7.5. `StringIO` — Read and write strings as files

This module implements a file-like class, `StringIO`, that reads and writes a string buffer (also known as *memory files*). See the description of file objects for operations (section File Objects). (For standard strings, see `str` and `unicode`.)

class `StringIO.` **StringIO**([*buffer*])

When a `StringIO` object is created, it can be initialized to an existing string by passing the string to the constructor. If no string is given, the `StringIO` will start empty. In both cases, the initial file position starts at zero.

The `StringIO` object can accept either Unicode or 8-bit strings, but mixing the two may take some care. If both are used, 8-bit strings that cannot be interpreted as 7-bit ASCII (that use the 8th bit) will cause a `UnicodeError` to be raised when `getvalue()` is called.

The following methods of `StringIO` objects require special mention:

`StringIO.` **getvalue**()

Retrieve the entire contents of the "file" at any time before the `StringIO` object's `close()` method is called. See the note above for information about mixing Unicode and 8-bit strings; such mixing can cause this method to raise `UnicodeError`.

`StringIO.` **close**()

Free the memory buffer. Attempting to do further operations with a closed `StringIO` object will raise a `ValueError`.

Example usage:

```python
import StringIO

output = StringIO.StringIO()
output.write('First line.\n')
print >>output, 'Second line.'

# Retrieve file contents -- this will be
# 'First line.\nSecond line.\n'
contents = output.getvalue()

# Close object and discard memory buffer --
# .getvalue() will now raise an exception.
output.close()
```

7.6. `cStringIO` — Faster version of `StringIO`

The module `cStringIO` provides an interface similar to that of the `StringIO` module. Heavy use of `StringIO.StringIO` objects can be made more efficient by using the function `StringIO()` from this module instead.

`cStringIO.` **StringIO**([*s*])

Return a StringIO-like stream for reading or writing.

Since this is a factory function which returns objects of built-in types, there's no way to build your own version using subclassing. It's not possible to set attributes on it. Use the original `StringIO` module in those cases.

Unlike the `StringIO` module, this module is not able to accept Unicode strings that cannot be encoded as plain ASCII strings.

Another difference from the `StringIO` module is that calling `StringIO()` with a string parameter creates a read-only object. Unlike an object created without a string parameter, it does not have write methods. These objects are not generally visible. They turn up in tracebacks as `StringI` and `StringO`.

The following data objects are provided as well:

cStringIO. **InputType**

The type object of the objects created by calling `StringIO()` with a string parameter.

cStringIO. **OutputType**

The type object of the objects returned by calling `StringIO()` with no parameters.

There is a C API to the module as well; refer to the module source for more information.

Example usage:

```python
import cStringIO

output = cStringIO.StringIO()
output.write('First line.\n')
print >>output, 'Second line.'

# Retrieve file contents -- this will be
# 'First line.\nSecond line.\n'
contents = output.getvalue()

# Close object and discard memory buffer --
# .getvalue() will now raise an exception.
output.close()
```