maXbox

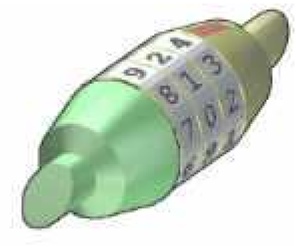
maXbox Starter 10

Start with Statistic Programming

1.1 Find the Probability

Today we spend time in programming with Statistics and in our case with probability. Statistic is a branch of applied mathematics concerned with the collection and interpretation of quantitative data and the use of probability theory to estimate population parameters.

A parameter is a numeric measurement that describes some property of the population for example mean or max height of a random sample from a population.
In this large field we deal with some basics today called probability. Probability is the study of chance or the likelihood of an event happening. Directly or indirectly, probability plays a role in all activities. For example, we may say that it will probably possible to unlock the lock because most of the combinations are too easy.



Our question will be to find out the probability to unlock a lock in 4 cases. All 4 examples discuss a different combination lock within math and security.

1. Permutation = n!
2. Permutation (Variation without repeating) = nPr = n!/(n-k)!
3. Combination (binominal coefficient) = nCr = nPr / k!
4. Variation (Permutation with repeating) = n^k

Permutation means arrangement of things. The word arrangement is used, if the order of things is considered. Combination means selection of things. The word selection is used, when the order of things has no importance.

So, in Math we use a more precise language:
If the order doesn't matter, it's a Combination.
If the order does matter it's a Permutation.

Hope you did already work with the Starter 1 to 9 available at:

http://www.softwareschule.ch/maxbox.htm

This lesson will introduce you to the probability of an event. The event in our case is to manipulate or hack the lock. If the probability of an event, A, is P(A), then the probability that the event would not occur (also called the complementary event) is 1 – P(A).
So the P(A) to unlock a lock with 10000 (10^4) possibilities is **1/10000**.

1.2 Visualize the Possibilities

As you already know the tool is split up into the toolbar across the top, the editor or code part in the centre and the output window at the bottom will show you the 4 cases with all possibilities.

Before this starter code will work you will need to download maXbox from the website. It can be down-loaded from http://sourceforge.net/projects/maxbox site. Once the download has finished, unzip the file, making sure that you preserve the folder structure as it is. If you double-click `maxbox3.exe` the box opens a default program. Make sure the version is at least 3.2 because the number cruncher will need that;). Test it with F9 or press **Compile** and you should hear a sound first and a browser will open. So far so good now we'll open the example:

`212_statisticmodule4.txt`

If you can't find the file use the link:

http://www.softwareschule.ch/download/212_statisticmodule4.txt

Or get it as a pdf - document:

http://www.softwareschule.ch/download/statisticmodule.pdf

Or you use the `Save Page as…` function of your browser[1] and load it from `examples` (or wherever you stored it). One important thing: You need some time to run the whole simulation, it depends on your CPU power means you have to wait about 30 seconds for a run. Now let's take a look at the code of this project first with the procedures set. Our first line is

```
08 Program Statistics_Module_4;
```

We have to name the program it's called `Statistics_Module_4.` Now we jump to line 55:

```
55 Procedure One_Permutation_4;
56 var
57  i,k,l,m: byte;
58  count: integer;
59 begin
60 count:= 0;
61 for i:= 0 to 3 do
62   for k:= 0 to 3 do
63     for l:= 0 to 3 do
64       for m:= 0 to 3 do begin
```

1.3 First Case "One_Permutation_4"

How many different 4-digit numbers can be formed from the digits 0, 1, 2, 3 where digits in each number is found exactly one times, means no repeating.
The lock has only 4 numbers with 4 elements, e.g.:

1	2	0	3

[1] Or copy & paste

The lock has a four number code consisting of four digits without repeating. How many possible code combinations are there in the Procedure on line 55?

Possible outcomes are the numbers 0, 1, 2, 3 and sample space, S = {0, 1, 2, 3}.

So, your first choice would have 4 possibilities, and your next choice would then have 3 possibilities, then 2, 1. And the total permutations would be:

$$4 \times 3 \times 2 \times 1 = 24$$

The above expression $4 \times 3 \times 2 \times 1 =$ can be written as 4!, which is read as “six factorial.” In general, N! is the product of all the counting numbers beginning with n and counting backwards to 1. We define 0! to be simply as 1.

☞ Our first lock looks like a normal lock but it has only 4 numbers [0..3] to set and you can’t set a number twice in one line, so 4114 is impossible anyway!

The maximum security would be 10! With the numbers {0..9} and with 10 elements built on the lock.

1.4 Second Case “Two_Permutation_4_of_10“

The second case is more realistic, you can set 10 numbers {0..9} but also again no repeating of the numbers in one line (or row), for e.g.:.

1	2	8	9

```
80 Procedure Two_Permutation_4_of_10;
81 var
82  i,k,l,m: byte;
83  count: integer;
84 begin
85 count:= 0;
86 for i:= 0 to 9 do
87   for k:= 0 to 9 do
88     for l:= 0 to 9 do
89       for m:= 0 to 9 do begin
```

Once the range from 0 to 9 is set, your application shows all the permutations in the output window, but how many are there? Our "order of 4 out of 10 numbers example" would be:

$$10! / (10-4)! = 5040$$

☞ This example requires 5 local variables and a filter to avoid repetition. Instead of writing the whole formula: nPr = n!/(n-k)!, people use different notations such as these:

$$P(n,r) = {}^{n}P_r = {}_{n}P_r = \frac{n!}{(n-r)!}$$

So you can build your own lock depends on the brain memory you have to find out a 7 digit code;) for example the lock has a 7 character code with 40 numbers and characters of the readable ASCII-Code:
40! / (40-7)! = 9.3963 E+10

So the maximum range on Delphi depends on your operating system types, means nowadays an int64 range is the max big int. With a Big Integer Library (bigintlib) you'll see the full range of Fact(40):

815915283247897734345611269596115894272000000000

Another way is to use the type `extended`, but the limitation is precision like

```
  Writeln(FloatToStr(Fact(40)))
```

You can test this Big Int Library with the script: `161_bigint_class.txt`

1.5 Third Case “Three_Combination_without_Order_4_of_20“

Next is a really amazing thing, you’ll see the first real digital lock just with the numbers of 0 and 1 and a hardware lock with 20 elements on it!

0	0	0	**1**	0	0	**1**	**1**	0	0	0	0	0	**1**	0	0	0	0	0	0

It’s a really long lock but thin, cause only 0 and 1 can be set.
An arrangement of numbers in which the order is not important is called a combination. This is different from permutation where the order matters. For example, suppose we are arranging the letters A, B and C. In a permutation, the arrangement ABC and ACB are different. But, in a combination, the arrangements ABC and ACB are the same because the order is not important.

Those are the so called binominal coefficients and on a calculator you find the nCr() button for example 4 of 20 = 4845 = NCR(20,4) or a lotto 4 of 20. Yes you may know this formula to compute all lotto combinations you had to mark a cross!
The number of combinations of n things taken r at a time is written as C(n, r). So the normal case is no repetition: such as lottery numbers (2,14,15,27,30,33)

The question now is the possible outcome in our lock.
That formula is so important it is often just written in big parentheses like this:

$$C(n,r) = {}^{n}C_r = {}_{n}C_r = \binom{n}{r} = \frac{n!}{r!(n-r)!}$$

☞The result of nCr(20,4) = 4845 and the probability to hack the code is 1/4845.

```
for i:= 0 to 1 do
  for k:= 0 to 1 do
    for l:= 0 to 1 do
       for m:= 0 to 1 do //…20 times
```

You can also use **Pascal's Triangle** to find the values. Go down to row "n" (the top row is 0), and then along "r" places and the value there is your answer. Here is an extract showing row 16:

```
      1   14   91   364  ...

    1   15   105   455   1365  ...

  1   16   120   560   1820   4368  ...
```

And that’s how to compute the output for each case in line 138:

```
138 Writeln(Format('Case: %d - %d%d%d%d%d%d%d%d%d%d%d%d%d%d%d%d%d%d%d%d',
                        [count,i,k,l,m,n,o,p,q,r,s,t,u,v,w,q1,r1,s1,t1,u1,v1]));
```

If you’re interested in the Pascal Triangle, here’s the script:
`172_pascal_triangleform2.TXT`

I got in this case a combination with 20 elements with a code combination of 4845 codes so you can compare this combination with the case 3 which has 5040 possible codes.
You can really see a pattern in the output too:

```
Case: 4829 - 11100000000000000001
Case: 4830 - 11100000000000000010
Case: 4831 - 11100000000000000100
Case: 4832 - 11100000000000001000
Case: 4833 - 11100000000000010000
Case: 4834 - 11100000000000100000
Case: 4835 - 11100000000001000000
Case: 4836 - 11100000000010000000
Case: 4837 - 11100000000100000000
Case: 4838 - 11100000001000000000
Case: 4839 - 11100000010000000000
Case: 4840 - 11100000100000000000
Case: 4841 - 11100001000000000000
Case: 4842 - 11100010000000000000
Case: 4843 - 11100100000000000000
Case: 4844 - 11101000000000000000
Case: 4845 - 11110000000000000000
```

1.6 Forth Case “Four_Variation_Lock_4“

So the last one is the easiest one to calculate and the normal case everywhere, it’s also called permutations with repetition or simply variation:

1	1	1	2

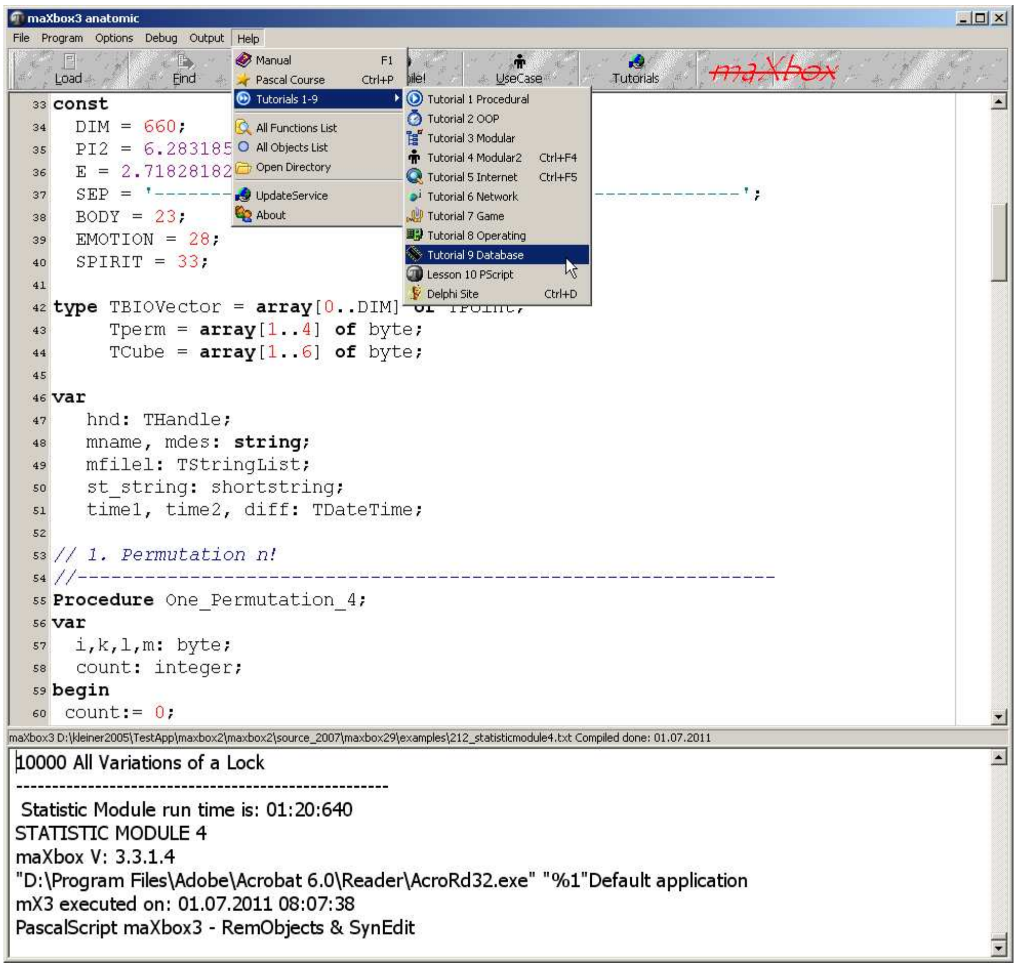
When you have n things to choose from ... you have n choices each time!

When choosing r of them, the permutations are:

$$n \times n \times ... (r \text{ times}) = n^r$$

```
for i:= 0 to 9 do
  for k:= 0 to 9 do
    for l:= 0 to 9 do
      for m:= 0 to 9 do begin
```

It’s almost the same like case 2 but without a filter so the numbers can repeat in one line.



```
const
  DIM = 660;
  PI2 = 6.283185
  E = 2.71828182
  SEP = '-------                                   -------------';
  BODY = 23;
  EMOTION = 28;
  SPIRIT = 33;

type TBIOVector = array[0..DIM] of TPoint;
     Tperm = array[1..4] of byte;
     TCube = array[1..6] of byte;

var
   hnd: THandle;
   mname, mdes: string;
   mfilel: TStringList;
   st_string: shortstring;
   time1, time2, diff: TDateTime;

// 1. Permutation n!
//---------------------------------------------------------------
Procedure One_Permutation_4;
var
  i,k,l,m: byte;
  count: integer;
begin
 count:= 0;
```

maXbox3 D:\kleiner2005\TestApp\maxbox2\maxbox2\source_2007\maxbox29\examples\212_statisticmodule4.txt Compiled done: 01.07.2011

```
10000 All Variations of a Lock
----------------------------------------------------
 Statistic Module run time is: 01:20:640
STATISTIC MODULE 4
maXbox V: 3.3.1.4
"D:\Program Files\Adobe\Acrobat 6.0\Reader\AcroRd32.exe" "%1"Default application
mX3 executed on: 01.07.2011 08:07:38
PascalScript maXbox3 - RemObjects & SynEdit
```

1: The Simulation of the four Cases

And once again you can also use the functions of your calculator nPr and nCr.

As well as the "big parentheses" in a combination, people also use different notations.

Permutation or Combination Using combination think of a lotto game or a digital lock.

It’s also interesting to note how this formula of combination is nice and symmetrical:

$$\frac{n!}{r!(n-r)!} = \binom{n}{r} = \binom{n}{n-r}$$

Probability theory is required to describe nature.[19] A revolutionary discovery of early 20th century physics was the random character of all physical processes that occur at sub-atomic scales and are governed by the laws of quantum mechanics. The objective wave function evolves deterministically but, according to the Copenhagen interpretation, randomness is explained by a wave function collapse when an observation is made. However, the loss of determinism for the sake of instrumentalism did not meet with universal approval. Albert Einstein famously remarked in a letter to Max Born: "I am convinced that God does not play dice".[20] (http://en.wikipedia.org/wiki/Probability)

But knowing how these formulas work is only half the battle. Figuring out how to interpret a real world situation can be quite hard. But at least now you know how to calculate all 4 cases.

Task: Try to change the Output in the procedure `One_Permutation_4` with numbers from 1 to 4 in order to avoid the 0.

Experiment: Picking a card:

In an experiment, a card is picked from a stack of six cards, which spells the word PASCAL.

Possible outcomes are P, A 1, S, C, A 2 and L.
Sample space, S = {P, A 1, S, C, A 2 L}. There are 2 cards with the letter ‘A’ why and what for?

1.7 Conclusion

Let’s consider in the end the 4 cases in comparison with a lock and his elements:

1. Permutation = n! Num. [0..3], 4 Elements, Calculation 4!=24

1	2	0	3

2. Permutation (Variation without repeating) Num. [0..9], 4 Elements, Calculation nPr(10,4) = 5040

1	2	8	9

3. Combination = nCr = nPr / k! Num. [0..1] 20 Elements, Calculation nPr / k! = 4845

0	0	0	**1**	0	0	**1**	**1**	0	0	0	0	0	**1**	0	0	0	0	0	0

4. Variation (repeating) = n^k Num. [0..9] 4 Elements, Calculation n^k=10^4=10000

1	1	1	2

max@kleiner.com
Links of maXbox and an Article of Big Numbers:

http://www.softwareschule.ch/maxbox.htm
http://sourceforge.net/projects/maxbox
http://sourceforge.net/apps/mediawiki/maxbox/
http://www.delphi3000.com/articles/article_6712.asp?SK=

Reward 2010 - Algorithm of the Year 2010 solves all Lotto Combinations also with Compiler!

http://www.softwareschule.ch/download/97_pas_lottocombinations_beat_plus.pdf
http://www.softwareschule.ch/download/maxbox97_lottoproject_allcombinations_dpr.txt

# 19 Burgi, Mark. ”Interpretations of Negative Probabilities”. 2009, p. 1.
# 20 Jedenfalls bin ich überzeugt, daß der Alte nicht würfelt.