Code Comments

What are they good for?

Clara Bennett, @csojinb

PyCon 2015

For Beginners

- Comment everything!
- Helps you learn to read and understand code

```
better_things = []  # initialize list of better things
for thing in things:  # loop over all the things
    better_thing = make_better(thing)  # make thing better
    better_things.append(better_thing)  # add to list of better things
```

- Helps you understand more advanced moves

```
# loop over all the things make them better, return in new list
better_things = [make_better(thing) for thing in things]
```

For everyone else

- If writing a code comment, stop to think about why
- Comments go stale much faster than code
- Most comments try to:
  - Explain what the code accomplishes
  - Explain what the code is doing
  - Explain why the code isn't doing something that the reader might reasonably expect

(Clarification)

- I'm talking about `# comments`, not `'''docstrings'''`
- Comments are written to others who read your code
- Docstrings are written so that people can use your code *without* reading it

Comments that explain what code *accomplishes* ...

```
def parse_puppy_info(line, shelter_code):

    # initialize puppy
    fields = ('Age', 'Breed', 'IsHousebroken', 'IsLeashTrained')
    puppy = DictReader(StringIO(line), fieldnames=fields).next()

    # add shelter code
    puppy['Shelter'] = shelter_code

    # clean up formatting: strip off trailing **
    # example 'Breed' value: "PEMBROKE WELSH CORGI **"
    breed = puppy['Breed']
    if breed.endswith('**'):
        breed = breed[:-2]
    puppy['Breed'] = breed.strip()

    return puppy
```

… are begging for an extract refactor

- Encapsulate behavior details with a function whose name communicates intent

```
def parse_puppy_info(line, shelter_code):
    puppy = extract_raw_puppy_info(line)
    puppy['Shelter'] = shelter_code
    puppy['Breed'] = remove_extraneous_chars(puppy['Breed'])

    return puppy
```

Comments that explain what code *does*

- A sign that your code is too obscure
- The comments are crutches to avoid refactoring -- just refactor it already!

```
# prevent overwrite if already exists
sale.save(False)
```

vs

```
sale.save(overwrite=False)
```

```python
def save_accounts(lines):
    header = lines.pop(0).split(',')
    old_accounts = Accounts.all()
    for line in lines:
        saved = False
        if line.startswith('id'):
            continue  # skip header
        account_dict = dict(zip(header, line.split(',')))
        for old_account in old_accounts:
            if old_account.id != account_dict['id']:
                continue

            # matching account
            account.balance = round(float(account_dict['balance']))
            account.account_holder = account_dict['owner']
            account.save()
            saved = True
            break  # stop looking for matching account

        # move to next line if saved
        if saved:
            continue

        # accounts that make it here are new
        account = Account(account_dict)
        account.save()
```

wha? didn't we already get the header?

difficult to tell which loop is being broken

where did this come from again?

needed help figuring out which accounts are still left

Comments that explain why the code *isn't* doing something

- Can be okay, e.g. for compatibility needs

```
def do_the_thing():
    # can't use try...except...finally because we're supporting python 2.4
    try:
        try:
            ask_zhu_li()
        except ZhuLiMissingError, e:
            act_deflated()
    finally:
        do_something_eccentric()
```

- Avoid covering for hacks elsewhere with comments

In conclusion:

Instead of code comments, consider refactoring