Chapter 4: Comments and Documentation

Section 4.1: Single line, inline and multiline comments

Comments are used to explain code when the basic code itself isn't clear.

Python ignores comments, and so will not execute code in there, or raise syntax errors for plain English sentences.

Single-line comments begin with the hash character (#) and are terminated by the end of line.

- Single line comment:

```
# This is a single line comment in Python
```

- Inline comment:

```
print("Hello World")  # This line prints "Hello World"
```

- Comments spanning multiple lines have `"""` or `'''` on either end. This is the same as a multiline string, but they can be used as comments:

```
"""
This type of comment spans multiple lines.
These are mostly used for documentation of functions, classes and modules.
"""
```

Section 4.2: Programmatically accessing docstrings

Docstrings are - unlike regular comments - stored as an attribute of the function they document, meaning that you can access them programmatically.

An example function

```
def func():
    """This is a function that does nothing at all"""
    return
```

The docstring can be accessed using the `__doc__` attribute:

```
print(func.__doc__)
```

> This is a function that does nothing at all

```
help(func)
```

> Help on function `func` in module `__main__`:
>
> `func()`
>
> This is a function that does nothing at all

Another example function

`function.__doc__` is just the actual docstring as a string, while the `help` function provides general information about a function, including the docstring. Here's a more helpful example:

```
def greet(name, greeting="Hello"):
    """Print a greeting to the user `name`

    Optional parameter `greeting` can change what they're greeted with."""

    print("{} {}".format(greeting, name))
```

```
help(greet)
```

> Help on function `greet` in module `__main__`:
>
> `greet(name, greeting='Hello')`
>
> Print a greeting to the user `name`
> Optional parameter `greeting` can change what they're greeted with.

Advantages of docstrings over regular comments

Just putting no docstring or a regular comment in a function makes it a lot less helpful.

```
def greet(name, greeting="Hello"):
    # Print a greeting to the user `name`
    # Optional parameter `greeting` can change what they're greeted with.

    print("{} {}".format(greeting, name))
```

```
print(greet.__doc__)
```

> None

```
help(greet)
```

> Help on function greet in module **main**:
>
> `greet(name, greeting='Hello')`

Section 4.3: Write documentation using docstrings

A docstring is a multi-line comment used to document modules, classes, functions and methods. It has to be the first statement of the component it describes.

```
def hello(name):
    """Greet someone.

    Print a greeting ("Hello") for the person with the given name.
    """

    print("Hello "+name)
```

```
class Greeter:
    """An object used to greet people.
```

```
    It contains multiple greeting functions for several languages
    and times of the  day.
    """
```

The value of the docstring can be accessed within the program and is - for example - used by the `help` command.

Syntax conventions
PEP 257

PEP 257 defines a syntax standard for docstring comments. It basically allows two types:

- One-line Docstrings:

According to PEP 257, they should be used with short and simple functions. Everything is placed in one line, e.g:

```
def hello():
    """Say hello to your friends."""
    print("Hello my friends!")
```

The docstring shall end with a period, the verb should be in the imperative form.

- Multi-line Docstrings:

Multi-line docstring should be used for longer, more complex functions, modules or classes.

```
def hello(name, language="en"):
    """Say hello to a person.

    Arguments:
    name: the name of the person
    language: the language in which the person should be greeted
    """

    print(greeting[language]+" "+name)
```

They start with a short summary (equivalent to the content of a one-line docstring) which can be on the same line as the quotation marks or on the next line, give additional detail and list parameters and return values.

Note PEP 257 defines what information should be given within a docstring, it doesn't define in which format it should be given. This was the reason for other parties and documentation parsing tools to specify their own standards for documentation, some of which are listed below and in this question.

Sphinx

Sphinx is a tool to generate HTML based documentation for Python projects based on docstrings. Its markup language used is reStructuredText. They define their own standards for documentation, pythonhosted.org hosts a very good description of them. The Sphinx format is for example used by the pyCharm IDE.

A function would be documented like this using the Sphinx/reStructuredText format:

```
def hello(name, language="en"):
    """Say hello to a person.

    :param name: the name of the person
    :type name: str
    :param language: the language in which the person should be greeted
    :type language: str
```

```
    :return: a number
    :rtype: int
    """

    print(greeting[language]+" "+name)
    return 4
```

Google Python Style Guide

Google has published Google Python Style Guide which defines coding conventions for Python, including documentation comments. In comparison to the Sphinx/reST many people say that documentation according to Google's guidelines is better human-readable.

The pythonhosted.org page mentioned above also provides some examples for good documentation according to the Google Style Guide.

Using the Napoleon plugin, Sphinx can also parse documentation in the Google Style Guide-compliant format.

A function would be documented like this using the Google Style Guide format:

```
def hello(name, language="en"):
    """Say hello to a person.

    Args:
        name: the name of the person as string
        language: the language code string

    Returns:
        A number.
    """

    print(greeting[language]+" "+name)
    return 4
```