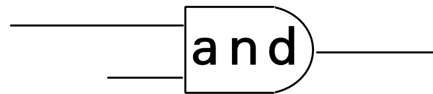
Mathematics
for the Digital Age



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
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

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

5 Number Systems

5.1 Prologue

What is 5, mathematically speaking? There are many possible answers. Five is how many fingers I have on my right hand. Five is what follows four. And five is the property that all sets of "five" objects have in common. But the last definition seems to be circular: How can we tell that two sets have the same number of elements without counting? And to count, you already need to know one, two, three, four, five...

It turns out, however, there is a simple method of comparing the numbers of elements in two sets without counting. If we can establish a correspondence between the two sets, such that each element of the first set is paired up with exactly one element of the second set and vice-versa, then we can be sure the two sets have the same number of elements. Suppose, for example, that there are several students and several backpacks on the playground. Ask each student to pick up one backpack. If there are not enough backpacks for all the students, the set of students has more elements, and if there are some backpacks left on the ground, the set of backpacks has more elements. If every student gets one backpack, and there are no backpacks left, then the number of students and the number of backpacks are the same. Another example: if you can touch with each finger of one hand a different object in a set, and no objects remain untouched, then the set has five elements.

It would be nice to create one special set of five different objects, so that we could compare every other set to it. Suppose the world is populated only by sets, as in pure set theory. How can we devise five different objects in such a world? Take the empty set $\varnothing$ — call it 0. Now take a set of one element: $\{\varnothing\}$ — call it 1. Now take the set that contains 0 and 1 as elements: $\{\varnothing, \{\varnothing\}\}$ — call it 2. $\{\varnothing, \{\varnothing\}, \{\varnothing, \{\varnothing\}\}\}$ is 3. And so on.

Copyright © 2008-2010 by Skylight Publishing

5.2 Positional Number Systems

Once we have figured out, more or less, what a number is, the question remains how to represent different numbers. What is 34, for example? This combination of two digits represents a crucial invention of human civilization, a *positional number system*. Without it we would still be writing 34 as XXXIV or something like that. Whereas in our system it is easy to see: 34 is three times ten plus 4. Ten is a magic number, the *base* of our positional system. We probably picked ten simply because humans have ten fingers. If you need to tell a stranger who doesn't speak your language that you are willing to swap your iPod for 34 apples, it is convenient to gesture three times with both hands open and then once more with four outstretched fingers. So the importance of ten is a human universal, constant across languages and cultures.

If you have a number $\underbrace{d_n ... d_2 d_1 d_0}_{\text{decimal digits}}$, where d_i is the *i*-th digit, then that number is equal to $d_n \cdot 10^n + ... + d_2 \cdot 10^2 + d_1 \cdot 10 + d_0$. For example, $347 = 3 \cdot 10^2 + 4 \cdot 10 + 7$.

Now imagine what would happen if we humans had only three fingers. We would count 1, 2, ... and then what? Three would become our magic number! After 2 we would write 10, 11, 12, then 20, 21, 22, and then... 100. 21_3 (21 base 3) would mean $2 \cdot 3 + 1 = 7_{10}$ (7 base 10). 14 would become 112, and instead of 34 we would write 1021 ($1021_3 = 1 \cdot 3^3 + 0 \cdot 3^2 + 2 \cdot 3 + 1 = 34_{10}$).

The "base 3" system would have its advantages. For one, there would be fewer digits to learn, and it would be easier to learn how to count: one, two, ten, eleven, twelve, twenty, and so on. And you could live to celebrate your 100th birthday (our age 9), 1000th birthday (our age 27), and even 10000th birthday (our age 81). On the other hand, your phone number, instead of 10 digits, would need 20 or 21 — hard to remember.

We can do arithmetic on numbers written in base 3 (or any other base) in the same manner as we do arithmetic in base 10.

Copyright © 2008-2010 by Skylight Publishing

Example 1

Calculate $2211_3 + 102_3$

Solution

```
  2211
+  102
 -----
 10020
```

$1 + 2\ (= 3_{10}) = 10_3 \Rightarrow$ write 0, carry 1.
$1 + 0 + 1\ (= 2_{10}) = 2_3 \Rightarrow$ write 2, carry 0.
$2 + 1 + 0\ (= 3_{10}) = 10_3 \Rightarrow$ write 0, carry 1.
$2 + 0 + 1\ (= 3_{10}) = 10_3 \Rightarrow$ write 0, carry 1.
Write 1.

Example 2

Calculate $2 \cdot 1423_5$

Solution

```
  1423
×    2
 -----
  3401
```

$3 \cdot 2\ (= 6_{10}) = 11_5 \Rightarrow$ write 1, carry 1.
$2 \cdot 2 + 1\ (= 5_{10}) = 10_5 \Rightarrow$ write 0, carry 1.
$2 \cdot 4 + 1\ (= 9_{10}) = 14_5 \Rightarrow$ write 4, carry 1.
$2 \cdot 1 + 1\ (= 3_{10}) = 3_5 \Rightarrow$ write 3.

Exercises

1. Find all two-digit numbers (base 10) such that the number is equal to twice the sum of its digits. Find all two-digit numbers such that the number is equal to 7 times the sum of its digits. ✓

2. Take any number and subtract from it the sum of all its digits (base 10). Show that that difference is always evenly divisible by 9.

Copyright © 2008-2010 by Skylight Publishing

3.■ In a Sudoku puzzle you need to fill a 9 by 9 grid with numbers in such a way that each row, each column, and each of the nine 3 by 3 squares contains all the numbers from 1 to 9. Take any completed Sudoku grid and combine the first three columns into one column of 3-digit numbers. Do the same for the next three columns and the last three columns. Prove that the sum of the numbers in all three new columns is the same. What is the value of that sum?

4. Write 15_{10} and 24_{10} in base 3. ✓

5. Write 121_3 and 2020_3 in base 10.

6. Write 10011100_2 in base 10. ✓

7. Subtract 1 from 10000_3 and write the answer in base 3.

8.■ Write 243_5 in base 7.

9. Calculate $201_3 + 12_3$ and $201_3 - 12_3$ without converting the numbers into decimal representation. ✓

10.■ How can you quickly tell whether a number, written in base 3, is even (that is, evenly divisible by 2)?

11.◆ Take a 3 by 3 magic square:

```
8 1 6
3 5 7
4 9 2
```

It contains all numbers from 1 to 9, and the sum of the numbers in each row, each column, and each of the two diagonals is the same. Subtract 1 from each number. Convert each number into a two-digit base-3 number (adding a leading zero where necessary). In the resulting grid, each row has 0, 1, 2 as first digits and as second digits, in some order. The same for columns. All nine combinations of two digits are different. Such an arrangement is known as *Greco-Roman square*. (The original design used the Greek letters α, β, γ instead of 0, 1, 2 as the first digits and the Latin letters *A*, *B*, *C* as second digits. Each row and each column would have all three Greek letters and all three Latin letters, and all nine combinations would be different.) Construct a 5 by 5 Greco-Roman square and then a 5 by 5 magic square.

Copyright © 2008-2010 by Skylight Publishing

5.3 The Binary, Octal, and Hexadecimal Systems

The next step, of course, is to pretend that we don't have any fingers at all, just two hands. Then we would have only two digits, 0 and 1. This base 2 system is called the *binary system*. In the binary system, non-negative integers are represented like this:

Base 10:	Binary:
0	0
1	1
2	10
3	11
4	100
5	101
6	110
7	111
8	1000
...	...

It is very convenient to use binary representation of numbers in computers: as we know, each bit in the computer memory can represent only two values, 0 and 1. However, binary numbers are hard to work with for humans because even small numbers are far too long in binary representation. For example, $98_{10} = 1100010_2$. We need a compromise: a system that is easy to convert to binary and also easy to read for humans (at least for programmers).

There are two such systems: *octal* and *hexadecimal*. The octal system uses base 8 and has eight digits: 0 - 7. Each of these digits can be represented by a three-digit binary number (with leading zeros):

```
0        000
1        001
2        010
3        011
4        100
5        101
6        110
7        111
```

To convert an octal number into binary, simply replace each octal digit with its 3-digit binary representation. Experienced programmers know the bit patterns for the octal digits and can translate them instantaneously.

Copyright © 2008-2010 by Skylight Publishing

Example 1

3561_8 = 011 101 110 001_2 = 011101110001_2

The reverse conversion, from binary to octal, is also easy: split the string of binary digits into triplets, starting from the rightmost digit (add leading zeros on the left if necessary), then write the corresponding octal digits.

Example 2

11010111100_2 = 011 010 111 100_2 = 3274_8

The hexadecimal system uses base 16. Its 16 digits are 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, and F. A stands for 10, B for 11, etc.; F stands for 15. Each of these digits can be represented by a four-digit binary number, as follows:

```
0        0000
1        0001
2        0010
3        0011
4        0100
5        0101
6        0110
7        0111
8        1000
9        1001
A        1010
B        1011
C        1100
D        1101
E        1110
F        1111
```

To convert a "hex" (hexadecimal) number into binary, just replace each hex digit with its 4-digit binary representation. To convert back, split the string of binary digits into quads, starting from the right, and replace each quad with its hex equivalent. Again, experienced programmers remember the bit patterns for the hex digits and can do these conversions very quickly.

Copyright © 2008-2010 by Skylight Publishing

Example 3

$\texttt{B6F8}_{16}$ = $\texttt{1011 0110 1111 1000}_2$

$\texttt{0111 0110 1110 1010}_2$ = $\texttt{76EA}_{16}$

These days, octal numbers are used less and less; the hex system is far more popular, simply because you can conveniently represent the value of one byte (eight bits) as two hex digits.

To convert a binary number into base 10 manually, it is faster to first convert it into hex, then hex into base 10.

Example 4

$111010010_2 \;=\; \text{hex } 1\text{D}2 \;=\; 256 + 13 \cdot 16 + 2 \;=\; 466$

The same is true for converting decimal into binary: first convert the number into hex, then hex into binary.

Programming languages support octal and hex numbers. In Python, for example, a sequence of octal digits precedeed by 0o (zero and the letter 'o') represents a number base 8. Try it:

```
>>> 0o46 # octal digits
38
>>> -0o46
-38
```

A hex number is written with the `0x` prefix. You can use uppercase or lowercase letters for the hex digits:

```
>>> 0x1D2
466
>>> 0x1d2
466
```

Copyright © 2008-2010 by Skylight Publishing

A binary number is written using the 0b prefix, followed by binary digits.

Python has a way to convert a number into a string of hex, octal, or binary digits. Try:

```
>>> hex(466)
'0x1d2'
>>> '{0:x}'.format(466)
'1d2'
>>> '{0:04x}'.format(466)
'01d2'

>>> oct(38)
'0o46'
>>> print('{0:o}'.format(38))
46

>>> bin(38)
'0b100110'
```

You will often see hex numbers written with leading zeros, because hex numbers often represent the contents of a fixed-length group of bytes. In Python, an `int` value is usually represented in 4 bytes.

The built-in `int` function provides a way to convert a string of digits, written in any given base, into an `int` value. Try:

```
>>> x = int('46')   #default base 10
>>> x
46
>>> x = int('46', 8)
>>> x
38
>>> x = int('1d2', 16)
>>> x
466
>>> x = int('111010010', 2)
>>> x
466
```

Copyright © 2008-2010 by Skylight Publishing

As you can see, “binary” arithmetic is similar to regular arithmetic (Figure 5-1). To add two binary numbers, write them one underneath the other with the unit digits aligned. Add the unit digits. If the result is 2, subtract 2 and set the carry bit. Write the result. Add the 2s digits and the carry bit. If the result is 2 or 3, subtract 2 and set the carry bit. Write the result. And so on. Subtraction is similar.

```
 0 0 0 0 1 0 0 0 0 1 1 1  ←—— Carry bit
 1011010100011
+     11000111
 -------------
 1011101101010
```

Figure 5-1. Binary addition

Exercises

1. What is the largest integer that has 7 binary digits? 15 digits? ✓

2. Convert hex 90AB into binary.

3. Convert binary 01011101011 into hex. ✓

4. Convert hex F0C2 into octal.

5. What are the results, written in base 2, of the following operations? ✓

$1011100_2 * 4_{10}$ $1011100_2 / 4_{10}$

6.■ Write and test a function that counts and returns the number of bits (binary digits) set to 1 in a positive integer:

```
def countBits(n):
    'Returns the count of bits set to 1 in a positive integer'
    ...
```

For example, `countBits(12)` should return 2, because $12 = 0...01100_2$.

Hints:

1. Check the digits from right to left.
2. `n % 2` returns the rightmost (least significant) digit in the binary representation of `n`. (`a % b` is read "a modulo b," and means the remainder when `a` is divided by `b`.)
3. `n //= 2` divides `n` by 2 and truncates the result to an integer, in effect shifting the binary digits to the right by one and getting rid of the least significant digit.

Copyright © 2008-2010 by Skylight Publishing

7.◆ Write a program that prompts the user to enter a non-negative integer, converts it into a string of binary digits, and prints out the string. Pretend that the built-in function `bin` does not exist and implement the conversion in a separate function:

```
# Returns a string that represents a given non-negative
# integer n as a string of binary digits.
# Precondition: n is an int; n >= 0
def intToBin(n):
    '''Converts n >= 0 into a string of binary digits
    and returns that string'''
    ...
```

Hints:

1. You might want to get rid of the case `n == 0` right away:

```
if n == 0:
    return '0'
```

2. Go from right to left and append subsequent digits <u>at the front</u> of the result string:

```
s = str(lastDigit) + s
```

3. Convert the string of binary digits back into an `int` (using the `str` function) and print it out to test your `intToBin` function:

```
n = ...
s = intToBin(n)
print(int(s, 2))
```

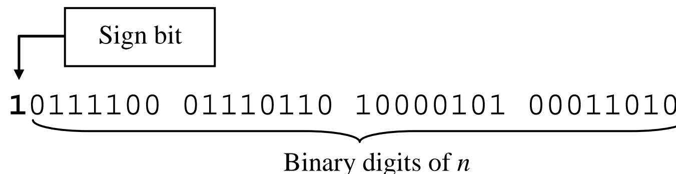
4. Adapt the code from `Py\Ch04\Sums1toN.py` to prompt the user for a non-negative integer.

Copyright © 2008-2010 by Skylight Publishing

5.4 Representation of Numbers in Computers

In Python, `int` and `float` values are represented in a manner convenient for popular microprocessors.

An `int` usually takes 32 bits (4 bytes). If we used all 32 bits for representing unsigned (non-negative) integers in binary, we would be able to represent all the numbers in the range from 0 to $2^{32} - 1$. Python interprets only the numbers in the range from 0 to $2^{31} - 1$ as non-negative; for these numbers, the most significant bit is 0. The numbers in the range from 2^{31} to $2^{32} - 1$ (that is, the numbers with the most significant bit set to 1) are interpreted as negative. The representation of negative numbers is called *two's complement*: unsigned n is interpreted as $n - 2^{32}$. In the two's complement system, `11....111` (thirty-two 1s) is −1; `11....110` is −2; and so on. The most significant bit becomes an indicator of sign (Figure 5-2).



If the sign bit is 0, the number is interpreted as n (non-negative).
If the sign bit is 1, the number is interpreted as $n - 2^{32}$ (negative).

Figure 5-2. Representation of integers in 32 bits. Negative numbers are represented in two's complement system.

The two's complement system is convenient because $\left(2^{32} - x\right) + x = 2^{32}$, which is outside of the 32-bit range; the most significant bit gets lost, and the result is 0. The CPU can perform arithmetic as if operating on 32-bit unsigned numbers — it is not even aware of what we interpret as negative numbers.

In Python, `hex(-1)` returns `'-0x1'` and `0xffffffff` is converted into a positive integer. In other languages (C, Java) `-1` corresponds to `'0xffffffff'`.

(In Chapter 7 you will write your own function that converts an `int` into a string of hex digits without the sign.)

Copyright © 2008-2010 by Skylight Publishing

A `float` usually takes 64 bits (8 bytes) and represents a number in the standard way expected by the CPU. Like scientific notation, this representation consists of a sign, a fractional part (*mantissa*) and an *exponent* part, but here both the mantissa and the exponent are represented as binary numbers. The IEEE (Institute of Electrical and Electronics Engineers) standard for an 8-byte (64-bit) representation uses 1 bit for the sign, 11 bits for the exponent and 52 bits for the mantissa. 1023 is added to the exponent to ensure that negative exponents are still represented by non-negative numbers.

This 8-byte format allows us to represent numbers in the range from approximately -1.8×10^{308} to 1.8×10^{308}, with about 17 decimal digits of precision. Note that you can represent in this format "only" 2^{64} different numbers, while there are infinitely many real numbers. Conversion from decimal to binary may introduce a small error.

There are many applets (little programs embedded into web pages) on the Internet that illustrate binary and hex representation of numbers and allow you to play with number conversions.

With a little work, we can also represent *rational numbers* in a computer. A rational number is a fraction with an integer numerator and denominator. As we said earlier, Python offers a way to introduce a new *class* of objects and redefine common operators for it. So we could define a class to represent fractions:

```
class Fraction(object):
    def __init__(self, num, denom):  # num is numerator
        self.num = num
        self.denom = denom
        reduce(self)
    def __str__(self):
        return '{0:d}/{1:d}'.format(self.num, self.denom)
    def __add__(self, other):
        return Fraction(self.num * other.denom +    \
                        other.num * self.denom, \
                        self.denom * other.denom)

    ... # etc. The code for the reduce function is not shown.
```

Copyright © 2008-2010 by Skylight Publishing

Then

```
>>> f1 = Fraction(1, 2)
>>> f2 = Fraction(1, 3)
>>> print('{0:s} + {1:s} = {2:s}'.format(f1, f2, f1 + f2))
```

would display

```
1/2 + 1/3 = 5/6
```

Exercises

1. If integers are represented in four bytes and negative numbers are represented in the two's complement system, what number does $\texttt{FFFFFACE}_{16}$ represent? ✓

2. Explain the result of

```
>>> 1000000000.0 + 0.0000000001
```

3. Explain the result of

```
>>> 3 * 0.1 == 0.3
```

4. Find the smallest positive integer *n*, such that in Python `int((17.0 / n) * n) != 17.0`. ✓

5. Python allows you to enter floating-point values in "calculator" notation, similar to scientific notation. For example, `1.23e5` represents 1.23×10^5 ; it will be displayed as `123000.0`. By default, Python also displays large `floats` in "calculator" notation. Find the smallest power of 10 and the smallest power of 0.1 that will be displayed in this way.

Copyright © 2008-2010 by Skylight Publishing

6.◆ Find the fraction with a denominator less than or equal to 20 that best approximates π.

⋞ Hints:

1.
```
from math import pi
```

2. The built-in function `round` rounds a `float` to the nearest `int`; the built-in function `abs` returns the absolute value of a number.

3.
```
if distance < bestDistance:
      bestDistance = distance
      bestNum = num
      bestDenom = denom
```
⋟

5.5 Irrational Numbers

Between integers and fractions, all the numbers should be covered, right? That's what Pythagoras and his disciples believed, about 2500 years ago. Around 550 BC, Pythagoras founded a small religious "brotherhood," made up of Greek immigrants, in the city of Croton on the shores of southern Italy. Pythagoreans made great contributions to geometry, and they were the first to study numeric relations in music: they figured out simple ratios of frequencies for pleasant combinations of musical sounds. They literally worshiped numbers. But, as it often happens, their own discoveries eventually challenged their beliefs.

Copyright © 2008-2010 by Skylight Publishing

Pythagoreans believed that any number is either a whole number or a *rational number* (that is, a *ratio*, a fraction with an integer numerator and denominator). Pythagoreans thought of numbers in terms of geometry, as ratios of lengths of segments. They knew how to find the hypotenuse of a right triangle with the given legs (using the Pythagorean theorem, of course). So, eventually, they had to ask: what is the ratio of the diagonal of a square to its side? From the Pythagorean theorem, if the side is *a*, and the diagonal is *d*, we should have $a^2 + a^2 = d^2$ or $\frac{d^2}{a^2} = \left(\frac{d}{a}\right)^2 = 2$. So what kind of number is $\frac{d}{a}$? Is it a rational number? (The legend has it that when one of the disciples proved that this ratio cannot be a rational number, he didn't end well: fellow Pythagoreans drowned him in a fountain for this "heresy.")

The first proof of irrationality of $\sqrt{2}$ most likely used geometry. Something like this. Suppose the side of a square is mx and the diagonal is nx where m and n are integers and x is a "unit segment," a "common measure" of a and d. Let's assume that m and n are the smallest possible such integers. However, we can construct a smaller square (Figure 5-3) with the side $(n-m)x$, and the diagonal $(2m-n)x$ which has the same property: $n-m$ and $2m-n$ are integers. But these integers are smaller than m and n, respectively. So the initial hypothesis was wrong. This method of proof is called *proof by contradiction*.

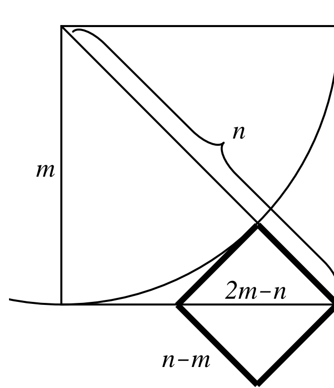


Figure 5-3. A geometric proof of irrationality of $\sqrt{2}$.

Here is a more modern, algebraic version of the proof. Suppose there exists a rational number $\frac{n}{m}$ such that $\frac{n^2}{m^2}=2$. Let's take the smallest such numbers m and n (by reducing the fraction). $n^2=2m^2$, so n must be an even number. (If n were odd, n^2 would be odd, too.) So $n=2k$, for some k. Then $n^2=4k^2 \Rightarrow$ $4k^2=2m^2 \Rightarrow 2k^2=m^2$. So m must be even, too. Since both m and n are even, we can reduce the fraction further and get a smaller integer numerator and denominator. This contradicts the assumption that m and n exist in the first place.

Copyright © 2008-2010 by Skylight Publishing

That's how the first *irrational number* was discovered. $a + b\sqrt{2}$ (with integer a and b), $\sqrt{3}$, and so on are all irrational, too. Eventually mathematicians proved that π is irrational. Much later, in the 19th century, a German mathematician Georg Cantor proved that there are "significantly more" irrational numbers than rational numbers. Both sets are infinite, of course. Cantor devised a way to compare infinite sets. In his theory, the set of rational numbers is "countable," that is, it belongs to the same category of infinity as the set of positive integers. The set of irrational numbers is in a different category of infinity: it is not countable.

Exercises

1.▪ Positive integers a, b, and c are said to form a *Pythagorean triple* if $a^2 + b^2 = c^2$. It is fairly easy to show that if we take any integers $0 < m < n$, then $a = n^2 - m^2$, $b = 2mn$, and $c = n^2 + m^2$ form a Pythagorean triple. For example, if we take m = 1, n = 2, we get a = 3, b = 4, c = 5. How many different pairs of integers $0 < m < n \le 7$ are there? Write a Python program that will generate Pythagorean triples for all such pairs. ✓

⋞ Hint:

Python supports objects called "tuples": a tuple is a list of items, separated by commas, within parentheses. A Python function can return a tuple. For example:

```
return (3, 4, 5)
```

You can print a tuple using the `print` function. For example:

```
t = (3, 4, 5)
print(t)
```

⋟

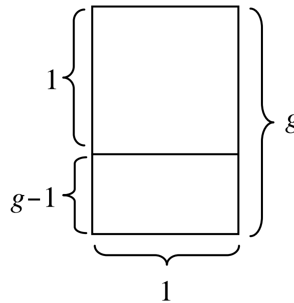
Copyright © 2008-2010 by Skylight Publishing

2.◆ If $a = \sqrt{2}$, then $a = \frac{2}{a}$. Let's start with $x = 1$ and repeatedly replace *x* with the arithmetic mean of *x* and $\frac{2}{x}$. If we keep repeating this operation, *x* will get closer and closer to $\sqrt{2}$. Write a program that uses this method to evaluate $\sqrt{2}$ accurately to at least 5 digits after the decimal point. This accuracy is achieved when $\left| x - \frac{2}{x} \right| < 0.00001$. Display the resulting estimate and the number of iterations that was required to obtain it. Compare your result with the value returned by `sqrt(2)`.

Hints:

1. `from math import sqrt`
2. `x = (x + 2 / x) / 2`
3. `abs(d)` returns the absolute value of `d`.

3.◆ The *golden ratio g* is defined by the proportion $\frac{g}{1} = \frac{1}{g-1}$, as shown in the picture below:



Solve the equation (see Question 7 in Section 3.5), then find the fraction with a denominator under 50 that best approximates the golden ratio.

Copyright © 2008-2010 by Skylight Publishing

5.6 Review

Terms and notation introduced in this chapter:

Positional number system
Base of a number system
Binary system
Octal system
Hexadecimal system
Two's complement representation of negative integers
Floating-point number
Rational number
Irrational number
Golden ratio

Some of the Python features introduced in this chapter:

```
0x12AB, 0x12ab

0o123

0b100110

int('12AB', 16)

hex(466)

print('{0:04x}'.format(466))

from math import pi

from math import sqrt

t = (3, 4, 5); print(t)

abs(d)
```

Copyright © 2008-2010 by Skylight Publishing