Connecting the Dots: American Transcendentalism Meets Pythonic Math

Kirby Urner
4D Solutions
Portland, Oregon

A talk proposal for
EuroPython 2007
Vilnius, Lithuania

June, 2007

Abstract

My paper for EuroPython 2005 explored what I call Pythonic Mathematics, a way of presenting pre-computer analytical content within the object-oriented paradigm, including pre-college.[1] This thinking informed my participation in Shuttleworth Foundation planning meetings and my presentation to the London Knowledge Lab in the following year.[2]

This year, I'm delving yet more deeply into Pythonic Math, while also weaving in some more cultural threads, especially the "design science" thread with its geodesic spheres and other graphical content, the theme of my OSCON 2005 presentation.[3] I've been field testing these combinations in my home town of Portland, through a school called Saturday Academy.[4]

Whereas Guido named Python for Monty Python, begetting allusions which aren't going to go away, there's more we might do to make our snake come across as charming and smooth, not too slimy or oily (negative attributes customarily associated with snakes by the more snake-unfriendly).[5]

Disclaimer

My business here is not to ensnare or entrap our snake (meaning Python, a computer language) within a singular web or matrix of my own devising. Rather it is my intention to sketch a “namespace,” one among many, against which Python’s continued spread and adoption might co-occur, along with its thriving elsewhere.

I consider this writing a kind of ethnography therefore, as well as philosophy, perhaps interesting to tourists who would like to visit a specific tribe or nation in which Python is shared with the children, and valued as a power tool by adults – but not in such a way as to preclude its use in other neighborhoods, likewise worth visiting.

So now I will proceed to weave my singular web or matrix, by means of interconnected topical sections, deliberately over-ambitious sometimes, in what they purport to cover.

But rest assured that Python is not entrapped herein, nor am I practicing “guilt by association” in supposing that my journey’s chronicles reflect the experiences of all those other stars and leading lights within our community (more we bask in one another’s glory).

Scientific Reasoning

One of the hallmarks of rationality or ratiocination is the ability to reason by analogy. The very word “ratio” implies comparing two measures, contemplating a relationship.

By means of analogy, we're able to translate the unfamiliar into the more familiar and thereby reach valid conclusions, perhaps avoiding bad outcomes while securing happy results. Such is the hope, in any case.

Of course as soon as one mentions analogies, we're able to think of "bad" or "inappropriate" analogies. Finding the right analogy (or model) is what's critical, and that's often easier said than done, as the whole reason we're looking for analogies in the first place is our target system is only partially comprehended, perhaps mostly unknown.

So what analogies are the right ones? This again is a description of what rational thinking is like, or we could call it the scientific method. We need to be ready to discard analogies, once we determine we're probably using a bad one.

A Short History of Computer Programming

Per the history of computer science, we've gone through eras or ages vis-à-vis our techniques for programming. Computations were initially hard-wired, with early mainframes looking like spaghetti monsters.

Moving to assembly language cleared up the wiring, but the same untamed Wild West of spaghetti code obtained. This isn't a criticism. The model for assembly language is computer memory with its sequence of addresses, and a CPU for doing work on the values stored in these addresses, or on the addresses themselves (the genesis of "pointers" in the C language).

Given that all but the most elementary programs hinge on conditional logic, the only way to control the logical flow is with a lot of "goto" or "jump" operations, which is about the only infrastructure provided at this very low level of the chip (e.g. by Intel or AMD), its instruction set, registers, random access to memory. However, with the emergence of higher level languages, such as FORTRAN, the Wild West approach needed to be superseded, and Edsger Dijkstra became the evangelist for the new way, called structured programming.[6]

Organize everything around a backbone or main call sequence while branching away to subroutines that return results in obvious ways. Don't rely on the subtleties of side effects. Pass things explicitly. You'll thank yourself in the morning, when it comes time to debug.

The move to structured programming wasn't the final paradigm shift, but one more in what was to become a long succession, still ongoing. Programmers were beginning to realize that new languages came with new ways of looking, in some cases *better* ways (more adapted ways) of tackling a given knowledge domain and getting the job done.

However, assembly language wasn't going to disappear (nor was FORTRAN for that matter) and old habits needn't "die hard" where still needed.

Model, View, Controller (MVC)

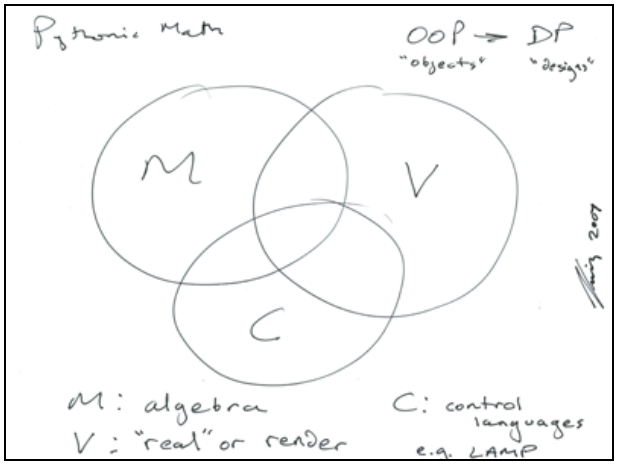

Figure 1: Model, View, Controller (Saturday Academy handout)

You may have noticed I included the word "model" as a drop-in for "analogy" in the above *Scientific Reasoning* section. The story here is we consider one special case in light of another ("to analogize") only to realize that it's what's common to both that we care about, which commonality then abstracts, as it were, to become its own special case.

We're tempted to use some arcane notations at this point. But let's just stop here and call this our Model.

Corresponding to our Model are the many Views of it. These may be static web pages, running simulations, or slowly sometimes painfully derived snap shots.

Then let's introduce a third party: a user wanting to switch views, tweak the model, and otherwise interact with the application. This requires an intermediary called a Controller. A computer language such as Python is often used, whereas the model and views may be implemented in other languages, such as SQL for the model, an XML for the view.

Python's Zope is likewise amenable to such an MVC-based analysis.[7]

Philosophy of Language

In a *Short History of Computer Programming* (above), I wrote: "Programmers were beginning to realize that new languages came with new ways of looking…" For example, the OO languages (object-oriented) would inspire a whole Design Patterns literature. Array-eating pipelines such as APL and J would call for yet different ways of looking. Meanwhile, LISP programmers would continue advertising how these multifarious "paradigms" were trivially expressed in the form of tail-recursive S-expressions (another paradigm).

Parallel to (or convergent with) these developments in computer science, were developments in psychology and philosophy. The psychologists wrote about the Whorfian Hypothesis (that language morphs perceptions) while the philosophers went around the bend -- later named "the linguistic turn" by Richard Rorty – with Wittgenstein its chief beacon (because he really whipped around it himself).

Whereas late 20th century psych and/or philo students might have found postmodern writings "too mushy" (or "woolly" as the sheep herder UKers like to say), computer programmers have gotten on-the-job experience with many a Logic, each pre-equipped with productivity-enhancing perceptions. For example, to think the "OO way" is to think "everything is an object, with attributes and powers" – not unlike a character in a multiuser domain (MUD), the precursor of today's more immersive environments.

But at another level, it's still about a stack and a heap, and the tradeoffs for using either. Different levels encourage different paradigms. Networking with TCP/IP is yet another namespace that begets its corresponding "ways of looking."[8]

American Transcendentalism

Meanwhile, in the literature department, we've had much ferment. Without attempting an exhaustive account, I will follow a couple threads, branching off from Wittgenstein in the previous section.

Sometime in the mid-1970s, Princeton's maverick celebrity philosopher, Walter Kaufmann, Nietzsche translator and former U.S. Army interrogator, started tipping his cap towards *est*, a homegrown San Francisco based philosophy, strongly influenced by Zen. He wrote an endorsement on the back of W.W. Bartley's Werner Erhard bio, and discussed it in one of his *Discovering the Mind* volumes.[9]

Around 1980, Werner Erhard and has cadre of "*est* trainers," then at the center of a nasty publicity storm, started collaborating with R. Buckminster Fuller, a self-styled pirate whom many considered a genius (others thought he was crazy).[10]

Events seemed to spiral out of control for awhile, against the backdrop of the Reagan Era's cold war, while a lot of interesting literature was taking shape behind the scenes, including the four volume *Synergetics Dictionary* by E.J. Applewhite, sitting right here in my office and frequently consulted.[11]

Also included in the Fuller syllabus is *Tetrascroll: Goldilocks and the Three Bears: a Cosmic Fairy Tale*, originally produced as a limited edition, tetrahedral "scroll" and later reprinted in book form.

In this latter work, Fuller celebrates serpent imagery in conjunction with a global maritime culture, in a mythology running somewhat counter to the more familiar genesis story wherein the snake plays a role in humanity's downfall. Naga, the sea dragon, is a source of wisdom in Fuller's telling, not a "bad guy" at all.

Jumping back a bit, American Transcendentalism inherits from the New England variety, championed by such as Fuller's great aunt, Margaret Fuller Osoli, editor of *Dial*, a rogues' gallery for such trend setters as Ralph Waldo Emerson and Henry David Thoreau.

Emerson liked Walt Whitman a lot but thought there was too much sex in his poetry. Whitman refused to back off (much to the disgust of T.S. Eliot) probably with his future HBO-minded American audience in mind (i.e. he was prescient).[12]

Fuller of Harvard comes later in the game, dubbed "World Game" by him, with "Guinea Pig B" a star player. Fuller, an inventor, invented his own namespace fairly completely, after taking time off post-1927 to divest himself thoroughly of any unwanted "reflex conditioning," and laying the groundwork for some life-long self-disciplines, somewhat described in his penultimate non-posthumously published magnum opus *Critical Path*.[13]

Tracking Bucky Fuller, *and* Ezra Pound, *and* James Joyce, was Hugh Kenner, a friend of E.J. Applewhite's and likewise well-versed enough in computer science to hold forth as a columnist for *Byte Magazine* (McGraw-Hill). Here's how I describe these relationships in a recent posting to the Math Forum at Drexel University (still at Swarthmore College when I joined):

> Keep in mind that Hugh Kenner, author of *The Pound Era* and renowned Joyce scholar, was also the author of *Bucky* (a bio) and *Geodesic Math and How to Use It* (a how-to, recently republished). Hugh was also a columnist for *Byte Magazine* (McGraw-Hill), the company for which I was working around the time he got Eliza (a pseudo-intelligent therapist, implemented in software) talking to Racter (another of the same type).[14]

My conclusion from all this is that recent trends have conspired to put computer science in a promising position. We have ample literature to explore regarding our intellectual precursors, much of which promises to be entertaining and inspiring (e.g. with pirates galore).

This bodes well for our recruiting efforts.

Given the concentration and years of study this discipline requires (on a par with medicine in some respects), it's good to be able to advertise some of these benefits up front.

Early Math Pedagogy

Against this backdrop, let's just assume that computer science remains sexy enough to command a large following of wannabe "rock stars" over the long haul. How will we

properly prepare them to have a reasonable shot at rewarding careers? That shouldn't be too tall of an order.

A lot of us have reached a similar conclusion by now: machine executable languages have the satisfying rigor, the lack of ambiguity, we associate with proving or disproving theorems. Given this likeness, it makes sense to cultivate hybrids crossing computer languages with math topics. We're only just beginning to explore the pedagogical ramifications of this endeavor, with "Python Nation" doing a lot of the trailblazing.[15]

If this view catches on, then it's retrospectively easy to see where Guido was pointing when he wrote his CP4E proposal and went looking for, and got, some funding from the USA's DoD.[16] He was pointing to a world wherein electronic computing is ubiquitous, is mostly taken for granted, and wherein newly checking in youth will spontaneously assume responsibility for maintaining and improving our shared residence, as the seniors continue to check out. That's pretty much the world-hotel we live in now.

Python, being an OO language, comes pre-equipped to support the object-oriented way of looking. However, within that broad category, containing many other languages, we come to what's special about Python, its "special names" for example, which all come with "under under" syntax. Every language is terminally idiosyncratic in this way i.e. boils down to some quirky, seemingly arbitrary "way that it is" set of design decisions.

That's not a criticism. But then there's a shakeout process nevertheless: not every language gains a foothold in the same ecosystemic niche (some die in academia – a fate for which many were consciously intended). Plus quite ancient languages (by today's standards) aren't necessarily going anywhere anytime soon – rewriting some big time-tested library in a new language may be less feasible than simply recruiting volunteers to master the old one (old languages can be fun, is a message CS departments need to deliver – without lying). Hence FORTRAN retains its relevance, given its Fast Fourier Transforms, matrix inversions or whatever it's good at.[17]

So let's assume CP4E points to computing ubiquity, and Python represents one of many controller languages, each with its own peculiar "way of looking." Let's not assume that human beings are tied too closely to any one particular language. As humans, we're not entirely "locked in" to our mortal world views, or at least we try not to be. Remember, that's the essence of the scientific disposition: be ready to abandon ship, if your analogy proves wrong and/or obsolete.

Python in the __future__

People will take it in different directions, but I'm thinking some grounding in core Python would facilitate making mathematics come to life in an OO type zoo. Dry abstractions will take on more malleable and hands-on aspects, as students work with class definitions (templates, boilerplates) with their methods and attributes, and with their myriads of instantiated objects.

In some versions of this curriculum, I drill towards an explanation of RSA, herding my students through some hoops in group theory, with its Galois Fields of P-objects (totatives of some prime, modulo that prime) or simply groups if the modulus is composite.[18]

```
IDLE 1.2
>>> import goodies
>>> showme = goodies.test()
>>> showme.next()
  * |  1  2  4  5  8 10 11 13 16 17 19 20
    ------------------------------------------------
  1 |  1  2  4  5  8 10 11 13 16 17 19 20
  2 |  2  4  8 10 16 20  1  5 11 13 17 19
  4 |  4  8 16 20 11 19  2 10  1  5 13 17
  5 |  5 10 20  4 19  8 13  2 17  1 11 16
  8 |  8 16 11 19  1 17  4 20  2 10  5 13
 10 | 10 20 19  8 17 16  5  4 13  2  1 11
 11 | 11  1  2 13  4  5 16 17  8 19 20 10
 13 | 13  5 10  2 20  4 17  1 19 11 16  8
 16 | 16 11  1 17  2 13  8 19  4 20 10  5
 17 | 17 13  5  1 10  2 19 11 20 16  8  4
 19 | 19 17 13 11  5  1 20 16 10  8  4  2
 20 | 20 19 17 16 13 11 10  8  5  4  2  1
```

Figure 2: Cayley Table for 21's Totatives

In other versions, I'm using a Gibbsian brand of vector arithmetic to generate polyhedra as math objects with methods to (a) scale (b) translate and (c) rotate, and with attributes such as volume, number of edges, vertices, openings, even texture and permeability if destined to be realized as time/size views, such as ray-tracings.[19]

What's unique about Python includes what I call __ribs__ (snakes have a lot of 'em), or "special names" such as __add__, __sub__, __neg__, __mul__, __div__, and __pow__. With this apparatus, we're able to see how different math objects share a need for "addition" and "multiplication" as binary operators, or of "negation" and "reciprocation" as unary ones.

Given these syntactical commonalities across families of math object, we have a basis for imparting the abstractions of abstract algebra, such as the broad categories of group, ring and field. We can do this pre-college, if we start with our training early enough.

Modern K-12 mathematics curricula often advertise that they cover these topics, in the form of "properties of whole numbers" for example, but these sections are often cursory and anemic. A more Pythonic math would be well positioned to compete with these other treatments, in terms of credibility. Which curriculum do *you* think is giving a stronger grasp of the basics?

Namespaces and Language Games

Ludwig Wittgenstein is perhaps best known for coining the term "language game." He used these as tools of investigation. Where a real, workaday language might be too complicated to adequately capture and hold still, a simpler construct might serve as a working model or analogy. In *Philosophical Investigations*, among other writings, LW constructs language games with an eye towards demystifying certain awkward corners in grammar, where many philosophers become shipwrecked.

Starting with the idea of libraries, computer languages have tended to feature ways to include or import. By not importing unnecessary features, programs stay smaller. By sharing libraries, programmers avoid always reinventing the same wheels. Plus libraries have ways of insulating a core language from the nitty-gritty details of a specific platform or device.

With libraries comes the possibility, even the likelihood, of "name collisions." Only a small set of words do most of the work, and programmers tend to gravitate to this same small set. Preserving the identity of functional components may become difficult in heterogeneous environments, unless there's a well thought out strategy for preventing name collisions, unwanted ambiguities. In Python that strategy would be "namespaces."

Namespaces are a lot like language games in conception, in that each defines its own domain of interconnected rule-governed objects. We often use "dot notation" to keep the namespaces separate (a syntactical feature shared by several other object-oriented languages, including JavaScript and Java). A tenet of Pythonic Math would be that dot notation is contemporaneously a math notation.

Math Objects and Polyhedra

In writing about "math objects" I've been referencing the object oriented paradigm in computer science, wherein we have "classes" or "types" of thing, organized in inheritance structures as templates, and actual objects with selves, called "instantiations" or "incarnations" of these templates.

In the mathematical domain, "math types" are likewise abstract in the sense of maybe having no obvious non-notational representation. However, polyhedra would prove an exception to this rule, in that their visual representation is one of their signature traits.

My envisioned curriculum takes advantage of this convergence of "logical types" in two disciplines, and develops polyhedra within an OO framework. A superclass embodies the basic idea of a polyhedron, as a spinnable, scalable, translatable shape, comprised of vertices, edges and faces (per Euler: $V + F = E + 2$, given some fairly liberal stipulations about the polyvertexia in question).

Each subclass of this superclass defines a specific polyhedron.

Then come the representations, with bindings to OpenGL (or DirectX or any such real time engine, able to sustain interactivity, a frame rate), or perhaps to a ray tracing engine (such as POV-Ray, the one I use in Saturday Academy classes in Portland). The content is now graphical, spatial.

Constructing the views described above will require some coordinate system savvy. The radial vectors, connecting the origin to the polyhedron's corners, will have (x, y, z) termini. Face-tuples wire these termini into faces.[20] Once the face tuples are given, all unique edges might be distilled or derived.

Rather than run through (x, y, z) several times, as is the common practice, we could establish our Vector and Edge classes on a first pass, in tandem with vector arithmetic, and as Pythonic math objects. The groundwork would have already been laid, in the form of previous experience coding other math objects, the rational numbers for example (see below).

However even before we get to XYZ_t (t for time, i.e. a frame rate) another lexical-graphical bridge could be visited, much earlier in a student's career. I'm speaking of the figurate and polyhedral numbers, as treated by H.S.M. Coxeter (a student of Wittgenstein's at Cambridge) and by Conway and Guy in *The Book of Numbers*, and by Bucky Fuller in *Synergetics*.

Even without the apparatus of XYZ_t, it's easy to link a rule-based generator with a gnomon or shape. The square and triangular numbers, the cubic and tetrahedral numbers, become very short (perhaps polynomial) functions in the Python shell. Experience with functions and generators lays the foundation for later moving to class-bound methods and the object-self concept.

```
IDLE 1.2
>>> from __future__ import division
>>> def fibo(a=1, b=1):
        while True:
                yield a
                a, b = a + b, a


>>> fgen = fibo()
>>> to_phi = [1/(fgen.next()/fgen.next()) for x in range(11)]
>>> to_phi
[2.0, 1.6666666666666667, 1.625, 1.6190476190476191, 1.61818181818
18184, 1.6180555555555554, 1.6180371352785146, 1.6180344478216819,
1.6180340557275541, 1.6180339985218033, 1.6180339901755971]
```

Figure 3: Fibonaccis converging to phi

In other words, we have two ways to make the connection between algebra (lexical) and geometry (graphical): through polyhedral numbers when learning about functions and sequences; through polyhedra as "math objects" when writing some early class definitions, instantiating them as objects and visualizing them in real time or render time.

The Geography Connection

Some readers will be comfortable with this emphasis on abstractions. Others will chomp at the bit, wishing for stronger, more literal interpretations, which is where geography comes in, and web services like Google Earth's.[21]

Geometry may degenerate into a game for mystics and Pythagoreans, is always a source of fascinating relationships, but we mustn't forget the mundane empirical requirements of the real estate industry, the need for surveys and reports, titles to lands. Vast repositories of geographical information still need to be compiled and systematized. Given that the Earth isn't static, this need to gather intelligence, to feed the models and perpetually re-visualize them, is ongoing.

It's in conjunction with geographic information that I recommend introducing such topics as SQL and relational databases. We're also in the world of energy transactions, as the bioregions work to sort out their many interdependencies in terms of world trade.

However, we shouldn't look at this sudden shift into economics and/or general systems theory as a complete break with the earlier, more Platonic approach. The bridging concept is again polyhedra. Polyhedra "connect around in all circumferential directions" as Bucky liked to put it. They're like globes in that way. His geodesic dome derived from a complete geodesic sphere, from which likewise derived the Geoscope (a kind of electronic data-displaying globe) and the Fuller Projection.

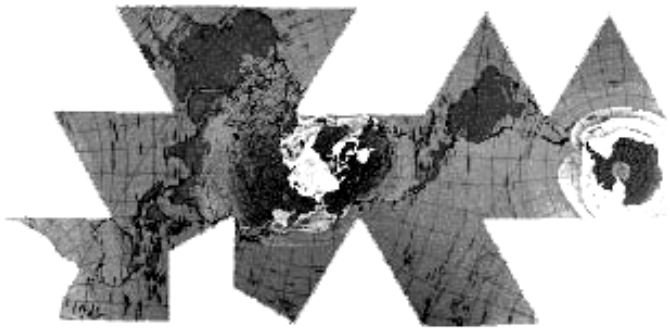

Figure 4: One Island in One Ocean Dymaxion World Map

Supermarket Mathematics

It's when we're zoomed out and considering the entire planet as a spaceship ("Spaceship Earth" was Fuller's coin) that we're well positioned to consider the energy economics of the "solar gradient" as some call it. Terawatts of solar energy get absorbed by the Earth, and most of it gets reflected back into space. A percentage, however, remains impounded within the Earthian biosphere, and a percentage of that is more specifically channeled into human affairs by agriculture and other energy harvesting.[22]

Having considered databases as repositories for geographic information (the latitudes and longitudes of the world's cities and villages for example), we now regard them as more at the service of supermarket clerks. We have inventory to track, cash income to monitor, and the real work of shopping intelligently, a sorting out and distribution process whereby individual families exchange time/energy to feed their lifestyles.

At this point games like *SimCity* and/or *The Sims* have obvious relevance, as we've started to discuss households and their ties to a marketplace and stores. At this zoomed out level, we're still thinking about energy and the solar gradient we're riding. Sunlight feeds biomass, which decays into byproducts. Most of the sun's energy goes back out to space, while we humans pioneer new circuit designs for "Motherboard Earth," and/or keep using the old ones.

SimCity and *The Sims* are likewise good for explaining object oriented concepts. A given Sim has a self, a lifestyle, characteristic clothing and personality traits. Likewise a city contains many structures stamped from the same template, yet each has its various individualizing attributes, such as its location, its occupants.

A single template may source any number of special case instantiations. We're pointing back towards the abstractions again. When we actually implement the visualization, both bodies and buildings turn out to be polyhedra under the surface.

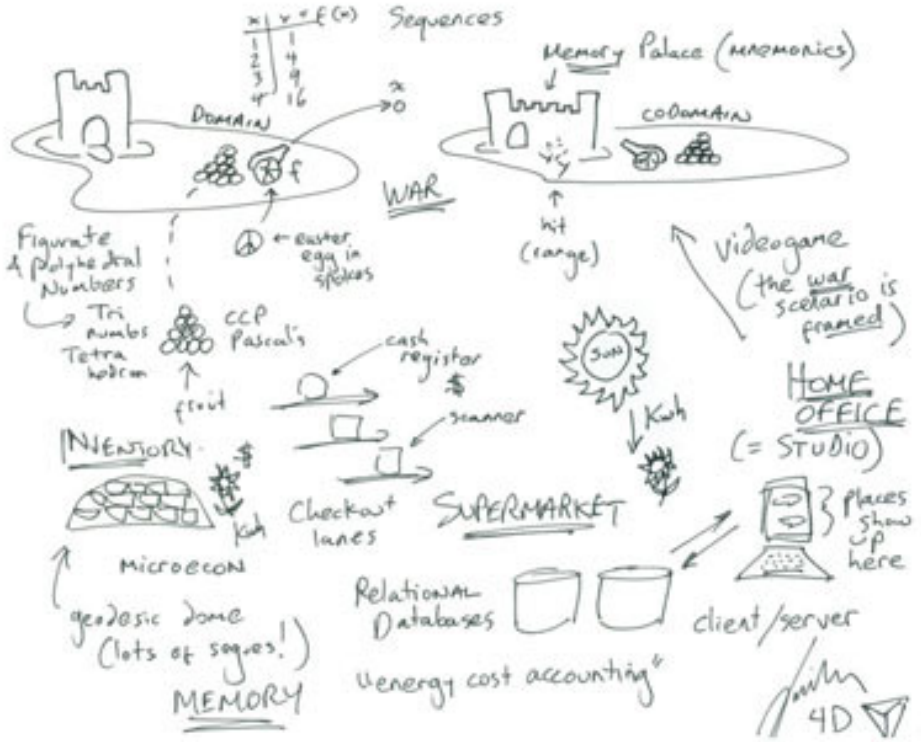

Figure 5: "A mathcast from ToonTown" (from a storyboard)

How does Python feature in all of this?

Perhaps Python has bindings to the game engines (which are also simulators) implemented by the game's designers.

Perhaps we use Python to write skeletal code for a supermarket type, importing other modules to define our cash registers and shelves, our payroll, our taxes.

Perhaps our shopping cart pushers will tend to queue in lanes at the cash registers. Shall we introduce multi-threading here then? Some teachers do.

Perhaps here is our segue to some sections on LAMP, with threads in Apache.[23]

Cardinality versus Ordinality

In his book *Number*, Midhat Gazale well distinguishes numbers according to their cardinal and ordinal uses.[24] Cardinal usage involves naming without ranking. We don't know which is more or less, but we know that they're different. Ordinality introduces greater than, less than, other such relative terms. We now have multiple axes along which to sort our objects. We're doing more than just naming what we find in the garden.

In the world of polyhedra, a similar distinction obtains, in that we're able to set them side by side, but mainly to distinguish them topologically, in terms of shape. We care about the different inventories of faces, edges and corners, how these come together, but without worrying about relative volume, which needn't be set in stone nor even discussed.

But once we *do* set in stone, create a sculpture, ordinality kicks in. How shall we rank these polyhedra in a hierarchy, from smallest to largest? Many ways are possible. One solution: call the edge of each shape "unity" and compute volume accordingly. Another solution: do pretty much what we just did, but make sure shapes that are dual to one another have intersecting edges, not necessarily same-length ones.

Here is where R. Buckminster Fuller came up with a pretty good design: make the tetrahedron "unity" in some sense, because it's topologically speaking the simplest polyhedron, and relate other shapes to that propitious beginning.[25]

The "duo-tet cube" falls out from the tetrahedron and its dual (edges intersecting, as proposed above), the octahedron falls out as the dual to that, along with the rhombic dodecahedron as the cube-plus-octahedron combo (they appear as the twelve short and twelve long face diagonals of each rhombus).

If we make our starting tetrahedron "unit volume," then those other shapes, the cube, octahedron, and rhombic dodecahedron, have volumes three, four and six respectively.[26]

The volume-six dodecahedron, a non-Platonic and favorite of Kepler's, is both a space-filler and a domain for each of the CCP's closest-packed spheres.[27]

Edge-connecting those sphere centers through the twelve K-points of intertangency (where the rhombs' face diagonals cross) nets us the octet-truss, an important space-frame, found in human architecture as well as in crystals. We may treat this scaffolding on a par with XYZ's, in terms of using it to provide a "holodeck" or "spatial grid" or "coordinate system" for our studies.

So when writing a polyhedron module in Python, one might choose to make *self.volume* a default attribute, assigned at birth during the construction process (__init__). Scaling a shape by a linear scale factor would then cause the corresponding volume to change as a 3rd power of that scale factor. Halve the edges and reduce volume to one eighth of what it was, double them to increase volume eight-fold. Likewise surface area varies, as a 2nd power of the linear scale factor.

We implement these first, second and third powering changes at the superclass level i.e. it doesn't matter what the specific Angles are (what the shape is), these Frequency (size) relationships will always pertain.[28]

Number Sets

The curriculum I envision takes the Model-View-Controller design pattern as a framework for exploring and discussing the tools we use to guide our thinking, musical and logical notations included. Historically, our concept of number has progressed through many levels, a journey summarized by the letters N, Z, Q, R and C.[29]

The story doesn't end with C (the complex numbers, not the computer language – another example of a name collision), but we do want to take it *at least* that far in our telling. In talking about the complex plane, we'll get to tie back to our earlier discussion of sequences (including those figurate and polyhedral sequences), their convergence and divergence, periodic and aperiodic behaviors.[30]

In Python, Z may be regarded as a superset of all instances of the built-in type *int*, while adding type *long* contributes to this intersection.

The rational numbers Q, expressed as an equivalence class of relatively prime pairs ("lowest terms fractions"), is non-native to core Python (in the sense of not a built-in). This presents us with an opportunity: to implement rational numbers in Python to reinforce our understanding of fractions and their operations. Here'd be some 8th grader's chance to encounter Guido's famously compact version of Euclid's greatest common divisor algorithm:

```
IDLE 1.2
>>> def gcd(a, b):
    while b:
        a, b = b, a % b
    return a

>>> gcd(12, 561)
3
```

We consider type *float* an "approximation" of the real numbers (R), or better yet, a number type unto itself, as is Python's newer, context-sensitive *decimal* type. Arbitrary precision decimals give us a powerful tool for experimenting with Ramanujan type sequences (partial sums approaching pi and so on).[31]

$$\frac{1}{\pi} = \frac{\sqrt{8}}{9801} \sum_{n=0}^{\infty} \frac{(4n)!}{(n!)^4} \times \frac{26390n + 1103}{396^{4n}}$$

Figure 6: One of Ramanujan's convergent generators for pi

Mathematics as an Extensible Type System (ETS)

At the end of the day, we want graduates of our program to feel comfortable with the idea of types and classes, and their endless extensibility in terms of what has gone before. We have plenty of room to be original, yet we don't have to start from scratch.

Python, the computer language, has given us real insight into what this looks like in practice (importing, from the Standard Library or elsewhere), but in the process of analogizing and abstracting so much, we're now prepared to let go of Python in particular, or any specific notation, and gaze thoughtfully at a pure vista, a principled realm, hard to permanently summarize visually perhaps, but definitely worth further investigation and modeling.

Concluding Philosophical Remarks

Kurt Gödel's purpose, in proving his Incompleteness Theorems, was to leave the door open to Platonism, which in somewhat cursory form means taking abstractions seriously, seeing temporal special case phenomena as but "pale shadows" of a more ethereal and eternal domain.[32]

Comparatively recent valuations along these lines, or namespaces, are Werner Erhard's Being/Mind distinction and Bucky Fuller's Mind/Brain distinction. In each instance, a second term (Mind and Brain respectively) represents a creature of habit, somewhat reflex-conditioned and robotic. In both cases, this second term "self" and/or "slave boy" (per Socrates) remains connected to a first term of higher rank (Being and Mind respectively) by means of intuition, a muse, divine grace or whatever (Erhard: by a

willingness to assume responsibility for, to be a root cause of, at some deeply existential level well beyond the reach of guilt or blame).[33]

Computers, being programmed machines, logically serve as stand-ins for whatever is automatic about life, including unconscious, unreflective thinking. Sir Roger Penrose, in considering the Platonic Realm to be non-computable, in the sense of not closed under any given set of rules (all such systems are incomplete), regards humanity as bearing some responsibility for synthesizing new truths, perhaps proving them retroactively to be *a priori*. His too is a synergetic thesis then: we gain access to the Platonic Realm by transcending our mechanical nature.

According to this way of reading the stars, I see this emerging curriculum continuing to place emphasis on the individual programmer as a principal source of intuitions and new insights. We're not in a mad rush to automate everything and then sit back and relax, as machinery caters to our every need. However appealing such imagery may be, at least to some at some times, humans apparently serve an integral function, one which our machines cannot adequately fulfill. Computer science needn't enslave itself to the AI camp, which promises to make humans redundant, but never delivers on that promise.

I accept that my concluding remarks put me in a camp, and that philosophy is in some sense an eternal war among encampments (schools). This is as it should be. We're not *ever* expecting to come to rest in some comfy set of purely reflex- or rule-driven beliefs in this model, and that's *good* news, not bad.

ENDNOTES

[1] http://www.4dsolutions.net/presentations/urner_europython4.pdf

[2] http://www.bfi.org/bfi_community/pythonic_mathematics_talk_by_kirby_urner

[3] http://worldgame.blogspot.com/2007/01/reviewing-my-oscon-2005-talk.html

[4] http://www.saturdayacademy.org/

[5] http://worldgame.blogspot.com/2007/06/pro-python-propaganda.html

[6] Cite conversations with Arch Davis of Princeton, C.S. Lewis fan, regarding the religious conversion and/or fervent resistance Dijkstra encountered when preaching his new gospel to already-seasoned programmers. He saw guys storm out of the talk, enraged at the notion they weren't structuring their programs properly.

Note: Dijkstra and van Rossum are both alumni of *CWI*, originally the *Mathematisch Centrum* in Dutch, a kind of Institute for Advanced Study in Amsterdam.

[7] Per a presentation on Zope 3, I forget by whom, at Free Geek, Portland Python Interest Group (PORPIG). I also enjoyed a Plone sprint in Vancouver BC, with Alan Runyan, Andy McKay and other luminaries.

[8] Part II of Wittgenstein's Philosophical Investigations was most focused on what psychologists call "gestalt switches" – i.e. switching a gestalt may be the hard part of getting someone's meaning, is not always doable, except maybe for some talented translator.

[9] Kaufmann shared his enthusiasm for *est* as a part of his banter with undergrads, hosting in his home; David Raymond completed the registration process for me. Later I served as a Centers Network volunteer, rising through the ranks, in the New York Area Center, then housed within some east side bus terminal building.

[10] These two shared a speakers' platform a few times, including at Madison Square Garden, New York City. The "self-styled pirate" allusion is to *Operating Manual for Spaceship Earth* wherein Bucky casts those with overview as the great pirates, and himself as someone with overview – or maybe he was more the "Leonardo type" always at the captain's side?

[11] my copy a gift from the author. See: http://mybizmo.blogspot.com/2007/03/synergetics-dictionary.html

[12] Per my conversation with Johnny Stallings, actor, at *Common Ground* on Hawthorne

[13] my biography of the guy is available here: http://www.grunch.net/synergetics/bio.html

[14] http://mathforum.org/kb/message.jspa?messageID=5749045&tstart=0

[15] "right next to the Republic of Perl" I sometimes add in response to raised eyebrows

[16] The funding was enough for Guido to launch IDLE, a capable cross-platform graphical shell for the Python interpreter. I'd say DARPA certainly got its money's worth.

[17] cocktail party discussion with FORTRAN compiler designers at OSCON 2005, plus per a SciPy talk mentioning Python bindings to trusty FORTRAN libraries

[18] See my *Vegetable Group Soup* and/or *Python for Wanderers* papers at my 4dsolutions.net

[19] Philosophers of mathematics will recognize the "primary versus secondary characteristics" meme.

[20] See for example: http://www.4dsolutions.net/ocn/python/polyhedra.py

[21] Integrating Python with Google Earth and GIS concepts was my theme in 2005-06 when I taught the 8th graders at Winterhaven Public School as a volunteer. See: http://www.4dsolutions.net/ocn/winterhaven/

[22] This vision of humanity "riding the solar gradient" is more fully rendered in *Into the Cool: Energy Flow, Thermodynamics, and Life* by Eric D. Schneider and Dorion Sagan (University of Chicago, 2006).

[23] LAMP referred to Linux + Apache + MySQL + "a P-language" (PHP, Perl or Python), a common platform for e-commerce sites. Since Ruby and the Ruby on Rails framework, the LAMP acronym is had to stretch a bit.

[24] Princeton University Press, 2000.

[25] The many ways in which collaborations around open source software have helped us pursue studies related to Bucky Fuller's pioneering work was the subject of my talk at OSCON 2005, as summarized in my blog: http://worldgame.blogspot.com/2007/01/reviewing-my-oscon-2005-talk.html

[26] For more on these primitive volume ratios, see my http://www.grunch.net/synergetics/volumes.html

[27] CCP = cubic close packing, although cuboctahedral close packing might be the better description

[28] Core to Synergetics is the angle/frequency distinction, the former relating to shape independent of size, the latter relating to size and therefore physical energy attributes. There's resonance here with the class/object distinction. I've used 4D/4D++ as nomenclature, with 4D "above the line" in pure principle.

[29] …the Natural, Integer, Rational, Real and Imaginary number types.

[30] I haven't talked a lot about cellular automata, ala Wolfram's new kind of science or John Conway's Game of Life, but topics of chaos and "chaordic self-organization" (Dee Hock) could branch off at this juncture.

[31] My daughter Tara likes Mark Engelberg's song about him: http://www.archive.org/details/Ramanujan

[32] Cite conversation with and subsequent lecture by Rebecca Goldstein, leading interpreter of Gödel and Spinoza, plus a former grad student at 1879 Hall (the philosophy building at Princeton, our in-common degree focus and alma mater) http://controlroom.blogspot.com/2007/05/incompleteness.html

[33] The fact that these two high voltage philosophies used "mind" in a pivotal role, but with almost opposite meanings, resulted in some turbulence at the time – another good example of a "name collision" resolved by the "namespaces" strategy for making key words more context sensitive (more system specific).

Figure 6: "__rib__ syntax" (notebook doodle)

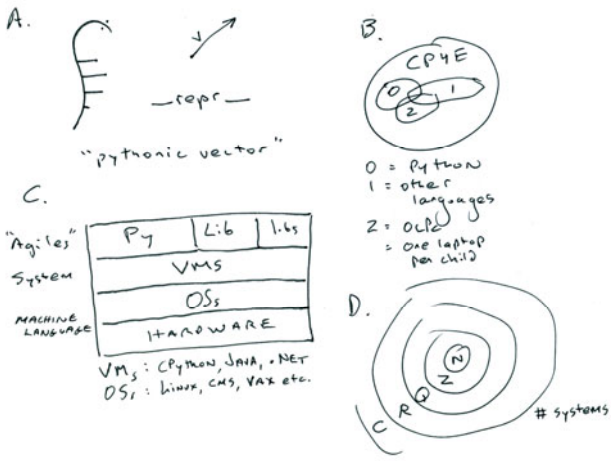

Figure 7: "cave paintings"