Sphinx

Python 3.0's documentation system

(works for Python 2.4 and later)

http://sphinx.pocoo.org/

Sphinx output

- Customisable HTML and LaTeX
- HTML in multiple files with:
  - Extensive cross-references
  - Hierarchical structure
- Indices and search facility
- Code highlighting with Pygments

(Python 3.0 example on next slide)

Built-in Types

The following sections describe the standard types that are built into the interpreter.

The principal built-in types are numerics, sequences, mappings, files, classes, instances and exceptions.

Some operations are supported by several object types; in particular, practically all objects can be compared, tested for truth value, and converted to a string (with the `repr()` function or the slightly different `str()` function). The latter function is implicitly used when an object is written by the `print()` function.

Truth Value Testing

Any object can be tested for truth value, for use in an `if` or `while` condition or as operand of the Boolean operations below. The following values are considered false:

- `None`
- `False`
- zero of any numeric type, for example, `0`, `0L`, `0.0`, `0j`.
- any empty sequence, for example, `''`, `()`, `[]`.
- any empty mapping, for example, `{}`.
- instances of user-defined classes, if the class defines a `__bool__()` or `__len__()` method, when that method returns the integer zero or `bool` value `False`. [1]

All other values are considered true — so objects of many types are always true.

Operations and built-in functions that have a Boolean result always return `0` or `False` for false and `1` or `True` for true, unless otherwise stated. (Important exception: the Boolean operations `or` and `and` always return one of their operands.)

Boolean Operations — and, or, not

These are the Boolean operations, ordered by ascending priority:

Operation	Result	Notes
`x or y`	if *x* is false, then *y*, else *x*	(1)
`x and y`	if *x* is false, then *x*, else *y*	(2)
`not x`	if *x* is false, then `True`, else `False`	(3)

Sphinx input

- Text files in reStructuredText format
- Extensions to reStructuredText for “semantic markup”
- Documentation can be in separate files, or can be automatically pulled from docstrings in your code

reStructuredText

- Half way between something like HTML and plain text, e.g.:

```
Title
=====
Blah, *blah*, **blah**.

Section
-------
* Bullet
* List
```

Title

Blah, *blah*, **blah**.

Section

- Bullet
- List

Sphinx semantic markup

```
.. class:: Shark(numberofteeth=Many)
  Creates a shark, has the following
  methods:
  .. method:: bite(victim='passerby')
     Tells the shark who to bite, after a
     biting :attr:`number_of_victims`
     increases by 1.
  and attributes:
  .. attribute:: number_of_victims
     What it sounds like…

Avoid using the :meth:`Shark.bite` method.
```

Using docstrings

- Just use the following Sphinx markup:

```
.. autoclass:: Shark
.. autofunction:: rescue_mission
```

Etc.

Index

- Classes, functions, etc. automatically generate index entries.
- In addition, you can add your own index entries as follows:

```
.. index:
   pair: shark; attack
   single: Jaws
```

B

And in conclusion…

- Sphinx seems a good choice of documentation system as it is:
  - Easy to use and powerful
  - Generates very nice output
  - Likely to be well supported as it is being used for Python 3.0 docs
- See it in action: http://docs.python.org/dev/3.0/
- Download it: http://sphinx.pocoo.org/index.html