Interference Between Frames & Paragraph Borders

This document is provided to give a working copy of a bug as reported in Python3docs Bugs #5. This bug is as follows:-

> The bottom border of a Sidebar (which is a frame that contains text) will coalesce with the top inline-border of a paragraph. In other words, the below-frame spacing of the frame will be added to the internal top-spacing of the next paragraph, rather than being placed between the two entity's borders.

To produce this document the following steps were taken:

(a) The styles from bug3.odt were loaded into this document under LO 24.8.3.2

(b) The Frame *"sidebar-02-01 Deleting Items Using the del Statement"* was copied from chapter_03.odt into this document.

(c) A *"Code Box 2"* was copied into the document below the image

Note:

> Almost certainly this is a copy of Python3docs Bugs #3 + 4.

Example (next page):

Deleting Items Using the del Statement

Although the name of the del statement is reminiscent of the word delete, it does not necessarily delete any data. When applied to an object reference that refers to a data item that is not a collection, the del statement unbinds the object reference from the data item and deletes the object reference. For example:

```
>>> x = 8143 # object ref. 'x' created; int of value 8143 created
>>> x
8143
>>> del x    # object ref. 'x' deleted; int ready for garbage collection
>>> x
Traceback (most recent call last):
…
NameError: name 'x' is not defined
```

When an object reference is deleted, Python schedules the data item to which it referred to be garbage-collected if no other object references refer to the data item. When, or even if, garbage collection takes place may be nondeterministic (depending on the Python implementation), so if any cleanup is required we must handle it ourselves. Python provides two solutions to the nondeterminism. One is to use a `try … finally` block to ensure that cleanup is done, and another is to use a `with` statement as we will see in Chapter 8.

When `del` is used on a collection data type such as a `tuple` or a `list`, only the object reference to the collection is deleted. The collection and its items (and for those items that are themselves collections, for their items, recursively) are scheduled for garbage collection if no other object references refer to the collection.

For mutable collections such as *lists*, `del` can be applied to individual items or slices—in both cases using the slice operator, `[]`. If the item or items referred to are removed from the collection, and if there are no other object references referring to them, they are scheduled for garbage collection.

```
for i in range(len(numbers)):
    numbers[i] += 1
```