# Functional Calculation 6 : The Year 1999

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 `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. This article is intended to allow the reader to consider how simple operations can be used in `J` by showing functional expressions producing functions that, applied to the number `1999`, or dyadically between the numbers `19` and `98`, produce whole numbers below `100`.

## Numerical calculations

Preceding articles have introduced numerical calculation using the interpreter for the `J` notation. This article gives a change of pace in which the notation already introduced is used in modest examples so that its differences with the more usual (though inconsistent) mathematical notation can become familiar.

To set up some simple examples, expressions to produce all the non-negative integers of fewer than three digits are to be sought. There are several restrictions, and these restrictions make the task rather difficult, which is why the examples given below are incomplete.

Firstly, operations must be used, as many different ones as possible. The following are the adverbs that might be useful.

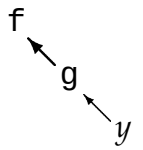
| | | | | | | | |
|---|---|---|---|---|---|---|---|
| `~` | both | swap | | | | | |
| `/` | across | between | `/.` | diagonals | sequester | | |
| `\` | prefixes | infixes | `\.` | suffixes | exfixes | | |
| `}` | extract | amend | `b.` | basic | | | |
| | | | `f.` | fix | fix | | |

The following are the conjunctions that might be useful.

| | | | ;. | cut | cut | ^: | power | power |
|---|---|---|---|---|---|---|---|---|
| " | rank | rank | !. | fit | fit | !:<br>L: | foreign<br>level | foreign<br>level |
| @<br>& | of<br>after | of<br>after | &. | dual | dual | @:<br>&: | of<br>after | of<br>after |

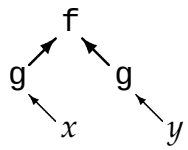
The following chart is given to remind you of how two of the more useful conjunctions work.

`f&g` *y*



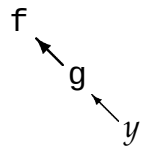

*Monadic After*

*x* `f&g` *y*



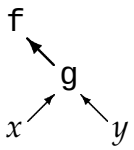

*Dyadic After*

`f@g` *y*




*Monadic Of*

*x* `f@g` *y*




*Dyadic Of*

Secondly, only the argument `1999` may be used in the monadic case, and in the dyadic case `19` must be the left argument and `99` the right. Note that all arguments are scalar, that is, they are single values without shape.

**Making 1999 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 `1999`. The second way is by applying a dyadic function to the arguments `19` and `99`.

The basic idea is to find the simplest or otherwise prettiest function in each case that will give the required result, providing at least one operation is used. For example, in the following table `[&]` is given as the dyadic function to produce `19`, but although `|@[` would not be longer, it would not be, to my taste, as pretty.

| | `f 1999` | `19 g 99` | | `f 1999` | `19 g 99` |
|---|---|---|---|---|---|
| 0 | `-~` | `\|&>:` | 10 | `*./@q:@>:` | `+.&-:&>:` |
| 1 | `%~` | `*@\|` | 11 | `+/@q:@>.@%:` | `<.&(>./)&q:` |
| 2 | `>.@^@%` | `+.&<:` | 12 | `+:@<.@%:@%:` | `\|&<:&+:` |
| 3 | `>.@%:@^.` | `<.@+:@^.` | 13 | `<.@%:@+:@+:@%:` | `+&<.&%:` |
| 4 | `>.@^.@%:` | `\|&>:&+:` | 14 | `>.@^.@*:@-:` | `>.@*&^.` |
| 5 | `>.@%@-:@%:` | `%~&>:` | 15 | `+/@q:@<.@%:` | `>,@+&%:` |
| 6 | `<.@%:@%:` | `>.@%~` | 16 | `+:@>.@^.` | `\|&+:&<:` |
| 7 | `>.@^.@-:` | `+/@,&#:` | 17 | `+/@:>:@#:@>:` | `\|&>:&>:` |
| 8 | `>.@^.` | `\|&+:` | 18 | `<.@^.@p:@*:` | `<:@[` |
| 9 | `>.@^.@o.` | `-:@<:@[` | 19 | `p:@>.@%:@%:` | `[&]` |

Of course, some of these functions are more complex that would be needed if the use of an operation were not required. Notice the heavy use of `%:` and `^.` to get the number `1999` down to a usable size.

**Making 1999 Give 20 to 99**

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

| | f 1999 | 19 g 99 | | f 1999 | 19 g 99 |
|---|---|---|---|---|---|
| 20 | +/@:>:@#: | +.&>: | 30 | +/@:>:@q:@>: | +&(-/@q:) |
| 21 | +/@q:@>.@-: | >.@^@<:@\| | 31 | <.@%:@-: | +/@:>:@>:@,&#: |
| 22 | <.@-:@%: | +&(<./@q:) | 32 | >.@%:@-: | *&>.&%:&-: |
| 23 | >.@-:@%: | +/@,&q:&>: | 33 | +/@:p:@q:@<.@%:@+: | +/@~.@,&q: |
| 24 | >.@o.@^. | +/@,&q:&<: | 34 | +:@<.@!@^.@%: | +/@,&q:&>.&o. |
| 25 | ^~/@~.@q:@>: | -~&>.&.%: | 35 | <.@+:@!@^.@%: | |
| 26 | | \|&p:&-:&<: | 36 | -/@q:@<: | *&<.&%: |
| 27 | ^~@>.@%:@^. | \|&-:&*: | 37 | >./@q:@<: | >.@!&%:&-: |
| 28 | | \|&>:&*: | 38 | >.@o.@%:@o.@%: | +/@:+:@:>:@,&#: |
| 29 | | {:@q:@>:@*:@] | 39 | | +/@:p:@>:@q:@>.@%: |

Here there is frequent use of `q:` occasional use of `#:` to produce lists to work with. Notice also that the @ operation is used in the monadic function list in preference to the & operation, whihc would work there much the same.

| | f 1999 | 19 g 99 | | f 1999 | 19 g 99 |
|---|---|---|---|---|---|
| 40 | | -:@-~ | 50 | | *.&-:&>: |
| 41 | | | 51 | <.@(%~/)@q:@>.@o. | >.@\|@j.&-: |
| 42 | +/@~.@q:@<: | *&%:&<: | 52 | >.@(%~/)@q:@>.@o. | \|&*:&<.&-: |
| 43 | +/@:*:@q:@>.@%: | <.@*&%: | 53 | <.@!@%:@j.~ | |
| 44 | <.@%: | \|&p: | 54 | >.@\|@%:@j.~ | \|&*: |
| 45 | >.@%: | >.@^.@!&<: | 55 | +/@q:@>:@>: | >.@^@\| |
| 46 | +/@q:@<.@-: | <.@^.@! | 56 | | <.@\|@j.~@-:@- |
| 47 | +/@:p:@q:@<.@%: | >.@^.@! | 57 | | +/@q:@>:@*:@] |
| 48 | +/@q:@<: | \|&p:&+: | 58 | >.@*:@^. | +&-:&<: |
| 49 | *:@<.@^. | -:@>.@<: | 59 | | -:@+ |

```
    f 1999     19 g 99            f 1999           19 g 99
60             +&-:&>:       70
61             +&>:&-:       71  +/@q:@<:@*:
62                           72                   |&+:&p:&>:
63  <.@%:@+:   -:@!&<.&%:    73  >.@o.@<.@o.@^.
64  >.@%:@+:   -:@>.@%:@p:@* 74
65                           75
66                           76
67                           77
68                           78
69                           79

    f 1999         19 g 99        f 1999           19 g 99
80                 -~        90  >.@+:@%:
81                           91  +/@:*:@q:@>:
82                           92
83                           93  <.@%:@-:@p:
84                 *&<.&%:&+: 94  >.@%:@-:@p:      <.@^.@!&+:
85  +/@,@(-:&+/~)@#:         95                      |&p:&p:&<.&-:
86  <.@%:@(+/)@q:@>:@*:      96  +/@:+:@q:@<:     |&+:&p:&+:
                   <.@&%:&+:
87  >.@%:@(+/)@q:@>:@*:      97
                   >.@&%:&+:
88  +:@<.@%:      |&+:&p:     98                     <:@>.
89  >.@%:@p:@<.@-:           99  +/@:-:@,@(+/~)@#: >.&|
```

**Remarks**

The examples given here can only suggest how arithmetic functions can be combined to produce a variety of functions. The reader is urged to consider these examples with a J interpreter to hand, to try examples out, to check them, and to try to find functions 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.