title: **Overwhelmed about consolidating code**
num_comments: 5
num_up_votes: 6
upvote_ratio: 0.8

Post Text

Hi everyone, I’m pretty new to programming (been consistently programming in Python for around a year). I’ve been having to use Jupyter notebooks for my internship, which is mostly making a bunch of graphs right now, but every time I need to consolidate my code, I feel enormously overwhelmed by the thought of having to combine everything together. I was wondering if anyone had tips for getting past feeling the overwhelm of cleaning up the code into one code block/file and adding better comments? Am I supposed to be doing a better job of combining on the way instead of all at once at the end? I’m looking for advice on handling both the emotional and technical process. I didn’t do CS in school, so I don’t really know about best practices for this kind of thing.

Comments

Commenter_2
ID: REDACTED! ~(o.o)~ <3, Upvotes: 5
I do two things:

1. Tooling that makes breaking things up as you go faster / easier
2. Decide on a set folder hierarchy and use it for each project / domain

I'm a web developer so my tools will probably be different than yours, but it's worth investing a few days at the least to make organizing your code feel delightfully copasetic with everything going into it's proper place. My strategy is just to do it as I go. It helps keep my thinking cleaner and more focused and then I don't have to untangle / break things at the end.

Tools:

VSCode IDE, and within VSCode I use prettier to style my code, intellisense to do super helpful autocompletion, I have a color plugin that overlays the actual color over a color's hexadecimal. I also make a lot of files of a certain type, so decided to use VSCode snippets to just put in boilerplate code with a three key shortcut instead of having to type it up every time.

Folder Hierarchy:

This will be different for you since you're in a different type of engineering, but you should decide ahead of time what folder structure makes sense for you. Here's mine from the root directory of the project. My strategy has two dimensions of organization: function and domain. This allows me to do a few things:

1. Differentiate between app-wide and domain specific folders in your folder hierarchy.
2. Have consistency in folder contents. For example, I know if I'm in any of the redux-config domain folders there will always be three files: slice.ts, reducers.ts, and actionCreators.ts. I know what's going to be in them, why they're there, even if they're just importing and exporting something.

index.tscomponents/- component1/- index.ts- component1.ts- types.ts

screens/- screenA/- index.ts- screenA.ts- types.ts- constants.ts

redux-config/- user/- reducers.ts- actionCreators.ts- slice.ts

\- games/- reducers.ts- actionCreators.ts- slice.ts

constants/

types/utilities/- validation/- isNameValid.ts- isEmailValid.ts- isPasswordValid.ts

\- authentication/- authorize.ts- getCredentials.ts

Unknown_User
ID: REDACTED! ~(o.o)~ <3, Upvotes: 2
I overwhelm myself pretty regularly with my own code as I'm a side-project/floater/random fixes/automation person, but still on our Core team. It requires writing new code, touching a lot of everything, planning small refactors that also touch everything, etc., and it can get overwhelming.

People here have recommended tools, but if you can't even get started, those won't help.

For me*in an ideal world, though I have my brain*the first thing I do when feeling the Overwhelm is pause. Sitting and spinning makes it worse.

The second thing I do is backtrack so that I understand my goal. Sounds silly, but the time I spend pursuing irrelevant code fixes is significant. I think it's a way to avoid the complex stuff because my brain tends to think about everything at once (probably like a lot of other people here). What's the actual problem you're solving, what is the endpoint you want to get to?

Then, I write pseudo-code as comments and fill in/copy+paste code around it. It's like writing papers; starting with an outline makes it feel less like you're staring at a blank page.

So without overthinking
the implementation (at least at first), write an outline of how you think the code should look/flow. What variables are used, why, what is this thing doing, etc. Then fill in the gaps. Also, now you have comments.

Lastly, code refactoring is an ongoing thing but you have to have the code to refactor first; vomiting the code out to do the things is better than producing *nothing at all* because you're stuck in analysis paralysis. Maybe I'm in a different boat than others, and tech debt *is* a thing, but in theory you can always refactor later.

> ###### Commenter_3
> ID: REDACTED! ~(o.o)~ <3, Upvotes: 2
> Hmmmm, I think I need to do a better job on writing pseudocode before I attempt things, which would help a lot with the comments. Thank you so much, this is really helpful and the exact kind of advice I was looking for :)

Commenter_4
ID: REDACTED! ~(o.o)~ <3, Upvotes: 1
Hi, I'm a contributor of LineaPy.
We're building a tool that solves this problem.
Our goal is to reduce the friction between developing Jupyter notebooks(or python scripts) and production codes.

One of the feature we have is extracting only the relevant code to the object you are interested in.
The object can be dataframes, figures, models, or any python object; we call these objects artifacts.

For instance, let's say we are working on a notebook(as following) that reads raw data and does some exploratory data analysis.
Then, we perform some data cleaning on the raw data and feed the clean data to train a model.
Finally, we want to output the model somewhere else(sharing with others or other downstream processes ...)

```
raw_df = load_raw_data()

do_some_eda(raw_df)
do_more_eda(raw_df)

df = cleanup_data(raw_df)
mod = fit_model(df)
```

The notebook might be quite messy at this moment because all the EDAs or developing process(modifying, re-executing, deleting, non-linear executing cells).
Cleaning the notebook might be quite overwhelm sometimes, and we think there should be an easier way to achieve that.

With LineaPy, we are able to solve this problem with minimal code change by adding three more lines of code in your existing notebook like following

```
# Load lineapy extenaion and import the library
%load_ext lineapy
import lineapy
```

```
# Your original notebook
raw_df = load_raw_data()

do_some_eda(raw_df)
do_more_eda(raw_df)

df = cleanup_data(raw_df)
mod = fit_model(df)
```

```
# Save the model object
lineapy.save(mod,'mod')
```

To extract the code that produce `mod`

```
artifact = lineapy.get('mod')
artifact.get_code()
```

```

and you will get following code block

```
raw_df = load_raw_data()
df = cleanup_data(raw_df)
mod = fit_model(df)
```

that contains only relevant code to generate the final `model` object.

We provide many other features in LineaPy; if you are interested in, you can find them in our GitHub page.

> ###### Unknown_User
> ID: REDACTED! ~(o.o)~ <3, Upvotes: 1
> Unfortunately, I'm not sure I can use your tool because of sensitivities with working for the government :/ but will definitely keep in mind for personal projects
```