}{N}; \]
\[ S_y = \frac{3}{N}; \]

for \( n = 0 \) to \( N \) do
  \[ x = 1 + S_x \times n; \]
  for \( m = 0 \) to \( N \) do
    \[ y = 2 + S_y \times m; \]
    procedure \((x, y)\);
  end
end

This is better, but still requires one addition and one multiplication in each inner loop, not counting what the procedure might do with \( x \) and \( y \).
Efficiently Computing Arithmetic Sequences (3)

We can save the multiplications by realizing that

\[ x(n) = 1 + Sx \times n \]

and

\[ y(n) = 2 + Sy \times m \]

are arithmetic sequences with first terms 1 and 3, and constant differences \( Sx \) and \( Sy \). We can get each \( x \) from the previous by adding \( Sx \), and each \( y \) from the previous by adding \( Sy \), i.e. by using the recursive definitions of the sequences. We just need to make sure we initialize \( x \) and \( y \) with the correct values – \( x \) just once, and \( y \) once in each outer loop.

The program with this final optimization is printed on the right.

```
Sx = 2/N;
Sy = 3/N;
x = 1;
for(n=0;n<=N;n++) {
    y = 2;
    for(m=0;m<=N;m++) {
        procedure(x,y);
        y += Sy;
    }
    x += Sx;
}
```
An Example of a Multi-Step Recurrence Relation

The **Fibonacci Sequence** is the sequence \( \{f_n\} \) defined by the initial conditions \( f_0 = 1, \ f_1 = 1 \) and the recurrence relation \( f_n = f_{n-1} + f_{n-2} \) for \( n = 2,3,4, \ldots \):

\[
\{f_n\} = 1,1,2,3,5,8,13,21,34,55, \ldots
\]

\( f_n = f_{n-1} + f_{n-2} \) is a two-step recurrence.
Computing The Fibonacci Sequence: A Cautionary Tale about Recursion (I)

Not every recursion lends itself directly to a computationally efficient recursive implementation.

Based on the recursive definition of the Fibonacci sequence, some students will implement the sequence as given in the box on the right.

```
int F(int n)
{
    if (n==0) or (n==1)
        return 1;
    else
        return F(n-1)+F(n-2);
}
```

If you run this code, you will find that it produces the first few terms of the sequence just fine. But once the n values get even moderately large, the function takes a long time to complete.

That is because this implementation isn’t just a little inefficient. It is disastrously, monstrously, inefficient. The problem is that the recursive computation of \( F(n-1) \) requires the full recursive computation of \( F(n-2) \), which is then repeated again to compute... \( F(n-2) \). This inefficiency compounds itself and causes this function to require a number of additions that **exponentially** increases with \( n \).
Computing The Fibonacci Sequence: A Cautionary Tale about Recursion (II)

Consider how our function computes $F(5)$. In the recursion tree, the entire $F(3)$ tree gets independently computed twice, which subsequently causes the $F(2)$ tree to get computed three times. The total number of additions is 7, when really only 4 additions were needed:

\[
\begin{align*}
F(2) &= F(1) + F(0), \\
F(3) &= F(2) + F(1), \\
F(4) &= F(3) + F(2), \\
F(5) &= F(4) + F(3).
\end{align*}
\]
It is not difficult to come up with a recursive formula for computing the number of additions required by our inefficient recursive algorithm.

The computation of each $F(n)$ requires the full and independent computation of $F(n-1)$ and $F(n-2)$, followed by an addition. Therefore, if we let $N(n)$ be the number of additions required to compute $F(n)$, then

$$N(n) = N(n-1) + N(n-2) + 1, \text{ with } N(1) = N(0) = 0.$$  

Using this formula, we can determine, for example, that computing $F(100)$ using our inefficient recursive algorithm requires 573,147,844,013,817,084,100 additions. An Intel Core i7 6950X running at 3 Ghz would take about 57 years to carry out this many additions.

An efficient, iterative algorithm that loops the recursion formula by repeatedly adding the last two terms together to compute the next one only requires 99 additions to compute $F(100)$. 
Summation

The **sigma notation** is a convenient way to express lengthy sums that follow a pattern:

\[
\sum_{k=n}^{N} f(k) = f(n) + f(n + 1) + \cdots + f(N)
\]

The index variable \( k \) always runs from the integer \( n \) to the integer \( N \) in steps of 1. Examples:

\[
\sum_{k=1}^{5} k^2 = 1^2 + 2^2 + 3^2 + 4^2 + 5^2
\]

\[
\sum_{k=4}^{8} 2^k = 16 + 32 + 64 + 128 + 256
\]

\[
\sum_{k=1}^{5} \frac{k}{2} = \frac{1}{2} + \frac{2}{2} + \frac{3}{2} + \frac{4}{2} + \frac{5}{2}
\]

Since the terms in a sum can be arbitrarily rearranged, and common multiplicative constants can be factored out, we have the general laws

\[
\sum_{k=n}^{N} (f(k) + g(k)) = \sum_{k=n}^{N} f(k) + \sum_{k=n}^{N} g(k)
\]

\[
\sum_{k=n}^{N} cf(k) = c \sum_{k=n}^{N} f(k).
\]
Sums of Consecutive Integers

There is a convenient summation formula available for the sum of the first $n$ positive integers:

$$\sum_{k=1}^{n} k = 1 + 2 + 3 + \cdots + (n - 1) + n = \frac{n(n + 1)}{2}$$

Such a formula for a sigma sum is known as a “closed form” formula.

For even $n$, this formula has a simple explanation. The first and the last term have a sum of $n + 1$. The second and the second-to-last term also have a sum of $n + 1$, and so on. Since there are $\frac{n}{2}$ such pairs, the sum is $\frac{n(n+1)}{2}$. The formula is also valid for odd $n$. Think about how this explanation needs to be adjusted for that case.

Examples:

$$1 + 2 + 3 + \cdots + 100 = \frac{100 \cdot 101}{2} = 50 \cdot 101 = 50050$$

$$\sum_{k=50}^{100} k = \sum_{k=1}^{100} k - \sum_{k=1}^{49} k = \frac{100 \cdot 101}{2} - \frac{49 \cdot 50}{2}$$
The index variable in a sigma sum is “invisible to the outside”!

Later, when we study proofs of summation formulas by induction, we will need to consider sequences defined by sigma sums, such as

\[ a_n = \sum_{k=1}^{n} k. \]

Observe that \( a_n \) is a function of \( n \) alone. It is not a function of \( k \). \( k \) is like a local variable used in a function in a computer program that is used to compute the output, but invisible to the outside. Evaluating this \( a_n \) for different \( n \) values means changing the upper limit in the summation only, while leaving the lower index and the expression of what is being summed alone. For example,

\[ a_{n+1} = \sum_{k=1}^{n+1} k. \]

The following equations are incorrect:

\[ a_{n+1} = \sum_{k=1}^{n+1} (k + 1), \quad a_{n+1} = \sum_{k=2}^{n+1} (k + 1) \]
Application to Arithmetic Sums

Using the summation formula we just learned, we can evaluate all arithmetic sums, i.e. all sums of the form

\[ \sum_{k=1}^{n} (a + kd) = \sum_{k=1}^{n} a + d \sum_{k=1}^{n} k = na + d \frac{n(n + 1)}{2} \]

Example:

\[ \sum_{k=1}^{100} (2 + 3k) = 200 + 3 \cdot \frac{100 \cdot 101}{2} \]
Sums of Consecutive Squares

Summation formulas are also available for the sum of the squares and cubes of the first \( n \) positive integers:

\[
\sum_{k=1}^{n} k^2 = 1^2 + 2^2 + 3^2 + \cdots + (n-1)^2 + n^2 = \frac{n(n+1)(2n+1)}{6}
\]

\[
\sum_{k=1}^{n} k^3 = 1^3 + 2^3 + 3^3 + \cdots + (n-1)^3 + n^3 = \frac{n^2(n+1)^2}{4}
\]

Such formulas for \( \sum_{k=1}^{n} k^p \) exist, in fact, for all positive integers \( p \).

Example: \( 1 + 4 + 9 + 16 + 25 + 36 + 49 + 64 + 81 + 100 = \frac{10 \cdot 11 \cdot 21}{6} \)
The Laws of Arithmetic Have Not Been Repealed Just Because We Are Using The Sigma Symbol For Sums

Some students are tempted to simplify sigma sums of products as the product of the sums, like this:

\[
\sum_{k=1}^{n} k(k + 1) = \sum_{k=1}^{n} k \cdot \sum_{k=1}^{n} (k + 1)
\]

This is bad algebra which ignores the distributive law. \((a+b)(x+y)\) is not \(ax + by\). When you distribute on the right side, you multiply every term in “k” sum by every term in the “k+1” sum (a total of \(n^2\) terms), not just corresponding terms with each other.

The correct simplification here is to distribute \(k(k + 1)\):

\[
\sum_{k=1}^{n} k(k + 1) = \sum_{k=1}^{n} k^2 + k = \sum_{k=1}^{n} k^2 + \sum_{k=1}^{n} k.
\]
Index Shifting

Let us consider another summation example:
\[
\sum_{k=1}^{50} (k + 1)^2 = 2^2 + 3^2 + \cdots + 51^2
\]
Since we have a summation formula for the \( k^2 \), we could evaluate this sum by expanding \( (k + 1)^2 \) into \( k^2 + 2k + 1 \):
\[
\sum_{k=1}^{50} (k + 1)^2 = \sum_{k=1}^{50} k^2 + 2 \sum_{k=1}^{50} k + \sum_{k=1}^{50} 1 = \frac{50 \cdot 51 \cdot 101}{6} + 2 \cdot \frac{50 \cdot 51}{2} + 50 = 45525
\]
There is a better way though, which is to perform an **index shift**. Index shifting means to increase the limits of the index variable \( k \) by some integer constant \( c \) and simultaneously substitute \( k - c \) for \( k \) in the expression being summed:
\[
\sum_{k=n}^{N} f(k) = \sum_{k=n+c}^{N+c} f(k - c)
\]
If we apply an index shift with \( c = 1 \) to our example sum, we get
\[
\sum_{k=1}^{50} (k + 1)^2 = \sum_{k=2}^{51} k^2 = \sum_{k=1}^{51} k^2 - 1 = \frac{51 \cdot 52 \cdot 103}{6} - 1 = 45525
\]
Geometric Sums

We shall determine a summation formula that helps us evaluate geometric sums, i.e. sums of the form

$$\sum_{k=0}^{n} aq^k$$

Since the constant multiplier $a$ can be factored out, we only require a formula for

$$\sum_{k=0}^{n} q^k = 1 + q + q^2 + \cdots + q^n$$

Let us multiply that expression by $(q - 1)$ and distribute:

$$(1 + q + q^2 + \cdots + q^n)(q - 1) = q + q^2 + \cdots + q^{n+1} - 1 - q - \cdots - q^n$$

Every “positive” term here is canceled by a “negative term”, except for two terms that remain:

$$(1 + q + q^2 + \cdots + q^n)(q - 1) = q^{n+1} - 1$$

If $q \neq 1$, we can divide by $(q - 1)$ and obtain

$$\sum_{k=0}^{n} q^k = \frac{q^{n+1} - 1}{q - 1}$$

For $q = 1$, $1 + q + q^2 + \cdots + q^n = n$. 
Geometric Sums II

To evaluate a geometric sum where the exponent does not start at zero, we could use the difference approach we have already encountered earlier:

\[ \sum_{k=n}^{N} q^k = \sum_{k=0}^{N} q^k - \sum_{k=0}^{n-1} q^k \]

There is a better way, however. We factor out the common highest factor of \( q \), which is \( q^n \) and then perform an index shift:

\[ \sum_{k=n}^{N} q^k = q^n \sum_{k=n}^{N} q^{k-n} = q^n \sum_{k=0}^{N-n} q^k = q^n \frac{q^{N-n+1} - 1}{q - 1} = \frac{q^{N+1} - q^n}{q - 1} \]
Geometric Sums III

Let us work a more complex example of a summation involving the geometric sum.

$$\sum_{k=5}^{20} \frac{3^{2k+1}}{5^{3k-1}} = \sum_{k=5}^{20} \frac{3^{2k} \cdot 3^1}{5^{3k} \cdot 5^{-1}} = 15 \sum_{k=5}^{20} \frac{9^k}{125^k} = 15 \sum_{k=5}^{20} \left( \frac{9}{125} \right)^k$$

We now use the formula we just discovered to evaluate:

$$15 \sum_{k=5}^{20} \left( \frac{9}{125} \right)^k = 15 \left( \frac{\left( \frac{9}{125} \right)^{21} - \left( \frac{9}{125} \right)^5}{9 \frac{125}{125} - 1} \right)$$
The geometric series (1)

If we let \( n \rightarrow \infty \) in the geometric sum \( \sum_{k=0}^{n} q^k \), we obtain the geometric series:

\[
\sum_{k=0}^{\infty} q^k
\]

Technically, this quantity is the limit of the geometric sums as \( n \rightarrow \infty \). It is a calculus fact that we shall not explain that this limit only exists when \( |q| < 1 \). In that case, the limit of \( q^{n+1} \) as \( n \rightarrow \infty \) is zero. Therefore,

\[
\sum_{k=0}^{\infty} q^k = \lim_{n \rightarrow \infty} \frac{q^{n+1} - 1}{q - 1} = \frac{1}{1 - q} \text{ for } |q| < 1.
\]

Example: \( 1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \frac{1}{16} + \cdots = \frac{1}{1-\frac{1}{2}} = 2 \). (In binary, this equation takes the form 1.1111111.... = 10.)
The geometric series (2)

Just like in the case of geometric sums, we can handle geometric series that don’t start with a power of 0, by factoring out the first power and then index shifting the remaining series:

$$\sum_{k=K}^{\infty} q^k = q^K \sum_{k=K}^{\infty} q^{k-K} = q^K \sum_{k=0}^{\infty} q^k$$

Then we use the known summation formula to obtain

$$\sum_{k=K}^{\infty} q^k = \frac{q^K}{1 - q} \text{ for } |q| < 1.$$  

Observe that the numerator in this formula is just the first term of the series.

Example: \(\frac{1}{4} + \frac{1}{8} + \frac{1}{16} + \frac{1}{32} + \frac{1}{64} + \cdots = \frac{\frac{1}{4}}{1 - \frac{1}{2}} = \frac{1}{2}.\) (In binary, this equation takes the form \(0.0111111\ldots = 0.1.)\)
You might already have realized by now that infinite, repeating decimal expansions are all actually geometric series.

Here is a famous decimal example:

\[
0.99999999... = 9 \sum_{k=1}^{\infty} \left( \frac{1}{10} \right)^k = 9 \cdot \frac{\frac{1}{10}}{1 - \frac{1}{10}} = 1.
\]

Generally, when you have a base-\(b\) number that starts with 0., followed by infinitely many digits (b-1), that number equals 1. For example, in binary, 0.11111111... = 1. In octal, 0.77777777... = 1. In hexadecimal, 0.FFFFFFFF... = 1.
Geometric Series and Infinite Repeating Decimal Expansions (2)

If a pattern of more than one digit repeats infinitely, we can also evaluate the number using the geometric series, thereby turning it into a fraction. Example (in decimal):

0.123123123123123123123123123123123123...

= $123 \cdot (0.001 + 0.000001 + 0.0000000001 + \cdots)$

= $123 \cdot \left(\left(\frac{1}{10}\right)^3 + \left(\frac{1}{10}\right)^6 + \left(\frac{1}{10}\right)^9 + \cdots\right)$

= $123 \cdot \left(\left(\frac{1}{1000}\right)^1 + \left(\frac{1}{1000}\right)^2 + \left(\frac{1}{1000}\right)^3 + \cdots\right)$

= $123 \cdot \sum_{k=1}^{\infty} \left(\frac{1}{1000}\right)^k = 123 \cdot \frac{\frac{1}{1000}}{1 - \frac{1}{1000}} = 123 \cdot \frac{1}{1000 - 1} = \frac{123}{999} = \frac{41}{333}$.

We could use the geometric series to prove the general formula: if $s = a_1 \ldots a_n$ is a finite sequence of decimal digits, then

$$0.\overline{a_1\ldots a_n} = \frac{a_1 \ldots a_n}{\overline{9\ldots9}}.$$
An Application of the Geometric Series to Percentage Problems (1)

The price of a product was raised by 2% and is now $100. What was the original price?

Many students who first learn percentages will think that the answer is 2% less of $100, or $98. This is wrong because of the shifting baseline: the increase was 2% of the original price, not 2% of the increased price. Reducing the increased price by 2% reduces it by too much. The correct original price must therefore have been more than $98.

The correct solution you would learn in a lower-level class is to set up an equation for the original price $p$, taking into account that increasing a quantity by 2% means multiplying it by 1.02:

\[ p \cdot 1.02 = 100 \]

From this, it follows that

\[ p = \frac{100}{1.02} \approx 98.04 \]

rounded to the nearest cent. It turns out that our wrong answer $98 was actually a fairly good approximation. This should not be too much of a surprise since the percent increase was small, so the difference between 2% of the original price and 2% of the increased price was small as well. The geometric series can help us understand this phenomenon quantitatively.
An Application of the Geometric Series to Percentage Problems (2)

On the previous page, we saw that the original price can be recovered as $p = \frac{100}{1.02}$. Raising the price by 2% meant multiplying by 1.02; un-doing that increase meant dividing by 1.02. Dividing by 1.02 however is not the same as decreasing the increased amount by 2%.

The geometric series sheds some light on this. Dividing by 1.02 means multiplying by $\frac{1}{1.02} = \frac{1}{1+0.02} = \frac{1}{1-(-0.02)}$. The reason we would write the multiplier that way is because that is the format of the right side of the geometric series:

$$\sum_{k=0}^{\infty} q^k = \frac{1}{1-q}$$

It follows that

$$\frac{1}{1-(-0.02)} = 1 + (-0.02) + (-0.02)^2 + (-0.02)^3 + (-0.02)^4 + \cdots$$

Since the number 0.02 is small in absolute value, its powers quickly become negligible. Retaining just the first three terms and simplifying, we get the approximation

$$\frac{1}{1-(-0.02)} \approx 1 - 0.02 + 0.0004$$
An Application of the Geometric Series to Percentage Problems (3)

We found \[
\frac{1}{1-(-0.02)} \approx 1 - 0.02 + 0.0004 = 100\% - 2\% + 0.04\% = 100\% - 1.96\%
\]

Remember that this is the multiplier that takes us back to the old price after a 2\% increase was applied. What this formula is saying is that to recover the old price approximately, we just need to subtract 2\% from the new price (i.e. apply the popular wrong solution) and correct it by adding 0.04\% of the new price. The 0.04\% does not come out of nowhere, it’s simply the 2\% squared.

Applied to the new price of $100, this produces the old price correctly to the nearest cent: $100-$2+$0.04=$98.04. The percentage decrease of the old price relative to the new price as a baseline is 1.96\%.

In other words, knowledge of the geometric series empowers you to solve an inverse percentage problem, which ordinarily requires division by a decimal fraction and therefore the use of a calculator, solely with simple arithmetic in your head with sufficient accuracy, based on the approximation

\[
\frac{1}{1+x} \approx 1 - x + x^2 = 1 - (x - x^2)
\]

for small |x|. Let us look at one more problem like this and apply our shortcut.
An Application of the Geometric Series
to Percentage Problems (4)

What percentage reduction is needed to undo a 3% increase?

Our approximation approach produces the answer
3% - 0.09% = 2.91%. (0.09% is 0.0009, which is 3% = 0.03 squared.)

The exact answer is more complicated and requires a calculator: the multiplier needed to undo a 3% increase is \( \frac{1}{1.03} = 0.97087.. = 97.087.. \% \). Therefore, the percentage reduction needed is 2.913..%.

It turns out that our back-of-the-napkin answer 2.91% is a good approximation.
Telescoping Sums

Let us consider

$$\sum_{k=1}^{n} \frac{1}{k(k+1)} = \frac{1}{1 \cdot 2} + \frac{1}{2 \cdot 3} + \cdots + \frac{1}{n(n+1)}.$$

The partial fraction decomposition of $\frac{1}{k(k+1)}$ is

$$\frac{1}{k(k+1)} = \frac{1}{k} - \frac{1}{k+1}.$$

By substituting this identity into the sum we get

$$\sum_{k=1}^{n} \frac{1}{k(k+1)} = \left(\frac{1}{1} - \frac{1}{2}\right) + \left(\frac{1}{2} - \frac{1}{3}\right) + \left(\frac{1}{3} - \frac{1}{4}\right) + \cdots + \left(\frac{1}{n} - \frac{1}{n+1}\right).$$

We can see that all terms in this sum cancel except $\frac{1}{1}$ and $-\frac{1}{n+1}$. The sum collapses like an old-style telescope and is therefore named a **telescoping sum**. Therefore,

$$\sum_{k=1}^{n} \frac{1}{k(k+1)} = 1 - \frac{1}{n+1}.$$