Python for Teachers: A Workshop

Pycon 2009, Notes by Kirby Urner, 4D Solutions, Portland, Oregon, USA

Premise:

The new numeracy of our day, arising in circumstances in some ways analogous to the advent of the abacus in dark ages Europe, via *Liber Abacci* by Fibonacci of Pisa, is arising in the wake of two computer revolutions, PC (personal computer) and FOSS (free and open source software).

Past:

Since Leibniz at least, people have considered the possiblity of Machine Logic, have forecast what we might today call computer languages. Now that these languages have grown up in our midst, we have a need to deploy them within our pre-existing curricula, thereby altering same.

The connection to mathematics seems especially close, with Kenneth Iverson, designer of APL (A Programming Language) thinking in terms of executable mathematics notations, e.g. APL is a mathematical language which just happens to have the added virtue of being machine-runnable.

However mathematics permeates other subjects, including geography, linguistics and graphic design. Bottom line: no field is entirely safe from the transformative influence of computer technology, or, in more positive terms, pretty much any walk of life and/or discipline stands to benefit in some way, from our growing mastery of integrated circuitry and digital networking.

Present:

At the outset of the 21st century, many technical professions are experiencing decreases in enrollment and there's broad concern that civilian engineering in particular is ineffective in recruiting. The pre-college mathematics track is considered largely to blame, coincidentally making it ripe for an overhaul (or "makeover"). Revitalizing mathematics by making it more computer language friendly is the strategy many have been suggesting. This is the strategy pursued in this workshop.

Future:

Without postulating a "one size fits all" outcome, we might reasonably presume that the object oriented approach to programming will galvanize a number of overdue reforms. Considering "types of object" is not a new aspect of logical and/or mathematical thought.

To treat polynomials, vectors, integers, polyhedra, sequences, sets, as "mathematical objects" is hardly a stretch, and therefore bridging from pre-computer mathematics to Python's classes and instances takes only a little groundwork. The edu-sig archive within the Python.org Web site provides many ideas for how to proceed, contributed by Python Teachers with many different backgrounds.

To characterize mathematics itself as "extensible type systems" is likewise to suggest a fairly sophisticated mental model, especially for a pre-college student eyeing a liberal studies program.

Of Technical Content and Lore

The topics below, far from exhaustive, give some idea of what a pre-college mathematics curriculum might include.

A lot of these topics are somewhat unfamiliar from the standpoint of the existing Precalculus through Calculus track. However, we consider the status quo track to be broken (see above).

Adding computer languages to the math learning experience doesn't just alter the budget (most the software is free), it alters the content, or at least has the potential to do so.

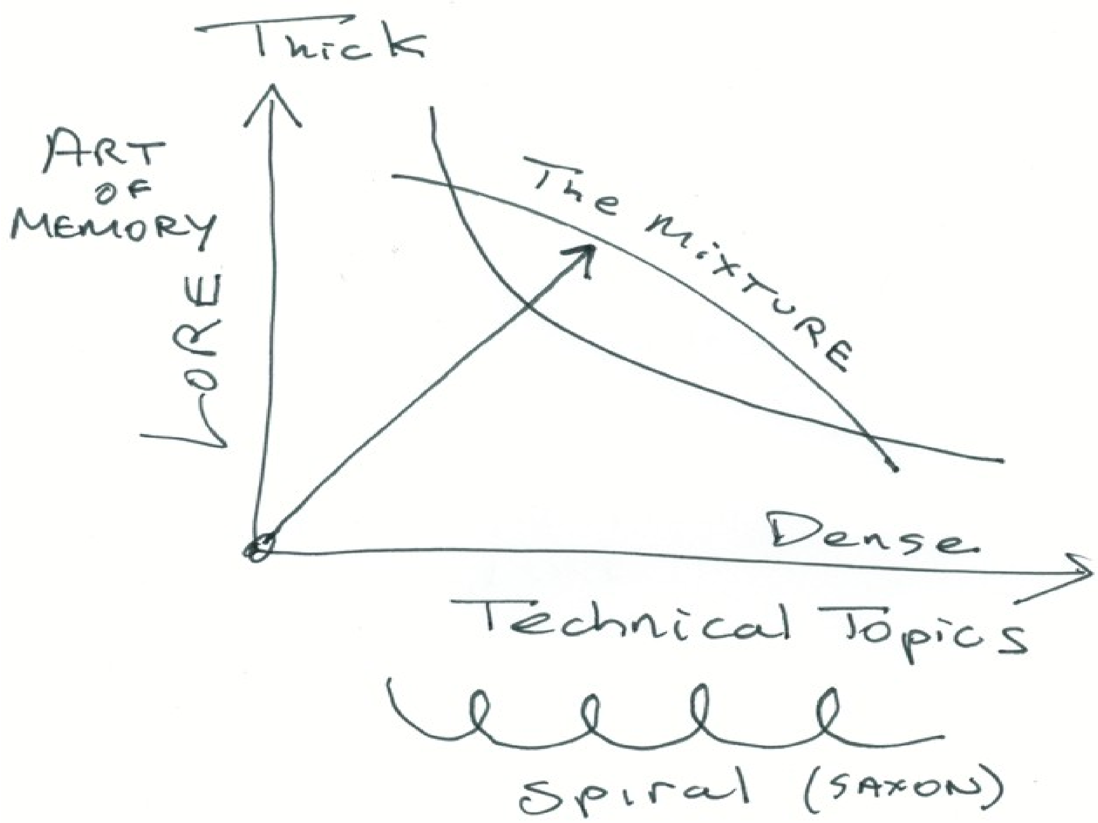
Prime Numbers (sieves)
Prime Numbers (trials by division)
Euclid's Algorithm (Guido's gcd)
Rational Numbers
Polyhedra (as Python objects: scale, rotate, translate)
Figurate Numbers
Polyhedral Numbers (icosahedral, geodesic spheres)
Pascal's Triangle (triangular and tetrahedral numbers)
Fibonacci Numbers (converge to phi, pentagon math)
Vectors (VPython -- xyz, spherical coordinates etc.)
Modulo Numbers (override __mul__, __add__)
Finite Groups (Python module)
Euclid's Extended Algorithm (needed for inverses)
Totient and Totative (gcd based)
Fermat's Little Theorem (assert...)
Euler's Theorem for Totients (assert...)
Mandelbrot Set (chaotic sequences)
Miller-Rabin (or Jython probablePrime)
RSA.encrypt(m, N)
RSA.decrypt(c, N, d=secretkey)

Let's start looking at the kind of pedagogy we might use, around Python in particular, to make this material accessible. The word “pedagogy” is perhaps misleading however, as our trainees may not be children or even young adults.

Just because we're redesigning the high school mathematics track doesn't mean older adults cannot or will not benefit from revisiting this material. Gaining mastery over computing is what the Renaissance was all about. Once the “abacus way” of doing arithemetic had spread to the hitherto innumerate, a new merchant class was able to bootstrap itself.

The “second axis” of technical skills building is the story line or narrative, also known as the lore. Why are we learning this material and what is it for? This is where the present curriculum is manifestly weak, as the case for needing calculus on the job is often somewhat underwhelming. On the other hand, a genre of story most students appreciate is the “how things work” genre. How does the Internet work? What must I know to manage a household, to cook, to find my way around in the world?

In sum, the “second axis” is about technical content passing relevance tests, and these tests vary with student demographics. In *Python for Teachers*, we look at, run and write source code, but we also tell stories about what we are doing and why. These stories vary by walk of life, task at hand and so on.



Implicit in the above diagram is an assumption of limited bandwidth (the learner's capacity to process information). Getting densely technical moves us away from the stories, which may be thick in their own way. In oscillating between stories and skills-building exercises, we vary the diet and help match student needs. This same diagram is useful for self teaching.

The spiral alludes to John Saxon's penchant for revisiting the same topics repeatedly, but at different levels and within different contexts, John H. Saxon (1923 – 17 October 1996) being a curriculum writer of some noteriety for his “maverick” views.

Included under Lore, in addition to “how things work” are “the stories of our time” i.e. some of the long term historical unfoldings, intellectual history as it were. Here are some examples of important stories:

- the rise of Unicode in overcoming language barriers (coding in Klingon)
- the emergence of free and open source software (and not just software)
- abuses of computer technology (database especially)
- the evolution of cryptography including public key (RSA in the browser)
- the renaissance in geographic visualization and planning (Google Earth, ESRI)

SOURCE CODE:

File Edit Format Run Options Windows Help

```python
"""module:  simplelife.py"""

class Biotum:

    def __init__(self, name):
        self.name = name
        self.stomach = []

    def __call__(self,food):
        self.stomach.append(food)

    def __repr__(self):
        return 'Biotum named ' + self.name
```

Ln: 1 Col: 0

File Edit Shell Debug Options Windows Help

```
>>> import simplelife
>>> imp.reload(simplelife)
<module 'simplelife' from '/home/ki
rby/simplelife.py'>
>>> from simplelife import Biotum
>>> cell1 = Biotum('xV')
>>> cell2 = Biotum('xY')
>>> cell1.stomach
[]
>>> cell1('**')
>>> cell1.stomach
['**']
>>> cell1('&&&')
>>> cell1.stomach
['**', '&&&']
>>> cell1(cell2)
>>> cell1.stomach
['**', '&&&', Biotum named xY]
>>> 
>>> 
```

Ln: 55 Col: 4

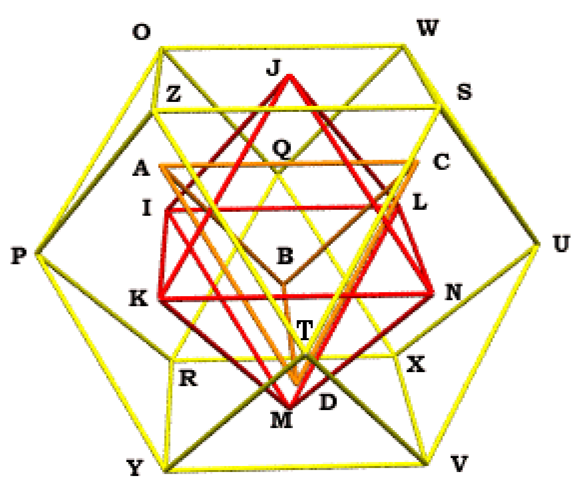
```
"""
Some infrastructure for working with Vectors and Edges, including
an xyplotter generator and axes maker.

By Kirby Urner, Sept 13, 2006

Updated Sept 29, 2006:
make Edge color a class-level attribute
add funky derivative demo
refactor a bit

Code:
http://www.4dsolutions.net/ocn/python/stickworks.py

For colorized source:
http://www.4dsolutions.net/cgi-bin/py2html.cgi?script=/ocn/python/stickworks.py

Some relevant discussion:
http://mail.python.org/pipermail/edu-sig/2006-September/007145.html
http://mail.python.org/pipermail/edu-sig/2006-September/007149.html
http://mail.python.org/pipermail/edu-sig/2006-September/007150.html
http://mail.python.org/pipermail/edu-sig/2006-September/007312.html
"""

from visual import vector, cylinder, cross, dot, diff_angle
import visual

class Vector (object):

    """
    A wrapper for visual.vector that expresses a cylinder via draw(),
    always pegged to the origin
    """

    radius = 0.03

    def __init__(self, xyz, color=(0,0,1)):
        self.v = vector(*xyz)
        self.xyz = xyz
        self.color = color
        self.cyl = None

    def draw(self):
        """define and render the cylinder"""
        self.cyl = cylinder(pos = (0,0,0), axis = self.v,
                   radius = self.radius, color = self.color)

    def erase(self):
        """toss the cylinder"""
        if self.cyl:
            self.cyl.visible = 0
        self.cyl = None

    def __repr__(self):
        return 'Vector @ (%s,%s,%s)' % self.xyz
```

```
    # some vector ops, including scalar multiplication

    def diff_angle(self, other):
        return self.v.diff_angle(other.v)

    def cross(self, other):
        temp = cross(self.v, other.v)
        return Vector((temp.x, temp.y, temp.z))

    def dot(self, other):
        return dot(self.v, other.v)

    def __sub__(self, other):
        temp = self.v - other.v
        return Vector((temp.x, temp.y, temp.z))

    def __add__(self, other):
        temp = self.v + other.v
        return Vector((temp.x, temp.y, temp.z))

    def __mul__(self, scalar):
        temp = self.v * scalar
        return Vector((temp.x, temp.y, temp.z))

    __rmul__ = __mul__

    def __neg__(self):
        return Vector((-self.v.x, -self.v.y, -self.v.z))

    def _length(self):
        return pow(self.v.x ** 2 + self.v.y ** 2 + self.v.z ** 2, 0.5)

    length = property(_length)

class Edge (object):

    """
    Edges are defined by two Vectors (above) and express as cylinder via draw().
    """

    radius = 0.03
    color = (1,0,0)

    def __init__(self, v0, v1, color=None):
        if not color==None:
            self.color = color
        self.v0 = v0
        self.v1 = v1
        self.cyl = None

    def draw(self):
        """define and render the cylinder"""
        temp = (self.v1 - self.v0).xyz
        self.cyl = cylinder(pos = self.v0.xyz, axis = vector(*temp),
                            radius = self.radius, color = self.color)
```

```
    def erase(self):
        """toss the cylinder"""
        if self.cyl:
            self.cyl.visible = 0
        self.cyl = None

    def __repr__(self):
        return 'Edge from %s to %s' % (self.v0, self.v1)


def xyplotter(domain, f):
    """
    domain should be an initialized generator, ready for next() triggering.
    f is any function of x.  Consecutive Vectors trace connected edges.
    """
    x0 = domain.next()
    y0  = f(x0)
    while True:
        x1 = domain.next()
        y1 =  f(x1)
        e = Edge( Vector((x0, y0, 0)), Vector((x1, y1, 0)) )
        e.draw()
        yield None
        x0, y0 = x1, y1

def axes(x=0,y=0,z=0):
    """
    Draw some axes on the VPython canvas
    """
    v0 = Vector((x,0,0))
    v0.draw()
    v0 = Vector((-x,0,0))
    v0.draw()

    v0 = Vector((0,y,0))
    v0.draw()
    v0 = Vector((0,-y,0))
    v0.draw()

    v0 = Vector((0,0,z))
    v0.draw()
    v0 = Vector((0,0,-z))
    v0.draw()


def dgen(start, step):
    """
    generic domain generator
    """
    while True:
       yield start
       start += step
```

```
def testme():
    """
    >>> from stickworks import testme
    Visual 2005-01-08
    >>> testme()

    See:
    http://www.4dsolutions.net/ocn/graphics/cosines.png # missing (sorry)
    """

    from math import cos

    def f(x):  return cos(x)

    d = dgen(-5, 0.1)
    axes(-5,1,0)
    graph = xyplotter(d, f)

    for i in xrange(100):
        graph.next()

def testmemore():
    """
    See:
    http://www.4dsolutions.net/ocn/graphics/pycalculus.png
    """

    def snakeywakey(x):
        """
        Polynomial with x-axis crossings at 3,2,-3,-7, with scaler
        to keep y-values under control (from a plotting point of view)
        """
    return 0.01 * (x-3)*(x-2)*(x+3)*(x+7)

    def deriv(f, h=1e-5):
        """
        Generic df(x)/dx approximator (discrete h)
        """
    def funk(x):
        return (f(x+h)-f(x))/h
    return funk


    d1 = dgen(-8, 0.1)
    d2 = dgen(-8, 0.1)
    d3 = dgen(-8, 0.1)

    axes(-8,5,3)

    deriv_snakeywakey = deriv(snakeywakey)
    second_deriv = deriv(deriv_snakeywakey)

    graph1 = xyplotter(d1, snakeywakey)
    graph2 = xyplotter(d2, deriv_snakeywakey)
    graph3 = xyplotter(d3, second_deriv)

    Edge.color = (1,0,0)  # make snakeywakey red
```

```
    for i in xrange(130):
        graph1.next()

    Edge.color = (0,1,0)  # make derivative green

    for i in xrange(130):
        graph2.next()

    Edge.color = (0,1,1)  # make 2nd derivative cyan

    for i in xrange(130):
        graph3.next()

if __name__ == '__main__':
    testme()


#==================[ Points of Interest ]====================

"""
* 26 data points A-Z define six polyhedra in the concentric hierarchy

                          Labels of   Numbers of
Shape              Volume Vertices    Vertices, Edges, Faces
-------------------------------------------------------------
Tetrahedron             1   A -D             4      6      4
Inv Tetra               1   E -H             4      6      4
Duo-tet Cube            3   A -H             8     12      6
Octahedron              4   I -N             6     12      8
Rh Dodecahedron         6   A -N            14     24     12
Cuboctahedron          20   O -Z            12     24     14

See: http://www.4dsolutions.net/ocn/graphics/povlabels.gif
"""
```



Addendum:

So how does one implement reform in education? Trying to steer from a national level might seem like a hopeless undertaking, yet constructivist and place based curricula may be implemented at the local level by small institutions seeking to niche market themselves as somehow unique. The private sector "mom & pop" storefront martial arts academy is a good analogy. On this model, one expects good ideas to spread through imitation, copying, rather than by edict in some top-down regime. Those writing the curriculum, including the very teachers who will be using the material, need to find willing sponsors and "guinea pigs" at the regional level. Buy in by parents, a few friendly politicians, is what's critical.

Given some freedom to reform on a small scale, by public charter and / or private academy, this incentive to "niche market" may lead to some exotic "boutique" offerings that nevertheless serve as pilots or prototypes for a larger demographic pool down the road. This is how theater companies and circus troupes operate, and so it's appropriate that Python be named for Monty Python's Flying Circus, a comedic theatrical troupe purveying a rather specialized brand of British humor likewise popular in the Netherlands when Guido van Rossum was pioneering this new computer language.

My own trajectory has involved intersecting the esoteric math-with-Python thread with a "new kind of geometry" somewhat related to Wolfram's "new kind of science" in that space-filling cellular automata, or "honeycombs" are involved. I draw much of my material from what are, by today's standards, esoteric sources. For example, the 26 "points of interest" A-Z on the previous page, along with the associated whole number volumes, trace to *Synergetics: Explorations in the Geometry of Thinking*, by R. Buckminster Fuller (Scribner / Macmillan, 1975, 1979). Hardly anyone is using that. I have very little competition. From a niche marketing viewpoint, that nets me a commercial advantage. For students just learning algebra, I promote Caleb Gattegno's approach, again almost unheard of in many math-teaching circles, yet quite likely to spread.