University of
South Australia

Problem Solving and Programming

Python Style Guide

University of South Australia

Copyright Notice

Do not remove this notice.

COMMONWEALTH OF AUSTRALIA
Copyright Regulations 1969

WARNING

This material has been produced and communicated to you by or on behalf of the University of South Australia pursuant to Part VB of the *Copyright Act 1968* (**the Act**).

The material in this communication may be subject to copyright under the Act. Any further reproduction or communication of this material by you may be the subject of copyright protection under the Act.

Do not remove this notice.

Sources / References

- PEP 8 – Style Guide for Python Code
  http://www.python.org/dev/peps/pep-0008/
- Deitel & Deitel. *C++ How To Program.* Fifth Edition. Pearson Education Inc. 2005.
- Eckel, Bruce. Thinking in C++, 2nd ed. Volume 1. 2000
  http://www.mindview.net/
- Bjarne Stroustrup's C++ Style and Technique FAQ
  http://public.research.att.com/~bs/bs_faq2.html
- JSF air vehicle C++ coding standards
  http://public.research.att.com/~bs/JSF-AV-rules.pdf
- Code Conventions for the Java Programming Language. Sun Microsystems, Inc. http://java.sun.com/docs/codeconv/

Why have code conventions?

- Consistent coding conventions provide a way of standardizing the coding style making code easy to read and understandable.

- Programs should-
  - be consistent in style.
  - be understandable and maintainable by yourself and other programmers (especially important when working in a team development environment).

- Enable programmers to produce code that is more correct, reliable and maintainable.

Why have code conventions?

- Code conventions are important to programmers for a number of reasons [Java Code Conventions, Sun Microsystems, Inc.]:
  - 80% of the lifetime cost of a piece of software goes to maintenance.
  - Hardly any software is maintained for its whole life by the original author.
  - Code conventions improve the readability of the software, allowing engineers to understand new code more quickly and thoroughly.
  - If you ship your source code as a product, you need to make sure it is as well packaged and clean as any other product you create.

Code layout

- Indentation
  - Use 4 spaces per indentation level.
  - Avoid mixing tabs and spaces.
  - Spaces are the most preferred way of indenting Python code.
  - Avoid the use of the tabs, as they may vary between systems – different default tab settings may result in unreadable code.

Code layout

- Blank lines
  - Don’t be afraid to use blank lines in your code in order to break up sections of code.
  - Use blank lines in code to indicate logical sections.
  - Blank lines are essential for readability.
  - If you write an essay without paragraphs you lose marks, if you write code without blank lines and appropriate whitespace (see next slide), you also lose marks.

- Place only one statement per line.
- Blank lines and space characters enhance a programs readability.
- Blank lines should be placed:
  - between a definition and executable statements.
  - between function definitions.
  - before a comment, block or significant section of code.

Code layout

- Whitespace in Expressions and Statements
  - Avoid extraneous whitespace in the following situations:

    (http://www.python.org/dev/peps/pep-0008/)

    - Immediately inside parentheses, brackets or braces.

```
Yes: spam(ham[1])
No:  spam( ham[ 1 ] )
```

    - Immediately before a comma, semicolon, or colon:

```
Yes: if x == 4:
         print(x, y)
No:  if x == 4 :
         print(x , y)
```

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

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

UniSA

Code layout

- Whitespace in Expressions and Statements
  - Avoid extraneous whitespace in the following situations:

    (http://www.python.org/dev/peps/pep-0008/)

    - Immediately before the open parenthesis that starts an indexing or slicing:

```
Yes: list[index]
No:  list [index]
```

    - More than one space around an assignment (or other) operator to align it with another.

```
Yes:
     x = 1
     long_var = 2
No:
     x        = 1
     long_var = 2
```

Code layout

- Whitespace in Expressions and Statements
  - Other recommendations:

    (http://www.python.org/dev/peps/pep-0008/)

    - Always surround the following operators with a single space on either side: assignment (=), augmented assignment (+=, -= etc.), comparisons (==, <, >, !=, <=, >=, in, not in, is, is not), Booleans (and, or, not).

    - Use spaces around arithmetic operators:

```
Yes: i = i + 1
     submitted += 1
     x = x * 2 - 1
     hypot2 = x * x + y * y
     c = (a + b) * (a - b)
No:  i=i+1
     submitted +=1
     x = x*2-1
     hypot2 = x*x + y*y
     c = (a+b) * (a-b)
```

Code layout

- Imports
  - Import statements usually be on separate lines.
  - Import statements should be placed at the top of the file, just after any comments and before any variable definitions.

Comments

- Every program that you write should begin with a comment that describes the author, date, and purpose of the program.
- Comments are applied to all –
  - variable definitions
  - functions
  - significant sections of code
- Comments should appear on a separate line and be preceded by a blank line.
- Comments should be indented to the same level as the code that is being commented.
- Comments should be complete sentences.
- If a comment is a phrase or sentence, its first word should be capitalized.

Comments

- Inline comments may be used for variable definitions only.
- Inline comments should be lined up.
- Define each variable on a separate line allowing for an inline comment next to the definition.

  For example:

```
randomNo = 0        # Random number generated.
guess = 0           # Users guess at number.
response = 'y'      # Users response to prompt.
```

Naming conventions

- Identifier names should be meaningful – designed to indicate intended use.

- Function and variable names
  - First letter is lowercase.
  - If it consists of more than one word, distinguish each word by capitalizing the first letter of each word or by using an underscore ('_').
  - For example-
    - Variable names

```
number;
countLetters;
count_letters;
```

    - Function names
      - Function names should be lowercase, with words separated by underscores as necessary to improve readability.

- Constant names
  - Capitalize entire word.
  - Distinguish each word by using an underscore ('_').
  - For example-

```
MAX_SIZE = 10
```

UniSA

Misc...

- Do not use break statements to exit out of loops.
- Avoid the use of magic numbers.

Remember...

- Consistency, consistency, consistency...
- Apply the conventions we have just discussed consistently to your code.
- This will improve the readability of your code, make it more understandable and maintainable.