PEP 8: Style Guide for Python Code

PEP 8, sometimes spelled PEP8 or PEP-8, is a document that provides guidelines and best practices on how to write Python code. It was written in 2001 by Guido van Rossum, Barry Warsaw, and Nick Coghlan. The primary focus of PEP 8 is to improve the readability and consistency of Python code.

What is PEP?

PEP stands for Python Enhancement Proposal, and there are several of them. A PEP is a document that describes new features proposed for Python and documents aspects of Python, like design and style, for the community. Check this link for all the PEPs. PEP 8 is one of the most important PEPs. Below, some of the important style guides from PEP 8 that will be used in this course. For the complete list please check this link.

Naming Convention

Type	Naming Convention	Examples
Function	Use a lowercase word or words. Separate words by underscores to improve readability.	`function`, `my_function`
Variable	Use a lowercase single letter, word, or words. Separate words with underscores to improve readability.	`x`, `var`, `my_variable`
Class	Start each word with a capital letter. Do not separate words with underscores. This style is called camel case.	`Model`, `MyClass`
Method	Use a lowercase word or words. Separate words with underscores to improve readability.	`class_method`, `method`
Constant	Use an uppercase single letter, word, or words. Separate words with underscores to improve readability.	`CONSTANT`, `MY_CONSTANT`, `MY_LONG_CONSTANT`
Module	Use a short, lowercase word or words. Separate words with underscores to improve readability.	`module.py`, `my_module.py`

Package	Use a short, lowercase word or words. Do not separate words with underscores.	package, mypackage

Choosing names

When you name your variables, use descriptive names. you may be tempted to choose simple, single-letter lowercase names, like x. But, unless you're using x as the argument of a mathematical function, it's not clear what x represents.

Example:

```
>>> # Not recommended
>>> x = 'John Smith'
>>> # Recommended
>>> name = 'John Smith'
```

Maximum Line Length and Line Breaking

PEP 8 suggests lines should be limited to 79 characters. This is because it allows you to have multiple files open next to one another, while also avoiding line wrapping.

Example on breaking before a binary operator:

```
# Not Recommended
total = (first_variable +
         second_variable -
         third_variable)
# Recommended
total = (first_variable
         + second_variable
         - third_variable)
```

Indentation

Indentation, or leading whitespace, is extremely important in Python. The indentation level of lines of code in Python determines how statements are grouped together.

Consider the following example:

```
x = 3
if x > 5:
    print('x is larger than 5')
```

The indented print statement lets Python know that it should only be executed if the if statement returns True.

The key indentation rules laid out by PEP 8 are the following:

- Use 4 consecutive spaces to indicate indentation.
- Prefer spaces over tabs.

Whitespace in Expressions and Statements

Whitespace can be very helpful in expressions and statements when used properly. If there is not enough whitespace, then code can be difficult to read, as it's all bunched together. If there's too much whitespace, then it can be difficult to visually combine related terms in a statement.

Whitespace Around Binary Operators

Surround the following binary operators with a single space on either side:

- Assignment operators (=, +=, -=, and so forth)
- Comparisons (==, !=, >, <. >=, <=) and (is, is not, in, not in)
- Booleans (and, not, or)

When there's more than one operator in a statement, adding a single space before and after each operator can look confusing. Instead, it is better to only add whitespace around the operators with the lowest priority, especially when performing mathematical manipulation. Here are a couple examples:

```
# Recommended
y = x**2 + 5
z = (x+y) * (x-y)
# Not Recommended
y = x ** 2 + 5
z = (x + y) * (x - y)
```

When to Avoid Adding Whitespace

In some cases, adding whitespace can make code harder to read. Too much whitespace can make code overly sparse and difficult to follow. PEP 8 outlines very clear examples where whitespace is inappropriate.

The most important place to avoid adding whitespace is at the end of a line. This is known as **trailing whitespace**. It is invisible and can produce errors that are difficult to trace.

The following list outlines some cases where you should avoid adding whitespace:

- Immediately inside parentheses, brackets, or braces:

```
# Recommended
my_list = [1, 2, 3]

# Not recommended
my_list = [ 1, 2, 3, ]
```

- Before a comma, semicolon, or colon:

```
x = 5
y = 6

# Recommended
print(x, y)

# Not recommended
print(x , y)
```

- Before the open parenthesis that starts the argument list of a function call:

```
def double(x):
   return x * 2

# Recommended
double(3)

# Not recommended
double (3)
```

- Before the open bracket that starts an index or slice:

```
# Recommended
list[3]

# Not recommended
list [3]
```

- Between a trailing comma and a closing parenthesis:

```
# Recommended
tuple = (1,)

# Not recommended
tuple = (1, )
```

- To align assignment operators:

```
# Recommended
```

```
var1 = 5
var2 = 6
some_long_var = 7

# Not recommended
var1          = 5
var2          = 6
some_long_var = 7
```

Comments

You should use comments to document code as it's written. It is important to document your code so that you, and any collaborators, can understand it. When you or someone else reads a comment, they should be able to easily understand the code the comment applies to and how it fits in with the rest of your code.

Here are some key points to remember when adding comments to your code:

- Limit the line length of comments and docstrings to 72 characters.
- Use complete sentences, starting with a capital letter.
- Make sure to update comments if you change your code.

Block Comments

PEP 8 provides the following rules for writing block comments:

- Indent block comments to the same level as the code they describe.
- Start each line with a # followed by a single space.
- Separate paragraphs by a line containing a single #.

Example:

```
for i in range(0, 10):
    # Loop over i ten times and print out the value of i, followed by a
    # new line character
    print(i, '\n')
```

Inline Comments

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

Example:

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