NumPy another Iverson Ghost

Posted: 31 Mar 2018 21:07:53

During my recent SmugMug API and Python adventures I was haunted by an Iverson ghost: `NumPy`

An Iverson ghost is an embedding of APL like array programming features in nonAPL languages and tools.

You would be surprised at how often Iverson ghosts appear. Whenever programmers are challenged with processing large numeric arrays they rediscover bits of APL. Often they're unaware of the rich heritage of array processing languages but in `NumPy's` case, they *indirectly* acknowledged the debt. In *Numerical Python* the authors wrote:

> *The languages which were used to guide the development of NumPy include the infamous APL family of languages, Basis, MATLAB, FORTRAN, S and S+, and others.*

I consider "infamous" an upgrade from "a mistake carried through to perfection."

Not only do developers frequently conjure up Iverson ghosts, they also invariably turn into little apostles of array programming that won't shut up about how cutting down on all those goddamn loops clarifies and simplifies algorithms. How learning to think about operating on entire arrays, versus one dinky number at a time, frees the mind. Why it's almost as if array programming is a tool of thought.

Where have I heard this before?

Ahh, I've got it, when I first encountered APL almost fifty years ago.

Yes, I am an old programmer, a fossil, a living relic. My brain is a putrid pool of punky programming languages. Python is just the latest in a longish line of languages. Some people collect stamps. I collect programming languages. And, just like stamp collectors have favorite stamps, I find some programming languages more attractive than others. For example, I recognize the undeniable utility of `C/C++`, for many tasks they are the only serious options, yet as useful and pervasive as `C/C++` are they have never tickled my fancy. The notation is ugly! Yeah, I said it; suck on it C people. Similarly, the world's most commonly used programming language `JavaScript` is equally ugly. Again, `JavaScript` is so damn useful that programmers put up with its many warts. Some have even made a few bucks writing books about its meager good parts.

I have similar inflammatory opinions about other widely used languages. The one that is making me miserable now is `SQL`, particularly Microsoft's variant `T-SQL`. On purely aesthetic grounds I find well-formed `SQL` queries less appalling than your average `C` pointer fest. Core `SQL` is fairly elegant but the macro programming features that have grown up around it are depraved. I feel dirty when forced to use them which is just about every other day.

At the end of my programming day, I want to look on something that is beautiful. I don't particularly care about how useful a chunk of code is or how much money it might make, or what silly little business problem it solves. If the damn code is ugly I don't want to see it.

People keep rediscovering array programming, best described in Ken Iverson's 1962 book *A Programming Language*, for two basic reasons:

1. It's an efficient way to handle an important class of problems.
2. It's a step away from the ugly and back towards the beautiful.

Both of these reasons manifest in `NumPy`'s resounding success in the Python world.

As usual, efficiency led the way. The authors of *Numerical Python* note:

> *Why are these extensions needed? The core reason is a very prosaic one, and that is that manipulating a set of a million numbers in Python with the standard data structures such as lists, tuples or classes is much too slow and uses too much space.*

Faced with a "does not compute" situation you can either try something else or fix what you have. The Python people fixed Python with `NumPy`. Pythonistas reluctantly embraced `NumPy` but quickly went *apostolic!* Now books like *Elegant SciPy* and the entire SciPy toolset that been built on NumPy take it for granted.

Is there anything in NumPy for programmers that have been drinking the array processing cool aid for decades? The answer is yes! J programmers, in particular, are in for a treat with the new Python3 addon that's been released with the latest J 8.07 beta. This addon directly supports NumPy arrays making it easy to swap data in and out of the J/Python environments. It's one of those best of both worlds things.

The following NumPy examples are from the SciPy.org NumPy quick start tutorial. For each NumPy statement, I have provided a J equivalent. J is a descendant of APL. It was largely designed by the same man: Ken Iverson. A scumbag lawyer or greedy patent troll might consider suing NumPy's creators after looking at these examples. APL's influence is obvious. Fortunately, Ken Iverson was more interested in promoting good ideas that profiting from them. I suspect he would be flattered that APL has mutated and colonized strange new worlds and I think even zealous Pythonistas will agree that Python is a delightfully strange world.

Some Numpy and J examples

Selected Examples from https://docs.scipy.org/doc/numpy-dev/user/quickstart.html Output has been suppressed here. For a more detailed look at these examples browse the Jupyter notebook: NumPy and J Make Sweet Array Love.

Creating simple arrays

```
# numpy
a = np.arange(15).reshape(3, 5)

NB. J
a =. 3 5 $ i. 15

# numpy
a = np.array([2,3,4])

NB. J
a =. 2 3 4

# numpy
b = np.array([(1.5,2,3), (4,5,6)])
```

```
NB. J
b =. 1.5 2 3 ,: 4 5 6

# numpy
c = np.array( [ [1,2], [3,4] ], dtype=complex )

NB. J
0 j.~ 1 2 ,: 3 4

# numpy - make complex numbers with nonzero real and imaginary parts
c + (0+4.7j)

NB. J - also for J
c + 0j4.7

# numpy
np.zeros( (3,4) )

NB. J
3 4 $ 0

# numpy - allocates array with whatever is in memory
np.empty( (2,3) )

NB. J - uses fill - safer but slower than numpy's trust memory method
2 3 $ 0.0001
```

Basic operations

```
# numpy
a = np.array( [20,30,40,50] )
b = np.arange( 4 )
c = a - b

NB. J
a =. 20 30 40 50
b =. i. 4
```

```
c =. a - b

# numpy - uses previously defined (b)
b ** 2

NB. J
b ^ 2

# numpy - uses previously defined (a)
10 * np.sin(a)

NB. J
10 * 1 o. a

# numpy - booleans are True and False
a < 35

NB. J - booleans are 1 and 0
a < 35
```

Array processing

```
# numpy
a = np.array( [[1,1], [0,1]] )
b = np.array( [[2,0], [3,4]] )
# elementwise product
a * b

NB. J
a =. 1 1 ,: 0 1
b =. 2 0 ,: 3 4
a * b

# numpy - matrix product
np.dot(a, b)

NB. J - matrix product
a +/ . * b
```

```
# numpy - uniform pseudo random
a = np.random.random( (2,3) )

NB. J - uniform pseudo random
a =. ? 2 3 $ 0

# numpy - sum all array elements - implicit ravel
a.sum(a)

NB. J - sum all array elements - explicit ravel
+/ , a

# numpy
b = np.arange(12).reshape(3,4)
# sum of each column
b.sum(axis=0)
# min of each row
b.min(axis=1)
# cumulative sum along each row
b.cumsum(axis=1)
# transpose
b.T

NB. J
b =. 3 4 $ i. 12
NB. sum of each column
+/ b
NB. min of each row
<./"1 b
NB. cumulative sum along each row
+/\"0 1 b
NB. transpose
|: b
```

Indexing and slicing

```
# numpy
a = np.arange(10) ** 3
a[2]
a[2:5]
a[ : :-1]   # reversal

NB. J
a =. (i. 10) ^ 3
2 { a
(2 + i. 3) { a
|. a
```