

# Functional Calculation 8 : The Year 1066

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  $\mathcal{J}$ , details were given of simple numerical calculation, examples based on generating numbers from the digits of the year 1997 were set out for study and experiment. Another such example was based on the enlisted digits of the year 1998.

The next article explained several simple operations, a kind of function that yields a function instead of a numeric value, and was accompanied by another article illustrating this by working with the digits of the year 1999.

The preceding article explained how arguments could be fed in parallel to more than one function at a time through the use of *trains*. This article is intended to allow the reader to consider how simple trains can be used in  $\mathcal{J}$  by showing functional expressions producing functions that, applied to the number 1066, or dyadically between the numbers 10 and 66, produce whole numbers below 100.

## Numerical calculations

To set up some simple examples, trains to produce all the non-negative integers of fewer than three digits are to be sought. There is a single main restriction, and that restriction makes the task rather difficult, which is why the examples given below are incomplete.

No operations are to be used. However, the use of trains allows the argument or arguments to be used repeatedly so that required values can be built up.

Otherwise, only the argument 1066 may be used in the monadic case, and in the dyadic case 10 must be the left argument and 66 the right. Note that all arguments are scalar, that is, they are single values without shape.

## Making 1066 Give 0 to 19

The task is to produce functionally all the integers from 0 to 19 in two restricted ways. The first way is by applying a monadic function to an argument of 1066. The second way is by applying a dyadic function to the arguments 10 and 66.

The basic idea is to find the simplest or otherwise prettiest function in each case that will give the required result.

	f 1066	10 g 66	f 1066	10 g 66
0	<*	*	10 ] ~[:<.^.*^.	<.-:
1	>-	<>	11 *{.[:q:>:	]%%
2	^.*:	+:	12 *{q:-*	+
3	*+*+*	-:	13 *{q:	(*.<:)%[
4	%^.*:	<.-	14 [:>.^.+^.	[:>.[%:!
5	*{[:q:<:	*.%]	15 *#.[:>:#:	+<.-<
6	[:<.[:* /q: ^.%:	+	16 [:>.-:%%:+*	<.+
7	[:>.^.	[:>.^.+ -<	17 [:>.-:%%:	[:<.[%:!*[
8	[:>.^.+*	[:>.^.+	18 [:>.%: *:	+ +
9	[:>.*+^.+*	[-<	19 [:>.%:-^.+^.	[:<.-~%^.+<

Notice that being reluctant to use operators in the monadic case, which must start with 1066 as its sole argument, soon forces the use of the *cap* function to allow an integer to be produced from a preliminary result that is not an integer.

### Making 1066 Give 20 to 99

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

	f 1066	10 g 66	f 1066	10 g 66
20	(%:^.*:)**{[:q:<:	<.+[	30 (]-/)q:	[:<.-~%^.
21	[:<.[ ~%:	<{< . (+q:)	31 [:-/q:+*	[:>.-~%^.
22	[:>.[%^.*^.	]%[: -:	32 [:>.[%%%:+*	[+ + +
23	[:>.[ ~%:+*	(+*)+ +	33 [:>.%:	*.%[
24	[:>.[%:-:	*<.-	34 [:>.%%:+*	[:+:[+<+
25	(*{q:)+*{q:-*	[:q: -<	35 [:<.] ~%:+^.	(<+ )* -<
26	[:*/*}.[: .q:	+ *.	36 *: ~[:>.%%:+^.	*
27	*+:+ ~[:<.%%:+^.	[:>. ^^.	37 ] ~[:>.^.*^.	[:p:<+<.
28	[:>.%:-%:%^.	-~%<+<	38 [:>.%%+%:%^.	+%<+<
29	[:-/q:-*	[:p: -<	39 [:({-{.-})q:	( -:)*(*.<:)%[

	f 1066	10 g 66		f 1066	10 g 66
40	[ :>. %: + ^ .	[ * < . -	50	[ :>. * + ^ . * ^ .	- ~
41	q: { ~ * + *	[ : < . + % ^ .	51		+   * - <
42	[ : * . / q: + *	[ : > . + % ^ .	52	[ : + : [ : > . % : - ^ .	+   *
43	[ : ( { . + { : } ) q:	+ - :	53	* # . [ : < : q:	< + +   *
44	[ : > . ^ . * ^ .   ]	*   ~ + + <	54	[ : + / * } . q:	+   * + [ : > . ^ .
45	[ : < . [   ~ ^ . * ^ .	[ : < . - ~ * ^ . - <	55	[ : < . ^ . + ^ . * ^ .	- ~ - <
46	[ : < . [ :   % : j . % :	- ~ - < .	56	* # . q:	] - [
47	[ : > . ^ . + ^ . + % :	< - - + < .	57		< - -
48	[ : + / [ : ~ . [ : q: q: # . q:   * < +   + <		58		+   * +
49	[ : > . ^ . * ^ .	[ : * :   + <	59	[ : + / q: + *	> . -   + <

XXXX

	f 1066	10 g 66		f 1066	10 g 66
60	[ : - : [ : * . / q: - *	< . *	70		< . *   + <
61		< + < . *	71	[ : { : [ : q: < :	( + - : ) ~
62	[ : + / q: + * + *	- -	72		> . +
63		[ : > . ( - % : ) ~	73		[ : > : ] + % ~
64	[ : > . + : % % : + *	( * . < : ) - ]	74		+ - < + <
65		] - <	75	( % ~ + : ) # . q:	+ < :
66	[ : > . % : + % :	> + > .	76	[ : > . ^ . * % : % * + * + *	+ ] +
67	[ : > . - : % ^ . + *	] + <	77	[ : > . - : % : ^ .	+ > :
68		< + > . + <	78	[ : - : [ : > . ]   ~ % : * ^ .	] +   +
69	[ : - / [ : q: < :	[ : < . ( + % : ) ~	79	[ : + / [ : q: < :	[ : p: < + < . + < .

XXXXX

	f 1066	10 g 66		f 1066	10 g 66
80	+ :   ~ [ : > . % : * ^ .		90	[ : > . - : % ^ . - *	
81		++ ( + . < : )	91		
82	[ : ( { . * { : } ) q:	[ + > . +	92		
83		( * . % ) * [ + % ~	93		
84	[ : > . ^ . + - : % ^ .	! < . - <	94		
85			95		
86	[ : ( { : - { . } ) [ : q: > :	[ + +	96	[ : + / [ : q: ] + * + *	
87			97	* { [ : q: > :	
88		++   +	98	[ : > . ^ . * ^ . + ^ .	
89	+ :   ~ [ : < . % : * ^ .		99		( * . < : ) ~ % + .

## Remarks

The examples given here can only suggest how arithmetic functions can be used in a simple to produce a variety of numbers. The reader is urged to consider these examples

with a  $\mathcal{J}$  interpreter to hand, to try 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.