```
 1: ////////////////////////////////////////////////////
 2: Code Tuning
 3: ____________________________________________________
 4: maXbox Starter 88 -Performance Tuning - Max Kleiner
 5:
 6: "There is always space for improvement"
 7:     — Oscar De La Hoya
 8:  https://my6.code.blog/2021/09/08/improver-4/
 9:
10:
11: Performance is a crucial thing with many aspects. Other Pascal or Python
    implementations (or older or still-under development versions of CPython
    or maXbox bytecode ) may have slightly different performance
    characteristics. However, it is generally safe to assume that they are not
    slower by more than a factor of O(log n).
12: So this is how we can measure or predict performance, namely by the O-
    Notation: Generally, 'n' is the number of elements currently in a list or
    container. 'k' is either the value of a parameter or the number of
    elements in the parameter.
13: For example a list with 4 values each:
14:
15:     {
16:       "date": "2021-10-25",
17:       "confirmed": 36223,
18:       "passed": 1483,
19:       "recovered": 33632
20:     }
21:
22: Internally, a list is represented as an array; our list is an array of
    records. The largest costs come from growing beyond the current allocation
    size (because everything must move), or from inserting or deleting
    somewhere near the beginning (because everything after that must move).
23:
24: Lets start with a prime number generator and measure the time it runs:
25:
26: https://www.wolframalpha.com/input/?i=primes+0+to+10000
27:
28: const PRIME =  def py_is_prime(n):
29:                   import math'+LB+
30:                   """ totally naive implementation """
31:                   if n <= 1:
32:                     return False
33:                   q = math.floor(math.sqrt(n))
34:                   for i in range(2, q + 1):
35:                     if (n % i == 0):
36:                       return False
37:                   return True
38:
39: We measure this runtime with a TStopwatch object:
40:
41:     writeln('CPUspeed before: '+cpuspeed)
42:     sw:= TStopWatch.Create();
43:     sw.Start;
44:     for it:= 0 to 10000 do
45:        primelist.add(eng.EvalStr('py_is_prime('+itoa(it)+')'));
46:     println('count primes1: '+itoa(SubstringCount('True',
47:                                                  primelist.text)));
48:     sw.Stop;
49:     //sw.ElapsedMilliseconds;
50:     writeln('Stop Watch Prime Tester1: '+sw.getValueStr)
51:
52: Then we get a first baseline time:
```

```
53:     >cpuspeed before: 2263
54:     >count primes1: 1229
55:     >Stop Watch Prime Tester1: 0:0:0.519
56:     >cpuspeed after: 2195
57:
58: The Average Case times listed for this range loop assume that the function
    has no threads and uses a tuning trick to shorten the range with
59: q = math.floor(math.sqrt(n))
60:
61: One easy way to generate primes numbers in a range [2,n][2, n][2,n] is to
    bruteforces from 2 to n and check if the number is a prime using the
    square root primilarity test. This is easy to understand but it's much
    slow since for each iteration we are performing a square root primilary
    test. But why this square root? If a number n is not a prime, it can be
    factored into two factors a and b: n = a * b or we say also n= p*q.
62: Now a and b cant be both greater than the square root of n, since then the
    product p*q would be greater than sqrt(n)*sqrt(n) = n. So in any
    factorization of n, at least one of the factors must be smaller than the
    square root of n, and if we do not find any factors less than or equal to
    the sqrt, so n must be a prime. Clearly, both of them cant be greater than
    sqrt(N) simultaneously.
63: In some cases you can use ceil() instead of floor(), this might be useful,
    but this all heavily depends on implementation details (programming
    language, hardware, linker, data types, libraries), none of which are
    known in this general consideration.
64:
65: A third implementation could be:
66:
67:    function D_isprime3py(n:integer):boolean;
68:    {Test an integer for "py primeness"}
69:    var i,q:integer;
70:    begin
71:       if n <= 1 then begin result:= false;
72:          exit; end;
73:       q:= Floor(Sqrt(n));
74:       for i:= 2 to q do
75:         if (n mod i = 0) then begin
76:           result:= false;
77:           Exit;
78:         end;
79:       result:= True
80:       Exit;
81:    end;
82:
83: >count primes3: 1229
84: <Stop Watch Prime Tester3: 0:0:0.469
85:
86: If we call precompiled code for the objects is sufficiently robust to make
    more speed common. An interpreter like maXbox generally uses one of the
    following strategies for program execution:
87: 1. parse the source code and perform its behavior directly;
88: 2. translate source code into some efficient intermediate representation
89:    and immediately execute this bytecode;
90: 3. explicitly execute stored precompiled code made by a compiler which is
91:    part of the interpreter system.
92:
93: In a second attempt we use two ways with procompiled code, an object-code
    and and a function code, so we start with the object-code:
94:
95:     writeln('Precompiled Performance_________________');
96:     mpr:= TPrimes.create;
97:     sw.Start;
```

```
 98:     writeln('compiled class pretest '+botostr(mpr.isprime(4703)))
 99:     for it:= 0 to primelimit do
100:       primelist.add(botostr(mpr.IsPrime(it)));
101:     writeln('count primes4: '+itoa(SubstringCount('TRUE',
102:                                                   primelist.text)));
103:     sw.stop;
104:     writeln('Stop Watch Prime Tester4: '+sw.getValueStr);
105:
106: >compiled class pretest TRUE
107: >count primes4: 1229
108: >Stop Watch Prime Tester4: 0:0:0.125
109:
110: We will have to unwind the class with a constructor create() like above to
     build a proper object with calling mpr.isprime() method at runtime for
     TPrimes:
111:
112: All of this initialization is done in as the program starts in the Create
     constructor method of a TPrimes object.  If you override a constructor,
     always call the constructor of the ancestor object first, TObject in this
     case. An instance of TPrimes, named Primes, is created in the
     Initialization section of the unit. Prime numbers are created up to 105
     (all 9,592 of them) in a second or two. This will allow direct lookup of
     primes less than 105 and testing or getting factors numbers up to 1010
     (since their largest prime factor will be less than 105 ).
113: The Mathslib unit contains also the IsPrime() or flcisPrime() function
     which tests whether each candidate number is prime. This we test at last:
114:
115:     writeln('compiled function pretest '+botostr(flcisprime(4703)))
116:     for it:= 0 to primelimit do
117:       primelist.add(botostr(flcIsPrime(it)));
118:     writeln('count primes5: '+itoa(SubstringCount('TRUE',
119:                                                   primelist.text)));
120:     sw.stop;
121:     writeln('Stop Watch Prime Tester5: '+sw.getValueStr);
122:     writeln('CPUspeed after: '+cpuspeed)
123:
124: >compiled function pretest TRUE
125: >count primes5: 1229
126: >Stop Watch Prime Tester5: 0:0:0.109
127: >CPUspeed after: 2263
128:
129: And this procompiled function is also the winner we about 100 milliseconds
     runtime.
130: The plot you can find at:
131:   http://www.softwareschule.ch/examples/1070_tprime_tshirt.png
132:
133: The TPrimes class introduced has been enhanced to include the GetPrevPrime
     function and moved into a MathsLib unit which is included here with the
     source code zip file and is also now included in our common library file
     DFFLIB registered in maXbox.
134:
135: Procedure testprimesPerformance;   // Performance Test
136: var count,beg,j,r: integer; f:boolean;
137: begin
138:    processmessagesOFF
139:    Println(' prime time performance check ?????: ');
140:    count:= 0;
141:    beg:= Random(1000000000)+2;
142:    for it:=beg to beg+5000 do begin
143:      f:= True;
144:      j:= 2;
145:      r:= Round(Sqrt(it));
```

```
146:      while f and (j<=r) do
147:      if it mod j = 0 then f:= False
148:      else inc(j);
149:      if f then begin
150:        Print(itoa(it)+' ');
151:        count:= count+ 1;
152:        if count mod 8 = 0 then
153:        Println('');
154:      end;
155:    end;
156:    processmessagesON;
157: end;
158:
159: To visualize this performance test we can use TEEChart; TeeChart is a
     charting library for programmers, developed and managed by Steema Software
     of Girona, Catalonia, Spain. Its available as commercial and non-
     commercial software. TeeChart has been included in most Delphi, Lazarus
     and C++Builder products since 1997.
160:
161: {$Conclusion}:
162: The advantage of running a .psb file is that maXbox doesnt have to incur
     the overhead of compiling it before running it (menu/Program/Run Mode).
     Its worth noting that while running a compiled script has a faster startup
     time (as it doesnt need to be compiled), it doesnt run any faster. Since
     mX could compile to byte-code before running a .pas or .txt file anyway,
     there should not be any performance improvement aside from that.
163:  Those are the supported functions (menu/Options/Save
     Bytecode//LoadBytecode):
164:
165: -----PS-BYTECODE (PSB) mX4-----14:07:09
166: -----BYTECODE saved as:
     C:\maXbox\mX39998\maxbox3\examples\1070_tshirt_prime2_tutor88.psb ---
167:
168: So we can really speed up a precompiled function as running with bytecode:
169: -Precompiled
170: >count primes5: 1229
171: >Stop Watch Prime Tester5: 0:0:0.109
172: >CPUspeed after: 2263
173:
174: -Precompiled as Bytecode
175: >count primes5: 1229
176: >Stop Watch Prime Tester5: 0:0:0.63
177: >CPUspeed after: 2195
178:
179: Advantages with precompiled bytecode (prebyte) functionality:
180: •  compilation results in smaller file, you will get faster load times.
181: •  skips the compilation step. Faster at intial load.
182: •  the more you comment, the smaller .psb file will be in comparison
183:    to source .pas file.
184: •  aiming at an embedded system, obtaining a smaller size file.
185: •  share temporary data in bytecode anywhere.
186: •  resized, defeatable obfuscation.
187:
188: Script Ref:
189:     http://www.softwareschule.ch/examples/1070_tshirt_prime2_tutor88.txt
190:     http://www.softwareschule.ch/examples/1070_tprime_tshirt.png
191:
     http://delphiforfun.org/Programs/Delphi_Techniques/PrimesFromDigits.htm
```