Kojo Idrissa @Transition

A Jr. Developer's Guide to Software Engineering October 2018

- Now: Jr. Developer, Pythonista/Djangonaut
- Then: Accountant & uni instructor w/ an MBA

Kojo Idrissa @Transition

Who Else Am I?

- DjangoCon US Organizer
- DEFNA North American Ambassador
- @DEFNADO for more info

DjangoCon US is starts a week from Sunday in San Diego! It could also come to your city in 2020!

Define DEFNA has grants for Django-related events in your area.

But what's my talk about?

Kojo Idrissa @Transition

Coding?

Why?

How?

This talk is focused on people learning to code. But there are different reasons to learn to code. I’m going to focus on one of the two motivations I’ve identified.

Kojo Idrissa @Transition

Two Styles of Coding

Programming	Software Engineering
• Individuals	• Teams
• Author	• Others
• Less robust	• More robust

Programming:
• often done solo
• author may be the only user of the code
Software Engineering
• Higher level of robustness required; the user isn't intended to FIX things that break. This is true with open source projects as well. The software is expected to WORK.
This talk is about Software Engineering

Interjection(!): “Real” coders

- If your code runs & does what you want it to do, you’re a “real” coder.
- Choose your style

The prior slide is not about making a distinction b/w “real” programmers and “not real” programmers. It’s meant to draw attention to two different styles of building software. Self-taught career changers often focus on the FIRST style, while trying to get hired or make FLOSS contributions. But commercial and FLOSS software are most often built using the SECOND style. I’ll focus on fundamentals you need to know to build software in THAT style for the rest of the talk.

Lone Genius Myth is Myth

The Issue

- Career changers seek internships or "Jr. Developer" jobs
- Jr. Developer == ???
- “When am I ready?”

Kojo Idrissa @Transition

The Issue

MOAR Programming!

- With no solid definition for “Jr. Developer”, new developers focus PRIMARILY on the language or framework. They try to become “better developers” by learning more of that, overlooking the other key tools.
- You don’t become an author or journalist by memorizing a dictionary
- “How much Python do I need to know to get a job?” isn’t the best question.

Kojo Idrissa @Transition

Secondary Issue

How do you mentor a new developer?

This is important for people who’re hiring new developers. Trying to throw the most money at experienced developers is a losing game.

Kojo Idrissa @Transition

What I'll Talk About

- Version Control
- Documentation
- Testing
- Dependency Management/Deployment
- Development Environment

Tell & Show (to avoid jumping back and forth)

Version Control: Why?

- Protect yourself from mistakes
- Enable yourself to try new things
- Learn to work on a team

I’m focusing mostly on Git here.

Version Control: When?

- More code than you want to retype
- Working code you're scared to break

- 10 lines is my arbitrary, “I don’t want to retype all that!” threshold
- If you’re scared to break your code by changing it, AFTER you start using Git, that’s the PERFECT time to learn to use feature branches.

Version Control: How?

- Solo usage
- Basic feature branch/merge workflow
- Git != Github
- Ask me about a practical example

- Learn to use Git by yourself, on your own projects. It helps you develop the logical, “atomic” commits approach to your code. THAT makes it easier to “go back” if you break something.
- git add -p can be VERY helpful
- Learning to branch/merge on your own makes working with a team much easier.
- It also makes code review from a more experienced developer easier.
- You can use Git alone or Github alone before you work with a team

Version Control: How?

- Reference
  - Pro Git Chapters 1-3
  - Free!

80% of what you'll need to know

Kojo Idrissa @Transition

Documentation

- Refine your thinking
- Help your future self
- Help your teammates help you

- In Code vs. “how to run this code”
- Think about what you want to code BEFORE you write the code
- You’re GOING to forget why you wrote that code from 3 weeks/months/years ago
- Let you co-workers to RTD!

Kojo Idrissa @Transition

(!) Self-Documenting Code

- Kinda, not really
- Naming is hard
- Start to think about it now

You may hear “Good code is self-documenting”. There’s a KERNEL of truth there, but that’s based on

1. Your variables, functions, classes, etc having EXCELLENT names that tell you exactly what they do, and maybe even HOW they do it. eg. average or weighted_average or monte_carlo_simulation
2. the reader understanding the naming conventions used

That’s hard. But it gets easier with experience and practice.

Kojo Idrissa @Transition

Documentation: When?

- Writing code using a new technique
- Writing functions, especially if they call other functions

- Remind yourself of your design choices and decisions. Why'd you use a tuple instead of a list? This is where in-code comments shine.
- Functions are one of the first places your code becomes "less obvious".
- Docstrings in functions are used by the builtin help()
- Help help() help you!

Documentation: How

- Docstrings for functions (minimums)
  - What the function does
  - What's the input? (include data type)
  - What's the output? (include data type)

In ADDITION to inline comments:
Slide is notes
functions in a class are methods

Kojo Idrissa @Transition

Testing & TDD

So.

Many.

Reasons.

Kojo Idrissa @Transition

Why Testing: The Big Picture

- Tests ->Automated tests
- Automated Tests ->CI
- CI -> CD (Continuous Delivery)
- CD -> CD (Continuous Deployment)

Testing is a HUGE topic but here's why you should start writing and running tests ASAP.
- Automated tests: you don't need a person to run them, they can be run by a machine
- Branching and merging == integration. But you want to make sure the new merges don't BREAK anything. So, you run the tests. But if they're automated, you can Continuously Integrate. So, each time new code is merged in or added, the tests can run
- This usually happens on a CI server. Travis CI, Jenkins, Circle CI
- CI, you can move to CDelivery: your software is READY to be deployed each time new code is added and tests pass

Why should YOU write tests?

- Protection from future changes breaking things
- Assurance past fixes still work

LOTS of different types of tests. Here I'm focusing on two types, as I think these are the best starting points for Jr. Developers
- Unit tests: test a small piece of code, to make sure it does what you expect
- Regression tests: when you FIND and FIX a bug, write a test to make sure it stays fixed

Testing: When?

- As soon as possible
- Art + Science: experience helps

Testing: How?

- doctest & unittest: builtin
- pytest: *`pip install pytest`*

- doctests and unittest are built in to Python, so there's nothing to install
- pytest is VERY quickly becoming the first choice for running tests. The good news is, any tests you write using doctests or unittest can also be run with pytest.

Dependency Mgmt & Deployment

- Virtual Env*(ironments)*
- Containers
- Virtual Machines (VMs)

Two separate but related issues

- Making sure you have the correct “parts” you need for your software to run
    - Beyond libraries, this includes the correct versions
    - Helps you keep things updated (my die roller is SO old, Github hates it…)
- Getting your software to run outside your laptop

Other

- Development Environment
- A shell language
- Terminal multiplexer

- CLI vs. GUI: can you navigate?
- Bash, Xonsh, etc
- screen, tmux, byobu: especially for web devs

Kojo Idrissa @Transition

Finding New Developers

To:

- Engineering Managers
- Dev Leads
- Others trying to hire developers…

- If you're trying to hire "Sr. Developers", ESPECIALLY if you're a a startup, guess what? So's EVERYONE ELSE
- You don’t have MS/Google/Facebook/Apple/Rackspace money or appeal (find Patio’s term)
- Simple math: Number of People with LESS than 3 years of experience > than people WITH 3+ years of experience
- That’s a different talk

Questions/Comments?

- @Transition on Twitter
- kojoidrissa.com
- https://github.com/kojoidrissa/pygotham_2018