Pete's Log: 100 Days Of Code Challenge (Round 1)

Day 19: April 7, 2020: Tuesday

gotokens package Preliminary version

Today's Progress: Developed my first github-hosted standalone Go package, for use with TweetCommit (among others.)

Thoughts: This was quite a heavy day, full of steep learning curves. Among my achievements:

- Relocated the development of the package to the relevant path (`src/github.com/pjsoftware`) of my `$GOPATH` folder. Up until now I've been working elsewhere because, well, because I come from a non-Go, non-git, primarily Windows background, and all of my development until recently has been within my SVN working folder tree. When I fully embraced the beauty (and horror) of `git` on the command line (for a while I was using `svn` to keep my `github` repository up to date; it works, but it is definitely *sub-optimal*!) I created myself a git working folder! (Hey, if it works...!)
- Developed a standalone package intended to be sourced via github. No doubt there are things missing--a *licence*, for instance--but I figured it was time to work out how to make this work, and the best way to learn is to do. One of the big decisions I had to make was what to call the repository. While I'm currently focussing heavily on `Go` , I *will* be playing with other languages too, so I ultimately opted for the 'go' prefix to the name. My first impulse was to do something like 'gopackages/tokens', but it seems `git` does not really handle (or allow) subrepositories such as I would have used with `svn` . (Comparing the two, `git` has both advantages and disadvantages vs `svn` !)
- Built a full error handling system within the package. Back on **Day 13** I made a preliminary foray into the wonders of extended error handling in `Go` . However, I was tacking it onto existing code, so it was a shoddy attempt at best--and I just did not *grok* it. Having built it in from the ground up, I think I have a much better idea of what it can do, and how to use it properly.
- Full unit testing was not only added after the fact, but built alongside my code from the ground up. This helped me find a few errors which I would otherwise never have detected until tomorrow--or, y'know, months down the road.
- I gained a huge appreciation for just how productive `VSCode` can be when developing and, specifically, testing `Go` code. I'd already discovered how much better a modern `IDE` could be than the old, simple syntax-highlighting editor I had been using for so long. Full code completion, syntax error detection, all those awesome things are great--and `Go` even specifies the formatting to use, so having it reformat your code as you go just eliminates the old formatting arguments/decisions. (They never really affected me when I was happily working in isolation, but I was always aware of them.) **Also**, when running your package tests from inside the editor, `VSCode` not only tells you the percentage of code coverage you have achieved, it also highlights the code that has and has not actually been tested! Invaluable!

TweetCommit Revised to use gotokens

Day 18: April 6, 2020: Monday

Sudoku Solver Day 10

Today's Progress: Refactored my first three solvers, and my cell collection code.

Thoughts: I actually got quite a bit done, without making any actual progress! Because I *know* that the fourth solver is going to be complicated to write, I decided, first off, to split each solver into its own `.go` file. In the process of doing this, I discovered that I no longer remembered how `solver03` works, and the code did *not* clarify anything. Because it now lived in its own file, it was much easier to refactor it in an attempt to make the code more readable.

This, of course, led to the solver breaking! I added a bunch of debugging print statements using the existing `showWorking` mechanism, and discovered that during the refactoring process I had managed to drop an important `else`. I fixed that up, and then decided that it made more sense for my `s3` to have its own independent `showWorking` code. More importantly, this then set the precedent for `s4` to be a lot more organised and easy to keep track of what I'm doing as I write it.

The entire process took longer than it should have, and involved a great wailing and gnashing of teeth, but it had the important result that my solver code, which was rapidly becoming cumbersome and monolithic, is now a lot more flexible, and as I move forward with more solvers, this will allow me a lot more flexibility to do whatever I need to achieve the required result.

Finally--or rather, initially, because it was one of the first things I did--I moved the `cellCollections` back into the `Grid` struct (where they originally existed before I broke them out!) I am pretty sure that in order to write `solver04` I will need to create duplicates of the current state of the data, and this will be easier if I just need to copy the Grid and don't have to also juggle independent collections.

Day 17: April 5, 2020: Sunday

TweetCommit Code Structure Refined

Today's Progress: Made the code more oop-oriented.

Thoughts: This was mostly a reorganisation of code to allow all the possible errors to be triggered *before* any non-reversible action has occurred. Making the `tweeter` package OO allowed me to easily split the init code from the tweeting code.

At the same time I moved the `main.go` file into a `cmd/TweetCommit` folder (as per yesterday's `Win-Spotlight` update) and added a `publish.sh` shell script to simplify building the `exe` file and copying it out to where it is needed. (Possibly a symlink would make more sense ... but we *are* still working on a Windows machine, even if we're currently using `git bash` to get around. A shortcut, maybe? I'll do some testing...)

Day 16: April 4, 2020: Saturday

Win-Spotlight Converted to Module

Today's Progress: Still trying to wrap my head around the benefits of `Go` modules.

Thoughts: It does seem to make my pathing work a little better--by forcing me to use the github paths to everything. Since I'm also trying to wrap my head around how best to use github to host `Go` packages, this seems to be a step in the right direction.

At the same time, I was wrestling with the fact that `go build` was, by default, creating a `Win-Spotlight.exe` file rather than the preferred `UpdateSpotlight.exe`. I could probably have renamed the repository to fix this, but I opted, instead, to use a `cmd/UpdateSpotlight` folder. This means I now have to `cd` into that folder to build my code, but maybe I'll add a `build.sh` to the parent folder to get around that.

Day 15: April 3, 2020: Friday

Log Incrementer increment-log.py Update 2

Today's Progress: Fixed the formatting of the date

Thoughts: While making my last tweak of `increment-log.py`, I spent a little time looking for a formatting code to use with `strftime` which would *not* zero-pad the day of the month. I found something online which told me I could use `%-d` instead of `%d`. Turns out this was not correct--and because that particular line of code is only interpreted at runtime, I would not find out until I tried to run the script today.

To get the format I actually wanted, I had to specifically check for the leading zero and remove it.

To ensure this does not happen again--well, I guess it's time to look into unit testing in Python!

Day 14: April 2, 2020: Thursday

TweetCommit Output Revised

Today's Progress: Revised output to better explain what is happening.

Thoughts: `TweetCommit.exe` is working okay, but it needs a little refinement. It actually *seems* to hang for a while, so I've added output to give a better idea of what is going on. (I *think* I know where the delay is happening, but until I run it with the new output enabled I really don't know for sure!)

Update: `git push` is where the biggest delay lies...

`increment-log.py` Revised

Thoughts: I thought there was something in my code which detected an attempt to add a date in the future to the `log.md` file, but apparently there was not.

Now there is:

```
    if date > date.today():
        print("Increment cancelled; new date would be in the future!")
        return
```

Day 13: April 1, 2020: Wednesday

Win-Spotlight Revisited

Lost Time: Okay, so I missed a few days between my last coding attempt and now. In that time I watched plenty of NetFlix and did zero coding. (You guys: `#TheOA` is amazing! Check it out!)

Today's Progress: Made a quick fix to file naming; added tests and error handling.

Wallpaper Naming

`Win-Spotlight` is one of the first things I wrote in Go. It looks for new `Spotlight` images (delivered on the regular by Windows 10) and copies them out of their delivery folder into a specified wallpaper folder. This gave me the opportunity to play with `INI` and `JSON` files in `Go` -- not to mention exploring working with `UTF16`. Plus it gives me a folder full of awesome wallpapers to use on my machine.

It also confirmed, once and for all, that I had to move away from using the old text editor I've been using for the last 18-odd years -- and, honestly, much as I loved it, that has been the best thing I could have done!

I noticed that, despite my efforts to intercept it in the renaming process, some of the files in my wallpaper folder had a doubled-up copyright symbol in their name. I renamed them there, but obviously the only way to truly fix the problem is by curing the illness rather than treating the symptoms. *(Signs that you are living through the **Coronapocalypse**: medical analogies in your log files!)* So my first work for today was to fix up that particular piece of code.

Unit Tests

The Plan:

So, did my fix actually do what I think it does? How would I know?

As somebody who is mostly self-taught, and who has done most of their coding in isolation*, I only picked up the concept of `TDD` (Test Driven Development) a couple of years ago. Great idea, loved it, and promptly started applying it (no doubt naively) to my `Perl` programming. It's probably time to work on doing the same in my ongoing `Go` work!

I probably won't write a completely comprehensive test suite today, but I'd at least like to get enough done that I can confirm my new code works as intended.

* This has shaped much of my approach to coding (obviously) and it will probably come up again!

Achieved: I added one unit test for `newFilename()` to ensure that our output is as expected. This enabled me to rewrite the function to be more efficient, and to put all our intended regexp transformations in a table. (I originally coded it as a `map`, but the order of transforms was important to the end result, so I redefined it as a `slice` of `struct`s.)

Error Handling