Some comments about the Python standard documentation - 2

Author:

Jacques Ducasse, PARIS, FRANCE
jacko21@aliceadsl.fr
07/06/2010

All of these comments concern the standard documentation of Python 3.1.1, distributed on the official Python site. Most of them concern information which was true in previous releases (2.5 or 2.6) and became false due to the changes with the 3.x release of Python: the documentation seems to struggle with the race against the software.

The Python Library Reference

Chapter 7 : String Services

7.1. `string`

♦ Header paragraph:
"…as well as some deprecated legacy functions that are also available as methods on strings."
In the past, this sentence referenced the sub-section **4 Deprecated string functions** (`atof()`, `capitalize()`,…). All theses functions are gone now, and the sentence in the header does no longer reference anything.

♦ "In addition,… regular expressions."
All this block of text (three sentences) is duplicated in the header of **7 String Services** as well as in the header of **7.1. `string`**. One of theses duplication has to disappear (the former, I think).

Chapter 9 : Numeric and Mathematical Modules

♦ `itertools`, `operator`, `functools`
This is not a new feature, but I don't understand the reason why the three modules `itertools`, `operator` and `functools` are located in this chapter dedicated to Numeric and Mathematical Modules. These (very interesting) modules implement some powerful generic capabilities of Python, but without any specialisation with either Mathematics nor Numeric computation. They fall better in a hypothetical Logical – or Functional – Modules section. Consequently, these misplaced (then hidden) modules are difficult to find when needed.

9.9. `operator`

♦ Header : "The functions fall into categories … sequence operations, and abstract type tests."
The last component "abstract type tests" referred to the functions `isCallable()`, `isMappingType()`, `isNumberType()` and `isSequenceType()` until release 2.6. All these functions disappeared in release 3.x ; the member "and abstract type tests" has to do the same.

Chapter 13 : File Formats

13.1. `csv` — CSV File Reading and Writing
13.1.1. Module Contents

`csv.reader()`
♦ "…and returns a string each time its `next()` method is called…"
Should be: `__next__()`.

Chapter 18 : Internet Data Handling

18.1.5. `email`: Internationalized headers

♦ Class `Header` : the method `__unicode__()` does no longer exist.

18.1.6. `email`: Representing character sets

Class `Charset` :

- ♦ The method `encoded_header_len()` does no longer exist.
- ♦ The new (since 3.0) method `header_encode_lines()` is not described.

Chapter 27 : Python Runtime Services

27.11. `inspect`

27.11.1. Types and members

♦ `getmoduleinfo()`
The returned tuple is described twice (carelessly changed from release 2.x doc). Moreover, le last term is named `module_type` in the first, `mtype` in the last and also in the comments. In fact, the true name in the named tuple is `module_type`.

27.11.3. Classes and functions

♦ `getargspec()`
The name of the third term of the returned named tuple is `keywords`, but the comment uses the name `varkw`.

`getargvalues()`

- ♦ The name of the third term of the returned named tuple is `keywords`, but the comment uses the name `varkw`.
- ♦ « it may contain nested lists » : I don't think this is still true, because the tuple arguments are gone in release 3.0 : `def foo(a, (b, c)):`... is now illegal.

♦ `formatargspec()`
The argument `join` is not described.

♦ `formatargvalues()`
The argument `join` is not described.

The Python Language Reference

2.4.1. String and Bytes literals

♦ Note (4) after the table: "Unlike in Standard C, exactly <u>two</u> hex digits are required."
I think it should be <u>four</u> hex digits `\uxxxx`.

4.1. Naming and binding

♦ "The following constructs bind names: … or after `as` in a `with` statement <u>or :keyword.`except` clause</u>."
Problem with meta data: should be:
"…or `except` clause."
with `except` written with the typography of keywords.

♦ Last paragraph but one : “The global statement has the same scope…”
Again, the term `global` should be written with the typography of keywords. Otherwise, this is a countersense.

5.9. Comparisons

♦ Sixth paragraph before the end:
“…the expression `x in y` is equivalent to `any(x is e or x == e for val e in y)`.”
The word `val` has nothing to do there. The expression should be:
“…the expression `x in y` is equivalent to `any(x is e or x == e for e in y)`.”

7.4. The `try` statement

♦ “`sys.exc_info()` returns a 3-tuple consisting of: `exc_type`, the exception class; `exc_value`, the exception instance; `exc_traceback`, a traceback object…”
The tuple returned by `exc_info()` is not a named-tuple, only a simple tuple, and its components are only designated by their index. So, there is no need to give them a name here (`exc_type` and so on) (only significant for old guys who remember theses killed `sys` variables).

7.7. Class definitions

♦ Nothing about keywords in the class parameters, and especially the `metaclass` keyword! Moreover, I think the formal syntax doesn’t include this idiom. `metaclass` keyword is described in chapter **3.3.3. Customizing class creation**, but why not here?

Glossary

iterator
♦ “Repeated calls to the iterator’s `__next__()` (or passing it to the builtin function) `next()` method return successive items in the stream.”
Muddled and overlapping sequence and parenthesis; should be:
“Repeated calls to the iterator’s `__next__()` method (or passing it to the builtin function `next()`) return successive items in the stream.”

♦ “…any further calls to its `next()` method...”
Should be:
“…any further calls to its `__next__()` method...”

That’s all folks!