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 $\times \{\text{upper, lower}\}$
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.

<table>
<thead>
<tr>
<th>A crash reduces</th>
</tr>
</thead>
<tbody>
<tr>
<td>your expensive computer</td>
</tr>
<tr>
<td>to a simple stone.</td>
</tr>
</tbody>
</table>
How to represent text?

1. Fixed-width records

\[
\text{A crash reduces}
\]
\[
\text{your expensive computer}
\]
\[
\text{to a simple stone.}
\]

\[
\begin{array}{cccccccc}
\text{A} & \text{c} & \text{r} & \text{a} & \text{sh} & \text{ } & \text{r} & \text{e} \text{d} \text{u} \text{c} \text{e} \text{s} \\
\text{y} & \text{o} & \text{u} & \text{r} & \text{ } & \text{e} \text{x} \text{p} \text{e} \text{n} \text{s} \text{i} \text{v} \text{e} & \text{c} \text{o} \text{m} \text{p} \text{u} \text{t} \text{e} \text{r} \\
\text{t} & \text{o} & \text{a} & \text{s} \text{i} \text{m} \text{p} \text{l} \text{e} & \text{s} \text{t} \text{o} \text{n} \text{e} \\
\end{array}
\]

Easy to get to line N
How to represent text?

1. Fixed-width records

A crash reduces your expensive computer to a simple stone.

<table>
<thead>
<tr>
<th>A</th>
<th>c r a s h</th>
<th>r e d u c e s</th>
<th>· · · · · · · ·</th>
</tr>
</thead>
<tbody>
<tr>
<td>y o u r</td>
<td>e x p e n s i v e</td>
<td>c o m p u t e r</td>
<td></td>
</tr>
<tr>
<td>t o</td>
<td>a</td>
<td>s i m p l e</td>
<td>s t o n e. · · · ·</td>
</tr>
</tbody>
</table>
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 | · | · | · | · | · | · | · | · | · | · | · |
| r | e | d | u | c | e | s |
| 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 | · | · | · | · | · | · | ·|

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.
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.

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.

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.

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.

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 ė, β, Я, ...?
Back to characters...
How to represent ě, β, Я, ...?
7 bits = 0...127
Back to characters...

How to represent ě, β, Я, ...?

7 bits = 0...127

8 bits (a byte) = 0...255
Back to characters...

How to represent ė, β, 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, β, 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, β, 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.

<table>
<thead>
<tr>
<th>0xxxxxxx</th>
<th>7 bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>110yyyyy</td>
<td>10xxxxxx</td>
</tr>
</tbody>
</table>

Python

Text
(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.

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Sequence</th>
<th>Number of Bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xxxxxxx</td>
<td>0xxxxxxx</td>
<td>7 bits</td>
</tr>
<tr>
<td>110yyyyyy</td>
<td>10xxxxxxx</td>
<td>11 bits</td>
</tr>
<tr>
<td>1110zzzzz</td>
<td>10yyyyyyy</td>
<td>16 bits</td>
</tr>
</tbody>
</table>
(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.

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>0xxxxxxx</td>
<td>7 bits</td>
</tr>
<tr>
<td>110yyyyy</td>
<td>10xxxxxx</td>
<td>11 bits</td>
<td></td>
</tr>
<tr>
<td>1110zzzz</td>
<td>10yyyyyy</td>
<td>16 bits</td>
<td></td>
</tr>
<tr>
<td>11110www</td>
<td>10zzzzzz</td>
<td>21 bits</td>
<td></td>
</tr>
</tbody>
</table>
(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.

<table>
<thead>
<tr>
<th>Bit Pattern</th>
<th>Description</th>
<th>Number of Bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xxxxxxx</td>
<td>7 bits</td>
<td></td>
</tr>
<tr>
<td>110yyyyy</td>
<td>11 bits</td>
<td></td>
</tr>
<tr>
<td>1110zzzz</td>
<td>16 bits</td>
<td></td>
</tr>
<tr>
<td>11110www</td>
<td>21 bits</td>
<td></td>
</tr>
</tbody>
</table>

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*