

# Code Archeology

Claire Le Goues

**Reminder: if you are in Sections A/B, fill out our survey so we can figure out what we're doing.**

# Technical homework question protocol.

- First: we want to help you, don't suffer for days!
  - Some of you don't ask *enough* questions.
- But: the online environment can make it almost a little, well, *too* easy.



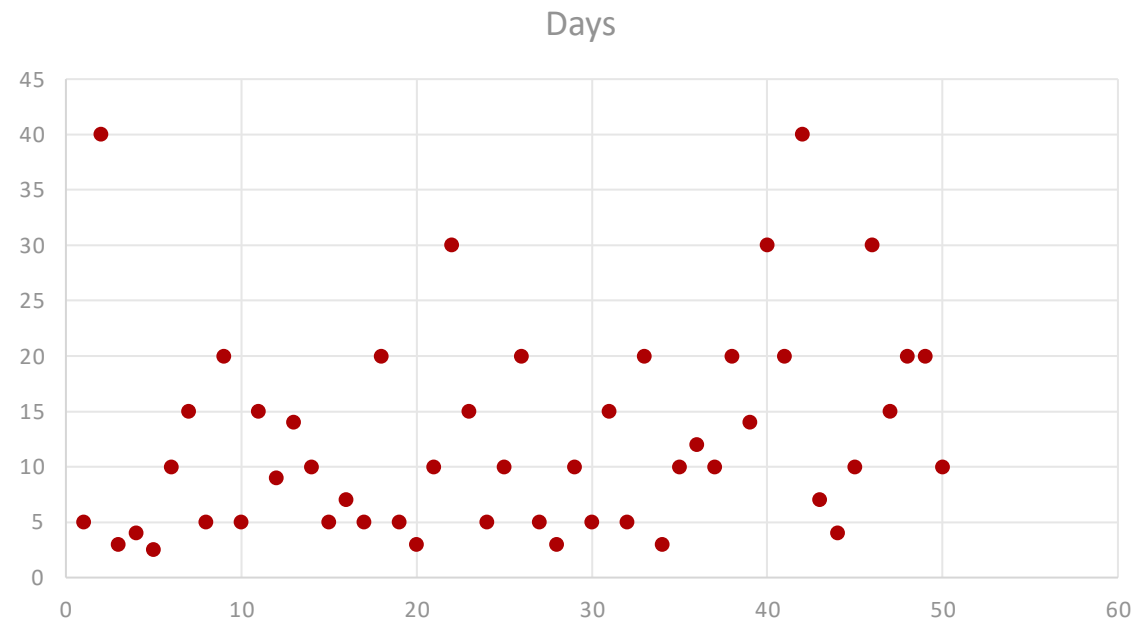
Chris is great! But there's only one of him, and 60+ of you...

# Requests for technical homework assignment.

- Please read the "protocol" I posted on #announcements
- Highlights:
  - Check the #homework-techsupport channel to see if someone else has had the same problem.
  - *Ask your question on #homework-techsupport.*
    - Do you REALLY want us to have both piazza AND Slack? No, you do not.
  - Once it is answered, *do not delete your question, that misses the whole point.*
  - If you insist on DMing instead, DM all three of us.

# Task: Estimate Time

- A: Simple web version of the Monopoly boardgame with Pittsburgh street names
  - Team: just you



- Left out: two outliers, 180 and 1125

# Learning goals

- Ask for technical help on homework effectively while being a good citizen.
- Understand and scope the task of taking on and understanding a new and complex piece of existing software.
- Appreciate the importance of configuring an effective IDE.
- Enumerate both static and dynamic strategies for understanding and modifying a new codebase.

Context: big ole pile of code.



**MAYAN**

...do something to it.

Like: Fix a bug, implement a feature, write a test...

**You cannot understand the entire  
system.**



## Goal: develop and test a working model or set of working hypotheses about how (some part of) a system works.

- Working model: an understanding of the pieces of the system (components), and the way they interact (connections).
- It is common in practice to consult documentation, experts.
- Prior knowledge/experience is also useful (see: frameworks, architectural patterns, design patterns).
- Today, we focus on individual information gathering via observation, probes, and hypothesis testing.

**TWO PROPERTIES OF SOFTWARE THAT ARE USUALLY ANNOYING THAT WE CAN TAKE ADVANTAGE OF.**

Software constantly changes → Software is easy to change!



Guess so!

Is this wall  
load-bearing?

Software is a big redundant mess → there's always something to copy as a starting point!



NYTimes quiz: <http://bit.ly/problemQuiz>

**BUT FIRST! AN EXERCISE.**

**Beware of cognitive biases.**

# Beware of cognitive biases

- **anchoring**
- **confirmation bias**
- **congruence bias**: The tendency to test hypotheses exclusively through direct testing, instead of testing possible alternative hypotheses
- conservatism (belief revision)
- curse of knowledge
- default effect
- expectation bias
- overconfidence effect
- plan continuation bias
- pro innovation bias
- recency illusion

[https://en.wikipedia.org/wiki/List\\_of\\_cognitive\\_biases](https://en.wikipedia.org/wiki/List_of_cognitive_biases)

# Static (+dynamic) information gathering

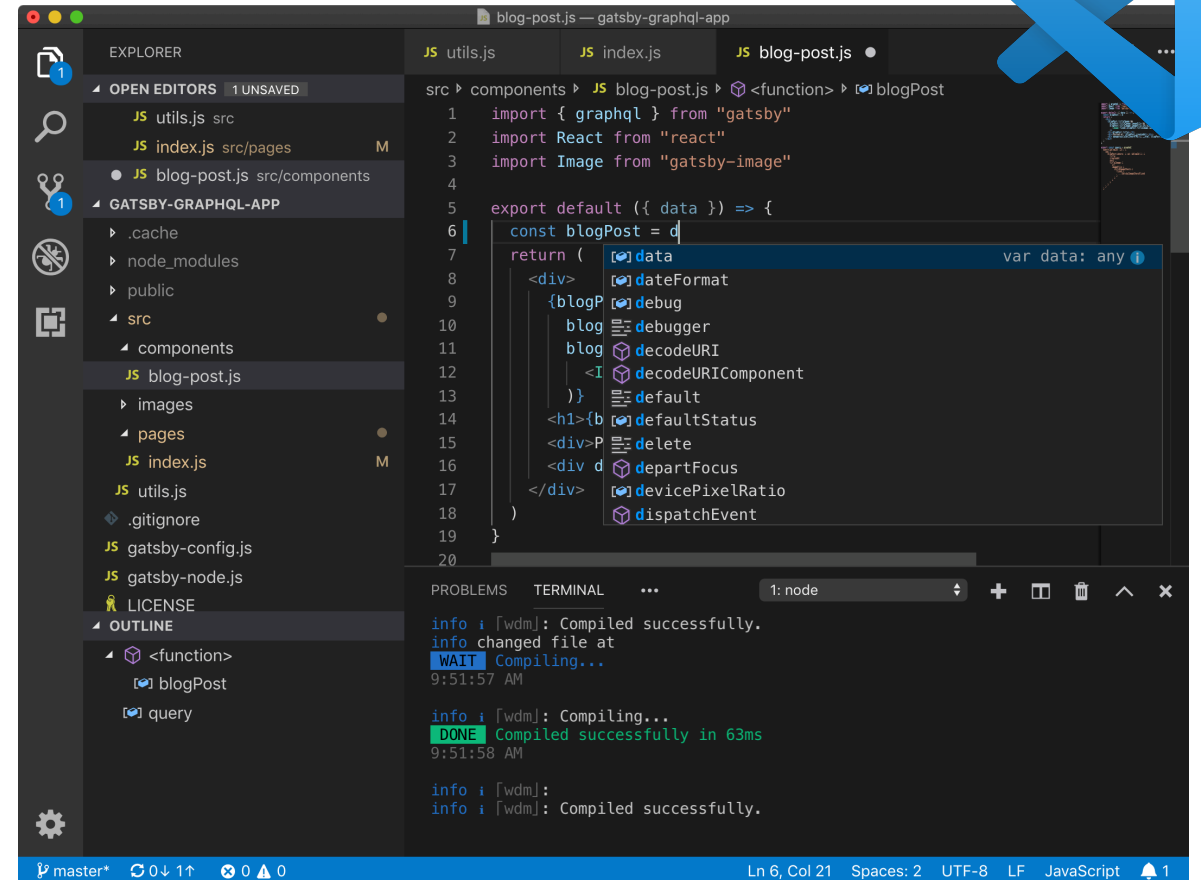
- Basic needs:
  - **Code/file search and navigation**
  - Code editing (probes)
  - Execution of code, tests
  - Observation of output (observation)
- Many choices here on tools! Depends on circumstance.
  - grep/find/command line/emacs
  - A decent IDE
  - Coverage computation
  - Testing tools
  - Debugger.
  - Etc.

At the command line: grep and find!  
I will find a tutorial and share it.



# Static information gathering: use tools to help manage complexity.

- Please configure and use a legitimate IDE.
  - Don't have a favorite? We like VSCode.
  - Configure: something like IntelliConfigure
- Why?
  - “search all files”
  - “jump to definition”
- Remember: real software is too complicated to keep in your head.



```
src > components > JS > blog-post.js > <function> > blogPost
1  import { graphql } from "gatsby"
2  import React from "react"
3  import Image from "gatsby-image"
4
5  export default ({ data }) => {
6  |  const blogPost = d
7      return (
8          <div>
9              {blogP
10                 blog
11                 blog
12                 <I
13                 )}
14             <h1>{b
15             <div>P
16             <div d
17         </div>
18     )
19 }
20
```

PROBLEMS TERMINAL ... 1: node

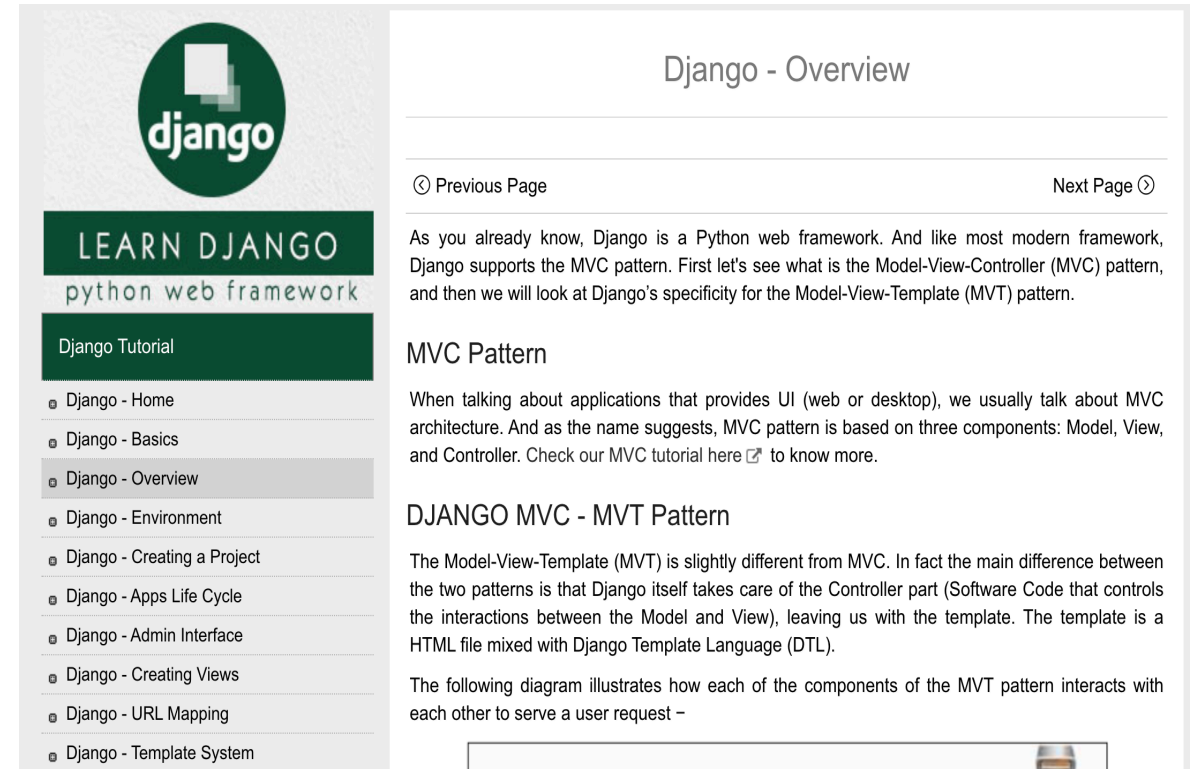
```
info i [wdm]: Compiled successfully.
info changed file at
WAIT Compiling...
9:51:57 AM

info i [wdm]: Compiling...
DONE Compiled successfully in 63ms
9:51:58 AM

info i [wdm]:
info i [wdm]: Compiled successfully.
```

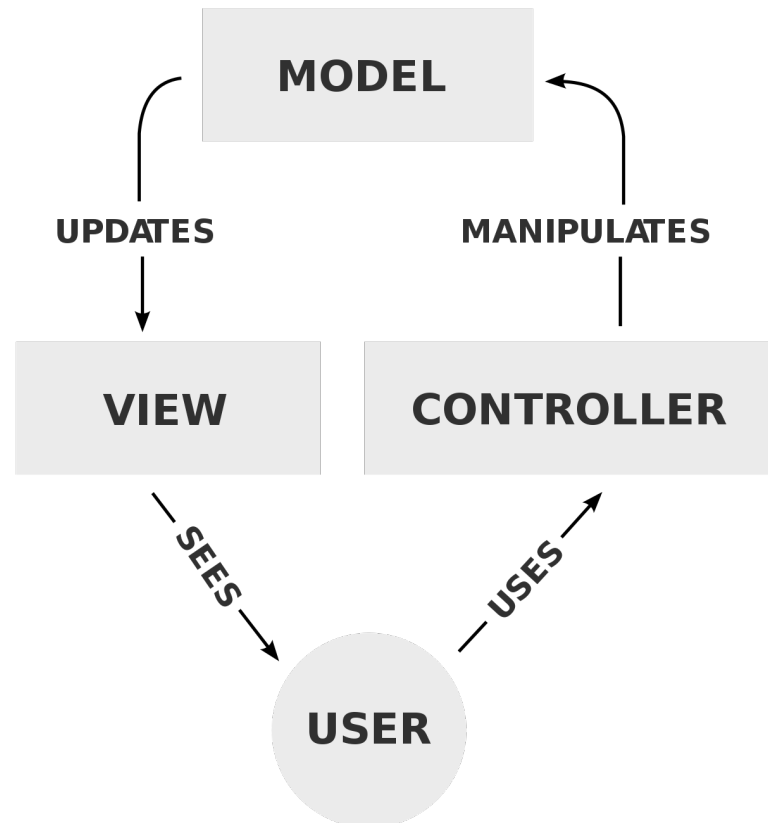
# Consider: Documentation and tutorials, judiciously

- Can teach you about general structure, architecture.
  - Forward-reference to architectural patterns!
- As you gain experience, you will recognize more of these, and you will immediately know something about how the program works.
- For example, next time you work on a webapp...



The screenshot shows the Django documentation website. On the left is a navigation menu with the Django logo at the top, followed by 'LEARN DJANGO python web framework' and 'Django Tutorial'. The 'Django Tutorial' section contains a list of links: 'Django - Home', 'Django - Basics', 'Django - Overview' (highlighted), 'Django - Environment', 'Django - Creating a Project', 'Django - Apps Life Cycle', 'Django - Admin Interface', 'Django - Creating Views', 'Django - URL Mapping', and 'Django - Template System'. The main content area is titled 'Django - Overview' and includes 'Previous Page' and 'Next Page' links. The text explains that Django is a Python web framework that supports the MVC pattern and introduces the Model-View-Template (MVT) pattern. It also mentions that Django handles the Controller part, leaving the user with a template (HTML mixed with Django Template Language). A diagram illustrating the MVT pattern interaction is partially visible at the bottom.

# Consider: Documentation and tutorials, judiciously



The screenshot shows a web page titled 'Django - Overview'. It features the Django logo and the text 'LEARN DJANGO python web framework'. Below this is a 'Django Tutorial' section with a list of links: Django - Home, Django - Basics, Django - Overview (highlighted), Django - Environment, Django - Creating a Project, Django - Apps Life Cycle, Django - Admin Interface, Django - Creating Views, Django - URL Mapping, and Django - Template System. The main content area contains the text: 'As you already know, Django is a Python web framework. And like most modern framework, Django supports the MVC pattern. First let's see what is the Model-View-Controller (MVC) pattern, and then we will look at Django's specificity for the Model-View-Template (MVT) pattern.' Below this is a section titled 'MVC Pattern' with the text: 'When talking about applications that provides UI (web or desktop), we usually talk about MVC architecture. And as the name suggests, MVC pattern is based on three components: Model, View, and Controller. Check our MVC tutorial here to know more.' A section titled 'DJANGO MVC - MVT Pattern' follows, with the text: 'The Model-View-Template (MVT) is slightly different from MVC. In fact the main difference between the two patterns is that Django itself takes care of the Controller part (Software Code that controls the interactions between the Model and View), leaving us with the template. The template is a HTML file mixed with Django Template Language (DTL). The following diagram illustrates how each of the components of the MVT pattern interacts with each other to serve a user request -'.

# Dynamic Information Gathering: High-level principles

- Key principle 1: change is a useful primitive to inform mental models about a software system.
- Key principle 2: systems almost always provide some kind of starting point.
- Put simply:
  1. Build it.
  2. Run it.
  3. Change it.
  4. Run it again.
- Can provide information both *bottom up* or *top down*, depending on the situation.

# Step 0: sanity check basic model + hypotheses.

- Confirm that you can build and run the code.
  - Ideally *both* using the tests provided, *and* by hand.
- Confirm that the code you are running *is the code you built*.
- Confirm that you can make *an externally visible change*.
  
- How? Where? Starting points:
  - Run an existing test, change it.
  - Write a new test.
  - Change the code, write or rerun a test that should notice the change.

# Demonstration: Live Coding By Chris

