



Python

Text



Copyright © Software Carpentry 2010

This work is licensed under the Creative Commons Attribution License

See <http://software-carpentry.org/license.html> for more information.

How to represent characters?

How to represent characters?

American English in the 1960s:

How to represent characters?

American English in the 1960s:

26 characters × {upper, lower}

How to represent characters?

American English in the 1960s:

26 characters × {upper, lower}

+ 10 digits

How to represent characters?

American English in the 1960s:

- 26 characters × {upper, lower}
- + 10 digits
- + punctuation

How to represent characters?

American English in the 1960s:

- 26 characters × {upper, lower}
- + 10 digits
- + punctuation
- + special characters for controlling teletypes
(new line, carriage return, form feed, bell, ...)

How to represent characters?

American English in the 1960s:

- 26 characters × {upper, lower}
- + 10 digits
- + punctuation
- + special characters for controlling teletypes
(new line, carriage return, form feed, bell, ...)
- = 7 bits per character (ASCII standard)

How to represent text?

How to represent text?

1. Fixed-width records

How to represent text?

1. Fixed-width records

*A crash reduces
your expensive computer
to a simple stone.*

How to represent text?

1. Fixed-width records

*A crash reduces
your expensive computer
to a simple stone.*

A		c	r	a	s	h		r	e	d	u	c	e	s
y	o	u	r		e	x	p	e	n	s	i	v	e		c	o	m	p	u	t	e	r
t	o		a		s	i	m	p	l	e		s	t	o	n	e

How to represent text?

1. Fixed-width records

*A crash reduces
your expensive computer
to a simple stone.*

A		c	r	a	s	h		r	e	d	u	c	e	s
y	o	u	r		e	x	p	e	n	s	i	v	e		c	o	m	p	u	t	e	r
t	o		a		s	i	m	p	l	e		s	t	o	n	e

Easy to get to line N

How to represent text?

1. Fixed-width records

*A crash reduces
your expensive computer
to a simple stone.*

A		c	r	a	s	h		r	e	d	u	c	e	s
y	o	u	r		e	x	p	e	n	s	i	v	e		c	o	m	p	u	t	e	r
t	o		a		s	i	m	p	l	e		s	t	o	n	e

Easy to get to line N

But may waste space

How to represent text?

1. Fixed-width records

*A crash reduces
your expensive computer
to a simple stone.*

A		c	r	a	s	h		r	e	d	u	c	e	s
y	o	u	r		e	x	p	e	n	s	i	v	e		c	o	m	p	u	t	e	r
t	o		a		s	i	m	p	l	e		s	t	o	n	e

Easy to get to line N

But may waste space

What if lines are longer than the record length?

How to represent text?

1. Fixed-width records
2. Stream with embedded end-of-line markers

How to represent text?

1. Fixed-width records
2. Stream with embedded end-of-line markers

*A crash reduces
your expensive computer
to a simple stone.*

A		c	r	a	s	h		r	e	d	u	c	e	s		y	o	u	r		e	x	p	e	n	s	i	v
e		c	o	m	p	u	t	e	r		t	o		a		s	i	m	p	l	e		s	t	o	n	e	.

How to represent text?

- 1. Fixed-width records
- 2. Stream with embedded end-of-line markers

*A crash reduces
your expensive computer
to a simple stone.*

A		c	r	a	s	h		r	e	d	u	c	e	s		y	o	u	r		e	x	p	e	n	s	i	v
e		c	o	m	p	u	t	e	r		t	o		a		s	i	m	p	l	e		s	t	o	n	e	.

More flexible

How to represent text?

1. Fixed-width records
2. Stream with embedded end-of-line markers

*A crash reduces
your expensive computer
to a simple stone.*

A		c	r	a	s	h		r	e	d	u	c	e	s		y	o	u	r		e	x	p	e	n	s	i	v
e		c	o	m	p	u	t	e	r		t	o		a		s	i	m	p	l	e		s	t	o	n	e	.

More flexible

Wastes less space

How to represent text?

- 1. Fixed-width records
- 2. Stream with embedded end-of-line markers

*A crash reduces
your expensive computer
to a simple stone.*

A		c	r	a	s	h		r	e	d	u	c	e	s		y	o	u	r		e	x	p	e	n	s	i	v
e		c	o	m	p	u	t	e	r		t	o		a		s	i	m	p	l	e		s	t	o	n	e	.

More flexible

Skipping ahead is harder

Wastes less space

How to represent text?

- 1. Fixed-width records
- 2. Stream with embedded end-of-line markers

*A crash reduces
your expensive computer
to a simple stone.*

A		c	r	a	s	h		r	e	d	u	c	e	s		y	o	u	r		e	x	p	e	n	s	i	v
e		c	o	m	p	u	t	e	r		t	o		a		s	i	m	p	l	e		s	t	o	n	e	.

More flexible

Skipping ahead is harder

Wastes less space

What to use for end of line?

Unix: newline ('\n')

Unix: newline ('\n')

Windows: carriage return + newline ('\r\n')

Unix: newline ('\n')

Windows: carriage return + newline ('\r\n')

Oh dear...

Unix: newline ('\n')

Windows: carriage return + newline ('\r\n')

Oh dear...

Python converts '\r\n' to '\n' and back on Windows

Unix: newline ('\n')

Windows: carriage return + newline ('\r\n')

Oh dear...

Python converts '\r\n' to '\n' and back on Windows

To prevent this (e.g., when reading image files)

open the file in *binary mode*

Unix: newline ('\n')

Windows: carriage return + newline ('\r\n')

Oh dear...

Python converts '\r\n' to '\n' and back on Windows

To prevent this (e.g., when reading image files)

open the file in *binary mode*

```
reader = open('mydata.dat', 'rb')
```

Back to characters...

Back to characters...

How to represent ě, ß, Я, ...?

Back to characters...

How to represent ě, ß, Я, ...?

7 bits = 0...127

Back to characters...

How to represent ě, ß, Я, ...?

7 bits = 0...127

8 bits (a byte) = 0...255

Back to characters...

How to represent ě, ß, Я, ...?

7 bits = 0...127

8 bits (a byte) = 0...255

Different companies/countries defined different meanings for 128...255

Back to characters...

How to represent ě, ß, Я, ...?

7 bits = 0...127

8 bits (a byte) = 0...255

Different companies/countries defined different meanings for 128...255

Did not play nicely together

Back to characters...

How to represent ě, ß, Я, ...?

7 bits = 0...127

8 bits (a byte) = 0...255

Different companies/countries defined different meanings for 128...255

Did not play nicely together

And East Asian "characters" won't fit in 8 bits

1990s: Unicode standard

1990s: Unicode standard

Defines mapping from characters to integers

1990s: Unicode standard

Defines mapping from characters to integers

Does *not* specify how to store those integers

1990s: Unicode standard

Defines mapping from characters to integers

Does *not* specify how to store those integers

32 bits per character will do it...

1990s: Unicode standard

Defines mapping from characters to integers

Does *not* specify how to store those integers

32 bits per character will do it...

...but wastes a lot of space in common cases

1990s: Unicode standard

Defines mapping from characters to integers

Does *not* specify how to store those integers

32 bits per character will do it...

...but wastes a lot of space in common cases

Use in memory (for speed)

1990s: Unicode standard

Defines mapping from characters to integers

Does *not* specify how to store those integers

32 bits per character will do it...

...but wastes a lot of space in common cases

Use in memory (for speed)

Use something else on disk and over the wire

(Almost) everyone uses a *variable-length encoding* called UTF-8 instead

(Almost) everyone uses a *variable-length encoding* called UTF-8 instead

First 128 characters (old ASCII) stored in 1 byte each

(Almost) everyone uses a *variable-length encoding* called UTF-8 instead

First 128 characters (old ASCII) stored in 1 byte each

Next 1920 stored in 2 bytes, etc.

(Almost) everyone uses a *variable-length encoding* called UTF-8 instead

First 128 characters (old ASCII) stored in 1 byte each

Next 1920 stored in 2 bytes, etc.

0xxxxxxx | 7 bits

(Almost) everyone uses a *variable-length encoding* called UTF-8 instead

First 128 characters (old ASCII) stored in 1 byte each

Next 1920 stored in 2 bytes, etc.

	0xxxxxxx	7 bits
110yyyyy	10xxxxxx	11 bits

(Almost) everyone uses a *variable-length encoding* called UTF-8 instead

First 128 characters (old ASCII) stored in 1 byte each

Next 1920 stored in 2 bytes, etc.

		0xxxxxxx	7 bits
	110yyyyy	10xxxxxx	11 bits
1110zzzz	10yyyyyy	10xxxxxx	16 bits

(Almost) everyone uses a *variable-length encoding* called UTF-8 instead

First 128 characters (old ASCII) stored in 1 byte each

Next 1920 stored in 2 bytes, etc.

			0xxxxxxx	7 bits
		110yyyyy	10xxxxxx	11 bits
	1110zzzz	10yyyyyy	10xxxxxx	16 bits
11110www	10zzzzzz	10yyyyyy	10xxxxxx	21 bits

(Almost) everyone uses a *variable-length encoding* called UTF-8 instead

First 128 characters (old ASCII) stored in 1 byte each

Next 1920 stored in 2 bytes, etc.

			0xxxxxxx	7 bits
		110yyyyy	10xxxxxx	11 bits
	1110zzzz	10yyyyyy	10xxxxxx	16 bits
11110www	10zzzzzz	10yyyyyy	10xxxxxx	21 bits

The good news is, you don't need to know

Python 2.* provides two kinds of string

Python 2.* provides two kinds of string
Classic: one byte per character

Python 2.* provides two kinds of string

Classic: one byte per character

Unicode: "big enough" per character

Python 2.* provides two kinds of string

Classic: one byte per character

Unicode: "big enough" per character

Write `u'the string'` for Unicode

Python 2.* provides two kinds of string

Classic: one byte per character

Unicode: "big enough" per character

Write `u'the string'` for Unicode

Must specify *encoding* when converting from

Unicode to bytes

Python 2.* provides two kinds of string

Classic: one byte per character

Unicode: "big enough" per character

Write `u'the string'` for Unicode

Must specify *encoding* when converting from

Unicode to bytes

Use UTF-8



created by

Greg Wilson

October 2010



Copyright © Software Carpentry 2010

This work is licensed under the Creative Commons Attribution License

See <http://software-carpentry.org/license.html> for more information.