Python Best Practice

Minchang (Carson) Zhang

Disclaimer

This presentation is entirely subjective, and is based on author’s experience, and what’s widely accepted by Python devs

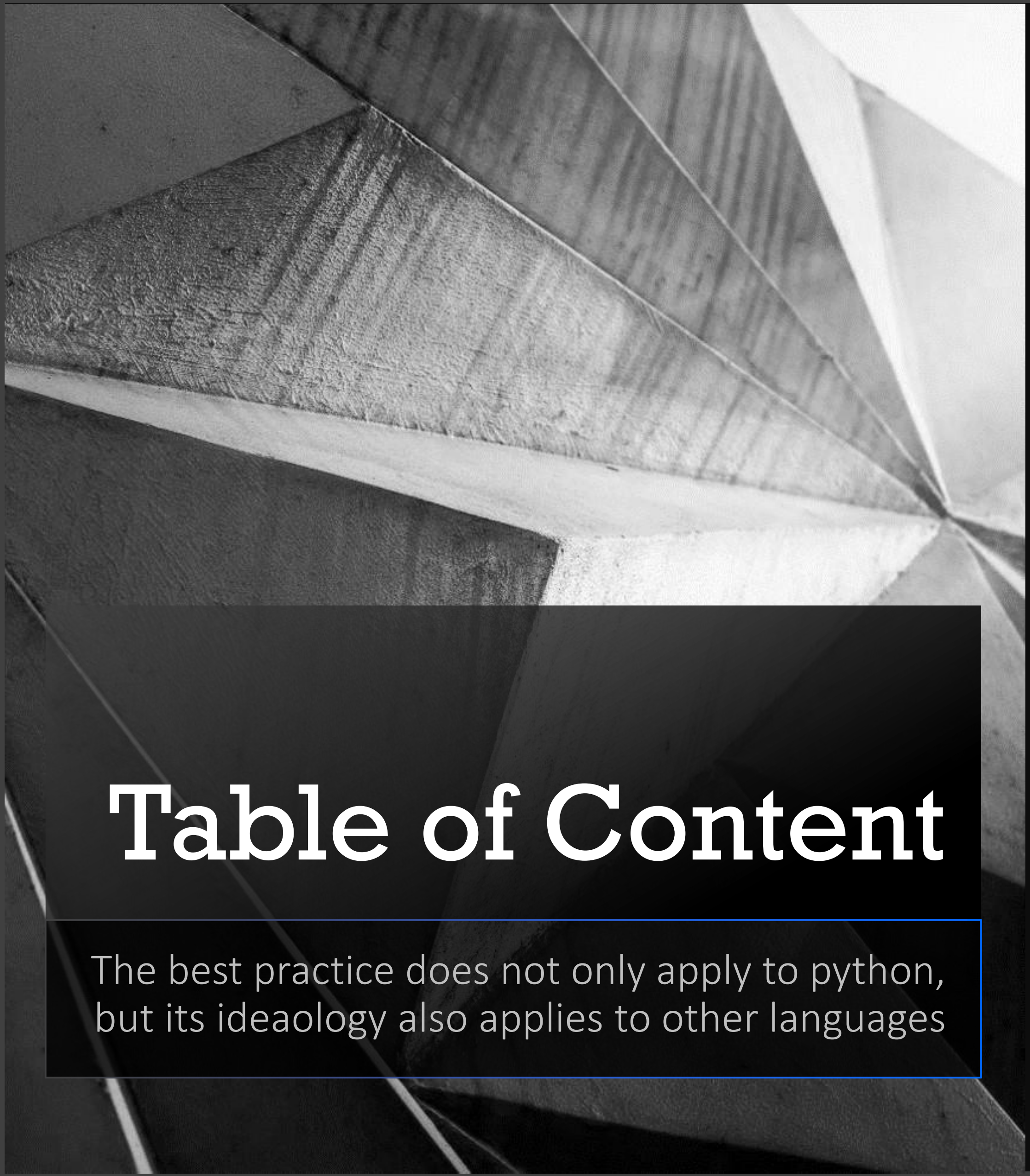


Table of Content

The best practice does not only apply to python, but its ideaology also applies to other languages

Python Best Practice

Python Enhancement Proposals
PEPs

Python Best Practice

Virtual Environment

Virtualenv

- Most widely used
- Virtualenvwrapper makes everything much easier
  - mkvirtualenv
  - workon & deactivate
  - setvirtualenvproject

Anaconda

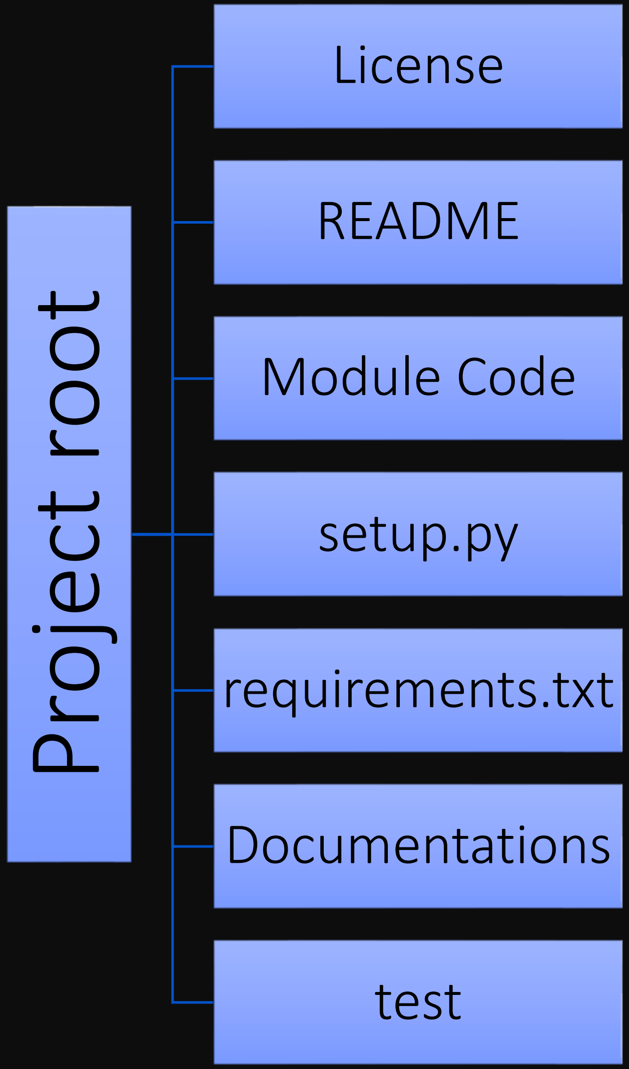
- Most widely used in the AI areas (ML, DS, etc)
- Integration of many packages
- Commands slightly different or Win and Mac
- Some commands
  - conda create –n <env_name>
  - conda install <package_name>
  - activate & deactivate

Pipenv

- New standard of python packages
- Combination of pip and virtualenv
- Dependency version lock
- Some Commands:
  - pipenv shell
  - pipenv install <package_name>
  - pipenv install --dev

Python Best Practice

Code Structure



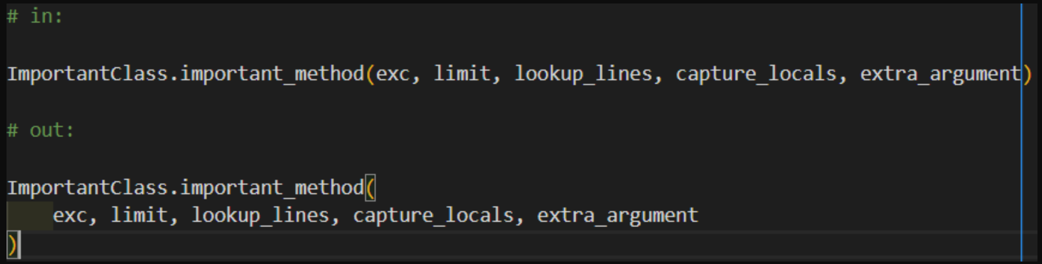
Python Best Practice

Formatting, linting, and typing - https://www.python.org/dev/peps/pep-0008/

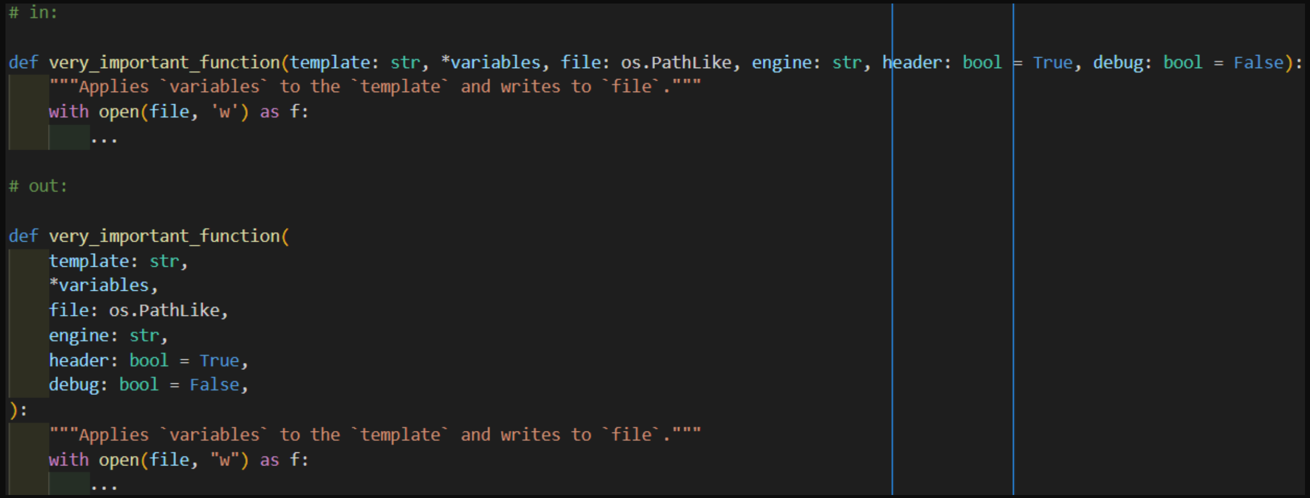
PEP 8 – Style Guide for Python Code

Black – the Uncompromising Code Formatter

- More readability, great formatting
- One of the official supported formatter for VSCode
- Django has accepted using black as formatting as of 5/10/19



```
# in:

ImportantClass.important_method(exc, limit, lookup_lines, capture_locals, extra_argument)

# out:

ImportantClass.important_method(
    exc, limit, lookup_lines, capture_locals, extra_argument
)
```



```
# in:

def very_important_function(template: str, *variables, file: os.PathLike, engine: str, header: bool = True, debug: bool = False):
    """Applies `variables` to the `template` and writes to `file`."""
    with open(file, 'w') as f:
        ...

# out:

def very_important_function(
    template: str,
    *variables,
    file: os.PathLike,
    engine: str,
    header: bool = True,
    debug: bool = False,
):
    """Applies `variables` to the `template` and writes to `file`."""
    with open(file, "w") as f:
        ...
```

Python Best Practice

Formatting, linting, and typing - https://www.python.org/dev/peps/pep-0008/

Indentation

- Python recognizes both space and tab for indentation level
- PEP8 suggests using space
- **4 spaces per indentation level**
- **must never mix spaces and tabs**

Strings

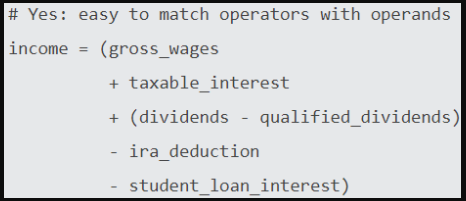
- People use to use single quote ‘’ coz its easier to type
- Double quote “” is used by English language
- **Black suggests to use double quotes to reduce confusion**
- You can use *--skip-string-normalization* to stop black formatting your quotations

Line length

- PEP8 suggests all lines limit to 79 characters, and 72 characters for comments
- Many companies use line length from 100 to 120 characters
- **Black suggests using 88 characters per line**

Line break

- **PEP8 suggests line break after binary operators for readability**

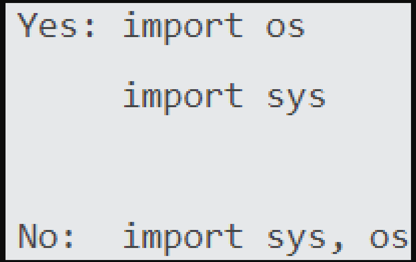


```
# Yes: easy to match operators with operands
income = (gross_wages
          + taxable_interest
          + (dividends - qualified_dividends)
          - ira_deduction
          - student_loan_interest)
```

- **Surround top-level function and class definitions with two blank lines**
- **Method definitions inside a class are surrounded by a single blank line**

Imports

- **Be explicit**
- **Imports should be on separate lines**



```
Yes: import os
     import sys


No:  import sys, os
```

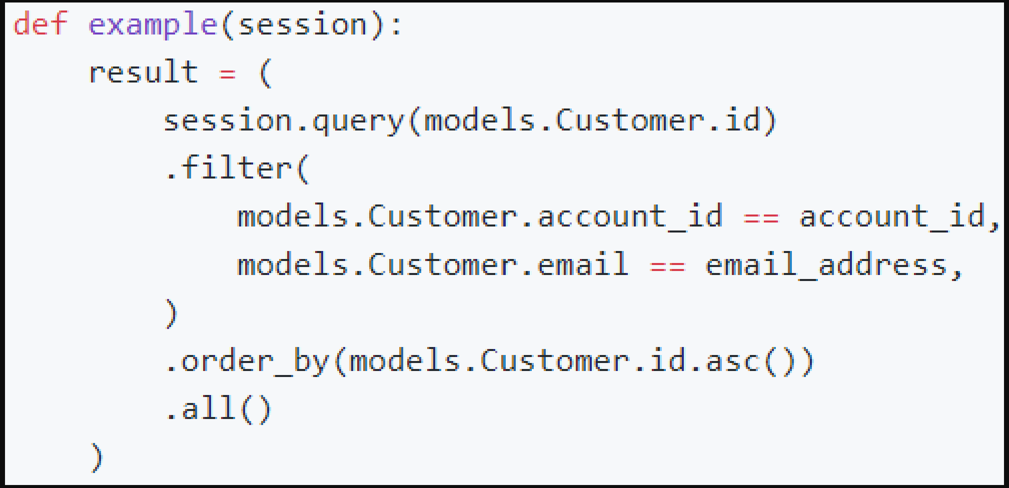
- Imports order:
  1. Standard lib
  2. Related 3rd party
  3. Local lib

Python Best Practice

Formatting, linting, and typing - https://www.python.org/dev/peps/pep-0008/

Call Chains

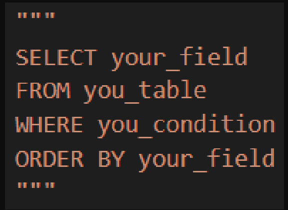
- Some popular APIs like ORMs use call chaining



```python
def example(session):
    result = (
        session.query(models.Customer.id)
        .filter(
            models.Customer.account_id == account_id,
            models.Customer.email == email_address,
        )
        .order_by(models.Customer.id.asc())
        .all()
    )
```

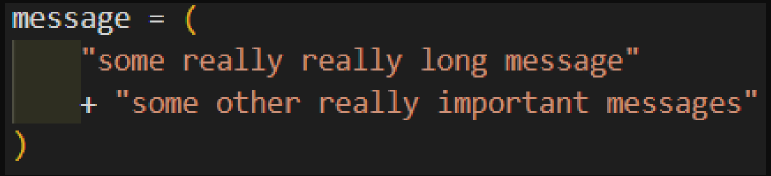
Long String

- When you have a super long string, you can break it up to a few lines
- Use triple quotation



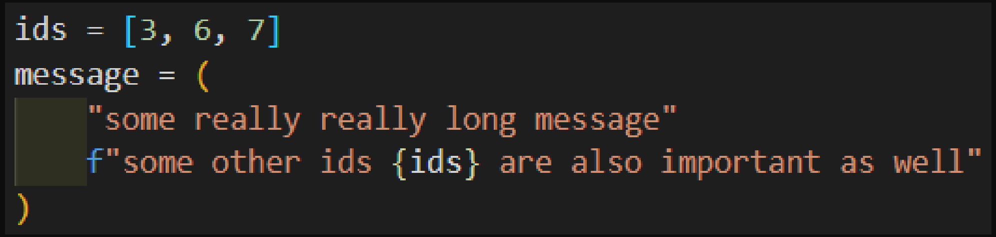
```
"""
SELECT your_field
FROM you_table
WHERE you_condition
ORDER BY your_field
"""
```

- Use plus sign, but it creates more strings



```python
message = (
    "some really really long message"
    + "some other really important messages"
)
```

- Or do multiple lines



```python
ids = [3, 6, 7]
message = (
    "some really really long message"
    f"some other ids {ids} are also important as well"
)
```

Python Best Practice

Formatting, linting, and typing - https://www.python.org/dev/peps/pep-0008/

Naming Convention

- Use meaningful words rather than single letters
- Function name: all lower case and separate with underscore
  i.e. def my_function()
- Class name: Capitalized words all together
  i.e. class MyClass:
- Constants: All capital words
  i.e. MY_CONSTANT = 5
- _single_leading_underscore: weak "internal use" indicator
- __double_leading_underscore: invokes name mangling for class attribute

White Space

- Immediately inside parentheses, brackets or braces
- Between a trailing comma and a following close parenthesis
- Immediately before a comma, semicolon, or colon
- Immediately before the open parenthesis that starts the argument list of a function call or starts an indexing or slicing
- More than one space around an assignment (or other) operator to align it with another

```
Yes: spam(ham[1], {eggs: 2})
No:  spam( ham[ 1 ], { eggs: 2 } )
```

```
Yes: foo = (0,)
No:  bar = (0, )
```

```
Yes: if x == 4: print x, y; x, y = y, x
No:  if x == 4 : print x , y ; x , y = y , x
```

```
Yes: spam(1)
No:  spam (1)
```

```
Yes: dct['key'] = lst[index]
No:  dct ['key'] = lst [index]
```

```
Yes:

  x = 1
  y = 2
  long_variable = 3
```

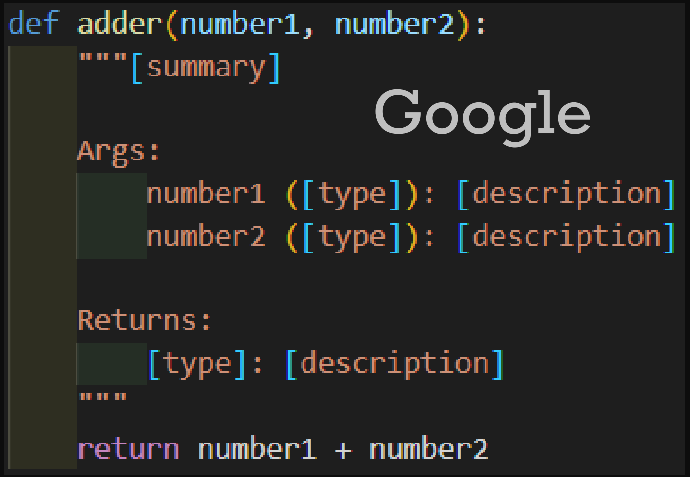
```
No:

  x             = 1
  y             = 2
  long_variable = 3
```

Python Best Practice

Formatting, linting, and typing - https://www.python.org/dev/peps/pep-0257/

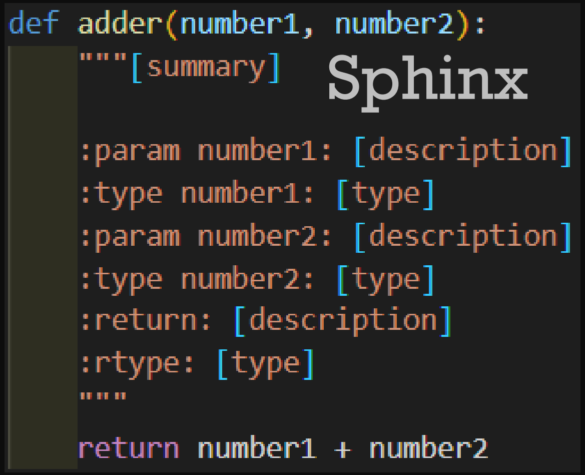
Doc String

- Write docstrings for all public modules, functions, classes, and methods
- Triple quotation marks “”””” for long string or comments
- PEP 257 describes good docstring conventions



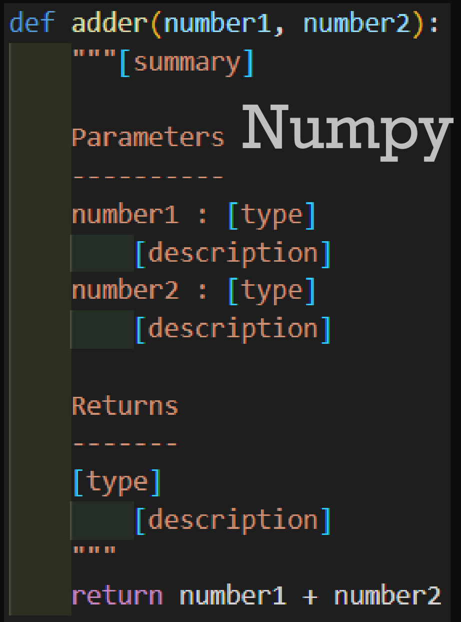
Google

```python
def adder(number1, number2):
    """[summary]

    Args:
        number1 ([type]): [description]
        number2 ([type]): [description]

    Returns:
        [type]: [description]
    """
    return number1 + number2
```



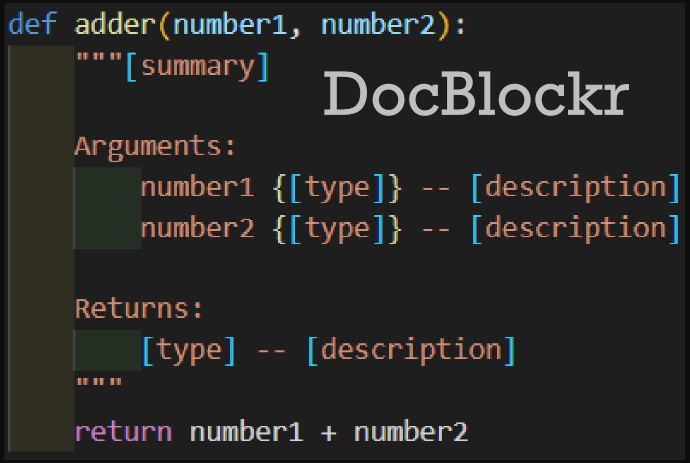
Sphinx

```python
def adder(number1, number2):
    """[summary]

    :param number1: [description]
    :type number1: [type]
    :param number2: [description]
    :type number2: [type]
    :return: [description]
    :rtype: [type]
    """
    return number1 + number2
```



Numpy

```python
def adder(number1, number2):
    """[summary]

    Parameters
    ----------
    number1 : [type]
        [description]
    number2 : [type]
        [description]

    Returns
    -------
    [type]
        [description]
    """
    return number1 + number2
```



DocBlockr

```python
def adder(number1, number2):
    """[summary]

    Arguments:
        number1 {[type]} -- [description]
        number2 {[type]} -- [description]

    Returns:
        [type] -- [description]
    """
    return number1 + number2
```

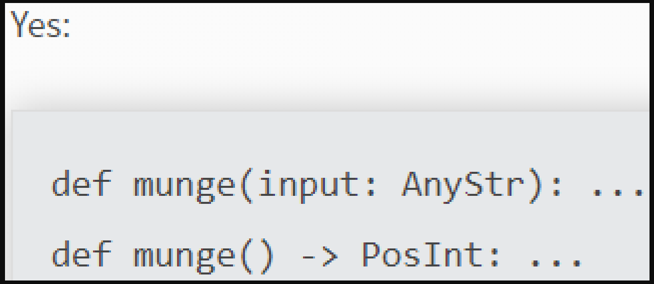
Python Best Practice

Formatting, linting, and typing - https://www.python.org/dev/peps/pep-0526/

Function Annotation

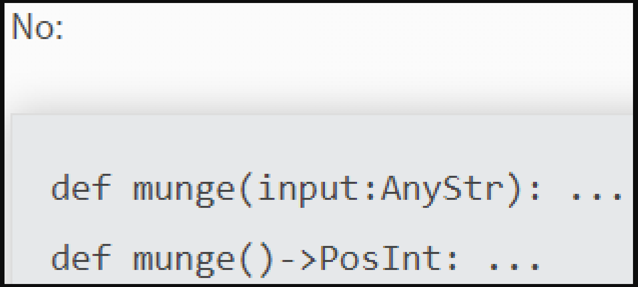
- Python is weak typing language
- PEP 526 is accepted for Python 3.6 as type hints (PEP 484) but not type reinforcement

```
def greeting(name: str) -> str:
    return 'Hello ' + name
```



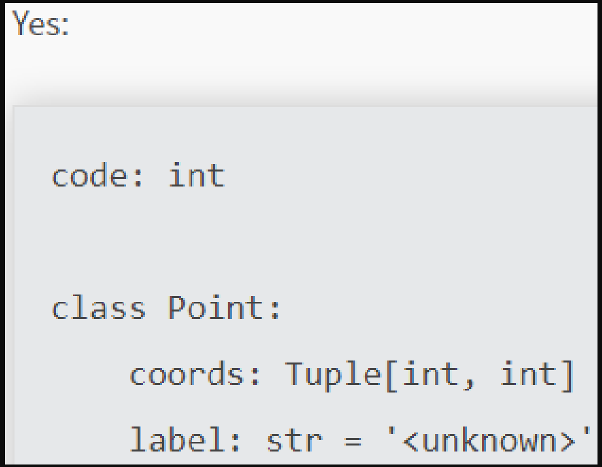
Yes:

```
def munge(input: AnyStr): ...
def munge() -> PosInt: ...
```



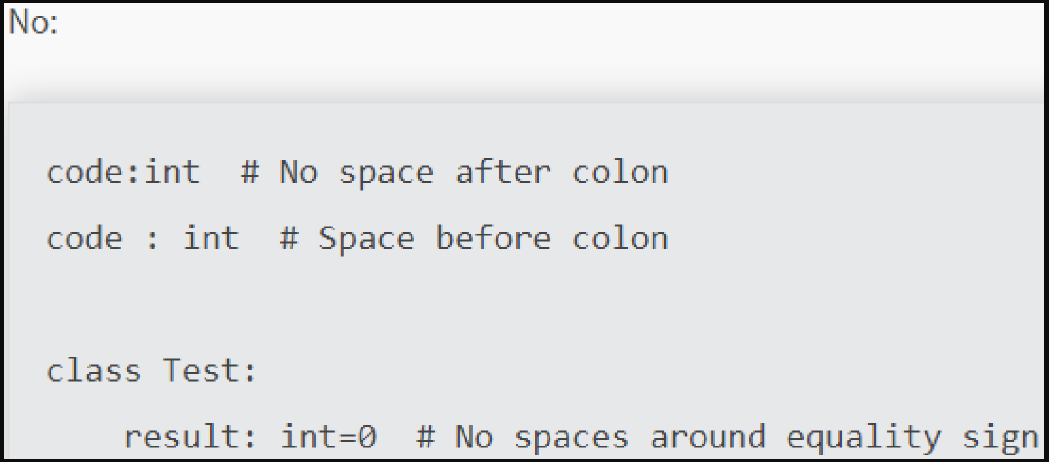
No:

```
def munge(input:AnyStr): ...
def munge()->PosInt: ...
```



Yes:

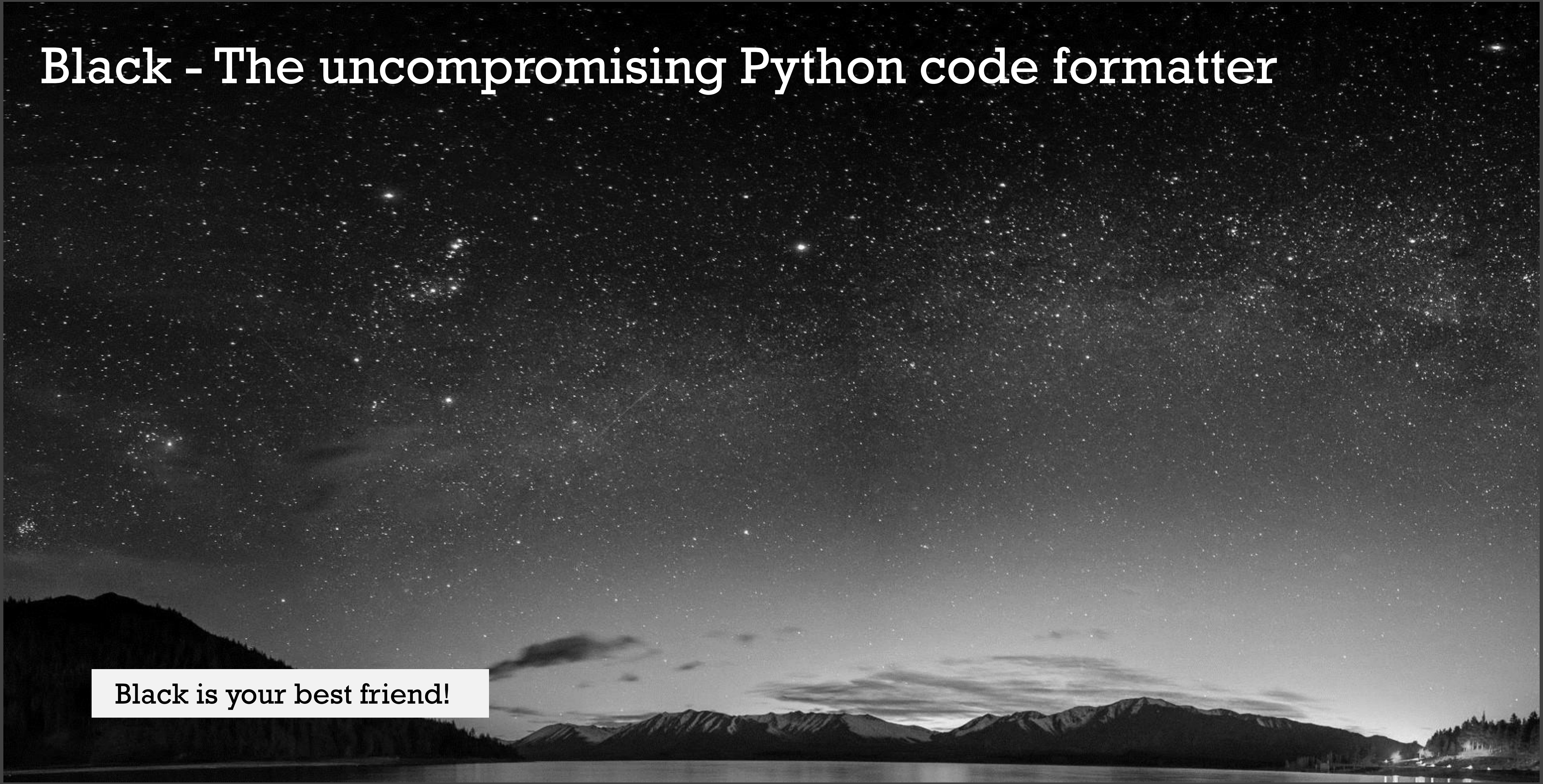
```
code: int

class Point:
    coords: Tuple[int, int]
    label: str = '<unknown>'
```



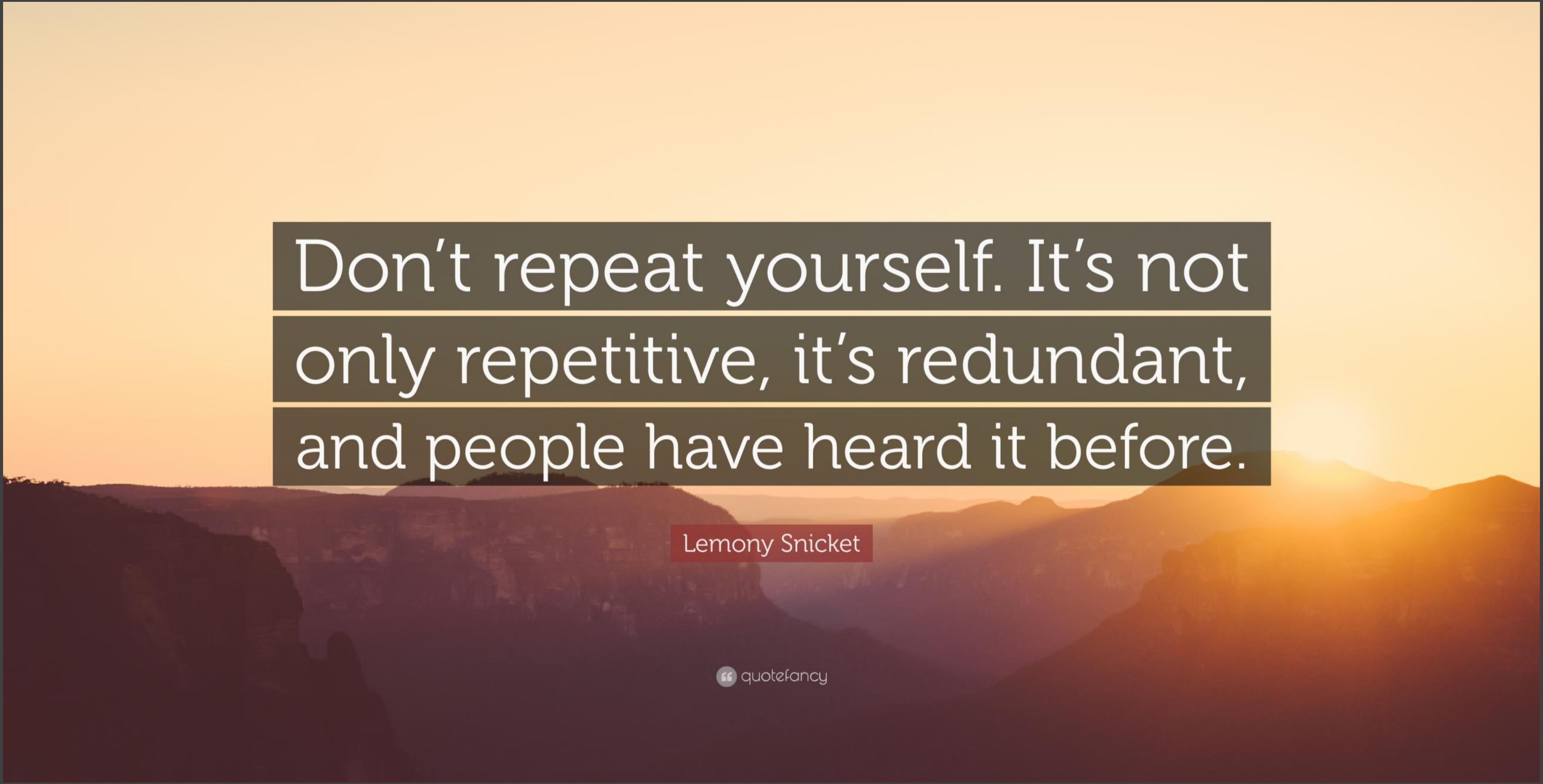
No:

```
code:int  # No space after colon
code : int  # Space before colon

class Test:
    result: int=0  # No spaces around equality sign
```

Black - The uncompromising Python code formatter



Black is your best friend!


Don’t repeat yourself. It’s not only repetitive, it’s redundant, and people have heard it before.

Lemony Snicket

quotefancy

Quote from here

Deal with Legacy Code

Take it easy
Piece by piece

Deal with Legacy Code

Code Formatting Refractor

- Talk to your team leader/manager first
- Respect your company policy
- Refractor piece by piece

Helpful Tools

- Black – Yes, again, black
- isort - sort imports alphabetically, and automatically separated into sections
- flake8 – style reinforcement
- mypy – static type check
- pytest – built upon but better than unittest
- pytest-cov – test code coverage
- Git hooks with pre-commit
- cookiecutter – project template
- pysnoob & pudb – better debugging

Thank You

Minchang (Carson) Zhang

yycdesign.slack.com

PyYYC