io

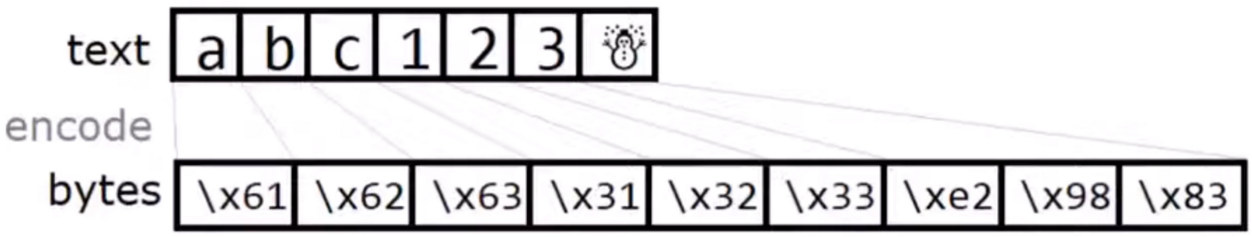
04 November 2022 10:06 AM

WHAT IS IO?

- files on disk are a series of 1s and 0s
- for simplicity, we'll think of them as a series of bytes
- we'll also consider stdin and stdout to be special "files"

WHAT IS IO?

- for textual data, these bytes have a specific encoding
- a character (codepoint) is represented as a specific series of byte(s)



WHAT IS IO?

- a **binary** io procedure takes in binary data (`bytes` / `bytearray`)
- this data is written to the file *unmodified*
- the bytes in your program are represented identically on disk

WHAT IS IO?

- a **text** io procedure takes in text data (py2 `unicode`, py3 `str`)
- this data is first *encoded* through an encoding (such as UTF-8)
- then this encoded data is written to the file
- the text in your program is represented by encoded bytes on disk

PYTHON 2 - open

- files opened with `open` will always be in binary mode
- using `'wb'` or `'rb'` as the mode is superfluous (but good for documenting intent!)
- as with most py2 apis, `unicode` may be implicitly converted by the `ASCII` codec

PYTHON 2 - open

```
>>> with open('f.txt', 'w') as f:
...     f.write(b'hello world!')
...     f.write(u'unicode ascii text')
...
>>> with open('f.txt') as f:
...     print(type(f.read()) is bytes)
...
True
>>> with open('f.txt', 'w') as f:
...     f.write(u'☃')
...
UnicodeEncodeError: 'ascii' codec can't encode character u'\u2603' in position
```

PYTHON 2 - `open`

- you *can* convert these into pseudo-text io objects with the C api
- `PyFile_SetEncoding` / `PyFile_SetEncodingAndErrors`
- let's look at some special streams which do just that!

PYTHON 2 - `stdout` / `stderr` / `print`

- `stdout` / `stderr` are just `file` objects too
- on interpreter startup, `PyFile_SetEncodingAndErrors` is called
- **but** it is only called when the streams are `tty`s

PYTHON 2 - `stdout` / `stderr` / `print`

- if environment may vary *or*
- if you might not have a tty
  - the only reliable way to use `stdout` / `stderr` / `print` is with bytes

PYTHON 2 - `cStringIO` / `StringIO`

- `cStringIO.StringIO` - a binary stream (similar to python2 open)
- `StringIO.StringIO`
  - allows mixed mode if the `ASCII` encoding can convert
  - produces text if any text inputs
  - produces bytes otherwise
  - (much slower than `cStringIO` as it is implemented in pure python)

PYTHON 3 - `open`

- just going to look at the `io` module
- `open` in python3 is just `io.open`
- `io.open(..., 'rb')` or `io.open(..., 'wb')` produce a binary io object
  - this object *requires* bytes
- `io.open` otherwise returns an `io.TextIOWrapper`
  - this object *requires* text

PYTHON 3 - `open`

```
>>> with open('f', 'wb') as f:
...     f.write('hi')
...
Traceback (most recent call last):
  File "<stdin>", line 2, in <module>
TypeError: a bytes-like object is required, not 'str'
```

PYTHON 3 - `open`

- The `encoding=` keyword argument may be passed to change what encoding the `TextIOWrapper` uses to write
- If `encoding=` is not passed, the encoding is determined using `locale.getpreferredencoding()`
  - This *usually* means looking at the `LANG` environment variable

PYTHON 3 - `print`

- `print` in python3 will write as if writing to a text stream
- Will convert by calling `str()` on arguments if necessary

```
>>> print(b'foo')
b'foo'
>>> print('hi')
hi
```

PY2 + PY3 - WRITING TEXT STDIO

- `io.TextIOWrapper` does not work with the stdio streams
- use `codecs.getwriter` instead!

```
if PY2:
    stdout_text = codecs.getwriter(locale.getprefferedencoding())(sys.stdout)
else:
    stdout_text = sys.stdout

stdout_text.write('🐍\n')
print('🐍', file=stdout_text)
```

PY2 + PY3 - WRITING IN CONSTRAINED ENVIRONMENT

- earlier showed that io encoding is subject to `LANG`

```
if PY2:
    stdout_text = codecs.getwriter('UTF-8')(sys.stdout)
else:
    stdout_text = io.TextIOWrapper(sys.stdout.buffer, encoding='UTF-8')

stdout_text.write('🐍\n')
print('🐍', file=stdout_text)
```

asottile

From <https://github.com/anthonywritescode/episodes-py3/blob/main/04-file-io/slides.md>

Credits -