Chapter 1

Numbers

Defendit numerus: There is safety in numbers

We begin by talking about numbers. This may seen rather elementary but is does set the scene and introduce a lot of notation. In addition much of what follows is important in computing.

1.0.1 Integers

We begin by assuming you are familiar with the *integers*

$$1,2,3,4,\ldots,101,102,\ldots,n,\ldots,2^{32582657}-1,\ldots,$$

sometime called the whole numbers. These are just the numbers we use for counting. To these integers we add the *zero*, 0, defined as

$$0+ \text{ any integer } n = 0+n = n+0 = n$$

Once we have the integers and zero mathematicians create negative integers by defining $(-n)$ as:
the number which when added to n gives zero, so $n+(-n)=(-n)+n=0$.

Eventually we get fed up with writing $n+(-n)=0$ and write this as $n-n=0$. We have now got the positive and negative integers $\{\ldots,-3,-2,-1,0,1,2,3,4,\ldots\}$

You are probably used to arithmetic with integers which follows simple rules. To be on the safe side we itemize them, so for integers a and b

1. $a+b=b+a$
2. $a\times b=b\times a$ or $ab=ba$
3. $-a\times b=-ab$

4. $(-a) \times (-b) = ab$
5. To save space we write a^k as a shorthand for a multiplied by itself k times. So $3^4 = 3 \times 3 \times 3 \times 3$ and $2^{10} = 1024$. Note $a^n \times a^m = a^{n+m}$
6. Do note that n^0=1.

Factors and Primes

Many integers are products of smaller integers, for example $2 \times 3 \times 7 = 42$. Here 2, 3 and 7 are called the *factors* of 42 and the splitting of 42 into the individual components is known as *factorization*. This can be a difficult exercise for large integers, indeed it is so difficult that it is the basis of some methods in cryptography.

Of course not all integers have factors and those that do not, such as

$$3, 5, 7, 11, 13, \ldots, 2^{216091} - 1, \ldots$$

are known as *primes*. Primes have long fascinated mathematicians and others see

`http://primes.utm.edu/`,

and there is a considerable industry looking for primes and fast ways of factorizing integers.

To get much further we need to consider division, which for integers can be tricky since we may have a result which is not an integer. Division may give rise to a *remainder*, for example

$$9 = 2 \times 4 + 1.$$

and so if we try to divide 9 by 4 we have a remainder of 1 .

In general for any integers a and b

$$b = k \times a + r$$

where r is the *remainder.* If r is zero then we say a *divides* b written $a \mid b$. A single vertical bar is used to denote *divisibility.* For example $2 \mid 128$, $7 \mid 49$ but 3 does not divide 4, symbolically $3 \nmid 4$.

Aside

To find the factors of an integer we can just attempt division by primes i.e. $2, 3, 5, 7, 11, 19, \ldots$. If it is divisible by k then k is a factor and we try again. When we cannot divide by k we take the next prime and continue until we are left with a prime. So for example:

1. 2394/2=1197 can't divide by 2 again so try 3

2. 1197/3=399
3. 399/3 = 133 can't divide by 3 again so try 7 (not divisible by 5)
4. 133/7 = 19 which is prime so 2394 $= 2 \times 3 \times 3 \times 7 \times 19$

Modular arithmetic

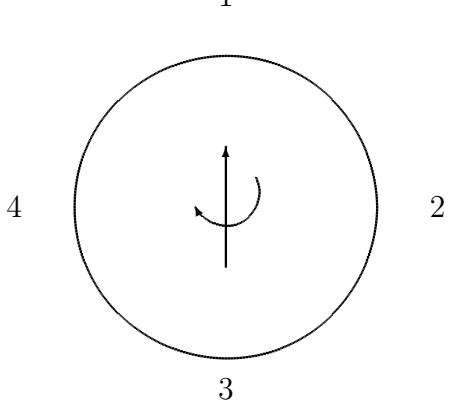
The `mod` operator you meet in computer languages simply gives the remainder after division. For example,

1. $25 \mod 4 = 1$ because $25 \div 4 = 6$ remainder 1.
2. $19 \mod 5 = 4$ since $19 = 3 \times 5 + 4$.
3. $24 \mod 5 = 4$.
4. $99 \mod 11 = 0$.

There are some complications when negative numbers are used, but we will ignore them. We also point out that you will often see these results written in a slightly different way i.e. $24 = 4 \mod 5$ or $21 = 0 \mod 7$. which just means $24 \mod 5 = 4$ and $21 \mod 7 = 0$

Modular arithmetic is sometimes called clock arithmetic. Suppose we take a 24 hour clock so 9 in the morning is 09.00 and 9 in the evening is 21.00. If I start a journey at 07.00 and it takes 25 hours then I will arrive at 08.00. We can think of this as 7+25 = 32 and 32 mod 24 = 8. All we are doing is starting at 7 and going around the (25 hour) clock face until we get to 8. I have always thought this is a complex example so take a simpler version.

Four people sit around a table and we label their positions 1 to 4. We have a pointer point to position 1 which we spin. Suppose it spins 11 and three quarters or 47 quarters. The it is pointing at $47 \mod 4$ or 3.



The Euclidean algorithm

Algorithms which are schemes for computing and we cannot resist putting one in at this point. The Euclidean algorithm for finding the gcd is one of the oldest algorithms known, it appeared in Euclid's Elements around 300 BC. It gives a way of finding the greatest common divisor (gcd) of two numbers. That is the largest number which will divide them both.

Our aim is to find a a way of finding the greatest common divisor, $\gcd(a, b)$ of two integers a and b.
Suppose a is an integer smaller than b.

1. Then to find the greatest common factor between a and b, divide b by a. If the remainder is zero, then b is a multiple of a and we are done.

2. If not, divide the divisor a by the remainder.

Continue this process, dividing the last divisor by the last remainder, until the remainder is zero. The last non-zero remainder is then the greatest common factor of the integers a and b.

The algorithm is illustrated by the following example. Consider 72 and 246. We have the following 4 steps:

1. $246 = 3 \times 72 + 30$ or $246 \mod 72 = 30$
2. $72 = 2 \times 30 + 12$ or $72 \mod 30 = 12$
3. $30 = 2 \times 12 + 6$ or $30 \mod 12 = 6$
4. $12 = 2 \times 6 + 0$

so the gcd is 6.

There are several websites that offer Java applications using this algorithm, we give a Python function

```
def gcd(a,b):
        """ the euclidean algorithm """
        if b == 0:
                return a
        else:
                return gcd(b, (a%b))
```

Those of you who would like to see a direct application of some these ideas to computing should look at the section on random numbers

1.0.2 Rationals and Reals

Of course life would be hard if we only had integers and it is a short step to the rationals or fractions. By a rational number we mean a number that can be written as P/Q where P and Q are integers. Examples are

$$\frac{1}{2} \quad \frac{3}{4} \quad \frac{7}{11} \quad \frac{7}{6}$$

These numbers arise in an obvious way, you can imagine a ruler divided into 'iths' and then we can measure a length in 'iths'. Mathematicians, of course, have more complicated definitions based on modular arithmetic . They would argue that for every integer n, excluding zero, there is an inverse, written $1/n$ which has the property that

$$n \times \frac{1}{n} = \frac{1}{n} \times n = 1$$

Of course multiplying $1/n$ by m gives a fraction m/n. These are often called *rational numbers*.

We can manage with the simple idea of fractions.

One problem we encounter is that there are numbers which are neither integers or rationals but something else. The Greeks were surprised and confused when it was demonstrated that $\sqrt{2}$ could not be written exactly as a fraction. Technically there are no integer values P and Q such that $P/Q = \sqrt{2}$.

From our point of view we will not need to delve much further into the details, especially as we can get good enough approximation using fractions. For example $22/7$ is a reasonable approximation for π while $355/113$ is better. You will find people refer to the *real numbers*, sometimes written $\mathbb{R}$, by which they mean all the numbers we have discussed to date.

Notation

As you will have realized by now there is a good deal of notation and we list some of the symbols and functions you may meet.

- If x is *less than* y then we write $x < y$. If there is a possibility that they might be equal then $x \leq y$. Of course we can write these the other way around. So $y > x$ or $y \geq x$. Obviously we can also say y is greater than x or greater than or equal to x

- The *floor function* of a real number x, denoted by $\lfloor x \rfloor$ or $\texttt{floor}(x)$, is a function that returns the largest integer less than or equal to x. So $\lfloor 2.7 \rfloor = 2$ and $\lfloor -3.6 \rfloor = -4$. The function `floor` in Java and Python performs this operation. There is an obvious(?) connection to mod since $b \mod a$ can be written $b - \texttt{floor}(b \div a) \times a$. So $25 \mod 4 = 25 - \lfloor 25/4 \rfloor \times 4 = 25 - 6 \times 4 = 1$

- A less used function is the *ceiling function*, written $\lceil x \rceil$ or `ceil`(x) or `ceiling`(x), is the function that returns the smallest integer *not less than* x. Hence $\lceil 2.7 \rceil = 3$.
- The *modulus* of x written $| x |$ is just x when $x \geq 0$ and $-x$ when $x < 0$. So $| 2 | = 2$ and $| -6 | = 6$. The famous result about the modulus is that for any x and y
$$| x + y | \leq | x | + | y |$$
- We met a^b when we discussed integers and in the same way we can have x^y when x and y are not integers. We discuss this in detail when we meet the exponential function. Note however
  - a^0=1 for all $a \neq 1$
  - $0^b = 0$ for all values of b including zero.

1.0.3 Number Systems

We are so used to working in a decimal system we forget that it is a recent invention and was a revolutionary idea. It is time we looked carefully at how we represent numbers. We normally use the decimal system so 3459 is shorthand for 3 x 1000 + 4 x 100 + 5 x 10 + 9. The *position* of the digit is vital as it enables us to distinguish between 30 and 3. The decimal system is a positional numeral system; it has positions for units, tens, hundreds and so on. The position of each digit implies the multiplier (a power of ten) to be used with that digit and each position has a value ten times that of the position to its right.

Notice we may save space by writing 1000 as 10^3 the 3 denoting the number of zeros. So $100000 = 10^5$. If the superscript is *negative* then we mean a fraction e.g $10^3 = 1/1000$. Perhaps the cleverest part of the positional system was the addition of the decimal point allowing us to include decimal fractions. Thus 123.456 is equivalent to

$$1 \times 100 + 2 \times 10 + 3 + \text{numbers after the point } + 4 \times 1/10 + 5 \times 1/100 + 6 \times 1/1000$$

Multiplier	...	10^2	10^1	10^0	.	10^{-1}	10^{-2}	10^{-3}	...
digits	...	1	2	3	.	4	5	6	...

↑
decimal point

However there is no real reason why we should use powers of 10, or base 10. The Babylonians use base 60 and base 12 was very common during the middle ages in Europe. Today the common number systems are

- Decimal number system: symbols 0-9; base 10
- Binary number system:symbols symbols 0,1; base 2
- Hexadecimal number system:symbols 0-9,A-F; base 16
  here A ≡ 10 , B ≡ 11 , C ≡ 12 , D ≡13 E ≡ 14 , F≡ 15.
- Octal number system: symbols 0-7; base 8

Binary

In the binary scale we express numbers in powers of 2 rather than the 10s of the decimal scale. For some numbers this is easy so, if recall $2^0 = 1$,

Decimal number		in powers of 2	power of 2				Binary number
			3	2	1	0	
8	=	2^3	1	0	0	0	1000
7	=	$2^2 + 2^1 + 2^0$	0	1	1	1	111
6	=	$2^2 + 2^1$	0	1	1	0	110
5	=	$2^2 + 2^0$	0	1	0	1	101
4	=	2^2	0	1	0	0	100
3	=	$2^1 + 2^0$	0	0	1	1	11
2	=	2^1	0	0	1	0	10
1	=	2^0	0	0	0	1	1

As in decimal we write this with the position of the digit representing the power, the first place after the decimal being the 2^0 position the next the 2^1 and so on. To convert a decimal number to binary we can use our `mod` operator.

As an example consider 88 in decimal or 88_{10}. We would like to write it as a binary. We take the number and successively divide `mod` 2. See below

Step number n	x_n	$\lfloor x_n/2 \rfloor$	x_n mod 2
0	88	44	0
1	44	22	0
2	22	11	0
3	11	5	1
4	5	2	1
5	2	1	0
6	1	0	1

Writing the last column *in reverse*, that is from the bottom up, we have 1011000 which is the binary for of 88, i.e.$88_{10} = 1011000_2$.

Binary decimals are less common but quite possible, thus 101.1011 is just $2^2+2^0+2^{-1}+2^{-3}+2^{-4}$ which is, after some calculation 5.6875. We have see how to turn the integer part of a decimal number into a binary number and we can do the same with a *decimal* fraction. Consider 0.6875. As before we draw up a table

Step number n	x_n	$x_n \times 2$	$\lfloor x_n \times 2 \rfloor$
0	0.6875	1.375	1
1	0.375	0.75	0
2	0.75	1.5	1
3	0.5	1	1

giving *reading down* $0.6875_{10} = 1011_2$

Beware it is possible to get into a non-ending cycle when we have a non terminating decimal. For example 0.4.

Step number n	x_n	$x_n 2$	$\lfloor x_n \times 2 \rfloor$	
0	0.4	0.8	0	
1	0.8	1.6	1	
2	0.6	1.2	1	
3	0.2	0.4	0	
4	0.4	0.8	0	← here we repeat
3	0.8	1.6	1	
4	...		...	

so $0.4_{10} = 0.0110011001100\ldots_2$

- Addition in binary
  - 0+0 = 0
  - 0+1 = 1
  - 1+1 = 10 so we carry 1 and leave a zero
  - 1+1+1 = 1+(1+0)=1+10=11 .

  We can write this in very much the same way as for a decimal addition

		1	1	0	1	0	1	
	+	1	0	1	1	1	0	
1		1	0	0	0	1	1	Sum
		↑			↑			

  the right hand uparrow show where we carry a 1. The left hand one shows where we have $1+1+1$ so we carry a 1 and have a 1 left over

- To subtract

	1	1	0	1	0	1	
-	1	0	1	1	1	0	
	0	0	0	1	1	1	difference

Multiplication in decimal

		1	2	5	6	7	8	Multiplicand
×					3	8	7	Multiplier
		8	7	9	7	4	6	times 7
1	0	0	5	4	2	4		Shift left one and times 8
3	7	7	0	3	4			Shift left two and times 3
4	8	6	3	7	3	8	6	Add to get product

Multiplication in binary

		1	0	0	1	1	1	0	Multiplicand
×						1	0	1	Multiplier
		1	0	0	1	1	1	0	times 1
	0	0	0	0	0	0	0		Shift left one and times 0
1	0	0	1	1	1	0			Shift left two and times 1
1	1	0	0	0	0	1	1	0	Add to get the product

As you can see multiplication in binary is easy.

Octal

Base 8 or octal does not bring any new problems. We use the symbols 0, 1, 2,. . . ,7 and the position denotes the power of 8. So 12_8 is $1 \times 8 + 2 = 10$ in decimal, while 3021_8 is

$$3 \times 8^3 + 0 \times 8^2 + 2 \times 8 + 1 \times 8^0 = 1536 + 16 + 1 = 1553$$

in decimal. Obviously we do not need the symbol for 9 as $9_{10} = 8 + 1 = 11_8$ in octal. To convert a decimal number to octal we can use our `mod` operator as we did in the binary case.

As an example consider 1553 in decimal or 1553_{10}. We would like to write it as an octal number. We take the number and successively divide `mod` 8. See below

Step number n	x_n	$\lfloor x_n/8 \rfloor$	$x_n \mod 8$
0	1553	194	1
1	194	24	2
2	24	3	0
3	3	0	3

Writing the last column *in reverse* we have 3021 which is the octal number we require since

$$3 \times 8^3 + 0 \times 8^2 + 2 \times 8 + 1 \times 8^0 = 1553$$

There is a simple link between octal and binary if we notice that

$7 = 2^2 + 2^1 + 2^0 = 111_2$	$3 = 2^1 + 2^0 = 11_2$
$6 = 2^2 + 2^1 = 110_2$	$2 = 2^1 = 10_2$
$5 = 2^2 + 2^0 = 101_2$	$1 = +2^1 = 1_2$
$4 = 2^2 = 100_2$	$0 = 0_2$

You might like to check that 1553 is

11000010001

in binary.

Separating this into blocks of 3 gives

11 000 010 001

If we use our table to write the digit corresponding to each binary block of 3 we have

3 0 2 1

which is our octal representation!

As in the binary case we can also have octal fractions, for example 0.3012_8. This is a way of representing

$$3 \times 1/8^1 + 0 \times 1/8^2 + 1 \times 1/8^3 + 2 \times 1/8^4$$

To convert 0.3012_8 to decimal we proceed as for the binary case only here we use 8 rather that 2 to give

Step number n	x_n	$8 \times x_n$	$\lfloor 8x_n \rfloor$
0	0.3012	2.4096	2
1	0.4096	3.2768	3
2	0.2768	2.2144	2
3	0.2144	1.7152	1
4	0.7152	5.72165	5
5	0.7216	5.7728	5
6	0.7728	6.1824	6
7	0.1824	1.4592	1
8	0.4592	3.6736	3
9	0.6736	5.3888	5
10	0.3888	3.1104	3
11.	0.1104002	0.8832016	0
12	0.8832016	7.0656128	7
13	0.06561279	0.52490234	0
14	0.5249023	4.1992188	4
15	0.1992188	1.5937500	1
16	0.59375	4.75000	4
17	0.75	6.00	6
18	0	0	

giving *reading down* $0.3012_8 = 0.232155613530704146_{10}$

hexadecimal

Base 16 is more complicated because we need more symbols. We have the integers 0 to 9 and we also use A $\equiv$ 10 , B $\equiv$ 11 , C $\equiv$ 12 , D $\equiv$13 E $\equiv$ 14 , F$\equiv$ 15.

So 123_{16} is $1 \times 16^2 + 2 \times 16^1 + 3$ in decimal and $A2E_{16}$ is $10 \times 16^2 + 2 \times 16^1 + 14$ in decimal. The good thing about hex is that each of the symbols corresponds to a 4 digit binary sequence (if we allow leading zeros). This means we can easily translate from hex to binary as below $01011110101101010010 2 =$ 0101 1110 1011 0101 0010

$$= 5 \quad E \quad B \quad 5 \quad 2_{16} = 5EB52_{16}$$

exercises

1. Factorize
   (a) 3096
   (b) 1234
   (c) $2^4 - 1$
2. It was thought that $2^p - 1$ was prime when p is a prime. Shown that this is not true when $p = 11$
3. Find the gcd for 3096 and 1234.
4. Write the following decimal numbers in binary
   (a) 256_{10}
   (b) $2^4 - 1$
   (c) 549
   (d) 12.34

5. Convert the following binary numbers into decimal numbers and explain your answers.

   (a) 101.001_2
   (b) 101111_2
   (c) 0.10101_2
   (d) 11.0001_2
   (e) 1001_2
   (f) 0.11_2

6. Convert the following decimal numbers into binary numbers and explain your answers.

   (a) 50_{10}
   (b) 70_{10}
   (c) 64_{10}
   (d) 39.56_{10}
   (e) 20.625_{10}
   (f) 13.11_{10} (8 significant digits)

7. Add the following numbers in binary and explain your answers.

   (a) $111_2 + 111_2$
   (b) $1110_2 + 11_2$
   (c) $11101_2 + 11001_2$

8. Multiply the following numbers in binary and explain your answers.

   (a) $1110_2 \times 11_2$
   (b) $111_2 \times 101_2$