If I Could Save Thyme in a Baggie

Paul R. Potts

18 Aug 2020

Tuesday

The week is underway. My stress levels, and pain levels, have dropped a bit, and that's been a relief.

On Sunday evening we had a meal out on the back deck, with a fan blowing to keep mosquitoes at bay. The boys and I were already slathered in eucalyptus oil bug repellent, which we had applied earlier in the day, when I took them for a walk around the perimeter of our woods. Grace ran out to get some barbecue, and made us a salad of tomatoes and curly parsley from our gardens. We've been bad about harvesting sprigs from the parsley plants regularly, especially since the parsley plants have tended to get covered up by the sprawling, and occasionally collapsing, borage and tomato plants, but the parsley plants don't seem to be much worse for wear, and they've been growing much faster than they did early in the season. Over time they seem to become a bit tougher, but they are still bright green and taste fine, especially after being marinated with the tomatoes and dressing. The parsley plants did very well planted in the holes in the concrete blocks that make up the border of the kitchen garden, growing to a convenient size without fighting to invade the personal space of their neighbor plants. This salad wasn't tabouli, one of my very favorite salads, but it was still delicious. Joy brought some summer squash, and so we had a dish of fried-up squash, too, delicious, and she joined us on the deck at a separate table, in order to maintain a bit of distance. It's difficult to get the young children to keep their distance, but it worked all right. We also popped open a bottle of mead flavored with mint, made by our friends the Martins. This one was really good — a bit sweeter than the previous one, but with a more delicate flavor.

Yesterday I dove into improving some Python code for a work project, and I was reminded all over again why, although I like many things about it, Python is not my favorite programming language, with tooling that still, in 2020, often feels more like a hobby project rather than a piece of critical infrastructure. I spent an embarrassingly long time trying to determine the cause of a bug that resulted in my program generating data files that were incorrect, and inconsistent with the previous version of my program. I'll explain that below for nerds who might be interested.

Python 2020: Lots of Movement, Little Improvement

I'm operating on the most-widely-used PC operating system, and trying to do something that should be fairly simple: write a "file filter" program which will allow me to drag files onto it. The program should run, open the file that was dropped onto it, verify that it has the expected format and content, and then generate an output file. This is something that has long been possible on other platforms. On MacOS, even old-school versions of MacOS before it became the BSD UNIX-based MacOS X, one easy way to do this was to create AppleScript "droplets," but there were other ways; I recall writing programs in Perl that would support this.

It's been surprisingly hard to make this work, and I still don't have it working. I wasted a lot of time reading blog posts and Stack Overflow questions and their answers and messed with file associations, edited the registry, killed and restarted the Windows Explorer program, etc. It still doesn't work. I also tried following various blog posts telling me how to turn a Python program into a Windows executable. But it turns out those methods don't work. I tried "py2exe," but it turned out that the project is hopelessly outdated. I tried "PyInstaller," and that didn't work, and the obscure error messages I got gave me no help at all. I tried "cx_freeze" and that also crashed; at least by searching on the errors, I was able to find out that this is a known bug. I tried a suggested workaround, installing a development version of "cx_freeze," but *that* blew up with a blizzard of error messages and I'm trying not to fall into the "sunk cost fallacy" of throwing more good time after wasted time.

I could try to "downgrade" my Python version by a few years, but then I'd have to rewrite some code, again, which I only recently rewrote because some library functions were deprecated in the latest Python. This has been a constant problem with Python over the years; a lot of the libraries, and even some of the core language, hasn't been stable, and this language is *not* new; version 1 came out in 1994.

On to my bug.

The bug happened when calling this relatively simple Python function. This function generates a Python byte array, which is a simple data structure that just grows as needed to hold a series of data bytes. Python has opaque data types that allow programs to be written with a high level of abstraction. That's great, but because I'm generating binary data files, sometimes I need to write code that operates on low-level data types with exactly the sizes and behaviors that I want. The NumPy (numeric Python) library provides some precise data types such as **uint32** that allow me to do what I need:

```
from numpy import uint32

def serialize_msbin_eeprom_record_header( record_data_size : uint32, record_data_checksum :
    msbin_record_header = bytearray()

    msbin_record_header.extend( uint32( 0x80000000 | AT25M01_HIGH_QUARTER_BASE_ADDRESS ) )
    msbin_record_header.extend( record_data_size )
    msbin_record_header.extend( record_data_checksum )

    return msbin_record_header
```

The bug was cropping up in the last line before the return statement, the one that operates on record_data_checksum. The byte array was being extended by eight bytes, not four. What?

I was calling this function from code that looked like this:

```
elements_v1_data_checksum_uint32 = uint32( sum( serialized_elements_bytesio.getvalue() ) )
elements_v1_extended_checksum_uint32 = elements_v1_data_checksum_uint32 + sum( serialized_v1

msbin_bytes.extend( serialize_msbin_eeprom_record_header( elements_v1_extended_size_uint32,
```

The first line starts with a **BytesIO** object, which is basically an in-memory binary file object; calling **getvalue()** on that object returns a byte array, and the **sum()** function, when passed a byte array, creates a byte-by-byte checksum.

The second line adds to this checksum to create an extended checksum. It sums another byte array and adds the value to the previous **uint32** object.

Then, I create a **uint32** object from this sum. This **uint32** object then gets passed to the **serialize_msbin_eeprom_record_header()** function I described above.

The function parameter has a *type annotation*: **record_data_checksum : uint32**. This tells the Python interpreter (or compiler) that the second parameter is, or at least ought to be, a **uint32** object. Type annotations exist in many other languages and have for decades. Python is a *dynamically typed* language, similar to Apple's Dylan language, but just recently has gained this new type annotation feature. Dylan had optional type annotations in the early 1990s: if you didn't specify types, the compiler would generate code without doing compile-time type-checking, instead doing run-time type checking. This allowed developers to use it more like a scripting language, writing code quickly without worrying about the exact types they were using. Then, the code could be tightened up later: the development environments had features that would indicate "hot spots" where the generated code was less efficient due to this run-time type checking, to help programmers add strict tying where it would be useful.

What was really happening in my code? The **uint32** object was being converted to a **uint64** object when I added another value to it, and then this **uint64** object was being sent to the **serialize_msbin_eeprom_record_header()** function instead of the **uint32** object it was expecting. And Python was fine with this, because the type annotations don't actually **do** anything when you run the code under "CPython," the standard and most widely-used Python interpreter. These annotations are designed to be used with separate type-checking programs that aren't part of the standard Python distribution for Windows.

So why did this change to **uint64** happen? Let's try creating a simple piece of code that replicates the problem:

```
from numpy import uint32

num_1 = uint32( 1 )
num_2 = uint32( 2 )
num_3 = 3
num_4 = num_1 + num_2
num_5 = num_1 + num_3
print ( 'num_4 value: ' + str( num_4 ) + ", type: " + str( type( num_4 ) ) )
print ( 'num_5 value: ' + str( num_5 ) + ", type: " + str( type( num_5 ) ) )
```

This prints the following output:

```
>num_4 value: 3, type: <class 'numpy.uint32'>
>num_5 value: 4, type: <class 'numpy.int64'>
```

These small values can't be overflowing. Instead, what we are seeing is the effect of rules designed to prevent overflow. Different languages do this in quite different ways, and so it was not immediately obvious to me, as I typed the original line of code, what would happen.

In languages like C, unsigned integer types by definition are allowed to overflow or underflow. If you have a 32-bit unsigned integer that is holding the maximum possible 32-bit value, **0xFFFFFFFF** in hexadecimal or 4,294,967,295 in decimal, and you add one to it, the value will "roll over" and become zero. The C standard guarantees this behavior.

This rationale is complex, but the short version is that back when computers were large and slow, doing otherwise would have required runtime checks, in software or in hardware, to detect overflows or underflows. And so this behavior was enshrined in the C standard, and developers like me are accustomed to it — indeed, we rely on it all the time.

But what about the **uint32** type in Python's NumPy library?

Well, the NumPy documentation is pretty vague. It says:

> The behavior of NumPy and Python integer types differs significantly for integer overflows and may confuse users expecting NumPy integers to behave similar [*sic*] to Python's **int**.

I know that Python's **int** handles overflow gracefully and safely, with some cost in efficiency.

But it doesn't give much detail describing how NumPy types actually work.

There are a couple of possible ways that the library could handle possible overflow conditions when adding two **uint32** objects:

- The value could simply be allowed to overflow, as in C. Because NumPy is designed for speed, to provide a faster alternative to Python's general-purpose data structures, this is the behavior I would have expected. The comment in the NumPy documentation suggests that this might be the case because NumPy integers "may confuse users" and don't behave like **int**.
- Adding two **uint32** objects could always generate a result with the next-larger type, **uint64**, no matter how large the actual numbers in those objects are. That would be simple but incur some space overhead whether it is needed or not.
- The generated code could actually look at the values in the objects, and if necessary, either return the larger type, or generate an error.

I didn't really know how the NumPy library handles this overflow, but I didn't think I needed to know, since I didn't expect that my checksums of small data structures would ever be large. I was writing a *script*, which is very problem-specific, rather than a *library*, which should be able to handle all boundary conditions safely. But let's take a quick look at how it works:

```
from numpy import uint32

num_1 = uint32( 0xFFFFFFFF )
num_2 = uint32( 1 )
num_3 = num_1 + num_2
print ( 'num_3 value: ' + str( num_3 ) + ", type: " + str( type( num_3 ) ) )
```

This prints the following output:

```
>./test.py:4: RuntimeWarning: overflow encountered in ulong_scalars
>  num_3 = num_1 + num_2
>num_3 value: 0, type: <class 'numpy.uint32'>
```

Interesting: the overflow was detected, but the program generated a warning, not an exception, and so terminated normally.

That's what happens when adding two **uint32** types. But that's not what my code was doing. My code was adding a Python **int** type to a NumPy **uint32** type. And that's where my expectations broke down.

```
from numpy import uint32

num_1 = uint32( 0xFFFFFFFF )
num_2 = 1
num_3 = num_1 + num_2
print ( 'num_3 value: ' + hex( num_3 ) + ", type: " + str( type( num_3 ) ) )
```

prints:

```
>num_3 value: 0x100000000, type: <class 'numpy.int64'>
```

And it does the promotion even if the values won't overflow:

```
from numpy import uint32

num_1 = uint32( 1 )
num_2 = 1
num_3 = num_1 + num_2
print ( 'num_3 value: ' + hex( num_3 ) + ", type: " + str( type( num_3 ) ) )
```

prints:

```
>num_3 value: 0x2, type: <class 'numpy.int64'>
```

In C, the behavior when using *mixed* types, unsigned and signed together, in expressions, is trickier. Integer types are "promoted" before operations are done.

It looks like the NumPy library attempts to replicate C's behavior in this case and avoid potential overflow conditions, and so the **int**, even though it contains only a small value, is promoted to the larger **uint64** type — which I didn't even import because I didn't plan to use it — before the addition is performed. I'd have to look at the source code for NumPy to figure out exactly what is happening, but the point is, I shouldn't have to.

So it seems like we have the worst of several worlds now; we know from our "RuntimeWarning" that the NumPy library can generate overflow warnings when adding two **uint32** objects, so there is code generated which does this range-checking. But when a **uint32** is added to a standard Python number, the return value is the larger **uint64**. Because Python is dynamically typed, from the interpreter's perspective, nothing has actually gone wrong. And then, even with type annotations, this type change isn't caught by CPython because type annotations aren't actually used for type-checking. I just get unexpected output.

By the way, I tried the 3rd-party **mypy** utility, which is billed as a "linter" for Python programs, and supposedly reads the type annotations, to see if it would have caught the problem. It didn't report an issue.

Python is now widely used in education, having displaced simpler and more tightly-specified languages such as Scheme. I'm trying to imagine a first-year or second-year computer science student trying to figure out this problem, when expressions can wind up with unexpected types, and the language and tools are not designed to help.

Python is also widely used in various industries; it's used very extensively at Google, and the language's designer was a Google employee for a time. With this kind of backing, one might think that Python might have had its foundations shored up and improved; it might have even re-tooled to take advantage of some advances in language implementation that came about in the 1980s and 1990s.

One of these big corporate backers might have invested money to make type annotations actually work to make the code safer and more efficient.

I've been programming and studying computer science long enough to have some understanding of language design, and I've lived to see the same mistakes repeated again and again, as the tools underlying our critical infrastructure emerge from informally-specified hobby projects, and language designers don't really ever seem to learn from their predecessors.

Can we hope for better, in the career and lifetime I've got left? I'm getting tired of wasting so much time debugging, and of the slow churn of language development that is really traveling without moving.

Wednesday

IKEA, Again

IKEA is now actually answering their phone at the Canton, Michigan location. Last time we went in person, they were out of both the bookcases and the standing desks I wanted to buy. The standing desks were supposed to come back into stock this week. Apparently they didn't. There's no word on when either the desks or the bookcases might be back in stock.

I mentioned that Grace and I had recorded another podcast episode, but did not include the link to the blog post with all the information. It's here.

I've also gone through some old memory cards and found two more recordings that I could turn into episodes. I listened to them today. One is from April 3rd, and it's just me, taking a walk. The second is from June 14th, the day before I went back to my job. I'll probably make an episode of at least the latter and possibly the former as well, although I am hesitating because right now it hurts to use my mouse or keyboard.

Ergonomics

Years ago I used those Apple ergonomic keyboards that came with wrist rests and were hinged to allow the keys to split apart. I found that they helped my carpal tunnel syndrome. But unfortunately these were not built to last; the electronics failed after only a few months of use. The product was quickly discontinued. I bought refurbished units, and went through two of them before giving up. They are very obsolete now, even if you can find a working unit, because they used the now long-defunct "Apple Desktop Bus" port, and computers that support that port are largely in landfills now.

Today I ordered a modern adjustable keyboard, in the hopes that it will provide a little relief from the pain I've been experiencing recently. It's an Ergodox EZ. It looks to be a lot nicer than the old Apple keyboard. And one of the reasons I chose it is that the iFixit web site gave it a repairability rating of ten points, the maximum. So, I have hopes that I might be able to keep it working longer than the old ones.

My Orwellian Bookshelf

Yesterday I received a boxed set of four large paperback books, Orwell's *Collected Essays, Journalism, and Leters*, edited by Sonia Orwell and Ian Angus, and published by Harvest Books. These particular collections are long out of print, but there are plenty of copies available on eBay and Alibris, and most of the essays are available elsewhere, including online. The arrival of this boxed set completes my long-planned shelf of Orwell's work, although at the moment, because I haven't been able to acquire more of the bookcases I want, I don't have an empty shelf for the books. They are:

This boxed set of *Collected Essays, Journalism, and Letters*, comprising:

- *Volume 1: An Age Like This 1920-1940*
- *Volume 2: My Country Right or Left 1940-1943*
- *Volume 3: As I Please 1943-1945*
- *Volume 4: In Front of Your Nose 1945-1950*

Two Modern Library editions, a very fat one and a very thin one:

- *Essays*
- *Animal Farm*

The Folio Society "Reportage" boxed set, comprising:

- *The Road to Wigan Pier*
- *Down and Out in Paris and London*
- *My Country Right or Left*
- *Homage to Catalonia*
- *Funny But Not Vulgar*

The Folio Society boxed set of Orwell's novels, comprising:

- *Nineteen Eighty-Four*
- *Coming Up for Air*
- *A Clergyman's Daughter*
- *Burmese Days*
- *Keep the Aspidistra Flying*

And one more volume, also from the Folio Society:

- *A Life in Letters and Diaries*

The Folio Society volume *My Country Right or Left* does not have the exact same content as volume 2 of the *Collected Essays, Journalism, and Letters*, although there is some overlap, including of course the essay of that name.

There is quite a bit of overlap between the *Essays* collection and the *Collected Essays, Journalism, and Letters* volumes, and between these collections and the two volumes of the "Reportage" set that are essay collections, *My Country Right or Left* and *Funny But Not Vulgar*. Ideally I would have been able to find a set of volumes that had everything without any overlap, but there is no such set, and in any case I acquired these volumes over time, opportunistically.

Animal Farm is a novella of only about 30,000 words, and so it was not included in the boxed set of novels; that's why I tracked down a copy of the very nice Modern Library edition.

It is a joy to dip into these volumes. I think I have mentioned that I read the kids *Down and Out in Paris and London* as a bedtime story. It's time to read them *Animal Farm*. There is a lot of material here that I haven't ever read. I've read *Nineteen Eighty-Four* and *Animal Farm*, of course, but I've never read any of his other novels. I've read some of the essays collected in the "Reportage" set, but by no means all of them. And I've never read *The Road to Wigan Pier* and *Homage to Catalonia*.

I wrote on Twitter that:

> Reading Orwell's essays, I so often have the shock of recognition that comes from reading an insight I have had, but which he articulated.

In "Why I Write," Orwell writes:

> What I have most wanted to do throughout the past ten years is to make political writing into an art. My starting point is always a feeling of partisanship, a sense of injustice. When I sit down to write a book, I do not say to myself, 'I am going to produce a work of art'. I write it because there is some lie that I want to expose, some fact to which I want to draw attention, and my initial concern is to get a hearing. But I could not do the work of writing a book, or even a long magazine article, if it were not also an aesthetic experience. Anyone who cares to examine my work will see that even when it is downright propaganda it contains much that a full-time politician would consider irrelevant. I am not able, and do not want, completely to abandon the world view that I acquired in childhood. So long as I remain alive and well I shall continue to feel strongly about prose style, to love the surface of the earth, and to take a pleasure in solid objects and scraps of useless information. It is no use trying to suppress that side of myself. The job is to reconcile my ingrained likes and dislikes with the essentially public, non-individual activities that this age forces on all of us.

The sunflowers, over twelve feet tall, have finally opened flowers, and there are heaps of herbs ready to harvest. I'm going out again, to "love the surface of the earth, and to take a pleasure in solid objects and scraps of useless information."

Thursday

Drunken Master (1978 Film)

Last night I watched the second half of *Drunken Master* with the boys. This film is one of Jackie Chan's early successes. Chan plays Wong Fei-hung, a historic figure in Chinese medicine and martial arts. Wong is a ne'er-do-well who is very good at the more traditional animal styles of Kung Fu. In folklore, Wong was taught by So Chan, nicknamed Beggar So. So Chin is played in this film by Yuen Siu-tien, who was in his mid-sixties, and who died in 1979.

These are old stock characters and there are a lot of stories about them; there's even an HBO Asia *wuxia* film film from 2017 about Beggar So, called Master of the Drunken Fist. *Drunken Master* is not really in the fantasy-oriented *wuxia* style, but is more of a Kung Fu comedy, in the genre sometimes labeled "chop socky."

The story is really just an excuse for many, many fight scenes. After Wong gets in trouble once too many times, his father disowns him, and throws him out. He discovers that despite his traditional fighting skills, there are martial artists out there who can handily defeat him with their unconventional individual styles. Wong is reduced to begging and stealing, literally reduced to running around in only his underpants, until Beggar So begins training him, initially against his will, as the training seems more like torment and humiliation. But we eventually learn that Wong's father asked Beggar So to train his son, and that the training will take one year.

Beggar So is a master of a form of Drunken Boxing called "The Eight Drunken Immortals." Drunken Boxing is real — it describes styles of Chinese martial arts where the fighters imitate the relaxed movements of a drunk person.

The film itself is a little problematic; we watched it and enjoyed it, but I can't whole-heartedly recommend it without providing a few details. The characterizations are extremely broad. Some seem bigoted, such as a character who is, I think, supposed to be Japanese, with grotesque buck teeth. The drunken master is actually an alcoholic, with a made-up red nose, and this is played for constant laughs, even when he has the shakes.

The fight choreography tends towards the sillier side of the genre. In the dubbed soundtrack, there are slapping noises heard whenever a fighter moves a fist or foot, even when it doesn't actually strike anyone. Imagine the Three Stooges if they did Kung Fu, and you'll have a partial idea of what some of the film's fight scenes are like.

Just about every dumb and gross gag imaginable winds up incorporated into the film. For example, in one scene, the bad guys punch Wong in the belly to make him vomit up, one by one, the courses of a huge restaurant meal he's just eaten and refused to pay for. That's a bit hard to watch, even if you have a strong stomach. There are testicle-crushing gags, gags involving hitting bad guys on the head with a hammer, and gags involving bad guys getting faces full of manure. There are very sexist moments played for laughs, although Wong is redeemed of his sexism somewhat in the final fight scene, when he finally learns to adopt the fighting style of Drunken Immortal and pedagogical archetype "Drunken Miss Ho," which he has previously refused to do. It's a weird take on getting in touch with one's feminine side.

So, I've got mixed feelings about the film — it's dumb as hell, and there is plenty to offend. I couldn't really blame anyone for turning it off. But it's also often funny, if a bit on the sadistic side. Chan and the other actors display incredible agility and speed in the fight scenes, even though the scenes are broadly comic. Chan also shows off his impressive strength in the scenes illustrating Wong's training.

I wouldn't want to leave the kids with the impression that drinking heavily will give them superb fighting abilities; Beggar So tosses Wong a gourd full of straight moonshine in the final fight scene, and only after chugging it down is Wong able to reach the highest level of drunken — literally drunken — boxing prowess. But I would also hate to deprive the kids of the chance to see such amazing fighting and training sequences.

The kids wanted to watch it with the English soundtrack turned on, because not all of them can keep up with subtitles yet. But I highly recommend turning on the Chinese soundtrack instead, and watching it with English subtitles. The excessively silly sound effects and speech styles were not part of the original film, and it is a lot easier to bear when these distractions are removed and the film is allowed to speak for itself.

A technical note on the film: we watched a used copy of the 2002 Sony Pictures DVD release of the film, which I found on eBay. The digital transfer does not look great. It doesn't seem like much restoration work was done, and there are sections of the film that contain visible scratches and dirt, especially in the first few scenes. It is watchable, but really could use a full restoration treatment. There are other editions out there, including a "Masters of Cinema" region-free restored Blu-ray, but these editions are much more expensive, and since I have not seen them, I can't vouch for them.

Garden Update

The Amish Paste tomatoes are ripe and last night Grace roasted a big tray of them, to blister the skins and concentrate the flavor a bit. Then this morning she pulled the skins off and will put the tomatoes into the freezer to go into a future Sunday Gravy. We didn't actually cook them into paste this time. It's been hard to get a lot of time in the kitchen, and since it's been quite hot this week, no one has wanted to stand over a steaming pot in a hot kitchen. But they will add some great texture and flavor to slow-cooked sauces just the same.

We've also got a heap of Black Crim tomatoes, but fortunately they haven't been ripening faster than we can eat them. So every night I've been eating fresh slices of these delicious monster tomatoes sprinkled with a bit of salt and pepper and nothing else.

I've been harvesting herbs while we can. The dill is producing only seed heads now, so there are no more tender leaves. The basil is probably just about done for the year. Most of the basil plants appear to be dead or dying, although we might get a few more good leaves from the lemon and lime basil plants. But there are more herbs that we can eat now or try to preserve.

I've made a number of rolls of butter mashed up with single herbs. I simply chop up fresh herbs and fold them into softened unsalted butter. Then I lay out three layers of plastic wrap on the counter, spoon out the butter onto the plastic wrap, and cajole it into the shape of a log, then roll it up tightly. Then I put these logs into the freezer. I've made them with caraway, thyme, sage, French tarragon, dill, chives, and both green and bronze fennel. I will make a few more with sage and chives, and maybe dill if there are some tender leaves left in the back yard, because I expect those kinds to be popular. I have not tried it yet, but I've read that plain pasta tossed with sage butter is delicious, and I know I am very fond of both dill and chives in potato dishes.

If I Could Save Thyme in a Baggie

We drying some herbs, such as peppermint and stevia, but I'm also experimenting with freezing whole sprigs of some of the herbs — the ones with less water content in their leaves. Last night I gave the French cooking thyme, silver thyme, variegated lemon thyme, orange thyme, and lime thyme severe haircuts. I stuffed handfuls of each kind of thyme sprig into baggies, squoze out the air, and then labeled and froze them. I'm told the frozen thyme won't be quite as flavorful as fresh thyme, but will still have more flavor than fresh. I also hacked the creeping rosemary plant way back and stuffed a baggie with rosemary sprigs. There's enough to stuff sprigs into many whole fish before roasting them, or to flavor many trays of roasted potatoes, or to sprinkle over whole chickens. After handling all these aromatic herbs, my hands smelled wonderful for hours.

I had to quit working on this project last night and come inside because the mosquitoes were swarming, but I want to freeze more rosemary varieties, and some whole sage sprigs, too. The parsley plants have grown only very slowly for most of the season, but right now they are shooting up rapidly. So I will freeze some parsley. One of my herb cookbooks suggests bagging up parsley and freezing it, and then when taking it out of the freezer, immediately smashing up the stiff frozen parsley sprigs while they are still in the bag, rather than chopping it. So I am looking forward to trying that.

I will try freezing some whole anise hyssop leaves, too.

Some of the herbs with softer leaves might require a different strategy — taking the leaves off of the sprigs, and freezing them in water, into ice cubes. This method might be especially useful for mint leaves that are off the stems. Folks also freeze herbs in oil, so I might try freezing oregano in oil.

(The Next) Tuesday

Well, my Sunday deadline came and went and I didn't get this newsletter finished, and then I didn't get it finished on Monday either. So this one is going out a couple of days late, and I am uncertain about Sunday's issue. A few things have happened so I will share those things briefly today and then get this wound up.

Crashing the Art Scene

On Saturday, Grace and I took the kids to the Detroit Institute of Arts. They have a special program in August to let groups in for free, but you have to reserve times in advance. They are limiting the number of people in the facility. So we loaded everyone up and off we went.

The staff was not screwing around when it came to COVID-19 safety, with arrows indicating the direction of foot traffic and signs everywhere promoting 6-foot distancing. Everyone was masked. So we put masks on everyone, including Malachi and Elanor and, of course, Elanor's doll. The two babies were not so good at retaining their masks, and seemed to think the masks were food, but we did our best.

Our first stop was in Rivera Court housing the famous Detroit Industry murals by Diego Rivera. I have seen the murals before, but I am always happy to see them again. They are stunning. I am so glad that they are still there, a landmark artistic and political statement. The courtyard has an enormous skylight, and so it is sunlight that illuminates the murals. This means that they look different at each moment in time. They were painted on plaster using the "fresco" technique and so the colors are still very vivid, but no doubt they are gradually suffering damage from the sunlight. It is humbling to consider. Instead of bricking up the ceiling to preserve them, the Institute has chosen to continue to present them to visitors the way they were originally created.

I really enjoyed our visit but there were some difficulties. With my injured shoulder I could not carry a baby in a backpack, push a stroller, or even pick up a baby. So I felt slightly useless. Keeping everyone moving and keeping everyone from wandering off or getting too close to the artwork is a continual strain, especially when one of the younger kids decided he was through with the D.I.A. long before the rest of us were planning to leave. We certainly didn't try to see everything. We took Joshua to see the Modern collection, which was disappointing, and then we took the kids to see the Japanese section. We wanted to look at a few more paintings on our way out — some by Van Gogh, a borrowed Frida Kahlo self-portrait, and some borrowed works by Dali.

The Kahlo portrait is intriguing — it shows her trademark unibrow and facial hair, and in this one she portrayed herself with a very long neck. In the background is a money. So we talked about symbolism in paintings and what it might mean. The works by Dali were small and hard to see clearly, unfortunately.

You know how they say "quit while you're ahead?" Yeah, we probably should have done that. And so we unfortunately had an incident which cast a pall over the day. Two of the older kids — old enough to know better — tried taking a selfie in front of a self-portrait by Van Gogh. Being awkward adolescents, they somehow lost track of exactly where their bodies were with respect to the paintings and so one of our children accidentally shoulder-checked a Van Gogh. That was really not ideal.

The frame is not the kind of frame you might pick up at Target and it was not hung the way you or I might hang a piece of art on a wall, and it was mounted behind protective glass or plexiglass, so the painting was not harmed, but it was a hugely stressful and embarrassing moment, and led to a very apologetic discussion with a security guard. And so from there we took everyone out of the building as quickly as we could, which unfortunately when traveling with seven children, means "with painful slowness." But no one was arrested and (we hope) no one will be appearing on social media in a viral video taken from security camera footage.

We've since had several long talks with our awkward adolescents. If we go back to the D.I.A., Grace will take a smaller group. It is just too stressful to take everyone.

Scotch Bonnet Time

The rabbits have taken to climbing right into the pepper bed and have been eating up the plants from below, even while the fruits above continue to develop and ripen. We've recently been seeing peppers with bites taken out of them, apparently spat out and discarded in favor of the leaves. So, we have now put a fence around the pepper bed.

There are more ripening, but so far we have only managed to pluck one undamaged, ripe, brilliant yellow-orange Scotch Bonnet pepper. I was very excited to taste it so I challenged all the kids to join me in slicing it up and eating it like an apple. Only one of my strong sons and strong daughters was brave enough (or, perhaps, foolish enough) to join me, but I am pleased to report that might Sir Joshua of the Stout Heart and Scorched Tongue didst dine with me upon that pepper. And he and only jumped around and waved his arms and guzzled coconut milk a little bit.

Scotch Bonnet peppers are quite hot — they are probably the hottest peppers that I'm willing to eat raw like this, but of course they also go into jerk sauces and salsas, and one will nicely flavor a whole pot of whatever you care to cook with it.

There are much hotter peppers now — ghost peppers and Carolina Reapers and other extremely hot peppers — but I don't actually want to wind up with anaphylactic shock, and some of these extremely hot peppers can do that to a person; they are better suited to chemical warfare than to cuisine, in my opinion. But this Scotch Bonnet was delicious. They have a fruity flavor alongside the slow-growing heat. The initial burn is pretty intense, but it fades quickly, and subsequent bites don't seem nearly as hot. And eating hot chili peppers gives me a buzz — an endorphin rush similar to a runner's high. Keep in mind that if you are going to chop up or handle these peppers, you should wear disposable gloves, because they will leave your hands red and irritated, and you'd better make sure your hands are completely free of any residue from the peppers before touching your face.

Mosquito Assault

It was threatening to rain all weekend — rain was in the forecast — and some storm clouds blew past but dropped almost no rain. So on Sunday evening I went out to harvest some herbs and water the plants and the mosquitoes were crazier than ever. I came back with a couple of dozen bites, including bites all over my forehead, which hadn't happened to me before, and I did not feel it happening at the time, but a mosquito had bitten my eyelid. My eyelid was very swollen. I was starting to have a mild fever and waves of joint pain from my body's immune response, so I took a big dose of Benadryl to knock it down.

It took the swelling and itching and joint pain improved immediately, and an ice pack on my eye did the rest. But the Benadryl made me pretty useless for the rest of the evening, and so I did not finish the newsletter. The babies did not want to let me sleep, and so despite the medication, I couldn't really get to sleep until after 2:00 a.m. I was planning to work from home on Monday, but Monday morning, bright and early, I got an urgent phone call from the office — they were having a problem with one of our instruments in producting, and it needed to ship today, so I needed to come in immediately and fix it. I had not entirely shaken off the Benadryl, but Grace made me a large coffee and I packed up a few things from my home office and got in to the office as quickly as I could.

And so I'm back in the office this week, at least for a few days. I brought some more gear back in. My shoulder is still very touchy. I had one of my co-workers help me bring some things in from my car. It causes me considerable pain to lift anything above my waist with my right arm, and it is painful at night. I think that means this is likely a rotator cuff injury, and those are unfortunately notorious for taking a long time to heal. I took an ice pack in to the office to use during a couple of times during the day.

Everyone is wearing masks at the office now.

Last night we harvested our first few potatoes! They were small, and there weren't very many, but we expect to find more in the other beds.

Have a great week!

About This Newsletter

This newsletter by Paul R. Potts is available for your use under a Creative Commons Attribution-NonCommercial 4.0 International License. If you'd like to help feed my coffee habit, you can leave me a tip via PayPal. Thanks!