Software Archaeology

In today's recitation, we will practice getting to know an unknown codebase specifically in the context of fixing bugs in an open-source software system. We will do this by exploring bugs in Pandas, a popular python library for data analysis.

Download and build a development version of Pandas

Ahead of recitation (really, please, it takes a while to get it to build the first time, please download and build the source code for Pandas.

1. Clone the source from: https://github.com/pandas-dev/pandas
2. Follow the instructions at https://pandas.pydata.org/pandas-docs/stable/development/contributing.html#creating-a-python-environment to build the source and set up a development environment where you can import the development version of pandas that you build from source into a python environment.

The most reliable way we have found to do this is using the instructions for Anaconda, which is linked from those instructions.

The first build step can take a while! This is why we are encouraging you to do it ahead of time.

Confirm that your build works by following the instructions that start with "At this point you should be able to import pandas from your locally built version:" and confirming that the pandas.__version__ that prints out is a dev version (it will correspond to the one you built).

Although we confirmed that things worked from a terminal, I will assume throughout these instructions that you are using a reasonably configured IDE. :-)

Familiarize yourself with Pandas as an idea

You may optionally wish to peruse these docs:
https://pandas.pydata.org/pandas-docs/stable/user_guide/10min.html

Which can introduce you to pandas very quickly.

Let’s fix some bugs!

Pandas is a library, which means you import and use it from within Python. This gives you an easy way to test the code manually. Consider the following two open bugs in Pandas:
https://github.com/pandas-dev/pandas/issues/36308
https://github.com/pandas-dev/pandas/issues/28928

(we picked bugs that are not closed in master, and are marked as “good first issue.” At least one is associated with a pull request that would close it. Don’t look at that pull request!!! You’ll defeat the purpose of the exercise...)

Note that for both of these bugs, the submitters have included very clear steps to reproduce. You can also reproduce the bugs by hand. Let’s begin with 36308:

```
(pandas-dev) clegoues@clegoues-macbook-air pandas % python
Python 3.7.3 (default, Apr 24 2020, 18:51:23)
[Clang 11.0.3 (clang-1103.0.32.62)] on darwin
Type "help", "copyright", "credits" or "license" for more
information.
>>> import pandas as pd
>>> print (pd.__version__)
1.1.0.dev0+273.g0ca9ddd6e
>>> df = pd.DataFrame({"A": [1,2,3], "B":[4,5,6]})
>>> df.groupby([1,1],axis=1).transform("shift")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
<snip for brevity>
    self._data.set_axis(axis, labels)
  File
"/Users/clegoues/Desktop/recitation/pandas/pandas/core/internals
/managers.py", line 178, in set_axis
    f"Length mismatch: Expected axis has {old_len} elements, new
"
ValueError: Length mismatch: Expected axis has 2 elements, new
values have 3 elements
>>>
```

Task 1: reproduce both bugs, optionally first manually as above, and then by writing some kind of test. Pytest has been broken for us (see below), but you can also just put the code you entered manually in the interactive environment in mytest.py and run it as

```
(pandas-dev) % python mytest.py
```

...and that also qualifies as a test.

Tasks 2-N: Let's learn more about these bugs! Your high level goal is, for both bugs, to *identify the pandas code that is implicated in the bug and learn more about the bug en route to fixing them.* Note that we don't expect you to actually fix them, they're open for a reason, but we'd like you to learn about the code being executed and learn a little bit about what's going on.

We are treating this like a scavenger hunt, so document what you do and how much progress you make using this form: https://forms.gle/xN97kcw8mqgfVq9h7

(It's OK if you don't get to all of these tasks.)

Try some or all of the following tasks, and think about whether you're doing them *statically* or *dynamically:*

- Write a new test (for anything, not necessarily the bugs we're looking at) that passes (if pytest isn't working --- see below --- you can just write your own python that uses the pandas library)
- Change the pandas code so that that your new test fails (then undo your change)
- Change the code so that the corresponding error isn't thrown, or so that a different error is thrown.
- Change your tests such that an error *isn't* thrown.
- Find similar code to the code that's involved in the error.
- Find tests in the tests/ directory that seems to test any of the code in the above. Consider doing this statically, and then using dynamic approaches (i.e., even if you can't run pytest, can you copy code out of the tests into your own manual test to confirm?)
- Use your IDE to jump to a definition implicated in the code you're debugging
- *Optional because it may be broken:* We've been testing mostly by hand. Pandas also heavily uses tests and test-driven development. You may want to look at how to run tests using their framework, listed here: https://pandas.pydata.org/pandas-docs/stable/development/contributing.html#test-driven-development-code-writing
  ...But note that when we tried to run pytest (using python -m pytest, to make sure it runs in the environment), all of our tests errored, we think because they have an import

error in their current environment. YMMV. That said, it may be instructive for you to dig through their tests.

- Even if you can't run them, can you (statically) find a test that looks like it might test the same code you've been exploring in these bugs? Can you copy that code to make your own test, manually?

If you've finished all that:

- Try using the *debugger* in your IDE to set breakpoints and step through execution. Don't know how? Dig through your IDE's documentation and see what you can learn.
- Does your IDE have a way to compute *coverage* information, that is, which code is executed by a test or test suite? Read some documentation and see if you can learn how to configure and run it.

Here's a link to the progress form:
https://docs.google.com/forms/d/e/1FAIpQLSdyURaqYH-DEpk6DvsibXLjXb3tvtpqeznzdv6-aVV-Uvh3mQ/viewform