Hello!

TODO: add comments

5 Tips for *Winning* at Code Comments

(according to Nik Kantar)

Nik Kantar

- I make software, usually with Python.
- web(log): nkantar.com
- code: @nkantar (GitHub)
- toots: @nkantar (Twitter)
- email: nik@nkantar.com (plz no recruiters)
- slides: nkantar.com/talks

This talk...

- ...is opinionated,
- aims to be helpful,
- and is *hilarious*,
- but could probably use some work.

  (Read: feedback welcome!)

Wise quote

"Bad documentation is worse than no documentation."
— some wise people (and also me)

The basics

- Q: What are code comments?
  - A: `# this stuff`
- Q: Why are they important?
  - A: Code is written for humans.
- Q: Who are they for?
  - A: Me, you 6 months later, everyone else.

Another wise quote

"Programs must be written for people to read, and only incidentally for machines to execute."
— Harold Abelson, *Structure and Interpretation of Computer Programs* (1979)

The 5 tips

1. Make comments stand out in your editor.
2. Explain the *why*, not the *what*.
3. Don't fear the paragraph.
4. Read what you wrote out loud.
5. Ask for help!

1. Make comments stand out.

```
def prepare_body(self, data, files, json=None):
    """Prepares the given HTTP body data."""

    # Check if file, fo, generator, iterator.
    # If not, run through normal process.

    # Nottin' on you.
    body = None
    content_type = None

    if not data and json is not None:
        # urllib3 requires a bytes-like body. Python 2's json.dumps
        # provides this natively, but Python 3 gives a Unicode string.
        content_type = 'application/json'
        body = complexjson.dumps(json)
        if not isinstance(body, bytes):
            body = body.encode('utf-8')
```

```
def prepare_body(self, data, files, json=None):
    """Prepares the given HTTP body data."""

    # Check if file, fo, generator, iterator.
    # If not, run through normal process.

    # Nottin' on you.
    body = None
    content_type = None

    if not data and json is not None:
        # urllib3 requires a bytes-like body. Python 2's json.dumps
        # provides this natively, but Python 3 gives a Unicode string.
        content_type = 'application/json'
        body = complexjson.dumps(json)
        if not isinstance(body, bytes):
            body = body.encode('utf-8')
```

```
def prepare_body(self, data, files, json=None):
    """Prepares the given HTTP body data."""

    # Check if file, fo, generator, iterator.
    # If not, run through normal process.

    # Nottin' on you.
    body = None
    content_type = None

    if not data and json is not None:
        # urllib3 requires a bytes-like body. Python 2's json.dumps
        # provides this natively, but Python 3 gives a Unicode string.
        content_type = 'application/json'
        body = complexjson.dumps(json)
        if not isinstance(body, bytes):
            body = body.encode('utf-8')
```

```python
def prepare_body(self, data, files, json=None):
    """Prepares the given HTTP body data."""

    # Check if file, fo, generator, iterator.
    # If not, run through normal process.

    # Nottin' on you.
    body = None
    content_type = None

    if not data and json is not None:
        # urllib3 requires a bytes-like body. Python 2's json.dumps
        # provides this natively, but Python 3 gives a Unicode string.
        content_type = 'application/json'
        body = complexjson.dumps(json)
        if not isinstance(body, bytes):
            body = body.encode('utf-8')
```

```
def prepare_body(self, data, files, json=None):
    """Prepares the given HTTP body data."""

    # Check if file, fo, generator, iterator.
    # If not, run through normal process.

    # Nottin' on you.
    body = None
    content_type = None

    if not data and json is not None:
        # urllib3 requires a bytes-like body. Python 2's json.dumps
        # provides this natively, but Python 3 gives a Unicode string.
        content_type = 'application/json'
        body = complexjson.dumps(json)
        if not isinstance(body, bytes):
            body = body.encode('utf-8')
```

```
def prepare_body(self, data, files, json=None):
    """Prepares the given HTTP body data."""

    # Check if file, fo, generator, iterator.
    # If not, run through normal process.

    # Nottin' on you.
    body = None
    content_type = None

    if not data and json is not None:
        # urllib3 requires a bytes-like body. Python 2's json.dumps
        # provides this natively, but Python 3 gives a Unicode string.
        content_type = 'application/json'
        body = complexjson.dumps(json)
        if not isinstance(body, bytes):
            body = body.encode('utf-8')
```

```python
def prepare_body(self, data, files, json=None):
    """Prepares the given HTTP body data."""

    # Check if file, fo, generator, iterator.
    # If not, run through normal process.

    # Nottin' on you.
    body = None
    content_type = None

    if not data and json is not None:
        # urllib3 requires a bytes-like body. Python 2's json.dumps
        # provides this natively, but Python 3 gives a Unicode string.
        content_type = 'application/json'
        body = complexjson.dumps(json)
        if not isinstance(body, bytes):
            body = body.encode('utf-8')
```

Invisibility is bad.

- Invisible things are easy to ignore.
- Invisible comments get ignored.
- Ignored comments suffer in quality.
- Leftovers:
  - outdated comments
  - commented-out blocks of code

```
def prepare_body(self, data, files, json=None):
    """Prepares the given HTTP body data."""

    # Check if file, fo, generator, iterator.
    # If not, run through normal process.

    # Nottin' on you.
    body = None
    content_type = None

    if not data and json is not None:
        # urllib3 requires a bytes-like body. Python 2's json.dumps
        # provides this natively, but Python 3 gives a Unicode string.
        content_type = 'application/json'
        body = complexjson.dumps(json)
        if not isinstance(body, bytes):
            body = body.encode('utf-8')
```

```
def prepare_body(self, data, files, json=None):
    """Prepares the given HTTP body data."""

    # Check if file, fo, generator, iterator.
    # If not, run through normal process.

    # Nottin' on you.
    body = None
    content_type = None

    if not data and json is not None:
        # urllib3 requires a bytes-like body. Python 2's json.dumps
        # provides this natively, but Python 3 gives a Unicode string.
        content_type = 'application/json'
        body = complexjson.dumps(json)
        if not isinstance(body, bytes):
            body = body.encode('utf-8')
```

```python
def prepare_body(self, data, files, json=None):
    """Prepares the given HTTP body data."""

    # Check if file, fo, generator, iterator.
    # If not, run through normal process.

    # Nottin' on you.
    body = None
    content_type = None

    if not data and json is not None:
        # urllib3 requires a bytes-like body. Python 2's json.dumps
        # provides this natively, but Python 3 gives a Unicode string.
        content_type = 'application/json'
        body = complexjson.dumps(json)
        if not isinstance(body, bytes):
            body = body.encode('utf-8')
```

Conclusion #1:
Make comments stand out.

2. *Why*, not *what*.

What vs why

- **Myth: "Code is self-documenting."**
- **Truth: "Code can self-document *what* happens."**
- **Comments: *Why* is this...**
  - **...here?**
  - **...done this way?**
  - **...done at all?!**

What-vs-why example

```
file_data = file_data.replace("\\\\\\\\", "\\\\")
```

What-vs-why example

```
# replace 8 backslashes with 4
file_data = file_data.replace("\\\\\\\\", "\\\\")
```

What-vs-why example

```
# wat? why. just why.
file_data = file_data.replace("\\\\\\\\", "\\\\")
```

What-vs-why example

```
# y tho
file_data = file_data.replace("\\\\\\\\", "\\\\")
```

What-vs-why example

```
# This file is submitted with backslashes escaped,
# the validator escapes them again, and the first
# round of processing does it *again*, so here we
# remove the last layer we don't actually want.
file_data = file_data.replace("\\\\\\\\", "\\\\")
```

What-vs-why example

```
# TODO: clean up this whole escaping mess
# This file is submitted with backslashes escaped,
# the validator escapes them again, and the first
# round of processing does it *again*, so here we
# remove the last layer we don't actually want.
file_data = file_data.replace("\\\\\\\\", "\\\\")
```

Conclusion #2:
Explain the *why*, not the *what*.

3. Don't fear the paragraph.

Brevity is a virtue, until it isn't.

- No bonus points for brevity at the expense of thoroughness.
- It's OK to have more comments than code.

  (Don't worry about all the extra bytes of text—it's fine.)

Conclusion #3:
Don't be unnecessarily brief.

4. Read what you wrote out loud.

Language matters

- Hearing your words will help you evaluate them.
- It helps with...
  - ...typos!
  - ...bad grammar!
  - ...just plain nonsense (especially applicable if you've been staring at the screen for too long)!

Conclusion #4:
Read your comments out loud.

5. Ask for help!

Multiple perspectives

- Comments are largely for others—why not ask for feedback?
- Experts can verify correctness and completeness.
- Newbies can verify clarity.
- Everyone can pinpoint obvious issues.
- Great writers have great editors.

Yet another wise quote

"Be a good editor. The Universe needs more good editors, God knows."

— Kurt Vonnegut, *Letters* (2014)

Conclusion #5:
Ask for help!

Quick recap

1. Make comments stand out in your editor.
2. Explain the *why*, not the *what*.
3. Don't fear the paragraph.
4. Read what you wrote out loud.
5. Ask for help!

Thank you!

- Hope you had a good time.
- Send ~~help~~ feedback.
- Slides: nkantar.com/talks
- Questions, comments, et cetera: hallway, Twitter, email
- Bonus: PyBeach 2020, Los Angeles, CA 😎 (pybeach.org)