A standard for the discrete representation of written text
The Big Picture

glyphs
- m, Φ, €, 好

characters
- Cyrillic ef, Euro sign, Tei chou ten
- Latin M, Apostrophe

code points
- U+006D, U+0444, U+2019, U+5975

binary encoding
- 6D, D1 84, E2 82 AC
- E2 80 99, E5 A5 BD
The Big Picture

glyphs

characters

code

points

binary encoding

m φ , € 好

Cyrillic ef Apostrophe Tei chou ten

Latin M

Apostrophe

U+006D U+0444 U+2019 U+5975

U+006D U+20AC

U+2019 U+0444

20AC U+5975

6D D1 84 E2 80 99 E5 A5 BD

E2 82 AC
Text: A Sequence of Glyphs

- **Glyph:** “An individual mark on a written medium that contributes to the meaning of what is written.”
  - See foyer floor in main library

- One *character* can have many *glyphs*
  - Example: Latin E can be e, e, e, e, e, e,…

- One *glyph* can be different *characters*
  - A is both (capital) Latin A and Greek Alpha

- One unit of text can consist of *multiple* *glyphs*
  - An accented letter (é) is two glyphs
  - The ligature of f+i (fi) is two glyphs
Glyphs vs Characters

glyphs

characters

Latin small E

Latin capital A

Greek capital alpha
Security Issue

- Visual homograph: Two different characters that look the same
  - Would you click here:  www.paypal.com  ?
Security Issue

- Visual homograph: Two different characters that look the same
  - Would you click here: www.paypal.com?
  - Oops! The second ‘a’ is actually CYRILLIC SMALL LETTER A
  - This site successfully registered in 2005

- “Solution”
  - Heuristics that warn users when languages are mixed and homographs are possible
Unicode Code Points

- Each character is assigned a unique code point

- A code point is defined by an integer value, and is also given a name
  - one hundred and nine (109, or 0x6d)
  - LATIN SMALL LETTER M

- Convention: Write code points as U+hex
  - Example: U+006D

- As of June 2015 (version 8.0):
  - Contains 120,000+ code points
  - Covers 129 scripts (and counting...)

The Ohio State University
Computer Science and Engineering
Unicode: Mapping to Code Points

glyphs

characters

code points

code

binary encoding

U+006D

U+0444

U+2019

U+20AC

U+0597

U+5A1C

' m

φ

€

好

Cyrillic ef

Euro sign

Tei chou ten

Latin M

Apostrophe

E2 82 AC

E5 A5 BD

D1 84

E2 80 99

E2 82 AC
Organization

- Code points are grouped into categories
  - Basic Latin, Cyrillic, Arabic, Cherokee, Currency, Mathematical Operators, ...
- Standard allows for $17 \times 2^{16}$ code points
  - 0 to 1,114,111 (i.e., > 1 million)
  - U+0000 to U+10FFFF
- Each group of $2^{16}$ called a plane
  - U+nnnnnnnn, same green ==> same plane
- Plane 0 called basic multilingual plane (BMP)
  - Has (practically) everything you could need
  - Convention: code points in BMP written U+nnnnn, others written with 5 or 6 hex digits
## Basic Multilingual Plane

The Basic Multilingual Plane is a part of the Unicode Standard that provides additional characters for scripts and symbols beyond the Basic Multilingual Plane. It includes a wide range of scripts and symbols from around the world, representing different cultural and linguistic backgrounds. The plane is divided into 16 blocks, each with a specific range of code points, allowing for the encoding of a vast array of characters for use in various applications.

### Grid Layout

The grid layout is color-coded to represent different scripts and symbols. Each cell in the grid corresponds to a specific code point, with the following categories:

- **Latin scripts and symbols**
- **Linguistic scripts**
- **Other European scripts**
- **Middle Eastern and Southwest Asian scripts**
- **African scripts**
- **South Asian scripts**
- **Southeast Asian scripts**
- **East Asian scripts**
- **Unified CJK Han**
- **Canadian Aboriginal scripts**
- **Symbols**
- **Diacritics**
- **UTF-16 surrogates and private use**
- **Miscellaneous characters**
- **Unallocated code points**

### Code Point Analysis

- **A0-AF**: Represent South Asian scripts.
- **B0-BF**: Represent Southeast Asian scripts.
- **C0-CF**: Represent East Asian scripts.
- **D0-DF**: Represent unified CJK Han characters.
- **E0-EF**: Represent African scripts.
- **F0-FF**: Represent miscellaneous characters.
UTF-8

- Encoding of code point (integer) in a sequence of bytes (octets)
  - Standard: all caps, with hyphen (UTF-8)
- Variable length
  - Some code points require 1 octet
  - Others require 2, 3, or 4
- Consequence: Can not infer number of characters from size of file!
- No endian-ness: just a sequence of octets
  D0 BF D1 80 D0 B8 D0 B2 D0 B5 D1 82...
UTF-8: Code Points & Octets

glyphs

characters

code points

binary encoding
UTF-8 Encoding Recipe

- 1-byte encodings
  - First bit is 0
  - Example: 0110 1101 (encodes U+006D)

- 2-byte encodings
  - First byte starts with 110...
  - Second byte starts with 10...
    - Example: 1101 0000 1011 1111
    - Payload: 1101 0000 1011 1111
      = 100 0011 1111
      = 0x043F
    - Code point: U+043F
      *i.e.* н, Cyrillic small letter pe
UTF-8 Encoding Recipe

- Generalization: An encoding of length k:
  - First byte starts with k 1’s, then 0
    - Example 1110 0110 ==> first byte of a 3-byte encoding
  - Subsequent k-1 bytes each start with 10
  - Remaining bits are payload

- Example:
  11100010 10000010 10101100
  - Payload: 0x20AC (i.e., U+20AC, €)

- Consequence: Stream is self-synchronizing
  - A dropped byte affects only that character
## UTF-8 Encoding Summary

<table>
<thead>
<tr>
<th>Unicode</th>
<th>Byte1</th>
<th>Byte2</th>
<th>Byte3</th>
<th>Byte4</th>
<th>example</th>
</tr>
</thead>
<tbody>
<tr>
<td>U+0000−U+007F</td>
<td>0xxxx</td>
<td></td>
<td></td>
<td></td>
<td>'§' U+0024</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>→ 00100100</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>→ 0x24</td>
</tr>
<tr>
<td>U+0080−U+07FF</td>
<td>110yy</td>
<td>10xxxx</td>
<td></td>
<td></td>
<td>'¢' U+00A2</td>
</tr>
<tr>
<td></td>
<td>yxxx</td>
<td></td>
<td></td>
<td></td>
<td>→ 11000010, 10100010</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>→ 0xC2, 0xA2</td>
</tr>
<tr>
<td>U+0800−U+FFFF</td>
<td>1110y</td>
<td>10yyyy</td>
<td>10xxxx</td>
<td></td>
<td>'€' U+20AC</td>
</tr>
<tr>
<td></td>
<td>yyyy</td>
<td></td>
<td></td>
<td></td>
<td>→ 11100010, 10000010, 10101100</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>→ 0xE2, 0x82, 0xAC</td>
</tr>
<tr>
<td>U+10000−U+10FFFF</td>
<td>11110zz</td>
<td>10zzyyyy</td>
<td>10yyyy</td>
<td>10xxxxx</td>
<td>'€' U+024B62</td>
</tr>
<tr>
<td></td>
<td>zzzz</td>
<td>yyyy</td>
<td>yyyy</td>
<td>yyyy</td>
<td>→ 11110000, 10100100, 10101101, 10100010</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>→ 0xF0, 0xA4, 0xAD, 0xA2</td>
</tr>
</tbody>
</table>

(from wikipedia)
Your Turn

□ For the following UTF-8 encoding, what is the corresponding code point(s)?
  ■ F0 A4 AD A2

□ For the following Unicode code point, what is its UTF-8 encoding?
  ■ U+20AC
Security Issue

- Not all octet sequence encodings
  - “overlong” encodings are illegal
  - example: C0 AF
    - \[1100\ 0000\ 1010\ 1111\]
    - = U+002F (should be encoded 2F)

- Classic security bug (IIS 2001)
  - Should reject URL requests with “../..”
    - Scanned for 2E 2E 2F 2E 2E (in encoding)
    - Accepted “..%c0%af..” (doesn’t contain x2F)
      - 2E 2E C0 AF 2E 2E
  - *After* accepting, server *then* decoded
    - 2E 2E C0 AF 2E 2E decoded into “../..”

- Moral: Work in “code point” space!
Recall: URL encoding

- Concrete invariant (convention)
  - No space, ;, :, & in representation
  - To represent these characters, use %hh instead (hh is ASCII code in hex)
    - %20 for space
  - Q: What about % in abstract value?

- Recall: correspondence relation
Other (Older) Encodings

- In the beginning...
- Character sets were small
  - ASCII: only 128 characters (ie 2^7)
  - 1 byte/character, leading bit always 0
- Globalization means more characters...
  - But 1 byte/character seems fundamental
- Solutions:
  - Use that leading bit!
    - Text data now looks just like binary data
  - Use more than 1 encoding!
    - Must specify data + encoding used
### ASCII: 128 Codes

#### ASCII Code Chart

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>NUL</td>
<td>SOH</td>
<td>STX</td>
<td>ETX</td>
<td>EOT</td>
<td>ENQ</td>
<td>ACK</td>
<td>BEL</td>
<td>BS</td>
<td>HT</td>
<td>LF</td>
<td>VT</td>
<td>FF</td>
<td>CR</td>
<td>SO</td>
<td>SI</td>
</tr>
<tr>
<td>1</td>
<td>DLE</td>
<td>DC1</td>
<td>DC2</td>
<td>DC3</td>
<td>DC4</td>
<td>NAK</td>
<td>SYN</td>
<td>ETB</td>
<td>CAN</td>
<td>EM</td>
<td>SUB</td>
<td>ESC</td>
<td>FS</td>
<td>GS</td>
<td>RS</td>
<td>US</td>
</tr>
<tr>
<td>2</td>
<td>!</td>
<td>=</td>
<td>#</td>
<td>$</td>
<td>%</td>
<td>&amp;</td>
<td>(</td>
<td>)</td>
<td>*</td>
<td>+</td>
<td>,</td>
<td>-</td>
<td>.</td>
<td>/</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>:</td>
<td>;</td>
<td>&lt;</td>
<td>=</td>
<td>&gt;</td>
<td>?</td>
</tr>
<tr>
<td>4</td>
<td>@</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
<td>F</td>
<td>G</td>
<td>H</td>
<td>I</td>
<td>J</td>
<td>K</td>
<td>L</td>
<td>M</td>
<td>N</td>
<td>O</td>
</tr>
<tr>
<td>5</td>
<td>P</td>
<td>Q</td>
<td>R</td>
<td>S</td>
<td>T</td>
<td>U</td>
<td>V</td>
<td>W</td>
<td>X</td>
<td>Y</td>
<td>Z</td>
<td>[</td>
<td>\</td>
<td>]</td>
<td>^</td>
<td>_</td>
</tr>
<tr>
<td>6</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
<td>e</td>
<td>f</td>
<td>g</td>
<td>h</td>
<td>i</td>
<td>j</td>
<td>k</td>
<td>l</td>
<td>m</td>
<td>n</td>
<td>o</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>p</td>
<td>q</td>
<td>r</td>
<td>s</td>
<td>t</td>
<td>u</td>
<td>v</td>
<td>w</td>
<td>x</td>
<td>y</td>
<td>z</td>
<td>{</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

4B = Latin capital K
ISO-8859 family (eg -1 Latin)

<table>
<thead>
<tr>
<th>0-7F match ASCII</th>
<th>reserved (control characters)</th>
<th>A0-FF differ, eg:</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-7F match ASCII</td>
<td>reserved (control characters)</td>
<td>A0-FF differ, eg:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>-1 &quot;Western&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>-2 &quot;East European&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>-9 &quot;Turkish&quot;</td>
</tr>
</tbody>
</table>
Windows Family (eg 1252 Latin)

<table>
<thead>
<tr>
<th>Windows-1252 (CP1252)</th>
</tr>
</thead>
<tbody>
<tr>
<td>x0</td>
</tr>
<tr>
<td>---</td>
</tr>
<tr>
<td>0x</td>
</tr>
<tr>
<td>1x</td>
</tr>
<tr>
<td>2x</td>
</tr>
<tr>
<td>3x</td>
</tr>
<tr>
<td>4x</td>
</tr>
<tr>
<td>5x</td>
</tr>
<tr>
<td>6x</td>
</tr>
<tr>
<td>7x</td>
</tr>
<tr>
<td>8x</td>
</tr>
<tr>
<td>9x</td>
</tr>
<tr>
<td>Ax</td>
</tr>
<tr>
<td>Bx</td>
</tr>
<tr>
<td>Cx</td>
</tr>
<tr>
<td>Dx</td>
</tr>
<tr>
<td>Ex</td>
</tr>
<tr>
<td>Fx</td>
</tr>
</tbody>
</table>

92 = apostrophe
Early Unicode and UTF-16

- Unicode started as $2^{16}$ code points
  - The BMP of modern Unicode
  - Bottom 256 code points match ISO-8859-1
- Simple 1:1 encoding (UTF-16)
  - Code point $\leftrightarrow$ 2 bytes (16 bits, 1 word)
  - Simple, but leads to bloat of ASCII text
- Later added code points outside of BMP
  - A pair of words (surrogate pairs) carry 20-bit payload split, 10 bits in each word
    - First: 1101 10xx xxxx xxxx (xD800-DBFF)
    - Second: 1101 11yy yyyy yyyy (xDC00-DFFF)
- Consequence: U+D800 to U+DFFF became reserved code points in Unicode
  - And now we are stuck with this legacy, even for UTF-8
Basic Multilingual Plane

<table>
<thead>
<tr>
<th>A0</th>
<th>A1</th>
<th>A2</th>
<th>A3</th>
<th>A4</th>
<th>A5</th>
<th>A6</th>
<th>A7</th>
<th>A8</th>
<th>A9</th>
<th>AA</th>
<th>AB</th>
<th>AC</th>
<th>AD</th>
<th>AE</th>
<th>AF</th>
</tr>
</thead>
<tbody>
<tr>
<td>B0</td>
<td>B1</td>
<td>B2</td>
<td>B3</td>
<td>B4</td>
<td>B5</td>
<td>B6</td>
<td>B7</td>
<td>B8</td>
<td>B9</td>
<td>BA</td>
<td>BB</td>
<td>BC</td>
<td>BD</td>
<td>BE</td>
<td>BF</td>
</tr>
<tr>
<td>C0</td>
<td>C1</td>
<td>C2</td>
<td>C3</td>
<td>C4</td>
<td>C5</td>
<td>C6</td>
<td>C7</td>
<td>C8</td>
<td>C9</td>
<td>CA</td>
<td>CB</td>
<td>CC</td>
<td>CD</td>
<td>CE</td>
<td>CF</td>
</tr>
<tr>
<td>D0</td>
<td>D1</td>
<td>D2</td>
<td>D3</td>
<td>D4</td>
<td>D5</td>
<td>D6</td>
<td>D7</td>
<td>D8</td>
<td>D9</td>
<td>DA</td>
<td>DB</td>
<td>DC</td>
<td>DD</td>
<td>DE</td>
<td>DF</td>
</tr>
</tbody>
</table>

- Latin scripts and symbols
- Linguistic scripts
- Other European scripts
- Middle Eastern and Southwest Asian scripts
- African scripts
- South Asian scripts
- Southeast Asian scripts
- East Asian scripts
- Unified CJK Han
- Canadian Aboriginal scripts
- Symbols
- Diacritics
- UTF-16 surrogates and private use
- Miscellaneous characters
- Unallocated code points
UTF-16 and Endianness

- A multi-byte representation must distinguish between big & little endian
- One solution: Specify encoding in name
  - UTF-16BE or UTF-16LE
- Another solution: require byte order mark (BOM) at the start of the file
  - U+FEFF (ZERO WIDTH NO BREAK SPACE)
  - There is no U+FFFE code point
  - So FE FF $\Rightarrow$ BigE, while FF FE $\Rightarrow$ LittleE
  - Not considered part of the text
BOM and UTF-8

- Should we add a BOM to the start of UTF-8 files too?
  - UTF-8 encoding of U+FEFF is EF BB BF
- Advantages:
  - Forms magic-number for UTF-8 encoding
- Disadvantages:
  - Not backwards-compatible to ASCII
  - Existing programs may no longer work
  - *E.g.*, In Unix, shebang (#!, *i.e.* 23 21) at *start* of file is significant: file is a script
    
    ```bash
    #! /bin/bash
    ```
Summary

- **Text vs binary**
  - In pre-historic times: most significant bit
  - Now: data is data

- **Unicode code points**
  - Integers U+0000..U+10FFFF
  - BMP: Basic Multilingual Plane

- **UTF-8**
  - A variable-length, self-synchronizing encoding of unicode code points
  - Backwards compatible with ISO 8859-1, and hence with ASCII too