Python
Documentation and Test

Documentation

Documentation is automatically extracted to explain your code.

These are, for example, comments enclosed in triple-double quotes after function declarations.

```
def add(num1, num2):
    """Add two numbers."""
```

Example

```
def add (num1, num2):
    """
    Add two numbers.

    Keyword arguments:
    num1 -- an integer or double number (no default)
    num2 -- an integer or double number (no default)

    Returns:
    The sum of the numbers.
    """

    return num1 + num2
```

Style details:

https://www.python.org/dev/peps/pep-0257/

PyDoc

PyDoc is the documentation system distributed with Python.

Best way to invoke it is to import any code, and then type:

```
>>> help(x)
```

Where "x" is a function, module, dotted object method etc.

If you want to see docstrings, then:

```
print(function_name.__doc__)
```

PyDoc

To generate a webpage from documentation, at command prompt:

```
pydoc -w filename
```

(note without the .py)

For more, see:

https://docs.python.org/3/library/pydoc.html

Other software

There is a starter list of alternative software at:

https://wiki.python.org/moin/DocumentationTools

A popular one is Sphinx, which comes with Anaconda:

http://www.sphinx-doc.org/en/stable/

http://www.sphinx-doc.org/en/stable/tutorial.html

http://www.sphinx-doc.org/en/stable/invocation.html#invocation-apidoc

DocTest

DocTest runs short tests built into your documentation.

```
"""
Add two numbers together.


>>> add(1,2)
3
"""
def add (num1, num2):
    return num1 + num2
```

DocTest

To run:

```
python -m doctest -v filename.py
```

Or:

```
>>> import doctest
>>> import filename
>>> doctest.testmod(filename, verbose=True)
```

See:

https://docs.python.org/3/library/doctest.html

Unit tests

Write the tests first, defining success.

Then write the code that satisfies the tests.

For example:

```
#docs.py
def add(num1, num2):
    return num1 + num2
----------------------------------------
import docs
def test_add(self):
        self.assertEqual(docs.add(1,2), 3)
```

See:

https://docs.python.org/3/library/unittest.html

For a list of assertion functions:

https://docs.python.org/3/library/unittest.html#assert-methods

Test Driven Development

Write the tests first, and then write the software to match the tests.

Good for working in teams: if the code matches the test, it shouldn't matter how it does it.

All team code uploaded is tested in a continuous integration process to make sure it works before integration.

https://en.wikipedia.org/wiki/Test-driven_development

https://en.wikipedia.org/wiki/Continuous_integration

References

- http://www.geog.leeds.ac.uk/courses/computing/materials/python/documentation-and-tests/documentation-and-tests.pptx