

Functions as First Class Citizens

Walter G Spunde
University of Southern Queensland
Toowoomba, Queensland, 4350
Australia

Abstract

While most students can probably re-produce a textbook definition of *function* that is beyond criticism, they may nevertheless proceed to confuse a function's name, f , with its values, $f(x)$, declare specific inputs as the first step in a function's computer definition and not be able to pass variables effectively from one function to another. Classical mathematics does not seem to give students enough experience in the use of functions to apply their knowledge effectively in 'real-world' situations.

In this article we discuss aspects of a functional computer language that exposes students to working with and thinking in terms of functions. Functions are the verbs of our mathematical language and we will discuss adverbs and conjunctions besides composite verbs and some of the implications of definitions in purely functional terms. Finally we will touch on the balance between classical algebra and functional expressions.

Introduction

When students write things like " $f(t)$ when $x = 3$ ", it might not necessarily indicate a lack of understanding of the concept of *function*. It may mean that the description of functions encountered has been so highly stylized and seldom used in practice that students lack the understanding that comes from experience in manipulating functional notation. If students are to construct their own meanings from their experiences, [1, 5], then we must look carefully at the experiences that we provide with functional notation. Looking across a range of calculus texts, *reformed* or otherwise, it should not come as any surprise if students still see functions as formulae – because that is what they work with when they work with 'functions'.

Mastery of functional notation is more important in a computer environment than in the classical algebraic setting since what, in the latter case, might appear to be some small confusion of expression is completely debilitating in a computer environment where nothing will work if the use of variables and function syntax is not understood. It is the ideal environment for experience with functions.

Calculus teachers themselves contribute to the perception that functional notation is only good for *substituting in values*. If we can take usage in texts as a guide, few instructors would be

happy with a statement like $(\sin \star \cos)(3)$. All texts, of course, contain the statement that $f \star g$ is the function whose values at x , are given by $f(x) \star g(x)$ but it seems that it is not expected that this notation will actually be used. $(f \star g)' = f' \star g + f \star g'$ is acceptable but $(\sin \star \cos)' = (\sin' \star \cos) + (\sin \star \cos')$ is seldom seen.

Presumably such expressions would not be well received. This discomfort indicates a perception that function notation is only to be used generically and is not useful for the manipulation of actual functions – and that seems to be what students pick up.

Is the notation useful? If the focus is on computations and these must be done by hand, then, of course, it is not, since writing $h = f + g$ gets one no closer to finding values for the function h . However, in a computer environment what was simply a statement of relationship on paper becomes an executable statement that *creates* a new entity which can be used to deliver results.

In the following, we will assume that we have a computer language (K.E. Iverson’s **J**, [4], is one such language) where we can define a function f by executing, for example, $f =: \sin \star \cos$ and discuss the change in perspective that thinking in functional terms involves. Naturally we assume that mathematics is being taught with a computer at hand.

Adverbs

From the formulae above it is clear that we often use operators on functions with a post-fix syntax as in f' and f *squared*. Linguistically, if we think of functions as *verbs* then these modifiers of functional behaviour are *adverbs*.

Not all adverbs need be post-fix operators. In fact, since post-fix operators can be replaced by pre-fix compositions (*square of f*, *derivative of f*), they would hardly justify the introduction of the adverb. However, adverbs that modify the behaviour of a function in different ways prove to be very attractive.

Working with data that may consist of lists, tables or multi-dimensional arrays, the application of a verb such as *sum* will need to be modified so as to specify what is going to be added up – the elements in the rows or columns of a table, corresponding elements in a list of tables, etc. The advantage of having an adverb to make this modification is that the same modifications may be used with other verbs, like *max*, *min*, *mean* or *standard_deviation* without creating a proliferation of special verbs.

The case of applying the arithmetic operators over a numeric list, as the verb *sum* would do, justifies its own adverb, *over*. Thus, “+ over” would return the sum of the numbers in a list; “– over” would return their alternating sum and “* over” the product of the entries.

$$/ \text{ over } 1, 2, 3, 4, 5, \dots, 2n = \frac{1 \star 3 \star 5 \star 7 \star \dots \star (2n - 1)}{2 \star 4 \star 6 \star 8 \star \dots \star 2n}$$

and “+ reciprocal over” will produce a continued fraction. The symbols Σ and Π hide the unifying concept of operations over lists that the adverb highlights.

Often we want partial sums of series or partial products, so an adverb that returns cumulative results is very useful. For example,

$$+ \text{ over cumulatively } 2 \ 4 \ 8 \ 12 \ 13 \ 15 \ 19 \ 27 \ \text{ gives } 2 \ 6 \ 14 \ 26 \ 39 \ 54 \ 73 \ 100$$

Adverbs that apply functions over lists or arrays in particular ways are complemented by others, such as *each* which might apply a function to each element within an array. For example, if we have an array of coordinates, A :

| | | | |
|-----------|-----------|-------------|------------|
| (1, 4, 8) | (2, 3, 6) | (2, 10, 11) | (3, 4, 12) |
| (6, 6, 7) | (2, 6, 9) | (1, 12, 12) | (8, 9, 12) |

then *mod each A* gives

| | | | |
|----|----|----|----|
| 9 | 7 | 15 | 13 |
| 11 | 11 | 17 | 17 |

Clearly this makes it much easier to compute surface areas and integrals from first principles then could be done in a language where both lack of suitable data structures and available operators impede expression, [2].

Another useful adverb (\sim) is one that reverses the arguments, so that if we wanted a function that divided the first argument into the second then it would be $/ \sim$. This adverb is particularly useful when used in conjunction with other functional combinations. The list of useful adverbs is limited only by one's mastery of the language of mathematics.

Function Composition and Forks

Adverbs coming from postfix operators can, alternatively, be viewed as a composition of two functions with prefix syntax. Using @ ('atop of' or just 'of') as the conjunction for function composition rather than the familiar, but less evocative, \circ , we have

$$\cos \text{ squared} = \text{square} @ \cos \quad \text{with } \text{square} \text{ appropriately defined.}$$

Composition is the only function combination considered in elementary textbooks beside arithmetic combinations but no link is made between the two. For prefix, or monadic, functions F and U , the composition $F @ U$ sees function U applied to deliver the argument for function F . If F were a *dyadic* function (that is, one with infix syntax), then we would expect composition to have functions delivering both arguments for F so that we would be evaluating $U(x) F V(x)$. If F is one of the common arithmetic functions ($+$, $-$, \times , \div) we see that the arithmetic combinations are just special cases of dyadic composition.

If U and V were themselves dyadic functions, then we would have $(x U y) F (x V y)$. We will refer to this very common mathematical construction as a *fork* and write it simply as the train $U F V$. In an environment rich in carefully defined dyadic functions, the fork is a delightful combination of functions.

For common examples, however, function F must be one of the arithmetic functions, since there are so few other widely accepted dyadic functions. Unfortunately, most mathematical software does not allow the user to define dyadic functions, to the great impoverishment of the language.

Nevertheless there are numerous common examples of forks.

- The mean is the sum of the entries of a list divided by a count of the entries ($\text{sum} / \text{count}$);
- The range is the smallest of the entries subtracted from the largest, ($\text{max} - \text{min}$);
- Standard scores are the deviations from the mean divided by the standard deviation of the scores (dev / std).
- Difference quotients for a sample of values of function, f , are $(d @ f) / d$ where d is a function that returns differences between successive entries in a list. These quotients constitute an exact sample for the derivative function at (unknown) points in the partition. [3]
- The integral of a function is given by $f_r \text{ dot } d$ where f_r is a function that returns the average function value over each subinterval of a partition (in practice we have to find functions that

approximate f_r). If we modified the dot product function to return partial sums then this function would return values of the integral function to every partition point.

- The definition of the determinant as the dot product of any row or column of a matrix and its cofactors could be obtained from the fork *entries dot cofactors* applied to a matrix and a row/column identifier with appropriately defined functions *entries* and *cofactors*.

Wherever the two inputs to a dyadic function are themselves obtained from functions on the original data we have a fork structure. Since most mathematical programming languages do not allow for user-defined dyadic functions, many of these ideas remain foreign to most people's thinking and, in a vicious cycle, software vendors, fettered by commercial considerations, remain largely remain uninterested.

Hooks

For the standard scores of a list x , we compute $z = (dev\ x)/(std\ x)$. The deviations from the mean are $x - mean\ x$ and illustrate another very common functional structure, which we will call a *hook*.

The hook is a two verb train in which the first verb is dyadic and which, applied monadically, is defined as $(F\ G)(x) = x\ F\ G(x)$. Again we are restricted to showing only arithmetic functions in the role of F but freedom to work in this environment produces many more interesting examples.

- For deviations from the mean $- mean$.
- If *sig* is a function that returns a list of Booleans depending on whether an entry in a list is above a certain magnitude, then we can set those entries of smaller modulus to zero with the hook $*\ sig$.
- The fraction each entry contributes to the sum of a list is given by $/\ sum$.

Conjunctions

Composition @ is the most common example of a conjunction. It looks like a dyadic verb but since the result is a verb, the operator is properly speaking a conjunction.

Another common conjunction is bonding (&) of a numeric value to a function. Thus $\wedge&2$ attaches '2' as the right argument of the power function \wedge and gives a *square* function. Transformations like converting degrees to radians are easily constructed with a constant multiplier $\star&(\pi/180)$. When this conjunction connects two functions F and G where F is dyadic, we have: $x\ (F\ \&\ G)\ y = G(x)\ F\ G(y)$. The cross product of the vector differences $(b - a)$ and $(c - a)$ might be written as $b\ (cross\ \&\ (-\ \&\ a))\ c$.

Similarly $+ \&\ log$ would return the sum of the logarithms and the anti-log of this sum will give the product of the numbers. **J** provides another conjunction (&.) to perform the inversion as well. $x\ (F\ \&.\ G)\ y = G^{-1}(G(x)\ F\ G(y))$ where G^{-1} has to be a known, or appropriately defined inverse function. The opportunity to use this conjunction presents itself wherever operations under transformation are involved. For a monadic F there is no difference between $\&$ and @, but $(F\ \&.\ G)\ x = G^{-1}(F\ G(x))$

Formulae in variables or in functional expressions?

The standard deviation written in functional terms is $\text{sqrt} @ \text{mean} @ \text{square} @ (- \text{mean})$.

$$\sqrt{\frac{\sum_{n=1}^N \left(x_n - \frac{\sum_{n=1}^N x_n}{N} \right)^2}{N}}$$

Compare this with the classical notation at left. Notice that N has been identified as the count of both the number of entries and the number of squared deviations. The results of computations are (judiciously) mixed in with functional instructions (sum or \sum) in classical notation in order to achieve the maximum ‘simplification’.

This raises the question of the role of algebraic manipulation. Clearly the above computation is equivalent to $\text{sqrt} @ (\text{mean} @ \text{square}) - (\text{square} @ \text{mean})$. A derivation in functional terms begins by noting that squaring the difference is equivalent to summing the squares minus doubling the product, or: $(\text{square} @ -) = (+ \& \text{square}) - (\text{double} @ *)$.

Consequently, $\text{square} @ (- \text{mean}) = (\text{square} + (\text{square} @ \text{mean})) - (\text{double} @ (* \text{mean}))$

and hence $(\text{mean} @ \text{square}) + (\text{square} @ \text{mean}) - \text{double} @ (\text{mean} * \text{mean})$

which is clearly the same as $\text{mean} @ \text{square} - \text{square} @ \text{mean}$.

Like any new road travelled, it will seem longer and less convenient than a well trod path, but algebraic identities are relationships between functions and variables should be irrelevant. It is interesting to speculate what sort of knowledge will prove to be more useful when most algebraic manipulation is done by a computer algebra system.

Conclusion

A computer environment that permits the implementation of functions provides experiences for students that they would not otherwise have. When the environment allows dyadic functions, function trains (such as forks and hooks), adverbs and conjunctions and the definition of functions in purely functional terms without reference to input variables, functions become objects in their own right, objects *that can be manipulated*. This opens a new perspective on the role of functions for instructors as well as for students and one that surely is deserving of some attention.

References

- [1] Skemp, Richard R. *The Psychology of Learning Mathematics*, Lawrence Erlbaum Associates., Hillsdale, New Jersey, 1987.
- [2] Spunde, W.G. *Integration (multi-variable included) from First Principles*, Communications: $\Delta'03$: Fourth Southern Hemisphere Symposium on Undergraduate Mathematics Teaching, Queenstown, New Zealand, 2003, pp. 240-246.
- [3] Spunde, Walter G. & Richard D. Neidinger, *Sample Calculus*, The Mathematics Magazine, 72(3), 1999, pp. 171–182.
- [4] Thomson, Norman J. *J - The Natural Language for Analytic Computing*, Research Studies Press, Baldock, Hertfordshire, UK, 2001.
- [5] Tobin, K. & Tippins, D. *Constructivism as a referent for teaching and learning*, in K.Tobin (Ed.) *The Practice of Constructivism in Science Education*, Lawrence Erlbaum Associates., Hillsdale, New Jersey, 1993, pp. 3–23.