Einfache Probleme

Problem 1

Link zum Originalproblem (http://projecteuler.net/index.php?section=problems&id=1)

Wenn wir alle natürlichen Zahlen unter 10 betrachten, die Vielfache von 3 oder 5 sind, so erhalten wir 3, 5, 6 und 9. Die Summe dieser Vielfachen ist 23.

Aufgabe: Finde die Summe aller Vielfachen von 3 oder 5 unter 1000.

Mathematische Grundlagen

Lösung A

```
initialize
  ^ ((1 to: 999)
    select: [:each | (each isDivisibleBy: 3)
        or: [each isDivisibleBy: 5]]) sum
```

Lösung B

```
initialize
  | n summe |
  n := 999.
  summe := 0.
  1
    to: n
    do: [:i | (i \\ 3 = 0
          or: [i \\ 5 = 0])
        ifTrue: [summe := summe + i]].
  ^ summe
```

Lösung C

```
initialize
   | summe |
   summe := 0.
   0
      to: 999
      by: 3
      do: [:each | summe := summe + each].
   0
      to: 999
      by: 5
      do: [:each | summe := summe + each].
   0
      to: 999
      by: 15
      do: [:each | summe := summe - each].
   ^ summe
```

Lösung D

```
arithmeticSeriesOf: n upTo: limit
  | numberOfTerms |
  numberOfTerms := limit // n.
  ^ numberOfTerms * (n + (n * (numberOfTerms - 1) / 2))
```

```
initialize
  ^ (self arithmeticSeriesOf: 3 upTo: 999)
    + (self arithmeticSeriesOf: 5 upTo: 999)
    - (self arithmeticSeriesOf: 15 upTo: 999)
```

Lösung E (Rüdeger)

```
Integer->summeVon1BisN
   ^ self * (self + 1) // 2
```

```
Integer->summeVon1BisNmitK: schrittweite
   ^ schrittweite * (self // schrittweite) summeVon1BisN
```

```
initialize
   | n |
   n := 999.
   ^ (n summeVon1BisNmitK: 3)
      + (n summeVon1BisNmitK: 5)
```

```
    - (n summeVon1BisNmitK: 15)
```

Lösung F

```
initialize
   ^ (1 to: 999)
      inject: 0
      into: [:sum :i | i \\ 3 = 0 | (i \\ 5 = 0)
            ifTrue: [sum + i]
            ifFalse: [sum]]
```

Lösung G

```
initialize
  ^ (1 to: 999)
    detectSum: [:i | i \\ 3 = 0 | (i \\ 5 = 0)
        ifTrue: [i]
        ifFalse: [0]]
```

Lösung H

```
initialize
   ^ (1 to: 999)
      detectSum: [:i | i
            * (#(0 0 1 0 1 1 0 0 1 1 0 1 0 0 1 ) atWrap: i)]
```

Laufzeiten im Vergleich

Die Werte für die einzelnen Laufzeiten sollen nur eine Tendenz zeigen. Sie sind abhängig von Prozessor, Speicherausbau, virtueller Maschine und Image und können stark variieren.

Testumgebung (http://squeakde.cmsbox.ch/eulerprobleme-geloest/testumgebung)

Laufzeiten im Vergleich

für Zahlen im Bereich	Lösung A	Lösung B	Lösung C	Lösung D
1 bis 999	< 2 ms	< 2 ms	< 2 ms	< 2 ms
1 bis 1 Million	> 500 ms	> 300 ms	> 100 ms	< 2 ms
1 bis 10 Millionen	< 6 s	< 4 s	< 3 s	< 2 ms
1 bis 100 Millionen	25 s / Speicherwarnung	> 30 s	< 30 s	< 2 ms

Laufzeiten im Vergleich

für Zahlen im Bereich	Lösung E	Lösung F	Lösung G	Lösung H
1 bis 999	< 2 ms	< 2 ms	< 2 ms	< 2 ms
1 bis 1 Million	< 2 ms	< 2 s	> 2 s	> 2 s
1 bis 10 Millionen	< 2 ms	> 13 s	> 20 s	> 20 s
1 bis 100 Millionen	< 2 ms	< 300 s	< 300 s	< 300 s

Lesbarkeit, Speicherverbrauch und Geschwindigkeit im Vergleich

Die Lösungen D und E sind - wie oben ersichtlich - deutlich schneller als der Rest. Dafür ist die Mathematik dahinter auch schwerer zu verstehen. Einzelheiten kann man im oben verlinkten PDF nachlesen. Lösung H ist nicht nur relativ langsam, sondern auch schwer zu lesen. Für Lösung B, C, F und G steigt die Rechenzeit linear. Ähnliches gilt für Lösung A, welche aber zusätzlich den meisten Speicher benötigt. Da nützt bei größeren Zahlen auch die beste Lesbarkeit nichts.

Speicherverbrauch für Zahlen im Bereich 1 bis 10 Millionen

Lösung	Speicherverbrauch in MB
A	18
B	1
C	0.2
D	nicht messbar
E	nicht messbar
F	0.3
G	0.1
H	1.3

Einfache Probleme

Problem 2

Link zum Originalproblem (http://projecteuler.net/index.php?section=problems&id=2)

Jedes neue Glied der Fibonacci-Folge wird erzeugt, indem man die beiden vorangehenden Glieder addiert. Da man mit 1 und 2 beginnt, erhält man für die ersten 10 Glieder

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

Finde die Summe aller geraden Zahlen in dieser Folge, die die Obergrenze von vier Millionen nicht überschreitet.

Lösung A - für ganz Faule

```
initialize
   | generator sum next |
   sum := 0.
   generator := GeneratorTest new fibonacciSequence.
   [sum < 4000000]
      whileTrue: [next := generator next.
         next even
            ifTrue: [sum := sum + next]].
   ^ sum
```

Lösung B

```
initialize
   | generator sum |
   sum := 0.
   generator := GeneratorTest new fibonacciSequence.
   [sum < 4000000]
      whileTrue: [sum := sum + (generator next; next; next)].
   ^ sum
```

Anmerkung: Wenn man sich die ersten Terme der Fibonacci-Folge ansieht, so erkennt man, dass nur jeder dritte Term geradzahlig ist.

0, 1, 1, **2**, 3, 5, **8**, 13, 21, **34**, 55, 89, **144**, 233, 377, **610** ....

Man kann daher die zwei anderen Terme einfach verwerfen.

Lösung C (ohne Generator)

```
initialize
   | a b c n summe |
   n := 4 * 1000000.
   a := b := 1.
   c := a + b.
   summe := 0.
   [c < n]
      whileTrue: [summe := summe + c.
         a := b + c.
         b := a + c.
         c := a + b].
   ^ summe
```

Lösung D

```
folgt
```

Einfache Probleme

Problem 3

Link zum Originalproblem (http://projecteuler.net/index.php?section=problems&id=3)

Die Primfaktoren von 13195 sind 5, 7, 13 und 29.
Welches ist der größte Primfaktor der Zahl 600851475143 ?

Lösung A

```
Integer>>primfaktoren
  | liste n d s |
  liste := SortedCollection new.
  n := self.
  [2 * (n // 2) = n]
    whileTrue: [liste add: 2.
      n := n // 2].
  [3 * (n // 3) = n]
    whileTrue: [liste add: 3.
      n := n // 3].
  d := 5.
  s := 2.
  [d * d <= n]
    whileTrue: [[d * (n // d) = n]
        whileTrue: [liste add: d.
          n := n // d].
      d := d + s.
      s := 6 - s].
  n > 1
    ifTrue: [liste add: n].
  ^ liste
```

```
600851475143 primfaktoren max
```

Anmerkung: Die Methode primfaktoren wurde hier direkt für alle Exemplare der Klasse Integer definiert. Falls die Zahl n gerade ist, wird solange durch 2 geteilt und eine 2 zur Liste der Primfaktoren hinzugefügt, bis das Ergebnis nicht weiter durch zwei teilbar ist. Ebenso wird mit der 3 verfahren. Die 4 entfällt, da sie keine Primzahl ist und schon beim Durchlauf der 2 erfasst wurde. Der nächste zu prüfende Divisor ist daher die 5. Da Divisor und Dividend bei der Division austauschbar sind, braucht man nur Teiler bis Wurzel n berücksichtigen.

Alle Primzahlen größer 3 liegen in einer der beiden arithmetischen Folgen 6*n-1 (http://de.wikipedia.org/wiki/Primzahl#Eigenschaften_von_Primzahlen) (also: 5, 11, 17,) und 6*n+1 (http://de.wikipedia.org/wiki/Primzahl#Eigenschaften_von_Primzahlen) (also: 7, 13, 19,). Somit darf ab 5 der Probeteiler d abwechselnd um 2 und 4 vergrößert werden. Nachdem die Faktoren 2 und 3 herausgezogen sind, wird mit d = 5 und dem Schritt s = 2 begonnen. Die Zuweisung s := 6 s sorgt dafür, dass s abwechselnd die Werte 2 und 4 annimmt.

Einfache Probleme

Problem 4

Link zum Originalproblem (http://projecteuler.net/index.php?section=problems&id=4)

Palindromzahlen lassen sich spiegelbildlich lesen. Beispiele sind 747, 24642 oder 19844891.
Die größte Palindromzahl, die man bei der Multiplikation von zwei 2-stelligen Zahlen erhält, ist 9009. (91*99)

Finde die größte Palindromzahl, die man durch Multiplikation von zwei 3-stelligen Zahlen erhält.

Lösung A

```
Integer>>gespiegelt
   | zahl spiegelzahl |
   zahl := self.
   spiegelzahl := 0.
   [zahl > 0]
      whileTrue: [spiegelzahl := 10 * spiegelzahl + (zahl \\ 10).
         zahl := zahl // 10].
   ^ spiegelzahl
```

```
initialize
  | a b zuKlein max |
  a := 999.
  max := 0.
  [a >= 100]
    whileTrue: [b := 999.
      zuKlein := false.
      [b >= a
        and: [zuKlein not]]
        whileTrue: [| p |
          p := a * b.
          p <= max
            ifTrue: [zuKlein := true].
          p gespiegelt = p
            ifTrue: [max := p].
          b := b - 1].
      a := a - 1].
  ^ max
```

Anmerkung: Die Methode gespiegelt wurde hier direkt für alle Exemplare der Klasse Integer definiert.

Wir beginnen mit dem Produkt der beiden größten 3-stelligen Zahlen a (999) und b (999). Falls dieses Produkt nun eine Palindromzahl ergibt (was zu Beginn nicht der Fall ist) und sie vor allem nicht kleiner als ein möglicherweise bereits gefundenes Palindrom ist, so wird sie als aktuell größtes Palindrom betrachtet. Solange a kleiner ist als b, zählen wir b um eins herunter und wiederholen den Vorgang.

Anmerkung 2: Da bei der Multiplikation die Faktoren austauschbar sind, reicht es aus, das Produkt a * b nur für die Fälle a < b zu bilden. a * b = b * a.

Der gesamte Vorgang wird dann erneut wiederholt, allerdings mit nun verkleinertem a. Sobald a 2-stellig wird (<100), kann man die Suche abbrechen und das bis hierhin ermittelte größte Palindrom zurückgeben.

Einfache Probleme

Problem 5

[Link zum Originalproblem (http://projecteuler.net/index.php?section=problems&id=5)](http://projecteuler.net/index.php?section=problems&id=5)

Die kleinste Zahl, die sich durch n = 1, 2, … 9, 10 ohne Rest teilen lässt, ist 2520.

Welches ist die kleinste Zahl mit dieser Eigenschaft für n = 1, 2, … 19, 20?

Lösung A

```
initialize
  ^ (1 to: 20)
    inject: 1
    into: [:dividend :n | dividend := dividend / (dividend gcd: n) * n]
```

Einfache Probleme

Problem 6

Link zum Originalproblem (http://projecteuler.net/index.php?section=problems&id=6)

Die Summe der Quadrate der ersten zehn natürlichen Zahlen ist:

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

Das Quadrat der Summe der ersten zehn natürlichen Zahlen dagegen ist:

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

Die erste von der zweiten Summe abgezogen, ergibt 3025 – 385 = 2640.

Gesucht ist die entsprechende Differenz für die ersten hundert natürlichen Zahlen.

Lösung A

```
initialize
   ^(1 to: 100) sum squared - (1 to: 100) squared sum
```

Lösung B

```
calculateFor: nTerms
   ^ nTerms * (nTerms + 1) * (3 * nTerms squared - nTerms - 2) / 12
```

Anmerkung: Nimmt man die bereits aus Problem 1 bekannte Summenformel (http://de.wikipedia.org/wiki/Formelsammlung_Algebra#Arithmetische_Reihen) , zieht sie von der Potenzsumme (http://de.wikipedia.org/wiki/Formelsammlung_Algebra#Potenzsummen) ab und vereinfacht das Ganze, so erhält man obige Formel für beliebige n.

Einfache Probleme

Problem 7

Link zum Originalproblem (http://projecteuler.net/index.php?section=problems&id=7)

Beim Aufschreiben der ersten sechs Primzahlen 2, 3, 5, 7, 11 und 13 erkennen wir, dass 13 die 6-te Primzahl ist. Wie lautet die 10001-te Primzahl?

Lösung A

```
initialize
   | anzahl kandidat |
   anzahl := 1.
   kandidat := 1.
   [kandidat := kandidat + 2.
   kandidat isPrime
      ifTrue: [anzahl := anzahl + 1].
   anzahl < 10001] whileTrue.
   ^ kandidat
```

Einfache Probleme

Problem 8

[Link zum Originalproblem (http://projecteuler.net/index.php?section=problems&id=8)](http://projecteuler.net/index.php?section=problems&id=8)

Man finde das größte Produkt von fünf aufeinanderfolgenden Ziffern in einer 1000-stelligen Zahl

---

Lösung A

```
zahlwort
   ^ '
7316717653133062491922511967442657474235534919493496983520312774506326239578318016984801869478851843858615607891129494954595017379583319528532088055111254069874715852386305071569329096329522744304355766896648950445244523161731856403098711121722383113622298934233803081353362766142828064444866452387493035890729629049156044077239071381051585930796086670172427121883998797908792274921901699720888093776657273330010533678812202354218097512545405947522435258490771167055601360483958644670632441572215539753697817977846174064955149290862569321978468622482839722413756570560574902614079729686524145351004748216637048440319989000889524345065854122758866688116427171479924442928230863465674813919123162824586178664583591245665294765456828489128831426076900422421902267105562632111110937054421750694165896040807198403850962455444362981230987879927244284909188845801561660979191338754992005240636899125607176060588611646710940507754100225698315520005593572972571636269561882670428252483600823257530420752963450
'
```

```
initialize
   | produkt ziffer max ziffernblock wert stelle blockMax p |
   produkt := [:wort |
         p := 1.
         1
            to: wort size
            do: [:i |
               ziffer := (wort at: i) digitValue.
               p := p * ziffer].
         p].
   max := 1.
   1
      to: self zahlwort size - 4
      do: [:n |
         ziffernblock := self zahlwort copyFrom: n to: n + 4.
         wert := produkt value: ziffernblock.
         max < wert
            ifTrue: [max := wert.
               stelle := n.
               blockMax := ziffernblock]].
   ^ max
```

Einfache Probleme

Problem 9

[Link zum Originalproblem (http://projecteuler.net/index.php?section=problems&id=9)](http://projecteuler.net/index.php?section=problems&id=9)

Ein pythagoräisches Tripel besteht aus drei natürlichen Zahlen a, b, c mit $a^2 + b^2 = c^2$. Beispielsweise ist $3^2 + 4^2 = 25 = 5^2$. Es gibt genau ein solches Tripel mit a + b + c = 1000.
Finde das zugehörige Produkt a * b * c.

Lösung A

```
initialize
  | umfang |
  umfang := 1000.
  1
    to: umfang
    do: [:a | a
        to: umfang
        do: [:b |
          | c produkt |
          c := umfang - a - b.
          produkt := a * b * c.
          a squared + b squared = c squared
            ifTrue: [^ {a. b. c. produkt}]]]
```

Lösung B (premature optimization is the root of all evil)

```
initialize
  | upperLimitB a c |
  upperLimitB := 499.
  [upperLimitB > 1]
    whileTrueWithBreakAndContinue: [:break :continue |
      a := 500000 - (1000 * upperLimitB) / (1000 - upperLimitB).
      a isFraction
        ifTrue: [upperLimitB := upperLimitB - 1.
          continue value].
      c := 1000 - (a + upperLimitB).
      c squared = (a squared + upperLimitB squared)
        ifTrue: [break value].
      upperLimitB := upperLimitB - 1].
  ^ upperLimitB * a * c
```

Anmerkung: Die Werte für a, b und c müssen nur einmalig berechnet werden. Lösung A ist hinreichend schnell und deutlich lesbarer als die schnellere Lösung B, die sich folgenden Sachverhalt zu Nutze macht.

$a + b + c = 1000.$
$c^2 = a^2 + b^2.$
$c = a + b - 1000.$
$c^2 = (a + b - 1000)(a + b - 1000).$
$c^2 = a^2 + ab - 1000a + ab + b^2 - 1000b - 1000a - 1000b + 1000000.$
$a^2 + b^2 = a^2 + b^2 + 2ab - 2000a - 2000b + 1000000.$
$1000a - ab = 500000 - 1000b.$
$a(1000-b) = 500000 - 1000b.$
$a = (500000 - 1000b) / (1000 - b).$

Daraus folgt, dass b < 500 sein muss, da a sonst Null oder negativ werden würde. Da nach ganzzahligen Lösungen gesucht wird, kann man ohne Test mit dem nächsten b weitermachen, wenn a ein Bruch ist.

Lösung C (b wird immer um eins verringert, unabhängig vom isFraction-Test)

```
initialize
  | upperLimitB a c |
  upperLimitB := 500.
  [upperLimitB > 1]
    whileTrueWithBreakAndContinue: [:break :continue |
      upperLimitB := upperLimitB - 1.
      a := 500000 - (1000 * upperLimitB) / (1000 - upperLimitB).
      a isFraction
        ifTrue: [continue value].
      c := 1000 - (a + upperLimitB).
      c squared = (a squared + upperLimitB squared)
        ifTrue: [break value]].
  ^ upperLimitB * a * c
```

Einfache Probleme

Problem 10

Link zum Originalproblem (http://projecteuler.net/index.php?section=problems&id=10)

Die Summe der Primzahlen kleiner 10 ist 2 + 3 + 5 + 7 = 17. Finde die Summe aller Primzahlen kleiner zwei Millionen.

Lösung A

```
(Integer primesUpTo: 2000000) sum
```

Lösung B

```
initialize
   | grenze summe |
   grenze := 2000000.
   summe := 5.
   5
      to: grenze
      by: 2
      do: [:p | p isPrime
            ifTrue: [summe := summe + p]].
   ^ summe
```

Einfache Probleme

Problem 11

[Link zum Originalproblem (http://projecteuler.net/index.php?section=problems&id=11)](http://projecteuler.net/index.php?section=problems&id=11)

Wie lautet das größte Produkt vier aufeinanderfolgender Zahlen (in vertikaler, horizontaler oder diagonaler Richtung) innerhalb eines 20x20 Feldes?

Lösung A

```
initialize
   | max products |
   matrix := #(
   #(8 2 22 97 38 15 0 40 0 75 4 5 7 78 52 12 50 77 91 8 )
   #(49 49 99 40 17 81 18 57 60 87 17 40 98 43 69 48 4 56 62 0 )
   #(81 49 31 73 55 79 14 29 93 71 40 67 53 88 30 3 49 13 36 65 )
   #(52 70 95 23 4 60 11 42 69 24 68 56 1 32 56 71 37 2 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 3 45 2 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 2 62 12 20 95 63 94 39 63 8 40 91 66 49 94 21 )
   #(24 55 58 5 66 73 99 26 97 17 78 78 96 83 14 88 34 89 63 72 )
   #(21 36 23 9 75 0 76 44 20 45 35 14 0 61 33 97 34 31 33 95 )
   #(78 17 53 28 22 75 31 67 15 94 3 80 4 62 16 14 9 53 56 92 )
   #(16 39 5 42 96 35 31 47 55 58 88 24 0 17 54 24 36 29 85 57 )
   #(86 56 0 48 35 71 89 7 5 44 44 37 44 60 21 58 51 54 17 58 )
   #(19 80 81 68 5 94 47 69 28 73 92 13 86 52 17 77 4 89 55 40 )
   #(4 52 8 83 97 35 99 16 7 97 57 32 16 26 26 79 33 27 98 66 )
   #(88 36 68 87 57 62 20 72 3 46 33 67 46 55 12 32 63 93 53 69 )
   #(4 42 16 73 38 25 39 11 24 94 72 18 8 46 29 32 40 62 76 36 )
   #(20 69 36 41 72 30 23 88 34 62 99 69 82 67 59 85 74 4 36 16 )
   #(20 73 35 29 78 31 90 1 74 31 49 71 48 86 81 16 23 57 5 54 )
   #(1 70 54 71 83 51 54 69 16 92 33 48 61 43 52 1 89 19 67 48 ) ).
   products := Array new: 4.
   max := 0.
   (1 to: matrix first size)
      do: [:x | (1 to: matrix size)
            do: [:y |
               currentX := x.
               currentY := y.
               products at: 1 put: self computeRight.
               products at: 2 put: self computeDown.
               products at: 3 put: self computeDownstairsRight.
               products at: 4 put: self computeDownstairsLeft.
               products max > max
                  ifTrue: [max := products max]]].
   ^ max
```

```
computeDown
   currentY > (matrix size - 4)
      ifTrue: [^ 0]
      ifFalse: [^ (0 to: 3)
            inject: 1
            into: [:subTotal :next | subTotal
                  * ((matrix at: currentX) at: (currentY + next))]]
```

```
computeDownstairsLeft
   (currentX < 4
         or: [currentY > (matrix size - 4)])
      ifTrue: [^ 0]
      ifFalse: [^ (0 to: 3)
            inject: 1
            into: [:subTotal :next | subTotal
                  * ((matrix at: currentX - next) at: currentY + next)]]
```

```
computeDownstairsRight
   (currentX > (matrix first size - 4)
         or: [currentY > (matrix size - 4)])
      ifTrue: [^ 0]
      ifFalse: [^ (0 to: 3)
            inject: 1
            into: [:subTotal :next | subTotal
                  * ((matrix at: currentX + next) at: currentY + next)]]
```

```
computeRight
   currentX > (matrix first size - 4)
      ifTrue: [^ 0]
      ifFalse: [^ (0 to: 3)
```

```
inject: 1
into: [:subTotal :next | subTotal
      * ((matrix at: currentX + next) at: currentY)]]
```

Einfache Probleme

Problem 12

Link zum Originalproblem (http://projecteuler.net/index.php?section=problems&id=12)

Die Folge von Dreieckszahlen ist definiert als Summe der natürlichen Zahlen. So ist die siebente Dreieckszahl 28, da 1 + 2 + 3 + 4 + 5 + 6 + 7 = 28. Die zehn ersten Terme der Reihe sind:
1, 3, 6, 10, 15, 21, 28, 36, 45, 55, ...

Die Faktoren der ersten sieben Dreiecksnummern lauten:
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

Man kann sehen, dass 28 die erste Dreickszahl mit 5 Teilern ist.
Wie heißt die erste Dreickszahl, die mehr als 500 Teiler besitzt?

Lösung A

```
Integer->teilerzahl
   | anzahl probeteiler gegenteiler |
   self = 0
      ifTrue: [^ 0].
   self = 1
      ifTrue: [^ 1].
   anzahl := 2.
   probeteiler := 2.
   gegenteiler := self // 2.
   [probeteiler < gegenteiler]
      whileTrue: [probeteiler * gegenteiler = self
            ifTrue: [anzahl := anzahl + 2].
         probeteiler := probeteiler + 1.
         gegenteiler := self // probeteiler].
   "whileTrue"
   probeteiler * probeteiler = self
      ifTrue: [anzahl := anzahl + 1].
   ^ anzahl
```

```
initialize
   | n tz dz |
   n := 1.
   tz := 0.
   [tz < 500]
      whileTrue: [dz := n * (n + 1) / 2.
         tz := dz teilerzahl.
         n := n + 1]
```

Einfache Probleme

Problem 13

[Link zum Originalproblem (http://projecteuler.net/index.php?section=problems&id=13)](http://projecteuler.net/index.php?section=problems&id=13)

Wie lauten die ersten 10 Ziffern der Summe der folgenden 100 50-stelligen Zahlen?

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

Lösung A

```
initialize

| aStream |

aStream :=  (FileStream readOnlyFileNamed: '/Users/Enno/Problem13.txt').

^((1 to: 100) detectSum: [:each | (aStream next: 50) asInteger]) asString first: 10.
```

Einfache Probleme

Problem 14

[Link zum Originalproblem (http://projecteuler.net/index.php?section=problems&id=14)](http://projecteuler.net/index.php?section=problems&id=14)

Die Collatz-Sequenz ist wie folgt für positive ganze Zahlen definiert:

Regel 1: n = n / 2 (falls n gerade)
Regel 2: n = 3n + 1 (falls n ungerade)

Beim Anwenden der obigen Regeln mit dem Startwert 13, erhalten wir folgende Sequenz:

13 40 20 10 5 16 8 4 2 1

Man kann sehen, dass diese Sequenz 10 Terme enthält. Obwohl es keinen Beweis gibt, geht man davon aus, dass jeder beliebige Startwert irgendwann zur 1 führt.

Welcher Startwert unter einer Million erzeugt die längste Sequenz?

ANMERKUNG: Nach dem Start sind Werte über einer Million zulässig.

Lösung A

```
initialize
   | currentSize maxNumber aCollection maxCollectionSize |
   aCollection := OrderedCollection new.
   maxNumber := 0.
   maxCollectionSize := 0.
   currentSize := 0.
   (999999 to: 1 by: -1)
      do: [:each |
         currentSize := (self track: each) size.
         currentSize > maxCollectionSize
            ifTrue: [maxCollectionSize := currentSize.
               maxNumber := each]].
   ^ maxNumber
```

```
track: aNumber
   | currentNumber aCollection |
   aCollection := OrderedCollection new.
   currentNumber := aNumber.
   [currentNumber = 1]
      whileFalse: [aCollection add: currentNumber.
         currentNumber even
            ifTrue: [currentNumber := currentNumber / 2]
            ifFalse: [currentNumber := currentNumber * 3 + 1]].
   aCollection add: 1.
   ^ aCollection
```

Einfache Probleme

Problem 15

Link zum Originalproblem (http://projecteuler.net/index.php?section=problems&id=15)

Gitterwege

Startet man in der linken oberen Ecke eines 2x2 Gitters, so gibt es 6 Wege zur rechten unteren Ecke. (bei unerlaubter Richtungsumkehr - Einbahnstraßenprinzip)

Wie viele Wege sind in einem 20x20 Gitter möglich?

Lösung A

```
initialize
   counter := 0.
   maxX := 12.
   maxY := 12.
   self trackFrom: 1 @ 1.
   ^ counter + 2
```

```
trackFrom: passedPoint
   passedPoint y = maxY
      ifFalse: [self trackDownFrom: passedPoint].
   passedPoint x = maxX
      ifFalse: [self trackRightFrom: passedPoint]
```

```
trackRightFrom: passedPoint
   | npoint |
   counter := counter + 1.
   npoint := passedPoint x + 1 @ passedPoint y.
   (npoint x = maxX
         and: [npoint y = maxY])
      ifTrue: [^ npoint]
      ifFalse: [self trackFrom: npoint]
```

```
trackDownFrom: passedPoint
   | npoint |
   counter := counter + 1.
   npoint := passedPoint x @ (passedPoint y + 1).
   (npoint x = maxX
         and: [npoint y = maxY])
      ifTrue: [^ npoint]
      ifFalse: [self trackFrom: npoint]
```

Einfache Probleme

Problem 16

Link zum Originalproblem (http://projecteuler.net/index.php?section=problems&id=16)

Was ist die Summe der Ziffern von 2^1000?

Lösung A

```
initialize
   ^ (2 raisedTo: 1000) asString
      detectSum: [:each | each asString asInteger]
```

Einfache Probleme

Problem 17

[Link zum Originalproblem (http://projecteuler.net/index.php?section=problems&id=17)](http://projecteuler.net/index.php?section=problems&id=17)

Länge von Zahlwörtern

Schreibt man die englischen Zahlen von 1 bis 5 auf - one, two, three, four, five - so ergeben sich in der Summe 3 + 3 + 5 + 4 + 4 = 19 Buchstaben.

Wie viele Buchstaben braucht man um die Zahlen von 1 (one) bis einschließlich
1000 (one thousand) auszuschreiben?

ANMERKUNG: Leerzeichen oder Bindestriche werden nicht mitgezählt. 342, z.B. (three hundred and forty-two) besteht aus 23 und 115 (one hundred and fifteen) aus 20 Buchstaben. Die Benutzung des "and" in Zahlwörtern entspricht der britischen Rechtschreibung.

Lösung A

```
initialize
   self populateNameDict.
      ^ (1 to: 1000)
      inject: 0
      into: [:size :each | size := size + (self translate: each asString) size]
```

```
translate: aNumberString
   (dict includesKey: aNumberString asNumber)
      ifTrue: [^ dict at: aNumberString asNumber].
   aNumberString first = $0
      ifTrue: [aNumberString size > 1
            ifTrue: [^ self translate: aNumberString allButFirst]
            ifFalse: [^ '']].
   aNumberString size = 1
      ifTrue: [^ dict at: aNumberString asNumber].
   aNumberString size = 2
      ifTrue: [aNumberString asNumber > 20
            ifTrue: [^ (dict at: (aNumberString first asString , '0') asNumber)
                  , (self translate: aNumberString second asString)]
            ifFalse: [^ dict at: aNumberString asNumber]].
   aNumberString size = 3
      ifTrue: [(self translate: aNumberString allButFirst)
               ~= ''
            ifTrue: [^ (self translate: aNumberString first asString)
                  , 'hundredand'
                  , (self translate: aNumberString allButFirst)]
            ifFalse: [^ (self translate: aNumberString first asString)
                  , 'hundred']]
```

```
populateNameDict
   dict := Dictionary newFrom: {
     1 -> 'one'. 2 -> 'two'. 3 -> 'three'. 4 -> 'four'. 5 -> 'five'. 6 -> 'six'. 7 -> 'seven'.
      8 -> 'eight'. 9 -> 'nine'. 10 -> 'ten'. 11 -> 'eleven'. 12 -> 'twelve'. 13 -> 'thirteen'.
     14 -> 'fourteen'. 15 -> 'fifteen'. 16 -> 'sixteen'. 17 -> 'seventeen'. 18 -> 'eighteen'.
     19 -> 'nineteen'. 20 -> 'twenty'. 30 -> 'thirty'. 40 -> 'forty'. 50 -> 'fifty'. 60 -> 'sixty'.
     70 -> 'seventy'. 80 -> 'eighty'. 90 -> 'ninety'. 1000 -> 'onethousand'}
```

Einfache Probleme

Problem 18

Link zum Originalproblem (http://projecteuler.net/index.php?section=problems&id=18)

Wege durchs Dreieck

Lösung A

Einfache Probleme

Problem 19

Link zum Originalproblem (http://projecteuler.net/index.php?section=problems&id=19)

Wieviele Monatserste im 20sten Jahrhundert fallen auf einen Sonntag?

Lösung A

```
initialize
   | sundays startDate endDate |
   sundays := 0.
   startDate := Date
            newDay: 1
            month: 1
            year: 1901.
   endDate := Date
            newDay: 31
            month: 12
            year: 2000.
   (startDate to: endDate)
      datesDo: [:each | (each dayOfWeek = 1
               and: [each dayOfMonth = 1])
            ifTrue: [sundays := sundays + 1]].
   ^ sundays
```

Einfache Probleme

Problem 20

Link zum Originalproblem (http://projecteuler.net/index.php?section=problems&id=20)

Wie lautet die Summe der Ziffern von 100 Fakultät?

Lösung A

```
initialize
   ^ 100 factorial asString
      detectSum: [:each | each asString asInteger]
```

Einfache Probleme

Problem 21

Link zum Originalproblem (http://projecteuler.net/index.php?section=problems&id=21)

Befreundete Zahlen

d(**n**) sei definiert als die Summe ganzzahliger Teiler von **n**.
Falls d(**a**) = **b** und d(**b**) = **a**, wobei **a** ungleich **b** sei, dann sind **a** und **b** befreundete Zahlen.

Die ganzzahligen Teiler von - beispielsweise - **220** sind
1, 2, 4, 5, 10, 11, 20, 22, 44, 55 und 110, also ist d(220) = **284**.

Die ganzzahligen Teiler von **284** sind 1, 2, 4, 71 und 142, also d(284) = **220**.
220 und 284 sind befreundet.

Finde die Summe aller befreundeter Zahlen unter 10000.

Lösung A

```
initialize
   "automatically remove double hits"
   | divisors pair amicableNumbers |
   amicableNumbers := Set new.
   (9999 to: 1 by: -1)
      do: [:each |
         divisors := self properDivisorsOf: each.
         pair := (self properDivisorsOf: divisors sum) sum.
         (each = pair
               and: [each ~= divisors sum])
            ifTrue: [amicableNumbers add: each.
               amicableNumbers add: divisors sum]].
   ^ amicableNumbers sum
```

```
properDivisorsOf: aNumber
   | divisors |
   divisors := OrderedCollection new.
   aNumber = 1
      ifTrue: [^ OrderedCollection with: 1].
   (aNumber - 1 to: 1 by: -1)
      do: [:each | (aNumber isDivisibleBy: each)
            ifTrue: [divisors add: each]].
   ^ divisors
```

Einfache Probleme

Problem 22

Link zum Originalproblem (http://projecteuler.net/index.php?section=problems&id=22)

Namensliste

Die Datei names.txt (http://projecteuler.net/project/names.txt) enthält mehr als 5000 Vornamen.
Sortiere die Namen in alphabetischer Reihenfolge und berechne einen **alphabetischen Wert** für jeden dieser Namen. Anschließend multipliziere diese Zahl mit der **Position** des Namens in der **sortierten** Liste, um eine **Punktzahl** für diesen Namen zu erhalten:

Ein Beispiel: Ist die Liste sortiert, so befindet sich COLIN
(**alphabetischer Wert 3+15+12+9+14 = 53**) an 938ter Stelle.
COLIN hat also eine **Punktzahl** von 938 * 53 = **49714**.

Wie lautet die Gesamtsumme aller Punktzahlen?

Lösung A

```
initialize
   | aStream namesList |
   aStream := FileStream readOnlyFileNamed: '/home/enno/euler/names.txt'.
   namesList := (aStream contents findTokens: $,) asArray sorted.
   ^ (namesList
      collectWithIndex: [:eachName :eachIndex | eachIndex
            * (eachName withoutQuoting
                  detectSum: [:eachCharacter | eachCharacter asciiValue - $A asciiValue + 1])]) sum
```

Einfache Probleme

Problem 23

[Link zum Originalproblem (http://projecteuler.net/index.php?section=problems&id=23)](http://projecteuler.net/index.php?section=problems&id=23)

Abundante Zahlen

Eine perfekte Zahl ist eine Zahl für die die Summe der ganzzahligen Teiler wieder die ursprüngliche Zahl ergibt. So ist beispielsweise die Summe der ganzzahligen Teiler von 28 (1, 2, 4, 7, 14) wieder 28. Somit ist 28 eine perfekte Zahl.

Eine Zahl n wird defizient genannt, wenn die Summe der ganzzahligen Teiler kleiner n ist. Ist sie größer n, nennt man die Zahl abundant.

12 ist die kleinste abundante Zahl (1 + 2 + 3 + 4 + 6 = 16) und die kleinste Zahl, die als Summe zweier abundanter Zahlen geschrieben werden kann ist 24. Es kann gezeigt werden, dass alle ganzen Zahlen größer 28123 als Summe zweier abundanter Zahlen geschrieben werden können. Diese obere Grenze kann nicht weiter gesenkt werden, obwohl es bekannt ist, dass die größte Zahl, die nicht als Summe zweier abundanter Zahlen ausgedrückt werden kann, kleiner ist als diese Grenze.

Finde die Summe aller positiven ganzen Zahlen, die nicht als Summe zweier abundanter Zahlen geschrieben werden können.

Lösung A

```
initialize
   | notSum abundantNumbers sum |
   abundantNumbers := OrderedCollection new.
   sum := Set new.
   notSum := OrderedCollection new.
   (1 to: 28123)
      do: [:each | (self isAbundant: each)
            ifTrue: [abundantNumbers add: each]].
   abundantNumbers
      combinations: 2
      atATimeDo: [:each | sum add: each sum].
   abundantNumbers
      do: [:each | sum add: 2 * each].
   (1 to: 28123)
      do: [:each | (sum includes: each)
            ifFalse: [notSum add: each]].
   ^ notSum sum
```

```
isAbundant: aNumber
   ^ (self divisors: aNumber) sum - aNumber > aNumber
```

```
divisors: aNumber
   | end aSet |
   aSet := Set new.
   end := aNumber sqrt ceiling.
   (1 to: end)
      do: [:each | (aNumber isDivisibleBy: each)
            ifTrue: [aSet add: each.
               aSet add: aNumber / each]].
   ^ aSet
```

Einfache Probleme

Problem 24

Link zum Originalproblem (http://projecteuler.net/index.php?section=problems&id=24)

Eine Permutation ist eine Anordnung von Objekten. So ist z.B. 3124 eine mögliche Permutation der Ziffern 1, 2, 3 und 4. Listet man alle Permutationen in numerischer oder alphabetischer Reihenfolge auf, so erhält man eine lexikographische Ordnung. Die lexikographischen Permutationen der Ziffern 0, 1 und 2 sind:

012   021   102   120   201   210

Wie lautet die millionste lexikographische Permutation der Ziffern 0, 1, 2, 3, 4, 5, 6, 7, 8 und 9?

Lösung A

```
initialize
   | permutatedStream |
   permutatedStream := WriteStream on: ''.
   #($0 $1 $2 $3 $4 $5 $6 $7 $8 $9 )
      permutationsDo: [:each |
         permutatedStream nextPutAll: each.
         permutatedStream nextPut: Character space].
   ^ (permutatedStream contents asString findTokens: ' ') asSortedCollection at: 1000000
```

Bemerkung: Dieses Programm liefert zwar das richtige Ergebnis, als Lösung scheidet es allerdings aus. Eulerprobleme sollen von einem handelsüblichen Rechner in weniger als einer Minute gelöst werden. Die Laufzeit dieses Programmes beträgt aber mehr als drei Minuten.
Ansatz für weitere Lösungen: Lösung A erzeugt alle 3.628.800 Permutationen und sortiert sie anschließend. Es ist sinnvoller, die Permutationen bereits sortiert zu erzeugen und bei der millionsten abzubrechen. Man kann annehmen, dass sich dadurch die Laufzeit erheblich verkürzt.

Lösung B

```
folgt
```

Lösung C

```
folgt
```

Lösung D (Rüdeger)

```
initialize
   | eingabe index perm |
   eingabe := (0 to: 9) asOrderedCollection.
   index := 999999.
   perm := OrderedCollection new.
   9
      to: 0
      by: -1
      do: [:stufe |
         | fak |
         fak := stufe factorial.
         perm
            add: (eingabe removeAt: index // fak + 1).
         index := index \\ fak].
   ^ perm
```

Bemerkung: Der obige Algorithmus arbeitet mit dem (von Georg Cantor stammenden) Fakultäten-Zahlensystem.

Wikipedia (http://de.wikipedia.org/wiki/Fakultätsbasiertes_Zahlensystem#Anwendung)

Einfache Probleme

Problem 25

Link zum Originalproblem (http://projecteuler.net/index.php?section=problems&id=25)

Wie lautet der erste 1000-stellige Term der Fibonacci-Folge?

Lösung A

```
initialize
   | fibSequence highestTerm limit |
   limit := 10 raisedTo: 999.
   fibSequence := OrderedCollection with: 1 with: 2.
   [(highestTerm := (fibSequence last: 2) sum) < limit]
      whileTrue: [fibSequence add: highestTerm].
   fibSequence add: highestTerm.
   ^ fibSequence size + 1
```