C: Handling Unicode in Python

Complications arises with Unicode and multi-byte characters:

Example:

días

Char:	d			a	s
Hex:	64	c3	ad	61	73
Dec:	100	195	173	97	115

Complications:

1. How long is 'días'?

As characters:	4
As bytes:	5

2. What's at location 1?

As characters:	í	(one character but two bytes)
As bytes:	hex c3	(one byte, half a character)

Cannot avoid completely: need each in different contexts

Python handles this via **two data types** and **file modes**:

1. Using data as **characters**

   - Length and subscripts based on **characters**
   - Each **character** counts as **1 unit** even if composed of multiple bytes

Data type: **str**

File mode: **text**

```
svar = 'días'


fh = open(filename)
fh = open(filename, 'w')
```

2. Using data as **raw bytes**

- Length and subscripts based on **bytes**
- Each **byte** counts as **1 unit** but may not be a complete character
- May need to include hex codes in scripts via \x

Data type: **bytes**

File mode: **binary**

bvar = **b**'d\x**c3**\x**ad**as'	Note the b before the first quote \x**c3** indicates hex (\x) code **c3** \x**ad** indicates hex code **ad**
fh = open(filename,'**rb**')	Note the ending b
fh = open(filename,'w**b**')	

- Broadly speaking: works well but **don't cross the streams**:

Use **strings with strings** and **bytes with bytes**

'1,2'.split(',')	ok: both str
b'1,2'.split(b',')	ok: both bytes

b'1,2'.split(',')   error: byte and str

When issues arise, usually due to using the **wrong file mode**

- How to switch between data types:

bvar = svar.encode()	Make bytes **bvar** from string **svar**
svar = bvar.decode('utf-8')	Make string **svar** from bytes **bvar**

'días'.**encode()**	→	b'd\xc3\xadas'
b'd**\xc3\xad**as'.**decode('utf-8')**	→	'días'