STAT243 Lecture 2.5 File and String Encodings

1 File and string encodings

Logic

文本数据的本质是**字符 ↔ 数值码位 ↔ 字节序列**之间的映射。理解 ASCII、Unicode、UTF-8 的关系，能系统性解决“乱码”“解码失败”等常见问题。

1.1 ASCII 基础与十六进制表示

- ASCII 含 2^7 = 128 个字符与控制码，基本等同于 US 键盘字符集。每字符占 1 byte (8 位)。
- 为了方便表示, 常以 2 个十六进制数表示 1 个字节。例如 `'M'` 的二进制 `01001101` 即十六进制 `0x4d`。(`0x` 表示 16 进制)
- 在 Python 中可以**手工写入 ASCII 字节**到文件，再读回验证。

Python

```python
# 用十六进制字节写入 ASCII 文本 "Mom\n"
hexvals = b'\x4d\x6f\x6d\x0a'
    # \x 表示后面是一个16进制编码的字节
    # b 表示这是一个字节串 bytes object

with open('tmp.txt', 'wb') as textfile:
    nbytes = textfile.write(hexvals)    # 会返回写入的字节数
print(nbytes)
# 输出:
# 4
```

Python

```python
subprocess.run(["ls", "-l", "tmp.txt"], capture_output=True).stdout
# 输出:
# b'-rw-r--r-- 1 paciorek scfstaff 4 Sep  3 08:56 tmp.txt\n'

with open('tmp.txt', 'r') as textfile:
     line = textfile.readlines()
line
# 输出:
# ['Mom\n']
```

1.2 Unicode 与 UTF-8：码位与编码

- **Unicode** 为字符分配**唯一整数码位**（code point）。Python 中 `str` 持有 Unicode 字符。
- **UTF-8** 是将 Unicode 码位编码为**变长字节序列**的通用方案：ASCII 仍为 1 byte，其他多为 2–4 bytes。

Remark

- Unicode 是字符的“抽象编号系统”；
- UTF-8 是一种把这些编号“具体编码成字节”的方法。

Python

```python
# Python str 是 Unicode
```

```python
x2_unicode = 'Pe\u00f1a 3\u00f72'  # 包含 ñ 和 ÷
    # \u 表示这是一个 Unicode 字符的码点
print(x2_unicode)
# 输出:
# Peña 3÷2

print(type(x2_unicode))
# 输出:
# <class 'str'>
```

Python

```python
# 从字符到码位（整数）再到十六进制
print(ord('ñ')) # ñ 的 Unicode 编号是 U+00F1 = 241（十进制）
# 输出:
# 241

print(hex(ord('ñ')))
# 输出:
# 0xf1

# 查看 UTF-8 编码后的字节序列（十六进制转义）
print(bytes('\u00f1', 'utf-8'))  # 'ñ' 在 UTF-8 中占用两个字节：`0xC3 0xB1`
# 输出:
# b'\xc3\xb1'

print(bytes('\u00f7', 'utf-8'))  # ÷
# 输出:
# b'\xc3\xb7'
```

Python

```python
# 直接写入 UTF-8 字节到文件，再从 shell 验证
x2_utf8 = b'Pe\xc3\xb1a 3\xc3\xb72'
with open('tmp2.txt', 'wb') as textfile:
    nbytes = textfile.write(x2_utf8)
print(nbytes)
# 输出:
# 10
```

Shell

```shell
# 文件大小与内容（n-tilde 与 division symbol 各占 2 字节）
ls -l tmp2.txt
# 输出:
# -rw-r--r-- 1 user group 10 Sep 3 08:55 tmp2.txt

cat tmp2.txt
# 输出:
# Peña 3÷2
```

1.3 UTF-8 的 bit-wise representation 设计要点

- 兼容 ASCII：保证旧系统可直接读取。
- 避免混淆: **短编码模式的比特**不会出现在更长模式的内部，
- 前缀自描述：通过**前导比特模式**判断字符所占字节数，首位为 0 的字节即 ASCII。
- 实务含义：顺序扫描时可**无歧义**地解析字符边界；随机访问时仍需自前定位边界。

1.4 识别与转换：工具与参数

- **UNIX 工具**：
  - `file path` 可粗略识别文件类型与编码。
  - `iconv -f src -t dst` 可进行编码转换。
- **Python**：
  - 读文本时可在 `open(..., encoding='utf-8')` 显式声明编码。
  - 已在内存的 `str` 可用 `.encode('utf-8' | 'latin1' | 'ascii' ...)` 得到 `bytes`，实现**编码转换**。

Python

```python
# 显式指定读取编码
with open('file_nonascii.txt', 'r', encoding='latin1') as f:
    lines = f.readlines()
print(lines[0][:40])
# 输出:
# （示例）前 40 个字符...
```

1.5 Python 默认编码与 locale

- 现代 Python 默认源代码与 I/O 多以 **UTF-8** 为默认。
- 可用 `locale.getlocale()` 查看环境区域设置与默认编码。

Python

```python
import locale
print(locale.getlocale())
# 输出:
# ('en_US', 'UTF-8')
```

1.6 变量名中的 Unicode（可读性与可移植性）

- Python、R、Julia 等均可在**变量名**中使用 Unicode 字符，但渲染与工具链支持可能不同。
- 建议在**教学或多语言协作**中谨慎使用，仅在上下文明确时采用。

Python

```python
peña = 7   # 变量名包含 ñ
print(peña)
# 输出:
# 7

# 某些 PDF/终端不一定正确显示希腊字母等符号，注意渲染差异
```

1.7 编码转换示例与常见错误

- 同一 Unicode 文本经不同编码得到不同 `bytes` 表示；`latin1` 对西欧字符常更短，但**字符集更窄**。
- 将包含非 ASCII 字符的 `str` 用 `'ascii'` 编码会报错。

Python

```python
text = 'Pe\u00f1a 3\u00f72'  # 'Peña 3÷2'

print(text.encode('utf-8'))
# 输出:
# b'Pe\xc3\xb1a 3\xc3\xb72'

print(text.encode('latin1'))
# 输出:
# b'Pe\xf1a 3\xf72'
```

```python

try:
    text.encode('ascii')
except Exception as error:
    print(error)
# 输出:
# 'ascii' codec can't encode character '\xf1' in position 2: ordinal not in range(128)
```

> ⚠ **Remark**
>
> 上例说明：其中 2 个非 ASCII 字符在 UTF-8 需各 2 bytes，而在 Latin-1 各 1 byte。Latin-1 覆盖约 191 个附加字符，主要是西欧语言的带重音字母等，但**远不及 Unicode 完整**。

1.8 UnicodeDecodeError 的定位与修复

- 典型症状：以默认 UTF-8 读取实际为 Latin-1（或其他编码）的文件会报错，例如：

Python

```python
# 错误示例：未显式声明编码
with open('file_nonascii.txt', 'r') as textfile:
    lines = textfile.readlines()
# 输出:
# UnicodeDecodeError: 'utf-8' codec can't decode byte 0xac in position 7922: invalid start byte
```

- 解决：**显式指定真实编码**。确认后重读即可。

Python

```python
with open('file_nonascii.txt', 'r', encoding='latin1') as textfile:
    lines = textfile.readlines()
print(lines[16925])
# 输出:
# 'from 5#c¿a7lw8lz2nX,%@ [128.32.244.179] by ncpc-email with ESMTP\n'
```

> **Logic**
>
> 排查顺序：先用 `file` 粗判 → 尝试以 UTF-8 读取 → 若失败，结合数据来源与地区试 `latin1` 等 → 使用 `iconv` 或 `.encode/.decode` 做转换。始终**在边界上验证**：抽样查看有无异常字符、混合换行或不可见控制符。