

ASCII 1/2

American Standard Code for Information Interchange

Dec	Hex	Oct	Bin	Char	Dec	Hex	Oct	Bin	Char	Dec	Hex	Oct	Bin	Char	Dec	Hex	Oct	Bin	Char
0	0x00	000	00000000	NUL	32	0x20	040	01000000	space	64	0x40	100	10000000	@	96	0x60	140	11000000	`
1	0x01	001	00000001	SOH	33	0x21	041	01000001	!	65	0x41	101	10000001	A	97	0x61	141	11000001	a
2	0x02	002	00000010	STX	34	0x22	042	01000010	"	66	0x42	102	10000010	B	98	0x62	142	11000010	b
3	0x03	003	00000011	ETX	35	0x23	043	01000011	#	67	0x43	103	10000011	C	99	0x63	143	11000011	c
4	0x04	004	00000100	EOT	36	0x24	044	01000100	\$	68	0x44	104	10000100	D	100	0x64	144	11000100	d
5	0x05	005	00000101	ENQ	37	0x25	045	01000101	%	69	0x45	105	10000101	E	101	0x65	145	11000101	e
6	0x06	006	00000110	ACK	38	0x26	046	01000110	&	70	0x46	106	10000110	F	102	0x66	146	11000110	f
7	0x07	007	00000111	BEL	39	0x27	047	01000111	'	71	0x47	107	10000111	G	103	0x67	147	11000111	g
8	0x08	010	00010000	BS	40	0x28	050	01010000	(72	0x48	110	10010000	H	104	0x68	150	11010000	h
9	0x09	011	00010001	TAB	41	0x29	051	01010001)	73	0x49	111	10010001	I	105	0x69	151	11010001	i
10	0x0A	012	00010010	LF	42	0x2A	052	01010010	*	74	0x4A	112	10010010	J	106	0x6A	152	11010010	j
11	0x0B	013	00010011	VT	43	0x2B	053	01010011	+	75	0x4B	113	10010011	K	107	0x6B	153	11010011	k
12	0x0C	014	00011000	FF	44	0x2C	054	01011000	,	76	0x4C	114	10011000	L	108	0x6C	154	11011000	l
13	0x0D	015	00011001	CR	45	0x2D	055	01011001	-	77	0x4D	115	10011001	M	109	0x6D	155	11011001	m
14	0x0E	016	00011010	SO	46	0x2E	056	01011010	.	78	0x4E	116	10011010	N	110	0x6E	156	11011010	n
15	0x0F	017	00011011	SI	47	0x2F	057	01011011	/	79	0x4F	117	10011011	O	111	0x6F	157	11011011	o
16	0x10	020	00100000	DLE	48	0x30	060	01100000	0	80	0x50	120	10100000	P	112	0x70	160	11100000	p
17	0x11	021	00100001	DC1	49	0x31	061	01100001	1	81	0x51	121	10100001	Q	113	0x71	161	11100001	q
18	0x12	022	00100010	DC2	50	0x32	062	01100010	2	82	0x52	122	10100010	R	114	0x72	162	11100010	r
19	0x13	023	00100011	DC3	51	0x33	063	01100011	3	83	0x53	123	10100011	S	115	0x73	163	11100011	s
20	0x14	024	00101000	DC4	52	0x34	064	01101000	4	84	0x54	124	10101000	T	116	0x74	164	11101000	t
21	0x15	025	00101001	NAK	53	0x35	065	01101001	5	85	0x55	125	10101001	U	117	0x75	165	11101001	u
22	0x16	026	00101010	SYN	54	0x36	066	01101010	6	86	0x56	126	10101010	V	118	0x76	166	11101010	v
23	0x17	027	00101011	ETB	55	0x37	067	01101011	7	87	0x57	127	10101011	W	119	0x77	167	11101011	w
24	0x18	030	00110000	CAN	56	0x38	070	01110000	8	88	0x58	130	10110000	X	120	0x78	170	11110000	x
25	0x19	031	00110001	EM	57	0x39	071	01110001	9	89	0x59	131	10110001	Y	121	0x79	171	11110001	y
26	0x1A	032	00110010	SUB	58	0x3A	072	01110010	:	90	0x5A	132	10110010	Z	122	0x7A	172	11110010	z
27	0x1B	033	00110011	ESC	59	0x3B	073	01110011	;	91	0x5B	133	10110011	[123	0x7B	173	11110011	{
28	0x1C	034	00111000	FS	60	0x3C	074	01111000	<	92	0x5C	134	10111000	\	124	0x7C	174	11111000	
29	0x1D	035	00111001	GS	61	0x3D	075	01111001	=	93	0x5D	135	10111001]	125	0x7D	175	11111001	}
30	0x1E	036	00111010	RS	62	0x3E	076	01111010	>	94	0x5E	136	10111010	^	126	0x7E	176	11111010	~
31	0x1F	037	00111011	US	63	0x3F	077	01111011	?	95	0x5F	137	10111011	_	127	0x7F	177	11111011	DEL

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

<http://www.catonmat.net/download/ascii-cheat-sheet.png>

Representing Simple Strings

- Each character is represented by a number between 0 and 256 stored in 8 bits of memory
- We refer to "8 bits of memory as a **"byte"** of memory – (i.e. my disk drive contains 3 Terabytes of memory)
- The **ord()** function tells us the numeric value of a simple ASCII character


```
>>> print(ord('H'))  
72  
>>> print(ord('e'))  
101  
>>> print(ord('\n'))  
10  
>>>
```

ASCII 2/2

```
>>> print(ord('H'))
72
>>> print(ord('e'))
101
>>> print(ord('\n'))
10
>>>
```

In the 1960s and 1970s,
we just assumed that
one byte was one
character

Dec	Hex	Oct	Bin	Char	Dec	Hex	Oct	Bin	Char	Dec	Hex	Oct	Bin	Char	Dec	Hex	Oct	Bin	Char
0	0x00	000	00000000	NUL	32	0x20	040	01000000	space	64	0x40	100	10000000	@	96	0x60	140	11000000	`
1	0x01	001	00000001	SOH	33	0x21	041	01000001	!	65	0x41	101	10000001	A	97	0x61	141	11000001	a
2	0x02	002	00000010	STX	34	0x22	042	01000010	"	66	0x42	102	10000010	B	98	0x62	142	11000010	b
3	0x03	003	00000011	ETX	35	0x23	043	01000011	#	67	0x43	103	10000011	C	99	0x63	143	11000011	c
4	0x04	004	00000100	EOT	36	0x24	044	01000100	\$	68	0x44	104	10000100	D	100	0x64	144	11000100	d
5	0x05	005	00000101	ENQ	37	0x25	045	01000101	%	69	0x45	105	10000101	E	101	0x65	145	11000101	e
6	0x06	006	00000110	ACK	38	0x26	046	01000110	&	70	0x46	106	10000110	F	102	0x66	146	11000110	f
7	0x07	007	00000111	BEL	39	0x27	047	01000111	'	71	0x47	107	10000111	G	103	0x67	147	11000111	g
8	0x08	010	00010000	BS	40	0x28	050	01010000	(72	0x48	110	10010000	H	104	0x68	150	11010000	h
9	0x09	011	00010001	TAB	41	0x29	051	01010001)	73	0x49	111	10010001	I	105	0x69	151	11010001	i
10	0x0A	012	00010010	LF	42	0x2A	052	01010010	*	74	0x4A	112	10010010	J	106	0x6A	152	11010010	j
11	0x0B	013	00010011	VT	43	0x2B	053	01010011	+	75	0x4B	113	10010011	K	107	0x6B	153	11010011	k
12	0x0C	014	00010100	FF	44	0x2C	054	01010100	,	76	0x4C	114	10010100	L	108	0x6C	154	11010100	l
13	0x0D	015	00010101	CR	45	0x2D	055	01010101	-	77	0x4D	115	10010101	M	109	0x6D	155	11010101	m
14	0x0E	016	00010110	SO	46	0x2E	056	01010110	.	78	0x4E	116	10010110	N	110	0x6E	156	11010110	n
15	0x0F	017	00010111	SI	47	0x2F	057	01010111	/	79	0x4F	117	10010111	O	111	0x6F	157	11010111	o
16	0x10	020	00100000	DLE	48	0x30	060	01100000	0	80	0x50	120	10100000	P	112	0x70	160	11100000	p
17	0x11	021	00100001	DC1	49	0x31	061	01100001	1	81	0x51	121	10100001	Q	113	0x71	161	11100001	q
18	0x12	022	00100010	DC2	50	0x32	062	01100010	2	82	0x52	122	10100010	R	114	0x72	162	11100010	r
19	0x13	023	00100011	DC3	51	0x33	063	01100011	3	83	0x53	123	10100011	S	115	0x73	163	11100011	s
20	0x14	024	00100100	DC4	52	0x34	064	01100100	4	84	0x54	124	10100100	T	116	0x74	164	11100100	t
21	0x15	025	00100101	NAK	53	0x35	065	01100101	5	85	0x55	125	10100101	U	117	0x75	165	11100101	u
22	0x16	026	00100110	SYN	54	0x36	066	01100110	6	86	0x56	126	10100110	V	118	0x76	166	11100110	v
23	0x17	027	00100111	ETB	55	0x37	067	01100111	7	87	0x57	127	10100111	W	119	0x77	167	11100111	w
24	0x18	030	00110000	CAN	56	0x38	070	01110000	8	88	0x58	130	10110000	X	120	0x78	170	11110000	x
25	0x19	031	00110001	EM	57	0x39	071	01110001	9	89	0x59	131	10110001	Y	121	0x79	171	11110001	y
26	0x1A	032	00110010	SUB	58	0x3A	072	01110010	:	90	0x5A	132	10110010	Z	122	0x7A	172	11110010	z
27	0x1B	033	00110011	ESC	59	0x3B	073	01110011	;	91	0x5B	133	10110011	[123	0x7B	173	11110011	{
28	0x1C	034	00110100	FS	60	0x3C	074	01110100	<	92	0x5C	134	10110100	\	124	0x7C	174	11110100	
29	0x1D	035	00110101	GS	61	0x3D	075	01110101	=	93	0x5D	135	10110101]	125	0x7D	175	11110101	}
30	0x1E	036	00110110	RS	62	0x3E	076	01110110	>	94	0x5E	136	10110110	^	126	0x7E	176	11110110	~
31	0x1F	037	00110111	US	63	0x3F	077	01110111	?	95	0x5F	137	10110111	_	127	0x7F	177	11110111	DEL



Code Charts

Home | Site Map | Search

Unicode 9.0 Character Code Charts

SCRIPTS | SYMBOLS | NOTES

Find chart by hex code: Related links: [Name index](#) [Help & links](#)

Scripts

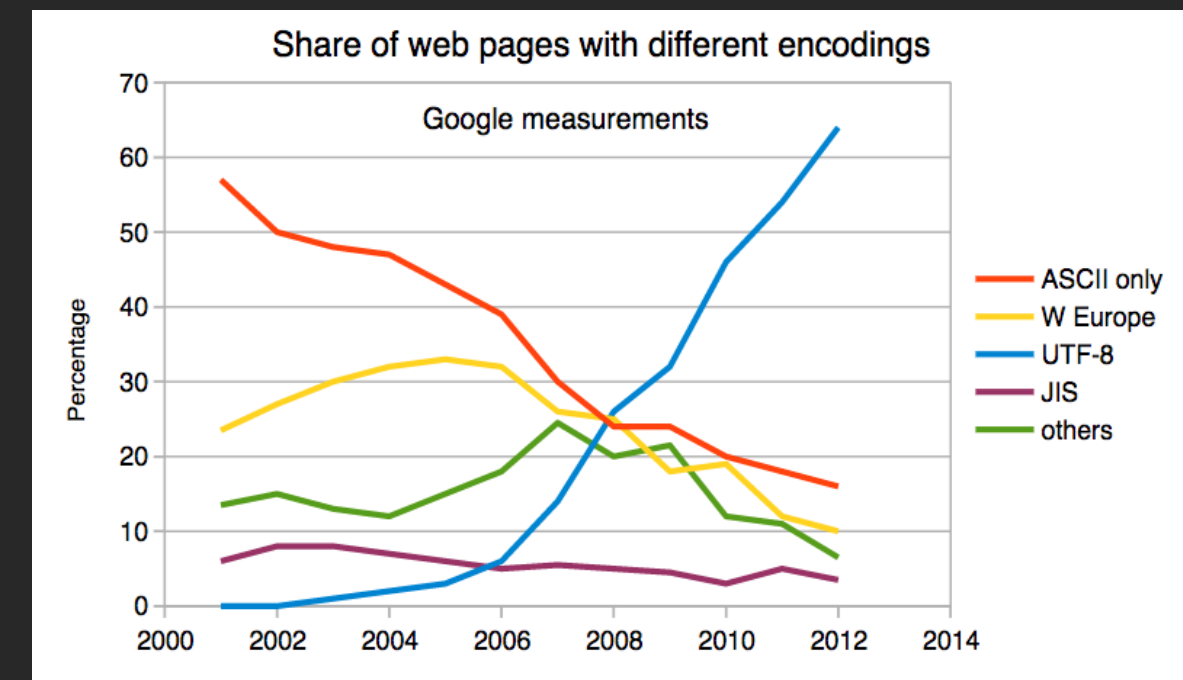
European Scripts	African Scripts	South Asian Scripts	Indonesia & Oceania Scripts
Armenian	Adlam	Ahom	Balinese
Armenian Ligatures	Bamum	Bengali and Assamese	Batak
Caucasian Albanian	Bamum Supplement	Bhaiksuki	Buginese
Cypriot Syllabary	Bassa Vah	Brahmi	Buhid
Cyrillic	Coptic	Chakma	Hanunoo
Cyrillic Supplement	Coptic in Greek block	Devanagari	Javanese
Cyrillic Extended-A	Coptic Epact Numbers	Devanagari Extended	Rejang
Cyrillic Extended-B	Egyptian Hieroglyphs (1MB)	Grantha	Sundanese
Cyrillic Extended-C	Ethiopic	Gujarati	Sundanese Supplement
Elbasan	Ethiopic Supplement	Gurmukhi	Tagalog
Georgian	Ethiopic Extended	Kaithi	Tagbanwa
Georgian Supplement	Ethiopic Extended-A	Kannada	East Asian Scripts
Glagolitic	Mende Kikakui	Kharoshthi	Bopomofo
Glagolitic Supplement	Meroitic	Khojki	Bopomofo Extended
Gothic	Meroitic Cursive	Khudawadi	CJK Unified Ideographs (Han) (35MB)
Greek	Meroitic Hieroglyphs	Lepcha	CJK Extension-A (6MB)
Greek Extended	N'Ko	Limbu	CJK Extension B (40MB)
Ancient Greek Numbers	Osmanya	Mahajani	CJK Extension C (3MB)
Latin	Tifinagh	Malayalam	CJK Extension D
Basic Latin (ASCII)	Vai	Meetei Mayek	CJK Extension E (3.5MB)
Latin-1 Supplement	Middle Eastern Scripts	Meetei Mayek Extensions	(see also UniHan Database)
Latin Extended-A	Anatolian Hieroglyphs	Modi	CJK Compatibility Ideographs

http://unicode.org/charts/

Multi-Byte Characters

- To represent the wide range of characters computers must handle we represent characters with more than one byte
 - UTF-16 – Variable length – Two or Four Bytes
 - UTF-32 – Fixed Length – Four Bytes
 - **UTF-8** – 1-4 bytes
 - *Upwards compatible with ASCII*
 - *Automatic detection between ASCII and UTF-8*
 - *UTF-8 is recommended practice for encoding data to be exchanged between systems*

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



Two Kinds of Strings in Python

Python 2.7.10

```
>>> x = '이광춘'
>>> type(x)
<type 'str'>
>>> x = u'이광춘'
>>> type(x)
<type 'unicode'>
>>>
```

Python 3.5.1

```
>>> x = '이광춘'
>>> type(x)
<class 'str'>
>>> x = u'이광춘'
>>> type(x)
<class 'str'>
>>>
```

In Python 3, all strings are Unicode

Python 2 Versus Python 3

Python 2.7.10

```
>>> x = b'abc'
```

```
>>> type(x)
```

```
<type 'str'>
```

```
>>> x = '이광춘'
```

```
>>> type(x)
```

```
<type 'str'>
```

```
>>> x = u'이광춘'
```

```
>>> type(x)
```

```
<type 'unicode'>
```

Python 3.5.1

```
>>> x = b'abc'
```

```
>>> type(x)
```

```
<class 'bytes'>
```

```
>>> x = '이광춘'
```

```
>>> type(x)
```

```
<class 'str'>
```

```
>>> x = u'이광춘'
```

```
>>> type(x)
```

```
<class 'str'>
```

Python 3 and Unicode

In Python 3, all strings internally are UNICODE

Working with string variables in Python programs and reading data from files usually "just works"

When we talk to a network resource using sockets or talk to a database we have to encode and decode data (usually to UTF-8)

Python 3.5.1

```
>>> x = b'abc'
```

```
>>> type(x)
```

```
<class 'bytes'>
```

```
>>> x = '이광춘'
```

```
>>> type(x)
```

```
<class 'str'>
```

```
>>> x = u'이광춘'
```

```
>>> type(x)
```

```
<class 'str'>
```


Python Strings to Bytes

- When we talk to an external resource like a network socket we send bytes, so we need to encode Python 3 strings into a given character encoding
- When we read data from an external resource, we must decode it based on the character set so it is properly represented in Python 3 as a string

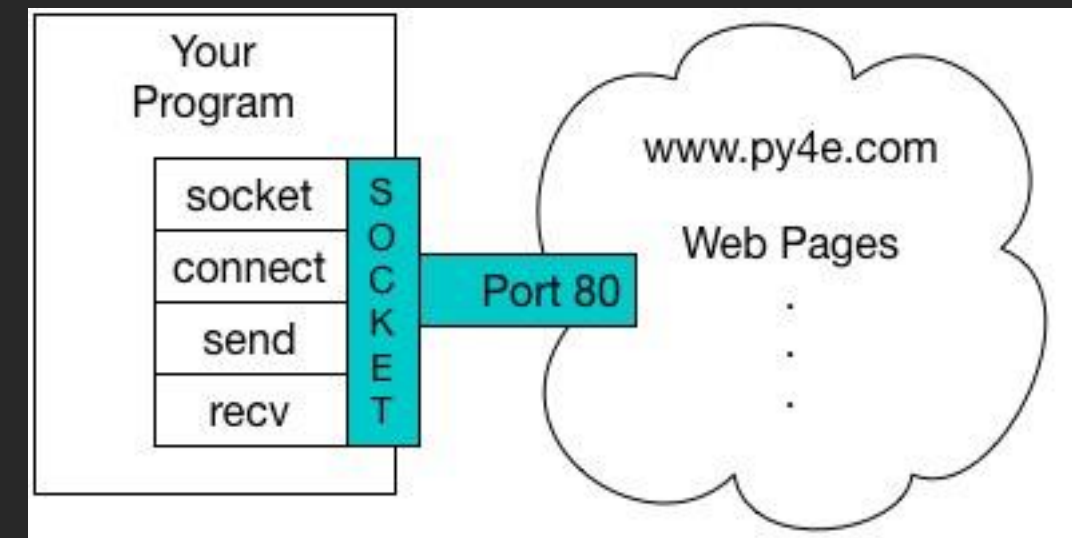
```
while True:
    data = mysock.recv(512)
    if ( len(data) < 1 ) :
        break
    mystring = data.decode()
    print(mystring)
```

An HTTP Request in Python 1/3

```
import socket

mysock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
mysock.connect(('data.pr4e.org', 80))
cmd = 'GET http://data.pr4e.org/romeo.txt HTTP/1.0\n\n'.encode()
mysock.send(cmd)

while True:
    data = mysock.recv(512)
    if (len(data) < 1):
        break
    print(data.decode())
mysock.close()
```



```
bytes.decode(encoding="utf-8", errors="strict")
```

```
bytearray.decode(encoding="utf-8", errors="strict")
```

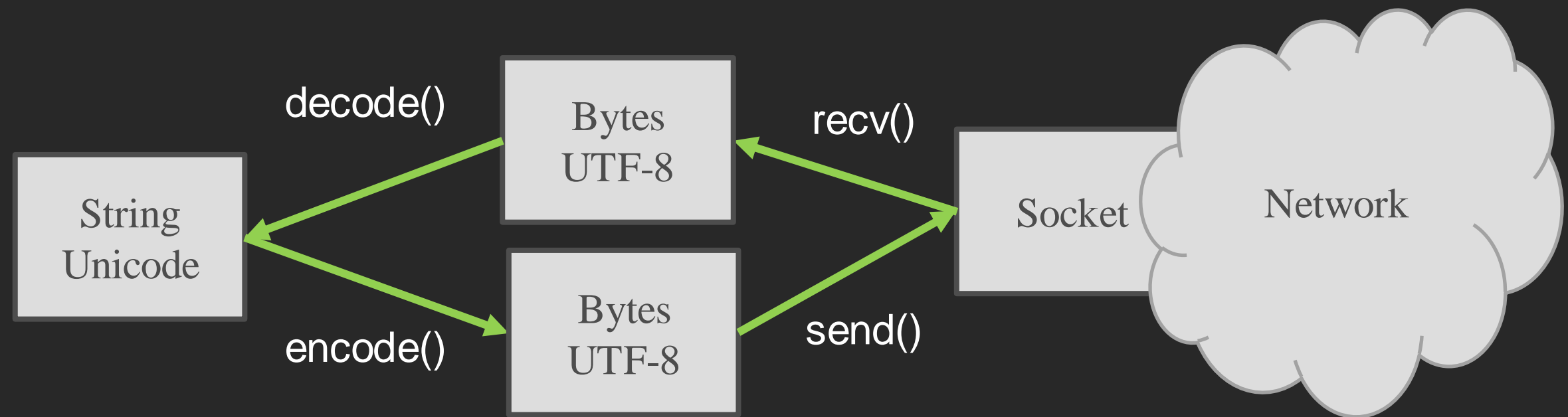
Return a string decoded from the given bytes. Default encoding is 'utf-8'. *errors* may be given to set a different error handling scheme. The default for *errors* is 'strict', meaning that encoding errors raise a `UnicodeError`. Other possible values are 'ignore', 'replace' and any other name registered via `codecs.register_error()`, see section [Error Handlers](#). For a list of possible encodings, see section [Standard Encodings](#).

```
str.encode(encoding="utf-8", errors="strict")
```

Return an encoded version of the string as a bytes object. Default encoding is 'utf-8'. *errors* may be given to set a different error handling scheme. The default for *errors* is 'strict', meaning that encoding errors raise a `UnicodeError`. Other possible values are 'ignore', 'replace', 'xmlcharrefreplace', 'backslashreplace' and any other name registered via `codecs.register_error()`, see section [Error Handlers](#). For a list of possible encodings, see section [Standard Encodings](#).

<https://docs.python.org/3/library/stdtypes.html#bytes.decode>

<https://docs.python.org/3/library/stdtypes.html#str.encode>



```
import socket
```

```
mysock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

```
mysock.connect(('data.pr4e.org', 80))
```

```
cmd = 'GET http://data.pr4e.org/romeo.txt HTTP/1.0\n\n'.encode()
```

```
mysock.send(cmd)
```

```
while True:
```

```
    data = mysock.recv(512)
```

```
    if (len(data) < 1):
```

```
        break
```

```
    print(data.decode())
```

```
mysock.close()
```


Making HTTP (even) Easier With urllib



Acknowledgements / Contributions



These slides are Copyright 2010- Charles R. Severance (www.dr-chuck.com) of the University of Michigan School of Information and open.umich.edu and made available under a Creative Commons Attribution 4.0 License. Please maintain this last slide in all copies of the document to comply with the attribution requirements of the license. If you make a change, feel free to add your name and organization to the list of contributors on this page as you republish the materials.

Initial Development: Charles Severance, University of Michigan School of Information

... Insert new Contributors here