

Chapter 12 — String Encoding

Introduction

Strings are made up of a collection of bytes (8 binary digits) that represent the characters that the string contains. In Python 3 string are encoded following the UTF-8 standard, and may contain 1,112,064 different code points (or symbols). This allows Python programs to process strings of all languages, throughout the world.

Objectives

Upon completion of this chapter's exercises, you should be able to:

- Use the ASCII character set to represent characters as numbers and to convert numbers back to their ASCII character.
- Define and apply the UNICODE character encoding to extend the ASCII set to represent a myriad of international characters and symbols.
- Specifically understand the UTF-8 method of representing UNICODE characters.
- Differentiate a byte array from a string and convert one to another,

Prerequisites

This Chapter requires...

ASCII

The American Standard Code for Information Interchange (ASCII) was created in 1963 to standardize the way string data was to be stored and communicated between computer systems. Before this standard was widely adopted, there were several encodings adopted by different computer manufactures.

ASCII uses the first seven bits in a byte to encode 128 different characters, or code points as they can be generically called. Because ASCII was an American standard, it did not include a method to store string data from other regions of the world.

Even though ASCII has been generally replaced by the more inclusive encoding of Unicode, it still is used and is actually a subset of the widely used UTF-8 encoding.

An ASCII example

We can easily loop through a string, letter by letter, using a for loop. The Python built in function `ord()` returns the integer number representing the ASCII code.

<code>ord(character)</code>	Function
The <code>ord()</code> function will return a value representing the UNICODE number for that character. Because ASCII is a sub-set of UNICODE, this function will return the ASCII values for ASCII characters.	
REF	

```
1| text = 'Python 3'
2| for c in text:
3|     a = ord(c) # get ascii code for a character
4|     print(c, bin(a), hex(a), a)
```

```
P 0b1010000 0x50 80
y 0b1111001 0x79 121
t 0b1110100 0x74 116
h 0b1101000 0x68 104
o 0b1101111 0x6f 111
n 0b1101110 0x6e 110
 0b100000 0x20 32
3 0b110011 0x33 51
```

	BIN	000	001	010	011	100	101	110	111
BIN	HEX	0	1	2	3	4	5	6	7
0000	0	NUL	DLE	SP	0	@	P	`	p
0001	1	SOH	DC1	!	1	A	Q	a	q
0010	2	STX	DC2	"	2	B	R	b	r
0011	3	ETX	DC3	#	3	C	S	c	s
0100	4	EOT	DC4	\$	4	D	T	d	t
0101	5	ENQ	NAK	%	5	E	U	e	u
0110	6	ACK	SYN	&	6	F	V	f	v
0111	7	BEL	ETB	'	7	G	W	g	w
1000	8	BS	CAN	(8	H	X	h	x
1001	9	HT	EM)	9	I	Y	i	y
1010	A	LF	SUB	*	:	J	Z	j	z
1011	B	VT	ESC	+	;	K	[k	{
1100	C	FF	FS	,	<	L	\	l	
1101	D	CR	GS	-	=	M]	m	}
1110	E	S0	RS	.	>	N	^	n	~
1111	F	S1	US	/	?	O	_	o	DEL

Table 9: ASCII Character Encoding Table

Unicode

In the late 1980s and early 1990s Xerox, Apple, Microsoft, and others begin working on a new way to represent characters. The early idea was to widen the existing character set to 16 bits, to allow 65,536 code points. It was originally thought that this would cover the vast majority of characters in modern languages. This technique was known as UCS-2 but was found to be too large and limiting. This required creating a better and more flexible method for encoding characters, we call that UTF-8.

The Unicode Consortium is a collection of many of the largest companies in the tech world. Members include: Apple, Oracle, IBM, Microsoft, Google and others. The Unicode specification is a living document that is being revised on a regular basis. The Unicode 11.0 specification even defines code points for 1644 emojis.

UTF-8

UTF-8 was initially specified in 1996, and by 2009 had become the dominant character encoding for

paged on the World Wide Web. Since Python 3.0 was introduced, UTF-8 is how strings are stored in memory. It can encode over 1.1 million different code points.

UTF-8 Is a variable length coding method that allows for the most common code points to be represented with one or two bytes, while the least common code points may take up to four bytes to represent. This variable length encoding is accomplished by setting certain bits in the byte stream signifying how many bytes long this code is. The ASCII code points in the range of 0-127 (when the high order bit is set to 0) are the codes that fit on a single byte. This allows ASCII text files to be compliant with the UTF-8 format.

```
1| # -*- coding: utf-8 -*-
2| text = 'Python is hard work. 😊'
3| for c in text:
4|     a = ord(c) # get unicode code for a character
5|     print(c, bin(a), hex(a), a)
```

```
P 0b1110100000 0x3a0 928
t 0b1110100 0x74 116
h 0b1101000 0x68 104
o 0b1101111 0x6f 111
n 0b1101110 0x6e 110
   0b100000 0x20 32
i 0b1101001 0x69 105
s 0b1110011 0x73 115
   0b100000 0x20 32
h 0b1101000 0x68 104
a 0b1100001 0x61 97
r 0b1110010 0x72 114
d 0b1100100 0x64 100
   0b100000 0x20 32
w 0b1110111 0x77 119
o 0b1101111 0x6f 111
r 0b1110010 0x72 114
k 0b1101011 0x6b 107
. 0b101110 0x2e 46
   0b100000 0x20 32
😊 0b111110111000000101 0x1f605 128517
```

NOTE: You will notice that the first line of the previous Python program begins with a comment statement like `# -*- coding: utf-8 -*-`. This line tells Python and most Python editors (PyCharm, Spyder and others) to read and process the file as UTF-8 and not as ASCII. This was required because of the Unicode characters in the text of the program. This "magic comment" was defined in [PEP-263](https://www.python.org/dev/peps/pep-0263/).

Bytes (Constants)

In previous chapters we have seen many types of constants (integer, binary numbers, hexadecimal numbers, floating-point numbers, and strings). There is another called Bytes that represents a sequence of bytes. Bytes is a collection of raw 8 bit data and is not encoded in any special way.

Constants of the Bytes type may be put in your code by prefixing a string of ASCII characters with the letter 'b'. The quoted sequence of bytes may only contain ASCII characters, and not the full collection of Unicode code points. If you need to embed bytes by their hexadecimal values, use the `\x##` escape sequence with two hexadecimal characters.

```
b'a group of ASCII characters!!'  
b"another GROuP."  
b'mixed\xFF\x10\xd0.'  
b'''triple single "quoted" ASCII letters'''  
b"""triple double 'quoted' ASCII letters"""
```

Bytes may also be defined using a string of hexadecimal digits. This becomes useful if we want to include bytes outside the range from 32-127 in our bytes constant. To do this we can use the `bytes.fromhex()` method.

<code>bytes.fromhex(hex_string)</code>	Method of the byte class
Because many bytes are non printing, especially the ones less than 32 or greater than 127, you may represent an array of bytes as a string of hexadecimal values. Each pair of characters represent a number from 0-255, a byte.	
REF	

```
a = bytes.fromhex("FFFEF405099A0")
```

Converting Strings to Bytes and Bytes to Strings

If a constant string contains only ASCII characters, it can easily be converted to Bytes by prefixing it with 'b', as seen above. To convert a UTF-8 or any other type of encoded string we need to use a second argument on the `bytes()` or `str()` that specifies how the string is encoded or how we want the string encoded.

```
1 | # -*- coding: utf-8 -*-
```

```
2| b = bytes("abc 😊 XYZ", 'UTF_8')
3| print(b)
4| c = b'abc \xf0\x9f\x98\x85 XYZ'
5| print(str(c, 'UTF_8'))
6| a = b'an ASCII string with Unicode \xE2\x80\x9Cquotes\xE2\x80\x9D.'
7| print(str(a, "utf_8"))
```

```
b'abc \xf0\x9f\x98\x85 XYZ'
abc 😊 XYZ
an ASCII string with Unicode "quotes".
```

A few common encoding are in the table below, but a complete list can be found at <https://docs.python.org/3.7/library/codecs.html#standard-encodings>.

Codec	Aliases	Languages
ascii	646, us-ascii	English
cp850	850, IBM850	Western Europe
utf_32	U32, utf32	all languages
utf_16	U16, utf16	all languages
utf_8	U8, UTF, utf8	all languages

Table 10: Common Character Encodings

Converting Bytes to Integers and Integers to Strings

Under the hood, Python stores integers with a variable number of bytes, so that very large or very small numbers may be represented efficiently. In many programs and systems, integers are stored as a fixed number of bytes. Python includes a method to convert an Integer into a fixed binary representation and another to decode the integer from a collection of bytes.

A problem with storage of numbers in this raw format is that there are several different ways the bits and bytes may be arranged. The two most common arrangements of bytes are known as little-endian and big-endian encoding. In little-endian encoding the low order byte is placed first in the collection where in the big-endian encoding the high order byte is placed first. You can read more about endianness at <https://en.wikipedia.org/wiki/Endianness>.

Example:

4350 is 0001000011111110 in binary and 10FE in hexadecimal.

b"\x10\xfe" is big-endian and
b"\xfe\x10" is little-endian

The method `to_bytes` requires two arguments, The first is the width in bytes and the second is how to encode the bytes. The second method `from_bytes` takes a collection of bytes, how it was ordered, and will return an integer.

```
1| a = 8853778
2|
3| # converting an integer to a "big-endian" collection of bytes
4| b = a.to_bytes(4, byteorder='big')
5| print(b)
6| print(int.from_bytes(b, byteorder='big'))
7|
8| # converting an integer to a "little-endian" collection of bytes
9| b = a.to_bytes(4, byteorder='little')
10| print(b)
11| print(int.from_bytes(b, byteorder='little'))

b'\x00\x87\x19\x12'
8853778
b'\x12\x19\x87\x00'
8853778
```

Summary

Goes here

Important Terms

here

Exercises

Here

Word Search

References

<https://en.wikipedia.org/wiki/UTF-8>

<https://en.wikipedia.org/wiki/ASCII>

<https://docs.python.org/3/howto/unicode.html>

<https://en.wikipedia.org/wiki/Unicode>

<http://syw2l.org>

