io — Text, Binary, and Raw Stream I/O Tools

Purpose: Implements file I/O and provides classes for working with buffers using file-like API.

The `io` module implements the classes behind the interpreter's built-in `open()` for file-based input and output operations. The classes are decomposed in such a way that they can be recombined for alternate purposes, for example to enable writing Unicode data to a network socket.

In-memory Streams

`StringIO` provides a convenient means of working with text in memory using the file API (`read()`, `write()`, etc.). Using `StringIO` to build large strings can offer performance savings over some other string concatenation techniques in some cases. In-memory stream buffers are also useful for testing, where writing to a real file on disk may slow down the test suite.

Here are a few standard examples of using `StringIO` buffers:

```
# io_stringio.py

import io

# Writing to a buffer
output = io.StringIO()
output.write('This goes into the buffer. ')
print('And so does this.', file=output)

# Retrieve the value written
print(output.getvalue())

output.close()  # discard buffer memory

# Initialize a read buffer
input = io.StringIO('Inital value for read buffer')

# Read from the buffer
print(input.read())
```

This example uses `read()`, but the `readline()` and `readlines()` methods are also available. The `StringIO` class also provides a `seek()` method for jumping around in a buffer while reading, which can be useful for rewinding if a look-ahead parsing algorithm is being used.

```
$ python3 io_stringio.py

This goes into the buffer. And so does this.

Inital value for read buffer
```

To work with raw bytes instead of Unicode text, use `BytesIO`.

```
# io_bytesio.py

import io

# Writing to a buffer
output = io.BytesIO()
output.write('This goes into the buffer. '.encode('utf-8'))
output.write('ÁÇÊ'.encode('utf-8'))

# Retrieve the value written
print(output.getvalue())

output.close()  # discard buffer memory

# Initialize a read buffer
```

```
# Initialize a read buffer
input = io.BytesIO(b'Inital value for read buffer')

# Read from the buffer
print(input.read())
```

The values written to the `BytesIO` must be `bytes` rather than `str`.

```
$ python3 io_bytesio.py

b'This goes into the buffer. \xc3\x81\xc3\x87\xc3\x8a'
b'Inital value for read buffer'
```

Wrapping Byte Streams for Text Data

Raw byte streams such as sockets can be wrapped with a layer to handle string encoding and decoding, making it easier to use them with text data. The `TextIOWrapper` class supports writing as well as reading. The `write_through` argument disables buffering, and flushes all data written to the wrapper through to the underlying buffer immediately.

```
# io_textiowrapper.py

import io

# Writing to a buffer
output = io.BytesIO()
wrapper = io.TextIOWrapper(
    output,
    encoding='utf-8',
    write_through=True,
)
wrapper.write('This goes into the buffer. ')
wrapper.write('ÁÇÊ')

# Retrieve the value written
print(output.getvalue())

output.close()  # discard buffer memory

# Initialize a read buffer
input = io.BytesIO(
    b'Inital value for read buffer with unicode characters ' +
    'ÁÇÊ'.encode('utf-8')
)
wrapper = io.TextIOWrapper(input, encoding='utf-8')

# Read from the buffer
print(wrapper.read())
```

This example uses a `BytesIO` instance as the stream. Examples for `bz2`, `http.server`, and `subprocess` demonstrate using `TextIOWrapper` with other types of file-like objects.

```
$ python3 io_textiowrapper.py

b'This goes into the buffer. \xc3\x81\xc3\x87\xc3\x8a'
Inital value for read buffer with unicode characters ÁÇÊ
```

See also

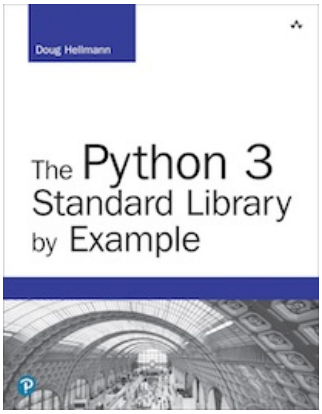
- Standard library documentation for io
- HTTP POST example – Uses the `detach()` of `TextIOWrapper` to manage the wrapper separately from the wrapped socket.
- Efficient String Concatenation in Python – Examines various methods of combining strings and their relative merits.

Quick Links

This page was last updated 2016-12-31.

Navigation

codecs — String Encoding and Decoding
Data Persistence and Exchange



Get the book

The output from all the example programs from PyMOTW-3 has been generated with Python 3.7.1, unless otherwise noted. Some of the features described here may not be available in earlier versions of Python.

Looking for examples for Python 2?

This Site

Module Index
Index

Other Writing

Blog
The Python Standard Library By Example