Wildenstein Plattner Institute

HPI Hasso Plattner Institut
Digital Engineering · Universität Potsdam

Tagging and Captioning Art-Historical Photographs

Alejandro Sierra, Hendrik Rätz, Jona Otholt

13.12.2022

Motivation

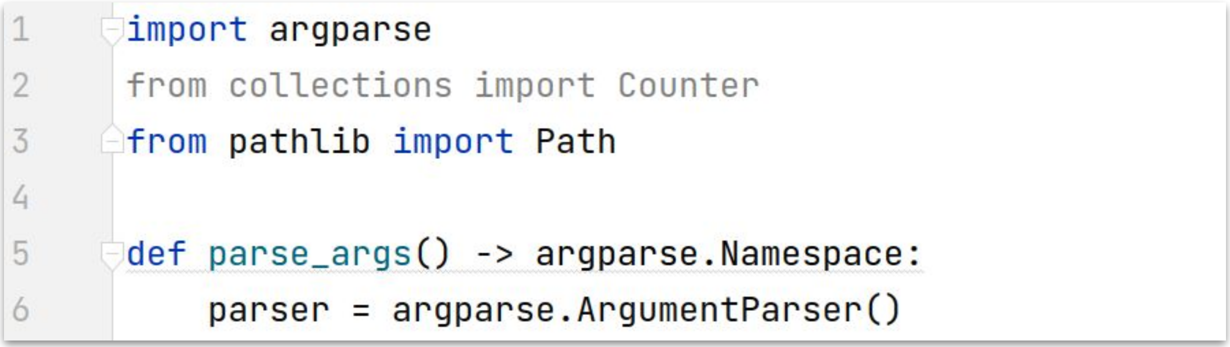
"One of Guido's key insights is that code is read much more often than it is written. The guidelines provided here are intended to improve the readability of code [...]."

— from the PEP 8 Style Guide

PEP 8 – Style Guide for Python Code

How To

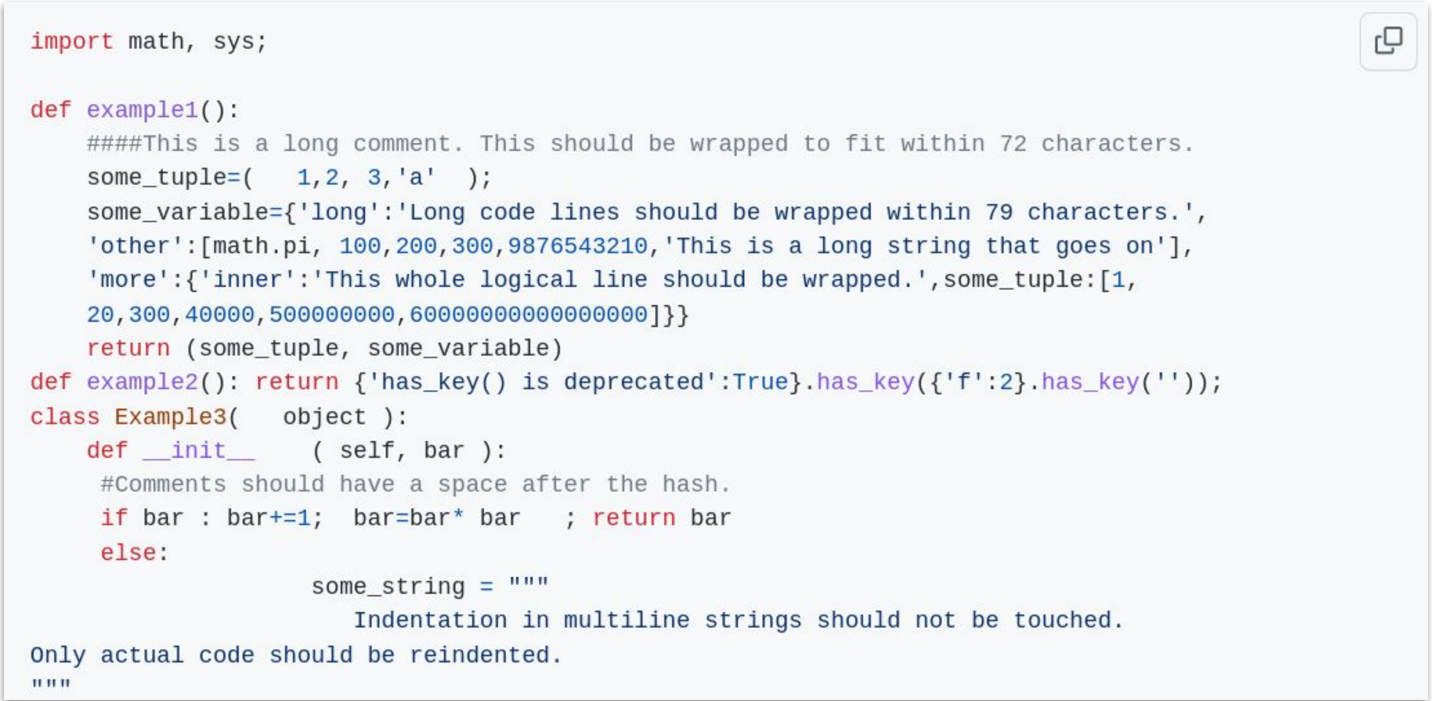
Use IDEs and Built-in Linters



```python
import argparse
from collections import Counter
from pathlib import Path

def parse_args() -> argparse.Namespace:
    parser = argparse.ArgumentParser()
```

Use Code Formatters



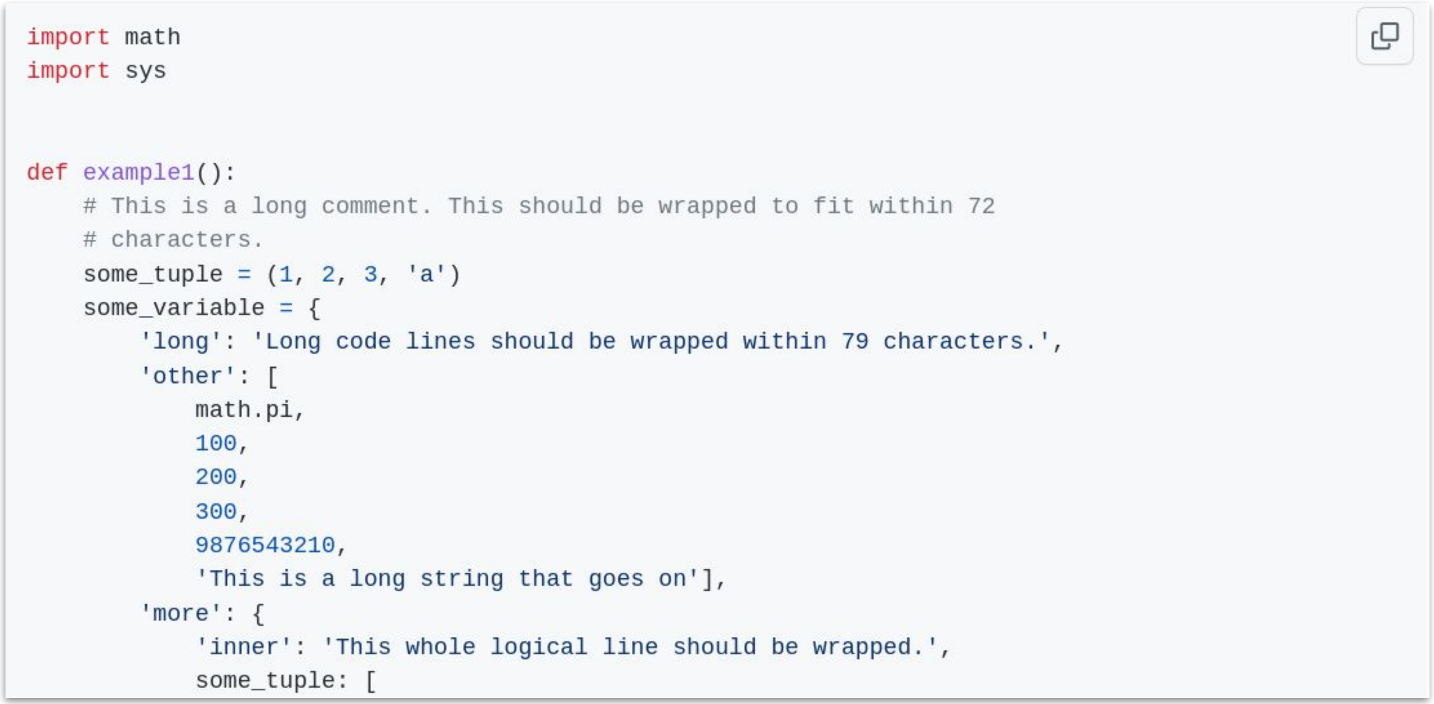
```python
import math, sys;

def example1():
    ####This is a long comment. This should be wrapped to fit within 72 characters.
    some_tuple=(   1,2, 3,'a'  );
    some_variable={'long':'Long code lines should be wrapped within 79 characters.',
    'other':[math.pi, 100,200,300,9876543210,'This is a long string that goes on'],
    'more':{'inner':'This whole logical line should be wrapped.',some_tuple:[1,
    20,300,40000,500000000,60000000000000000]}}
    return (some_tuple, some_variable)
def example2(): return {'has_key() is deprecated':True}.has_key({'f':2}.has_key(''));
class Example3(   object ):
    def __init__    ( self, bar ):
     #Comments should have a space after the hash.
     if bar : bar+=1;  bar=bar* bar   ; return bar
     else:
                    some_string = """
                       Indentation in multiline strings should not be touched.
Only actual code should be reindented.
"""
```

Tagging and Captioning Art-Historical Photographs

13.12.2022

Use Code Formatters



```
import math
import sys


def example1():
    # This is a long comment. This should be wrapped to fit within 72
    # characters.
    some_tuple = (1, 2, 3, 'a')
    some_variable = {
        'long': 'Long code lines should be wrapped within 79 characters.',
        'other': [
            math.pi,
            100,
            200,
            300,
            9876543210,
            'This is a long string that goes on'],
        'more': {
            'inner': 'This whole logical line should be wrapped.',
            some_tuple: [
```

Use Code Formatters

The Uncompromising Code Formatter

Test passing | docs passing | coverage 96% | license MIT | pypi v22.10.0 | downloads 300M | conda-forge 5.7M | code style black

> "Any color you like."

"A style guide is about consistency.
Consistency with this style guide is important.
Consistency within a project is more important."

— from the PEP 8 Style Guide

Good Reasons for Not Following a Guide

1. When applying the guideline **would make the code less readable**, even for someone who is used to reading code that follows this PEP.
2. To be **consistent with surrounding code** that also breaks it (maybe for historic reasons)
3. Because the code in question predates the introduction of the guideline and there is **no other reason to be modifying that code**.

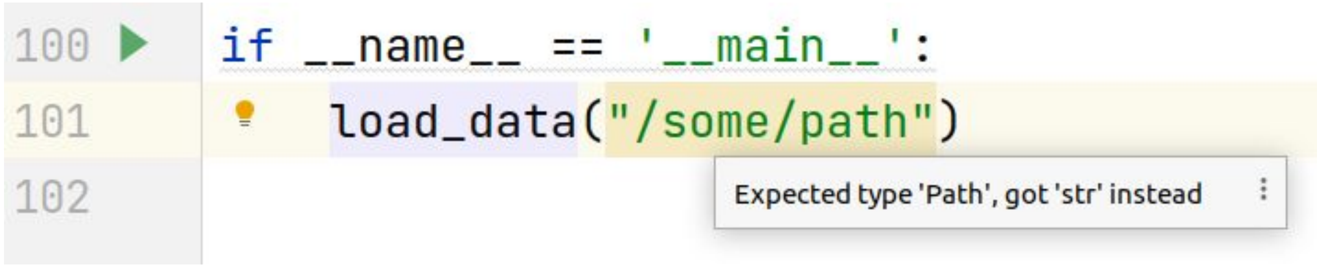
Suggestions

Naming Conventions

- `_single_leading_underscore`: weak “internal use” indicator.
- `__double_leading_underscore`: when naming a class attribute, invokes name mangling (inside class FooBar, __boo becomes _FooBar__boo
- `__double_leading_and_trailing_underscore__`
  “magic” objects or attributes that live in user-controlled namespaces. E.g. `__init__`, `__import__` or `__file__`. Never invent such names; only use them as documented.
- `trailing_underscore_`: If your public attribute name collides with a reserved keyword, append a single trailing underscore to your attribute name

Type Annotations

```
def load_data(path: Path) -> List[str]:
    with open(path) as f:
        return [line.strip() for line in f.readlines()]
```



```
if __name__ == '__main__':
    load_data("/some/path")
```

Expected type 'Path', got 'str' instead

Tagging and Captioning Art-Historical Photographs

13.12.2022

Important Examples

String Formatting: **f-Strings**

Correct: f-String

```
name = ‘Guido’
print(f‘Hello {name}’)
```

Wrong: %-Operator

```
print('Hello, %s' % name)
```

Also wrong: .format()

```
print('Hello, {}'.format(name))
```

Dealing with Paths: **pathlib**

```
file_path = root_dir / ‘file.txt’
```

vs.

```
file_path = os.path.join(root_dir, ‘file.txt’)
```

Miscellaneous

- Avoid code duplications
- Keep extensibility in mind
- Avoid magic strings/magic numbers
- Short code vs. readable code
- Try to be consistent with single and double quoted strings

# Use comments

Try to keep mentioned style guides and concepts in mind.

They will be applied in the final review and during grading ;)