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

  (Read: feedback welcome!)

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.

Advice disguised as a joke

Write your code as if the person inheriting it is an axe murderer who knows where you live.

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 get_field(self, field_name):
    """
    Return a field instance given the name of a forward or reverse field.
    """
    try:
        # In order to avoid premature loading of the relation tree
        # (expensive) we prefer checking if the field is a forward field.
        return self._forward_fields_map[field_name]
    except KeyError:
        # If the app registry is not ready, reverse fields are
        # unavailable, therefore we throw a FieldDoesNotExist exception.
        if not self.apps.models_ready:
            raise FieldDoesNotExist(
                "%s has no field named '%s'. The app cache isn't ready yet, "
                "so if this is an auto-created related field, it won't "
                "be available yet." % (self.object_name, field_name)
            )
```

```
def get_field(self, field_name):
    """
    Return a field instance given the name of a forward or reverse field.
    """
    try:
        # In order to avoid premature loading of the relation tree
        # (expensive) we prefer checking if the field is a forward field.
        return self._forward_fields_map[field_name]
    except KeyError:
        # If the app registry is not ready, reverse fields are
        # unavailable, therefore we throw a FieldDoesNotExist exception.
        if not self.apps.models_ready:
            raise FieldDoesNotExist(
                "%s has no field named '%s'. The app cache isn't ready yet, "
                "so if this is an auto-created related field, it won't "
                "be available yet." % (self.object_name, field_name)
            )
```

```
def get_field(self, field_name):
    """
    Return a field instance given the name of a forward or reverse field.
    """
    try:
        # In order to avoid premature loading of the relation tree
        # (expensive) we prefer checking if the field is a forward field.
        return self._forward_fields_map[field_name]
    except KeyError:
        # If the app registry is not ready, reverse fields are
        # unavailable, therefore we throw a FieldDoesNotExist exception.
        if not self.apps.models_ready:
            raise FieldDoesNotExist(
                "%s has no field named '%s'. The app cache isn't ready yet, "
                "so if this is an auto-created related field, it won't "
                "be available yet." % (self.object_name, field_name)
            )
```

```
def get_field(self, field_name):
    """
    Return a field instance given the name of a forward or reverse field.
    """
    try:
        # In order to avoid premature loading of the relation tree
        # (expensive) we prefer checking if the field is a forward field.
        return self._forward_fields_map[field_name]
    except KeyError:
        # If the app registry is not ready, reverse fields are
        # unavailable, therefore we throw a FieldDoesNotExist exception.
        if not self.apps.models_ready:
            raise FieldDoesNotExist(
                "%s has no field named '%s'. The app cache isn't ready yet, "
                "so if this is an auto-created related field, it won't "
                "be available yet." % (self.object_name, field_name)
            )
```

```
def get_field(self, field_name):
    """
    Return a field instance given the name of a forward or reverse field.
    """
    try:
        # In order to avoid premature loading of the relation tree
        # (expensive) we prefer checking if the field is a forward field.
        return self._forward_fields_map[field_name]
    except KeyError:
        # If the app registry is not ready, reverse fields are
        # unavailable, therefore we throw a FieldDoesNotExist exception.
        if not self.apps.models_ready:
            raise FieldDoesNotExist(
                "%s has no field named '%s'. The app cache isn't ready yet, "
                "so if this is an auto-created related field, it won't "
                "be available yet." % (self.object_name, field_name)
            )
```

```
def get_field(self, field_name):
    """
    Return a field instance given the name of a forward or reverse field.
    """
    try:
        # In order to avoid premature loading of the relation tree
        # (expensive) we prefer checking if the field is a forward field.
        return self._forward_fields_map[field_name]
    except KeyError:
        # If the app registry is not ready, reverse fields are
        # unavailable, therefore we throw a FieldDoesNotExist exception.
        if not self.apps.models_ready:
            raise FieldDoesNotExist(
                "%s has no field named '%s'. The app cache isn't ready yet, "
                "so if this is an auto-created related field, it won't "
                "be available yet." % (self.object_name, field_name)
            )
```

```
def get_field(self, field_name):
    """
    Return a field instance given the name of a forward or reverse field.
    """
    try:
        # In order to avoid premature loading of the relation tree
        # (expensive) we prefer checking if the field is a forward field.
        return self._forward_fields_map[field_name]
    except KeyError:
        # If the app registry is not ready, reverse fields are
        # unavailable, therefore we throw a FieldDoesNotExist exception.
        if not self.apps.models_ready:
            raise FieldDoesNotExist(
                "%s has no field named '%s'. The app cache isn't ready yet, "
                "so if this is an auto-created related field, it won't "
                "be available yet." % (self.object_name, field_name)
            )
```

Bad documentation is worse than no documentation.

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?!

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

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

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

```
# 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("\\\\\\\\", "\\\\")
```

```
# TODO: clean up this whole ecaping 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.

Advice disguised as a joke, repeated

Write your code as if the person inheriting it is an axe murderer who knows where you live.

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.

Conclusion #5:
Ask for help!

Thank you!

Send ~~help~~ feedback.

Questions?

Slides: nkantar.com/talks