**3.8 Comments and Docstrings**

Be sure to use the right style for module, function, method docstrings and inline comments.

**3.8.1 Docstrings**

Python uses *docstrings* to document code. A docstring is a string that is the first statement in a package, module, class or function. These strings can be extracted automatically through the \_\_doc\_\_ member of the object and are used by pydoc. (Try running pydoc on your module to see how it looks.) Always use the three double-quote """ format for docstrings (per [PEP 257](https://www.google.com/url?sa=D&q=http://www.python.org/dev/peps/pep-0257/)). A docstring should be organized as a summary line (one physical line not exceeding 80 characters) terminated by a period, question mark, or exclamation point. When writing more (encouraged), this must be followed by a blank line, followed by the rest of the docstring starting at the same cursor position as the first quote of the first line. There are more formatting guidelines for docstrings below.

**3.8.2 Modules**

Every file should contain license boilerplate. Choose the appropriate boilerplate for the license used by the project (for example, Apache 2.0, BSD, LGPL, GPL)

Files should start with a docstring describing the contents and usage of the module.

"""A one line summary of the module or program, terminated by a period.

Leave one blank line. The rest of this docstring should contain an

overall description of the module or program. Optionally, it may also

contain a brief description of exported classes and functions and/or usage

examples.

Typical usage example:

foo = ClassFoo()

bar = foo.FunctionBar()

"""

**3.8.3 Functions and Methods**

In this section, “function” means a method, function, or generator.

A function must have a docstring, unless it meets all of the following criteria:

* not externally visible
* very short
* obvious

A docstring should give enough information to write a call to the function without reading the function’s code. The docstring should describe the function’s calling syntax and its semantics, but generally not its implementation details, unless those details are relevant to how the function is to be used. For example, a function that mutates one of its arguments as a side effect should note that in its docstring. Otherwise, subtle but important details of a function’s implementation that are not relevant to the caller are better expressed as comments alongside the code than within the function’s docstring.

The docstring should be descriptive-style ("""Fetches rows from a Bigtable.""") rather than imperative-style ("""Fetch rows from a Bigtable."""). The docstring for a @property data descriptor should use the same style as the docstring for an attribute or a [function argument](https://google.github.io/styleguide/pyguide.html#doc-function-args) ("""The Bigtable path.""", rather than """Returns the Bigtable path.""").

A method that overrides a method from a base class may have a simple docstring sending the reader to its overridden method’s docstring, such as """See base class.""". The rationale is that there is no need to repeat in many places documentation that is already present in the base method’s docstring. However, if the overriding method’s behavior is substantially different from the overridden method, or details need to be provided (e.g., documenting additional side effects), a docstring with at least those differences is required on the overriding method.

Certain aspects of a function should be documented in special sections, listed below. Each section begins with a heading line, which ends with a colon. All sections other than the heading should maintain a hanging indent of two or four spaces (be consistent within a file). These sections can be omitted in cases where the function’s name and signature are informative enough that it can be aptly described using a one-line docstring.

[***Args:***](https://google.github.io/styleguide/pyguide.html#doc-function-args)

List each parameter by name. A description should follow the name, and be separated by a colon followed by either a space or newline. If the description is too long to fit on a single 80-character line, use a hanging indent of 2 or 4 spaces more than the parameter name (be consistent with the rest of the docstrings in the file). The description should include required type(s) if the code does not contain a corresponding type annotation. If a function accepts \*foo (variable length argument lists) and/or \*\*bar (arbitrary keyword arguments), they should be listed as \*foo and \*\*bar.

[***Returns: (or Yields: for generators)***](https://google.github.io/styleguide/pyguide.html#doc-function-returns)

Describe the type and semantics of the return value. If the function only returns None, this section is not required. It may also be omitted if the docstring starts with Returns or Yields (e.g. """Returns row from Bigtable as a tuple of strings.""") and the opening sentence is sufficient to describe the return value. Do not imitate ‘NumPy style’ ([example](http://numpy.org/doc/stable/reference/generated/numpy.linalg.qr.html)), which frequently documents a tuple return value as if it were multiple return values with individual names (never mentioning the tuple). Instead, describe such a return value as: “Returns: A tuple (mat\_a, mat\_b), where mat\_a is …, and …”. The auxiliary names in the docstring need not necessarily correspond to any internal names used in the function body (as those are not part of the API).

[***Raises:***](https://google.github.io/styleguide/pyguide.html#doc-function-raises)

List all exceptions that are relevant to the interface followed by a description. Use a similar exception name + colon + space or newline and hanging indent style as described in *Args:*. You should not document exceptions that get raised if the API specified in the docstring is violated (because this would paradoxically make behavior under violation of the API part of the API).

**def** **fetch\_smalltable\_rows**(table\_handle: smalltable.Table,

keys: Sequence[Union[bytes, str]],

require\_all\_keys: bool **=** False,

) **->** Mapping[bytes, tuple[str, ...]]:

"""Fetches rows from a Smalltable.

Retrieves rows pertaining to the given keys from the Table instance

represented by table\_handle. String keys will be UTF-8 encoded.

Args:

table\_handle: An open smalltable.Table instance.

keys: A sequence of strings representing the key of each table

row to fetch. String keys will be UTF-8 encoded.

require\_all\_keys: If True only rows with values set for all keys will be

returned.

Returns:

A dict mapping keys to the corresponding table row data

fetched. Each row is represented as a tuple of strings. For

example:

{b'Serak': ('Rigel VII', 'Preparer'),

b'Zim': ('Irk', 'Invader'),

b'Lrrr': ('Omicron Persei 8', 'Emperor')}

Returned keys are always bytes. If a key from the keys argument is

missing from the dictionary, then that row was not found in the

table (and require\_all\_keys must have been False).

Raises:

IOError: An error occurred accessing the smalltable.

"""

Similarly, this variation on Args: with a line break is also allowed:

**def** **fetch\_smalltable\_rows**(table\_handle: smalltable.Table,

keys: Sequence[Union[bytes, str]],

require\_all\_keys: bool **=** False,

) **->** Mapping[bytes, tuple[str, ...]]:

"""Fetches rows from a Smalltable.

Retrieves rows pertaining to the given keys from the Table instance

represented by table\_handle. String keys will be UTF-8 encoded.

Args:

table\_handle:

An open smalltable.Table instance.

keys:

A sequence of strings representing the key of each table row to

fetch. String keys will be UTF-8 encoded.

require\_all\_keys:

If True only rows with values set for all keys will be returned.

Returns:

A dict mapping keys to the corresponding table row data

fetched. Each row is represented as a tuple of strings. For

example:

{b'Serak': ('Rigel VII', 'Preparer'),

b'Zim': ('Irk', 'Invader'),

b'Lrrr': ('Omicron Persei 8', 'Emperor')}

Returned keys are always bytes. If a key from the keys argument is

missing from the dictionary, then that row was not found in the

table (and require\_all\_keys must have been False).

Raises:

IOError: An error occurred accessing the smalltable.

"""

**3.8.4 Classes**

Classes should have a docstring below the class definition describing the class. If your class has public attributes, they should be documented here in an Attributes section and follow the same formatting as a [function’s Args](https://google.github.io/styleguide/pyguide.html#doc-function-args) section.

**class** **SampleClass**:

"""Summary of class here.

Longer class information...

Longer class information...

Attributes:

likes\_spam: A boolean indicating if we like SPAM or not.

eggs: An integer count of the eggs we have laid.

"""

**def** **\_\_init\_\_**(self, likes\_spam: bool **=** False):

"""Inits SampleClass with blah."""

self.likes\_spam **=** likes\_spam

self.eggs **=** 0

**def** **public\_method**(self):

"""Performs operation blah."""