# Unicode and Python

Glyphs, Encodings and all that

# Unicode is a mapping of code points to glyphs

Unicode is a mapping of code points to glyphs

A “code point” is the number of a glyph

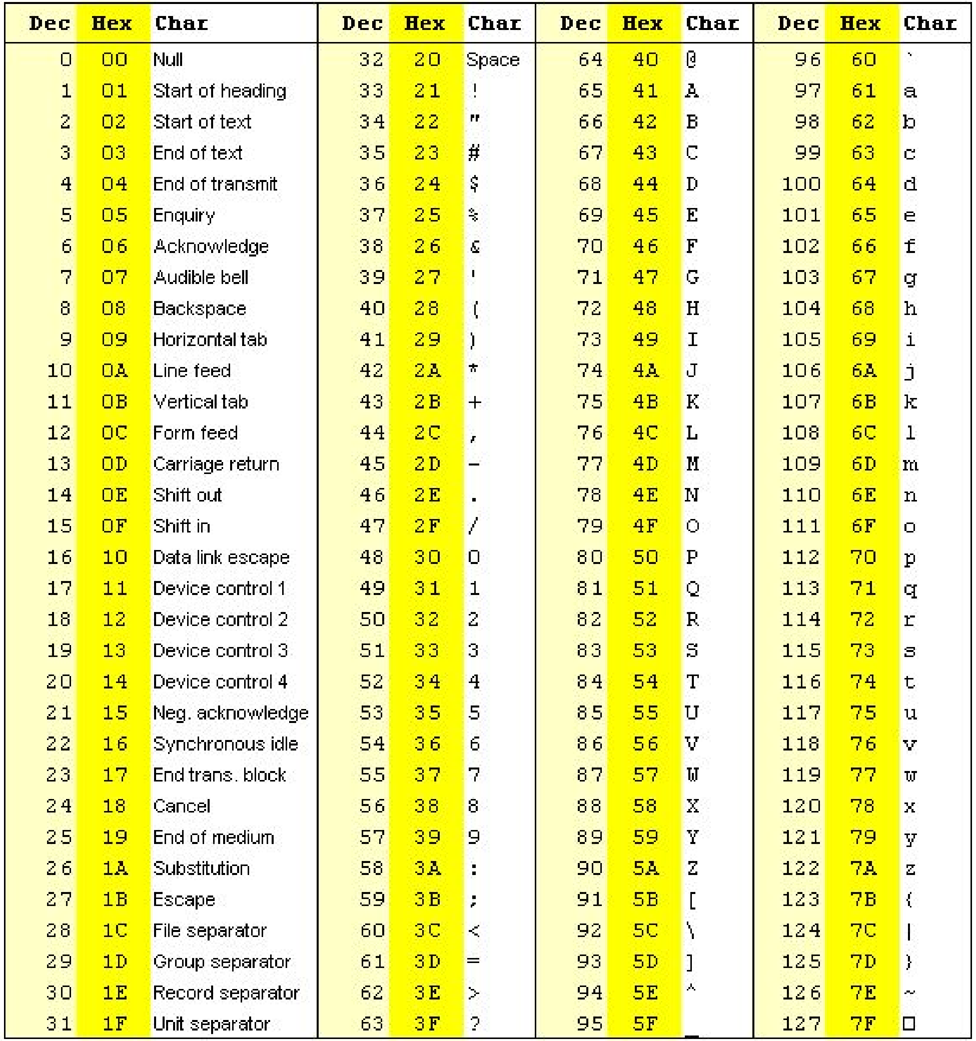
Unicode is a mapping of code points to glyphs

A “code point” is the number of a glyph

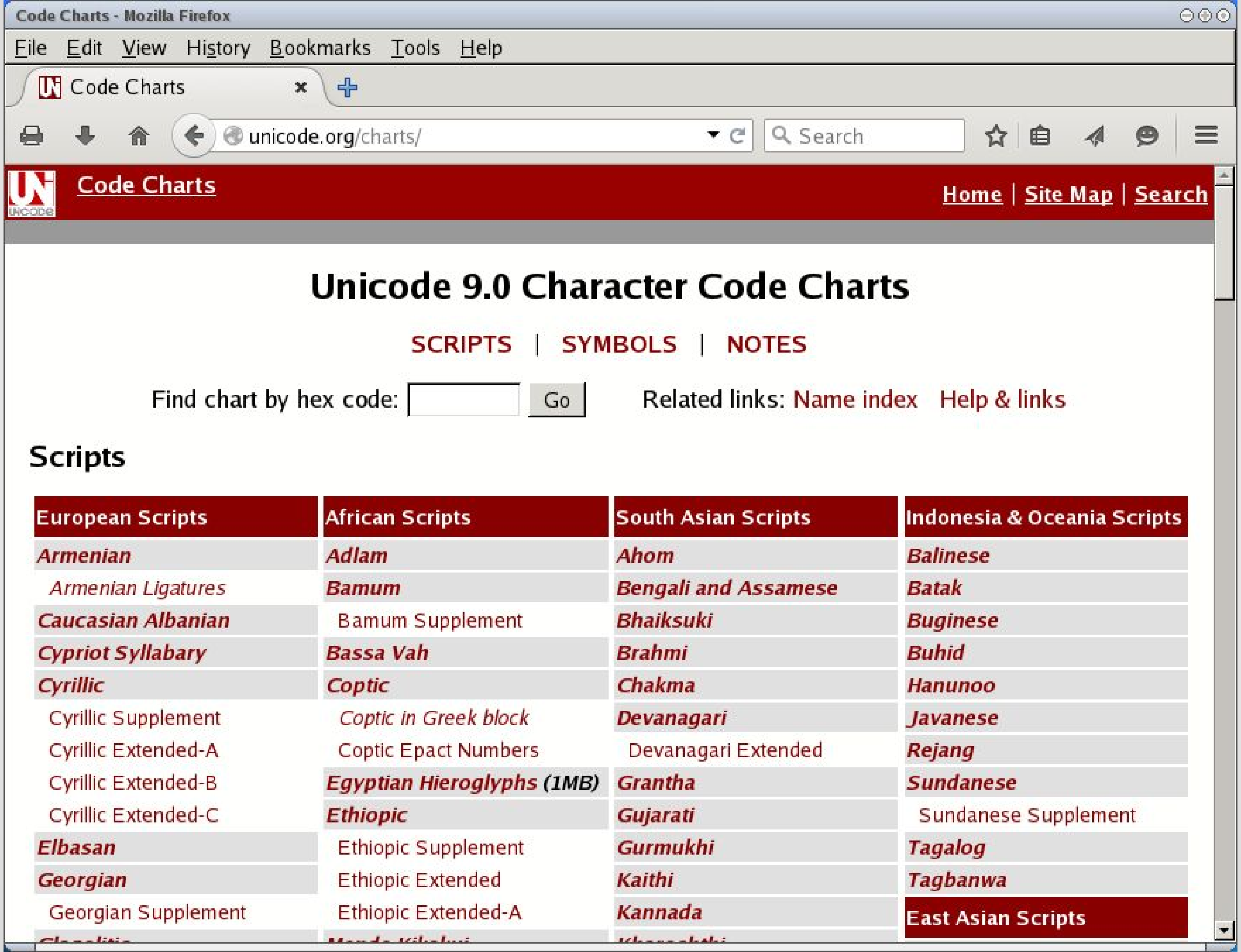
Code points written as “U+<hex digits>”
So U+41 = “A”

# American Standard Code for Information Interchange

Contains “control characters” and Glyphs



| Dec | Hex | Char | Dec | Hex | Char | Dec | Hex | Char | Dec | Hex | Char |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 00 | Null | 32 | 20 | Space | 64 | 40 | @ | 96 | 60 | ` |
| 1 | 01 | Start of heading | 33 | 21 | ! | 65 | 41 | A | 97 | 61 | a |
| 2 | 02 | Start of text | 34 | 22 | " | 66 | 42 | B | 98 | 62 | b |
| 3 | 03 | End of text | 35 | 23 | # | 67 | 43 | C | 99 | 63 | c |
| 4 | 04 | End of transmit | 36 | 24 | $ | 68 | 44 | D | 100 | 64 | d |
| 5 | 05 | Enquiry | 37 | 25 | % | 69 | 45 | E | 101 | 65 | e |
| 6 | 06 | Acknowledge | 38 | 26 | & | 70 | 46 | F | 102 | 66 | f |
| 7 | 07 | Audible bell | 39 | 27 | ' | 71 | 47 | G | 103 | 67 | g |
| 8 | 08 | Backspace | 40 | 28 | ( | 72 | 48 | H | 104 | 68 | h |
| 9 | 09 | Horizontal tab | 41 | 29 | ) | 73 | 49 | I | 105 | 69 | i |
| 10 | 0A | Line feed | 42 | 2A | * | 74 | 4A | J | 106 | 6A | j |
| 11 | 0B | Vertical tab | 43 | 2B | + | 75 | 4B | K | 107 | 6B | k |
| 12 | 0C | Form feed | 44 | 2C | , | 76 | 4C | L | 108 | 6C | l |
| 13 | 0D | Carriage return | 45 | 2D | - | 77 | 4D | M | 109 | 6D | m |
| 14 | 0E | Shift out | 46 | 2E | . | 78 | 4E | N | 110 | 6E | n |
| 15 | 0F | Shift in | 47 | 2F | / | 79 | 4F | O | 111 | 6F | o |
| 16 | 10 | Data link escape | 48 | 30 | 0 | 80 | 50 | P | 112 | 70 | p |
| 17 | 11 | Device control 1 | 49 | 31 | 1 | 81 | 51 | Q | 113 | 71 | q |
| 18 | 12 | Device control 2 | 50 | 32 | 2 | 82 | 52 | R | 114 | 72 | r |
| 19 | 13 | Device control 3 | 51 | 33 | 3 | 83 | 53 | S | 115 | 73 | s |
| 20 | 14 | Device control 4 | 52 | 34 | 4 | 84 | 54 | T | 116 | 74 | t |
| 21 | 15 | Neg. acknowledge | 53 | 35 | 5 | 85 | 55 | U | 117 | 75 | u |
| 22 | 16 | Synchronous idle | 54 | 36 | 6 | 86 | 56 | V | 118 | 76 | v |
| 23 | 17 | End trans. block | 55 | 37 | 7 | 87 | 57 | W | 119 | 77 | w |
| 24 | 18 | Cancel | 56 | 38 | 8 | 88 | 58 | X | 120 | 78 | x |
| 25 | 19 | End of medium | 57 | 39 | 9 | 89 | 59 | Y | 121 | 79 | y |
| 26 | 1A | Substitution | 58 | 3A | : | 90 | 5A | Z | 122 | 7A | z |
| 27 | 1B | Escape | 59 | 3B | ; | 91 | 5B | [ | 123 | 7B | { |
| 28 | 1C | File separator | 60 | 3C | < | 92 | 5C | \ | 124 | 7C | \| |
| 29 | 1D | Group separator | 61 | 3D | = | 93 | 5D | ] | 125 | 7D | } |
| 30 | 1E | Record separator | 62 | 3E | > | 94 | 5E | ^ | 126 | 7E | ~ |
| 31 | 1F | Unit separator | 63 | 3F | ? | 95 | 5F | _ | 127 | 7F | □ |


Code Charts - Mozilla Firefox
File Edit View History Bookmarks Tools Help
Code Charts
unicode.org/charts/
Search
Code Charts
Home | Site Map | Search

# Unicode 9.0 Character Code Charts

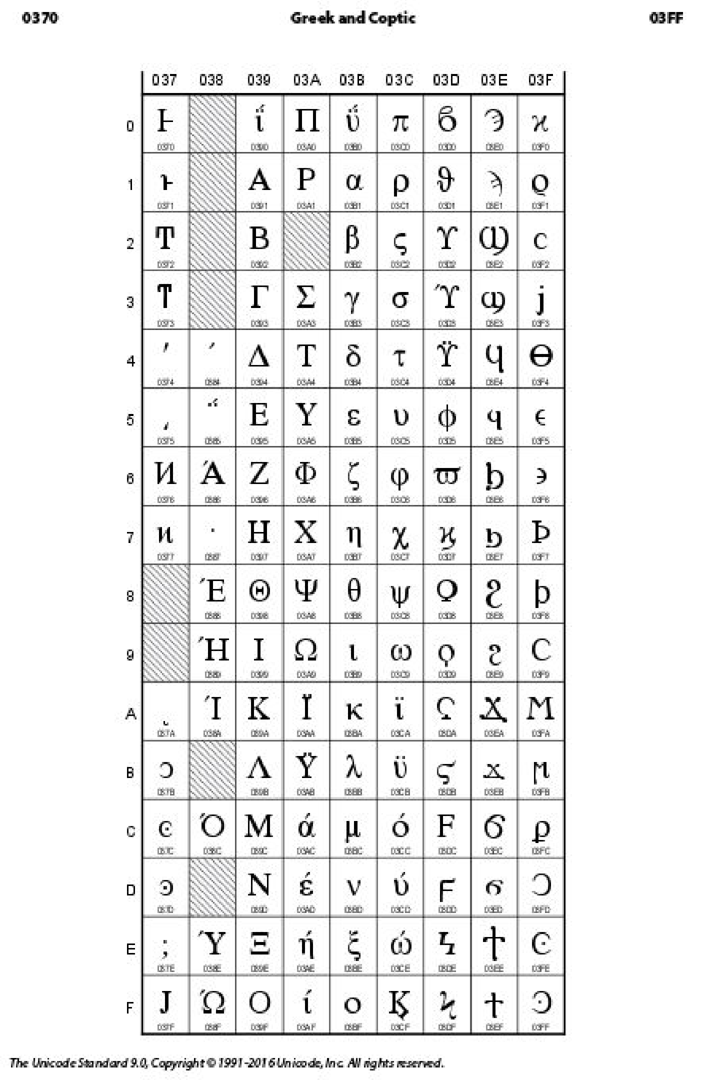
SCRIPTS | SYMBOLS | NOTES

Find chart by hex code: Go    Related links: Name index  Help & links

## Scripts

| European Scripts | African Scripts | South Asian Scripts | Indonesia & Oceania Scripts |
|---|---|---|---|
| ***Armenian*** | ***Adlam*** | ***Ahom*** | ***Balinese*** |
| *Armenian Ligatures* | ***Bamum*** | ***Bengali and Assamese*** | ***Batak*** |
| ***Caucasian Albanian*** | Bamum Supplement | ***Bhaiksuki*** | ***Buginese*** |
| ***Cypriot Syllabary*** | ***Bassa Vah*** | ***Brahmi*** | ***Buhid*** |
| ***Cyrillic*** | ***Coptic*** | ***Chakma*** | ***Hanunoo*** |
| Cyrillic Supplement | *Coptic in Greek block* | ***Devanagari*** | ***Javanese*** |
| Cyrillic Extended-A | Coptic Epact Numbers | Devanagari Extended | ***Rejang*** |
| Cyrillic Extended-B | ***Egyptian Hieroglyphs*** *(1MB)* | ***Grantha*** | ***Sundanese*** |
| Cyrillic Extended-C | ***Ethiopic*** | ***Gujarati*** | Sundanese Supplement |
| ***Elbasan*** | Ethiopic Supplement | ***Gurmukhi*** | ***Tagalog*** |
| ***Georgian*** | Ethiopic Extended | ***Kaithi*** | ***Tagbanwa*** |
| Georgian Supplement | Ethiopic Extended-A | ***Kannada*** | **East Asian Scripts** |

# Unicode Greek and Coptic



0370 **Greek and Coptic** 03FF

| | 037 | 038 | 039 | 03A | 03B | 03C | 03D | 03E | 03F |
|---|---|---|---|---|---|---|---|---|---|
| 0 | Ͱ<br>0370 | | ΐ<br>0390 | Π<br>03A0 | ΰ<br>03B0 | π<br>03C0 | ϐ<br>03D0 | Ϡ<br>03E0 | ϰ<br>03F0 |
| 1 | ͱ<br>0371 | | Α<br>0391 | Ρ<br>03A1 | α<br>03B1 | ρ<br>03C1 | ϑ<br>03D1 | ϡ<br>03E1 | ϱ<br>03F1 |
| 2 | Ͳ<br>0372 | | Β<br>0392 | | β<br>03B2 | ς<br>03C2 | ϒ<br>03D2 | Ϣ<br>03E2 | ϲ<br>03F2 |
| 3 | ͳ<br>0373 | | Γ<br>0393 | Σ<br>03A3 | γ<br>03B3 | σ<br>03C3 | ϓ<br>03D3 | ϣ<br>03E3 | ϳ<br>03F3 |
| 4 | ʹ<br>0374 | ΄<br>0384 | Δ<br>0394 | Τ<br>03A4 | δ<br>03B4 | τ<br>03C4 | ϔ<br>03D4 | Ϥ<br>03E4 | ϴ<br>03F4 |
| 5 | ͵<br>0375 | ΅<br>0385 | Ε<br>0395 | Υ<br>03A5 | ε<br>03B5 | υ<br>03C5 | ϕ<br>03D5 | ϥ<br>03E5 | ϵ<br>03F5 |
| 6 | Ͷ<br>0376 | Ά<br>0386 | Ζ<br>0396 | Φ<br>03A6 | ζ<br>03B6 | φ<br>03C6 | ϖ<br>03D6 | Ϧ<br>03E6 | ϶<br>03F6 |
| 7 | ͷ<br>0377 | ·<br>0387 | Η<br>0397 | Χ<br>03A7 | η<br>03B7 | χ<br>03C7 | ϗ<br>03D7 | ϧ<br>03E7 | Ϸ<br>03F7 |
| 8 | | Έ<br>0388 | Θ<br>0398 | Ψ<br>03A8 | θ<br>03B8 | ψ<br>03C8 | Ϙ<br>03D8 | Ϩ<br>03E8 | ϸ<br>03F8 |
| 9 | | Ή<br>0389 | Ι<br>0399 | Ω<br>03A9 | ι<br>03B9 | ω<br>03C9 | ϙ<br>03D9 | ϩ<br>03E9 | Ϲ<br>03F9 |
| A | ͺ<br>037A | Ί<br>038A | Κ<br>039A | Ϊ<br>03AA | κ<br>03BA | ϊ<br>03CA | Ϛ<br>03DA | Ϫ<br>03EA | Ϻ<br>03FA |
| B | ͻ<br>037B | | Λ<br>039B | Ϋ<br>03AB | λ<br>03BB | ϋ<br>03CB | ϛ<br>03DB | ϫ<br>03EB | ϻ<br>03FB |
| C | ͼ<br>037C | Ό<br>038C | Μ<br>039C | ά<br>03AC | μ<br>03BC | ό<br>03CC | Ϝ<br>03DC | Ϭ<br>03EC | ϼ<br>03FC |
| D | ͽ<br>037D | | Ν<br>039D | έ<br>03AD | ν<br>03BD | ύ<br>03CD | ϝ<br>03DD | ϭ<br>03ED | Ͻ<br>03FD |
| E | ;<br>037E | Ύ<br>038E | Ξ<br>039E | ή<br>03AE | ξ<br>03BE | ώ<br>03CE | Ϟ<br>03DE | Ϯ<br>03EE | Ͼ<br>03FE |
| F | Ϳ<br>037F | Ώ<br>038F | Ο<br>039F | ί<br>03AF | ο<br>03BF | Ϗ<br>03CF | ϟ<br>03DF | ϯ<br>03EF | Ͽ<br>03FF |

Now we know how to find code points.

The unicode ‘Latin-1’ code chart is ASCII, without the control or extended characters.

The problem (and most of the rest of the talk) is about “encoding”

The problem (and most of the rest of the talk) is about “encoding”

“Encoding” a one-byte is easy:

Each byte is a code point.

The problem (and most of the rest of the talk) is about “encoding”

“Encoding” a one-byte is easy:

Each byte is a code point.

This is the ASCII encoding

How to handle code points with more than eight bits?

Consider the capital Greek letter Delta Δ (0x0394)

# UTF-16 Encoding:  Two bytes per code point

# UTF-16 Encoding: Two bytes per code point

Two possibilities for 0x0394

| | | |
|---|---|---|
| UTF-16LE: | 0x94 0x03 | Little endian |
| UTF-16BE: | 0x03 0x94 | Big endian |

# How to tell which version you have?

# How to tell which version you have?

The “Byte Order Mark” (BOM)    U+FEFF

UTF-16LE strings start with    0xFF 0xFE
UTF-16BE strings start with    0xFE 0xFF

# UTF-8: A variable-length encoding

First rule: All printable ASCII characters are UTF-8 encoded

Second rule: High bits signal more bits to come

# How UTF-8 encodes glyphs

| First Byte | Second Byte | Third Byte | Fourth Byte | # of bits | Glyphs |
|---|---|---|---|---|---|
| 0xxxxxxx | | | | 7 | 0x7F = 127 |
| 110xxxxx | 10xxxxxx | | | 11 | 0x7FF = 2047 |
| 1110xxxx | 10xxxxxx | 10xxxxxx | | 16 | 0xFFFF = 65535 |
| 11110xxx | 10xxxxxx | 10xxxxxx | 10xxxxxx | 21 | 0x1FFFFF = 2,097,151 |

High bits of a start byte indicate how many bytes in glyph (maximum of six).

Glyph bits are extracted from the ‘x’ bits.

A byte of the form 10xx xxxx is always in the middle of a code sequence.

UTF-8 streams can be read backwards or starting from an arbitrary location

# Pros and Cons of different encodings

| | UTF-8 | UTF-16/32 |
|---|---|---|
| Memory usage | Compact | Wasteful |
| Processing speed | Slower | Faster |
| Self-synchronizing | Yes | No |
| Endian issues | No | Yes |
| Finding the nth glyph in a string | O(n) | O(1) |

# How does Unicode work with Python (2)?

# How does Unicode work with Python (2)?

Strings are sequences of 8-bit clean bytes.

# How does Unicode work with Python (2)?

Strings are sequences of 8-bit clean bytes.

Unicode strings are a subclass of the base string type.

# How does Unicode work with Python (2)?

Strings are sequences of 8-bit clean bytes.

Unicode strings are a subclass of the base string type.

Create unicode strings with unicode() built-in.  Takes string, returns unicode string.

# How does Unicode work with Python (2)?

Strings are sequences of 8-bit clean bytes.

Unicode strings are a subclass of the base string type.

Create unicode strings with unicode() built-in.  Takes string, returns unicode string.

Create unicode ‘characters’ with unichr() built-in.  Takes integer code point, returns a length-one unicode string.

# How does Unicode work with Python (2)?

Strings are sequences of 8-bit clean bytes.

Unicode strings are a subclass of the base string type.

Create unicode strings with unicode() built-in.  Takes string, returns unicode string.

Create unicode ‘characters’ with unichr() built-in.  Takes integer code point, returns a length-one unicode string.

Prefix constant string with ‘u’:    u’Make me Unicode!’

# How does Unicode work with Python (2)?

Strings are sequences of 8-bit clean bytes.

Unicode strings are a subclass of the base string type.

Create unicode strings with unicode() built-in. Takes string, returns unicode string.

Create unicode ‘characters’ with unichr() built-in. Takes integer code point, returns a length-one unicode string.

Prefix constant string with ‘u’:    u’Make me Unicode!’

Put a literal unicode code point in a string via \uxxxx.    ‘Delta = \u0394’

Unicode strings are like regular python strings.

Unicode strings can be:

- Added (concatenated)
- Multiplied (replicated)
- Sliced: var[10], var[3:7]
- ord() works: `hex(ord(unichr(0x394)))` -> ‘`0x394`’

Method functions like

.find(), .rfind(), .count(), .replace(), .startswith(), .endswidth(), .center(), .ljust(), .rjust(), .lstrip(), .strip(), .rstrip(), .split(), .join() work as usual.

These functions operate on glyphs, not characters.

# One quick question, how are Python Unicode strings encoded?

One quick question, how are Python Unicode strings encoded?

Answer:   It doesn’t matter!

One quick question, how are Python Unicode strings encoded?

Answer:   It doesn’t matter!

Encoding comes in when we want to do Input or Output.

Unicode strings are encoded, with the `.encode()` method.
Encoding returns a regular python string.

# Example:

```
>>> def hexstr(x):
...    return ‘ ‘.join([ ‘%02X’ % ord(c) for c in x ])
...
>>> f = ‘Delta = ‘ + unichr(0x394)
>>> hexstr(f.encode(‘UTF-8’))
‘44 65 6C 74 61 20 3D 20 CE 94’
>>>
```

# Example:

```
>>> def hexstr(x):
...     return ‘ ‘.join([ ‘%02X’ % ord(c) for c in x ])
...
>>> f = ‘Delta = ‘ + unichr(0x394)
>>> hexstr(f.encode(‘UTF-8’))
‘44 65 6C 74 61 20 3D 20 CE 94’
>>>
>>> hexstr(f.encode(‘UTF-16’))
‘FF FE 44 00 65 00 6C 00 74 00 61 00 20 00 3D 00 20 00 94 03’
>>>
```

# Example:

```
>>> def hexstr(x):
...    return ‘ ‘.join([ ‘%02X’ % ord(c) for c in x ])
...
>>> f = ‘Delta = ‘ + unichr(0x394)
>>> hexstr(f.encode(‘UTF-8’))
‘44 65 6C 74 61 20 3D 20 CE 94’
>>>
>>> hexstr(f.encode(‘UTF-16’))
‘FF FE 44 00 65 00 6C 00 74 00 61 00 20 00 3D 00 20 00 94 03’
>>>
>>> hexstr(f.encode(‘UTF-16BE’))
‘00 44 00 65 00 6C 00 74 00 61 00 20 00 3D 00 20 03 94’
>>>
```

We use encode to write output (from Python).

We use decode for input (to Python).

```
>>> line = open(‘file’, ‘r’).readline()
>>> type(line)
<type ‘str’>
>>> type(line.decode(‘UTF-8’))
<type ‘unicode’>
>>>
```

# A slicker way to read a Unicode encoded file is:

```
import codecs
f_obj = codecs.openfile(‘filename’, encoding=’UTF-16’)
for line in f_obj:
   print line.encode(‘UTF-8’)

f_obj.close()
```

This translates a UTF-16 file to a UTF-8 file.

Incidentally, there is an ‘ASCII’ encoding that does what it sounds like.
Peek inside the `codecs` module to see the encodings.

Python source files can be encoded in Unicode!

The encoding must be specified in a comment on the first or second line.

```
#!/usr/bin/env python
# coding: UTF-8

f = ‘Delta is Δ!‘
print f
```

# Questions?