Zeichenkodierungen

Zeichenkodierungen

Datentyp bytes enthält byte:

- Wert mit acht Bit, d.h. zwischen 0 und 255
- Instanziierung mit b""

```
>>> type(b'r')
<class 'bytes'>

>>> type(b'@')
<class 'bytes'>
```

Zeichenkodierungen

Datentyp bytes enthält byte:

- Verhalten meist wie String
- aber weniger Zeichen möglich

```
>>> "Püthøn"
'Püthøn'

>>> b"Püthøn"
File "<stdin>", line 1
SyntaxError: bytes can only contain ASCII literal characters
```

Zeichenkodierungen

bytes

- braucht weniger Speicherplatz
- dient z.b. für Kommunikation mit externen Elementen, die nur ASCII verwenden

Zeichenkodierungen

Im Speicher

- keine Zeichen sondern Binärzahlen
- für Darstellung: Zeichensatztabellen / Codepages

Zeichenkodierungen

Im Speicher

- keine Zeichen sondern Binärzahlen
- für Darstellung: Zeichensatztabellen / Codepages

00	NULL	**10**	DLE	**20**	SP	**30**	0	**40**	@	**50**	P	**60**		**70**	p
01	SOH	**11**	DC1	**21**	!	**31**	1	**41**	A	**51**	Q	**61**	a	**71**	q
02	STX	**12**	DC2	**22**	"	**32**	2	**42**	B	**52**	R	**62**	b	**72**	r
03	ETX	**13**	DC3	**23**	#	**33**	3	**43**	C	**53**	S	**63**	c	**73**	s
04	EOT	**14**	DC4	**24**	$	**34**	4	**44**	D	**54**	T	**64**	d	**74**	t
05	ENQ	**15**	NAK	**25**	%	**35**	5	**45**	E	**55**	U	**65**	e	**75**	u
06	ACK	**16**	SYN	**26**	&	**36**	6	**46**	F	**56**	V	**66**	f	**76**	v
07	BEL	**17**	ETB	**27**	'	**37**	7	**47**	G	**57**	W	**67**	g	**77**	w
08	BS	**18**	CAN	**28**	(	**38**	8	**48**	H	**58**	X	**68**	h	**78**	x
09	HT	**19**	EM	**29**	)	**39**	9	**49**	I	**59**	Y	**69**	i	**79**	y
0A	LF	**1A**	SUB	**2A**	*	**3A**	:	**4A**	J	**5A**	Z	**6A**	j	**7A**	z
0B	VT	**1B**	ESC	**2B**	+	**3B**	;	**4B**	K	**5B**	[	**6B**	k	**7B**	{
0C	FF	**1C**	FS	**2C**	,	**3C**	<	**4C**	L	**5C**	\	**6C**	l	**7C**	|
0D	CR	**1D**	GS	**2D**	-	**3D**	=	**4D**	M	**5D**	]	**6D**	m	**7D**	}
0E	S0	**1E**	RS	**2E**	.	**3E**	>	**4E**	N	**5E**	^	**6E**	n	**7E**	~
0F	S1	**1F**	US	**2F**	/	**3F**	?	**4F**	O	**5F**	_	**6F**	o	**7F**	DEL

33103

Zeichenkodierungen

Kodierung von nicht-ASCII Zeichen in bytes

- erst String
- str.encode()-Methode

```
>>> str_string = "Püthøn"

>>> bytes_string = str_string.encode("iso-8859-15")

>>> bytes_string
b'P\xfcth\xf8n'
```

Zeichenkodierungen

`iso-8859-15` Kodierung

- nicht-ASCII Zeichen als Escape-Sequenz
- \x und Hexadezimalzahl

```
>>> str_string = "Püthøn"

>>> bytes_string = str_string.encode("iso-8859-15")

>>> bytes_string
b'P\xfcth\xf8n'
```

Zeichenkodierungen

Rückübersetzung zu String

- bytes.decode()-Methode

```
>>> bytes_string.decode("iso-8859-15")
'Püthøn'
```


```
>>> bytes_string.decode("iso-8859-15")
'Püthøn'
```

Zeichenkodierungen

Heute meist Zeichensätze mit mehr Platz pro Zeichen

- Nachteil: für Englisch nur winziger Bruchteil verwendet
- viel ungenutzter Platz

Zeichenkodierungen

Lösung: Unicode

- variable Kodierungslänge
- Tabelle Zeichen ↔️ Code
- Einträge Codepoints
- Notation: U+x

Zeichenkodierungen

Verbreitetster Unicode: UTF-8

- ASCII-Zeichen mit nur einem Byte
- Zeichen, die nicht auf Tastatur: Escapesequenzen mit \u + Hexadezimalzahl (aus Codetable)

```
>>> s = "\u20ac"
>>> print(s)
€
```

Zeichenkodierungen

Verbreitetster Unicode: UTF-8

- außerdem Name für jedes Zeichen
- Verwendung mit \N und Klammern

```
>>> "\N{euro sign}"
'€'
```

Zeichenkodierungen

Funktionen chr() und ord()

- konvertieren Unicode-Codepoints und entsprechende Zeichen ineinander

```
>>> chr(8364)
'€'

>>> ord("€")
8364
```

Zeichenkodierungen

What is the output of the following code?

```
ch = chr('a')
or_ = ord(ch)
print(or_, ch)
```

```
a) 97 a

b) TypeError: an integer is required (got type str)

c) TypeError: chr() takes exactly two arguments (1 given)

d) TypeError: ord() takes exactly two arguments (1 given)
```

Zeichenkodierungen

What is the output of the following code?

```
ch = chr(97)
or_ = ord(ch)
print(or_, ch)
```

```
a) 97 a

b) TypeError: an integer is required (got type str)

c) TypeError: chr() takes exactly two arguments (1 given)

d) TypeError: ord() takes exactly two arguments (1 given)
```