PASADENA CITY COLLEGE

CIS 112

Intro to Programming Using Python

Module 3 Part 1

Agenda for the Day!

- PEP8,
- PEP 257,
- code layout,
- comments and docstrings,
- naming conventions,
- string quotes and whitespaces

Readability!

CONFORM (OR DIE!)

- Code is read much more often than it is written.
- The guidelines provided here are intended to improve the **readability** of code an make it consistent across the wide spectrum of Python code. **Note: this isn’t about functionality!**
- A style guide is about consistency. When code is presented in a consistent and easily readable format, it relieves the cognitive load of trying to parse out what a program is doing. It frees the reader to focus on the substance of the code rather than figuring out what the elements of it are.
- However, know when to be inconsistent – sometimes style guide recommendations just aren’t applicable.
- In particular: do not break backwards compatibility just to comply with this PEP!
- Some other good reasons to ignore a particular guideline:
    - When applying the guideline would make the code less readable, even for someone who is used to reading code.
    - To be consistent with surrounding code that also breaks it.
    - Because the code in question predates the introduction of the guideline and there is no other reason to be modifying that code.
    - When the code needs to remain compatible with older versions of Python that don’t support the feature recommended by the style guide

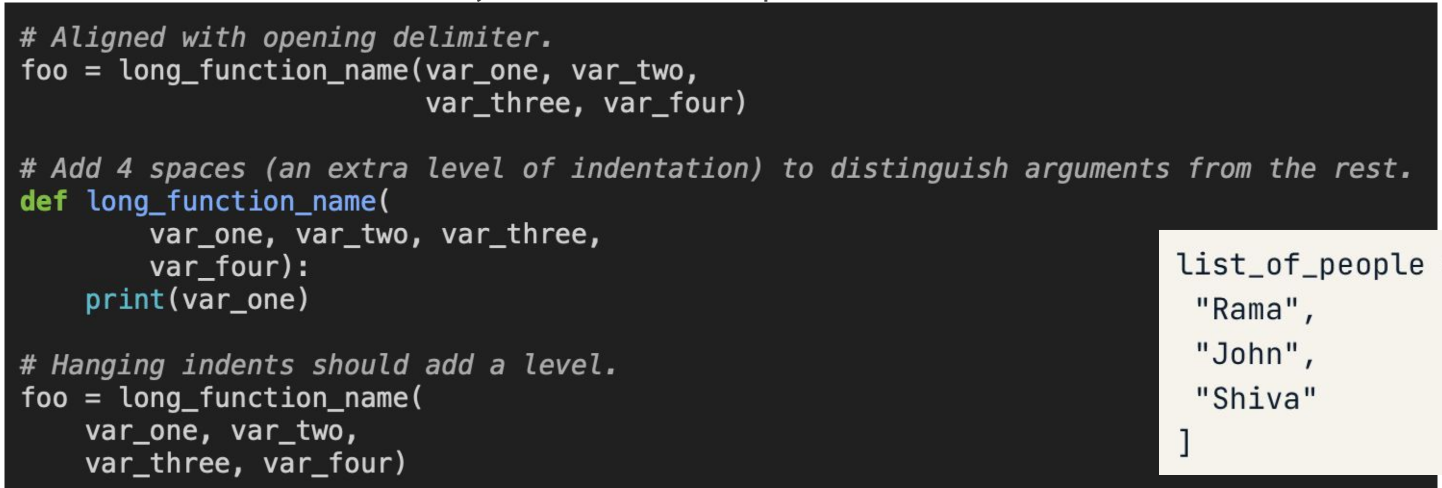
PEP-8

Enhance me!

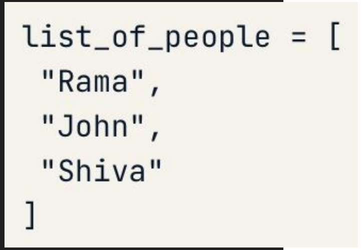
- Python is developed as a collaborative, free and open source project.
- A "PEP" (Python Enhancement Proposal) is a written proposal used in Python development.
- One of the earliest PEPs, PEP8, is a consensus set of style and formatting rules for writing "standard" style Python, so your code has the right look for anyone else reading or modifying it.

Key rules

- Indentation
    - Indent code blocks with **4 spaces**. A def/if/for/while line ends with a colon, and its contained lines begin on the next line, all indented 4 spaces
    - Early on with Python, some people use 2 spaces or tabs. Those practices have died out, and now 4 spaces is the standard.
-



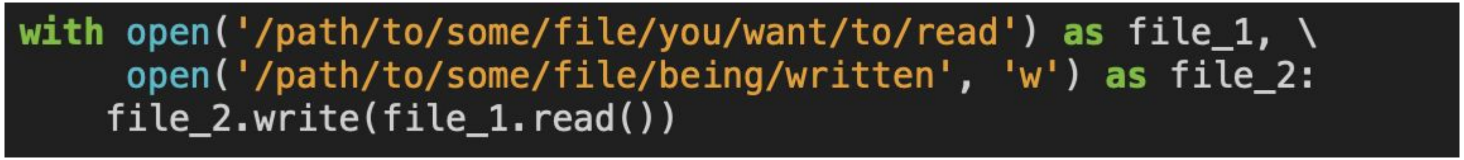
```
# Aligned with opening delimiter.
foo = long_function_name(var_one, var_two,
                         var_three, var_four)

# Add 4 spaces (an extra level of indentation) to distinguish arguments from the rest.
def long_function_name(
        var_one, var_two, var_three,
        var_four):
    print(var_one)

# Hanging indents should add a level.
foo = long_function_name(
    var_one, var_two,
    var_three, var_four)
```



```
list_of_people = [
 "Rama",
 "John",
 "Shiva"
]
```

Line Length

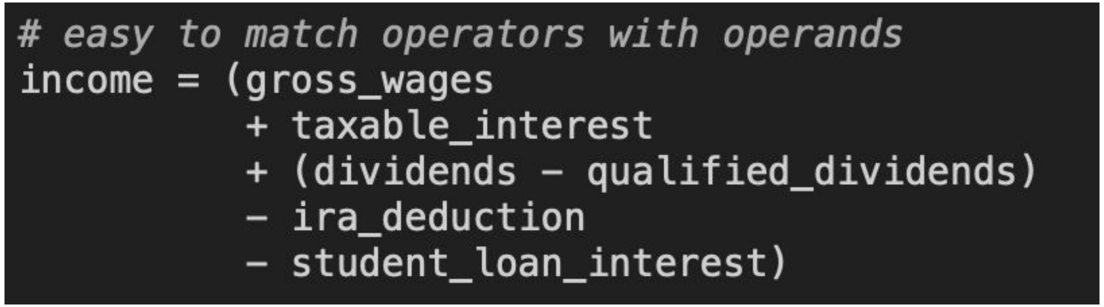
- Generally, it's good to aim for a line length of 79 characters in your Python code.
  - Following this target number has many advantages:
    - It is possible to open files side by side to compare;
    - You can view the whole expression without scrolling horizontally, which adds to better readability and understanding of the code.
  - Comments should have 72 characters of line length.
- The preferred way of wrapping long lines is by using Python’s implied line continuation inside parentheses, brackets and braces. Long lines can be broken over multiple lines by wrapping expressions in parentheses. These should be used in preference to using a backslash for line continuation.
- Backslashes may still be appropriate at times. For example, long, multiple with-statements could not use implicit continuation before Python 3.10, so backslashes were acceptable for that case:



```python
with open('/path/to/some/file/you/want/to/read') as file_1, \
     open('/path/to/some/file/being/written', 'w') as file_2:
    file_2.write(file_1.read())
```

Binary operator line breaks

- For decades the recommended style was to break after binary operators. But this can hurt readability so now breaking prior to it is preferred.
- In Python code, it is permissible to break before or after a binary operator, as long as the convention is consistent locally.



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

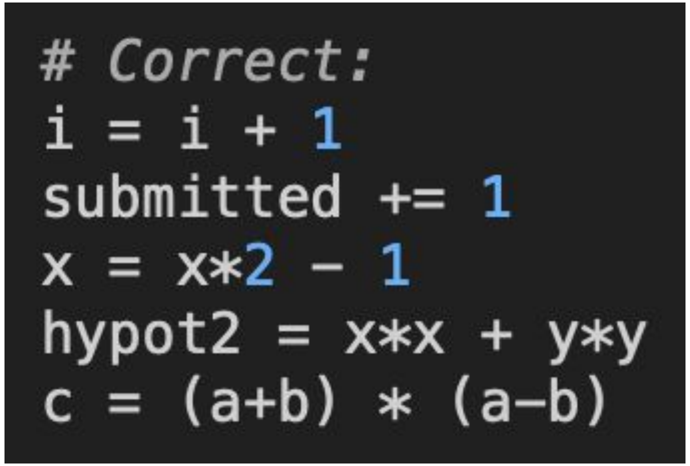
Blank Lines

- Surround top-level function and class definitions with two blank lines.
- Method definitions inside a class are surrounded by a single blank line.
- Extra blank lines may be used (sparingly) to separate groups of related functions. Blank lines may be omitted between a bunch of related one-liners (e.g. a set of dummy implementations).
- Use blank lines in functions, sparingly, to indicate logical sections.

```
import unittest

class SwapTestSuite(unittest.TestCase):
    """
        Swap Operation Test Case
    """
    def setUp(self):
        self.a = 1
        self.b = 2

    def test_swap_operations(self):
        instance = Swap(self.a,self.b)
        value1, value2 =instance.get_swap_values()
        self.assertEqual(self.a, value2)
        self.assertEqual(self.b, value1)


class OddOrEvenTestSuite(unittest.TestCase):
    """
        This is the Odd or Even Test case Suite
    """
    def setUp(self):
        self.value1 = 1
        self.value2 = 2
```

Whitespace

- Avoid trailing whitespace anywhere. Some editors don’t preserve it and many projects (like CPython itself) have pre-commit hooks that reject it.
- Always surround these binary operators with a single space on either side:
  a. assignment (=),
  b. augmented assignment (+=, -= etc.),
  c. comparisons (==, <, >, !=, <>, <=, >=, in, not in, is, is not),
  d. Booleans (and, or, not).
- If operators with different priorities are used, consider adding whitespace around the operators with the lowest priority(ies).
- Never use more than one space, and always have the same amount of whitespace on both sides of a binary operator
- Avoid extraneous whitespace in the following situations:
- Immediately inside parentheses, brackets or braces
  a. Between a trailing comma and a following close parenthesis:
  b. Immediately before a comma, semicolon, or colon:
  c. Immediately before the open parenthesis that starts the argument list of a function call:



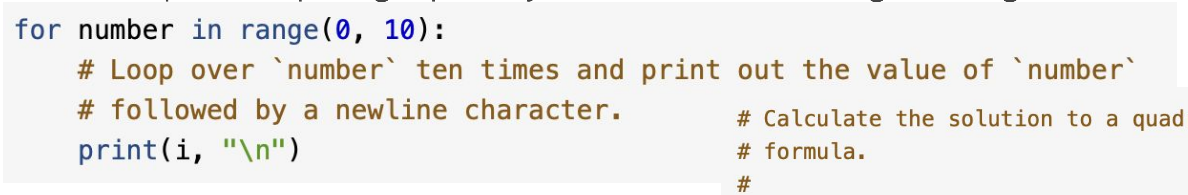
```
# Correct:
i = i + 1
submitted += 1
x = x*2 - 1
hypot2 = x*x + y*y
c = (a+b) * (a-b)
```

Comments

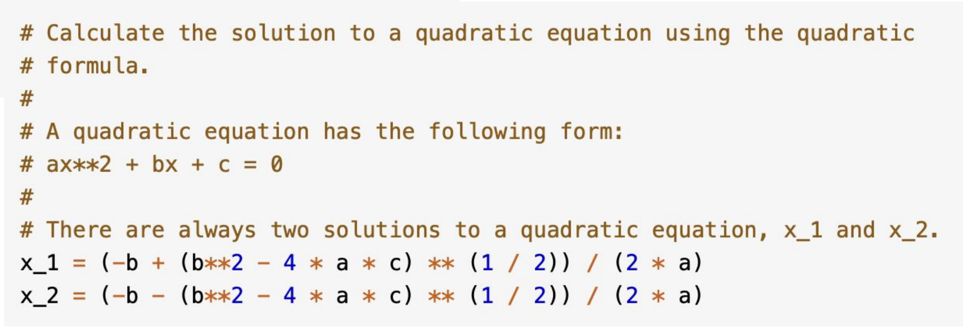
- Comments that contradict the code are worse than no comments. Always make a priority of keeping the comments up-to-date when the code changes!
- Comments should be complete sentences. The first word should be capitalized, unless it is an identifier that begins with a lower case letter (never alter the case of identifiers!).

Block Comments

- Use block comments to document a small section of code. These are useful when you have to write several lines of code to perform a single action, such as importing data from a file or updating a database entry. They’re important in helping others understand the purpose and functionality of a given code block.
- PEP 8 provides the following rules for writing block comments:
  - Indent block comments to the same level as the code that they describe.
  - Start each line with a # followed by a single space.
  - Separate paragraphs by a line containing a single #



```python
for number in range(0, 10):
    # Loop over `number` ten times and print out the value of `number`
    # followed by a newline character.
    print(i, "\n")
```

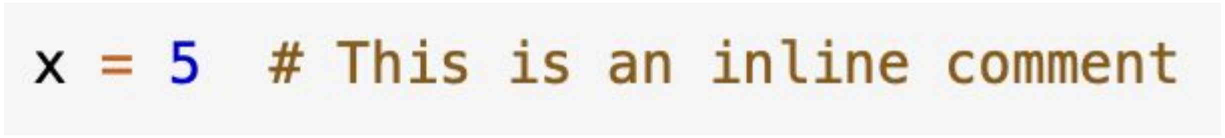


```python
# Calculate the solution to a quadratic equation using the quadratic
# formula.
#
# A quadratic equation has the following form:
# ax**2 + bx + c = 0
#
# There are always two solutions to a quadratic equation, x_1 and x_2.
x_1 = (-b + (b**2 - 4 * a * c) ** (1 / 2)) / (2 * a)
x_2 = (-b - (b**2 - 4 * a * c) ** (1 / 2)) / (2 * a)
```

Inline Comments

Inline comments explain a single statement in a piece of code. They’re useful to remind you, or explain to others, why a certain line of code is necessary. Here’s what PEP 8 has to say about them:

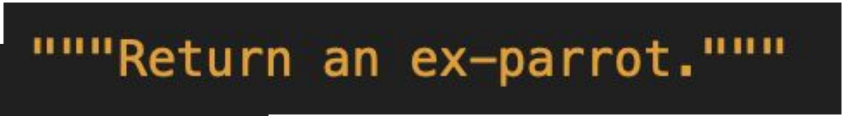
- Use inline comments sparingly.
- Write inline comments on the same line as the statement they refer to.
- Separate inline comments from the statement by two or more spaces.
- Start inline comments with a # and a single space, like block comments.
- Don’t use them to explain the obvious.



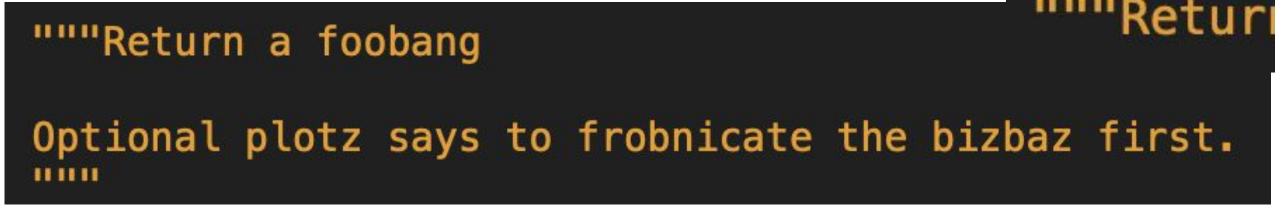
```python
x = 5  # This is an inline comment
```

Docstrings

- Documentation strings, or docstrings, are strings enclosed in triple double quotation marks (""") or triple single quotation marks (''') that appear on the first line of any function, class, method, or module.
- While PEP 8 mentions docstrings, they represent a large enough topic that there’s a separate document, PEP 257 that’s entirely dedicated to docstrings.
  - most notably: the """that ends a multiline docstring should be on a line by itself



```
"""Return an ex-parrot."""
```



```
"""Return a foobang

Optional plotz says to frobnicate the bizbaz first.
"""
```

Naming Conventions

- When you program in Python, you'll most certainly make use of a naming convention, a set of rules for choosing the character sequence that should be used for identifiers which denote variables, types, functions, and other entities in source code and documentation.
- If you're not sure what naming styles are out there, consider the following ones:
  a. b or a single lowercase letter;
  b. B or a single uppercase letter;
  c. lowercase
  d. UPPERCASE
  e. lower_case_with_underscores
  f. UPPER_CASE_WITH_UNDERSCORES
  g. CapitalizedWords, which is also known as CapWords, CamelCase or StudlyCaps.
  h. mixedCase
  i. Capitalized_Words_With_Underscores

General Recommendations

The following table shows you some general guidelines on how to name your identifiers

Note:

- Do not use 'l', 'O' or 'I' as a single variable name: these characters look similar to zero (0) and (1) in some fonts.

Identifier	***Convention***
Module	lowercase
Class	CapWords
Functions	lowercase
Methods	lowercase
Type variables	CapWords
Constants	UPPERCASE
Package	lowercase

PEP8 Guides

More references:

- https://peps.python.org/pep-0008/
- https://realpython.com/python-pep8/
- https://www.datacamp.com/tutorial/pep8-tutorial-python-code
- https://cs.stanford.edu/people/nick/py/python-style1.html#:~:text=A%20%22PEP%22%20(Python%20Enhancement,else%20reading%20or%20modifying%20it.