

PYTHON ДЛЯ СЕТЕВЫХ ИНЖЕНЕРОВ

UNICODE

ЗАЧЕМ НУЖНА КОДИРОВКА?

КОМПЬЮТЕРЫ РАБОТАЮТ С БАЙТАМИ

Программы, которые мы пишем, не изолированы в себе. Они скачивают данные из Интернета, читают и записывают данные на диск, передают данные через сеть.

Поэтому очень важно понимать разницу между тем, как компьютер хранит и передает данные, и как эти данные воспринимает человек. Мы воспринимаем текст, а компьютер - байты.

КОМПЬЮТЕРЫ РАБОТАЮТ С БАЙТАМИ

В Python 3, соответственно, есть две концепции:

- текст - неизменяемая последовательность Unicode символов. Для хранения этих символов используется тип строка (str)
- данные - неизменяемая последовательность байтов. Для хранения используется тип bytes

Мы получаем байты при работе с:

- сетью
- файлами

ЗАЧЕМ НУЖНА КОДИРОВКА?

Для записи символов в байты, нужна определенная договоренность как они будут выглядеть:

- A - 0x41
- F - 0x46

СТАНДАРТ ASCII

ASCII (American standard code for information interchange) - описывает соответствие между символом и его числовым кодом. Изначально описывал только 127 символов:

- коды от 32 до 127 описывали печатные символы
- коды до 32 описывали специальные управляющие символы

ASCII Code Chart

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

ISO LATIN 1 (ISO 8859-1)

	_0	_1	_2	_3	_4	_5	_6	_7	_8	_9	_A	_B	_C	_D	_E	_F
0_																
1_																
2_		!	"	#	\$	%	&	'	()	*	+	,	-	.	/
3_	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4_	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5_	P	Q	R	S	T	U	V	W	X	Y	Z	[\]	^	_
6_	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7_	p	q	r	s	t	u	v	w	x	y	z	{		}	~	
8_																
9_																
A_		ı	ç	£	¤	¥	¦	§	¨	©	ª	«	¬	-	®	¯
B_	°	±	²	³	´	µ	¶	·	,	¹	º	»	¼	½	¾	¿
C_	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï
D_	Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü	Ý	Þ	ß
E_	à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï
F_	ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù	ú	û	ü	ý	þ	ÿ

WINDOWS CP1252

	_0	_1	_2	_3	_4	_5	_6	_7	_8	_9	_A	_B	_C	_D	_E	_F
0_		Г	г	Л	л		—	•	◼			♂	□		♢	⦿
1_	†	◀	↕	∥	¶	⊥	⌊	‡	↑	‡	→	←				
2_		!	"	#	\$	%	&	'	()	*	+	,	-	.	/
3_	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4_	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5_	P	Q	R	S	T	U	V	W	X	Y	Z	[\]	^	_
6_	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7_	p	q	r	s	t	u	v	w	x	y	z	{		}	~	
8_	€		,	f	„	...	†	‡	^	‰	Š	‹	Œ		Ž	
9_		‘	’	“	”	•	—	—	~	™	š	›	œ		ž	ÿ
A_		ı	¢	£	¤	¥	¦	§	¨	©	ª	«	¬	-	®	¯
B_	°	±	²	³	´	µ	¶	·	,	¹	º	»	¼	½	¾	¿
C_	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï
D_	Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü	Ý	Þ	ß
E_	à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï
F_	ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù	ú	û	ü	ý	þ	ÿ

UNICODE

СТАНДАРТ UNICODE

- 1,114,112 кодов
- диапазон 0x0 - 0x10FFFF
- стандарт Unicode версии 10.0 (Июнь 2017) определяет 136 690 символов
- каждый код - это номер, который соответствует определенному символу
- стандарт также определяет кодировки - способ представления кода символа в байтах

ПРИМЕРЫ СИМВОЛОВ

- U+1F383 JACK-O-LANTERN
- U+2615 HOT BEVERAGE ☕
- U+1F600 GRINNING FACE

SCHÖN

U+0073 U+0063 U+0068 U+00F6 U+006E

!РЕПУС ЫТ ,EDOCINU

AND THAT'S NOT EVEN THE
WORST PART! THE WORST
PART IS THAT —

U+202e

...NEVE T'NDID YEHT—
?LLEH EHT TAHW...
...UOY DID WOH

.ELOHSSA...




КОДИРОВКИ

- UTF-8
- UTF-16
- UTF-32

UTF-8

- позволяет хранить символы Юникода
- использует переменное количество байт
- символы ASCII обозначаются такими же кодами

ПРИМЕРЫ СИМВОЛОВ

Н	і			
48	69	01 f6 c0	01 f6 80	26 03

UNICODE В PYTHON 3

UNICODE В PYTHON 3

В Python 3 есть:

- строки - неизменяемая последовательность Unicode символов. Для хранения этих символов используется тип строка (str)
- байты - неизменяемая последовательность байтов. Для хранения используется тип bytes

СТРОКИ

STR

Строка в Python 3 - это последовательность кодов Unicode.

```
In [1]: hi = 'привет'
```

```
In [2]: type(hi)
```

```
Out[2]: str
```

```
In [3]: hi.upper()
```

```
Out[3]: 'ПРИВЕТ'
```

STR

Так как строки - это последовательность кодов Юникод, можно записать строку разными способами.

Символ Юникод можно записать, используя его имя:

```
In [1]: "\N{LATIN SMALL LETTER O WITH DIAERESIS}"  
Out[1]: 'ö'
```

Или используя такой формат:

```
In [4]: "\u00F6"  
Out[4]: 'ö'
```

STR

Строку можно записать как последовательность кодов Unicode

```
In [19]: hi1 = 'привет'
```

```
In [20]: hi2 = '\u043f\u0440\u0438\u0432\u0435\u0442'
```

```
In [21]: hi2
```

```
Out[21]: 'привет'
```

```
In [22]: hi1 == hi2
```

```
Out[22]: True
```

```
In [23]: len(hi2)
```

```
Out[23]: 6
```

ORD

Функция `ord` возвращает значение кода Unicode для символа:

```
In [7]: ord('п')
```

```
Out[7]: 1087
```

```
In [8]: hex(ord("a"))
```

```
Out[8]: '0x61'
```

CHR

Функция chr возвращает строку Unicode, которая символу, чем код был передан как аргумент:

```
In [9]: chr(1087)
```

```
Out[9]: 'п'
```

```
In [10]: chr(8364)
```

```
Out[10]: '€'
```

```
In [11]: chr(9731)
```

```
Out[11]: '🤖'
```

BYTES

BYTES

Тип `bytes` - это неизменяемая последовательность байтов.

Байты обозначаются так же, как строки, но с добавлением буквы `"b"` перед строкой

BYTES

```
In [30]: b1 = b'\xd0\xb4\xd0\xb0'
```

```
In [31]: b2 = b"\xd0\xb4\xd0\xb0"
```

```
In [32]: b3 = b'''\xd0\xb4\xd0\xb0'''
```

```
In [36]: type(b1)
```

```
Out[36]: bytes
```

```
In [37]: len(b1)
```

```
Out[37]: 4
```


ASCII В BYTES

В Python байты, которые соответствуют символам ASCII, отображаются как эти символы, а не как соответствующие им байты. Это может немного путать, но всегда можно распознать тип bytes по букве b:

```
In [38]: bytes1 = b'hello'

In [39]: bytes1
Out[39]: b'hello'

In [40]: len(bytes1)
Out[40]: 5

In [42]: bytes2 = b'\x68\x65\x6c\x6c\x6f'

In [43]: bytes2
Out[43]: b'hello'
```

NON ASCII

Если попытаться написать не ASCII символ в байтовом литерале, возникнет ошибка:

```
In [44]: bytes3 = b'привет'
File "<ipython-input-44-dc8b23504fa7>", line 1
    bytes3 = b'привет'
              ^
SyntaxError: bytes can only contain ASCII literal characters.
```

BYTES

Можно работать с байтовыми строками, как с unicode строками:

```
In [17]: d = {b'hi': 'Hello', b'by': 'Goodbye'}

In [18]: d[b'hi']
Out[18]: 'Hello'

In [19]: d['hi']
-----
KeyError                                Traceback (most recent call last)
<ipython-input-38-259732fc8381> in <module>()
----> 1 d['hi']

KeyError: 'hi'
```

КОНВЕРТАЦИЯ МЕЖДУ БАЙТАМИ И СТРОКАМИ

ENCODE VS DECODE

Избежать работы с байтами нельзя. Например, при работе с сетью или файловой системой, чаще всего, результат возвращается в байтах.

Соответственно, надо знать, как выполнять преобразование байтов в строку и наоборот. Для этого и нужна кодировка.

UNICODE .ENCODE() → BYTES

BYTES .DECODE() → UNICODE

ENCODE VS DECODE

Кодировку можно представлять как ключ шифрования, который указывает:

- как "зашифровать" строку в байты (str -> bytes).
Используется метод encode (похож на encrypt)
- как "расшифровать" байты в строку (bytes -> str).
Используется метод decode (похож на decrypt)

Эта аналогия позволяет понять, что преобразования строка-байты и байты-строка должны использовать одинаковую кодировку.

ENCODE

Для преобразования строки в байты используется метод `encode`:

```
In [1]: hi = 'привет'

In [2]: hi.encode('utf-8')
Out[2]: b'\xd0\xbf\xd1\x80\xd0\xb8\xd0\xb2\xd0\xb5\xd1\x82'

In [3]: hi_bytes = hi.encode('utf-8')
```

DECODE

Чтобы получить строку из байт, используется метод decode:

```
In [4]: hi_bytes
Out[4]: b'\xd0\xbf\xd1\x80\xd0\xb8\xd0\xb2\xd0\xb5\xd1\x82'

In [5]: hi_bytes.decode('utf-8')
Out[5]: 'привет'
```


STR. ENCODE

Метод encode есть также в классе str:

```
In [6]: hi
Out[6]: 'привет'

In [7]: str.encode(hi, encoding='utf-8')
Out[7]: b'\xd0\xbf\xd1\x80\xd0\xb8\xd0\xb2\xd0\xb5\xd1\x82'
```

BYTES.DECODE

Метод decode есть у класса bytes (как и другие методы):

```
In [8]: hi_bytes
Out[8]: b'\xd0\xbf\xd1\x80\xd0\xb8\xd0\xb2\xd0\xb5\xd1\x82'

In [9]: bytes.decode(hi_bytes, encoding='utf-8')
Out[9]: 'привет'
```

STR. ENCODE, BYTES.DECODE

В этих методах кодировка может указываться как ключевой аргумент (примеры выше) или как позиционный:

```
In [10]: hi_bytes
Out[10]: b'\xd0\xbf\xd1\x80\xd0\xb8\xd0\xb2\xd0\xb5\xd1\x82'

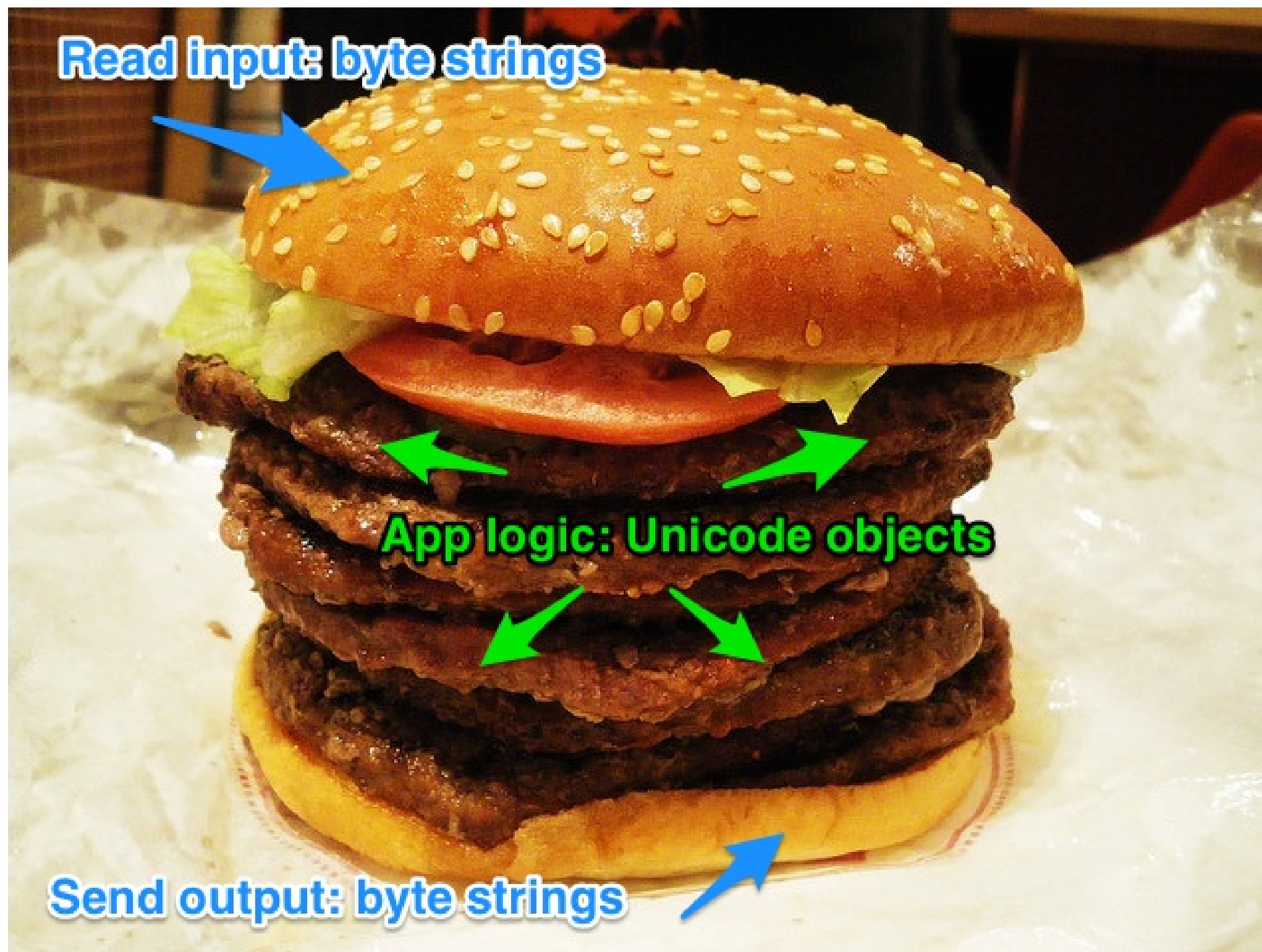
In [11]: bytes.decode(hi_bytes, 'utf-8')
Out[11]: 'привет'
```

КАК РАБОТАТЬ С ЮНИКОД И БАЙТАМИ

UNICODE SANDWICH

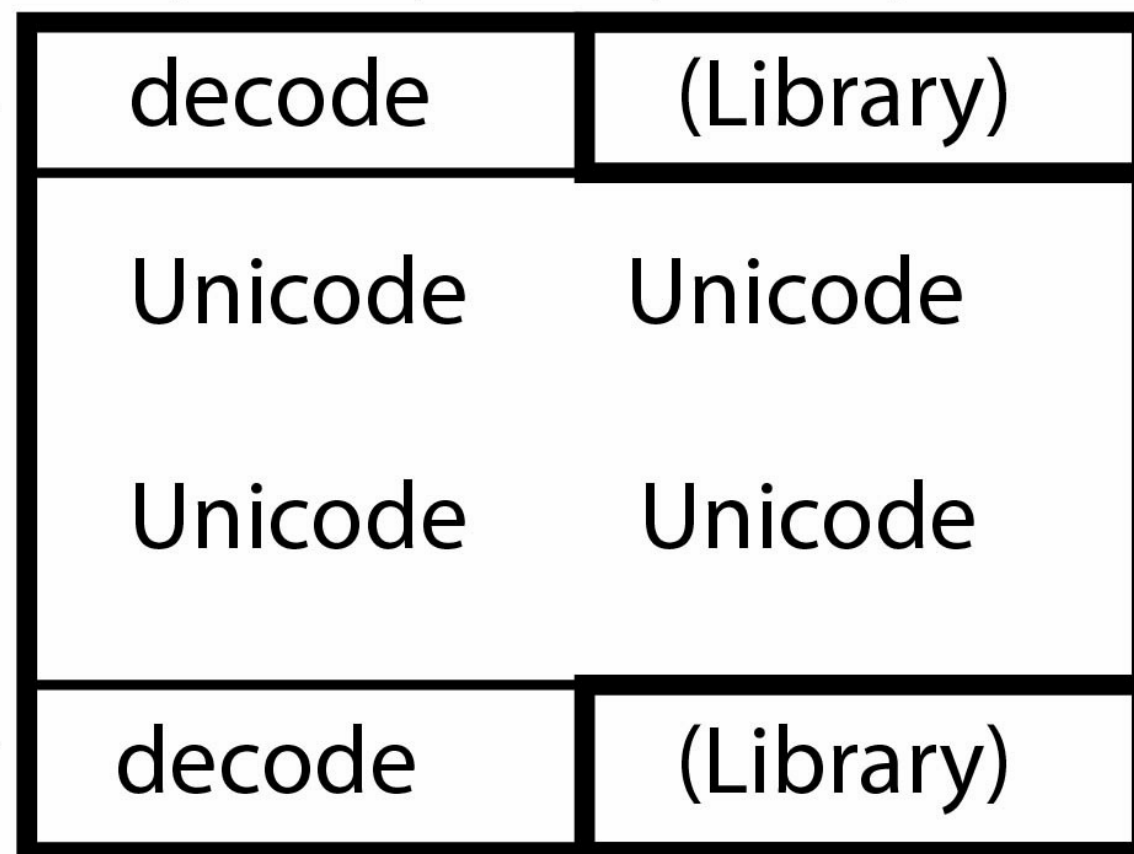
Есть очень простое правило, придерживаясь которого, можно избежать, как минимум, части проблем. Оно называется "Юникод сендвич":

- байты, которые программа считывает, надо как можно раньше преобразовать в юникод (строку)
- внутри программы работать с юникод
- юникод надо преобразовать в байты как можно позже, перед передачей



System
Boundary

bytes bytes bytes bytes



Your Code

bytes bytes bytes bytes

ПРИМЕРЫ КОНВЕРТАЦИИ МЕЖДУ БАЙТАМИ И СТРОКАМИ

SUBPROCESS

SUBPROCESS

Модуль subprocess возвращает результат команды в виде байт:

```
In [1]: import subprocess
```

```
In [2]: result = subprocess.run(['ping', '-c', '3', '-n', '8.8.8.8'],  
...:                             stdout=subprocess.PIPE)  
...:
```

```
In [3]: result.stdout
```

```
Out[3]: b'PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.\n64 bytes from 8.8.8.8: icmp_seq=1 ttl=43 time=59.4 r
```

4

▶

SUBPROCESS

Если дальше необходимо работать с ЭТИМ ВЫВОДОМ, надо сразу конвертировать его в строку:

```
In [4]: output = result.stdout.decode('utf-8')

In [5]: print(output)
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=43 time=59.4 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=43 time=54.4 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=43 time=55.1 ms

--- 8.8.8.8 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2002ms
rtt min/avg/max/mdev = 54.470/56.346/59.440/2.220 ms
```

SUBPROCESS ENCODING

Модуль subprocess поддерживает еще один вариант преобразования - параметр encoding.

Если указать его при вызове функции run, результат будет получен в виде строки:

```
In [6]: result = subprocess.run(['ping', '-c', '3', '-n', '8.8.8.8'],
...:                             stdout=subprocess.PIPE, encoding='utf-8')
...:

In [7]: result.stdout
Out[7]: 'PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.\n64 bytes from 8.8.8.8: icmp_seq=1 ttl=43 time=55.5 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=43 time=54.6 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=43 time=53.3 ms

--- 8.8.8.8 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2003ms
rtt min/avg/max/mdev = 53.368/54.534/55.564/0.941 ms'
```

TELNETLIB

TELNETLIB

В зависимости от модуля, преобразование между строками и байтами может выполняться автоматически, а может требоваться явно.

Например, в модуле telnetlib необходимо передавать байты в методах `read_until` и `write`:

```
import telnetlib
import time

t = telnetlib.Telnet('192.168.100.1')

t.read_until(b'Username:')
t.write(b'cisco\n')

t.read_until(b'Password:')
t.write(b'cisco\n')
t.write(b'sh ip int br\n')

time.sleep(5)

output = t.read_very_eager().decode('utf-8')
print(output)
```

РЕХРЕСТ

PEXPECT

Модуль `rexrest` как аргумент ожидает строку, а возвращает байты:

```
In [9]: import pexpect
```

```
In [10]: output = pexpect.run('ls -ls')
```

In [11]: output

```
Out[11]: b'total 8\r\n4 drwxr-xr-x 2 vagrant vagrant 4096 Aug 28 12:16 concurrent_futures\r\n4 drwxr-xr-x 2
```

```
In [12]: output.decode('utf-8')
```

```
Out[12]: 'total 8\r\n4 drwxr-xr-x 2 vagrant vagrant 4096 Aug 28 12:16 concurrent_futures\r\n4 drwxr-xr-x 2 \
```


PEXPECT ENCODING

И также поддерживает вариант передачи кодировки через параметр `encoding`:

```
In [13]: output = pexpect.run('ls -ls', encoding='utf-8')
```

```
In [14]: output
```

```
Out[14]: 'total 8\r\n4 drwxr-xr-x 2 vagrant vagrant 4096 Aug 28 12:16 concurrent_futures\r\n4 drwxr-xr-x 2 \'
```

4

▶

РАБОТА С ФАЙЛАМИ

РАБОТА С ФАЙЛАМИ

До сих пор при работе с файлами использовалась такая конструкция:

```
with open(filename) as f:  
    for line in f:  
        print(line)
```

КОДИРОВКА ПО УМОЛЧАНИЮ

При чтении файла происходит конвертация байт в строки. И при этом использовалась кодировка по умолчанию:

```
In [1]: import locale
```

```
In [2]: locale.getpreferredencoding()
```

```
Out[2]: 'UTF-8'
```

КОДИРОВКА ПО УМОЛЧАНИЮ

Кодировка по умолчанию в файле:

```
In [2]: f = open('r1.txt')
```

```
In [3]: f
```

```
Out[3]: <_io.TextIOWrapper name='r1.txt' mode='r' encoding='UTF-8'>
```

ЯВНОЕ ЗАДАНИЕ КОДИРОВКИ

При работе с файлами лучше явно указывать кодировку, так как в разных ОС она может отличаться:

```
In [4]: with open('r1.txt', encoding='utf-8') as f:
...:     for line in f:
...:         print(line, end='')
...:
!
service timestamps debug datetime msec localtime show-timezone year
service timestamps log datetime msec localtime show-timezone year
service password-encryption
service sequence-numbers
!
no ip domain lookup
!
ip ssh version 2
!
```

ОШИБКИ ПРИ КОНВЕРТАЦИИ

ОШИБКИ ПРИ КОНВЕРТАЦИИ

При конвертации между строками и байтами очень важно точно знать, какая кодировка используется, а также знать о возможностях разных кодировок.

ОШИБКИ

Например, кодировка ASCII не может преобразовать в байты кириллицу:

```
In [32]: hi_unicode = 'привет'

In [33]: hi_unicode.encode('ascii')
-----
UnicodeEncodeError                                Traceback (most recent call last)
<ipython-input-33-ec69c9fd2dae> in <module>()
----> 1 hi_unicode.encode('ascii')

UnicodeEncodeError: 'ascii' codec can't encode characters in position 0-5: ordinal not in range(128)
```

ОШИБКИ

Аналогично, если строка "привет" преобразована в байты, и попробовать преобразовать ее в строку с помощью `ascii`, тоже получим ошибку:

```
In [34]: hi_unicode = 'привет'

In [35]: hi_bytes = hi_unicode.encode('utf-8')

In [36]: hi_bytes.decode('ascii')
-----
UnicodeDecodeError                                Traceback (most recent call last)
<ipython-input-36-aa0ada5e44e9> in <module>()
----> 1 hi_bytes.decode('ascii')

UnicodeDecodeError: 'ascii' codec can't decode byte 0xd0 in position 0: ordinal not in range(128)
```

ОШИБКИ

Еще один вариант ошибки, когда используются разные кодировки для преобразований:

```
In [37]: de_hi_unicode = 'grüezi'

In [38]: utf_16 = de_hi_unicode.encode('utf-16')

In [39]: utf_16.decode('utf-8')
-----
UnicodeDecodeError                                Traceback (most recent call last)
<ipython-input-39-4b4c731e69e4> in <module>()
----> 1 utf_16.decode('utf-8')

UnicodeDecodeError: 'utf-8' codec can't decode byte 0xff in position 0: invalid start byte
```

НАДО ЗНАТЬ КАКАЯ КОДИРОВКА ИСПОЛЬЗОВАЛАСЬ

Но на самом деле, предыдущие ошибки - это хорошо. Они явно говорят, в чем проблема.

Хуже, когда получается так:

```
In [40]: hi_unicode = 'привет'

In [41]: hi_bytes = hi_unicode.encode('utf-8')

In [42]: hi_bytes
Out[42]: b'\xd0\xbf\xd1\x80\xd0\xb8\xd0\xb2\xd0\xb5\xd1\x82'

In [43]: hi_bytes.decode('utf-16')
Out[43]: '뽤帙례닐뽤苑'
```

ОБРАБОТКА ОШИБОК

ОБРАБОТКА ОШИБОК

У методов `encode` и `decode` есть режимы обработки ошибок, которые указывают, как реагировать на ошибку преобразования.

ENCODE REPLACE

По умолчанию encode использует режим 'strict' - при возникновении ошибок кодировки генерируется исключение UnicodeError. Примеры такого поведения были выше.

Режим replace заменит символ знаком вопроса:

```
In [44]: de_hi_unicode = 'grüezi'  
  
In [45]: de_hi_unicode.encode('ascii', 'replace')  
Out[45]: b'gr?ezi'
```

ENCODE NAMEREPLACE

Или `namereplace`, чтобы заменить символ именем:

```
In [46]: de_hi_unicode = 'grüezi'  
  
In [47]: de_hi_unicode.encode('ascii', 'namereplace')  
Out[47]: b'gr\\N{LATIN SMALL LETTER U WITH DIAERESIS}ezi'
```


ENCODE IGNORE

Кроме того, можно полностью игнорировать символы, которые нельзя закодировать:

```
In [48]: de_hi_unicode = 'grüezi'
```

```
In [49]: de_hi_unicode.encode('ascii', 'ignore')
```

```
Out[49]: b'grezi'
```

ПАРАМЕТР ERRORS В DECODE

В методе `decode` по умолчанию тоже используется режим `strict` и генерируется исключение `UnicodeDecodeError`.

DECODE IGNORE

Если изменить режим на ignore, как и в encode, символы будут просто игнорироваться:

```
In [50]: de_hi_unicode = 'grüezi'

In [51]: de_hi_utf8 = de_hi_unicode.encode('utf-8')

In [52]: de_hi_utf8
Out[52]: b'gr\xc3\xbcezi'

In [53]: de_hi_utf8.decode('ascii', 'ignore')
Out[53]: 'grezi'
```

DECODE REPLACE

Режим replace заменит символы:

```
In [54]: de_hi_unicode = 'grüezi'  
  
In [55]: de_hi_utf8 = de_hi_unicode.encode('utf-8')  
  
In [56]: de_hi_utf8.decode('ascii', 'replace')  
Out[56]: 'grøøezi'
```




1. Pragmatic Unicode
2. The Absolute Minimum Every Software Developer Absolutely, Positively Must Know About Unicode and Character Sets (No Excuses!)
3. Unicode HOWTO

