Unicorn Problems

Mane mange and horn homeopathy

Unicode Problems

The curse of the UnicodeDecodeError

A String is not what you think it is

* A string is a finite sequence of symbols
* The set of its symbols is the string's alphabet
* These symbols can be anything

There are two kinds of strings we care about

* Byte strings:

  containing numbers from 0 - 255

* Unicode Strings

  containing characters from the 109,000 in the Unicode set

Transformation

* You can transform strings between these two types using an encoding

* `ascii` is an encoding

* it is not Unicode

* utf8 is an encoding

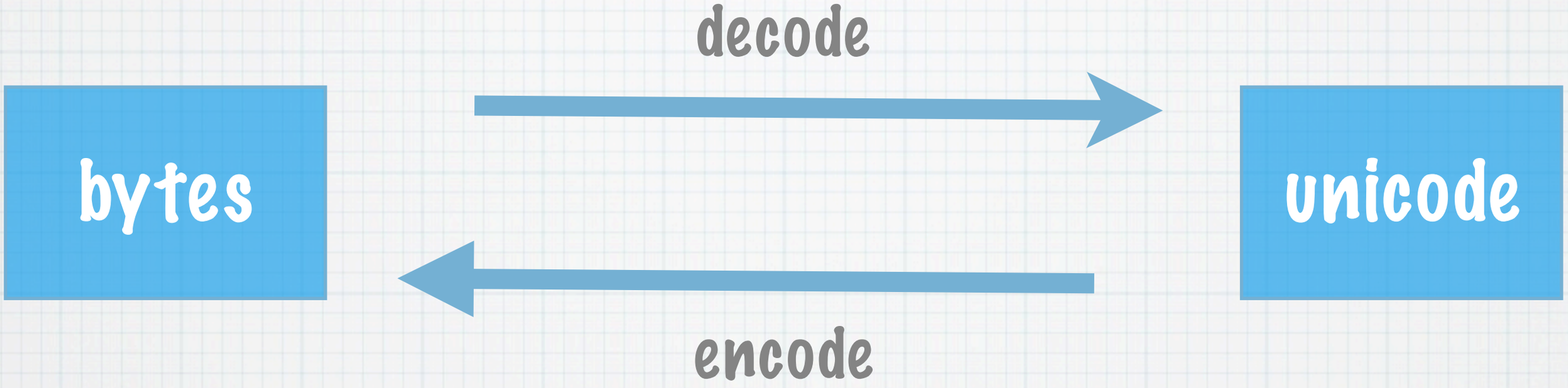
* it is also not Unicode

Following the rules leads to success

```
byte_str = 'this is a string of bytes'
```

```
unicode_str = byte_str.decode('utf8')

byte_from_uni_str = \
    unicode_str.encode('utf8')
```



`unicode` is a Python 2 type for unicode strings.
It is best to assume that its internal representation is PURE MAGIC.

Other operations are implicit and thus bad

```
byte_str = 'this is a string of bytes'
unicode_str = unicode(byte_str)
```

≈

```
unicode_str = byte_str.decode('ascii')
```

Concatenating bytes and unicode is insane

```
unicode_str + byte_str
```

≈

```
unicode_str \
   + byte_str.decode('ascii')
```

Python 2 allows impossible operations

* calling decode on a unicode object

I don't even know...

```
# a string with a utf8 non-breaking space
byte_str = 'lol\xc2\xa0wat'

# okay
unicode_str = byte_str.decode('utf8')

# LOL WTF?!?!
unicode_str = unicode_str.decode('utf8')
```

UnicodeEncodeError: 'ascii' codec can't encode character u'\xa0' in position 3: ordinal not in range(128)

Python 3 fixes all

* strings are unicode by default:

```
s = “this is unicode”
```

* byte strings are explicit:

```
b = b”byte string”
```

Python 3 fixes all

* you cannot implicitly concatenate unicode and bytes

```
s + b  # Raises an exception!
```

Python 3 fixes all

* You cannot call encode on a byte string
* You cannot call decode on a unicode string

We gotta do it ourselves

* We have to make sure we transform strings explicitly, never implicit

Ground rules

* Always declare strings as unicode:
  * `s = u"my string"`
* sqlalchemy always loads/stores unicode
* Genshi intelligently encodes unicode at the end

Ground rules

* Put non-ascii characters in your tests to uncover implicit evil
  * At top of file

```
# -*- coding: utf8 -*-
```

  * In your tests:

```
u' † ünicode røckß! † '
```

Ground rules

* Fear third-party libraries
* Determine if they return unicode, bytes, or unknown
* Carefully decode byte strings where appropriate