NINA ZAKHARENKO
CODE REVIEW
SKILLS FOR PYTHONISTAS
@NNJA
bit.ly/codereviewpy
LIVETWEET
USE #EUROPYTHON @NNJA
WHAT WE'LL LEARN TODAY [1/2]

> What are the proven benefits of review?
> Setting standards
> Time saving tools and automation options for Python
> How to review code helpfully
> How to submit PRs for maximum impact
> Use code review to build a stronger team
WHAT WILL YOU TAKE AWAY FROM THIS TALK?

> **Novice** - Comprehensive overview of code review best practices

> **Intermediate** - Tooling & automation

> **Advanced** - Hardest part – the people factor
NOT ONE SIZE FITS ALL!

> team size
  > 2 vs 10
> product type
  > agency vs in-house vs open source
WHY CODE REVIEW?
CODE REVIEWS CAN BE FRUSTRATING
APPARENT CODE REVIEW FRUSTRATIONS

> Adds time demand
> Adds process
> Can bring up team tensions
> “smart” devs think they don’t need it 😳
CODE REVIEW BENEFITS
FIND BUGS & DESIGN FLAWS

> Design flaws & bugs can be identified and remedied before the code is complete

> Case Studies on Review\(^5\):

> ↓ bug rate by 80%

> ↑ productivity by 15%

\(^5\) blog.codinghorror.com/code-reviews-just-do-it/
The goal is to find bugs before your customers do.
SHARED OWNERSHIP & KNOWLEDGE

> We’re in this together
> No developer is the only expert
LOTTERY FACTOR

New Yorker: Can Andy Byford Save the Subways?

When the NYC subway vending machines go down, there's apparently only one guy who knows how to fix them.

His name is Miguel, he lives in Port Jarvis (3 hrs from NYC), & apparently he likes to turn his cell phone off on the way home.

Via William Finnegan
newyorker.com/magazine/2018/ ...
CODE REVIEW BENEFITS?

> Find Bugs
> Shared Ownership
> Shared Knowledge
> Reduce "Lottery Factor"
HOW?
CONSISTENT CODE

> Your code isn’t yours, it belongs to your company

> Code should fit your company’s expectations and style (not your own)

> Reviews should encourage consistency for code longevity
CODE REVIEWS NEED TO BE UNIVERSAL & FOLLOW GUIDELINES

> Doesn’t matter how senior / junior you are
> Only senior devs reviewing == bottleneck
> Inequality breeds dissatisfaction
STYLE GUIDE

> Distinguishes personal taste from opinion
> Should be agreed upon beforehand
> Go beyond PEP8
> See: Google's pyguide.md or plone styleguide
FORMATTERS

- autopep8
- Black
- YAPF
thumbs up 🍌 BLACK

DEMO @: black.now.sh

```python
- if very_long_variable_name is not None and \
  - very_long_variable_name.field > 0 or \
  - very_long_variable_name.is_debug:
- z = 'hello ' + 'world'
+ if (  
  + very_long_variable_name is not None  
  + and very_long_variable_name.field > 0  
  + or very_long_variable_name.is_debug  
  +):
+   z = "hello " + "world"
```

github.com/jpadilla/black-online
settings:
  "editor.formatOnSave": true
  "python.formatting.provider": "black"
  "python.formatting.blackArgs": [
    "--line-length", "100"
  ]
> VS Code Black support docs

@nnja
CONSISTENT CODE IS EASIER TO MAINTAIN BY A TEAM
Code review is done by your peers & not management
DON'T POINT FINGERS!
WHEN CODE REVIEWS ARE POSITIVE, DEVELOPERS DON'T EXPECT THEIR CHANGES TO BE REVIEWED, THEY WANT THEIR CHANGES TO BE REVIEWED.
LET'S REVIEW: CODE REVIEW
FUNDAMENTALS

> Universal code review
> Performed by Peers
> Style guides & formatters for consistency
> No blame culture
HOW SHOULD WE CODE REVIEW?
BE A GREAT SUBMITTER
No need to double check this change list, if some problems remain the reviewer will catch them.

No need to look at this change list too closely, I’m sure the author knows what he’s doing.
DON'T GET RUBBER-STAMPED.
DON'T BE CLEVER.
READABILITY COUNTS!
GOOD CODE IS LIKE A GOOD JOKE. IT NEEDS NO EXPLANATION. 
- RUSS OLSEN
STAGES OF REVIEW

> 0: before PR submission
> 1: PR submitted
> 2: (optional) work in progress PR (30-50%)
> 3: review almost done (90-100%)
> 4: review completed
STAGE 0:
BEFORE SUBMISSION
@nnja
PROVIDE CONTEXT (THE WHY)

> What was the motivation for submitting this code?
> Link to the underlying ticket/issue
> Document why the change was needed
> For larger PRs, provide a changelog
> Point out any side-effects
YOU ARE THE PRIMARY REVIEWER

> Review your code with the same level of detail you would give giving reviews.
> Anticipate problem areas.
THE PRIMARY REVIEWER

> Make sure your code works, and is thoroughly tested.
> Don’t rely on others to catch your mistakes.
BEFORE SUBMITTING, TRY A CHECKLIST
SMALL STUFF

> Did you check for reusable code or utility methods?
> Did I remove debugger statements?
> Are there clear commit messages?
> Is my code secure?
> Will it scale?
> Is it maintainable?
> Is it resilient against outages?

Tip: Read The Checklist Manifesto
STAGE 1:
SUBMITTED
YOU'RE STARTING A CONVERSATION

> Don’t get too attached to your code before the review process
> Anticipate comments and feedback
> Acknowledge you will make mistakes
Submit Work In Progress Pull Requests

Open them your code is 30-50% done

Good idea for bigger features

Don’t be afraid of showing incomplete, incremental work
WHEN CODE IS WORK IN PROGRESS

> Feedback to expect:
> Architectural issues
> Problems with overall design
> Design pattern suggestions
STAGE 3:
ALMOST DONE

@nnja
WHEN CODE IS ALMOST DONE

> Feedback to expect:
  > Nit Picks
  > Variable Names
  > Documentation & Comments
  > Small Optimizations
ONE REVIEW PER PR

> Solving multiple problems? Break them up into multiple PRs for ease of review.

> Solved an unrelated problem? Make a new PR with a separate diff
PREVENT REVIEWER BURNOUT

> Reviews lose value when they’re more than 500 lines of code

> Keep them small & relevant

> If a big PR is unavoidable, give the reviewer extra time


@nnja
CHECK CODE WITH AUTOMATED TOOLS
✔ LINTER

def product(x, y):
    return x * y

y = product(a, b)

flake8 F821 undefined name 'a'
flake8 F821 undefined name 'b'
Pylint

Star your Python code!

> Coding standard
> Error detection
> Refactoring help
> IDE & editor integration

@nnja
PYLINT RULE: trailing-comma-tuple

foo = (1,
      3,
      4,
    )

bar = 2,
USE VULTURE.PY TO FIND DEAD OR UNREACHABLE CODE

$ pip install vulture
$ vulture script.py package/

or

$ python -m vulture script.py package/

github.com/jendrikseipp/vulture
Sample code

def foo():
    print("foo")

def bar():
    print("bar")

def baz():
    print("baz")

foo()
bar()
GIT PRE-COMMIT HOOKS

> run linter
> check syntax
> check for TODOs, debugger statements, unused imports
> enforce styling (autopep8, black formatter, sort imports, etc)
> option to reject commit if conditions don't pass
pre-commit

A framework for managing and maintaining multi-language pre-commit hooks.
pre-commit.com

> autouppep8-wraper - Runs autouppep8 over source
> flake8 and pyflakes - Run flake8 or pyflakes on source
> check-ast - Check whether files contain valid python
> debug-statements - Check for debugger imports and breakpoint() calls
Tests

> Write them!
> Don’t know code health if tests are failing
> Tests identify problems immediately
SCUMBAG PROGRAMMER

COMMENTS UNTESTESTED CODE
CONTINUOUS INTEGRATION

CPYTHON USES VSTS

All checks have passed
9 successful checks

VSTS: Linux-PR — Linux-PR_20180725.27 succeeded
VSTS: Linux-PR-Coverage — Linux-PR-Coverage_20180725.32 succeeded
VSTS: Windows-PR — Windows-PR_20180725.27 succeeded
VSTS: docs — docs_20180725.35 succeeded
VSTS: macOS-PR — macOS-PR_20180725.27 succeeded

This branch has no conflicts with the base branch
Only those with write access to this repository can merge pull requests.
Coverage Report: 87%

<table>
<thead>
<tr>
<th>Module</th>
<th>statements</th>
<th>missing</th>
<th>excluded</th>
<th>coverage</th>
</tr>
</thead>
<tbody>
<tr>
<td>mymath.py</td>
<td>9</td>
<td>3</td>
<td>0</td>
<td>67%</td>
</tr>
<tr>
<td>test_mymath.py</td>
<td>14</td>
<td>0</td>
<td>0</td>
<td>100%</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>23</strong></td>
<td><strong>3</strong></td>
<td><strong>0</strong></td>
<td><strong>87%</strong></td>
</tr>
</tbody>
</table>

Coverage.py v4.1, created at 2016-07-18 15:04
Coverage tools integrate into GitHub

- coverage.py
- coveralls.io
STAGE 4:
REVIEW COMPLETE

@nnja
BE RESPONSIVE

> Reply to every comment
> Common Responses:
  > Resolved
  > Won’t Fix
  > If you won’t fix, make sure you’ve come to a mutual understanding with the reviewer

@nnja
If there were comments, let your reviewer know when you’ve pushed changes and are ready to be re-reviewed.
DON'T BIKE-SHED

> bikeshed.com
> back & forth > 3 times? step away from the keyboard
> talk instead!
> record the results of the conversation in the PR
VS CODE LIVE SHARE

Real-time sharing in tools you love.

Share the full context of your code.

Collaboratively edit. Navigate independently.

Collaboratively debug. Inspect on your own.

Download Extension
FIGHT FOR WHAT YOU BELIEVE, BUT GRACEFULLY ACCEPT DEFEAT.

IT'S OK
I UNDERSTAND

@nnja
DON'T TAKE FEEDBACK PERSONALLY. IT'S AN OPPORTUNITY FOR GROWTH.
HOW TO BE A GREAT SUBMITTER?

> Provide the why (context!)
> Review your own code
> Expect conversation
> Submit in progress work
HOW TO BE A GREAT SUBMITTER?

> Use automated tools
> Be responsive
> Accept defeat
#1: BE A GREAT REVIEWER
Why do you think you’re so hostile in code reviews?

If only I had been more popular in High School.

Have Empathy
BE OBJECTIVE

"THIS METHOD IS MISSING A DOCSTRING" INSTEAD OF "YOU FORGOT TO WRITE A DOCSTRING"
ASK QUESTIONS DON'T GIVE ANSWERS

> “Would it make more sense if... ?”
> “Did you think about... ? ”
OFFER SUGGESTIONS

> “It might be easier to...”
> “We tend to do it this way...”
AVOID THESE TERMS

> Simply
> Easily
> Just
> Obviously
> Well, actually...
... NOW, SIMPLY
HAVE CLEAR FEEDBACK

> Strongly support your opinions
> Share How & Why
> Link to supporting documentation, blog post, or stackoverflow examples

@nnja
THIS IS NOT CLEAR FEEDBACK

THAT LINE OF CODE GIVES ME GAS

@nnja
COMPLIMENT GOOD WORK AND GREAT IDEAS

nnja commented 2 minutes ago

👍 *100
The goal is better code, not "exactly the code you would have written"
DON'T BE A PERFECTIONIST

> For big issues, don’t let perfect get in the way of perfectly acceptable.

> Prioritize what’s important to you.

> Usually 90% there is good enough.
IT'S OK TO NIT-PICK

- Syntax Issues
- Spelling Errors
- Poor Variable Names
- Missing corner-cases
- Specify: Are your nitpicks blocking merge?

Save the nit-picks for last, after any pressing architecture, design, or other large scale issues have been addressed.

@nnja
Don't burn out. Studies show reviewer should look at 200-400 lines of code at a time for maximum impact\textsuperscript{2}.

\textsuperscript{2} https://smartbear.com/learn/code-review/best-practices-for-peer-code-review/
Limit reviews to 400 lines in 60 mins to maximize effectiveness³.

³ https://smartbear.com/learn/code-review/best-practices-for-peer-code-review/
TRY TO DO REVIEWS IN 24-48 HOURS

STILL WAITING
FOR A CODE REVIEW
HOW CAN WE BE A GREAT REVIEWER?

> Have Empathy
> Watch your Language
> Have Clear Feedback
> Give Compliments
HOW CAN WE BE A GREAT REVIEWER?

> Don’t be a perfectionist
> Avoid Burn Out
> Complete in 24-48 hours
CODE REVIEWS
BUILD A STRONGER TEAM

@nnja
FIRST DAY VIBES...

I HAVE NO IDEA WHAT I'M DOING

@nnja
NEWBIES

> Not everyone has experience being reviewed.
> Remember what it felt like when you introduced the process.
> Ease into it!

@nnja
ONBOARDING

> The first submitted PR is the hardest
> The first review done is challenging too
> Start by reading recently completed reviews
> First code review should be small
> Share the style guide
EVERYONE'S A REVIEWER

> Junior devs start by doing pair-reviews with a more experienced teammate.
> Use it as a mentorship opportunity.
Hiring senior engineers is hard. You can hire junior engineers, and grow them into functional productive parts of your team.  
- Sasha Laundy
IF YOU'RE NOT DOING CODE REVIEWS, YOU'RE MISSING A BIG OPPORTUNITY.
REMEMBER...

> Allocate the time
> Develop, don’t force the process
> Not one size fits all
> Or a one stop fix

> Use in addition to tests, QA, etc for maximum impact

@nnja
I've become a much better programmer by participating in code reviews.
WHAT DID WE LEARN?
Coworkers who are good at code review are worth their weight in gold.
REVIEWS DECREASE WTFs/M BY INCREASING CODE QUALITY LONG TERM

http://commandot.com
LESS WTFs ➡ HAPPIER DEVS!
THANKS!

SLIDES: bit.ly/codereviewpy aka.ms/python @nnja

(Additional resources on next slides)
RESOURCES & ADDITIONAL READING

> Microsoft Study on Effective Code Review
> Code Reviews: Just do it
> Code Project - Code Review Guidelines
> Great Example Checklist
> Best Practices for Code Review
> Rebecca's Rules for Constructive Code Review
> My Big Fat Scary Pull Request
> The Gentle Art of Patch Review - Sage Sharp
> Watch: Raymond Hettinger - Beyond PEP8
EXAMPLE STYLE GUIDES

> Python
> Plone

Google has many good, but strict style guides at: https://github.com/google/styleguide

Doesn't matter which one you use. Pick one and stick with it.

@nnja