

# Decoupled code

Good research code

Patrick Mineault

## Lesson 2

Keep things decoupled

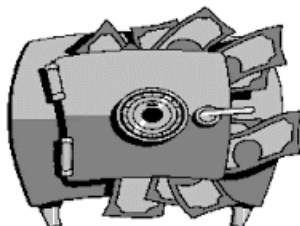
# Spaghetti code

## *Development AntiPattern:* **Spaghetti Code**

**spa-ghet-ti code** [Slang] an undocumented piece of software source code that cannot be extended or modified without extreme difficulty due to its convoluted structure.



Un-structured code  
is a liability



Well structured code  
is an investment.

# Do you know when your code smells?

- ▶ Maybe your code is written in a way where you're doing a little bit of everything all at once
- ▶ e.g. `wave_clus`
  - ▶ very useful software to sort spikes
  - ▶ has a GUI in Matlab GUIDE
  - ▶ GUIDE makes it exceptionally hard to write good code
  - ▶ Picked it because it's real code
- ▶ This stuff can happen in Matlab or in Python!

# Sample code

Link.

# What's going here?

This is a callback for a function in a GUI for spike sorting.

- ▶ Does many things at once
  - ▶ Manipulates the GUI
  - ▶ Modifies data
  - ▶ Reads a jpg file?
- ▶ Uses magic numbers and magic columns
- ▶ Uses various string formatting functions and `eval`
- ▶ Big function
- ▶ Not complex, but it's complicated

# Tightly coupled

- ▶ When code does a lot of unrelated things at once, it becomes very hard to reason about.
- ▶ Let's say your results are weird, are they weird because. . .
  - ▶ the data is bad?
  - ▶ you're loading the data wrong?
  - ▶ your model is incorrectly implemented?
  - ▶ your model is inappropriate for the data?
  - ▶ your statistical tests are inappropriate for the data distribution?

# Uncouple and simplify

- ▶ Keep each of the boxes separate with minimal interface
  - ▶ Separation of concerns:
    - ▶ Example: your data loading function should just load data
    - ▶ Your computation functions shouldn't load data, they should just compute
- ▶ Make each of the boxes small
  - ▶ Don't make giant monolithic functions
  - ▶ Make functions which are small
    - ▶ A screen's worth, 80 columns, 50 lines
- ▶ Avoid side effects, prefer pure functions



# What's a side effect?

*In computer science, an operation, function or expression is said to have a side effect if it modifies some state variable value(s) outside its local environment, that is to say has an observable effect besides returning a value (the main effect) to the invoker of the operation. State data updated “outside” of the operation may be maintained “inside” a stateful object or a wider stateful system within which the operation is performed. Example side effects include modifying a non-local variable, modifying a static local variable, modifying a mutable argument passed by reference, performing I/O or calling other side-effect functions. (Wikipedia)*

# Side effects

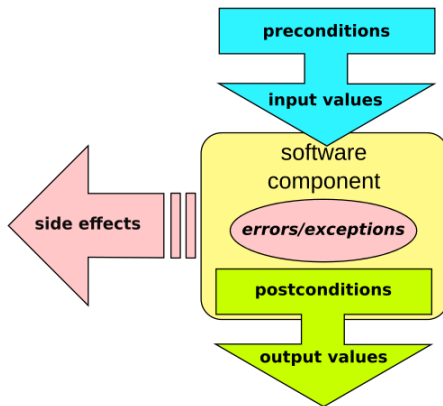


Figure 2: From Wikipedia

## A function with side effects

Q: what will be printed?

```
def reversi(arr):  
    """Reverses a list."""  
    for i in range(len(arr) // 2):  
        arr[-i - 1], arr[i] = arr[i], arr[-i - 1]  
    return arr
```

```
>>> a = [0, 1, 2]  
>>> b = reversi(a)  
>>> print(b)  
>>> print(a)
```

## A function which changes its arguments

```
In [12]: def reversi(arr):
...:     """Reverses a list."""
...:     for i in range(len(arr) // 2):
...:         arr[-i - 1], arr[i] = arr[i], arr[-i - 1]
...:     return arr
...:

In [13]: a = [0, 1, 2]

In [14]: b = reversi(a)

In [15]: print(b)
[2, 1, 0]

In [16]: print(a)
[2, 1, 0]
```

Figure 3: This function mutates its arguments

# Side effects

- ▶ Modifying arguments
- ▶ Printing
- ▶ Making API calls
- ▶ Changing globals

## Side effects are not the best

- ▶ Stuff happens outside of the normal flow from arguments → return value
- ▶ Need to know state of function to understand it
- ▶ Hard to test
- ▶ Let's box them
  - ▶ You can use closures or classes to encapsulate state

# Demo

- ▶ `fib.py`
- ▶ Fibonacci sequence,  $F(n) = F(n - 1) + F(n - 2)$
- ▶ Memoization

# Learn more about your language

- ▶ Sometimes (but not always!), code smells come from lack of knowledge
  - ▶ E.g. using magic column numbers in a raw numpy array rather than named columns in pandas because you don't know pandas
  - ▶ Using unnamed dimensions in numpy rather than xarray
  - ▶ Using `+` and bespoke casting for string formatting rather than the one true solution, the f-string
- ▶ Take time to learn more about the language you use
- ▶ Coming from Matlab? I have three tutorials: [1], [2], [3]



# Enough theory!

Let's de-couple CKA!

# Background on centered kernel alignment

Q: how can we compare how different brain areas and artificial neural networks represent the world?

A: Choose a standard battery of stimuli, measure responses across systems, compare the responses between the systems. Many approaches, including:

- ▶ forward encoding models (e.g. ridge regression)
- ▶ canonical correlation analysis (CCA)
- ▶ representational similarity analysis (RSA).

# CKA

Kornblith et al. (2019) propose a new method to compare representations. You can think of it as a generalization of the (square of the) Pearson correlation coefficient, but with matrices instead of vectors.

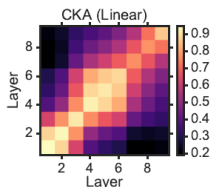


Figure 4: Alignment between layers of two neural nets initialized with different seeds

Importantly, CKA is not implemented in scipy or sklearn, github gives very few hits <sup>1</sup>... it's real research code!

# Centered kernel alignment

- ▶ We collect the responses of each system to our battery of  $n$  stimuli into matrices  $\mathbf{X}, \mathbf{Y}$ .
- ▶  $\mathbf{X}, \mathbf{Y}$  have shape  $n \times k, n \times l$ , and  $k$  and  $l$  are not necessarily the same.
- ▶ Center  $\mathbf{X}, \mathbf{Y}$  so each column has 0 mean, then:

$$CKA(\mathbf{X}, \mathbf{Y}) = \frac{\|\mathbf{X}^T \mathbf{Y}\|_2^2}{\|\mathbf{X}^T \mathbf{X}\|_2 \|\mathbf{Y}^T \mathbf{Y}\|_2}$$

- ▶ Min 0, max 1
- ▶ Check: if  $\mathbf{X}$  and  $\mathbf{Y}$  are one-dimensional, then  $CKA = \rho(\mathbf{X}, \mathbf{Y})^2$ .

# Open discussion

Q: What's not ideal about this code?  
`research_code.cka_not_great.py`

# Pain points

- ▶ IO, computation and plotting are all in one big blob
- ▶ Solution: isolate the computation in its own function independent of IO
- ▶ Put the controller in the `main` function, hide behind `__name__ == "__main__"`
  - ▶ Avoids module variables in Python
  - ▶ Makes the code importable

# Live coding!

(the result is `cka_step2.py`)

# You can apply this advice at a project-wide level as well

Advice from van Vliet (2020):

1. **Each analysis step is one script**
2. **A script either processes a single experimental replicate, or aggregates across replicates, never both.**
3. One master script to run the entire analysis
4. **Save all intermediate results**
5. Visualize all intermediate results
6. **Each parameter and filename is defined only once**
7. Distinguish files that are a part of the official pipeline



# Decoupling configuration

- ▶ Keep your configuration out of your code
  - ▶ Use `argparse` to specify options via the command line
  - ▶ Keep configuration options located in an importable `config.py` file
  - ▶ Use `python-dotenv` to store secrets in a `.env` file

## Lesson 2

- ▶ Keep things decoupled
- ▶ By keeping things decoupled, you can think about one part of your program at a time
- ▶ Save your WM slots
- ▶ Your 5-minute exercise: take existing code and wrap it in `main`