Language Technology
http://cs.lth.se/edan20/
Chapter 3: Encoding and Annotation Schemes

Pierre Nugues

Pierre.Nugues@cs.lth.se
http://cs.lth.se/pierre_nugues/

August 31st, 2023

Character Sets

Codes are used to represent characters.

ASCII has 0 to 127 code points and is only for English

Latin-1 extends it to 256 code points. It can be used for most Western European languages but forgot many characters, like the French *Œ*, *œ*, the German quote „ or the Dutch *IJ*, *ij*.

Latin-1 was not adopted by all the operating systems, MacOS for instance; Windows used a variant of it.

Latin-9 is a better character set (published in 1999).

Unicode

Unicode is an attempt to represent most alphabets.

From *Programming Perl* by Larry Wall, Tom Christiansen, Jon Orwant, O'Reilly, 2000:

> *If you don't know yet what Unicode is, you will soon—even if you skip reading this chapter—because working with Unicode is becoming a necessity.*

It started with 16 bits and now uses 32 bits.

Ranges from 0 to 10FFFF in hexadecimal.

The standard character representation in many OSes and programming languages, including Java

Characters have a code point and a name as:

```
U+0042 LATIN CAPITAL LETTER B
U+0391 GREEK CAPITAL LETTER ALPHA
U+00C5 LATIN CAPITAL LETTER A WITH RING ABOVE
```

Unicode and Python

We obtain the code point of a character and the character corresponding to a code point with the `ord()` and `chr()` functions, respectively:

```
ord('C'), ord('Γ')      # (67, 915)
hex(67), hex(915)       # ('0x43', '0x393')
chr(67), chr(915)       # ('C', 'Γ')
```

Character Composition and Normalization

Unicode allows the composition of accented characters from a base character and one or more diacritics: *Ê* or *Å*.
Single code point:

```
U+00CA LATIN CAPITAL LETTER E WITH CIRCUMFLEX
U+00C5 LATIN CAPITAL LETTER A WITH RING ABOVE
```

They can also be defined as a sequence of two keys: E + ˆ and A + ˚:

```
U+0045 LATIN CAPITAL LETTER E
U+0302 COMBINING CIRCUMFLEX ACCENT
```

and

```
U+0041 LATIN CAPITAL LETTER A
U+030A COMBINING RING ABOVE
```

Normalization

Unicode defines a normalization form decomposition (NFD) and composition (NFC).
NFD decomposes Ê into U+0045 + U+0302 (E and ˆ)

```
[hex(ord(cp)) for cp in unicodedata.normalize('NFD', 'Ê')]
# ['0x45', '0x302']
```

and 'NFC' decomposes and recomposes it:

```
[hex(ord(cp)) for cp in unicodedata.normalize('NFC', 'Ê')]
# ['0xca']
```

In addition, we have the normalization form compatibility decomposition (NFKD) and composition (NFKC).
NFKD decomposes the *ﬁ* ligature (U+FB01) into U+0066 + U+0069, and makes it equivalent to the sequence of two letters: *f i*.

Unicode Blocks (Simplified)

Code	Name	Code	Name
U+0000	Basic Latin	U+1400	Unified Canadian Aboriginal Syllab
U+0080	Latin-1 Supplement	U+1680	Ogham, Runic
U+0100	Latin Extended-A	U+1780	Khmer
U+0180	Latin Extended-B	U+1800	Mongolian
U+0250	IPA Extensions	U+1E00	Latin Extended Additional
U+02B0	Spacing Modifier Letters	U+1F00	Extended Greek
U+0300	Combining Diacritical Marks	U+2000	Symbols
U+0370	Greek	U+2800	Braille Patterns
U+0400	Cyrillic	U+2E80	CJK Radicals Supplement
U+0530	Armenian	U+2F80	KangXi Radicals
U+0590	Hebrew	U+3000	CJK Symbols and Punctuation
U+0600	Arabic	U+3040	Hiragana, Katakana
U+0700	Syriac	U+3100	Bopomofo
U+0780	Thaana	U+3130	Hangul Compatibility Jamo

Unicode Blocks (Simplified) (II)

Code	Name	Code	Name
U+0900	Devanagari, Bengali	U+3190	Kanbun
U+0A00	Gurmukhi, Gujarati	U+31A0	Bopomofo Extended
U+0B00	Oriya, Tamil	U+3200	Enclosed CJK Letters and Months
U+0C00	Telugu, Kannada	U+3300	CJK Compatibility
U+0D00	Malayalam, Sinhala	U+3400	CJK Unified Ideographs Extension A
U+0E00	Thai, Lao	U+4E00	CJK Unified Ideographs
U+0F00	Tibetan	U+A000	Yi Syllables
U+1000	Myanmar	U+A490	Yi Radicals
U+10A0	Georgian	U+AC00	Hangul Syllables
U+1100	Hangul Jamo	U+D800	Surrogates
U+1200	Ethiopic	U+E000	Private Use
U+13A0	Cherokee	U+F900	Others

Regular Expressions and Unicode Classes

Unicode classes are recommended

Unicode defines classes using the `\p{class}` construct that matches the symbols in `class` and `\P{class}` that matches symbols not in `class`.

Expression	Description	Equivalent	`\p{...}` equiv.
`\d`	Any digit	`[0-9]`	`\p{digit}`
`\D`	Any nondigit	`[^0-9]`	`\P{digit}`
`\s`	Any whitespace character: space, tabulation, new line, carriage return, or form feed	`[ \t\n\r\f]`	`\p{space}`
`\S`	Any nonwhitespace character	`[^\s]`	`\P{space}`
`\w`	Any word character: letter, digit, or underscore	`[a-zA-Z0-9_]`	
`\W`	Any nonword character	`[^\w]`	

Always prefer the `\p{}` notation

Regular Expressions and Unicode Classes

Individual code points:

- `\N{LATIN CAPITAL LETTER E WITH CIRCUMFLEX}` and `\x{CA}` that match Ê and
- `\N{GREEK CAPITAL LETTER GAMMA}` and `\x{393}` that match γ.

Sets:

- We match code points in blocks, categories, and scripts with the `\p{property}` construct or its complement `\P{property}`.

For example,

- `\p{L}`, the letters, and `\P{L}` the nonletters,
- `\p{InGreek_and_Coptic}`, for a block,
- `\p{Currency_Symbol}` matches currency symbols,
- `\p{Greek}`, the Greek characters.

e.g., \p{N} gives all the numbers

Regular Expressions and Unicode Classes

Alway prefer Unicode to legacy classes

To match a string a letters (a word), use:

`\p{L}+` Always use this

and not

`\w+`

that is not standardized

Python and Java will not produce the same results

General Unicode Category

All categories for unicode

Major classes		Subclasses		Major classes		Subclasses	
Short	Long	Short	Long	Short	Long	Short	Long
L	Letter			Z	Separator		
		Lu	Uppercase_Letter			Zs	Space_Separator
		Ll	Lowercase_Letter			Zl	Line_Separator
		Lt	Titlecase_Letter			Zp	Paragraph_Separator
		Lm	Modifier_Letter				
		Lo	Other_Letter				
M	Mark						
		Mn	Nonspacing_Mark				
		Mc	Spacing_Mark				
		Me	Enclosing_Mark				
N	Number						
		Nd	Decimal_Number				
		Nl	Letter_Number				
		No	Other_Number				
P	Punctuation			C	Control		
		Pc	Connector_Punctuation			Cc	Control
		Pd	Dash_Punctuation			Cf	Format
		Ps	Open_Punctuation			Cs	Surrogate
		Pe	Close_Punctuation			Co	Private_Use
		Pi	Initial_Punctuation			Cn	Unassigned
		Pf	Final_Punctuation				
		Po	Other_Punctuation				
S	Symbol						
		Sm	Math_Symbol				
		Sc	Currency_Symbol				
		Sk	Modifier_Symbol				
		So	Other_Symbol				

Python and Unicode

The instructions below match lines consisting respectively of ASCII characters, of characters in the Greek and Coptic block, and of Greek characters:

```
import regex as re

alphabet = 'αβγδεζηϑικλμνξοπρστυφχψω'
match = re.search(r'^\p{InBasic_Latin}+$', alphabet)
print(match)  # None
match = re.search(r'^\p{InGreek_and_Coptic}+$', alphabet)
print(match)  # matches alphabet
match = re.search(r'^\p{Greek}+$', alphabet)
print(match)  # matches alphabet
match = re.search(r'\N{GREEK SMALL LETTER ALPHA}', alphabet)
print(match)  # matches 'α'
match = re.search('α', alphabet)
print(match)  # matches 'α'
```

The Unicode Encoding Schemes

Unicode offers three different encoding schemes: UTF-8, UTF-16, and UTF-32. All characters are represented but takes up 4 times more memory than UTF-8

UTF-16 was the standard encoding scheme.

It uses fixed units of 16 bits – 2 bytes –

FÊTE `0046 00CA 0054 0045`

UTF-8 is a variable length encoding.

It maps the ASCII code characters U+0000 to U+007F to their byte values 0x00 to 0x7F.

All the other characters in the range U+007F to U+FFFF are encoded as a sequence of two or more bytes.

UTF-8

Range	Encoding
U-0000 – U-007F	0xxxxxxx
U-0080 – U-07FF	110xxxxx 10xxxxxx
U-0800 – U-FFFF	1110xxxx 10xxxxxx 10xxxxxx
U-010000 – U-10FFFF	11110xxx 10xxxxxx 10xxxxxx 10xxxxxx

Encoding FÊTE in UTF-8

The letters F, T, and E are in the range U-00000000..U-0000007F.
Ê is `U+00CA` and is in the range U-00000080..U-000007FF.
Its binary representation is 0000 0000 1100 1010.
UTF-8 uses the eleven rightmost bits of 00CA.
The first five underlined bits together with the prefix 110 form the octet 1100 0011 that corresponds to `C3` in hexadecimal.
The seven next boldface bits with the prefix 10 form the octet 1000 1010 or `8A` in hexadecimal.
The letter Ê is encoded or `C3 8A` in UTF-8.
FÊTE and the code points U+0046 U+00CA U+0054 U+0045 are encoded as 46 `C3 8A` 54 45

problems occur when transforming ASCII to UTF

Locales and Word Order

Depending on the language, dates, numbers, time is represented differently:

Numbers: 3.14 or 3,14?

Time: 01/02/03

The order of words are vastly different in different languages and thus the representation of dates, numbers are affected

- 3 februari 2001?
- January 2, 2003?
- 1 February 2003?

Collating strings: is Andersson before or after Åkesson?

The Unicode Collation Algorithm

The Unicode consortium has defined a collation algorithm that takes into account the different practices and cultures in lexical ordering. It has three levels for Latin scripts:

- The primary level considers differences between base characters, for instance between A and B.
- If there are no differences at the first level, the secondary level considers the accents on the characters.
- And finally, the third level considers the case differences between the characters.

Differences

These level features are general, but not universal.

Accents are a secondary difference in many languages but Swedish sorts accented letters as individual ones and hence sets a primary difference between A and Å or O and Ö.

1. First level: {a, A, á, Á, à, À, etc.} < {b, B} < {c, C, ć, Ć, ĉ, Ĉ, ç, Ç, etc.} < {e, E, é, É, è, È, ê, Ê, ë, Ë, etc.} < ...
2. Second level: {e, E} << {é, É} << {è, È} << {ê, Ê} << {ë, Ë}
3. Third level: {a} <<< {A}

The comparison at the second level is done from the left to the right of a word in English, the reverse in French.

Sorting Words in French and English

English	French
Péché	*pèche*
PÉCHÉ	*pêche*
pèche	*Pêche*
pêche	*Péché*
Pêche	*PÉCHÉ*
pêché	*pêché*
Pêché	*Pêché*
pécher	*pécher*
pêcher	*pêcher*

Markup Languages

Extracting and absorbing content from the web, usually there are more than text.

Markup defines the presentation of the text

Markup languages are used to annotate texts with a structure and a presentation

Annotation schemes used by word processors include LaTex, RTF, etc.

XML, which resembles HTML, is now a standard annotation and exchange language

XML is a coding framework: a language to define ways of structuring documents.

XML is also used to create tabulated data (database-compatible data)

XML

XML uses plain text and not binary codes.

It separates the definition of structure instructions from the content – the data.

Structure instructions are described in a document type definition (DTD) that models a class of XML documents.

Document type definitions contain the specific tagsets to mark up texts.

A DTD lists the legal tags and their relationships with other tags.

XML has APIs available in many programming languages: Java, Perl, SWI Prolog, etc.

XML Elements

A DTD is composed of three kinds of components: elements, attributes, and entities.
The elements are the logical units of an XML document.
A DocBook-like description (`http://www.docbook.org/`)

```
<!-- My first XML document -->
<book>
  <title>Language Processing Cookbook</title>
  <author>Pierre Cagné</author>

  <!-- The image to show on the cover -->
  <img></img>
  <text>Here comes the text!</text>
</book>
```

Differences with HTML

XML tags must be balanced, which means that an end tag must follow each start tag.

Empty elements `<img></img>` can be abridged as `<img/>`.

XML tags are case sensitive: `<TITLE>` and `<title>` define different elements.

An XML document defines one single root element that spans the document, here `<book>`

XML Attributes

An element can have attributes, i.e. a set of properties.
A <title> element can have an alignment: flush left, right, or center, and a character style: underlined, bold, or italics.
Attributes are inserted as name-value pairs in the start tag

```
<title align="center" style="bold">
  Language Processing Cookbook
</title>
```

Entities

Entities are data stored somewhere in a computer that have a name. They can be accented characters, symbols, strings as well as text or image files.

An entity reference is the entity name enclosed by a start delimiter `&` and an end delimiter `;` such as `&EntityName;`

The entity reference will be replaced by the entity.

Useful entities are the predefined entities and the character entities

Entities (II)

There are five predefined entities recognized by XML.
They correspond to characters used by the XML standard, which can't be used as is in a document.

Symbol	Entity encoding	Meaning
<	`<`	Less than
>	`>`	Greater than
&	`&`	Ampersand
"	`"`	Quotation mark
'	`'`	Apostrophe

A character reference is the Unicode value for a single character such as `Ê` for Ê (or `Ê`)

Writing a DTD: Elements

A DTD specifies the formal structure of a document type.
A DTD file contains the description of all the legal elements, attributes, and entities.
The description of the elements is enclosed between the delimiters `<!ELEMENT` and `>`.

```
<!ELEMENT book (title, (author | editor)?, img, chapter+)>
<!ELEMENT title (#PCDATA)>
```

Character Types

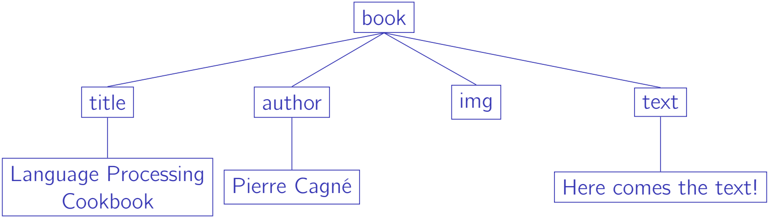
Character type	Description
`PCDATA`	Parsed character data. This data will be parsed and must only be text, punctuation, and special characters; no embedded elements
`ANY`	`PCDATA` or any DTD element
`EMPTY`	No content – just a placeholder

Writing an XML Document

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE book [
<!ELEMENT book (title, (author | editor)?, img, chapter+)>
<!ELEMENT title (#PCDATA)>
...
]>
<book>
  <title style="i">Language Processing Cookbook</title>
  <author style="b">Pierre Cagné</author>
  <img src="pierre.jpg"/>
  <chapter number="c1">
    <subtitle>Introduction</subtitle>
    <para>Let&apos;s start doing simple things: collect texts.
    </para>
    <para>First, choose a site you like</para>
  </chapter>
</book>
```

Tree Representation

The tree representation after parsing e.g., web content



Parsing HTML

```
import bs4
import requests

url_en = 'https://en.wikipedia.org/wiki/Aristotle'
html_doc = requests.get(url_en).text
parse_tree = bs4.BeautifulSoup(html_doc, 'html.parser')
```

Good parser and we get a structured tree

The `parse_tree` variable contains the parsed HTML document from which we can access its elements and their attributes.

Parsing HTML (II)

We access the title and its markup through the title attribute of `parse_tree` (`parse_tree.title`) and the content of the title with `parse_tree.title.text`:

```
parse_tree.title
# <title>Aristotle - Wikipedia, the free encyclopedia</title>
parse_tree.title.text
# Aristotle - Wikipedia, the free encyclopedia
```

Parsing HTML (III)

The first heading `h1` corresponds to the title of the article,

```
parse_tree.h1.text
# Aristotle
```

while the `h2` headings contain its subtitles. We access the list of subtitles using the `find_all()` method:

```
headings = parse_tree.find_all(’h2’)
[heading.text for heading in headings]
# [’Contents’, ’Life’, ’Thought’, ’Loss and preservation of
his works’, ’Legacy’, ’List of works’, ’Eponyms’, ’See also’,
’Notes and references’, ’Further reading’, ’External links’,
’Navigation menu’]
```