WRITING QUALITY CODE

IDEAS, TECHNIQUES AND TOOLS FOR IMPROVING THE QUALITY OF WRITTEN CODE

Radosław Jankiewicz / @radekj / radekJ
WHO AM I

- Programming in Python since 2007
- Web applications (mostly)
- STX NEXT
RANDOM FACT...
AGENDA

1. Definition of code quality
2. Why is it important
3. How to measure the quality of code
4. How to improve quality of written code
   - Good practices
   - Useful tools
   - Other hints
5. Q&A
HOW TO DEFINE HIGH QUALITY CODE?
CHARACTERISTICS OF SOFTWARE QUALITY

- External
- Internal
External characteristics of software quality

- Correctness
- Usability
- Efficiency
- Reliability
- Integrity
- Adaptability
- Accuracy
- Robustness
Internal characteristics software quality

- Maintainability
- Flexibility
- Portability
- Reusability
- Readability
- Testability
- Understandability
GOOD CODE FROM THE DEVELOPER'S POINT OF VIEW:
UNDERSTANDABLE
UNDERSTANDABLE

Time spent by a programmer

- Understanding code
- Modyfying existing code
- Writing new code

http://blog.codinghorror.com/when-understanding-means-rewriting
HOW IMPORTANT IS HIGH QUALITY OF CODE
POOR QUALITY CODE COSTS

Code quality vs time for feature
Dependency of code quality and time required for implementing a new feature
HOW TO MEASURE THE QUALITY OF CODE
The only valid measurement of code quality: WTFs/minute

Good code.

Bad code.
SOFTWARE QUALITY METRICS

- Cyclomatic Complexity
- Halstead complexity measures
- Maintainability Index
<table>
<thead>
<tr>
<th>Construct</th>
<th>Effect on CC</th>
<th>Reasoning</th>
</tr>
</thead>
<tbody>
<tr>
<td>if</td>
<td>+1</td>
<td>An <em>if</em> statement is a single decision.</td>
</tr>
<tr>
<td>else</td>
<td>+0</td>
<td>The <em>else</em> statement does not cause a new decision.</td>
</tr>
<tr>
<td>for</td>
<td>+1</td>
<td>There is a decision at the start of the loop.</td>
</tr>
<tr>
<td>Boolean Operator</td>
<td>+1</td>
<td>Every boolean operator (<em>and</em>, <em>or</em>) <em>adds a decision point</em>.</td>
</tr>
</tbody>
</table>

Full table: https://radon.readthedocs.org/en/latest/intro.html
def example(foo, bar, baz):
    if foo > bar:
        if foo > baz:
            return foo
        else:
            return baz
    elif foo == bar:
        return bar
    else:
        return baz

CC = 4
HALSTEAD METRICS

- \( \eta_1 \) = the number of distinct operators
- \( \eta_2 \) = the number of distinct operands
- \( N_1 \) = the total number of operators
- \( N_2 \) = the total number of operands
def example(foo, bar, baz):
    if foo > bar:
        if foo > baz:
            return foo
        else:
            return (baz / 3)
    elif foo == bar:
        return bar
    else:
        return baz

- $\eta_1 = 7$ (example, if, else, elif, (, ), >, ==, /, return)
- $\eta_2 = 4$ (foo, bar, baz, 3)
- $N_1 = 16$ (all operators)
- $N_2 = 14$ (all operands)
HALSTEAD METRICS

- Program vocabulary: $\eta = \eta_1 + \eta_2$
- Program length: $N = N_1 + N_2$
- Calculated program length:
  \[ \hat{N} = \eta_1 \log_2 \eta_1 + \eta_2 \log_2 \eta_2 \]
- Volume: $V = N \log_2 \eta$
- Difficulty: $D = \frac{\eta_1}{2} \cdot \frac{N_2}{\eta_2}$
- Effort: $E = D \cdot V$
- Time required to program: $T = \frac{E}{18}$ seconds
MAINTAINABILITY INDEX

\[ MI = 171 - 5.2 \ln V - 0.23G - 16.2 \ln L \]

- \( V \) is the Halstead Volume
- \( G \) is the total Cyclomatic Complexity
- \( L \) is the number of Source Lines of Code (SLOC)
RADON

CC number

$ radon cc ./url.py -s
 ./url.py
     M 287:4 URLMethodsMixin.resource_url - C (18)
     M 35:4 URLMethodsMixin._partial_application_url - C (17)
     M 85:4 URLMethodsMixin.route_url - C (16)
     C 31:0 URLMethodsMixin - B (7)
     M 539:4 URLMethodsMixin.static_url - A (5)
     F 753:0 static_url - A (3)

MI index

$ radon mi ./url*.py -s
 ./urldispatch.py - A (56.71)
 ./url.py - A (46.64)
## RADON - CC RESULTS

<table>
<thead>
<tr>
<th>CC score</th>
<th>Rank</th>
<th>Risk</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 - 5</td>
<td>A</td>
<td>low - simple block</td>
</tr>
<tr>
<td>6 - 10</td>
<td>B</td>
<td>low - well structured and stable block</td>
</tr>
<tr>
<td>11 - 20</td>
<td>C</td>
<td>moderate - slightly complex block</td>
</tr>
<tr>
<td>21 - 30</td>
<td>D</td>
<td>more than moderate - more complex block</td>
</tr>
<tr>
<td>31 - 40</td>
<td>E</td>
<td>high - complex block, alarming</td>
</tr>
<tr>
<td>41+</td>
<td>F</td>
<td>very high - error-prone, unstable block</td>
</tr>
</tbody>
</table>
# RADON - MI RESULTS

<table>
<thead>
<tr>
<th>MI score</th>
<th>Rank</th>
<th>Maintainability</th>
</tr>
</thead>
<tbody>
<tr>
<td>100 - 20</td>
<td>A</td>
<td>Very high</td>
</tr>
<tr>
<td>19 - 10</td>
<td>B</td>
<td>Medium</td>
</tr>
<tr>
<td>9 - 0</td>
<td>C</td>
<td>Extremely low</td>
</tr>
</tbody>
</table>
## WEB FRAMEWORKS - MI RESULTS

<table>
<thead>
<tr>
<th>Rank</th>
<th>Pyramid (187 files)</th>
<th>Flask (61 files)</th>
<th>Django (836 files)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>97.8%</td>
<td>100%</td>
<td>98.3%</td>
</tr>
<tr>
<td>B</td>
<td>1.6%</td>
<td>0%</td>
<td>0.3%</td>
</tr>
<tr>
<td>C</td>
<td>0.5%</td>
<td>0%</td>
<td>1.3%</td>
</tr>
</tbody>
</table>
PYLINT

- Static code analysis
- Coding Standard
- Error detection
- Refactoring help
- Fully customizable
- Editor/IDE integration
def example(foo, a, blah):
    qux = 123
    if foo > a:
        return foo
    else:
        return datetime.now()

************** Module a
C: 1, 0: Missing module docstring (missing-docstring)
C: 1, 0: Black listed name "foo" (blacklisted-name)
C: 1, 0: Invalid argument name "a" (invalid-name)
C: 1, 0: Missing function docstring (missing-docstring)
E: 6,15: Undefined variable 'datetime' (undefined-variable)
W: 1,20: Unused argument 'blah' (unused-argument)
W: 2, 4: Unused variable 'qux' (unused-variable)

Global evaluation
_______________
Your code has been rated at -8.33/10
MORE TOOLS

R. Ganczarek

*Code Quality in Python - tools and reasons*

Tomorrow, 16:45 (Barria 2 room)
HOW TO IMPROVE QUALITY OF WRITTEN CODE
PEER CODE REVIEW

- Decreases number of bugs
- Enforces writing neat code
- Speeds up learning
- Enhances the team culture
CODE REVIEW - USEFUL RULES

- All changesets get code reviewed
- Automate everything you can
- Everyone makes code reviews / everybody gets code reviewed
CODE REVIEW TOOLS

- Pull requests inline comments (Github / Bitbucket / ...)
- Gerrit
- Crucible
- Phabricator
- many more...
READABILITY COUNTS
CODING CONVENTION

Keep the code consistent with the project's convention.
Use automatic syntax/code style guide checking.
PEP-8 is a good option.
NAMING VARIABLES, CLASSES, METHODS...

“There are only two hard things in Computer Science: cache invalidation and naming things.”

Phil Karlton
Variable name for the maximum number of people in a car

At first - it should be descriptive

```python
x = 5  # bad
data = 5  # bad
max = 5  # very bad
```

... but not too long ...

```python
maximum_number_of_people_in_the_car = 123  # bad
```

abbreviations are acceptable

```python
num_seats = 5  # not that bad
total_seats = 5  # good
max_passengers = 5  # good
```
Avoid double negative boolean logic

```python
seat.is_not_occupied = True   # bad

seat.is_empty = True          # ok
```
DOCSTRINGS

- MUST be valid. Wrong docstring is worse than no docstring at all. Keep it up to date.
- Do not explain the implementation details
- Summarize the function's behaviour
- Document function's arguments, return value(s), side effects, exceptions raised, and restrictions on when it can be called (all if applicable)
COMMENTS

• MUST be valid. Wrong comment is worse than no comment at all
• Inline comments are unnecessary and in fact distracting if they state the obvious. Don't do this:

```plaintext
x = x + 1  # Increment x
```
KEEP YOUR TESTS CLEAN

“If you let the tests rot, then your code will rot too. Keep your tests clean.”

Rober C. Martin - "Clean Code"
SELF EFFORT

What else could you do to increase the quality of written code?
KNOW PYTHON IDIOMS

>>> x > 10 and x <= 20

More pythonic:

>>> 10 < x <= 20
More pythonic:

```python
>>> colors = ['blue', 'red', 'green', 'red', 'blue', 'red']

>>> [(x, colors.count(x)) for x in set(colors)]
[('blue', 2), ('green', 1), ('red', 3)]

>>> from collections import Counter
>>> Counter(colors)
Counter({'red': 3, 'blue': 2, 'green': 1})
```
KNOW PYTHON'S SYNTAX

EXPRESSIONS

```python
>>> from contextlib import contextmanager
>>> @contextmanager
...     def tag(name):
...         yield
...         print "</%s>" % name
...
>>> with tag("h1"):
...     print "foo"
...
<h1>foo</h1>
```
SELF EFFORT

Read valuable books
Read documentation
Practice a lot
You are given a text, which contains different English letters and punctuation symbols. You should find the most frequent letter in the text. The letter returned must be in lower case. While checking for the most wanted letter, casing does not matter, so for the purpose of your search, "A" == "a". Make sure you do not count punctuation symbols, digits and whitespaces, only letters.

If you have **two or more letters with the same frequency**, then return the letter which comes first in the Latin alphabet. For example -- "one" contains "o", "n", "e" only once for each, thus we choose "e".

**Input:** A text for analysis as a string (unicode for py2.7).

**Output:** The most frequent letter in lower case as a string.

**Example:**

1. `checkio("Hello World!") == "l"`
2. `checkio("How do you do?") == "o"
3. `checkio("One") == "e"
4. `checkio("Oops!") == "o"
5. `checkio("AAAAAa!!!") == "a"
6. `checkio("abe") == "a"`
import string
from collections import Counter

def checkio(text):
    delete_chars = string.punctuation + string.whitespace + string.digits
    trans = text.maketrans('', '', delete_chars)
    clean_text = text.translate(trans)
    counter = Counter(clean_text.lower())
    max_occurrences = max(counter.values())
    most_common_letters = [k for k, v in counter.items() if v == max_occurrences]
    return sorted(most_common_letters)[0]
<table>
<thead>
<tr>
<th>Rank</th>
<th>Username</th>
<th>Level</th>
<th>Language</th>
<th>Date</th>
<th>Version</th>
<th>Votes</th>
<th>Comments</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>bryukh</td>
<td>20</td>
<td>Python 3.3</td>
<td>Feb 14, 2014</td>
<td>Open</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>yeky</td>
<td>19</td>
<td>Python 3.3</td>
<td>Jan 04, 2015</td>
<td>Open</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>ForeverYoung</td>
<td>9</td>
<td>Python 3.3</td>
<td>Dec 18, 2013</td>
<td>Open</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Happiness</td>
<td>11</td>
<td>Python 2.7</td>
<td>Feb 13, 2015</td>
<td>Open</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>yama_k_1101</td>
<td>11</td>
<td>Python 3.3</td>
<td>Jan 02, 2015</td>
<td>Open</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>GoodOldBoy</td>
<td>10</td>
<td>Python 3.3</td>
<td>Nov 09, 2014</td>
<td>Open</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>panaro32</td>
<td>14</td>
<td>Python 3.3</td>
<td>Apr 30, 2014</td>
<td>Open</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>walkingpendulum</td>
<td>12</td>
<td>Python 2.7</td>
<td>Mar 25, 2015</td>
<td>Open</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>yukirin</td>
<td>14</td>
<td>Python 3.3</td>
<td>Mar 29, 2015</td>
<td>Open</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Boy howdy</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
import string

def checkio(text):
    
    """
    We iterate through latin alphabet and count each letter in the text.
    Then 'max' selects the most frequent letter.
    For the case when we have several equal letter,
    'max' select the first from them.
    """
    text = text.lower()
    return max(string.ascii_lowercase, key=text.count)
QUESTIONS?
THANKS AND CREDITS

- reveal.js by Hakim El Hattab (MIT)
- Steve McConnell "Code complete 2"
- Robert Martin - "The clean coder"
- http://blog.codinghorror.com
- http://docs.python-guide.org
- https://radon.readthedocs.org
- http://www.pylint.org
- http://www.checkio.org
- STX Next
- My wife Karolina