

Functional Calculation 4 : The Year 1998

Neville Holmes

Email: Neville.Holmes@utas.edu.au

Abstract

Functional calculation does with operations applied to functions and numbers what numerical calculation does with functions applied to numbers. In preceding articles an introduction was given to what could be done with one commonly available tool for functional calculation, using a notation called \mathcal{J} , then details were given of simple numerical calculation, and then simple structural calculation. This article is intended to allow the reader to consider how simple structural calculation can be done in \mathcal{J} by showing numeric expressions to produce whole numbers below 100 starting from the enlisted digits of the number 1998.

Structural calculations

Preceding articles have been introducing numerical calculation and structural calculation using the interpreter for the \mathcal{J} notation. This article reverts to the pattern of the third one, showing how the whole numbers from 0 to 99 can be constructed from the digits of a year—the year 1998. This time, however, digits are to be used in lists or in a list, so that the structural functions reviewed in the previous articles can be used.

Of course, the scalar functions used in the previous exercises can also be used here, but the following structural functions are to be preferred. The first table shows structural functions used for building lists. These are a fairly mixed bunch, and indeed the shape and tally functions are not actually used for building but for reporting what has been built.

\$	shape	reshape						
#	tally	copy	#.	unbits	undigits	#:	bits	digits
?		deal						
<	box		+.	cartesian		" :	format	format
>	unbox		*.	polar		;	words	
;	raze	link	i.	integers		q:	factors	

The second table shows structural functions used for extracting or rearranging the values in a list. Again, these functions are a mixed bunch and not all will be found useful in the task of generating numbers.

,	ravel append	.	reverse rotate	:	transpose transpose
		,.	knit stitch	,:	itemise laminate
				/:	sort up
				\:	sort down
{	from	{.	head take	{:	tail
		}.	behead drop	}:	curtail
		~.	nub		

The third table shows some miscellaneous functions which extract data about their arguments.

		%.	invert project		
=	classify			-:	match
		e.	raze in member	~:	sieve
		i.	index	/:	grade up
				\:	grade down

Finally, there are two operations which are so useful for working with structures that it would be masochistic to ignore them. Both these operations, whose symbols are \sim and $/$, are monadic, which means that they suffix a function which is the target of their operation. The resulting function may be used monadically or dyadically.

For example, $2\% \sim 3$ will divide 2 *into* 3 rather than *by* 3 (its arguments are reversed), while $\wedge \sim 3$ raises 3 to the power 3 (its left argument is copied from its right argument).

Further, $+ / i . 100$ will add up the first hundred integers, $2 \ 3 + / 4 \ 5 \ 6$ will give a table with two rows and three columns containing the sum of each of 2 3 with each and every 4 5 6, and $* / \sim i . 12$ will give a "times" table for the first twelve integers, or rather $* / \sim > : i . 12$ will give the more familiar table.

Making 1998 Give 0 to 19

The four digits of 1998 may be used as a single list, that is, as 1 9 9 8, 19 9 8, 1 99 8, 19 9 8, 199 8, 19 98, or 1 998. Otherwise two lists may be used, that is, the two lists 1 9 and 9 8, or a scalar and a list, such as 1 and the list 9 98 or as the list 1 9 9 and the scalar 8.

The task is to combine them in as short and simple an expression as possible, to yield each of the numbers between 0 and 19 (here, 99 later), and to yield them as scalars.¹

To save listing space, in the following it will be assumed that

$x =: 1 \ 9 \ 9 \ 8$

has been issued. Expressions that use x are preferred as using the simplest but most extensive list of integers.

¹As a matter of æsthetics, parentheses are avoided as far as possible. Also as a matter of style, the negative sign is avoided and subtraction or negation is used to similar effect.

0	/x	=/19 98	10	{.1 9+9 8	#1 9":9 8
1	1[9 9 8	<./x	11	--~/x	+/*:i.~x
2	1 9 i.98	#1 9;9 8	12	-.~/x	+/19 9-8
3	1 9 9 i.8	#~.x	13	<.+/-:x	>.>./^.!x
4	#x	+/*x	14	>.*/^.19 9 8	+/>:19 9-8
5	+/i.~x	>.{:!:%:x	15	#.*x	19 */9 8
6	+//:x	>./!%:x	16	#./:x	+/19 9 8
7	-/ .x	#":x	17	+/1 9]9 8	>.%/19 9 8
8	{:x	%%/x	18	+/~.x	-/19 9 8
9	>./x	1{9 9 8	19	+/:x	{.19 9 8

Two expressions are given in the table for each number. Note carefully that at least one of the arguments is a list.

Making 1998 Give 20 to 99

Making numbers beyond 19 follows a similar pattern, and it's convenient here to take them twenty at a time.

20	--~/19 9 8	#.<:-:x	30	#.~.x	+/1,>:9 9 8
21	19+#9 8	<.+/ 1 9 j.9 8	31	#.}:x	+/>:x
22	#.\:x	+/,~>:#.x	32	-:*/19 9 8	*/<.+:%1 o.9 9 8
23	+/<:x	p{:x	33	+/<:19 9 8	*/#.1 9+./>:9 8
24	#.1 9 9 8	!-:{:x	34	<.!%:+>/x	+/,~}.19 9 8
25	--~/ .x	<.*/%:x	35	+/,q:19 98	#.-:x
26	<.#.%:19 9 8	1#.9 9 8	36	+/19 9 8	</*/%:19 9 8
27	+/x	19+{:9 8	37	>.*/%:19 9 8	#.1 9+9 8
28	{.+/1 9+/9 8	+//:,~x	38	{.+:19 9 8	+/19>.9 8
29	>./p:x	+/1+9 9 8	39	+/>:19 9 8	-:->:-/19 98

Once into the twenties, the integer 19 becomes very useful, though it was also handy in the teens.

40	<.*/-:x	+/-<:p:x	50	<.+:%:*/x	#.>:-:x
41	<.!%:{.19 9 8	-/1 9,-:98	51	#.-:19 9 8	-:1+99+#, 8
42	+/#. :+:#:x	+/-:p:x	52	+/+}:.x	<.+/%:19 9 8
43	<:+/+:+/q:>:x	+/1 9 9+8	53	>./>.!-:x	1{+//~9 9 8
44	+/1 9,,~9 8	+/19 9+8	54	<.^#x	+/,+:x
45	#.#.=x	*/>.%:19 9 8	55	+/19+9 8	#.<:x
46	+:p{:x	+/::, x	56	<.o.-/19 9 8	+/,~}:19 9 8
47	<.*/+:%1 o.9 9 8		57	1+*./<:9 9 8	+/19+>:9 8
48	*/#. =x	#.<.-:19 9 8	58	+/>+:x	<:#.}: .x
49	-:{:1 9 98	-/*:1-9 9 8	59	-:>:+/19 98	+/1 9,-:98

From the forties on, the integer 98 comes into play more often.

60	-:#.1 9 98	>.o.{19 9 8	70	#.x	<:p:>:-/19 9 8
61	#.1 C.9 9 8	+/p:}:x	71	+/1 9#<:9 8	p:{.19 9 8
62	1*#.9 9 8	#.}.x	72	*./x	>./1 9*9 8
63	-/*:x	-/p:<:19 9 8	73	1+*./9 9 8	
64	/*:19 9 8	-.-/*:x	74	+/#. :++>:#:~.x	
65	+/*:1 9-9 8	-/p:19 9 8	75	19+*/<:9 8	*/>.+.%1 o.9 9 8
66	+/:<:19 9 8	#.19 _9 8	76	+/1 9 9#-:8	+/1 9 9*-:8
67	p:-/19 9 8	#.-:>: .x	77	-/*:>:x	+/,1 9\$9 8
68	-:#.19 98	+/,1 9\$<:9 8	78	-.-/*:>:x	+:/>:19 9 8
69	#. .19 9 8	#.1+9 9 8	79	-~/19 98	1 9#.<:9 8

In the sixties and seventies, squaring (*:) becomes useful.

80	*/<:+/>: :=x	-.-/19 98	90	-/1 9 98	*./>:x
81	1 9#. .9 8	+/1 9#9 8	91	-/1+99 8	19+*/9 8
82	+/1 9,*/9 8		92	-/>:-:1 99 8	+/<:p:>:x
83	+/<:#.>:#:x	#.<:#.+:x	93	-.~/1 99 8	*/p:1 9!>:9 8
84	%:*/+:1 9 98	+/p:x	94	>.-/%-.1 o.9 9 8	
85	#.>:x	-:+/ ,1 9*/9 8	95	#.<:19 9 8	
86	+/,1 9\$>:9 8		96	-:>:-/199 8	*./#.*:+=x
87	+/#.>:#:x		97	<:1 9]98	#.>:#.+:x
88	<./+>:o.x	+/>:p:x	98	{:1 9 98	+/,1 9*./9 8
89	1 9#.9 8	-/1-9 98	99	1{.99 8	1 9#.>:9 8

Once in eighties, simpler expressions become possible because integers like 98, 99, and 199 can be brought into play.

Further Examples

The examples given above can only suggest how arithmetic functions can be used in a simple way to produce a variety of numbers. The reader is urged to consider the examples above with a J interpreter to hand, to try the examples out, to check them, and to try to find expressions that are better or in some way more interesting than those given here. When generating these numbers begins to pall, the reader perhaps should go on to consider how to generate the three digit numbers using the same rules. This could start +/}:1 99 8 then +/1 99,*8.

Alternatively, expressions might be sought for other years. Some years will present special challenges. The following table gives a start for the year 2000, in which a choice is made between 0=0, 0!0 and 0^0 to give a 1 largely on æsthetic grounds.

In the following table it has been assumed that

$$x =: 2 0 0 0$$

has been issued.

0	<./x	*/x	10	*/*:>:~.x	20%#0 0
1	./x	2 0 0 i.0	11	>.//^x	#.2 0 0>.-.0
2	+/x	-/x	12	+/*:>:x	+/,>:=x
3	>:{.x	*/>:x	13	>./o.>:x	20-<.^+/>:0 0
4	#x	+/*:x	14	#.2+0 0 0	20-<.o.+/>:0 0
5	2+#0 0 0	2++/>:0 0 0	15	+/<.o.>:}:x	#.=~x
6	+/>:x	+/2+0 0 0	16	#.x	^/2>.0 0 0
7	2#.-.0 0 0	+/*:2,>:0 0 0	17	<.-/^>:~.x	>.2*+/^>:0 0 0
8	#.,#::~.x	-/*:>:x	18	20-#0 0	+/<.o.2+0 0 0
9	*/*:>:x	+/2+>:0 0 0	19	20-=/0 0	>./o.2+0 0 0

Another amusing possibility, though ultimately monotonous because expressions are restricted to monadic functions, is to try to develop all the numbers from four zeroes in at least one list and at most one scalar.

In the following table it has been assumed that

z =: 0 0 0 0

has been issued.

0	~.z	0{0 0 0	10	+!//:z	>./o.0=0 0 0
1	=/z	0 e.0 0 0	11	#./:z	>.//^-.z
2	--//:z	+/0 0=0 0	12	+/:/:z	*!//:z
3	{:/:z	+/0=0 0 0	13	+/>:+:#}:/:z	>./o.-.z
4	#z	+/<.^0 0=0 0	14	+/*:/:z	#.>:0=0 0 0
5	-.-/+:/:z	<.//^0 0=0 0	15	#.-.z	+/>.!^0=0 0 0
6	+//:z	+/<.o.0 0=0 0	16	+/<.!^-z	>.^/^0 0=0 0
7	#.0=0 0 0	-.-/*/:/:z	17	+/{.>}.o./:z	<.*/>:o.0 0=0 0
8	>.-.-/o./:z	+/:-z	18	+*:{./:z	+/<.o./:z
9	*:/{:/:z	*:+/0=0 0 0	19	>./o./:z	<:+/,~!/:z