# Project Euler

I used the Project Euler problems as a test of XLISP. Since the answers to the problems are available by doing a Google search (but not in XLISP!) I don't feel I'm spoiling anything by making this public. It does show the flexibility of the language.

I started out doing then when I looked at Project Euler and saw some problems that I felt could be uniquely and easily programmed in LISP. I then continued and even did some that were not well suited at all! In doing this I did uncover a couple of obscure bugs, so it was a good exercise.

Tom Almy

August 2013 (although problems mostly done in the Fall of 2011).

**Problem 1**
If we list all the natural numbers below 10 that are multiples of 3 or 5, we get 3, 5, 6 and 9. The sum of these multiples is 23.

Find the sum of all the multiples of 3 or 5 below 1000.

**Answer 1**
Quick and Dirty in Lisp. Didn't even write a function.

```
(let ((n 0))
     (dotimes (i 1000)
              (when (or (zerop (mod i 5))(zerop (mod i 3)))
                    (incf n i)))
     n)
```

**Problem 2**
Each new term in the Fibonacci sequence is generated by adding the previous two terms. By starting with 1 and 2, the first 10 terms will be:

1, 2, 3, 5, 8, 13, 21, 34, 55, 89, ...

By considering the terms in the Fibonacci sequence whose values do not exceed four million, find the sum of the even-valued terms.

**Answer 2**
Another brute force solution in LISP:

```
(do ((n 2 (+ n nm1)) (nm1 1 n) (sum 0))
    ((> n 4000000) sum)
    (when (evenp n) (incf sum n)))
```

It makes use of the parallel assignment done by the do form.

**Problem 3**
The prime factors of 13195 are 5, 7, 13 and 29.

What is the largest prime factor of the number 600851475143 ?

**Answer 3**
I already had an efficient factoring function (it builds a table of primes which it reuses). So my LISP solution was

(apply #'max (factor 600851475143))

**Problem 4**
A palindromic number reads the same both ways. The largest palindrome made from the product of two 2-digit numbers is 9009 = 91 × 99.

Find the largest palindrome made from the product of two 3-digit numbers.

**Answer 4**
13 msec in LISP (XLISP-PLUS on a 2.8GHz i7 iMac).

Lots of time saved by exiting loops when no further values can yield a larger result, however it is brute force in that it could conceivably test 100*100.

```
(defun palindromep (val)
       (let ((str (format nil "~s" val)))
          (equal str (reverse str))))

(defun euler4 ( &aux (maxfound 0))
       (do ((v1 999 (1- v1)))
         ((or (< v1 100) (< (* v1 v1) maxfound))
          maxfound)
         (do* ((v2 v1 (1- v2))
             (product (* v1 v2)  (* v1 v2)))
            ((or (< v2 100) (<= product maxfound)))
            (when (palindromep product)
                  (setf maxfound product)))))

(euler4)
```

**Problem 5**
2520 is the smallest number that can be divided by each of the numbers from 1 to 10 without any remainder.

What is the smallest positive number that is evenly divisible by all of the numbers from 1 to 20?

**Answer 5**
In LISP (XLISP-PLUS) there is a least common multiple function, so the simple brute-force solution is:

```
(lcm 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1)
```

But with the generation of all the integers it becomes:

```
(setf v nil)
(dotimes (i 20) (push (1+ i) v))
(apply #'lcm v)
```

Execution time for lcm is 4.1 microseconds (yes!) on my 2.8GHz i7 iMac.

**Problem 6**

The sum of the squares of the first ten natural numbers is,

$$1^2 + 2^2 + ... + 10^2 = 385$$

The square of the sum of the first ten natural numbers is,

$$(1 + 2 + ... + 10)^2 = 55^2 = 3025$$

Hence the difference between the sum of the squares of the first ten natural numbers and the square of the sum is 3025 — 385 = 2640.

Find the difference between the sum of the squares of the first one hundred natural numbers and the square of the sum.

**Answer 6**

With the addition of a function to create a list of consecutive integers it becomes very straightforward in LISP (XLISP-PLUS):

```
(defun iota (n &aux v) "return a list of integers 1 through n"
       (dotimes (i n) (push (1+ i) v))
       v
       )

(defun sumsq (n)
       (reduce #'+ (mapcar #'(lambda (x) (* x x)) n)))

(defun sqsum (n)
       (expt (reduce #'+ n) 2))

(defun euler6 ()
       (let ((n (iota 100)))
          (abs (- (sumsq n) (sqsum n)))))
```

**Problem 7**

By listing the first six prime numbers: 2, 3, 5, 7, 11, and 13, we can see that the 6th prime is 13.

What is the 10,001st prime number?

**Answer 7**

I already had a factoring program that creates a list of primes on the fly, so I used that as a basis for this program in XLISP-PLUS.

First I make sure I've got at least 10001 primes:

```
(do () ((> (length *primes*) 10000)) (factor::addprime))
```

Then I get the 10001th element (index 10000) in the list:

```
(nth 10000 *primes*)
```

Even when no primes are known by the algorithm it takes only 26 msec on my 2.8GHz i7 iMac to solve.

**Problem 8**
Find the greatest product of five consecutive digits in the 1000-digit number.

```
73167176531330624919225119674426574742355349194934
96983520312774506326239578318016984801869478851843
85861560789112949495459501737958331952853208805511
12540698747158523863050715693290963295227443043557
66896648950445244523161731856403098711121722383113
62229893423380308135336276614282806444486645238749
30358907296290491560440772390713810515859307960866
70172427121883998797908792274921901699720888093776
65727333001053367881220235421809751254540594752243
52584907711670556013604839586446706324415722155397
53697817977846174064955149290862569321978468622482
83972241375657056057490261407972968652414535100474
82166370484403199890008895243450658541227588666881
16427171479924442928230863465674813919123162824586
17866458359124566529476545682848912883142607690042
24219022671055626321111109370544217506941658960408
07198403850962455444362981230987879927244284909188
84580156166097919133875499200524063689912560717606
05886116467109405077541002256983155200055935729725
71636269561882670428252483600823257530420752963450
```

**Answer 8**
In LISP (XLISP-PLUS). I converted to a list of digits (as integers) in the first step to aid processing. Done with an apply, a maplist, and a map -- no use of "program feature", just traditional functional coding.

```
(apply #'max
       (maplist #'(lambda (x) (if (> (length x) 5)
                                  (* (car x) (cadr x) (caddr x) (cadddr x) (fifth x))
                                  0))
            (map 'list #'(lambda (x) (- (char-int x) (char-int #\0)))
                 (strcat
```

```
"7316717653133062491922511967442657474235534919493 4"
"9698352031277450632623957831801698480186947885184 3"
"8586156078911294949545950173795833195285320880551 1"
"1254069874715852386305071569329096329522744304355 7"
"6689664895044524452316173185640309871112172238311 3"
"6222989342338030813533627661428280644448664523874 9"
"3035890729629049156044077239071381051585930796086 6"
"7017242712188399879790879227492190169972088809377 6"
"6572733300105336788122023542180975125454059475224 3"
"5258490771167055601360483958644670632441572215539 7"
"5369781797784617406495514929086256932197846862248 2"
"8397224137565705605749026140797296865241453510047 4"
"8216637048440319989000889524345065854122758866688 1"
"1642717147992444292823086346567481391912316282458 6"
"1786645835912456652947654568284891288314260769004 2"
"2421902267105562632111110937054421750694165896040 8"
"0719840385096245544436298123098787992724428490918 8"
"8458015616609791913387549920052406368991256071760 6"
"0588611646710940507754100225698315520005593572972 5"
"7163626956188267042825248360082325753042075296345 0"))))
```

Execution time is 3.27 msec on 2.8GHz i7 iMac.

**Problem 9**

A Pythagorean triplet is a set of three natural numbers, $a < b < c$, for which,

$$a^2 + b^2 = c^2$$

For example, $3^2 + 4^2 = 9 + 16 = 25 = 5^2$.

There exists exactly one Pythagorean triplet for which $a + b + c = 1000$.

Find the product $abc$.

**Answer 9**

Brute force in LISP (XLISP-PLUS). 20 msec on 2.8GHz i7 iMac.

```
(defun euler9 ()
       (do ((c 500 (1- c))) ; hypotenuse can't be greater than 500
          () ; since a solution is known to exist, no termination test needed
          (do* ((b (floor (- 1000 c) 2) (1- b)) ; second side to half of remainder
               (arequired (- 1000 c b) (- 1000 c b))
               (csq (* c c)))
              ((< b 1)) ; b must be greater than zero
              (when (eql csq (+ (* b b) (* arequired arequired)))
                    (return-from euler9 (list arequired b c (* arequired b c))))
             )))
```

**Problem 10**

The sum of the primes below 10 is 2 + 3 + 5 + 7 = 17.

Find the sum of all the primes below two million.

**Solution 10**

In LISP (XLISP-PLUS) I've got a function nextprime that finds the next prime after any value. It makes for easy coding but not the most efficient way to solve this problem. 26 seconds, and orders of magnitude longer than any other problem I've done here, but only took me under a minute to write.

```
(setq sum 0)
(do ((prime 2 (nextprime prime)))
    ((> prime 2000000))
    (incf sum prime))
```

**Problem 11**

In the 20×20 grid below, four numbers along a diagonal line have been marked in red.

```
08 02 22 97 38 15 00 40 00 75 04 05 07 78 52 12 50 77 91 08
49 49 99 40 17 81 18 57 60 87 17 40 98 43 69 48 04 56 62 00
81 49 31 73 55 79 14 29 93 71 40 67 53 88 30 03 49 13 36 65
52 70 95 23 04 60 11 42 69 24 68 56 01 32 56 71 37 02 36 91
22 31 16 71 51 67 63 89 41 92 36 54 22 40 40 28 66 33 13 80
24 47 32 60 99 03 45 02 44 75 33 53 78 36 84 20 35 17 12 50
32 98 81 28 64 23 67 10 26 38 40 67 59 54 70 66 18 38 64 70
67 26 20 68 02 62 12 20 95 63 94 39 63 08 40 91 66 49 94 21
24 55 58 05 66 73 99 26 97 17 78 78 96 83 14 88 34 89 63 72
21 36 23 09 75 00 76 44 20 45 35 14 00 61 33 97 34 31 33 95
78 17 53 28 22 75 31 67 15 94 03 80 04 62 16 14 09 53 56 92
16 39 05 42 96 35 31 47 55 58 88 24 00 17 54 24 36 29 85 57
86 56 00 48 35 71 89 07 05 44 44 37 44 60 21 58 51 54 17 58
19 80 81 68 05 94 47 69 28 73 92 13 86 52 17 77 04 89 55 40
04 52 08 83 97 35 99 16 07 97 57 32 16 26 26 79 33 27 98 66
88 36 68 87 57 62 20 72 03 46 33 67 46 55 12 32 63 93 53 69
04 42 16 73 38 25 39 11 24 94 72 18 08 46 29 32 40 62 76 36
20 69 36 41 72 30 23 88 34 62 99 69 82 67 59 85 74 04 36 16
20 73 35 29 78 31 90 01 74 31 49 71 48 86 81 16 23 57 05 54
01 70 54 71 83 51 54 69 16 92 33 48 61 43 52 01 89 19 67 48
```

The product of these numbers is 26 × 63 × 78 × 14 = 1788696.

What is the greatest product of four adjacent numbers in any direction (up, down, left, right, or diagonally) in the 20×20 grid?

**Solution 11**

Hardest part of this brute force solution is that XLISP-PLUS doesn't have multidimensional arrays, so I had to use an array of arrays. But I probably picked up some efficiency that way. 1.67 msec on my 2.8GHz i7 iMac.

```
(defconstant grid #(
#(08 02 22 97 38 15 00 40 00 75 04 05 07 78 52 12 50 77 91 08)
#(49 49 99 40 17 81 18 57 60 87 17 40 98 43 69 48 04 56 62 00)
#(81 49 31 73 55 79 14 29 93 71 40 67 53 88 30 03 49 13 36 65)
#(52 70 95 23 04 60 11 42 69 24 68 56 01 32 56 71 37 02 36 91)
#(22 31 16 71 51 67 63 89 41 92 36 54 22 40 40 28 66 33 13 80)
#(24 47 32 60 99 03 45 02 44 75 33 53 78 36 84 20 35 17 12 50)
#(32 98 81 28 64 23 67 10 26 38 40 67 59 54 70 66 18 38 64 70)
```

```
#(67 26 20 68 02 62 12 20 95 63 94 39 63 08 40 91 66 49 94 21)
#(24 55 58 05 66 73 99 26 97 17 78 78 96 83 14 88 34 89 63 72)
#(21 36 23 09 75 00 76 44 20 45 35 14 00 61 33 97 34 31 33 95)
#(78 17 53 28 22 75 31 67 15 94 03 80 04 62 16 14 09 53 56 92)
#(16 39 05 42 96 35 31 47 55 58 88 24 00 17 54 24 36 29 85 57)
#(86 56 00 48 35 71 89 07 05 44 44 37 44 60 21 58 51 54 17 58)
#(19 80 81 68 05 94 47 69 28 73 92 13 86 52 17 77 04 89 55 40)
#(04 52 08 83 97 35 99 16 07 97 57 32 16 26 26 79 33 27 98 66)
#(88 36 68 87 57 62 20 72 03 46 33 67 46 55 12 32 63 93 53 69)
#(04 42 16 73 38 25 39 11 24 94 72 18 08 46 29 32 40 62 76 36)
#(20 69 36 41 72 30 23 88 34 62 99 69 82 67 59 85 74 04 36 16)
#(20 73 35 29 78 31 90 01 74 31 49 71 48 86 81 16 23 57 05 54)
#(01 70 54 71 83 51 54 69 16 92 33 48 61 43 52 01 89 19 67 48)))

(defun euler11 ()
       (let ((result 0))
            ; handle horizontals
            (dotimes (irow 20)
                     (let ((row (aref grid irow)))
                          (dotimes (icol 17)
                                   (let ((val (* (aref row icol)
                                                 (aref row (1+ icol))
                                                 (aref row (+ icol 2))
                                                 (aref row (+ icol 3)))))
                                        (when (> val result)
                                              (setf result val))))))
            ; handle verticals and diagonals
            (dotimes (irow 17)
                     (let ((row1 (aref grid irow))
                           (row2 (aref grid (1+ irow)))
                           (row3 (aref grid (+ irow 2)))
                           (row4 (aref grid (+ irow 3))))
                          ; verticals
                          (dotimes (icol 20)
                                   (let ((val (* (aref row1 icol)
                                                 (aref row2 icol)
                                                 (aref row3 icol)
                                                 (aref row4 icol))))
                                        (when (>  val result)
                                              (setf result val))))
                          ; top-left to bottom-right diagonal
                          (dotimes (icol 17)
                                   (let ((val (* (aref row1 icol)
                                                 (aref row2 (1+ icol))
                                                 (aref row3 (+ icol 2))
                                                 (aref row4 (+ icol 3)))))
                                        (when (>  val result)
                                              (setf result val))))
                          ; bottom-left to top-right diagonal
                          (dotimes (icol 17)
                                   (let ((val (* (aref row1 (+ icol 3))
                                                 (aref row2 (+ icol 2))
                                                 (aref row3 (1+ icol))
                                                 (aref row4 icol))))
                                        (when (>  val result)
                                              (setf result val))))
                          ))
            result))
```

**Problem 12**
The sequence of triangle numbers is generated by adding the natural numbers. So the 7th triangle number would be 1 + 2 + 3 + 4 + 5 + 6 + 7 = 28. The first ten terms would be:

1, 3, 6, 10, 15, 21, 28, 36, 45, 55, ...

Let us list the factors of the first seven triangle numbers:

**1**: 1

**3**: 1,3

**6**: 1,2,3,6

**10**: 1,2,5,10

**15**: 1,3,5,15

**21**: 1,3,7,21

**28**: 1,2,4,7,14,28

We can see that 28 is the first triangle number to have over five divisors.

What is the value of the first triangle number to have over five hundred divisors?

**Answer 12**
I already have a prime factoring function in LISP (XLISP-PLUS) so the only real challenge here was to write a function that would count the number factors given the list of prime factors. I tried to make it all as efficient as possible, even bypassing the all factors calculation when the number of prime factors was <=8, but it still took 0.6 seconds on my 2.8GHz i7 iMac. Still not the longest so far.

```
(defun allfactors (primefactors)
       (let ((current 0) (curcount 0) (total 1))
          (mapcar #'(lambda (factor)
                        (if (eql factor current)
                          (incf curcount)
                          (setf total (* total (1+ curcount))
                              current factor
                              curcount 1)))
                primefactors)
          (* total (1+ curcount))))

(defun euler12 ()
       (do* ((i 1 (1+ i))
           (triangle i (+ triangle i))
           (factors (factor triangle) (factor triangle)))
          ((and (> (length factors) 8) (> (allfactors factors) 500))
           triangle)
          ))
```

**Problem 13**

Work out the first ten digits of the sum of the following one-hundred 50-digit numbers.

```
37107287533902102798797998220837590246510135740250
46376937677490009712648124896970078050417018260538
74324986199524741059474233309513058123726617309629
91942213363574161572522430563301811072406154908250
23067588207539346171171980310421047513778063246676
89261670696623633820136378418383684178734361726757
28112879812849979408065481931592621691275889832738
44274228917432520321923589422876796487670272189318
47451445736001306439091167216856844588711603153276
70386486105843025439939619828917593665686757934951
62176457141856560629502157223196586755079324193331
64906352462741904929101432445813822663347944758178
92575867718337217661963751590579239728245598838407
58203565325359399008402633568948830189458628227828
80181199384826282014278194139940567587151170094390
35398664372827112653829987240784473053190104293586
86515506006295864861532075273371959191420517255829
71693888707715466499115593487603532921714970056938
54370070576826684624621495650076471787294438377604
53282654108756828443191190634694037855217779295145
36123272525000296071075082563815656710885258350721
45876576172410976447339110607218265236877223636045
17423706905851860660448207621209813287860733969412
81142660418086830619328460811191061556940512689692
51934325451728388641918047049293215058642563049483
62467221648435076201727918039944693004732956340691
15732444386908125794514089057706229429197107928209
55037687525678773091862540744969844508330393682126
18336384825330154686196124348767681297534375946515
80386287592878490201521685554828717201219257766954
78182833757993103614740356856449095527097864797581
16726320100436897842553539920931837441497806860984
48403098129077791799088218795327364475675590848030
87086987551392711854517078544161852424320693150332
59959406895756536782107074926966537676326235447210
69793950679652694742597709739166693763042633987085
41052684708299085211399427365734116182760315001271
65378607361501080857009149939512557028198746004375
35829035317434717326932123578154982629742552737307
94953759765105305946966067683156574377167401875275
88902802571733229619176668713819931811048770190271
25267680276078003013678680992525463401061632866526
36270218540497705585629946580636237993140746255962
24074486908231174977792365466257246923322810917141
91430288197103288597806669760892938638285025333403
34413065578016127815921815005561868836468420090470
23053081172816430487623791969842487255036638784583
11487696932154902810424020138335124462181441773470
63783299490636259666498587618221225225512486764533
67720186971698544312419572409913959008952310058822
95548255300263520781532296796249481641953868218774
76085327132285723110424803456124867697064507995236
```

```
37774242535411291684276865538926205024910326572967
23701913275725675285653248258265463092207058596522
29798860272258331913126375147341994889534765745501
18495701454879288984856827726077713721403798879715
38298203783031473527721580348144513491373226651381
34829543829199918180278916522431027392251122869539
40957953066405232632538044100059654939159879593635
29746152185502371307642255121183693803580388584903
41698116222072977186158236678424689157993532961922
62467957194401269043877107275048102390895523597457
23189706772547915061505504953922979530901129967519
86188088225875314529584099251203829009407770775672
11306739708304724483816533873502340845647058077308
82959174767140363198008187129011875491310547126581
97623331044818386269515456334926366572897563400500
42846280183517070527831839425882145521227251250327
55121603546981200581762165212827652751691296897789
32238195734329339946437501907836945765883352399886
75506164965184775180738168837861091527357929701337
62177842752192623401942399639168044983993173312731
32924185707147349566916674687634660915035914677504
99518671430235219628894890102423325116913619626622
73267460800591547471830798392868535206946944540724
76841822524674417161514036427982273348055556214818
97142617910342598647204516893989422179826088076852
87783646182799346313767754307809363333018982642090
10848802521674670883215120185883543223812876952786
71329612474782464538636993009049310363619763878039
62184073572399794223406235393808339651327408011116
66627891981488087797941876876144230030984490851411
60661826293682836764744779239180335110989069790714
85786944089552990653640447425576083659976645795096
66024396409905389607120198219976047599490197230297
64913982680032973156037120041377903785566085089252
16730939319872750275468906903707539413042652315011
94809377245048795150954100921645863754710598436791
78639167021187492431995700641917969777599028300699
15368713711936614952811305876380278410754449733078
40789923115535562561142322423255033685442488917353
44889911501440648020369068063960672322193204149535
41503128880339536053299340368006977710650566631954
81234880673210146739058568557934581403627822703280
82616570773948327592232845941706525094512325230608
22918802058777319719839450180888072429661980811197
77158542502016545090413245809786882778948721859617
72107838435069186155435662884062257473692284509516
20849603980134001723930671666823555245252804609722
53503534226472524250874054075591789781264330331690
```

**Answer 13**

In LISP I typed in "(+" then copy and pasted the numbers in the problem statement, typed in ")" and got the sum:

5537376230390876637302048746832985971773659831892672

Then I copy and pasted the first ten digits into ProjectEuler.net.

Too simple.

**Problem 14**
The following iterative sequence is defined for the set of positive integers:

$n \rightarrow n/2$ ($n$ is even)

$n \rightarrow 3n + 1$ ($n$ is odd)

Using the rule above and starting with 13, we generate the following sequence:

$13 \rightarrow 40 \rightarrow 20 \rightarrow 10 \rightarrow 5 \rightarrow 16 \rightarrow 8 \rightarrow 4 \rightarrow 2 \rightarrow 1$

It can be seen that this sequence (starting at 13 and finishing at 1) contains 10 terms. Although it has not been proved yet (Collatz Problem), it is thought that all starting numbers finish at 1.

Which starting number, under one million, produces the longest chain?

**NOTE:** Once the chain starts the terms are allowed to go above one million.

**Answer 14**
First time I did it, it took 70 seconds. Then I added memoization (automatic hash table based shortcuts to a recursive routine) and it got worse! Iterative solution instead of recursive took 85 seconds. So I got creative and used a hash table for odd values and no hash table for even (since the values are heading down anyway!) and the time dropped to 4.2 seconds. I'm done!The program prints out each new best values as it finds them so I get a quick judge of execution speeds.

The hash table ends up with 793745 entries, if anyone is interested.

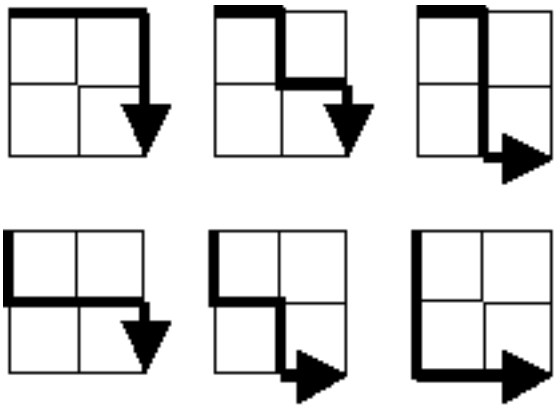
```
(setq h (make-hash-table :size 999983))

(defun euler14c (n &aux (hash (gethash n h))) ; 4.5 seconds
(cond
 (hash hash)
 ((evenp n) (1+ (euler14c (/ n 2))))
 ((eql n 1) 1)
 (t (setf (gethash n h) (1+ (euler14c (+ (* n 3) 1)))))))


(defun euler14 ()
       (do* ((i 1 (1+ i))
           (besti 0)
           (bestlen 0)
```

```
 (val (euler14c i) (euler14c i)))
((>= i 1000000) (cons besti bestlen))
(when (> val bestlen)
    (format t "~s ~s\n" i val)
    (setf besti i bestlen val))))
```

**Problem 15**
Starting in the top left corner of a 2×2 grid, there are 6 routes (without backtracking) to the bottom right corner.



How many routes are there through a 20×20 grid?

**Answer 15**
Nothing really new to add. I've got a factorial function in LISP which I used. Then it is a simple one-liner.

```
(floor (fact 40) (expt (fact 20) 2))
```

**Problem 16**
$2^{15}$ = 32768 and the sum of its digits is 3 + 2 + 7 + 6 + 8 = 26.

What is the sum of the digits of the number $2^{1000}$?

**Answer 16**
Not many people using LISP these days. This solution runs (well under a millisecond) in XLISP-PLUS. IMHO it makes better use of the MAP function than the previous LISP solution. IMHO it makes better use of the MAP function than the previous LISP solution realizing that MAP takes any sequence (explicit coercion from string to list not needed) and at least in XLISP-PLUS using an array result to REDUCE is faster than a list.

```
(defconstant asciizero (char-int #\0))
```

```
(defun euler16 ()
       (reduce #'+ (map 'array
                        #'(lambda (n) (- (char-int n) asciizero))
                        (format nil "~s" (expt 2 1000)))))
```

**Problem 17**
If the numbers 1 to 5 are written out in words: one, two, three, four, five, then there are 3 + 3 + 5 + 4 + 4 = 19 letters used in total.

If all the numbers from 1 to 1000 (one thousand) inclusive were written out in words, how many letters would be used?

**NOTE:** Do not count spaces or hyphens. For example, 342 (three hundred and forty-two) contains 23 letters and 115 (one hundred and fifteen) contains 20 letters. The use of "and" when writing out numbers is in compliance with British usage.

**Answer 17**
Common LISP (and XLISP-PLUS which I use) has a nice feature in the output formatter to output a number as English text. All I had to add was a function to count the alphabetic characters.

```
(defun count-alpha (str) ; 6.9msec
       (length (remove #\Space (remove #\- str))))

(defun euler17 ()
       (do ((i 1 (1+ i))
            (sum 0))
           ((> i 1000) sum)
           (incf sum (count-alpha (format nil "~r" i)))))
```

**Problem 18**
By starting at the top of the triangle below and moving to adjacent numbers on the row below, the maximum total from top to bottom is 23.

3
7 4
2 4 6
8 5 9 3

That is, 3 + 7 + 4 + 9 = 23.

Find the maximum total from top to bottom of the triangle below:

75
95 64
17 47 82
18 35 87 10
20 04 82 47 65
19 01 23 75 03 34
88 02 77 73 07 63 67
99 65 04 28 06 16 70 92
41 41 26 56 83 40 80 70 33
41 48 72 33 47 32 37 16 94 29
53 71 44 65 25 43 91 52 97 51 14
70 11 33 28 77 73 17 78 39 68 17 57
91 71 52 38 17 14 91 43 58 50 27 29 48
63 66 04 68 89 53 67 30 73 16 69 87 40 31
04 62 98 27 23 09 70 98 73 93 38 53 60 04 23

**NOTE:** As there are only 16384 routes, it is possible to solve this problem by trying every route. However, Problem 67, is the same challenge with a triangle containing one-hundred rows; it cannot be solved by brute force, and requires a clever method! ;o)

**Answer 18**

Somehow I thought out this problem from triangle top down, then seeing the answers here I re-solved for bottom up, which does take fewer calculations. Both solutions are provided below.

```
(defconstant triangle #(
#(75)
#(95 64)
#(17 47 82)
#(18 35 87 10)
#(20 04 82 47 65)
#(19 01 23 75 03 34)
#(88 02 77 73 07 63 67)
#(99 65 04 28 06 16 70 92)
#(41 41 26 56 83 40 80 70 33)
#(41 48 72 33 47 32 37 16 94 29)
#(53 71 44 65 25 43 91 52 97 51 14)
#(70 11 33 28 77 73 17 78 39 68 17 57)
#(91 71 52 38 17 14 91 43 58 50 27 29 48)
#(63 66 04 68 89 53 67 30 73 16 69 87 40 31)
#(04 62 98 27 23 09 70 98 73 93 38 53 60 04 23)
))

(defun row-summed-from-next (cur next)
       (let ((res (copy-seq cur))
             (reslen (length cur)))
            (dotimes (i reslen)
                     (incf (aref res i)
                           (max (aref next i) (aref next (1+ i)))))
            res
```

```
            ))

(defun row-summed-from-previous (prev cur)
       (let ((res (copy-seq cur))
             (reslen (length cur)))
            ; do leftmost
            (incf (aref res 0) (aref prev 0))
            ; do rightmost
            (incf (aref res (1- reslen)) (aref prev (- reslen 2)))
            ; do middles
            (dotimes (i (- reslen 2))
                     (incf (aref res (1+ i))
                           (max (aref prev i) (aref prev (1+ i)))))
            res
            ))

(defun euler18 (&aux (result (aref triangle 0))) "Top Down solution"
       (dotimes (i (1- (length triangle)))
                (setf result
                      (row-summed-from-previous result (aref triangle (1+ i))))
                )
       (reduce #'max result))

(defun euler18a (&aux (result (aref triangle (1- (length triangle))))) "Bottom Up
solution"
       (dotimes (i (1- (length triangle)))
                (setf result
                      (row-summed-from-next (aref triangle (- (length triangle) i 2))
                                            result)))
       result)
```

**Problem 19**

You are given the following information, but you may prefer to do some research for yourself.

1 Jan 1900 was a Monday.

Thirty days has September,
April, June and November.
All the rest have thirty-one,
Saving February alone,
Which has twenty-eight, rain or shine.
And on leap years, twenty-nine.

A leap year occurs on any year evenly divisible by 4, but not on a century unless it is divisible by 400.

How many Sundays fell on the first of the month during the twentieth century (1 Jan 1901 to 31 Dec 2000)?

**Answer 19**

If I hadn't constrained myself to using XLISP-PLUS I certainly would have done this assignment using a language with calendar functions built-in. But a search on Wikipedia yielded a simple algorithm to do the calculation.

```
; Implementation of Sakamoto's day of week algorithm in LISP
; From Wikipedia http://en.wikipedia.org/wiki/Calculating_the_day_of_the_week

(defun day-of-week (y m d) ; returns Sunday=0, Monday=1,...Saturday=7
       (when (< m 3) (decf y))
       (rem (+ y
               (floor y 4)
               (- (floor y 100))
               (floor y 400)
               (aref #(0 3 2 5 0 3 5 1 4 6 2 4) (1- m))
              d
               )
            7)
       )

(defun euler19 ()
       (do ((result 0)
            (day 1)
            (month 1 (1+ (rem month 12)))
            (year 1901 (if (eql month 12) (1+ year) year)))
           ((eql year 2001) result)
           (when (zerop (day-of-week year month day))
                 (incf result))))
```

**Problem 20**
*n*! means *n* × (*n* − 1) × ... × 3 × 2 × 1

For example, 10! = 10 × 9 × ... × 3 × 2 × 1 = 3628800,
and the sum of the digits in the number 10! is 3 + 6 + 2 + 8 + 8 + 0 + 0 = 27.

Find the sum of the digits in the number 100!

**Answer 20**
XLISP-PLUS has bignum support and makes it easy to add up the digits. I already had a factorial function written. So:

```
(require "fact")
(defconstant asciizero (char-int #\0))

(defun euler20 ()
       (reduce #'+
               (map 'array
                    #'(lambda (n) (- (char-int n) asciizero))
                    (format nil "~s" (fact 100)))))
```

**Problem 21**
Let d(*n*) be defined as the sum of proper divisors of *n* (numbers less than *n* which divide evenly into *n*).

If d($a$) = $b$ and d($b$) = $a$, where $a \neq b$, then $a$ and $b$ are an amicable pair and each of $a$ and $b$ are called amicable numbers.

For example, the proper divisors of 220 are 1, 2, 4, 5, 10, 11, 20, 22, 44, 55 and 110; therefore d(220) = 284. The proper divisors of 284 are 1, 2, 4, 71 and 142; so d(284) = 220.

Evaluate the sum of all the amicable numbers under 10000.

**Answer 21**

My first XLISP-PLUS solution used MAPCAN and MAPCON and looks elegant from a LISP point of view (excepting the proper-divisors calculating function), but was order $n^2$. I built a list of conses of numbers and their sum of factors and then searched for pairs. Slick but a dumb approach. 20 seconds.

```
(defun proper-divisors (n)
       (let ((result '(1))
             (sqrtn (sqrt n))
             (initi (if (evenp n) 2 3))
             (inci (if (evenp n) 1 2)))
            (do ((i initi (+ i inci)))
                ((>= i sqrtn)
                 (when (and (eql (* i i) n) (zerop (rem n i)))
                       (push i result)))
                (when (zerop (rem n i))
                      (push i result)
                      (push (floor n i) result)))
            result))

(defun build-d-table (n)
       (let ((result))
            (dotimes (i n)
                     (push (cons (1+ i) (apply #'+ (proper-divisors (1+ i)))) result))
            (nreverse result)))

(defun find-pairs (table)
       (mapcon #'(lambda (l)
                         (mapcan #'(lambda (e)
                                           (when (and (eql (caar l) (cdr e))
                                                      (eql (cdar l) (car e)))
                                                 (list (list (caar l) (car e)))))
                                 (cdr l))
                         )
               table))

(defun euler21 (&aux table)
       (setf table (delete-if #'(lambda (x) (eql 1 (cdr x))) (build-d-table 10000)))
       (reduce #'+ (mapcan #'identity (find-pairs table))))
```

I got rid of the table of sums of factors and got a further improvement, but I still like the LISPness of the first solution best. 0.39 seconds.

```
(defun sum-divisors (n)
       (let ((result 1)
             (sqrtn  (sqrt n))
```

```
             (initi (if (evenp n) 2 3))
             (inci (if (evenp n) 1 2)))
            (do ((i initi (+ i inci)))
                ((>= i sqrtn)
                 (when (and (eql i sqrtn) (zerop (rem n i)))
                       (incf result i)))
                (when (zerop (rem n i))
                      (incf result (+ i (floor n i)))))
            result))

(defun euler21b (&aux (result 0)) ;; No table version -- everso slightly faster
       (do* ((i 1 (1+ i))
            (sumfact (sum-divisors i) (sum-divisors i)))
            ((eql i 10000) result)
            (when (and (> sumfact i)
                       (<= sumfact 10000)
                       (eql i (sum-divisors sumfact)))
                  (format t "~s ~s\n" i sumfact)
                  (incf result (+ i sumfact)))))
```

**Problem 22**

Using names.txt (right click and 'Save Link/Target As...'), a 46K text file containing over five-thousand first names, begin by sorting it into alphabetical order. Then working out the alphabetical value for each name, multiply this value by its alphabetical position in the list to obtain a name score.

For example, when the list is sorted into alphabetical order, COLIN, which is worth 3 + 15 + 12 + 9 + 14 = 53, is the 938th name in the list. So, COLIN would obtain a score of 938 × 53 = 49714.

What is the total of all the name scores in the file?

**Answer 22**

I didn't really want to waste time parsing the file so I stuck the file with the commas removed between "(defconstant names #(" and "))". Then I ran the function below which sorts the list (using LISP's quicksort function) then adds up the values of each name using a MAP function nested within a DO.

```
(defun euler22 ()
       (let ((nl (sort (copy-seq names) #'string<))
             (result 0))
            (do ((i 1 (1+ i)))
                ((> i (length nl)))
                (incf result
                      (* i
                         (apply #'+
                                (map 'list #'(lambda (n) (- (char-int n) 64))
                                     (aref nl (1- i)))))))
            result))
```

**Problem 23**
A perfect number is a number for which the sum of its proper divisors is exactly equal to the number. For example, the sum of the proper divisors of 28 would be 1 + 2 + 4 + 7 + 14 = 28, which means that 28 is a perfect number.

A number $n$ is called deficient if the sum of its proper divisors is less than $n$ and it is called abundant if this sum exceeds $n$.

As 12 is the smallest abundant number, 1 + 2 + 3 + 4 + 6 = 16, the smallest number that can be written as the sum of two abundant numbers is 24. By mathematical analysis, it can be shown that all integers greater than 28123 can be written as the sum of two abundant numbers. However, this upper limit cannot be reduced any further by analysis even though it is known that the greatest number that cannot be expressed as the sum of two abundant numbers is less than this limit.

Find the sum of all the positive integers which cannot be written as the sum of two abundant numbers.

**Answer 23**
This one I had problems with because there was an error in my sum-divisors function (amazingly, the nonworking algorithm was still successful with problem 21!). Then I went through several coding iterations to get the speed up by over 100x. My approach:

1. Find abundant numbers <20162.

2. Create a boolean array of 20162 elements and mark those that are abundant.

3. Create a list of odd abundant numbers.

4. Iterate through all numbers <20162, i

   a. If number is odd look for values in odd list for which (i - value) is abundant.

      If no such value is found, then i isn't the sum of two abundant numbers.

   b. If number is even examine all values < i/2 and see if the value and (i - value)

      are abundant. If no such value is found, then i isn't the sum of two abundant numbers.

I could combine steps 1,2, and 3 into a single loop and save some time, but I preferred this factoring because it's easier to debug. Also, step 4 takes the bulk of the execution time.

I could factor step 4 differently -- one loop for odd and another for even, but I suspect that with XLISP-PLUS that would have been slower since DOTIMES is more efficient than DO.

```
(defun sum-divisors (n)
       (let ((result 1)
             (sqrtn (sqrt n))
             (initi (if (evenp n) 2 3))
             (inci (if (evenp n) 1 2)))
            (do ((i initi (+ i inci)))
                ((>= i sqrtn)
                 (when (and (eql (* i i) n) (zerop (rem n i)))
                       (incf result i)))
                (when (zerop (rem n i))
                      (incf result (+ i (floor n i)))))
            result))

(defun find-abundant-less-than (n &aux result)
       (do ((i 12 (1+ i)))
           ((eql i n) (coerce (nreverse result)'array))
           (when (> (sum-divisors i) i)
                 (push i result))))

(defun euler23d (&aux (result 0))
       (let* ((abundant (find-abundant-less-than 20162))
              (abarray (make-sequence 'array 20162))
              (oddarray (mapcan #'(lambda (n) (when (oddp n) (list n)))
                                (coerce abundant 'list))))
            (map nil #'(lambda (n) (setf (aref abarray n) t)) abundant)
            (dotimes (i 20162)
                     (when
                      (if (oddp i)
                          (do* ((list oddarray (cdr list))
                                (found nil))
                               ((or found (null list)) (null found))
                               (setf found (and (> (- i (car list)) 0)
                                                (aref abarray (- i (car list))))))
                          (do* ((j 1 (1+ j))
                                (idiv2 (floor i 2))
                                (found nil))
                               ((or found (> j idiv2))
                                (null found))
                               (setf found (and (aref abarray j) (aref abarray (- i
j))))))
                      (incf result i)
;                      (format t "~s\n" i)
                     ))
            result))
```

This took 1.13 seconds, about 100 times faster than the original attempt, not shown.

### Problem 24

A permutation is an ordered arrangement of objects. For example, 3124 is one possible permutation of the digits 1, 2, 3 and 4. If all of the permutations are listed

numerically or alphabetically, we call it lexicographic order. The lexicographic permutations of 0, 1 and 2 are:

012 021 102 120 201 210

What is the millionth lexicographic permutation of the digits 0, 1, 2, 3, 4, 5, 6, 7, 8 and 9?

**Answer 24**

I wrote a function to convert an integer to factoradic representation and another function to calculate a permutation based on a factoradic value. The top level function simply calculates the factoradic representation of 999999 and then uses that to permute values 0...9.

Fast and very clean LISP code.

```
(require "fact")
(defconstant factorials
        (let (result)
             (dotimes (i 10)
                      (push (fact i) result))
             result))

(defconstant digits '(0 1 2 3 4 5 6 7 8 9))

(defun to-factoradic (n)
       (mapcar #'(lambda (f)
                         (let ((result (floor n f)))
                              (decf n (* result f))
                              result))
               factorials))

(defun permute (factoridic choices)
       (mapcar #'(lambda (digit) (let ((result (nth digit choices)))
                                       (setf choices (remove result choices))
                                       result))
               factoridic))


(defun euler24 ()
       (permute (to-factoradic 999999) digits))
```

**Problem 25**

The Fibonacci sequence is defined by the recurrence relation:

F$n$ = F$n-1$ + F$n-2$, where $F_1$ = 1 and $F_2$ = 1.

Hence the first 12 terms will be:

$F_1 = 1$

$F_2 = 1$

$F_3 = 2$

$F_4 = 3$

$F_5 = 5$

$F_6 = 8$

$F_7 = 13$

$F_8 = 21$

$F_9 = 34$

$F_{10} = 55$

$F_{11} = 89$

$F_{12} = 144$

The 12th term, $F_{12}$, is the first term to contain three digits.

What is the first term in the Fibonacci sequence to contain 1000 digits?

**Answer 25**

This would have been so simple had I not hit a bug in the bignum code in my XLISP-PLUS interpreter. I spent far more time fixing that than writing the solution to this problem!

```
(defconstant digits1000 (expt 10 999))

(defun euler25 (n)
      (do   ((i 1 (1+ i)) ; term number
             (fib-i-1 0 fib-i)
             (fib-i   1 (+ fib-i fib-i-1)))
            ((> fib-i n) i)))

(euler25 digits1000)
```

**Problem 26**

A unit fraction contains 1 in the numerator. The decimal representation of the unit fractions with denominators 2 to 10 are given:

$^1/_2 = 0.5$

$^1/_3 = 0.(3)$

$^1/_4 = 0.25$

$^1/_5 = 0.2$

$^1/_6 = 0.1(6)$

$^1/_7 = 0.(142857)$

$^1/_8 = 0.125$

$^1/_9 = 0.(1)$

$^1/_{10} = 0.1$

Where 0.1(6) means 0.166666..., and has a 1-digit recurring cycle. It can be seen that $^1/_7$ has a 6-digit recurring cycle.

Find the value of $d < 1000$ for which $^1/_d$ contains the longest recurring cycle in its decimal fraction part.

**Answer 26**

```
(Defun Cycle-Length (N)
       (Do* (Prev-Fracs
           (Frac N (Mod (* Frac 10) 1))
           (I (Position Frac Prev-Fracs) (Position Frac Prev-Fracs)))
          (I (- (Length Prev-Fracs) I))
;         (Format T "~S " Frac)
          (Setf Prev-Fracs (Append Prev-Fracs (List Frac)))))

(Defun Euler26 ()
       (Let ((Max 0) (Answer 0))
          (Dotimes (I 1000)
                 (Let ((Val (Cycle-Length (/ 1 (1+ I)))))
                    (When (> Val Max)
                        (Setf Max Val Answer I))))
          Answer))
```

**Problem 27**

Euler published the remarkable quadratic formula:

$$n^2 + n + 41$$

It turns out that the formula will produce 40 primes for the consecutive values $n = 0$ to 39. However, when $n = 40$, $40^2 + 40 + 41 = 40(40 + 1) + 41$ is divisible by 41, and certainly when $n = 41$, $41^2 + 41 + 41$ is clearly divisible by 41.

Using computers, the incredible formula $n^2 - 79n + 1601$ was discovered, which produces 80 primes for the consecutive values $n = 0$ to 79. The product of the coefficients, −79 and 1601, is −126479.

Considering quadratics of the form:

$$n^2 + an + b, \text{ where } |a| \leq 1000 \text{ and } |b| \leq 1000$$

where $|n|$ is the modulus/absolute value of $n$

e.g. $|11| = 11$ and $|-4| = 4$

Find the product of the coefficients, $a$ and $b$, for the quadratic expression that produces the maximum number of primes for consecutive values of $n$, starting with $n = 0$.

**Answer 27**

Brute force, using my prime number server since we know that b must be prime. We also know that a must be odd, and greater than or equal to 2-b, and can save some time there.

```
; For polynomial n*n +an + b, given abs(a) < 1000 and abs(b) < 1000
; We know that b must be prime (considering n=0) and that a must be odd
; (considering n=1).

(require "factor")

(defun primelength (a b)
       (do* ((i 0 (1+ i))
             (val b (+ (* (+ i a) i) b)))
            ((or (< val 0) (null (primep val)))
             i)))


(defun euler27 ()
       (let (besta bestb (bestlen 0))
            (do ((b 3 (nextprime b)))
                ((> b 1000))
                (do* ((a (- 2 b) (+ 2 a)))
                     ((>= a 1000))
                     (let ((pl (primelength a b)))
                          (when (> pl bestlen)
                                (setf besta a
                                      bestb b
                                      bestlen pl)))))
            (* besta bestb)))
```

**Problem 28**
Starting with the number 1 and moving to the right in a clockwise direction a 5 by 5 spiral is formed as follows:

21 22 23 24 25
20 7 8 9 10
19 6 1 2 11
18 5 4 3 12
17 16 15 14 13

It can be verified that the sum of the numbers on the diagonals is 101.

What is the sum of the numbers on the diagonals in a 1001 by 1001 spiral formed in the same way?

**Answer 28**
I Noticed values in four corners of each concentric square and wrote a quick

solution.

```
(defun euler28 ()
       (let ((result 1)) ; Inner most is 1
            (do ((i 3 (+ i 2)))
                ((> i 1001))
                (incf result (+ (* i i 4) (* i -6) 6)))
            result))
```

**Problem 29**
Consider all integer combinations of $ab$ for $2 \leq a \leq 5$ and $2 \leq b \leq 5$:

$2^2=4$, $2^3=8$, $2^4=16$, $2^5=32$

$3^2=9$, $3^3=27$, $3^4=81$, $3^5=243$

$4^2=16$, $4^3=64$, $4^4=256$, $4^5=1024$

$5^2=25$, $5^3=125$, $5^4=625$, $5^5=3125$

If they are then placed in numerical order, with any repeats removed, we get the following sequence of 15 distinct terms:

4, 8, 9, 16, 25, 27, 32, 64, 81, 125, 243, 256, 625, 1024, 3125

How many distinct terms are in the sequence generated by $ab$ for $2 \leq a \leq 100$ and $2 \leq b \leq 100$?

**Answer 29**

Brute force. I did find that using a hash table was much faster than creating a list, sorting the list, and then removing duplicates. Should I have been surprised?

```
(defun euler29 ()
       (let (result)
            (do ((a 2 (1+ a)))
                ((> a 100))
                (do ((b 2 (1+ b)))
                    ((> b 100))
                    (push (expt a b) result)))
            (setq result (sort result #'<))
            (setq result
                  (mapcon #'(lambda (x)
                                    (if (eql (car x) (cadr x))
                                        nil
                                        (list (car x))))
                              result))
            (length result)))

(defun euler29a () ; WITH HASH TABLE
       (let ((result (make-hash-table :size 5001)))
            (do ((a 2 (1+ a)))
                ((> a 100))
                (do ((b 2 (1+ b)))
                    ((> b 100))
                    (setf (gethash (expt a b) result) t)))
            (hash-table-count result)))
```

**Problem 30**

Surprisingly there are only three numbers that can be written as the sum of fourth powers of their digits:

$1634 = 1^4 + 6^4 + 3^4 + 4^4$

$8208 = 8^4 + 2^4 + 0^4 + 8^4$

$9474 = 9^4 + 4^4 + 7^4 + 4^4$

As $1 = 1^4$ is not a sum it is not included.

The sum of these numbers is $1634 + 8208 + 9474 = 19316$.

Find the sum of all the numbers that can be written as the sum of fifth powers of their digits.

**Answer 30**

This was done with a straightforward brute force approach.

```
(defun euler30 () ; 9^5 is 59049, so we need to check to about 6*(9^5) =
```

```
(let ((result 0))
   (do ((i 2 (1+ i)))
     ((> i 354298))
     (when (eql i (+ (expt (rem i 10) 5)
                     (expt (rem (floor i 10) 10) 5)
                     (expt (rem (floor i 100) 10) 5)
                     (expt (rem (floor i 1000) 10) 5)
                     (expt (rem (floor i 10000) 10) 5)
                     (expt (rem (floor i 100000) 10) 5)))
           (print i)
           (incf result i)))
   result))
```

**Problem 31**

In England the currency is made up of pound, £, and pence, p, and there are eight coins in general circulation:

1p, 2p, 5p, 10p, 20p, 50p, £1 (100p) and £2 (200p).

It is possible to make £2 in the following way:

1×£1 + 1×50p + 2×20p + 1×5p + 1×2p + 3×1p

How many different ways can £2 be made using any number of coins?

**Answer 31**

Done using recursion. Works fine for two pounds but is too inefficient for larger amounts. In that case I'd try the memoization.

```
(Defun Euler31 (Money &Optional (Coins '(200 100 50 20 10 5 2 1)))
       (If (Null (Cdr Coins))
         1  ; Only One Choice If Only One Coin Type Left
         (Let ((Ways 0))
            (Do ((Quant 0 (+ (Car Coins) Quant)))
                 ; Take Varying Amounts Of First Coin In List
                ((> Quant Money)) ; Stop When Too Many Coins
                (Incf Ways (Euler31 (- Money Quant) (Cdr Coins))))
           Ways)))
```

**Problem 32**

We shall say that an $n$-digit number is pandigital if it makes use of all the digits 1 to $n$ exactly once; for example, the 5-digit number, 15234, is 1 through 5 pandigital.

The product 7254 is unusual, as the identity, 39 × 186 = 7254, containing multiplicand, multiplier, and product is 1 through 9 pandigital.

Find the sum of all products whose multiplicand/multiplier/product identity can be written as a 1 through 9 pandigital.

HINT: Some products can be obtained in more than one way so be sure to only include it once in your sum.

**Answer 32**

My XLISP-PLUS solution, which runs in 73 milliseconds. Unlike the other LISP solutions on the Project Euler website, I do the pandigital check with just a

string. Also XLISP-PLUS doesn't (yet) have the LOOP macro, but I don't

think my solution suffers for it's absence.

Values can be 1 digit times 4 digit with a 4 digit product, in which case the 4 digit value would be at least 1234 but less than 4987 and the single digit must be greater than 1, or a 2 digit times 3 digit where the three digit number is at least 123 but less than or equal to 498 and the two digit number at least 12. We can also end the inner loop when the product exceeds 4 digits.

```
(defun euler32 ()
       (let (answer)
           (do ((i 2 (1+ i)))
              ((> i 9))
              (do* ((j 1234 (1+ j)) (prod (* i j)(* i j)))
                  ((> prod 9999))
                  (when
                   (equal "123456789"
                         (sort (format nil "~s~s~s" i j prod) #'char<))
;                   (format t "~s x ~s = ~s\n" i j prod)
                   (setq answer (adjoin prod answer)))))
           (do ((i 12 (1+ i)))
              ((> i 98))
              (do* ((j 123 (1+ j)) (prod (* i j) (* i j)))
                  ((> prod 9999))
                  (when
                   (equal "123456789"
                         (sort (format nil "~s~s~s" i j prod) #'char<))
;                   (format t "~s x ~s = ~s\n" i j prod)
                   (setf answer (adjoin prod answer)))))
           (apply #'+ answer)))
```

**Problem 33**

The fraction $^{49}/_{98}$ is a curious fraction, as an inexperienced mathematician in attempting to simplify it may incorrectly believe that $^{49}/_{98} = {}^{4}/_{8}$, which is correct, is obtained by canceling the 9s.

We shall consider fractions like, $^{30}/_{50} = {}^{3}/_{5}$, to be trivial examples.

There are exactly four non-trivial examples of this type of fraction, less than one in value, and containing two digits in the numerator and denominator.

If the product of these four fractions is given in its lowest common terms, find the value of the denominator.

**Answer 33**
420 microseconds.  Brute force with reasonable limits set for each digit. Simple nested loops but relies on LISP's rational arithmetic to automatically reduce the resulting fractions.

```
(defun Euler33 ()
       (Let ((Result 1))
       (Do ((J 1 (1+ J)))
         ((> J 9))
         (Do ((I 1 (1+ I)))
             ((>= I J))
             (Do ((K I (1+ K)))
               ((> K 9))
               (Let ((Ij (+ (* I 10) J))
                     (Jk (+ (* J 10) K)))
                  (When (Eql (* Ij K) (* Jk I))
                        (Format T "~S ~S \N" Ij Jk)
                        (Setf Result (* Result (/ I K))))))))))
       Result))
```

**Problem 34**
145 is a curious number, as 1! + 4! + 5! = 1 + 24 + 120 = 145.

Find the sum of all numbers which are equal to the sum of the factorial of their digits.

Note: as 1! = 1 and 2! = 2 are not sums they are not included.

**Answer 34**
I calculated an upper limit of 2540160. To speed up execution I made an array of 0 to 9 factorial and just did table lookups.

```
(require "fact")
(setq *factarray* (let ((result (list 1)))
                    (do ((i 1 (+ 1 i)))
                      ((> i 9))
                      (push (facti i) result))
                    (coerce (nreverse result) 'array)))
(defun euler34 ()
       (let ((result 0))
          (do ((i 3 (+ i 1)))
            ((> i 2540160))
            (when (eql i
```

```
                    (apply #'+ (map 'list
                                    #'(lambda (x) (aref *factarray* (- (char-int
x) 48)))
                                    (format nil "~s" i))))
                  (format t "~s\n" i)
                  (incf result i)))
          result))
```

**Problem 35**

The number, 197, is called a circular prime because all rotations of the digits: 197, 971, and 719, are themselves prime.

There are thirteen such primes below 100: 2, 3, 5, 7, 11, 13, 17, 31, 37, 71, 73, 79, and 97.

How many circular primes are there below one million?

**Answer 35**

I went through a list of all primes < 1,000,000, rotated them and checked for primality. The functions I used for primes only built a table up to the square-root of the number being tested, which isn't the best way to go for this problem.

```
(require "factor")

(defun is-circular-prime (n)
       (let* ((len (length (format nil "~s" n)))
            (dig (expt 10 (1- len))))
           (dotimes (i (1- len))
                  (setf n (let ((v (multiple-value-list (floor n dig))))
                           (+ (car v) (* 10 (cadr v)))))
                  (unless (primep n)
                        (return-from is-circular-prime nil)))
           t))

(defun euler35 ()
       (let ((count 0))
          (do ((i 2 (nextprime i)))
            ((> i 1000000))
            (when (is-circular-prime i)
                  (format t "~s\n" i)
                  (incf count)))
       count))
```

I rewrote it using a sieve to find the primes in advance. The prime number package I had previously written was optimized for finding the factors of individual very large numbers as opposed to finding the primeness of a large quantity of numbers. Time went from 11.65 seconds to 4.52 (for sieve) plus 1.53 seconds for the problem.

```
(defvar *notprimes*)
```

```
(defun sieve (n &aux (limit (/ n 2)))
       (setf *notprimes* (make-array n))
       (setf (aref *notprimes* 0) t)
       (setf (aref *notprimes* 1) t)
       (do ((i 2 (1+ i)))
         ((> i limit))
         (when (null (aref *notprimes* i))
             (do ((j (* 2 i) (+ j i)))
                 ((>= j n))
                 (setf (aref *notprimes* j) t)))))

(sieve 999999)

(defun is-circular-prime-a (n)
       (let* ((len (length (format nil "~s" n)))
            (dig (expt 10 (1- len))))
           (dotimes (i (1- len))
                  (setf n (let ((v (multiple-value-list (floor n dig))))
                            (+ (car v) (* 10 (cadr v)))))
                  (when (aref *notprimes* n)
                      (return-from is-circular-prime-a nil)))
           t))

(defun euler35a ()
       (let ((count 1)) ; start with value 2 already "found"
          (do ((i 3 (+ i 2)))
            ((> i 999998))
            (when (and (null (aref *notprimes* i))
                       (is-circular-prime-a i))
                  (format t "~s\n" i)
                  (incf count)))
       count))
```

**Problem 36**

The decimal number, 585 = $1001001001_2$ (binary), is palindromic in both bases.

Find the sum of all numbers, less than one million, which are palindromic in base 10 and base 2.

(Please note that the palindromic number, in either base, may not include leading zeros.)

**Answer 36**

We know the value must be odd. Then it's just a brute force approach.

```
(defun euler36 ()
       (let ((answer 0))
          (do* ((i 1 (+ i 2))
              (str10 "1" (format nil "~s" i)))
             ((> i 999999))
             (when (equal str10 (reverse str10))
                   (let ((str2 (format nil "~2r" i)))
                      (when (equal str2 (reverse str2))
```

```
                        (format t "~a ~a\n" str10 str2)
                        (incf answer i)))))
          answer))
```

**Problem 37**

The number 3797 has an interesting property. Being prime itself, it is possible to continuously remove digits from left to right, and remain prime at each stage: 3797, 797, 97, and 7. Similarly we can work from right to left: 3797, 379, 37, and 3.

Find the sum of the only eleven primes that are both truncatable from left to right and right to left.

NOTE: 2, 3, 5, and 7 are not considered to be truncatable primes.

**Answer 37**

Brute force. No upper limit needed to be set since problem statement said there were 11 values!

```
(require "factor")

(defun  euler37 ()
      (let ((result 0) (count 0))
           (do ((failflag nil nil)
              (i (nextprime 7) (nextprime i)))
             ((eql count 11) result)
             (do ((left (mod i (expt 10 (floor (log i 10))))
                        (mod left (expt 10 (floor (log left 10)))))
                  (right (floor i 10) (floor right 10)))
                 ((or failflag (zerop right)))
                 (unless (and (primep left) (primep right))
                       (setf failflag t)))
             (unless failflag
                   (format t "~s\n" i)
                   (incf result i)
                   (incf count)))
           result))
```

Then I rewrote using a sieve. Time was reduced from 8.49 to .84 seconds, but doing the sieve takes an additional 4.5. Additionally, this only works because I know the 11th value is less than a million.

```
(defun sieve (n)
       (setf *notprimes* (make-array n))
       (setf (aref *notprimes* 0) t)
       (setf (aref *notprimes* 1) t)
       (do ((i 2 (1+ i)))
         ((> i (/ n 2)))
         (when (null (aref *notprimes* i))
             (do ((j (* 2 i) (+ j i)))
```

```
                ((>= j n))
                (setf (aref *notprimes* j) t)))))

(sieve 999999)

(defun  euler37a ()
      (let ((result 0) (count 0))
           (do ((failflag nil nil)
              (i 11 (+ i 2)))
             ((eql count 11) result)
             (unless (aref *notprimes* i)
                   (do ((left (mod i (expt 10 (floor (log i 10))))
                              (mod left (expt 10 (floor (log left 10)))))
                        (right (floor i 10) (floor right 10)))
                       ((or failflag (zerop right)))
                       (when (or (aref *notprimes* left)
                                 (aref *notprimes* right))
                             (setf failflag t)))
                   (unless failflag
                         (format t "~s\n" i)
                         (incf result i)
                         (incf count))))
           result))
```

**Problem 38**
Take the number 192 and multiply it by each of 1, 2, and 3:

192  1 = 192

192  2 = 384

192  3 = 576

By concatenating each product we get the 1 to 9 pandigital, 192384576. We will call 192384576 the concatenated product of 192 and (1,2,3)

The same can be achieved by starting with 9 and multiplying by 1, 2, 3, 4, and 5, giving the pandigital, 918273645, which is the concatenated product of 9 and (1,2,3,4,5).

What is the largest 1 to 9 pandigital 9-digit number that can be formed as the concatenated product of an integer with (1,2, ... , n) where n  1?

**Answer 38**
By inspection we know that the sequence length must be 2. Reasoning:

1. Number must start with 9, otherwise 918273645 given in problem would be best.
2. For a sequence of 3 98 gives 8 digits and 921 gives 11 digits.
3. There are no possible values for a sequence of 4, 5 is shown, and 6 or more would likewise be impossible

4. For n=2 the answer must be in the range of 9123 through 9876. We start at the top and go down. There had better be a solution!

1.97 milliseconds.

```
(defun euler38 ()
       (let ((result nil))
          (do ((i 9876 (1- i)))
            (result result)
            (let ((string (format nil "~s~s" i (* 2 i))))
                 (when (equal (sort string #'char<)
                              "123456789")
                     ; we know we have 4 digit in front of 5 digit to get 9
                     (setf result (+ (* i 100000) (* 2 i))))))))
```

**Problem 39**

If p is the perimeter of a right angle triangle with integral length sides, {a,b,c}, there are exactly three solutions for p = 120.

{20,48,52}, {24,45,51}, {30,40,50}

For which value of p  1000, is the number of solutions maximized?

**Answer 39**

Knowing a*a + b*b = c*c and a+b+c=p, solved for b=p(p-2a)/2(p-a). By inspection we know that P is even and we only have to check 502 through 1000 inclusive. For each p I check out values of a > 1 and <= ceiling(p/(2+sqrt(2))). 66 milliseconds.

```
(defun euler39 ()
       (let ((bestp 0) (lenbestp 0))
          (do ((len 0 0)
             (p 502 (+ p 2)))
            ((> p 1000))
            (do ((a (ceiling (/ p 3.14159)) (1- a)))
                ((< a 2))
                (when (zerop (rem (* p (- p (* a 2))) (* 2 (- p a))))
                    (incf len)
                    ))
            (when (> len lenbestp)
                  (setf bestp p lenbestp len))
            )
          (list bestp lenbestp)))
```

**Problem 40**
An irrational decimal fraction is created by concatenating the positive integers:

0.12345678910111213141516171819202 1...

It can be seen that the 12th digit of the fractional part is 1.

If $d_n$ represents the $n^{th}$ digit of the fractional part, find the value of the following expression.

$$d1 \times d10 \times d100 \times d1000 \times d10000 \times d100000 \times d1000000$$

**Answer 40**
My first attempt involved creating a 1,000,000 character long list (XLISP-PLUS wasn't compiled to handle a 1,000,000 character long string and I didn't want to recompile.) Unfortunately it took too long to run. So I rewrote the solution to create the running product while generating the million digits, thus eliminating the need to save them. 0.23 seconds.

```
(defun euler40 ()
       (let ((result 1) (terms (list 1 10 100 1000 10000 100000 1000000)))
          (do* ((i 1 (1+ i))
              (pos 1 (incf pos (length (format nil "~s" i)))))
             ((null terms) result)
             (when (>= pos (car terms))
                   (let* ((val (format nil "~s" i))
                        (digit (-
                              (char-int (aref val (- (car terms)
                                                     (- pos (length val))
                                                     1)))
                              48)))
;                       (format t "~s\n" digit)
                       (setf result (* digit result))
                       (setf terms (cdr terms)))))))
```

**Problem 41**
We shall say that an n-digit number is pandigital if it makes use of all the digits 1 to n exactly once. For example, 2143 is a 4-digit pandigital and is also prime.

What is the largest n-digit pandigital prime that exists?

**Answer 41**
Ugly. I missed that n=9 and n=8 can't be solutions. Program errors out with those arguments. Sorry. But I've improved my style here by using flet.

```
(require "fact")
(require "factor")

; From problem 24, with modifications:

(defun
 euler41
 (n)
 (flet ((make-factorials (n)
                   (let (result)
                        (dotimes (i n)
                               (push (fact i) result))
                        result))
      (make-digits (n)
                 (let (result)
                    (do ((i n (1- i)))
                        ((zerop i))
                        (push i result))
                    (nreverse result)))
      (to-factoradic (n factorials)
                   (mapcar #'(lambda (f)
                               (let ((result (floor n f)))
                                    (decf n (* result f))
                                    result))
                           factorials))

      (permute (factoridic choices)
             (mapcar #'(lambda (digit) (let ((result (nth digit choices)))
                                    (setf choices (remove result choices))
                                    result))
                     factoridic)))
       ; body of euler41 code
       (let ((factorials (make-factorials n))
           (digits (make-digits n)))
          (do ((i 0 (1+ i)))
            ((let* ((val (permute (to-factoradic i factorials) digits))
                  (valstring (coerce (mapcar #'(lambda (x) (int-char (+ x
48)))
                                             val)
                                     'string))
                  (valnum (read-from-string valstring)))
                 (when (primep valnum)
                     (format t "Result is = ~s\n" valnum)
                     t)))))))
```

**Problem 42**

The $n^{th}$ term of the sequence of triangle numbers is given by, $t_n = \frac{1}{2}n(n+1)$; so the first ten triangle numbers are:

1, 3, 6, 10, 15, 21, 28, 36, 45, 55, ...

By converting each letter in a word to a number corresponding to its alphabetical position and adding these values we form a word value. For example, the word

value for SKY is 19 + 11 + 25 = 55 = $t_{10}$. If the word value is a triangle number then we shall call the word a triangle word.

Using words.txt (right click and 'Save Link/Target As...'), a 16K text file containing nearly two-thousand common English words, how many are triangle words?

**Answer 42**

4.6 msec. Used word value calculation from problem 22 and created a hashtable for quick lookup of triangle numbers. I did an initial (apply #'max (mapcar #'wordvalue +words+)) to see how big a table I'd need, but made it oversized anyway.

```
(defconstant +words+ '("A" "ABILITY" "ABLE" "ABOUT" "ABOVE" "ABSENCE"
```

A whole bunch of lines not shown...

```
"YEAH" "YEAR" "YES" "YESTERDAY" "YET" "YOU" "YOUNG" "YOUR" "YOURSELF"
"YOUTH"))

(defconstant +trianglenums+
           (let ((result (make-array 500)))
              (dotimes (i 30)
                     (setf (aref result (/ (* (1+ i) (+ 2 i)) 2)) t))
              result))


; taken from euler22
(defun wordvalue (w)
       (apply #'+ (map 'list #'(lambda (n) (- (char-int n) 64))
                       w)))


(defun euler42 ()
       (length (mapcan #'(lambda (w)
                           (when (aref +trianglenums+ (wordvalue w))
;                                (format t "~a\n" w)
                                 (list t)))
                       +words+)))
```

**Problem 43**

The number, 1406357289, is a 0 to 9 pandigital number because it is made up of each of the digits 0 to 9 in some order, but it also has a rather interesting sub-string divisibility property.

Let $d_1$ be the 1st digit, $d_2$ be the 2nd digit, and so on. In this way, we note the following:

- $d_2d_3d_4$=406 is divisible by 2
- $d_3d_4d_5$=063 is divisible by 3

- $d_4d_5d_6$=635 is divisible by 5
- $d_5d_6d_7$=357 is divisible by 7
- $d_6d_7d_8$=572 is divisible by 11
- $d_7d_8d_9$=728 is divisible by 13
- $d_8d_9d_{10}$=289 is divisible by 17

Find the sum of all 0 to 9 pandigital numbers with this property.

### Answer 43

Done probably the wrong way. I calculated all the pandigital values and then found out which met the criteria. I think the other way around would have been faster! Ran in 54 seconds and calculating the pandigital values alone took 50. Yep, bad algorithm!

```
(require "fact")

; I keep using the code from problem 23 that can go through all permutations

(defconstant factorials
        (let (result)
             (dotimes (i 10)
                      (push (fact i) result))
             result))

(defconstant digits '(0 1 2 3 4 5 6 7 8 9))

(defun to-factoradic (n)
       (mapcar #'(lambda (f)
                         (let ((result (floor n f)))
                              (decf n (* result f))
                              result))
               factorials))

(defun permute (factoradic choices)
       (map 'array #'(lambda (digit) (let ((result (nth digit choices)))
                                     (setf choices (remove result choices))
                                     result))
          factoradic))

(defun test-permute ()
       (do* ((i 0 (1+ i))
             (n (permute (to-factoradic i) digits)
               (permute (to-factoradic i) digits)))
           ((null (aref n 0)) i))) ;; reached end


(defun euler43 ()
       (let ((result 0))
           (do* ((i 0 (1+ i))
                 (n (permute (to-factoradic i) digits)
                    (permute (to-factoradic i) digits)))
               ((null (aref n 0)) result) ;; reached end
```

```
(when (and (evenp (aref n 3)) ; divisible by 2
           (eql (aref n 5) 5) ; divisible by 5 (and also deduction
                              ; that it can't be zero because of
                              ; divisibility by 11 test
           (zerop (rem (+ (* (aref n 7) 100) ; by 17
                          (* (aref n 8) 10)
                          (aref n 9)) 17))
           (zerop (rem (+ (aref n 2) (aref n 3) (aref n 4)) 3)) ; by 3
           (zerop (rem (+ (* (aref n 4) 100) ; by 7
                          (* (aref n 5) 10)
                          (aref n 6)) 7))
           (zerop (rem (+ (* (aref n 5) 100) ; by 11
                          (* (aref n 6) 10)
                          (aref n 7)) 11))
           (zerop (rem (+ (* (aref n 6) 100) ; by 13
                          (* (aref n 7) 10)
                          (aref n 8)) 13))
                                             )
      (let ((value (reduce #'(lambda (x y) (+ (* x 10) y)) n)))
           (format t "~s\n" value)
           (incf result value))))))
```

**Problem 44**

Pentagonal numbers are generated by the formula, $P_n=n(3n-1)/2$. The first ten pentagonal numbers are:

1, 5, 12, 22, 35, 51, 70, 92, 117, 145, ...

It can be seen that $P_4 + P_7 = 22 + 70 = 92 = P_8$. However, their difference, 70 - 22 = 48, is not pentagonal.

Find the pair of pentagonal numbers, $P_j$ and $P_k$, for which their sum and difference is pentagonal and $D = |P_k - P_j|$ is minimised; what is the value of D?

**Answer 44**

The real issue here is what bound is needed. Luckily it appears that the lowest bound that produces any answer produces the correct answer.

I was originally going to calculate pentagonals on the fly, but decided to create an array of pentagonals. So the calcpentagonal macro isn't really needed.

```
(defmacro calcpentagonal (v) `(floor (* ,v (- (* ,v 3) 1)) 2))

(defun pentagonalp (v) (zerop (rem (/ (+ (sqrt (+ (* v 24) 1)) 1) 6) 1)))

(defvar *pents* nil)

(defun fillpentarray (n) ; Create an array of pentagonal numbers
```

```
        (setf *pents* nil)
        (dotimes (i n)
             (push (calcpentagonal (1+ i)) *pents*))
        (setf *pents* (nreverse (coerce *pents* 'array))))


(defun euler44 (n) ; N is limit of number of pentagonal numbers to check
        (fillpentarray n)
        (let (besti bestj (diff (aref *pents* (1- n))))
           (do* ((i (1- n) (1- i)) ; go down from max
                 (ai (aref *pents* i) (aref *pents* i)))
                ((zerop i))
                (do ((j (1- i) (1- j))) ; go down from j
                    ((or (< j 0) ; either no match for this i or a winning value
                       (when (and (pentagonalp (- ai (aref *pents* j)))
                                  (pentagonalp (+ ai (aref *pents* j))))
                             (when (< (- ai (aref *pents* j)) diff)
                                   (setf diff (- ai (aref *pents* j))
                                       besti i
                                       bestj j))
                             t)))))
           (list besti bestj diff)))
```

**Problem 45**

Triangle, pentagonal, and hexagonal numbers are generated by the following formulae:

| | | |
|---|---|---|
| Triangle | $T_n=n(n+1)/2$ | 1, 3, 6, 10, 15, ... |
| Pentagonal | $P_n=n(3n-1)/2$ | 1, 5, 12, 22, 35, ... |
| Hexagonal | $H_n=n(2n-1)$ | 1, 6, 15, 28, 45, ... |

It can be verified that $T_{285} = P_{165} = H_{143} = 40755$.

Find the next triangle number that is also pentagonal and hexagonal.

**Answer 45**

I stepped through all the hexagonal numbers and at each step I stepped through triangle numbers until >= the hexagonal number and stepped through the pentagonal numbers until >= the hexagonal number. This seemed faster than solving for n in each of the triangle and pentagon formulas and finding those that were integers.

```
(defun euler45 ()
        (let ((nt 285) ; n for each of triangle, pentagonal, and hexagonal
            (np 165)
```

```
          (nh 143)
          (tn 40755) ; triangle, pentagonal, and hexagonal numbers
          (pn 40755)
          (hn 0))      ; set to zero so we don't get first match
         (do* ()
            ((and (eql tn pn) (eql pn hn)) tn)
            ; go to next hexagonal number
            (incf nh)
            (setf hn (* nh (- (* nh 2) 1)))
            ; Find next triangle number >= hn
            (do ()
                ((>= tn hn))
                (incf nt)
                (setf tn (floor (* nt (1+ nt)) 2)))
            ; Find next Pentagonal number >= hn
            (do ()
                ((>= pn hn))
                (incf np)
                (setf pn (floor (* np (1- (* np 3))) 2))))))
```

"After solving and reading the thread on projecteuler, I removed the triangle number generation/comparison and just used pentagonal and hexagonal. Execution time dropped from 138 msec to 71 msec.

```
(defun euler45a ()
       (let ((np 165)
           (nh 143)
           (pn 40755)
           (hn 0))      ; set to zero so we don't get first match
          (do* ()
            ((eql pn hn) pn)
            ; go to next hexagonal number
            (incf nh)
            (setf hn (* nh (- (* nh 2) 1)))
            ; Find next Pentagonal number >= hn
            (do ()
                ((>= pn hn))
                (incf np)
                (setf pn (floor (* np (1- (* np 3))) 2))))))
```

Going back and solving the quadratic equation for pentagonal numbers, the solution time did drop, to 41 msec.

```
(defun euler45b ()
       (do* ((nh 144 (1+ nh))
           (hn (* nh (- (* nh 2) 1)) (* nh (- (* nh 2) 1))))
          ((zerop (rem (/ (+ (sqrt (+ (* hn 24) 1)) 1) 6) 1))
           hn)))
```

**Problem 46**
It was proposed by Christian Goldbach that every odd composite number can be written as the sum of a prime and twice a square.

$9 = 7 + 2\times1^2$
$15 = 7 + 2\times2^2$
$21 = 3 + 2\times3^2$
$25 = 7 + 2\times3^2$
$27 = 19 + 2\times2^2$
$33 = 31 + 2\times1^2$

It turns out that the conjecture was false.

What is the smallest odd composite that cannot be written as the sum of a prime and twice a square?

**Answer 46**
nextprime and primep are convienient but not optimal for this sort of problem. Solution time of 1.85 seconds.

```
(require "factor")

(defun euler46 ()
       (do ((i 37 (+ i 2)))
         ()
         (when (null (primep i))
             (do ((j 2 (nextprime j)))
                 ((and (< j i) (zerop (rem (sqrt (/ (- i j) 2)) 1))))
                 (when (> j i)
                     (return-from euler46 i))))))
```

**Problem 47**
The first two consecutive numbers to have two distinct prime factors are:

$$14 = 2 \times 7$$
$$15 = 3 \times 5$$

The first three consecutive numbers to have three distinct prime factors are:

$$644 = 2^2 \times 7 \times 23$$
$$645 = 3 \times 5 \times 43$$
$$646 = 2 \times 17 \times 19.$$

Find the first four consecutive integers to have four distinct primes factors. What is the first of these numbers?

**Answer 47**

I used a sieve, modified to count the number of different factors (a trivial mod). Then the solution involved just finding 4 consecutive '4's. I allowed for up to about 1,000,000, for which it took 3.86 seconds. Cutting back to 200,000 reduced the time to .57 seconds.

```
;; Modified Sieve function from problems 35 and 37.
;; Gives count of distinct prime factors instead of just "T"

(defvar *notprimes*)

(defun modified-sieve (n &aux (limit (/ n 2)))
       (setf *notprimes* (make-array n :initial-element 0))
       (setf (aref *notprimes* 0) t)
       (setf (aref *notprimes* 1) t)
       (do ((i 2 (1+ i)))
         ((> i limit))
         (when (zerop (aref *notprimes* i))
             (do ((j (* 2 i) (+ j i)))
                 ((>= j n))
                 (incf (aref *notprimes* j))))))

(defun euler47 ()
       (modified-sieve 200000)
       (dotimes (i 199995)
            (when (and (eql (aref *notprimes* i) 4)
                       (eql (aref *notprimes* (1+ i)) 4)
                       (eql (aref *notprimes* (+ i 2)) 4)
                       (eql (aref *notprimes* (+ i 3)) 4))
                  (return-from euler47 i))))
```

**Problem 48**

The series, $1^1 + 2^2 + 3^3 + \ldots + 10^{10} = 10405071317$.

Find the last ten digits of the series, $1^1 + 2^2 + 3^3 + \ldots + 1000^{1000}$.

**Answer 48**

I wrote and executed the solution in under a minute. Why analyze the problem?

```
(defun euler48 ()
       (do ((result 0)
            (i 1 (1+ i)))
           ((eql i 1001)
            (rem result 10000000000))
           (incf result (expt i i))))
```

**Problem 49**
The arithmetic sequence, 1487, 4817, 8147, in which each of the terms increases by 3330, is unusual in two ways: (i) each of the three terms are prime, and, (ii) each of the 4-digit numbers are permutations of one another.

There are no arithmetic sequences made up of three 1-, 2-, or 3-digit primes, exhibiting this property, but there is one other 4-digit increasing sequence.

What 12-digit number do you form by concatenating the three terms in this sequence?

**Answer 49**
I interpreted the problem to be that the arithmetic sequence involved adding 3330 at each step, and the peculiar characteristic was the primeness and permutation. Then I made quick work of it.

```
(require "factor")

(defun euler49 ()
       (do ((i (nextprime 1487) (nextprime i)))
          ((> (+ i 6660) 9999))
          (when (and (primep (+ i 3330))
                     (primep (+ i 6660))
                     (let ((s1 (sort (format nil "~s" i) #'char<)))
                         (and (equal s1 (sort (format nil "~s" (+ i 3330)) #'char<))
                            (equal s1 (sort (format nil "~s" (+ i 6660)) #'char<)))))
               (format t "~s~s~s\n" i (+ i 3330) (+ i 6660)))))
```

**Problem 50**
The prime 41, can be written as the sum of six consecutive primes:

$$41 = 2 + 3 + 5 + 7 + 11 + 13$$

This is the longest sum of consecutive primes that adds to a prime below one-hundred.

The longest sum of consecutive primes below one-thousand that adds to a prime, contains 21 terms, and is equal to 953.

Which prime, below one-million, can be written as the sum of the most consecutive primes?

**Answer 50**
First version:

```
(require "factor")

(defun euler50 ( &aux primesums)
       ; First create a list of sums of consecutive primes, with "0" at the
       ; start of the list
       (setf primesums '(0))
       (do ((i 2 (nextprime i)))
          ((> i 1000000))
          (push (+ (first primesums) i) primesums))
       (setf primesums (nreverse (coerce primesums 'array)))
```

```
        (let ((maxseq 0) (prime 0) testval)
            (do ((i 1 (1+ i))) ;; loop through all sums (from "2") in list
               ((eql i (length primesums)))
               (do ((j (1- i) (1- j))) ;; loop for all preceding sums and
                                       ;; check the differences for being prime
                   ((or (< j 0)
                        (> (setf testval (- (aref primesums i)(aref primesums j)))
                           1000000)))
                   (when (and (primep testval) ;; If prime and longer sequence, use it
                              (> (- i j) maxseq))
                         (setf maxseq (- i j) prime testval))))
            (list (1+ maxseq) prime)))
```

Using a sieve speeds it up by a factor of 4.

```
(defvar *notprimes*)
(defvar *primesums*)

; Sieve modified to create sum of primes as well as the sieve
; which will be used for checking if a number is prime

(defun sieve (n &aux (limit (/ n 2)))
       (setf *notprimes* (make-array n))
       (setf *primesums* '(0))
       (setf (aref *notprimes* 0) t)
       (setf (aref *notprimes* 1) t)
       (do ((i 2 (1+ i)))
          ((> i limit))
          (when (null (aref *notprimes* i))
               (push (+ (first *primesums*) i) *primesums*)
               (do ((j (* 2 i) (+ j i)))
                   ((>= j n))
                   (setf (aref *notprimes* j) t))))
       (setf *primesums* (nreverse (coerce *primesums* 'array))))

(defun euler50a ()
       (sieve 999999)
       (let ((maxseq 0) (prime 0) testval)
           (do ((i 1 (1+ i))) ;; loop through all sums (from "2") in list
              ((eql i (length *primesums*)))
              (do ((j (1- i) (1- j))) ;; loop for all preceding sums and
                                      ;; check the differences for being prime
                  ((or (< j 0)
                       (> (setf testval (- (aref *primesums* i)(aref *primesums* j)))
                          999998)))
                  (when (and (null (aref *notprimes* testval)) ;; If prime and longer
sequence, use it
                             (> (- i j) maxseq))
                        (setf maxseq (- i j) prime testval))))
           (list (1+ maxseq) prime)))
```

**Problem 51**

By replacing the 1st digit of *3, it turns out that six of the nine possible values: 13, 23, 43, 53, 73, and 83, are all prime.

By replacing the 3rd and 4th digits of 56**3 with the same digit, this 5-digit number is the first example having seven primes among the ten generated numbers, yielding the family: 56003, 56113, 56333, 56443, 56663, 56773, and 56993. Consequently 56003, being the first member of this family, is the smallest prime with this property.

Find the smallest prime which, by replacing part of the number (not necessarily adjacent digits) with the same digit, is part of an eight prime value family.

### Answer 51

Not that difficult but for some reason I made lots of syntax errors. Straightforward -- generate the next prime, check for triplicate digit 0, 1, or 2 (since one of these must be in solution), then check that value substituting 1, 2, 3, ... 9 looking for 8 primes.

```
(load "factor")

(defun check8 (val dig)
       (let ((count 0)
           (lval (coerce val 'list)))
          (mapc #'(lambda (d)
                      (let ((v (read-from-string (coerce (subst d dig lval)
                                                         'string))))
                         (when (and (>= v 100000) (primep v))
                               (incf count))))
              '(#\0 #\1 #\2 #\3 #\4 #\5 #\6 #\7 #\8 #\9))
          (when (eql count 8)
              (format t "~s" val)
              t)))



(defun euler51 ()
       (do* ((i (nextprime 100000) (nextprime i))
           (found nil)
           (istr (format nil "~s" i) (format nil "~s" i)))
          ((or (> i 999999)
             (cond ((eql 3 (count #\0 istr))(check8 istr #\0))
                   ((eql 3 (count #\1 istr))(check8 istr #\1))
                   ((eql 3 (count #\2 istr))(check8 istr #\2))
                   (t nil))))))
```

### Problem 52

It can be seen that the number, 125874, and its double, 251748, contain exactly the same digits, but in a different order.

Find the smallest positive integer, *x*, such that 2*x*, 3*x*, 4*x*, 5*x*, and 6*x*, contain the same digits.

### Answer 52

Brute force approach, very clean looking and very fast (100 msec).

```
(defun euler52 ()
       (do* ((i 100000 (1+ i))
           (istr (sort (princ-to-string i) #'char<)
               (sort (princ-to-string i) #'char<)))
          ((> i 170000))
          (when (and (equal istr (sort (princ-to-string (* 2 i)) #'char<))
                     (equal istr (sort (princ-to-string (* 3 i)) #'char<))
                     (equal istr (sort (princ-to-string (* 4 i)) #'char<))
                     (equal istr (sort (princ-to-string (* 5 i)) #'char<))
```

```
                (equal istr (sort (princ-to-string (* 6 i)) #'char<)))
            (format t "~s" i)
            (return-from euler52))))
```

**Problem 53**
There are exactly ten ways of selecting three from five, 12345:

123, 124, 125, 134, 135, 145, 234, 235, 245, and 345

In combinatorics, we use the notation, ${}^5C_3 = 10$.

In general,

$${}^nCr = \frac{n!}{r!(n-r)!}$$ ,where $r \leq n$, $n! = n \times (n-1) \times \ldots \times 3 \times 2 \times 1$, and $0! = 1$.

It is not until $n = 23$, that a value exceeds one-million: ${}^{23}C_{10} = 1144066$.

How many, not necessarily distinct, values of ${}^nCr$, for $1 \leq n \leq 100$, are greater than one-million?

**Answer 53**
I could have used symmetry, simplify the calculation of n!/r!, or use memoization. But it ran so fast with brute force I just used that. 0.39 seconds. I did it two ways, the second builds a factorial table first (effectively memoized).

```
(require "fact")

(defun euler53 (&aux (result 0))
       (do ((n 1 (1+ n)))
         ((> n 100))
         (do ((r 1 (1+ r)))
             ((eql n r))
             (let ((nCr (floor (facti n) (* (facti r) (facti (- n r))))))
                (when (> ncr 1000000)
;                   (format t "~s ~s\n" n r)
                    (incf result)))))
       result)

(defun euler53a (&aux (result 0) (facts (make-array 101)))
       (setf (aref facts 0) 0)
       (setf (aref facts 1) 1)
       (do* ((i 2 (1+ i))
           (facti i (* i facti)))
          ((> i 100))
          (setf (aref facts i) facti))
       (do ((n 1 (1+ n)))
         ((> n 100))
         (do ((r 1 (1+ r)))
             ((eql n r))
             (let ((nCr (floor (aref facts n) (* (aref facts r)
                                                 (aref facts (- n r))))))
                (when (> ncr 1000000)
;                   (format t "~s ~s\n" n r)
```

```
                (incf result)))))
    result)
```

**Problem 54**
In the card game poker, a hand consists of five cards and are ranked, from lowest to highest, in the following way:

**High Card**: Highest value card.

**One Pair**: Two cards of the same value.

**Two Pairs**: Two different pairs.

**Three of a Kind**: Three cards of the same value.

**Straight**: All cards are consecutive values.

**Flush**: All cards of the same suit.

**Full House**: Three of a kind and a pair.

**Four of a Kind**: Four cards of the same value.

**Straight Flush**: All cards are consecutive values of same suit.

**Royal Flush**: Ten, Jack, Queen, King, Ace, in same suit.

The cards are valued in the order:
2, 3, 4, 5, 6, 7, 8, 9, 10, Jack, Queen, King, Ace.

If two players have the same ranked hands then the rank made up of the highest value wins; for example, a pair of eights beats a pair of fives (see example 1 below). But if two ranks tie, for example, both players have a pair of queens, then highest cards in each hand are compared (see example 4 below); if the highest cards tie then the next highest cards are compared, and so on.

Consider the following five hands dealt to two players:

| Hand | Player 1 | Player 2 | Winner |
|---|---|---|---|
| 1 | 5H 5C 6S 7S KD<br>Pair of Fives | 2C 3S 8S 8D TD<br>Pair of Eights | Player 2 |
| 2 | 5D 8C 9S JS AC<br>Highest card Ace | 2C 5C 7D 8S QH<br>Highest card Queen | Player 1 |
| 3 | 2D 9C AS AH AC<br>Three Aces | 3D 6D 7D TD QD<br>Flush with Diamonds | Player 2 |
| 4 | 4D 6S 9H QH QC<br>Pair of Queens<br>Highest card Nine | 3D 6D 7H QD QS<br>Pair of Queens<br>Highest card Seven | Player 1 |

| | | | |
|---|---|---|---|
| **5** | 2H 2D 4C 4D 4S<br>Full House<br>With Three Fours | 3C 3D 3S 9S 9D<br>Full House<br>with Three Threes | Player 1 |

The file, poker.txt, contains one-thousand random hands dealt to two players. Each line of the file contains ten cards (separated by a single space): the first five are Player 1's cards and the last five are Player 2's cards. You can assume that all hands are valid (no invalid characters or repeated cards), each player's hand is in no specific order, and in each hand there is a clear winner.

How many hands does Player 1 win?

**Answer 54**

27 milliseconds. I thought this a very straightforward problem. I read the file and parsed into hands of numeric card values and flushness. I assigned a score to each hand, checking in order 4-of-a-kind, full house, straight (including straight flush), flush, three-of-a-kind, two pair, pair, and high card.

It took longer to write than any others I've done so far, but was one of the most enjoyable.

```
(Defvar *spots* (make-hash-table :test #'eq))
(setf (gethash #\2 *spots*) 0)
(setf (gethash #\3 *spots*) 1)
(setf (gethash #\4 *spots*) 2)
(setf (gethash #\5 *spots*) 3)
(setf (gethash #\6 *spots*) 4)
(setf (gethash #\7 *spots*) 5)
(setf (gethash #\8 *spots*) 6)
(setf (gethash #\9 *spots*) 7)
(setf (gethash #\T *spots*) 8)
(setf (gethash #\J *spots*) 9)
(setf (gethash #\Q *spots*) 10)
(setf (gethash #\K *spots*) 11)
(setf (gethash #\A *spots*) 12)

(defun parse-hand (str)
       (list (gethash (aref str 0) *spots*)
            (gethash (aref str 3) *spots*)
            (gethash (aref str 6) *spots*)
            (gethash (aref str 9) *spots*)
            (gethash (aref str 12) *spots*)
            (let ((flushed (coerce (list
                                    (aref str 1)
                                    (aref str 4)
                                    (aref str 7)
                                    (aref str 10)
                                    (aref str 13))
                                   'string)))
               (or (equal flushed "SSSSS")
                   (equal flushed "HHHHH")
                   (equal flushed "DDDDD")
                   (equal flushed "CCCCC")))))

(defun evalhand (hand)
       ; We will rate hands:
       ; 0        high card
       ;   and (((c1*13+c2)*13+c3)*13+c4)*13+c5 to differentiate
       ; 1000000  one pair
```

```
;   and ((pair*13+c2)*13+c3)*13+c4 to differentiate
; 2000000  two pairs
;   and (hpair*13+lpair)*13+c5 to differentiate
; 3000000  three of a kind
;   and three to differentiate
; 4000000  straight
;   and topcard to differentiate
; 5000000  flush
;   and (((c1*13+c2)*13+c3)*13+c4)*13+c5 to differentiate
; 6000000  full house
;   and three to differentiate
; 7000000  four of a kind
;   and four to differentiate
; 8000000  straight/royal flush
;   and topcard to differentiate
(let ((flushed (sixth hand))
     (sorted (sort (cdr (reverse hand)) #'>)))
    (cond ((or (eql (first sorted) (fourth sorted)) ; 4 of a kind
               (eql (second sorted) (fifth sorted)))
           (+ 7000000 (second sorted)))
          ((or (and (eql (first sorted) (third sorted)) ; full house
                    (eql (fourth sorted) (fifth sorted)))
               (and (eql (first sorted) (second sorted))
                    (eql (third sorted) (fifth sorted))))
           (+ 6000000 (third sorted)))
          ((and (eql (first sorted) (1+ (second sorted))) ; straight
                (eql (second sorted) (1+ (third sorted)))
                (eql (third sorted) (1+ (fourth sorted)))
                (eql (fourth sorted) (1+ (fifth sorted))))
           (if flushed ; allow for straight flush
               (+ 8000000 (first sorted))
               (+ 4000000 (first sorted))))
          (flushed ; flush (other than straight flush)
           (+ (reduce #'(lambda (x y) (+ (* x 13) y)) sorted) 5000000))
          ((or (eql (first sorted) (third sorted)) ; three of a kind
               (eql (second sorted) (fourth sorted))
               (eql (third sorted) (fifth sorted)))
           (+ 3000000 (third sorted)))
          ((or (and (eql (first sorted) (second sorted)) ; Two pair
                    (or (eql (third sorted) (fourth sorted))
                        (eql (fourth sorted) (fifth sorted))))
               (and (eql (second sorted) (third sorted))
                    (eql (fourth sorted) (fifth sorted))))
           (+ 2000000 (* 13 (second sorted)) (fourth sorted)))
          ((eql (first sorted) (second sorted)) ; one pair, first possibility
           (+ (reduce #'(lambda (x y) (+ (* x 13) y)) (cdr sorted)) 1000000))
          ((eql (second sorted) (third sorted)) ; one pair, second possibility
           (+ (reduce #'(lambda (x y) (+ (* x 13) y))
                      (cons (second sorted)
                            (cons (first sorted) (cdddr sorted))))
              1000000))
          ((eql (third sorted) (fourth sorted)) ; one pair, third possibility
           (+ (reduce #'(lambda (x y) (+ (* x 13) y))
                      (list (third sorted)
                            (first sorted)
                            (second sorted)
                            (fifth sorted)))
              1000000))
          ((eql (fourth sorted) (fifth sorted)) ; one pair, fourth possibility
           (+ (reduce #'(lambda (x y) (+ (* x 13) y))
                      (list (fourth sorted)
                            (first sorted)
                            (second sorted)
                            (third sorted)))
```

```
                1000000))
              (t ; high card
                 (reduce #'(lambda (x y) (+ (* x 13) y)) sorted)))))



(defun readhands ()
       (let ((fp (open "poker.txt")) results)
           (do ((line (read-line fp nil) (read-line fp nil)))
              ((null line))
              (push (list (parse-hand line)
                          (parse-hand (subseq line 15)))
                    results))
           (close fp)
           results))

(defun euler54 ()
       (reduce #'+ (mapcar #'(lambda (x) (if (> (evalhand (first x))
                                                (evalhand (second x)))
                                             1
                                             0))
                           (readhands))))
```

**Problem 55**
If we take 47, reverse and add, 47 + 74 = 121, which is palindromic.

Not all numbers produce palindromes so quickly. For example,

349 + 943 = 1292,
1292 + 2921 = 4213
4213 + 3124 = 7337

That is, 349 took three iterations to arrive at a palindrome.

Although no one has proved it yet, it is thought that some numbers, like 196, never produce a palindrome. A number that never forms a palindrome through the reverse and add process is called a Lychrel number. Due to the theoretical nature of these numbers, and for the purpose of this problem, we shall assume that a number is Lychrel until proven otherwise. In addition you are given that for every number below ten-thousand, it will either (i) become a palindrome in less than fifty iterations, or, (ii) no one, with all the computing power that exists, has managed so far to map it to a palindrome. In fact, 10677 is the first number to be shown to require over fifty iterations before producing a palindrome: 4668731596684224866951378664 (53 iterations, 28-digits).

Surprisingly, there are palindromic numbers that are themselves Lychrel numbers; the first example is 4994.

How many Lychrel numbers are there below ten-thousand?

NOTE: Wording was modified slightly on 24 April 2007 to emphasise the theoretical nature of Lychrel numbers.

## Answer 55

286 milliseconds. Straight forward use of writing and reading from strings and the REVERSE function.

```
(defun reversed (val)
       (read-from-string (nreverse (princ-to-string val))))

(defun palindromep (val)
       (let ((str (princ-to-string val)))
            (equal str (reverse str))))

(defun euler55 ()
       (let ((result 0))
            (do ((i 1 (1+ i)))
                ((eql i 10000))
                (do ((iter 1 (1+ iter))
                     (val (+ i (reversed i)) (+ val (reversed val))))
                    ((or (palindromep val) (eql iter 50))
                     (when (eql iter 50)
;                          (format t "~s\n" i)
                           (incf result)
                           t))))
            result))
```

## Problem 56

A googol ($10^{100}$) is a massive number: one followed by one-hundred zeros; $100^{100}$ is almost unimaginably large: one followed by two-hundred zeros. Despite their size, the sum of the digits in each number is only 1.

Considering natural numbers of the form, $a^b$, where $a, b < 100$, what is the maximum digital sum?

## Answer 56

No assumptions made other than ignoring a,b < 2. It took 296 msec.

```
(defun euler56 ()
       (let ((result 0))
            (do ((a 2 (1+ a)))
                ((> a 99))
                (do ((b 2 (1+ b)))
                    ((> b 99))
                    (let* ((val (expt a b))
                           (sum (reduce #'+
                                        (map 'array
                                             #'(lambda (c) (- (char-int c) 48))
                                             (format nil "~s" val)))))
                          (when (> sum result)
;                               (format t "~s ~s ~s\n" a b sum)
                                (setf result sum)))))
            result))
```

**Problem 57**
It is possible to show that the square root of two can be expressed as an infinite continued fraction.

$$\sqrt{2} = 1 + 1/(2 + 1/(2 + 1/(2 + \ldots ))) = 1.414213\ldots$$

By expanding this for the first four iterations, we get:

1 + 1/2 = 3/2 = 1.5
1 + 1/(2 + 1/2) = 7/5 = 1.4
1 + 1/(2 + 1/(2 + 1/2)) = 17/12 = 1.41666...
1 + 1/(2 + 1/(2 + 1/(2 + 1/2))) = 41/29 = 1.41379...

The next three expansions are 99/70, 239/169, and 577/408, but the eighth expansion, 1393/985, is the first example where the number of digits in the numerator exceeds the number of digits in the denominator.

In the first one-thousand expansions, how many fractions contain a numerator with more digits than denominator?

**Answer 57**
1.56 seconds. Brute force using the built-in rational arithmetic.

```
(defun euler57 ()
       (let ((result 0))
          (do ((iter 1 (1+ iter))
               (val 3/2 (+ 1 (/ 1 (+ 1 val)))))
            ((> iter 1000))
            (when (> (length (princ-to-string (numerator val)))
                     (length (princ-to-string (denominator val))))
                  (incf result)))
          result))
```

**Problem 58**
Starting with 1 and spiraling anti-clockwise in the following way, a square spiral with side length 7 is formed.

```
37 36 35 34 33 32 31
38 17 16 15 14 13 30
39 18  5  4  3 12 29
40 19  6  1  2 11 28
41 20  7  8  9 10 27
42 21 22 23 24 25 26
43 44 45 46 47 48 49
```

It is interesting to note that the odd squares lie along the bottom right diagonal, but what is more interesting is that 8 out of the 13 numbers lying along both diagonals are prime; that is, a ratio of 8/13 ≈ 62%.

If one complete new layer is wrapped around the spiral above, a square spiral with side length 9 will be formed. If this process is continued, what is the side length of

the square spiral for which the ratio of primes along both diagonals first falls below 10%?

**Answer 58**
Based on knowledge from earlier problem. 6.5 seconds.

```
(require "factor")

(defun primepn (val) ; We want a primep that returns a numeric value
                     ; rather than T or NIL
       (if (primep val) 1 0))

(defun euler58 ()
       (do* ((primes 0 (+ primes
                          (primepn (+ dist n))
                          (primepn (+ dist (* n 2)))
                          (primepn (+ dist (* n 3))))) ; At n*4 it will never be prime
             (dist 1 (+ dist (* n 4))) ; Distance to start of new circle
             (n 2 (+ n 2)) ; length of side
             (ave 1.0 (/ primes (* n 2)))) ; Average set to 1 so we don't leave early
            ((< ave 0.1)
             (1+ n))))
```

**Problem 59**
Each character on a computer is assigned a unique code and the preferred standard is ASCII (American Standard Code for Information Interchange). For example, uppercase A = 65, asterisk (*) = 42, and lowercase k = 107.

A modern encryption method is to take a text file, convert the bytes to ASCII, then XOR each byte with a given value, taken from a secret key. The advantage with the XOR function is that using the same encryption key on the cipher text, restores the plain text; for example, 65 XOR 42 = 107, then 107 XOR 42 = 65.

For unbreakable encryption, the key is the same length as the plain text message, and the key is made up of random bytes. The user would keep the encrypted message and the encryption key in different locations, and without both "halves", it is impossible to decrypt the message.

Unfortunately, this method is impractical for most users, so the modified method is to use a password as a key. If the password is shorter than the message, which is likely, the key is repeated cyclically throughout the message. The balance for this method is using a sufficiently long password key for security, but short enough to be memorable.

Your task has been made easy, as the encryption key consists of three lower case characters. Using cipher1.txt (right click and 'Save Link/Target As...'), a file containing the encrypted ASCII codes, and the knowledge that the plain text must contain common English words, decrypt the message and find the sum of the ASCII values in the original text.

**Answer 59**
Brute force, checking for valid plaintext characters and the string " the ". Execution time is 1.73 seconds, including printing the plaintext.

```
(defconstant +text+ #(79 59 12 2 79 35 8 28 20 2 3 68 8 9 68 45 0 12 9
67 68 4 7 5 23 27 1 21 79 85 78 79 85 71 38 10 71 27 12 2 79 6 2 8 13
```

Lots of boring lines deleted...

```
71 7 13 20 79 3 11 0 22 30 67 68 19 7 1 71 8 8 8 29 29 71 0 2 71 27 12
2 79 11 9 3 29 71 60 11 9 79 11 1 79 16 15 10 68 33 14 16 15 10 22 73))

(defun successcalc (key &aux (result 0))
       (dotimes (i (length +text+))
                (let ((ch (logxor (aref key (mod i 3)) (aref +text+ i))))
                     (write-char (int-char ch))
                     (incf result ch)))
       (format t
               "Key=~a~a~a"
               (int-char (aref key 0))
               (int-char (aref key 1))
               (int-char (aref key 2)))
       result)

(defun isinvalid (ch) ; Not a valid alpha or space character
       (or (< ch 32) (> ch 126)))

(defun euler59 ()
       (let ((key #(0 0 0)) (lenm1 (1- (length +text+))))
            (dotimes
             (a 26)
             (setf (aref key 0) (+ a 97))
             (dotimes
              (b 26)
              (setf (aref key 1) (+ b 97))
              (dotimes
               (c 26)
               (setf (aref key 2) (+ c 97))
               (do* ((i 0 (1+ i))
                     (ch (logxor (aref key 0) (aref +text+ 0))
                         (logxor (aref key (mod i 3)) (aref +text+ i)))
                     (last5 (list 0 0 0 0 0) (nconc (cdr last5) (list ch)))
                     (marker nil (or marker (and (equal last5 '(32 116  104 101
32)))))
                     (fail (isinvalid ch)
                           (isinvalid ch)))
                    ((or fail (eql i lenm1))
                     (unless (or fail (null marker)) ; success!
                             (return-from euler59 (successcalc key)))))))))
```

**Problem 60**

The primes 3, 7, 109, and 673, are quite remarkable. By taking any two primes and concatenating them in any order the result will always be prime. For example, taking 7 and 109, both 7109 and 1097 are prime. The sum of these four primes, 792, represents the lowest sum for a set of four primes with this property.

Find the lowest sum for a set of five primes for which any two primes concatenate to produce another prime.

**Answer 60**

13 seconds to find (first) answer. Takes forever if I let it run. Brute force (nested loops) with "tricks" -- no use of strings and knowledge that pn mod 3 must equal pm mod 3.

```
(require "factor")

(defvar *pr* nil)
(defvar *prsize* nil)

(defun buildprimes (nmax)
       (setf *pr* nil *prsize* nil)
       (do ((i 3 (nextprime i)))
           ((> i nmax))
           (push i *pr*)
           (push (expt 10 (1+ (floor (log i 10)))) *prsize*))
       (setf *pr* (nreverse (coerce *pr* 'array)))
       (setf *prsize* (nreverse (coerce *prsize* 'array)))
       nil)


(defun euler60 (&optional (nmax 10000) &aux (result nil))
       (buildprimes nmax)
       (do ((i1 0 (1+ i1)))
           ((eql i1 (- (length *pr*) 4)))
           (do ((p1 (aref *pr* i1))
                (p1s (aref *prsize* i1))
                (i2 (1+ i1) (1+ i2)))
               ((eql i2 (- (length *pr*) 3)))
               (when (and (eql (mod p1 3) (mod (aref *pr* i2) 3))
                          (primep (+ (* p1 (aref *prsize* i2)) (aref *pr* i2)))
                          (primep (+ (* (aref *pr* i2) p1s) p1)))
                     (do ((p2 (aref *pr* i2))
                          (p2s (aref *prsize* i2))
                          (i3 (1+ i2) (1+ i3)))
                         ((eql i3 (- (length *pr*) 2)))
                         (when (and (eql (mod p1 3) (mod (aref *pr* i3) 3))
                                    (primep (+ (* p1 (aref *prsize* i3))
                                               (aref *pr* i3)))
                                    (primep (+ (* (aref *pr* i3) p1s) p1))
                                    (primep (+ (* p2 (aref *prsize* i3))
                                               (aref *pr* i3)))
                                    (primep (+ (* (aref *pr* i3) p2s) p2)))
                               (do ((p3 (aref *pr* i3))
                                    (p3s (aref *prsize* i3))
                                    (i4 (1+ i3) (1+ i4)))
                                   ((eql i4 (1- (length *pr*))))
                                   (when (and (eql (mod p1 3) (mod (aref *pr* i4) 3))
                                    (primep (+ (* p1 (aref *prsize* i4))
                                               (aref *pr* i4)))
                                    (primep (+ (* (aref *pr* i4) p1s) p1))
                                    (primep (+ (* p2 (aref *prsize* i4))
                                               (aref *pr* i4)))
                                    (primep (+ (* (aref *pr* i4) p2s) p2))
                                    (primep (+ (* p3 (aref *prsize* i4))
                                               (aref *pr* i4)))
                                    (primep (+ (* (aref *pr* i4) p3s) p3)))
                               (do ((p4 (aref *pr* i4))
                                    (p4s (aref *prsize* i4))
                                    (i5 (1+ i4) (1+ i5)))
                                   ((eql i5 (length *pr*)))
                                   (when (and (eql (mod p1 3) (mod (aref *pr* i5) 3))
                                              (primep (+ (* p1 (aref *prsize* i5))
                                                         (aref *pr* i5)))
                                              (primep (+ (* (aref *pr* i5) p1s) p1))
                                              (primep (+ (* p2 (aref *prsize* i5))
                                                         (aref *pr* i5)))
                                              (primep (+ (* (aref *pr* i5) p2s) p2))
                                              (primep (+ (* p3 (aref *prsize* i5))
```

```
                                        (aref *pr* i5)))
                                   (primep (+ (* (aref *pr* i5) p3s) p3))
                                   (primep (+ (* p4 (aref *prsize* i5))
                                              (aref *pr* i5)))
                                   (primep (+ (* (aref *pr* i5) p4s) p4)))
                              (let* ((p5 (aref *pr* i5))
                                     (sum (+ p1 p2 p3 p4 p5)))
                                (format t "~s ~s ~s ~s ~s =~s\n"
                                        p1 p2 p3 p4 p5 sum)
                                (return-from euler60)
                                (when (or (null result)
                                          (< sum result))
                                      (setf result sum))))))))))))
      result)
```

**Problem 61**

Triangle, square, pentagonal, hexagonal, heptagonal, and octagonal numbers are all figurate (polygonal) numbers and are generated by the following formulae:

| | | |
|---|---|---|
| Triangle | $P_{3,}n=n(n+1)/2$ | 1, 3, 6, 10, 15, ... |
| Square | $P_{4,}n=n^2$ | 1, 4, 9, 16, 25, ... |
| Pentagonal | $P_{5,}n=n(3n-1)/2$ | 1, 5, 12, 22, 35, ... |
| Hexagonal | $P_{6,}n=n(2n-1)$ | 1, 6, 15, 28, 45, ... |
| Heptagonal | $P_{7,}n=n(5n-3)/2$ | 1, 7, 18, 34, 55, ... |
| Octagonal | $P_{8,}n=n(3n-2)$ | 1, 8, 21, 40, 65, ... |

The ordered set of three 4-digit numbers: 8128, 2882, 8281, has three interesting properties.

The set is cyclic, in that the last two digits of each number is the first two digits of the next number (including the last number with the first).

Each polygonal type: triangle ($P_{3,127}$=8128), square ($P_{4,91}$=8281), and pentagonal ($P_{5,44}$=2882), is represented by a different number in the set.

This is the only set of 4-digit numbers with this property.

Find the sum of the only ordered set of six cyclic 4-digit numbers for which each polygonal type: triangle, square, pentagonal, hexagonal, heptagonal, and octagonal, is represented by a different number in the set.

**Answer 61**

3.6 msec. Use of recursion and fast executing mapping functions in LISP made for a nice problem and I feel nice solution.

```
(defun make-figurates ()
       (let (f3 f4 f5 f6 f7 f8)
            (do* ((i 10 (1+ i))
                  (q3 (floor (* i (1+ i)) 2)
                      (floor (* i (1+ i)) 2))
                  (q4 (* i i) (* i i))
                  (q5 (floor (* i (1- (* i 3))) 2)
                      (floor (* i (1- (* i 3))) 2))
                  (q6 (* i (1- (* i 2)))
                      (* i (1- (* i 2))))
```

```
                (q7 (floor (* i (- (* i 5) 3)) 2)
                    (floor (* i (- (* i 5) 3)) 2))
                (q8 (* i (- (* i 3) 2))
                    (* i (- (* i 3) 2))))
               ((> q3 9999))
               (when (and (> q3 999)
                          ;    (< q3 10000)
                          (> (mod (floor q3 10) 10) 0))
                     (push (list (floor q3 100)
                                 (mod q3 100)
                                 q3)
                           f3))
               (when (and (> q4 999)
                          (< q4 10000)
                          (> (mod (floor q4 10) 10) 0))
                     (push (list (floor q4 100)
                                 (mod q4 100)
                                 q4)
                           f4))
               (when (and (> q5 999)
                          (< q5 10000)
                          (> (mod (floor q5 10) 10) 0))
                     (push (list (floor q5 100)
                                 (mod q5 100)
                                 q5)
                           f5))
               (when (and (> q6 999)
                          (< q6 10000)
                          (> (mod (floor q6 10) 10) 0))
                     (push (list (floor q6 100)
                                 (mod q6 100)
                                 q6)
                           f6))
               (when (and (> q7 999)
                          (< q7 10000)
                          (> (mod (floor q7 10) 10) 0))
                     (push (list (floor q7 100)
                                 (mod q7 100)
                                 q7)
                           f7))
               (when (and (> q8 999)
                          (< q8 10000)
                          (> (mod (floor q8 10) 10) 0))
                     (push (list (floor q8 100)
                                 (mod q8 100)
                                 q8)
                           f8)))
         (list
          (nreverse f3)
          (nreverse f4)
          (nreverse f5)
          (nreverse f6)
          (nreverse f7)
          (nreverse f8)
          )))

(defun euler61r (n ; previous trailing digits
                 vals ; array of arrays of figurates
                 m ; first leading digits
                 )
       ; return list of figurates in chain
       (if (eql (length vals) 1)
           ; last one so must match on both sides
           (progn (mapc #'(lambda (x) (when (and (eql (first x) n)
```

```
                                              (eql (second x) m))
                                          (return-from euler61r (list (third x)))))
                        (car vals))
                  nil)
            ; not last, match to previous and recurse
            (do ((these (first vals) (first rests)) ; check out this list of figurates
                 (prevs nil (cons these prevs)) ; these we have checked to no avail
                 (rests (rest vals) (rest rests))) ; these are left to be checked
                ((null these)) ; return if none of the work
                (mapc #'(lambda (x) (when (eql (first x) n) ; this matches, so recurse
                                          (let ((val (euler61r (second x)
                                                               (append prevs rests)
                                                               m)))
                                            (when val
                                                  (return-from euler61r
                                                               (cons (third x) val))))))
                      these))))

(defun euler61 ()
       (let* ((figurates (make-figurates))
              (eights (car (last figurates)))
              (rest (butlast figurates)))
             (mapc #'(lambda (x) ; try each octagonal for a match
                             (let ((y (euler61r (second x)
                                                rest
                                                (first x))))
                               (when y (return-from euler61 (cons (third x) y)))))
                   eights)))
```

### Problem 62

The cube, 41063625 ($345^3$), can be permuted to produce two other cubes: 56623104 ($384^3$) and 66430125 ($405^3$). In fact, 41063625 is the smallest cube which has exactly three permutations of its digits which are also cube.

Find the smallest cube for which exactly five permutations of its digits are cube.

### Answer 62

40 msec. Simplest and probably fastest approach was to use a hash table indexed by the sorted string representation of the cube. For each i starting at 100, I calculate $i^3$ and its sorted string representation. I look up the string in a hash table and if not found I add a list of the cube and count=1 at that key. If found, I increment the count and if count=5 I stop and return the answer.

```
(defun euler62 ()
       (let ((hash (make-hash-table :size 997 :test #'equal)))
            (do* ((i 100 (1+ i))
                  (i3 (* i i i) (* i i i))
                  (i3s (sort (princ-to-string i3) #'char<)
                       (sort (princ-to-string i3) #'char<))
                  (intbl (gethash i3s hash)
                         (gethash i3s hash)))
                 ()
                 (if intbl
                     (when (eql (incf (cadr intbl)) 5)
                           (return-from euler62 (car intbl)))
                     (setf (gethash i3s hash)
                           (list i3 1))))))
```

**Problem 63**
The 5-digit number, $16807=7^5$, is also a fifth power. Similarly, the 9-digit number, $134217728=8^9$, is a ninth power.

How many *n*-digit positive integers exist which are also an *n*th power?

**Answer 63**
110 microseconds. Looped for base values 1 through 9 (10 to any power can't be successful), and within that looped for powers from 1 until the test failed. Easy problem! No logarithms.

```
(defun euler63 (&aux (result 0))
       (do ((n 1 (1+ n)))
         ((eql n 10) result)
         (do ((p 1 (1+ p)))
             ((null (eql p (length (princ-to-string (expt n p))))))
             (incf result))))
```

**Problem 64**
All square roots are periodic when written as continued fractions and can be written in the form:

$$\sqrt{N} = a_0 + \cfrac{1}{a_1 + \cfrac{1}{a_2 + \cfrac{1}{a_3 + \ldots}}}$$

For example, let us consider $\sqrt{23}$:

$$\sqrt{23} = 4 + \sqrt{23} - 4 = 4 + \cfrac{1}{\cfrac{1}{\sqrt{23}-4}} = 4 + \cfrac{1}{1 + \cfrac{\sqrt{23}-3}{7}}$$

If we continue we would get the following expansion:

$$\sqrt{23} = 4 + \cfrac{1}{1 + \cfrac{1}{3 + \cfrac{1}{1 + \cfrac{1}{8 + \ldots}}}}$$

The process can be summarised as follows:

$$a_0 = 4, \quad \cfrac{1}{\sqrt{23}-4} = \cfrac{\sqrt{23}+4}{7} = 1 + \cfrac{\sqrt{23}-3}{7}$$

$a_1 = 1, \quad \frac{7}{\sqrt{23}-3} = \frac{7(\sqrt{23}+3)}{14} = 3 + \frac{\sqrt{23}-3}{2}$

$a_2 = 3, \quad \frac{2}{\sqrt{23}-3} = \frac{2(\sqrt{23}+3)}{14} = 1 + \frac{\sqrt{23}-4}{7}$

$a_3 = 1, \quad \frac{7}{\sqrt{23}-4} = \frac{7(\sqrt{23}+4)}{7} = 8 + \sqrt{23}-4$

$a_4 = 8, \quad \frac{1}{\sqrt{23}-4} = \frac{\sqrt{23}+4}{7} = 1 + \frac{\sqrt{23}-3}{7}$

$a_5 = 1, \quad \frac{7}{\sqrt{23}-3} = \frac{7(\sqrt{23}+3)}{14} = 3 + \frac{\sqrt{23}-3}{2}$

$a_6 = 3, \quad \frac{2}{\sqrt{23}-3} = \frac{2(\sqrt{23}+3)}{14} = 1 + \frac{\sqrt{23}-4}{7}$

$a_7 = 1, \quad \frac{7}{\sqrt{23}-4} = \frac{7(\sqrt{23}+4)}{7} = 8 + \sqrt{23}-4$

It can be seen that the sequence is repeating. For conciseness, we use the notation $\sqrt{23} = [4;(1,3,1,8)]$, to indicate that the block (1,3,1,8) repeats indefinitely.

The first ten continued fraction representations of (irrational) square roots are:

$\sqrt{2}=[1;(2)]$, period=1

$\sqrt{3}=[1;(1,2)]$, period=2

$\sqrt{5}=[2;(4)]$, period=1

$\sqrt{6}=[2;(2,4)]$, period=2

$\sqrt{7}=[2;(1,1,1,4)]$, period=4

$\sqrt{8}=[2;(1,4)]$, period=2

$\sqrt{10}=[3;(6)]$, period=1

$\sqrt{11}=[3;(3,6)]$, period=2

$\sqrt{12}= [3;(2,6)]$, period=2

$\sqrt{13}=[3;(1,1,1,1,6)]$, period=5

Exactly four continued fractions, for $N \leq 13$, have an odd period.

How many continued fractions for $N \leq 10000$ have an odd period?

**Answer 64**

Straightforward, brute-force approach based on problem statement.

```
(defun cf-of-sqrt (n)
       (let ((sqrtn (sqrt n)))
          (do* ((a (floor sqrtn) (floor (+ sqrtn s) d))
                (s a (abs (- s (* a d))))
```

```
              (d (- n (* s s)) (truncate (- n (* s s)) d))
              (pregoal (cons s d))
              (goal nil pregoal) ; goal is first s,d but most not patch in
first iteration
              (cf 0 (1+ cf)))
             ((or (zerop d) ; rational -- end now!
                  (equal (cons s d) goal))
              cf))))

(defun euler64 (&aux (result 0))
       (do ((i 1 (1+ i)))
         ((> i 10000) result)
         (when (oddp (cf-of-sqrt i))
             (incf result))))
```

**Problem 65**

**Answer 65**

I wrote a function that evaluates continued fractions, then called it with the generated values for e. I took the numerator of the result and added up the digits. 4 milliseconds.

```
(defun eval-cont-frac (l)
       (do* ((list (reverse l) (cdr list))
             (element (first list) (first list))
             (val element (+ element (/ val))))
            ((null (rest list)) val)))

(defun euler65 ()
       (let ((l (list 2)))
          (dotimes (i 33) ; adds 33 * 3 additional terms
                 (setf l (cons 1 (cons (* 2 (1+ i)) (cons 1 l)))))
          (reduce #'+ (map 'list #'(lambda (c) (- (char-int c) 48))
                       (princ-to-string (numerator (eval-cont-frac l)))))))
```

**Problem 66**

**Answer 66**

Not Attempted

**Problem 67**

By starting at the top of the triangle below and moving to adjacent numbers on the row below, the maximum total from top to bottom is 23.

3
7 4
2 4 6
8 5 9 3

That is, 3 + 7 + 4 + 9 = 23.

Find the maximum total from top to bottom in triangle.txt (right click and 'Save Link/Target As...'), a 15K text file containing a triangle with one-hundred rows.

**NOTE:** This is a much more difficult version of Problem 18. It is not possible to try every route to solve this problem, as there are $2^{99}$ altogether! If you could check one trillion ($10^{12}$) routes every second it would take over twenty billion years to check them all. There is an efficient algorithm to solve it. ;o)

## Answer 67

```
(defconstant triangle67 #(
#(59)
#(73 41)
#(52 40 09)
#(26 53 06 34)
#(10 51 87 86 81)
#(61 95 66 57 25 68)
#(90 81 80 38 92 67 73)
#(30 28 51 76 81 18 75 44)
#(84 14 95 87 62 81 17 78 58)
#(21 46 71 58 02 79 62 39 31 09)
#(56 34 35 53 78 31 81 18 90 93 15)
#(78 53 04 21 84 93 32 13 97 11 37 51)
#(45 03 81 79 05 18 78 86 13 30 63 99 95)
#(39 87 96 28 03 38 42 17 82 87 58 07 22 57)
#(06 17 51 17 07 93 09 07 75 97 95 78 87 08 53)
#(67 66 59 60 88 99 94 65 55 77 55 34 27 53 78 28)
#(76 40 41 04 87 16 09 42 75 69 23 97 30 60 10 79 87)
#(12 10 44 26 21 36 32 84 98 60 13 12 36 16 63 31 91 35)
#(70 39 06 05 55 27 38 48 28 22 34 35 62 62 15 14 94 89 86)
#(66 56 68 84 96 21 34 34 34 81 62 40 65 54 62 05 98 03 02 60)
#(38 89 46 37 99 54 34 53 36 14 70 26 02 90 45 13 31 61 83 73 47)
#(36 10 63 96 60 49 41 05 37 42 14 58 84 93 96 17 09 43 05 43 06 59)
#(66 57 87 57 61 28 37 51 84 73 79 15 39 95 88 87 43 39 11 86 77 74 18)
#(54 42 05 79 30 49 99 73 46 37 50 02 45 09 54 52 27 95 27 65 19 45 26 45)
#(71 39 17 78 76 29 52 90 18 99 78 19 35 62 71 19 23 65 93 85 49 33 75 09 02)
#(33 24 47 61 60 55 32 88 57 55 91 54 46 57 07 77 98 52 80 99 24 25 46 78 79 05)
#(92 09 13 55 10 67 26 78 76 82 63 49 51 31 24 68 05 57 07 54 69 21 67 43 17 63 12)
#(24 59 06 08 98 74 66 26 61 60 13 03 09 09 24 30 71 08 88 70 72 70 29 90 11 82 41 34)
#(66 82 67 04 36 60 92 77 91 85 62 49 59 61 30 90 29 94 26 41 89 04 53 22 83 41 09 74
90)
#(48 28 26 37 28 52 77 26 51 32 18 98 79 36 62 13 17 08 19 54 89 29 73 68 42 14 08 16
70 37)
#(37 60 69 70 72 71 09 59 13 60 38 13 57 36 09 30 43 89 30 39 15 02 44 73 05 73 26 63
56 86 12)
#(55 55 85 50 62 99 84 77 28 85 03 21 27 22 19 26 82 69 54 04 13 07 85 14 01 15 70 59
89 95 10 19)
#(04 09 31 92 91 38 92 86 98 75 21 05 64 42 62 84 36 20 73 42 21 23 22 51 51 79 25 45
85 53 03 43 22)
#(75 63 02 49 14 12 89 14 60 78 92 16 44 82 38 30 72 11 46 52 90 27 08 65 78 03 85 41
57 79 39 52 33 48)
#(78 27 56 56 39 13 19 43 86 72 58 95 39 07 04 34 21 98 39 15 39 84 89 69 84 46 37 57
59 35 59 50 26 15 93)
#(42 89 36 27 78 91 24 11 17 41 05 94 07 69 51 96 03 96 47 90 90 45 91 20 50 56 10 32
36 49 04 53 85 92 25 65)
#(52 09 61 30 61 97 66 21 96 92 98 90 06 34 96 60 32 69 68 33 75 84 18 31 71 50 84 63
03 03 19 11 28 42 75 45 45)
#(61 31 61 68 96 34 49 39 05 71 76 59 62 67 06 47 96 99 34 21 32 47 52 07 71 60 42 72
94 56 82 83 84 40 94 87 82 46)
#(01 20 60 14 17 38 26 78 66 81 45 95 18 51 98 81 48 16 53 88 37 52 69 95 72 93 22 34
98 20 54 27 73 61 56 63 60 34 63)
```

#(93 42 94 83 47 61 27 51 79 79 45 01 44 73 31 70 83 42 88 25 53 51 30 15 65 94 80 44 61 84 12 77 02 62 02 65 94 42 14 94)
#(32 73 09 67 68 29 74 98 10 19 85 48 38 31 85 67 53 93 93 77 47 67 39 72 94 53 18 43 77 40 78 32 29 59 24 06 02 83 50 60 66)
#(32 01 44 30 16 51 15 81 98 15 10 62 86 79 50 62 45 60 70 38 31 85 65 61 64 06 69 84 14 22 56 43 09 48 66 69 83 91 60 40 36 61)
#(92 48 22 99 15 95 64 43 01 16 94 02 99 19 17 69 11 58 97 56 89 31 77 45 67 96 12 73 08 20 36 47 81 44 50 64 68 85 40 81 85 52 09)
#(91 35 92 45 32 84 62 15 19 64 21 66 06 01 52 80 62 59 12 25 88 28 91 50 40 16 22 99 92 79 87 51 21 77 74 77 07 42 38 42 74 83 02 05)
#(46 19 77 66 24 18 05 32 02 84 31 99 92 58 96 72 91 36 62 99 55 29 53 42 12 37 26 58 89 50 66 19 82 75 12 48 24 87 91 85 02 07 03 76 86)
#(99 98 84 93 07 17 33 61 92 20 66 60 24 66 40 30 67 05 37 29 24 96 03 27 70 62 13 04 45 47 59 88 43 20 66 15 46 92 30 04 71 66 78 70 53 99)
#(67 60 38 06 88 04 17 72 10 99 71 07 42 25 54 05 26 64 91 50 45 71 06 30 67 48 69 82 08 56 80 67 18 46 66 63 01 20 08 80 47 07 91 16 03 79 87)
#(18 54 78 49 80 48 77 40 68 23 60 88 58 80 33 57 11 69 55 53 64 02 94 49 60 92 16 35 81 21 82 96 25 24 96 18 02 05 49 03 50 77 06 32 84 27 18 38)
#(68 01 50 04 03 21 42 94 53 24 89 05 92 26 52 36 68 11 85 01 04 42 02 45 15 06 50 04 53 73 25 74 81 88 98 21 67 84 79 97 99 20 95 04 40 46 02 58 87)
#(94 10 02 78 88 52 21 03 88 60 06 53 49 71 20 91 12 65 07 49 21 22 11 41 58 99 36 16 09 48 17 24 52 36 23 15 72 16 84 56 02 99 43 76 81 71 29 39 49 17)
#(64 39 59 84 86 16 17 66 03 09 43 06 64 18 63 29 68 06 23 07 87 14 26 35 17 12 98 41 53 64 78 18 98 27 28 84 80 67 75 62 10 11 76 90 54 10 05 54 41 39 66)
#(43 83 18 37 32 31 52 29 95 47 08 76 35 11 04 53 35 43 34 10 52 57 12 36 20 39 40 55 78 44 07 31 38 26 08 15 56 88 86 01 52 62 10 24 32 05 60 65 53 28 57 99)
#(03 50 03 52 07 73 49 92 66 80 01 46 08 67 25 36 73 93 07 42 25 53 13 96 76 83 87 90 54 89 78 22 78 91 73 51 69 09 79 94 83 53 09 40 69 62 10 79 49 47 03 81 30)
#(71 54 73 33 51 76 59 54 79 37 56 45 84 17 62 21 98 69 41 95 65 24 39 37 62 03 24 48 54 64 46 82 71 78 33 67 09 16 96 68 52 74 79 68 32 21 13 78 96 60 09 69 20 36)
#(73 26 21 44 46 38 17 83 65 98 07 23 52 46 61 97 33 13 60 31 70 15 36 77 31 58 56 93 75 68 21 36 69 53 90 75 25 82 39 50 65 94 29 30 11 33 11 13 96 02 56 47 07 49 02)
#(76 46 73 30 10 20 60 70 14 56 34 26 37 39 48 24 55 76 84 91 39 86 95 61 50 14 53 93 64 67 37 31 10 84 42 70 48 20 10 72 60 61 84 79 69 65 99 73 89 25 85 48 92 56 97 16)
#(03 14 80 27 22 30 44 27 67 75 79 32 51 54 81 29 65 14 19 04 13 82 04 91 43 40 12 52 29 99 07 76 60 25 01 07 61 71 37 92 40 47 99 66 57 01 43 44 22 40 53 53 09 69 26 81 07)
#(49 80 56 90 93 87 47 13 75 28 87 23 72 79 32 18 27 20 28 10 37 59 21 18 70 04 79 96 03 31 45 71 81 06 14 18 17 05 31 50 92 79 23 47 09 39 47 91 43 54 69 47 42 95 62 46 32 85)
#(37 18 62 85 87 28 64 05 77 51 47 26 30 65 05 70 65 75 59 80 42 52 25 20 44 10 92 17 71 95 52 14 77 13 24 55 11 65 26 91 01 30 63 15 49 48 41 17 67 47 03 68 20 90 98 32 04 40 68)
#(90 51 58 60 06 55 23 68 05 19 76 94 82 36 96 43 38 90 87 28 33 83 05 17 70 83 96 93 06 04 78 47 80 06 23 84 75 23 87 72 99 14 50 98 92 38 90 64 61 58 76 94 36 66 87 80 51 35 61 38)
#(57 95 64 06 53 36 82 51 40 33 47 14 07 98 78 65 39 58 53 06 50 53 04 69 40 68 36 69 75 78 75 60 03 32 39 24 74 47 26 90 13 40 44 71 90 76 51 24 36 50 25 45 70 80 61 80 61 43 90 64 11)
#(18 29 86 56 68 42 79 10 42 44 30 12 96 18 23 18 52 59 02 99 67 46 60 86 43 38 55 17 44 93 42 21 55 14 47 34 55 16 49 24 23 29 96 51 55 10 46 53 27 92 27 46 63 57 30 65 43 27 21 20 24 83)
#(81 72 93 19 69 52 48 01 13 83 92 69 20 48 69 59 20 62 05 42 28 89 90 99 32 72 84 17 08 87 36 03 60 31 36 36 81 26 97 36 48 54 56 56 27 16 91 08 23 11 87 99 33 47 02 14 44 73 70 99 43 35 33)
#(90 56 61 86 56 12 70 59 63 32 01 15 81 47 71 76 95 32 65 80 54 70 34 51 40 45 33 04 64 55 78 68 88 47 31 47 68 87 03 84 23 44 89 72 35 08 31 76 63 26 90 85 96 67 65 91 19 14 17 86 04 71 32 95)
#(37 13 04 22 64 37 37 28 56 62 86 33 07 37 10 44 52 82 52 06 19 52 57 75 90 26 91 24 06 21 14 67 76 30 46 14 35 89 89 41 03 64 56 97 87 63 22 34 03 79 17 45 11 53 25 56 96 61 23 18 63 31 37 37 47)
#(77 23 26 70 72 76 77 04 28 64 71 69 14 85 96 54 95 48 06 62 99 83 86 77 97 75 71 66 30 19 57 90 33 01 60 61 14 12 90 99 32 77 56 41 18 14 87 49 10 14 90 64 18 50 21 74 14 16 88 05 45 73 82 47 74 44)

#(22 97 41 13 34 31 54 61 56 94 03 24 59 27 98 77 04 09 37 40 12 26 87 09 71 70 07 18
64 57 80 21 12 71 83 94 60 39 73 79 73 19 97 32 64 29 41 07 48 84 85 67 12 74 95 20 24
52 41 67 56 61 29 93 35 72 69)
#(72 23 63 66 01 11 07 30 52 56 95 16 65 26 83 90 50 74 60 18 16 48 43 77 37 11 99 98
30 94 91 26 62 73 45 12 87 73 47 27 01 88 66 99 21 41 95 80 02 53 23 32 61 48 32 43 43
83 14 66 95 91 19 81 80 67 25 88)
#(08 62 32 18 92 14 83 71 37 96 11 83 39 99 05 16 23 27 10 67 02 25 44 11 55 31 46 64
41 56 44 74 26 81 51 31 45 85 87 09 81 95 22 28 76 69 46 48 64 87 67 76 27 89 31 11 74
16 62 03 60 94 42 47 09 34 94 93 72)
#(56 18 90 18 42 17 42 32 14 86 06 53 33 95 99 35 29 15 44 20 49 59 25 54 34 59 84 21
23 54 35 90 78 16 93 13 37 88 54 19 86 67 68 55 66 84 65 42 98 37 87 56 33 28 58 38 28
38 66 27 52 21 81 15 08 22 97 32 85 27)
#(91 53 40 28 13 34 91 25 01 63 50 37 22 49 71 58 32 28 30 18 68 94 23 83 63 62 94 76
80 41 90 22 82 52 29 12 18 56 10 08 35 14 37 57 23 65 67 40 72 39 93 39 70 89 40 34 07
46 94 22 20 05 53 64 56 30 05 56 61 88 27)
#(23 95 11 12 37 69 68 24 66 10 87 70 43 50 75 07 62 41 83 58 95 93 89 79 45 39 02 22
05 22 95 43 62 11 68 29 17 40 26 44 25 71 87 16 70 85 19 25 59 94 90 41 41 80 61 70 55
60 84 33 95 76 42 63 15 09 03 40 38 12 03 32)
#(09 84 56 80 61 55 85 97 16 94 82 94 98 57 84 30 84 48 93 90 71 05 95 90 73 17 30 98
40 64 65 89 07 79 09 19 56 36 42 30 23 69 73 72 07 05 27 61 24 31 43 48 71 84 21 28 26
65 65 59 65 74 77 20 10 81 61 84 95 08 52 23 70)
#(47 81 28 09 98 51 67 64 35 51 59 36 92 82 77 65 80 24 72 53 22 07 27 10 21 28 30 22
48 82 80 48 56 20 14 43 18 25 50 95 90 31 77 08 09 48 44 80 90 22 93 45 82 17 13 96 25
26 08 73 34 99 06 49 24 06 83 51 40 14 15 10 25 01)
#(54 25 10 81 30 64 24 74 75 80 36 75 82 60 22 69 72 91 45 67 03 62 79 54 89 74 44 83
64 96 66 73 44 30 74 50 37 05 09 97 70 01 60 46 37 91 39 75 75 18 58 52 72 78 51 81 86
52 08 97 01 46 43 66 98 62 81 18 70 93 73 08 32 46 34)
#(96 80 82 07 59 71 92 53 19 20 88 66 03 26 26 10 24 27 50 82 94 73 63 08 51 33 22 45
19 13 58 33 90 15 22 50 36 13 55 06 35 47 82 52 33 61 36 27 28 46 98 14 73 20 73 32 16
26 80 53 47 66 76 38 94 45 02 01 22 52 47 96 64 58 52 39)
#(88 46 23 39 74 63 81 64 20 90 33 33 76 55 58 26 10 46 42 26 74 74 12 83 32 43 09 02
73 55 86 54 85 34 28 23 29 79 91 62 47 41 82 87 99 22 48 90 20 05 96 75 95 04 43 28 81
39 81 01 28 42 78 25 39 77 90 57 58 98 17 36 73 22 63 74 51)
#(29 39 74 94 95 78 64 24 38 86 63 87 93 06 70 92 22 16 80 64 29 52 20 27 23 50 14 13
87 15 72 96 81 22 08 49 72 30 70 24 79 31 16 64 59 21 89 34 96 91 48 76 43 53 88 01 57
80 23 81 90 79 58 01 80 87 17 99 86 90 72 63 32 69 14 28 88 69)
#(37 17 71 95 56 93 71 35 43 45 04 98 92 94 84 96 11 30 31 27 31 60 92 03 48 05 98 91
86 94 35 90 90 08 48 19 33 28 68 37 59 26 65 96 50 68 22 07 09 49 34 31 77 49 43 06 75
17 81 87 61 79 52 26 27 72 29 50 07 98 86 01 17 10 46 64 24 18 56)
#(51 30 25 94 88 85 79 91 40 33 63 84 49 67 98 92 15 26 75 19 82 05 18 78 65 93 61 48
91 43 59 41 70 51 22 15 92 81 67 91 46 98 11 11 65 31 66 10 98 65 83 21 05 56 05 98 73
67 46 74 69 34 08 30 05 52 07 98 32 95 30 94 65 50 24 63 28 81 99 57)
#(19 23 61 36 09 89 71 98 65 17 30 29 89 26 79 74 94 11 44 48 97 54 81 55 39 66 69 45
28 47 13 86 15 76 74 70 84 32 36 33 79 20 78 14 41 47 89 28 81 05 99 66 81 86 38 26 06
25 13 60 54 55 23 53 27 05 89 25 23 11 13 54 59 54 56 34 16 24 53 44 06)
#(13 40 57 72 21 15 60 08 04 19 11 98 34 45 09 97 86 71 03 15 56 19 15 44 97 31 90 04
87 87 76 08 12 30 24 62 84 28 12 85 82 53 99 52 13 94 06 65 97 86 09 50 94 68 69 74 30
67 87 94 63 07 78 27 80 36 69 41 06 92 32 78 37 82 30 05 18 87 99 72 19 99)
#(44 20 55 77 69 91 27 31 28 81 80 27 02 07 97 23 95 98 12 25 75 29 47 71 07 47 78 39
41 59 27 76 13 15 66 61 68 35 69 86 16 53 67 63 99 85 41 56 08 28 33 40 94 76 90 85 31
70 24 65 84 65 99 82 19 25 54 37 21 46 33 02 52 99 51 33 26 04 87 02 08 18 96)
#(54 42 61 45 91 06 64 79 80 82 32 16 83 63 42 49 19 78 65 97 40 42 14 61 49 34 04 18
25 98 59 30 82 72 26 88 54 36 21 75 03 88 99 53 46 51 55 78 22 94 34 40 68 87 84 25 30
76 25 08 92 84 42 61 40 38 09 99 40 23 29 39 46 55 10 90 35 84 56 70 63 23 91 39)
#(52 92 03 71 89 07 09 37 68 66 58 20 44 92 51 56 13 71 79 99 26 37 02 06 16 67 36 52
58 16 79 73 56 60 59 27 44 77 94 82 20 50 98 33 09 87 94 37 40 83 64 83 58 85 17 76 53
02 83 52 22 27 39 20 48 92 45 21 09 42 24 23 12 37 52 28 50 78 79 20 86 62 73 20 59)
#(54 96 80 15 91 90 99 70 10 09 58 90 93 50 81 99 54 38 36 10 30 11 35 84 16 45 82 18
11 97 36 43 96 79 97 65 40 48 23 19 17 31 64 52 65 65 37 32 65 76 99 79 34 65 79 27 55
33 03 01 33 27 61 28 66 08 04 70 49 46 48 83 01 45 19 96 13 81 14 21 31 79 93 85 50
05)
#(92 92 48 84 59 98 31 53 23 27 15 22 79 95 24 76 05 79 16 93 97 89 38 89 42 83 02 88
94 95 82 21 01 97 48 39 31 78 09 65 50 56 97 61 01 07 65 27 21 23 14 15 80 97 44 78 49

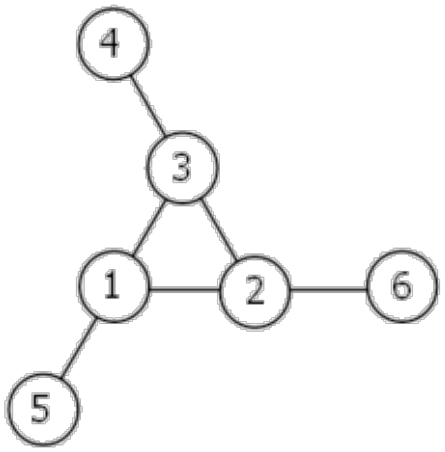
```
35 33 45 81 74 34 05 31 57 09 38 94 07 69 54 69 32 65 68 46 68 78 90 24 28 49 51 45 86
35)
#(41 63 89 76 87 31 86 09 46 14 87 82 22 29 47 16 13 10 70 72 82 95 48 64 58 43 13 75
42 69 21 12 67 13 64 85 58 23 98 09 37 76 05 22 31 12 66 50 29 99 86 72 45 25 10 28 19
06 90 43 29 31 67 79 46 25 74 14 97 35 76 37 65 46 23 82 06 22 30 76 93 66 94 17 96 13
20 72)
#(63 40 78 08 52 09 90 41 70 28 36 14 46 44 85 96 24 52 58 15 87 37 05 98 99 39 13 61
76 38 44 99 83 74 90 22 53 80 56 98 30 51 63 39 44 30 91 91 04 22 27 73 17 35 53 18 35
45 54 56 27 78 48 13 69 36 44 38 71 25 30 56 15 22 73 43 32 69 59 25 93 83 45 11 34 94
44 39 92)
#(12 36 56 88 13 96 16 12 55 54 11 47 19 78 17 17 68 81 77 51 42 55 99 85 66 27 81 79
93 42 65 61 69 74 14 01 18 56 12 01 58 37 91 22 42 66 83 25 19 04 96 41 25 45 18 69 96
88 36 93 10 12 98 32 44 83 83 04 72 91 04 27 73 07 34 37 71 60 59 31 01 54 54 44 96 93
83 36 04 45)
#(30 18 22 20 42 96 65 79 17 41 55 69 94 81 29 80 91 31 85 25 47 26 43 49 02 99 34 67
99 76 16 14 15 93 08 32 99 44 61 77 67 50 43 55 87 55 53 72 17 46 62 25 50 99 73 05 93
48 17 31 70 80 59 09 44 59 45 13 74 66 58 94 87 73 16 14 85 38 74 99 64 23 79 28 71 42
20 37 82 31 23)
#(51 96 39 65 46 71 56 13 29 68 53 86 45 33 51 49 12 91 21 21 76 85 02 17 98 15 46 12
60 21 88 30 92 83 44 59 42 50 27 88 46 86 94 73 45 54 23 24 14 10 94 21 20 34 23 51 04
83 99 75 90 63 60 16 22 33 83 70 11 32 10 50 29 30 83 46 11 05 31 17 86 42 49 01 44 63
28 60 07 78 95 40)
#(44 61 89 59 04 49 51 27 69 71 46 76 44 04 09 34 56 39 15 06 94 91 75 90 65 27 56 23
74 06 23 33 36 69 14 39 05 34 35 57 33 22 76 46 56 10 61 65 98 09 16 69 04 62 65 18 99
76 49 18 72 66 73 83 82 40 76 31 89 91 27 88 17 35 41 35 32 51 32 67 52 68 74 85 80 57
07 11 62 66 47 22 67)
#(65 37 19 97 26 17 16 24 24 17 50 37 64 82 24 36 32 11 68 34 69 31 32 89 79 93 96 68
49 90 14 23 04 04 67 99 81 74 70 74 36 96 68 09 64 39 88 35 54 89 96 58 66 27 88 97 32
14 06 35 78 20 71 06 85 66 57 02 58 91 72 05 29 56 73 48 86 52 09 93 22 57 79 42 12 01
31 68 17 59 63 76 07 77)
#(73 81 14 13 17 20 11 09 01 83 08 85 91 70 84 63 62 77 37 07 47 01 59 95 39 69 39 21
99 09 87 02 97 16 92 36 74 71 90 66 33 73 73 75 52 91 11 12 26 53 05 26 26 48 61 50 90
65 01 87 42 47 74 35 22 73 24 26 56 70 52 05 48 41 31 18 83 27 21 39 80 85 26 08 44 02
71 07 63 22 05 52 19 08 20)
#(17 25 21 11 72 93 33 49 64 23 53 82 03 13 91 65 85 02 40 05 42 31 77 42 05 36 06 54
04 58 07 76 87 83 25 57 66 12 74 33 85 37 74 32 20 69 03 97 91 68 82 44 19 14 89 28 85
85 80 53 34 87 58 98 88 78 48 65 98 40 11 57 10 67 70 81 60 79 74 72 97 59 79 47 30 20
54 80 89 91 14 05 33 36 79 39)
#(60 85 59 39 60 07 57 76 77 92 06 35 15 72 23 41 45 52 95 18 64 79 86 53 56 31 69 11
91 31 84 50 44 82 22 81 41 40 30 42 30 91 48 94 74 76 64 58 74 25 96 57 14 19 03 99 28
83 15 75 99 01 89 85 79 50 03 95 32 67 44 08 07 41 62 64 29 20 14 76 26 55 48 71 69 66
19 72 44 25 14 01 48 74 12 98 07)
#(64 66 84 24 18 16 27 48 20 14 47 69 30 86 48 40 23 16 61 21 51 50 26 47 35 33 91 28
78 64 43 68 04 79 51 08 19 60 52 95 06 68 46 86 35 97 27 58 04 65 30 58 99 12 12 75 91
39 50 31 42 64 70 04 46 07 98 73 98 93 37 89 77 91 64 71 64 65 66 21 78 62 81 74 42 20
83 70 73 95 78 45 92 27 34 53 71 15)
#(30 11 85 31 34 71 13 48 05 14 44 03 19 67 23 73 19 57 06 90 94 72 57 69 81 62 59 68
88 57 55 69 49 13 07 87 97 80 89 05 71 05 05 26 38 40 16 62 45 99 18 38 98 24 21 26 62
74 69 04 85 57 77 35 58 67 91 79 79 57 86 28 66 34 72 51 76 78 36 95 63 90 08 78 47 63
45 31 22 70 52 48 79 94 15 77 61 67 68)
#(23 33 44 81 80 92 93 75 94 88 23 61 39 76 22 03 28 94 32 06 49 65 41 34 18 23 08 47
62 60 03 63 33 13 80 52 31 54 73 43 70 26 16 69 57 87 83 31 03 93 70 81 47 95 77 44 29
68 39 51 56 59 63 07 25 70 07 77 43 53 64 03 94 42 95 39 18 01 66 21 16 97 20 50 90 16
70 10 95 69 29 06 25 61 41 26 15 59 63 35)
))

(defun row-summed-from-next (cur next)
       (let ((res (copy-seq cur))
             (reslen (length cur)))
            (dotimes (i reslen)
                     (incf (aref res i)
                           (max (aref next i) (aref next (1+ i)))))
            res
            ))
```

```
(defun euler67 (&aux (result (aref triangle67 (1- (length triangle67))))) "Bottom Up
solution"
       (dotimes (i (1- (length triangle67)))
                (setf result
                      (row-summed-from-next (aref triangle67 (- (length triangle67) i
2))
                                            result)))
       result)
```

**Problem 68**

Consider the following "magic" 3-gon ring, filled with the numbers 1 to 6, and each line adding to nine.



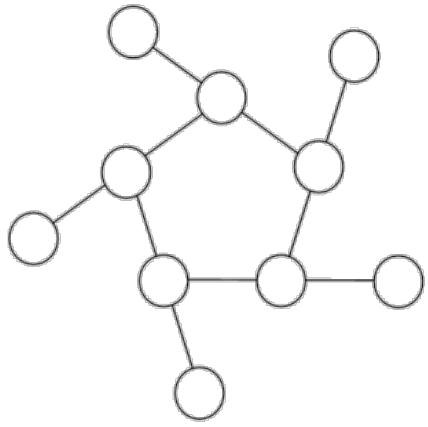

Working **clockwise**, and starting from the group of three with the numerically lowest external node (4,3,2 in this example), each solution can be described uniquely. For example, the above solution can be described by the set: 4,3,2; 6,2,1; 5,1,3.

It is possible to complete the ring with four different totals: 9, 10, 11, and 12. There are eight solutions in total.

| Total | Solution Set |
|---|---|
| 9 | 4,2,3; 5,3,1; 6,1,2 |
| 9 | 4,3,2; 6,2,1; 5,1,3 |
| 10 | 2,3,5; 4,5,1; 6,1,3 |
| 10 | 2,5,3; 6,3,1; 4,1,5 |
| 11 | 1,4,6; 3,6,2; 5,2,4 |
| 11 | 1,6,4; 5,4,2; 3,2,6 |
| 12 | 1,5,6; 2,6,4; 3,4,5 |
| 12 | 1,6,5; 3,5,4; 2,4,6 |

By concatenating each group it is possible to form 9-digit strings; the maximum string for a 3-gon ring is 432621513.

Using the numbers 1 to 10, and depending on arrangements, it is possible to form 16- and 17-digit strings. What is the maximum **16-digit** string for a "magic" 5-gon ring?



## Answer 68

I used the permutation functions from problem 23 (and others). Brute force except I know the 6 - 10 are on the outside and the first outside value needs to be 6. 29 msec.

```
(require "fact")

; I keep using the code from problem 23 that can go through all permutations

(defconstant factorials
        (let (result)
             (dotimes (i 5)
                      (push (fact i) result))
             result))

(defconstant odigits '(6 7 8 9 10))
(defconstant idigits '(1 2 3 4 5))

(defun to-factoradic (n)
       (mapcar #'(lambda (f)
                         (let ((result (floor n f)))
                              (decf n (* result f))
                              result))
               factorials))

(defun permute (factoradic choices)
       (map 'array #'(lambda (digit) (let ((result (nth digit choices)))
                                     (setf choices (remove result choices))
                                     result))
          factoradic))

(defconstant +dig+ #("0" "1" "2" "3" "4" "5" "6" "7" "8" "9" "10"))
; A "ring" is an array of 10 integers, 1 through 10, such that the first
; 5 are the outside points (and the first of those is 6) going clockwise,
; and the last five are the corresponding inside points. But we will split
; them into two arrays, outer and inner, for LISP efficiency

(defun correctringp (oring iring)
       (let ((val1 (+ (aref oring 0) (aref iring 0) (aref iring 1))))
           (and (eql val1 (+ (aref oring 1) (aref iring 1) (aref iring 2)))
```

```
              (eql val1 (+ (aref oring 2) (aref iring 2) (aref iring 3)))
              (eql val1 (+ (aref oring 3) (aref iring 3) (aref iring 4)))
              (eql val1 (+ (aref oring 4) (aref iring 4) (aref iring 0))))))

(defun evalring (oring iring)
       (read-from-string (concatenate 'string
                                  (aref +dig+ (aref oring 0))
                                  (aref +dig+ (aref iring 0))
                                  (aref +dig+ (aref iring 1))
                                  (aref +dig+ (aref oring 1))
                                  (aref +dig+ (aref iring 1))
                                  (aref +dig+ (aref iring 2))
                                  (aref +dig+ (aref oring 2))
                                  (aref +dig+ (aref iring 2))
                                  (aref +dig+ (aref iring 3))
                                  (aref +dig+ (aref oring 3))
                                  (aref +dig+ (aref iring 3))
                                  (aref +dig+ (aref iring 4))
                                  (aref +dig+ (aref oring 4))
                                  (aref +dig+ (aref iring 4))
                                  (aref +dig+ (aref iring 0)))))

(defun euler68 ( &aux (result 0))
       (dotimes (ol 24)
              (let ((ovals (permute (to-factoradic ol) odigits)))
                   (dotimes (il 120)
                          (let ((ivals (permute (to-factoradic il) idigits)))
                              (when (correctringp ovals ivals)
                                   (setf result
                                         (max result (evalring ovals ivals))))))))
       result)
```

**Problem 69**

Euler's Totient function, φ(*n*) [sometimes called the phi function], is used to determine the number of numbers less than *n* which are relatively prime to *n*. For example, as 1, 2, 4, 5, 7, and 8, are all less than nine and relatively prime to nine, φ(9)=6.

| n | Relatively Prime | φ(*n*) | *n*/φ(*n*) |
|---|---|---|---|
| 2 | 1 | 1 | 2 |
| 3 | 1,2 | 2 | 1.5 |
| 4 | 1,3 | 2 | 2 |
| 5 | 1,2,3,4 | 4 | 1.25 |
| 6 | 1,5 | 2 | 3 |
| 7 | 1,2,3,4,5,6 | 6 | 1.1666... |
| 8 | 1,3,5,7 | 4 | 2 |
| 9 | 1,2,4,5,7,8 | 6 | 1.5 |
| 10 | 1,3,7,9 | 4 | 2.5 |

It can be seen that *n*=6 produces a maximum *n*/φ(*n*) for $n \leq 10$.

Find the value of $n \leq 1{,}000{,}000$ for which *n*/φ(*n*) is a maximum.

**Answer 69**
From the example and a bit of intuition it can be seen that the maximum value can be obtained from the maximum number whose prime factors are one each of the maximum number of primes. Thus for values <= 10 we get 6 (2*3). 36 microseconds.

```
(require "factor")

(defun euler69 ( &aux (val 1))
       (do ((prime 2 (nextprime prime)))
           ((> (* val prime) 1000000) val)
           (setf val (* val prime))))
```

**Problem 70**

Euler's Totient function, φ($n$) [sometimes called the phi function], is used to determine the number of positive numbers less than or equal to $n$ which are relatively prime to $n$. For example, as 1, 2, 4, 5, 7, and 8, are all less than nine and relatively prime to nine, φ(9)=6.
The number 1 is considered to be relatively prime to every positive number, so φ(1)=1.

Interestingly, φ(87109)=79180, and it can be seen that 87109 is a permutation of 79180.

Find the value of $n$, $1 < n < 10^7$, for which φ($n$) is a permutation of $n$ and the ratio $n$/φ($n$) produces a minimum.

**Answer 70**
We know phi(n) = (p1 -1)(p2-1) where n=p1p2. This will produce the largest phi and hopefully the smallest n/phi. I'll find out if my answer is accepted. 1.21 seconds.

```
(defun sieve70 (n &aux (limit (floor n 2)) (notprimes (make-array n)) result)
       (do ((i 2 (1+ i)))
         ((> i limit))
         (when (null (aref notprimes i))
             (push i result)
             (do ((j (* 2 i) (+ j i)))
                 ((>= j n))
                 (setf (aref notprimes j) t))))
       (do ((i (if (oddp limit) (+ limit 2) (1+ limit)) (+ i 2)))
         ((>= i n))
         (when (null (aref notprimes i))
             (push i result)))
       (nreverse (coerce result 'array)))

(defun euler70 ()
       (let* ((limit 10000000)
```

```
            (sqrtl (floor (sqrt limit)))
            (primes (sieve70 (* sqrtl 2))) ; a good chunk of candidates
            (bestr 1000000) ; hope best ratio is less than this!
            (bestn nil)) ; The best n has the best (lowest) ratio
           (do ((i 0 (1+ i)))
             ((eql i (length primes)))
             (do ((j (1+ i) (1+ j)))
                 ((or (eql j (length primes))
                      (> (* (aref primes i) (aref primes j)) limit)))
                 (let ((phi (* (1- (aref primes i)) (1- (aref primes j))))
                       (n (* (aref primes i) (aref primes j))))
                   (when (and (equal (sort (princ-to-string phi) #'char<)
                                     (sort (princ-to-string n) #'char<))
                              (< (/ n phi) bestr))
;                        (format t "~s ~s ~s ~s\n" (aref primes i)
;                              (aref primes j) n phi (/ n phi))
                        (setf bestr (/ n phi))
                        (setf bestn n)))))
           bestn))
```

**Problem 71**

Consider the fraction, *n/d*, where *n* and *d* are positive integers. If $n<d$ and HCF(*n,d*)=1, it is called a reduced proper fraction.

If we list the set of reduced proper fractions for $d \leq 8$ in ascending order of size, we get:

1/8, 1/7, 1/6, 1/5, 1/4, 2/7, 1/3, 3/8, **2/5**, 3/7, 1/2, 4/7, 3/5, 5/8, 2/3, 5/7, 3/4, 4/5, 5/6, 6/7, 7/8

It can be seen that 2/5 is the fraction immediately to the left of 3/7.

By listing the set of reduced proper fractions for $d \leq 1{,}000{,}000$ in ascending order of size, find the numerator of the fraction immediately to the left of 3/7.

**Answer 71**

Made use of the information on the "Farey Sequence" to come up with a solution that took 190 msec. Not as efficient as it could be (double calculation of each (a+c)/(b+d)) but fast enough for the project. We know from the Farey sequence that the next value between a/b and c/d is (a+c)/(b+d)

```
(defun euler71 (&aux (num 1) (den 1))
       (do ((a 2 (numerator (/ (+ a c) (+ b d))))
          (b 5 (denominator (/ (+ a c) (+ b d))))
          (c 3)
          (d 7))
         ((> b 1000000) (list num den))
         (setf num a den b)))
```

**Problem 72**
Consider the fraction, *n/d*, where *n* and *d* are positive integers. If $n<d$ and HCF(*n,d*)=1, it is called a reduced proper fraction.

If we list the set of reduced proper fractions for $d \leq 8$ in ascending order of size, we get:

1/8, 1/7, 1/6, 1/5, 1/4, 2/7, 1/3, 3/8, 2/5, 3/7, 1/2, 4/7, 3/5, 5/8, 2/3, 5/7, 3/4, 4/5, 5/6, 6/7, 7/8

It can be seen that there are 21 elements in this set.

How many elements would be contained in the set of reduced proper fractions for $d \leq 1{,}000{,}000$?

**Answer 72**
dldmem - 49.4 seconds, after expand 100 19.76 seconds xldmem - 5.31 seconds after expand 100. Sure looks like xldmem is the way to go on systems with lots of memory (every system these days). Sum of phi(2) through phi(1,000,000). I made a sieve to calculate the phis. 5.3 seconds.

```
; Time to do a phi sieve!

(defvar *phi*)

(defun makesieve (n)
       (setf *phi* (make-array (1+ n)))
       (dotimes (i (1+ n)) (setf (aref *phi* i) i)) ; initialize phi array to identity
       (do ((i 2 (1+ i)))
           ((> i n))
           (when (eql (aref *phi* i) i) ; unmarked - a prime
                 (do ((j i (+ j i))
                      (newterm (/ (1- i) i)))
                     ((> j n))
                     (setf (aref *phi* j)
                           (* (aref *phi* j) newterm))))))


(defun euler72 ()
       (makesieve 1000000)
       (1- (reduce #'+ *phi*)) ; subtract 1 for erroneous phi(1)
       )
```

**Problem 74**
The number 145 is well known for the property that the sum of the factorial of its digits is equal to 145:

$$1! + 4! + 5! = 1 + 24 + 120 = 145$$

Perhaps less well known is 169, in that it produces the longest chain of numbers that link back to 169; it turns out that there are only three such loops that exist:

169 ⟶ 363601 ⟶ 1454 ⟶ 169
871 ⟶ 45361 ⟶ 871
872 ⟶ 45362 ⟶ 872

It is not difficult to prove that EVERY starting number will eventually get stuck in a loop. For example,

69 ⟶ 363600 ⟶ 1454 ⟶ 169 ⟶ 363601 (⟶ 1454)
78 ⟶ 45360 ⟶ 871 ⟶ 45361 (⟶ 871)
540 ⟶ 145 (⟶ 145)

Starting with 69 produces a chain of five non-repeating terms, but the longest non-repeating chain with a starting number below one million is sixty terms.

How many chains, with a starting number below one million, contain exactly sixty non-repeating terms?

**Answer 74**

I created two tables, one for factorials up to 9 and the second a hash table of values and their chain lengths. The table was seeded with all the numbers in the problem statement plus 1,2, and 40585 where are additional values with chain lengths of 1, like 145.

Then I just did a brute force solution going through sums of factorials of digits until I'd get a hit in the hash table.

The solution took 75 seconds, which is too long. So I modified the function to save all discovered chain lengths in the hash table during the processing. This new function, euler74a, gave the solution in 9.8 seconds.

```
; Make an array of factorials of digits
(require "fact")
(defconstant +facts+ (let ((val nil))
                   (dotimes (i 10) (push (facti i) val))
                   (nreverse (coerce val 'array))))

; Make a hash table of looping values
(defconstant +loopy+ (make-hash-table :size 999997))
(mapc #'(lambda (v w) (setf (gethash v +loopy+) w))
      '(1 2 40585 145 169 363601 1454 871 45361 872 45362) ; looping value
      '(1 1 1     1   3   3      3    2   2     2   2)  ; length of chain
      )

(defun sumfactdigits (n)
       (reduce #'+
               (map 'array
                    #'(lambda (v) (aref +facts+ (- (char-int v) 48)))
                    (princ-to-string n))))

(defun euler74 (&aux (result 0))
       (do ((i 69 (1+ i)))
           ((eql i 1000000) result)
           (do* ((inext i (sumfactdigits inext))
                 (count 0 (1+ count))
                 (hashval (gethash inext +loopy+) (gethash inext +loopy+)))
```

```
                (hashval
                 (when (eql 60 (+ count hashval))
                       (incf result))))))

(defun euler74a (&aux (result 0)) ; Increase size of loopy hash table
                                  ; by saving chain lengths
       (do ((i 69 (1+ i)))
           ((eql i 1000000) result)
           (do* ((inext (list i) (cons (sumfactdigits (car inext)) inext))
                 (count 0 (1+ count))
                 (hashval (gethash (car inext) +loopy+)
                          (gethash (car inext) +loopy+)))
                (hashval
                 (when (eql 60 (+ count hashval))
                       (incf result))
                 (mapc #'(lambda (v)
                                 (setf (gethash v +loopy+) (+ count hashval))
                                 (decf count))
                       (nreverse inext))))))
```

**Problem 75**

It turns out that 12 cm is the smallest length of wire that can be bent to form an integer sided right angle triangle in exactly one way, but there are many more examples.

> **12 cm**: (3,4,5)
> **24 cm**: (6,8,10)
> **30 cm**: (5,12,13)
> **36 cm**: (9,12,15)
> **40 cm**: (8,15,17)
> **48 cm**: (12,16,20)

In contrast, some lengths of wire, like 20 cm, cannot be bent to form an integer sided right angle triangle, and other lengths allow more than one solution to be found; for example, using 120 cm it is possible to form exactly three different integer sided right angle triangles.

> **120 cm**: (30,40,50), (20,48,52), (24,45,51)

Given that L is the length of the wire, for how many values of L $\leq$ 1,500,000 can exactly one integer sided right angle triangle be formed?

Note: This problem has been changed recently, please check that you are using the right parameters.

**Answer 75**

I had to look up algorithms for Pythagorean triples, then I make an array to keep count of the quantity for each length and finally counted how many lengths had only one triple. 1.5 seconds

```
(defun euler75 (&optional (maxval 1500000) &aux (res 0))
```

```
(let ((result (make-array (1+ maxval) :initial-element 0))
      (sqrtmaxval (floor (sqrt maxval))))
     (do ((i 1 (+ i 2)))
         ((> i sqrtmaxval))
         (do ((j 2 (+ j 2)))
             ((> j (- sqrtmaxval i)))
             (when (eql (gcd i j) 1)
                   (let ((circum (+ (abs (- (* j j) (* i i)))
                                    (* 2 i j)
                                    (* i i)
                                    (* j j))))
                        (do ((c circum (+ s circum)))
                            ((> c maxval))
                            (incf (aref result c))))))))
     (map nil #'(lambda (v) (when (eql v 1) (incf res))) result)
     res))
```

**Problem 76**
It is possible to write five as a sum in exactly six different ways:

4 + 1
3 + 2
3 + 1 + 1
2 + 2 + 1
2 + 1 + 1 + 1
1 + 1 + 1 + 1 + 1

How many different ways can one hundred be written as a sum of at least two positive integers?

**Answer 76**
Used problem 31 solution with every integer from 1 to 99 as a coin. However it took over 5 minutes!

```
(defun euler31 (money &optional (coins '(200 100 50 20 10 5 2 1)))
       (if (null (cdr coins))
         1  ; only one choice if only one coin type left
         (let ((ways 0))
            (do ((quant 0 (+ (car coins) quant))) ; take varying amounts of
                                                  ; first coin in list
                ((> quant money)) ; stop when too many coins
                (incf ways (euler31 (- money quant) (cdr coins))))
            ways)))

(defun euler76_slow (&optional (value 100) &aux coins)
       (dotimes (i (1- value)) (push (1+ i) coins))
       (euler31 value coins))
```

**Problem 77**
It is possible to write ten as the sum of primes in exactly five different ways:

7 + 3
5 + 5
5 + 3 + 2
3 + 3 + 2 + 2
2 + 2 + 2 + 2 + 2

What is the first value which can be written as the sum of primes in over five thousand different ways?

**Answer 77**
Used problem 31's answer here to solve, much like I used it for problem 76. I did have to change the problem 31 code to allow for the residual value not being divisible by the smallest coin/prime. The original problem had a 1 penny coin that could always be used for a solution. 270 milliseconds.

```
(defun euler31x (money &optional (coins '(200 100 50 20 10 5 2 1)))
       (if (null (cdr coins)) ; Only one coin left
         (if (zerop (rem money (car coins))) 1 0) ; One choice if divisible
                                                  ; by last coin
         (let ((ways 0))
            (do ((quant 0 (+ (car coins) quant))) ; take varying amounts of
                                                  ; first coin in list
                ((> quant money)) ; stop when too many coins
                (incf ways (euler31x (- money quant) (cdr coins))))
            ways)))


(defun euler77 ()
       (do* ((i 1 (1+ i)))
          ((> (euler31x i '(97 89 83 79 73 71 67 61 59 53 47 43 41 37 31 29
23 19 17 13 11 7 5 3 2))
            5000) i)))
```

**Problem 79**
A common security method used for online banking is to ask the user for three random characters from a passcode. For example, if the passcode was 531278, they may ask for the 2nd, 3rd, and 5th characters; the expected reply would be: 317.

The text file, keylog.txt, contains fifty successful login attempts.

Given that the three characters are always asked for in order, analyse the file so as to determine the shortest possible secret passcode of unknown length.

**Answer 79**
I did this by hand, but then went back to solve it in XLISP-PLUS since that's my personal goal. I converted the input to a list of lists of characters in each code. Then iterate over the following steps: 1. make a list of the unique first values of each code. 2. Prune that list of any values that appear beyond the first value of each code. 3. The remaining values in the list represent the next possible value(s) in the full passcode. 4. Remove these values from the list of lists. The iteration terminates as

soon as there are no possible values, which is the same as there being no more values in the list of lists. Execution time is 500 microseconds.

```
(defconstant +list+ '("319" "680" "180"
```

Lots of values omitted to keep this short

```
"716"))

(defun euler79 (&optional (samples +list+) &aux result)
       (let ((psamples (mapcar #'(lambda (s) (coerce s 'list)) samples)))

           (loop ; repeat until all processed
                (let ((possibles nil))
                     ; Get list of possible first values
                     (mapc #'(lambda (s) (setf possibles (adjoin (car s) possibles)))
                          psamples)
                     ; Get rid of nil that appears in list when a choice is complete
                     (setf possibles (delete nil possibles))
                     ; Check if we are done
                     (when (null possibles) (return-from euler79 result))
                     ; Diagnostics
;                    (format t "Possibles ~s\n" possibles)
;                    (y-or-n-p)
                     ; Eliminate those that cannot be first
                     (mapc #'(lambda (s)
                                  (setf possibles
                                       (set-difference possibles
                                                       (intersection (cdr s)
                                                                     possibles))))
                           psamples)
                     ; Must be at least one to choose from
                     (when (null possibles) (return-from euler79 "impossible"))
                     ; Report the value
;                    (format t "Next: ~a\n" possibles)
                     (setf result (append result possibles))
                     ; Eliminate it from the examples
                     (setf psamples
                          (mapcar #'(lambda (s) (reverse (set-difference s possibles)))
                                 psamples))
                     ))))
```

**Problem 89**

The rules for writing Roman numerals allow for many ways of writing each number (see About Roman Numerals...). However, there is always a "best" way of writing a particular number.

For example, the following represent all of the legitimate ways of writing the number sixteen:

IIIIIIIIIIIIIIII
VIIIIIIIIIII
VVIIIIII
XIIIIII
VVVI
XVI

The last example being considered the most efficient, as it uses the least number of numerals.

The 11K text file, roman.txt (right click and 'Save Link/Target As...'), contains one thousand numbers written in valid, but not necessarily minimal, Roman numerals; that is, they are arranged in descending units and obey the subtractive pair rule (see About Roman Numerals... for the definitive rules for this problem).

Find the number of characters saved by writing each of these in their minimal form.

Note: You can assume that all the Roman numerals in the file contain no more than four consecutive identical units.

### Answer 89

Uses the Common-LISP roman numeral output formatting. Unfortunately it doesn't go beyond 3999, so I had to modify it a bit. 8.9 milliseconds.

```
(defconstant +rn+ '( "MMMMDCLXXII" "MMDCCCLXXXIII" "MMMDLXVIIII"
"MMMMDXCV" "DCCCLXXII" "MMCCCVI" "MMMCDLXXXVII" "MMMMCCXXI" "MMMCCXX"
```

Lots of lines eliminated here to keep this short.

```
"MMXXVIII" "MMMMCCCXXX" "XXXXVIIII"))

(defconstant +rhash+ (make-hash-table))
(setf (gethash #\M +rhash+) 1000)
(setf (gethash #\D +rhash+) 500)
(setf (gethash #\C +rhash+) 100)
(setf (gethash #\L +rhash+) 50)
(setf (gethash #\X +rhash+) 10)
(setf (gethash #\V +rhash+) 5)
(setf (gethash #\I +rhash+) 1)


(defun evaluate-roman (str &aux (val 0) (maxdig 0))
       (mapc #'(lambda (d &aux (v (gethash d +rhash+)))
                       (if (>= v maxdig)
                           (progn
                                  (incf val v)
                                  (setf maxdig v))
                           (decf val v)))
             (nreverse (coerce str 'list)))
       val)


(defun euler89 ( &aux (result 0))
       (mapc #'(lambda (r &aux (evr (evaluate-roman r))
                          (better (if (> evr 3999)
                                      (format nil "M~@R" (- evr 1000))
                                      (format nil "~@R" evr))))
                       (incf result (- (length r) (length better))))
             +rn+)
       result)
```

**Problem 92**
A number chain is created by continuously adding the square of the digits in a number to form a new number until it has been seen before.

For example,

44 → 32 → 13 → 10 → **1** → **1**
85 → **89** → 145 → 42 → 20 → 4 → 16 → 37 → 58 → **89**

Therefore any chain that arrives at 1 or 89 will become stuck in an endless loop. What is most amazing is that EVERY starting number will eventually arrive at 1 or 89.

How many starting numbers below ten million will arrive at 89?

**Answer 92**
XLISP-PLUS, being interpreted, is at a disadvantage here. I got it down to 51 seconds by caching, like others have done.

```
(defun sq-of-digits (num)
       (reduce #'+ (map 'array #'(lambda (x &aux (dig (- (char-int x) 48)))
                                   (* dig dig))
                        (princ-to-string num))))

(defun euler92 ( &aux (result 0))
       (dotimes (i 10000000)
            (do ((v (sq-of-digits (1+ i)) (sq-of-digits v)))
                ((cond ((eql v 1) t)
                       ((eql v 89) (incf result))))))
       result)

; caching version
(defun euler92a ( &aux (result 0) (limit (* 9 9 7)) (shortcut (make-array (1+
limit))))
       (dotimes (i limit)
            (do ((v (sq-of-digits (1+ i)) (sq-of-digits v)))
                ((cond ((eql v 1) t)
                       ((eql v 89)
                        (setf (aref shortcut (1+ i)) t)
                        (incf result))))))
       (do ((i (1+ limit) (1+ i)))
         ((eql i 10000000))
         (when (aref shortcut (sq-of-digits i))
             (incf result)))

       result)
```

**Problem 97**
The first known prime found to exceed one million digits was discovered in 1999, and is a Mersenne prime of the form $2^{6972593}-1$; it contains exactly 2,098,960 digits. Subsequently other Mersenne primes, of the form $2p-1$, have been found which contain more digits.

However, in 2004 there was found a massive non-Mersenne prime which contains 2,357,207 digits: $28433 \times 2^{7830457}+1$.

Find the last ten digits of this prime number.

**Answer 97**

I Need To Optimize (Expt 2 N) In Xlisp-Plus. It Was Way To Slow. But (Ash 1 N) Works Great.

```
(Defun Euler97 ()
       (Mod (1+ (* 28433 (Ash 1 7830457))) 10000000000))
```