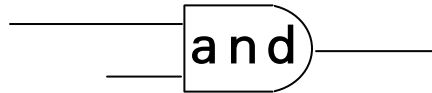


# Mathematics for the Digital Age



# Programming in Python

>>> Second Edition:  
with Python 3

Maria Litvin

Phillips Academy, Andover, Massachusetts

Gary Litvin

Skylight Software, Inc.

Skylight Publishing  
Andover, Massachusetts

Skylight Publishing  
9 Bartlet Street, Suite 70  
Andover, MA 01810

web: <http://www.skylit.com>  
e-mail: [sales@skylit.com](mailto:sales@skylit.com)  
[support@skylit.com](mailto:support@skylit.com)

**Copyright © 2010 by Maria Litvin, Gary Litvin, and  
Skylight Publishing**

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of the authors and Skylight Publishing.

Library of Congress Control Number: 2009913596

ISBN 978-0-9824775-8-8 (soft cover)

ISBN 978-0-9824775-4-0 (hard cover)

The names of commercially available software and products mentioned in this book are used for identification purposes only and may be trademarks or registered trademarks owned by corporations and other commercial entities. Skylight Publishing and the authors have no affiliation with and disclaim any sponsorship or endorsement by any of these products' manufacturers or trademarks' owners.

1 2 3 4 5 6 7 8 9 10      15 14 13 12 11 10

Printed in the United States of America

## 4 Sequences, Sums, Iterations

### 4.1 Prologue

In math, a *sequence* is an infinite list of values. An element of a sequence is often called a *term*. We will deal only with numeric sequences — their terms are real numbers. The terms of a sequence are numbered by integers, starting from 1 or sometimes from 0:

$$a_1, a_2, \dots, a_n, \dots$$

or

$$b_0, b_1, \dots, b_n, \dots$$

The simplest sequence is the sequence of positive integers themselves: 1, 2, 3, 4, .... Another simple sequence is the sequence of positive odd integers: 1, 3, 5, 7, ....

Sometimes it is not obvious how a sequence is defined, given only its first few terms. For example: 1, 2, 5, 12, 27, .... It turns out we used the formula  $a_n = 2^n - n$  to calculate the  $n$ -th term. So, it often helps to include the formula for the  $n$ -th term in the description of the sequence. For example: 1, 2, 5, ...,  $2^n - n$ , .... Or we can say simply: “Consider a sequence  $\{a_n = 2^n - n\}$ .” The  $n$ -th term formula is called the *general term* of the sequence.

You can view a sequence of real numbers as a function whose domain is all positive integers (or all non-negative integers) and whose outputs are real numbers:  $f(i) = a_i$ . Like any other function, we can describe a sequence in words or with a formula. Sometimes, the only way to generate a long enough segment of a sequence is by using a computer. For example, the sequence of all prime numbers: 2, 3, 5, 7, 11, ... (An integer is called a prime if it is greater than 1 and is evenly divisible only by 1 and by itself.) In this sequence, we can find each term by looking for the smallest number not evenly divisible by any of the preceding terms. Another example: the sequence of all digits of  $\pi$ : 3, 1, 4, ...

There is a database of sequences of integers on the Internet, at [www.research.att.com/~njas/sequences/Seis.html](http://www.research.att.com/~njas/sequences/Seis.html). It is called *The On-Line Encyclopedia of Integer Sequences*. The database contains over 100,000 “interesting” sequences (that is, sequences that came up in one mathematical problem or another). About 10,000 new interesting sequences arrive every year.



We say that a sequence *converges* to a certain number (called the *limit* of the sequence) if its terms get closer and closer to that number as  $n$  increases. For example,  $a_n = \frac{1}{2^n}$  converges to 0;  $a_n = \frac{(-1)^n}{2^n}$  also converges to 0. The sequence  $1, -1, 1, -1, 1, \dots$  bounces around zero but does not converge to any limit;  $1, 2, 4, 8, 16, \dots$  does not converge because its terms get bigger and bigger as  $n$  increases. When a sequence does not converge, we say that it *diverges*.

## 4.2 Arithmetic and Geometric Sequences

There are two types of sequences that come up frequently in mathematical problems: the *arithmetic sequence* and the *geometric sequence*.

**In an arithmetic sequence, the difference between any two consecutive terms is the same.**

In other words, in an arithmetic sequence,  $a_n = a_{n-1} + d$ , where  $d$  is some constant.  $d$  is called the *common difference*. Thus an arithmetic sequence has the form  $c, c + d, c + 2d, \dots, c + (n-1)d, \dots$ . The general term of an arithmetic sequence can be written as  $a_n = c + (n-1)d$ . The simplest arithmetic sequence is, again, the sequence of all positive integers.

### Example 1

$1, 3, 5, \dots, 2n-1, \dots$  ( $d = 2$ ) and  $5, 15, 25, \dots, 5+10(n-1), \dots$  ( $d = 10$ ) are both arithmetic sequences.



**In a geometric sequence the ratio of the next term to the previous is constant.**

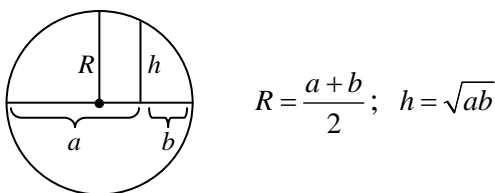
In other words, in a geometric sequence,  $a_n = a_{n-1} \cdot r$ , so it has the form  $c, cr, cr^2, \dots, cr^{n-1}, \dots$ . The constant  $r$  is called the *common ratio*. The general term of a geometric sequence can be written as  $a_n = cr^{n-1}$

### Example 2

1, 2, 4, 8, 16, ... and  $\frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \frac{1}{16}, \dots$  are examples of geometric sequences.



The concepts of arithmetic and geometric sequences are related to the concepts of the *arithmetic* and *geometric mean* (see Questions 7 and 8). The arithmetic mean (the average) of two numbers  $a$  and  $b$  is defined as  $\frac{a+b}{2}$ . The geometric mean of two positive numbers  $a$  and  $b$  is defined as  $\sqrt{ab}$ . Figure 4-1 gives a neat geometric interpretation of the arithmetic and geometric mean and demonstrates that the geometric mean never exceeds the arithmetic mean.



$$R = \frac{a+b}{2}; \quad h = \sqrt{ab}$$

**Figure 4-1. Arithmetic and geometric mean**

### Exercises

1. Come up with your own “interesting” sequence, defined in words. State its  $n$ -th term.

2. Determine a formula for the general term of the sequence  
 $\frac{1}{2}, \frac{1}{6}, \frac{1}{12}, \frac{1}{20}, \frac{1}{30}, \frac{1}{42}, \dots$  ✓
3. Show that if  $a_0, a_1, a_2, \dots$  is an arithmetic sequence, then  $a_0, a_3, a_6, a_9, \dots$  is also an arithmetic sequence.
4. Suppose the first term of an arithmetic sequence is 3 and the 7th term is 21. Find the 12th term. ✓
5. What is the common ratio of the geometric sequence 4, 12, 36, ...? What is its general term?
6. Suppose the first term of a geometric sequence is 1 and the 11th term is 1024. Find the 21st term.
7. Show that in an arithmetic sequence, each term (except the first) is the arithmetic mean of its left and right neighbors, that is,  $a_n = \frac{a_{n-1} + a_{n+1}}{2}$ .
8. Show that in a geometric sequence with positive terms, each term (except the first) is the geometric mean of its left and right neighbors, that is,  
 $a_n = \sqrt{a_{n-1}a_{n+1}}$ .
9. Are there any sequences that are arithmetic and geometric at the same time? ✓
10. What is the limit of the sequence  $0, \frac{1}{2}, \frac{2}{3}, \frac{3}{4}, \frac{4}{5}, \dots$ ? ✓
11. ■ Does the sequence  $\left\{ a_n = \frac{n-2}{n-5} \right\}$  converge? If so, what is the limit?
12. ♦ What is the limit of the sequence  $\left\{ a_n = \frac{n}{\sqrt{n^2-1}} \right\}$ ?

## 4.3 Sums

In many mathematical situations we are interested in the sum of the first  $n$  terms of a sequence. We have already seen in Chapter 1 that  $1 + 2 + \dots + n = \frac{n(n+1)}{2}$ . A more interesting example is  $1^3 + 2^3 + 3^3 + \dots + n^3$ . With a little algebraic technique it is fairly easy to show that this sum is equal to  $\left(\frac{n(n+1)}{2}\right)^2$ . In other words,  $1^3 + 2^3 + 3^3 + \dots + n^3 = (1 + 2 + 3 + \dots + n)^2$ . So it turns out that the sum of the first  $n$  cubes is always a perfect square.



There is special “sigma” notation for sums:  $a_1 + a_2 + \dots + a_n$  is often written as  $\sum_{i=1}^n a_i$ .

$\Sigma$  is the letter *sigma* of the Greek alphabet, written in the upper case.  $i$  is a *variable* — you can use  $j$ ,  $k$ , or any other symbol. So we can write the sum of the first ten cubes as  $\sum_{k=1}^{10} k^3$ . The above identity that involves the sum of the first  $n$  cubes can be

restated as  $\sum_{k=1}^n k^3 = \left(\sum_{k=1}^n k\right)^2$ .

### Example 1

#### The sum of a geometric sequence

Let’s start with a simple geometric sequence:  $1, 2, 4, \dots, 2^{n-1}, \dots$ . We want to find the sum of the first  $n$  terms of this sequence:

$$s_n = 1 + 2 + 4 + \dots + 2^{n-1}$$

If we multiply both sides by 2, we get:

$$2s_n = 2 + 4 + 8 + \dots + 2^{n-1} + 2^n$$

— a very similar sum, with almost the same terms, except the first term is missing and  $2^n$  is added on the right. If we subtract the first sum from the second, the same terms will cancel out, and we will be left with  $2s_n - s_n = s_n = 2^n - 1$ .

↩ There is another way to prove that  $1 + 2 + 4 + \dots + 2^{n-1} = 2^n - 1$ . We know that the formula works for  $n = 0$ :  $1 = 1$ . Let's assume that for any given  $n$ , the formula works for  $n - 1$ , that is  $s_{n-1} = 2^{n-1} - 1$ . Then

$$s_n = s_{n-1} + 2^{n-1} = (2^{n-1} - 1) + 2^{n-1} = 2 \cdot 2^{n-1} - 1 = 2^n - 1$$

so the formula works for  $n$ , too. From this we can conclude that the formula works for any  $n \geq 1$ . This method of proof is called *mathematical induction*; more on it in

↑ Chapter 11.

## Exercises

- In Chapter 1, we showed that the sum  $1 + 2 + \dots + n$  is equal to  $\frac{n(n+1)}{2}$ , which is the average of the first and last terms,  $\frac{1+n}{2}$ , times the number of terms  $n$ . Show that the same is true for any arithmetic sequence.
- Derive the formula for the sum of the first  $n$  terms of the arithmetic sequence  $\sum_{i=1}^n [a_1 + (i-1)d]$  in a different way, by reducing it to the known sum  $1 + 2 + \dots + (n-1)$ . After simplification, your result should be the same as in Question 1.  $\Leftarrow$  Hint:  $\sum_{i=1}^n c = nc$ ;  $\sum_{i=1}^n [(i-1)d] = d \sum_{i=1}^n (i-1)$ .  $\Rightarrow$
- Find the sum of the the first  $n$  terms of the *telescopic* sequence  $\frac{1}{1 \cdot 2} + \frac{1}{2 \cdot 3} + \frac{1}{3 \cdot 4} + \dots + \frac{1}{n \cdot (n+1)}$ .  $\Leftarrow$  Hint:  $\frac{1}{7 \cdot 8} = \frac{1}{7} - \frac{1}{8}$ , for example.  $\Rightarrow \checkmark$

4. Show graphically that  $\frac{1}{2} + \frac{1}{4} + \dots + \frac{1}{2^n} = 1 - \frac{1}{2^n}$ .  $\leq$  Hint: take a square pizza, 1 by 1 foot. Cut it in half. Cut one of the pieces in half. Cut one of the new pieces in half. Keep cutting...  $\ni$
5.  $\blacksquare$  Derive a formula for  $s_n = c + cr + cr^2 + \dots + cr^{n-1}$ . Verify the result by applying it to the geometric sequence from Question 4.  $\leq$  Hint: recall what we did with  $1 + 2 + 4 + \dots + 2^n$  and use a similar method.  $\ni$
6.  $\blacksquare$  Find  $\sum_{d=1}^6 d \cdot 10^d$ .  $\checkmark$
7.  $\blacksquare$  Find  $\sum_{k=1}^n (2^k - k)$ .
8.  $\blacklozenge$  Derive a formula for  $1^2 + 2^2 + \dots + n^2$ .  $\leq$  Hint: look for the formula in the form  $An^3 + Bn^2 + Cn + D$ ; plug in  $n = 0, n = 1, n = 2, n = 3$  to find  $A, B, C$ , and  $D$ .  $\ni$

## 4.4 Infinite Sums

An infinite sum? Can  $a_1 + a_2 + a_3 + \dots + a_n + \dots$  make any sense? Isn't such a "sum" always infinite? When you encounter infinity, it always raises infinitely many questions.

As it turns out, there is a way to give a precise mathematical meaning to an expression  $a_1 + a_2 + a_3 + \dots + a_n + \dots$ . Such an expression is called a *series*. The way to approach it is this: consider a sequence of sums  $s_1, s_2, \dots, s_n, \dots$ , where

$$s_1 = a_1$$

$$s_2 = a_1 + a_2$$

...

$$s_n = a_1 + \dots + a_n$$

...

Here  $s_n$  is a normal finite sum, for any particular  $n$ . It is called a *partial sum* of the series.

**If the sequence of partial sums converges to a number, that number is called the sum of the series. In that case, it is said that the series converges.**

You may still be not quite convinced: how can a sequence of partial sums ever converge? Aren't we adding more and more terms to the sum? But it is possible for the sums to converge if we add smaller and smaller amounts.

### Example 1

For the geometric series  $\frac{1}{2} + \frac{1}{4} + \dots + \frac{1}{2^n} + \dots$ ,  $s_n = \frac{1}{2} + \frac{1}{4} + \dots + \frac{1}{2^n} = 1 - \frac{1}{2^n}$ . As  $n$  increases,  $s_n$  approaches 1 (because  $\frac{1}{2^n}$  approaches 0). The partial sums never quite reach 1, but they get closer and closer to 1 as  $n$  increases, and the sum of the whole series is said to be 1.



A series is often written in “sigma” notation, as follows:  $\sum_{i=1}^{\infty} a_i$  or  $\sum_{k=0}^{\infty} b_k$ .  $\infty$  is the symbol used for infinity. So  $\sum_{i=1}^{\infty} \frac{1}{2^i} = 1$ .

**For a series to converge, its terms must be getting smaller and smaller, converging to 0.**

Is the *converse* true? That is, if the terms of a series get smaller and smaller, approaching 0, then does the series necessarily converge? It turns out this is not true.

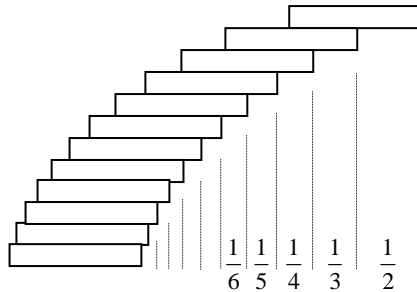
**Example 2**

The series  $1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \dots + \frac{1}{n} + \dots$  is called the *harmonic series*. Let us show that the harmonic series diverges. The idea is to split it into non-overlapping finite segments such that the sum of each segment exceeds a fixed number ( $\frac{1}{2}$ , for example). Here is the way to do it:

$$\begin{aligned} \frac{1}{2} &= \frac{1}{2} \\ \frac{1}{3} + \frac{1}{4} &> \frac{1}{4} + \frac{1}{4} = \frac{2}{4} = \frac{1}{2} \\ \frac{1}{5} + \frac{1}{6} + \frac{1}{7} + \frac{1}{8} &> \frac{1}{8} + \frac{1}{8} + \frac{1}{8} + \frac{1}{8} = \frac{4}{8} = \frac{1}{2} \\ \frac{1}{9} + \frac{1}{10} + \dots + \frac{1}{16} &> \underbrace{\frac{1}{16} + \frac{1}{16} + \dots + \frac{1}{16}}_{8 \text{ times}} = \frac{8}{16} = \frac{1}{2} \\ &\dots \end{aligned}$$

And so on. The segments get longer and longer, but we don't care: we have an infinite supply of terms to play with! So we can find a partial sum of this series that is greater than  $\underbrace{\frac{1}{2} + \frac{1}{2} + \dots + \frac{1}{2}}_{k \text{ times}}$ , for any  $k$ , and therefore the harmonic series cannot converge.

Figure 4-2 shows an odd construction based on the harmonic series. The tower rests on one brick and extends to the right as far as we want without any other support! You can keep adding bricks at the bottom, with the displacement  $\frac{1}{n}$ . It is not very hard to show that, for any  $n > 1$ , the center of gravity of the top  $n$  bricks falls on the border of the  $(n + 1)$ -th brick.



**Figure 4-2.** A tower of bricks with one support can extend to the right as far as we want

### Example 3

The series  $1 + \frac{1}{4} + \frac{1}{9} + \frac{1}{16} + \dots + \frac{1}{n^2} + \dots$  converges. Moreover, its sum is a surprise:  $\frac{\pi^2}{6}$ . It seems  $\pi$  pops up here out of the blue, but there is a deep mathematical connection.

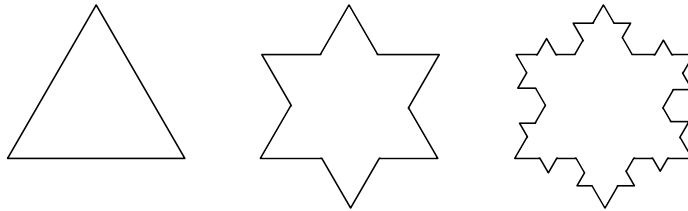



In general, the series is a fascinating topic that reveals many beautiful mathematical facts. Series are studied in Calculus, whose powerful methods help tell whether a particular series converges or not.

### Exercises

1. Find the sum of the *telescopic series*  $\frac{1}{1 \cdot 2} + \frac{1}{2 \cdot 3} + \frac{1}{3 \cdot 4} + \dots + \frac{1}{n \cdot (n+1)} + \dots$   
 $\Leftarrow$  Hint: see Question 3 in Section 4.3.  $\Rightarrow$
2. Does the series  $\sum_{n=1}^{\infty} \frac{n+1}{100n}$  converge? Explain. If yes, then what is its sum?  $\checkmark$

3. If  $\{d_0, d_1, d_2, \dots\}$  is the sequence of digits of  $\pi$ ,  $\{3, 1, 4, \dots\}$ , does  $\sum_{n=0}^{\infty} \frac{d_n}{10^n}$  converge? If yes, what is the sum?
4. ■ For which values of  $r$  does the geometric series  $1 + r + r^2 + \dots$  converge? ✓
5. Does the series  $1 + \frac{1}{3} + \frac{1}{5} + \dots + \frac{1}{2n-1} + \dots$  converge? Explain your reasoning.
6. ■ Does the series  $\sum_{n=1}^{\infty} \frac{1}{n^3}$  converge? Explain your reasoning.
7. ◆ The figure below illustrates the process of making a snowflake:



At each step, each straight line segment on the snowflake's perimeter is replaced by . If we keep repeating this process infinitely, we get a "curve" known as the *Koch Snowflake*. Suppose the perimeter of the first triangle is  $P$  and its area is  $A$  (where  $A = \frac{P^2\sqrt{3}}{36}$ ). Determine the perimeter and the area of the Koch snowflake after  $n$  iterations. Is the perimeter of the *Koch Snowflake* finite? What about the area?

## 4.5 Iterations in Python

In this section we will create a Python program that prompts the user to enter a positive integer `nMax` and prints out the sums  $1 + 2 + \dots + n$  for  $n$  from 1 to `nMax`. But first let us review what we have learned so far about Python's arithmetic operators.

Python supports `int` and `float` types of numbers. (It also supports the `complex` type, but we will leave this alone for now.) Python has `+`, `-`, `*`, and `/` operators. When `+`, `-`, and `*` are applied to two `ints`, the result is an `int` (as long as the result stays within the `int` range). If at least one of the operands is a `float`, the result is a `float`. The division operator `/`, when applied to two integers, gives a float.

Python has another division operator, `//`. When `//` is applied to two integers, `a` and `b`, the result is the largest integer that does not exceed  $\frac{a}{b}$ . For example, `15//2` gives 7, `-15//2` gives -8.

For your convenience, Python also has the *augmented assignment* operators `+=`, `-=`, `*=`, and so on. `a += b` is the same as `a = a + b`; `a -= b` is the same as `a = a - b`, and so on.



Our program should print  $n$  and the sum  $1 + 2 + \dots + n$  for all  $n$  from 1 to a certain number `nMax`, entered by the user. Since we don't know ahead of time what `nMax` will be, we can't just write

```
print('{0:3d} {1:6g}'.format(1, 1))
print('{0:3d} {1:6g}'.format(2, 3))
print('{0:3d} {1:6g}'.format(3, 6))
print('{0:3d} {1:6g}'.format(4, 10))
...
```

It would also make little sense, since we would have to compute all the sums by hand. Even if we knew `nMax`, say `nMax = 1000`, and used a formula for the sum —

```
n = 1
print('{0:3d} {1:6g}'.format(n, n*(n+1)/2))
n = n+1
print('{0:3d} {1:6g}'.format(n, n*(n+1)/2))
n = n+1
print('{0:3d} {1:6g}'.format(n, n*(n+1)/2))
n = n+1
...
```

— our program would be too long.

Fortunately, Python provides an *iterative statement*, called a *while loop*, which allows us to repeat the same block of statements multiple times (but with different values of variables) while a certain condition holds true. The syntax for the `while` loop is:

```
while <condition>:
    ...
    ...
```

<condition> is an expression that can use *relational operators* `<`, `>`, `<=`, `>=`, `==` (is equal), and `!=` (is not equal). Its value is either `True` or `False`. Try it:

```
>>> 6 <= 6
True
>>> 5 <= 6
True
>>> 5 <= 4
False
```

<condition> can also include *logical operators*: `and`, `or`, `not`. For example:

```
>>> x = 4
>>> x > 0 and x <= 3
False
>>> not x < 0
True
```

As long as the condition remains true, the program repeats the statements in the `while` block (the statements that are indented under `while`). Such repetitions are called *iterations*. Usually the condition includes a test for a variable that is updated on each iteration, so eventually the condition becomes false and the iterations stop. The program then continues with the first statement after the `while` block.

## Example 1

```
nMax = 10
n = 1
while n <= nMax:
    print('{0:3d} {1:6g}'.format(n, n*(n+1)/2))
    n += 1
```

The output is:

1	1
2	3
3	6
4	10
5	15
6	21
7	28
8	36
9	45
10	55



Thanks to iterations, the program code has the same length regardless of whether we run it with `nMax = 10` or `nMax = 10000`.

## Example 2

Suppose we don't know the formula for the sum and want to use a “brute force” approach. We can calculate the sum  $1+2+\dots+n$  using a loop:

```
sumln = 0
i = 1
while i <= n:
    sumln += i # same as sumln = sumln + i
    i += 1
```

This loop can be *nested* within the first loop:

```
nMax = 10
n = 1
while n <= nMax:
    sumln = 0
    i = 1
    while i <= n:
        sumln += i
        i += 1
    print('{0:3d} {1:6d}'.format(n, sumln))
    n += 1
```

**Reminder: avoid using the names of built-in functions (such as `sum`, `min`, `max`, `int`, `str`, `bytes`, `len`, `pow`, `list`, and `file`) for your variables.**



Our programs are getting bigger, and it is no longer practical to test them in an interactive environment. It is much better to create a program with a text editor of some kind (for example, *Notepad*) and save it in a file. You can even use a word processor such as *MS Word* — just make sure you save the file as “text only.” Or you can use an *IDE* (*I*ntegrated *D*evelopment *E*nvironment).

**It is customary to give Python *source files* (that is, files that hold the texts of Python programs) the extension `.py`.**

IDLE, included with the Python release, provides a simple editor that allows you to enter and edit a program, save it in a file and conveniently run it. Let’s try it. In IDLE, press `Ctrl-N` to open a new editor window (or use the menu command `File/New Window`). Type in the little program above and save it in a source file `Sums1toN.py` (Press `Ctrl-S` or use the menu command `File/Save`). Put the file in a folder of your choice, for example `C:\mywork`. Press `F5` to interpret and run the program. If you make changes to your program and press `F5` again, IDLE will prompt you to save the changed file.

You can have several editor windows open, and you can cut and paste text from one to another.

See [www.skylit.com/python/Appendix-A.html](http://www.skylit.com/python/Appendix-A.html) for details and for instructions on how to run Python programs in other ways.



Your program works, but it is inefficient: it keeps recalculating sums from scratch, over and over. If we have found  $1+2+\dots+49$ , we only need to add 50 to it to get  $1+2+\dots+50$ .

### Example 3

We can keep track of the sum and eliminate the nested `while` loop, like this:

```
nMax = 10
n = 1
sumln = 0
while n <= nMax:
    sumln += n
    print('{0:3d} {1:6d}'.format(n, sumln))
    n += 1
```



You might say: Who cares? Computers are fast anyway. However, such inefficiencies can visibly slow down even the fastest computer if the task is large enough. Here the version with nested loops will perform a number of operations proportional to  $nMax^2$ , while in the streamlined version, with one loop, the number of operations will be proportional to  $nMax$  — a big difference. In this case, the more efficient version also has shorter code.



We are almost done with our project. The only remaining question is: How do we get `nMax` from the user?

Python has a built-in function `input`, which lets you issue a prompt and read a string from the *standard input stream* `stdin`, that is, the keyboard.

### Example 4

```
>>> word = input('Enter any word: ')
Enter any word: Hello
>>> word
'Hello'
```



`input` returns a string. We can convert it into an `int` by applying the built-in function `int`.

## Example 5

```
>>> s = input('Enter a positive integer: ')
Enter a positive integer: 10
>>> nMax = int(s)
>>> nMax
10
```



This works, as long as the user enters a string that represents a valid integer. But what if the user mistypes? For example:

```
>>> s = input('Enter a positive integer: ')
Enter a positive integer: 1.0
>>> nMax = int(s)
...
ValueError: invalid literal for int() with base 10: '1.0'
```

The program is aborted with a cryptic error message. In the technical Python lingo, the program *raises an exception* — in this case, a `ValueError` exception. This is not very friendly! We should really give the user another chance. Luckily, Python lets us convert the entered string into an `int` tentatively and “catch” the exception if it occurs.

## Example 6

```
try:
    nMax = int(s)
except ValueError:
    print('Invalid input')
```

We will put this code inside a `while` loop to give the user as many tries as he wants:

```
nMax = -1
while nMax <= 0:
    s = input('Enter a positive integer: ')
    try:
        nMax = int(s)
    except ValueError:
        print('Invalid input')
```

We have initially set `nMax` to `-1`, so that the `while` loop executes the first time.

Look at the complete text of this program — it is in the `Sums1toN.py` file in the `Py\Ch04` folder on the Student Disk.



↓ We mentioned earlier that Python has a built-in function `sum`. To calculate  $1+2+\dots+n$  we could simply write `sum(range(1, n+1))`. But it would be inefficient to use this call within a loop (the same problem as with nested loops). It would also have spoiled all our fun!

## Exercises

1. Modify the `Sums1toN.py` program to accept only an odd positive integer `nMax` from the user and print the sums of positive odd integers  $1+3+\dots+n$  for  $n = 1, 3, \dots, nMax$ .
2. Write a program that prompts the user for a positive integer  $n$  and prints all positive multiples of 6 (6, 12, 18, etc.) that do not exceed  $n$ .
3. Using `Sums1toN.py` as a prototype, write a program that prompts the user to enter a positive integer `nMax` and displays  $n$  and  $n!$  ( $n$ -factorial) for  $n$  from 1 to `nMax`.  $n! = 1 \cdot 2 \cdot \dots \cdot n$ . ✓
4. ■ Write a program that prints  $n$ ,  $s_1(n) = \sum_{k=1}^n k$ ,  $s_2(n) = \sum_{k=1}^n k^2$ , and  $\frac{3s_2(n)}{s_1(n)}$  for  $n = 1, 2, 3, \dots, 20$ . Try to guess the general formula for  $\sum_{k=1}^n k^2$  from the resulting table.
5. ■ Write a function `printSquare(n)` that displays a “square” whose side has  $n$  stars. For example, for  $n = 5$ , the output should be:

```
*****
*     *
*     *
*     *
*     *
*****
```

Use only one `while` loop. (Actually, in Python, you can write this function without any loops, on one line. Can you figure out how?) ✓

6. Write a function `myPow(x, n)` that returns  $x^n$ , where  $x > 0$  and  $n$  is a non-negative integer. Do not use the `**` operator or the `math.pow` function — use one `while` loop.  $\Leftarrow$  Hint:  $x^0 = 1$ .  $\Rightarrow$   $\checkmark$
7.  $\blacksquare$  Write a function `smallestDivisor` that takes an integer  $n > 1$  and returns its smallest divisor that is greater than 1. For example: `smallestDivisor(15)` should return 3; `smallestDivisor(7)` should return 7.  $\Leftarrow$  Hint:  $d$  is not a divisor of  $n$  if and only if  $n \% d \neq 0$ .  $\Rightarrow$
8.  $\blacklozenge$  A positive integer is called a *perfect number* if it is equal to the sum of all of its divisors, including 1 but excluding the number itself. For example,  $6 = 1 + 2 + 3$ . Write a function `sumOfDivisors` that takes a positive integer  $n$  and returns the sum of all its divisors (excluding  $n$ ).  $\Leftarrow$  Hint:

```

        if n % d == 0:
            sumDivs += d
    
```

The smallest perfect number is 6. Write a program that finds and prints out the next perfect number.

9.  $\blacklozenge$  Using `Sums1toN.py` as a prototype, write a program that prompts the user to enter a positive integer `nMax` and displays the partial sums of the series  $1 + \frac{1}{4} + \frac{1}{9} + \frac{1}{16} + \dots + \frac{1}{n^2} + \dots$  for  $n$  from 1 to `nMax`. For each  $n$ , display  $n$ , the corresponding sum, and  $\sqrt{6 \cdot \text{sum}}$ . Test your program with a wide range of user input including a non-number, a negative integer, and a valid positive integer.

$\Leftarrow$  Hints:

1. Don't forget `from math import sqrt`. After that you can call `sqrt(x)` — it returns  $\sqrt{x}$ .
2. The statement

```
print('{0:3d} {1:8.6f} {2:8.6f}'.format(n, sum, p))
```

will print all three numbers in an appropriate format.

$\Rightarrow$

## 4.6 Review

Terms and notation introduced in this chapter:

<i>Sequence</i>	<i>Series</i>	$\sum_{k=1}^n a_k$
<i>Term (of a sequence)</i>	<i>Partial sum</i>	
<i>Converging sequence</i>	<i>Converging series</i>	$\sum_{k=1}^{\infty} a_k$
<i>Limit of a sequence</i>	<i>Diverging series</i>	
<i>Arithmetic sequence</i>	<i>Augmented assignment operators</i>	
<i>Geometric sequence</i>		
<i>Sigma notation</i>		

Some of the Python features introduced in this chapter:

```
from math import sqrt

+=, -=, *=, etc.

while <some condition is true>:
    ...
    ...

s = input('Enter ... ')

try:
    n = int(s)
except ValueError:
    print('Invalid input...')

print('n={0:3d} x={1:7.2g}'.format(n, x))
```