Byte and bytearray

Byte and Bytearray

- Python comes with three string object types one for textual data and two for binary data:
  - str for representing decoded Unicode text (including ASCII)
  - bytes for representing binary data (including encoded text)
  - bytearray, a mutable flavor of the bytes type

Byte and Bytearray

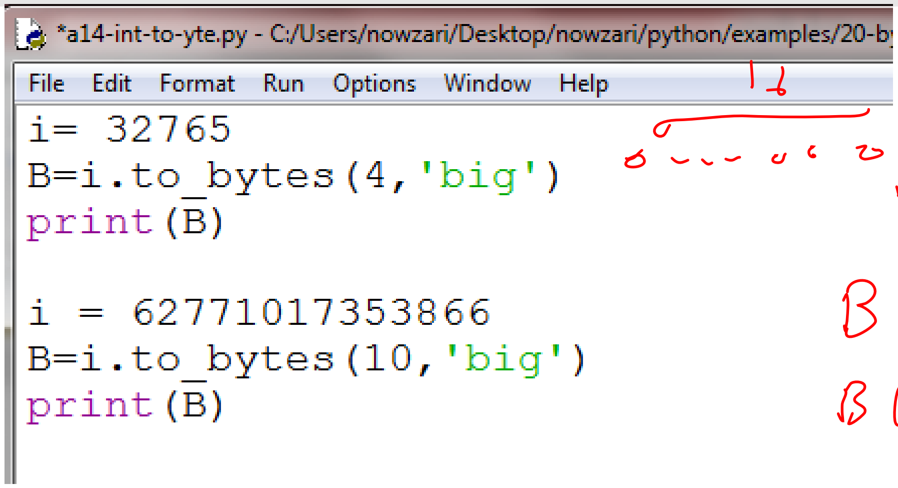
- A bytes object stores a mutable sequence of integers that are in the range 0 to 255. Unlike string objects, indexing a bytes object returns an integer.
- A byte type is immutable
- A bytearray type is a mutable version of byte type

Integers to byte

- We can convert the integers to bytes using the conversion methods:
- int.to_bytes(*length*, *byteorder*, *signed=False*)
- int.from_bytes(*bytes*, *byteorder*, *signed=False*)

- Byteorder=big or little

Integers to byte


*a14-int-to-yte.py - C:/Users/nowzari/Desktop/nowzari/python/examples/20-b
File Edit Format Run Options Window Help

```
i= 32765
B=i.to_bytes(4,'big')
print(B)

i = 62771017353866
B=i.to_bytes(10,'big')
print(B)
```

$\overbrace{0 \cdots 0}^{16},\ \overbrace{0111\ 1111,\ 1111\ 1101}^{16}$

B[0]

B[1]

```
b'\x00\x00\x7f\xfd'
b'\x00\x00\x00\x009\x17\x041\n\x8a'
```

Strings

- **string: A sequence of text characters in a program.**
  - Strings start and end with quotation mark `"` or apostrophe `'` characters.
  - Examples:

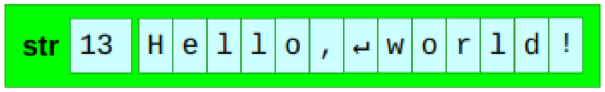
```
"This, too, is a string.   It can be very long!"
```

Strings

- **string: A sequence of text characters in a program.**
  - Strings start and end with quotation mark " or apostrophe ' characters.
  - Examples:

```
"This, too, is a string.   It can be very long!“
 str=“Hello, world!”
```

- **String representation**



- Strings are Immutable
- String type is str
- Each characters of a string is a cod point (0 to 0x10FFFF)

Indexes

- **Characters in a string are numbered with *indexes* starting at 0:**
  - Example:
    ```
    name = "P. Diddy"
    ```

index	0	1	2	3	4	5	6	7
character	P	.		D	i	d	d	y

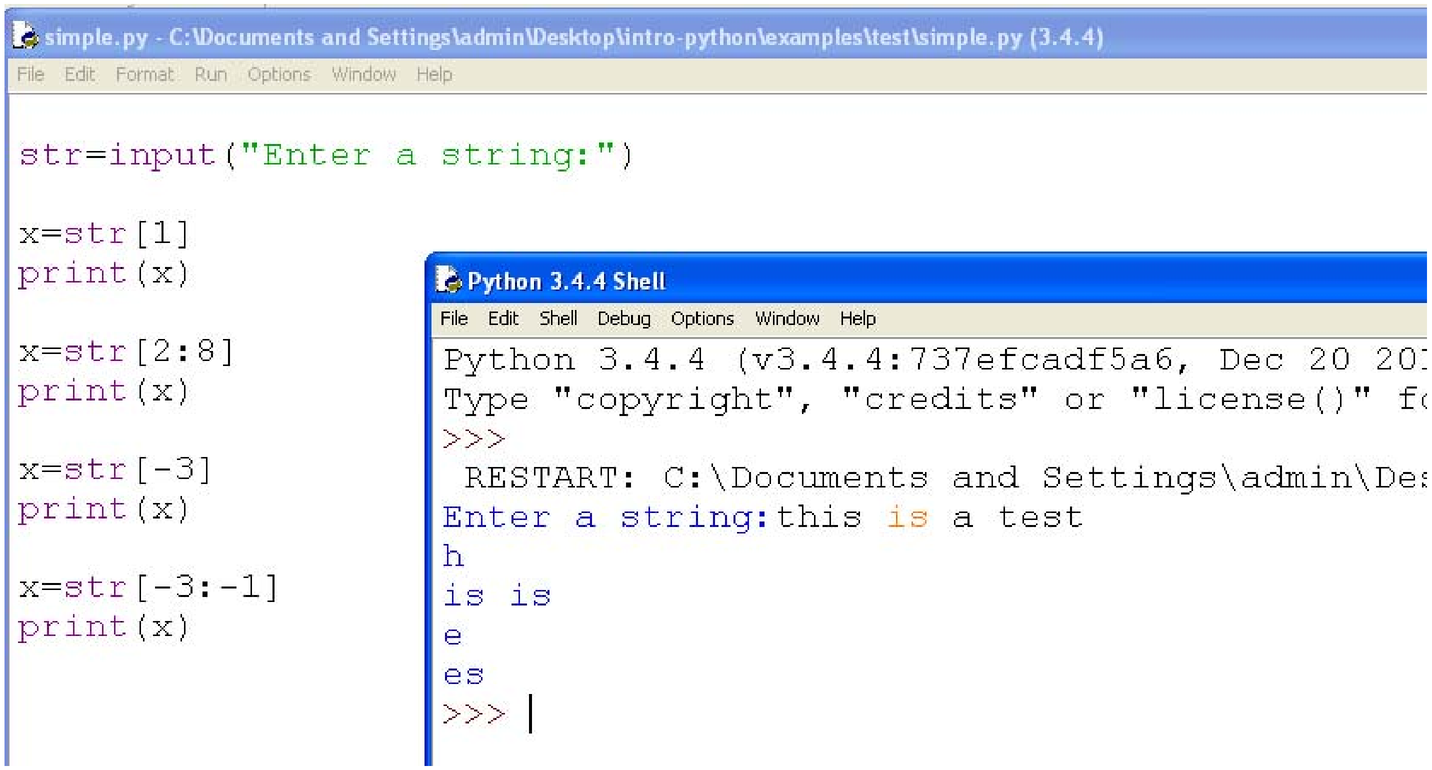
- **Accessing an individual character of a string:**

  variableName [***index***]

  - Example:
    ```
    print (name, "starts with", name[0])
    ```

    Output:
    ```
    P. Diddy starts with P
    ```

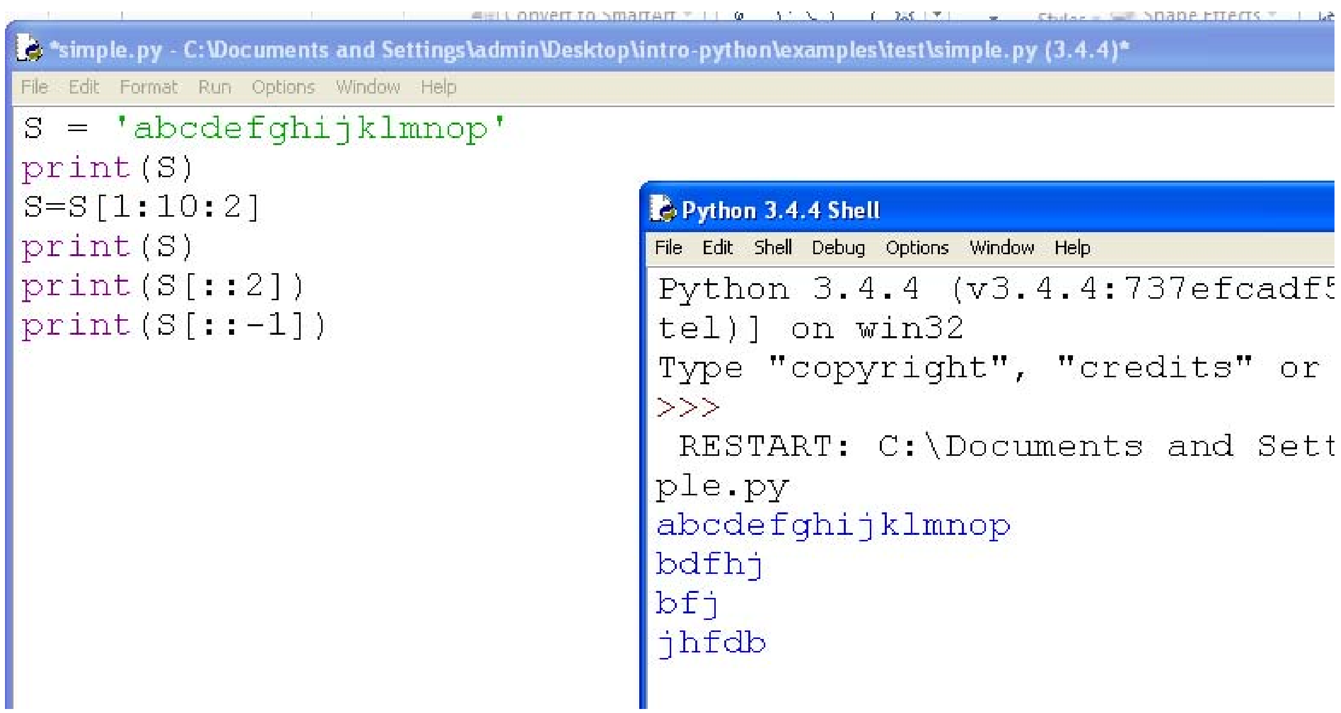
Indexes


simple.py - C:\Documents and Settings\admin\Desktop\intro-python\examples\test\simple.py (3.4.4)
File Edit Format Run Options Window Help
Python 3.4.4 Shell
File Edit Shell Debug Options Window Help

```python
str=input("Enter a string:")

x=str[1]
print(x)

x=str[2:8]
print(x)

x=str[-3]
print(x)

x=str[-3:-1]
print(x)
```

```
Python 3.4.4 (v3.4.4:737efcadf5a6, Dec 20 20
Type "copyright", "credits" or "license()" f
>>>
 RESTART: C:\Documents and Settings\admin\De
Enter a string:this is a test
h
is is
e
es
>>> |
```

Indexes

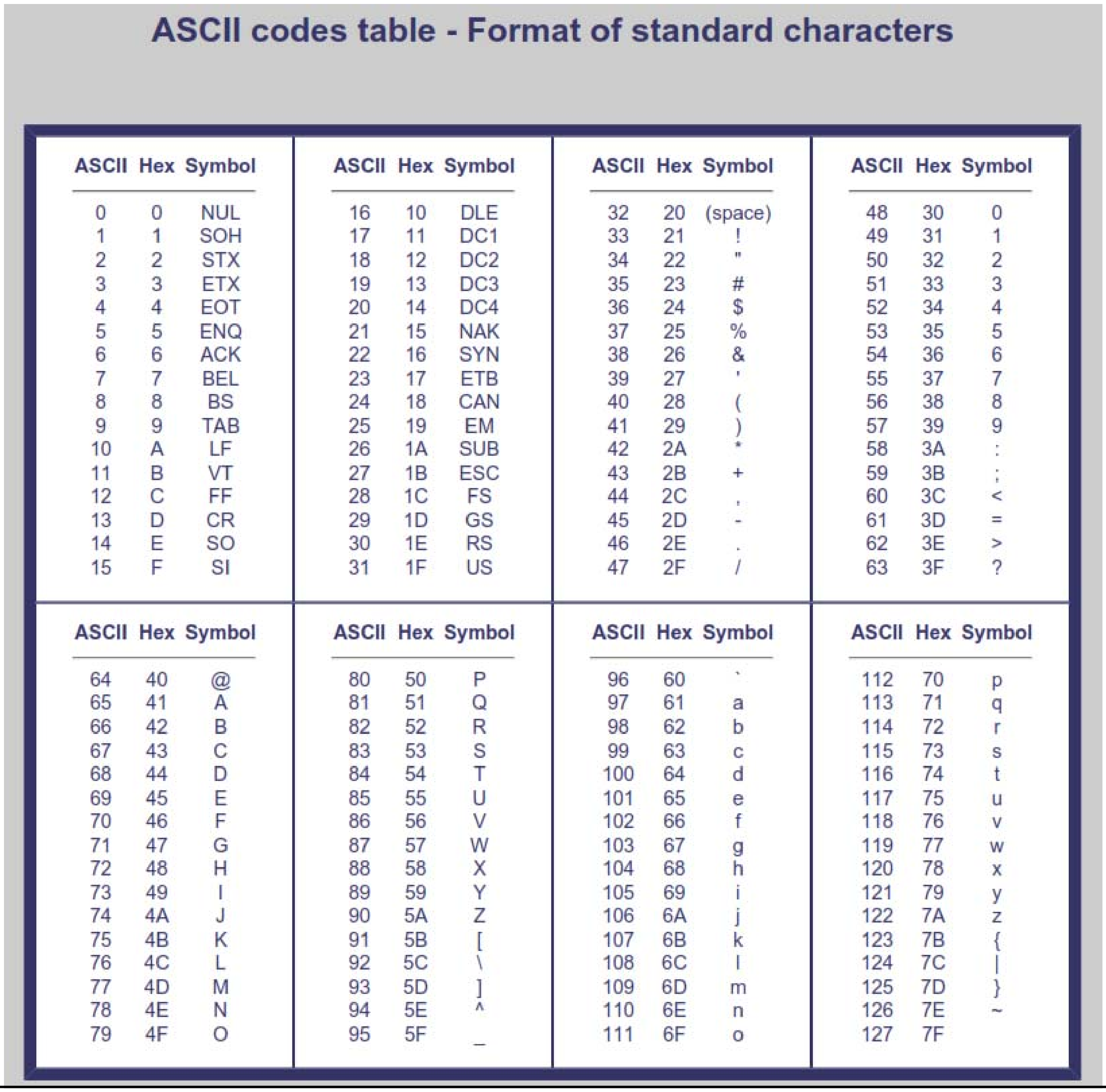


simple.py - C:\Documents and Settings\admin\Desktop\intro-python\examples\test\simple.py (3.4.4)
File Edit Format Run Options Window Help

```
S = 'abcdefghijklmnop'
print(S)
S=S[1:10:2]
print(S)
print(S[::2])
print(S[::-1])
```

Python 3.4.4 Shell
File Edit Shell Debug Options Window Help

```
Python 3.4.4 (v3.4.4:737efcadf5
tel)] on win32
Type "copyright", "credits" or
>>>
 RESTART: C:\Documents and Sett
ple.py
abcdefghijklmnop
bdfhj
bfj
jhfdb
```

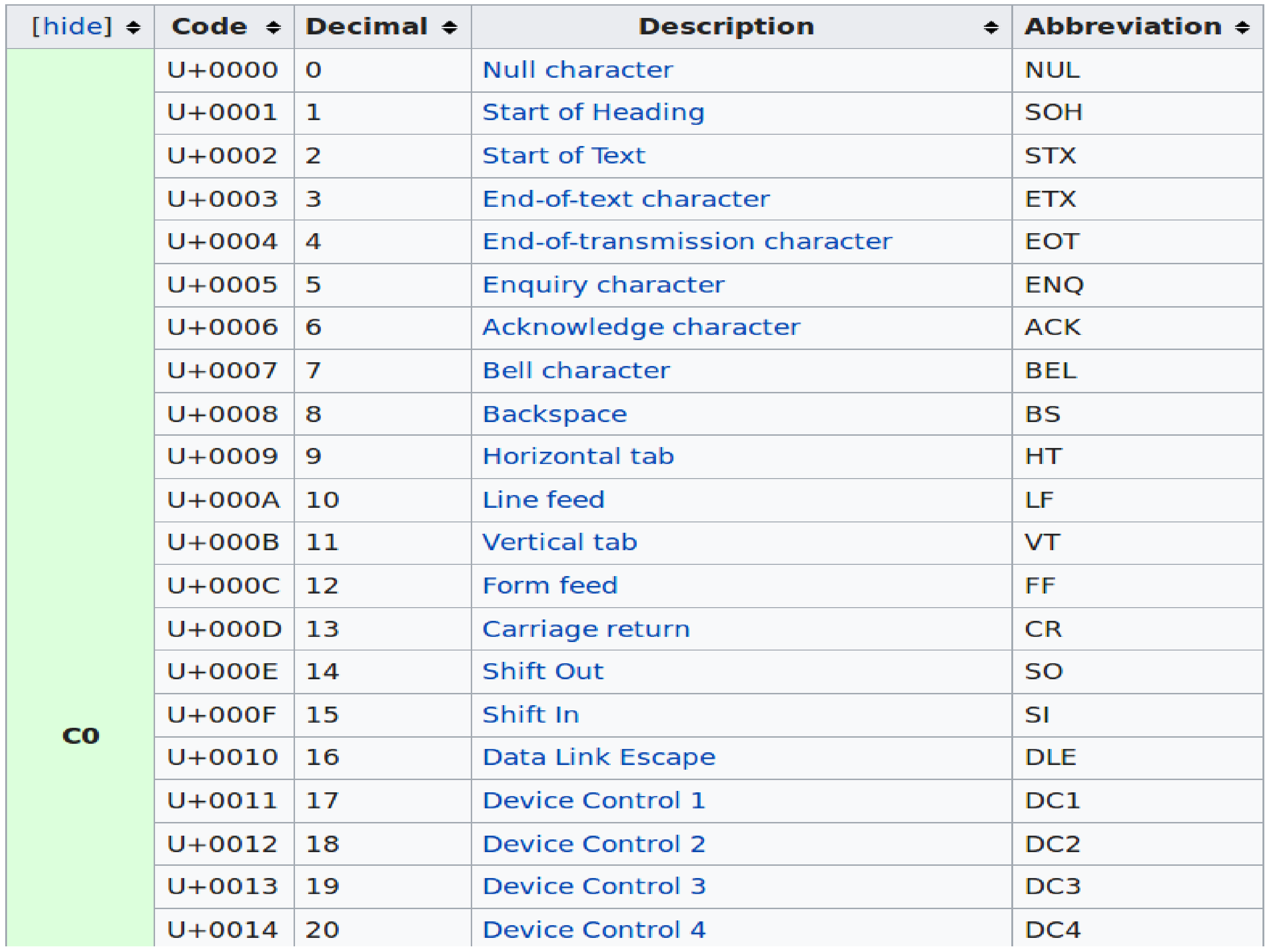


ASCII codes table - Format of standard characters

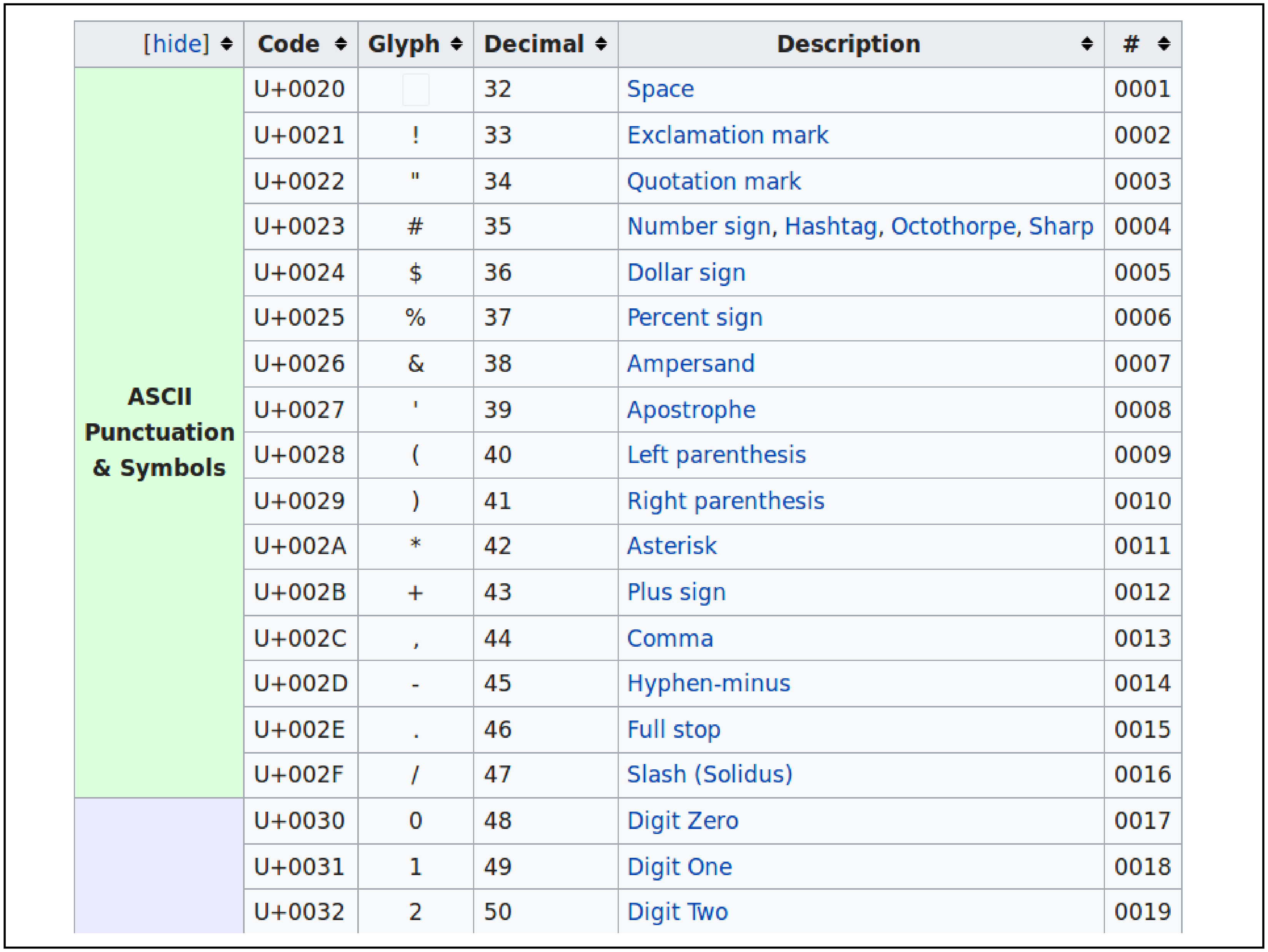
ASCII	Hex	Symbol	ASCII	Hex	Symbol	ASCII	Hex	Symbol	ASCII	Hex	Symbol
0	0	NUL	16	10	DLE	32	20	(space)	48	30	0
1	1	SOH	17	11	DC1	33	21	!	49	31	1
2	2	STX	18	12	DC2	34	22	"	50	32	2
3	3	ETX	19	13	DC3	35	23	#	51	33	3
4	4	EOT	20	14	DC4	36	24	$	52	34	4
5	5	ENQ	21	15	NAK	37	25	%	53	35	5
6	6	ACK	22	16	SYN	38	26	&	54	36	6
7	7	BEL	23	17	ETB	39	27	'	55	37	7
8	8	BS	24	18	CAN	40	28	(	56	38	8
9	9	TAB	25	19	EM	41	29	)	57	39	9
10	A	LF	26	1A	SUB	42	2A	*	58	3A	:
11	B	VT	27	1B	ESC	43	2B	+	59	3B	;
12	C	FF	28	1C	FS	44	2C	,	60	3C	<
13	D	CR	29	1D	GS	45	2D	-	61	3D	=
14	E	SO	30	1E	RS	46	2E	.	62	3E	>
15	F	SI	31	1F	US	47	2F	/	63	3F	?

ASCII	Hex	Symbol	ASCII	Hex	Symbol	ASCII	Hex	Symbol	ASCII	Hex	Symbol
64	40	@	80	50	P	96	60	`	112	70	p
65	41	A	81	51	Q	97	61	a	113	71	q
66	42	B	82	52	R	98	62	b	114	72	r
67	43	C	83	53	S	99	63	c	115	73	s
68	44	D	84	54	T	100	64	d	116	74	t
69	45	E	85	55	U	101	65	e	117	75	u
70	46	F	86	56	V	102	66	f	118	76	v
71	47	G	87	57	W	103	67	g	119	77	w
72	48	H	88	58	X	104	68	h	120	78	x
73	49	I	89	59	Y	105	69	i	121	79	y
74	4A	J	90	5A	Z	106	6A	j	122	7A	z
75	4B	K	91	5B	[	107	6B	k	123	7B	{
76	4C	L	92	5C	\	108	6C	l	124	7C	|
77	4D	M	93	5D	]	109	6D	m	125	7D	}
78	4E	N	94	5E	^	110	6E	n	126	7E	~
79	4F	O	95	5F	_	111	6F	o	127	7F	

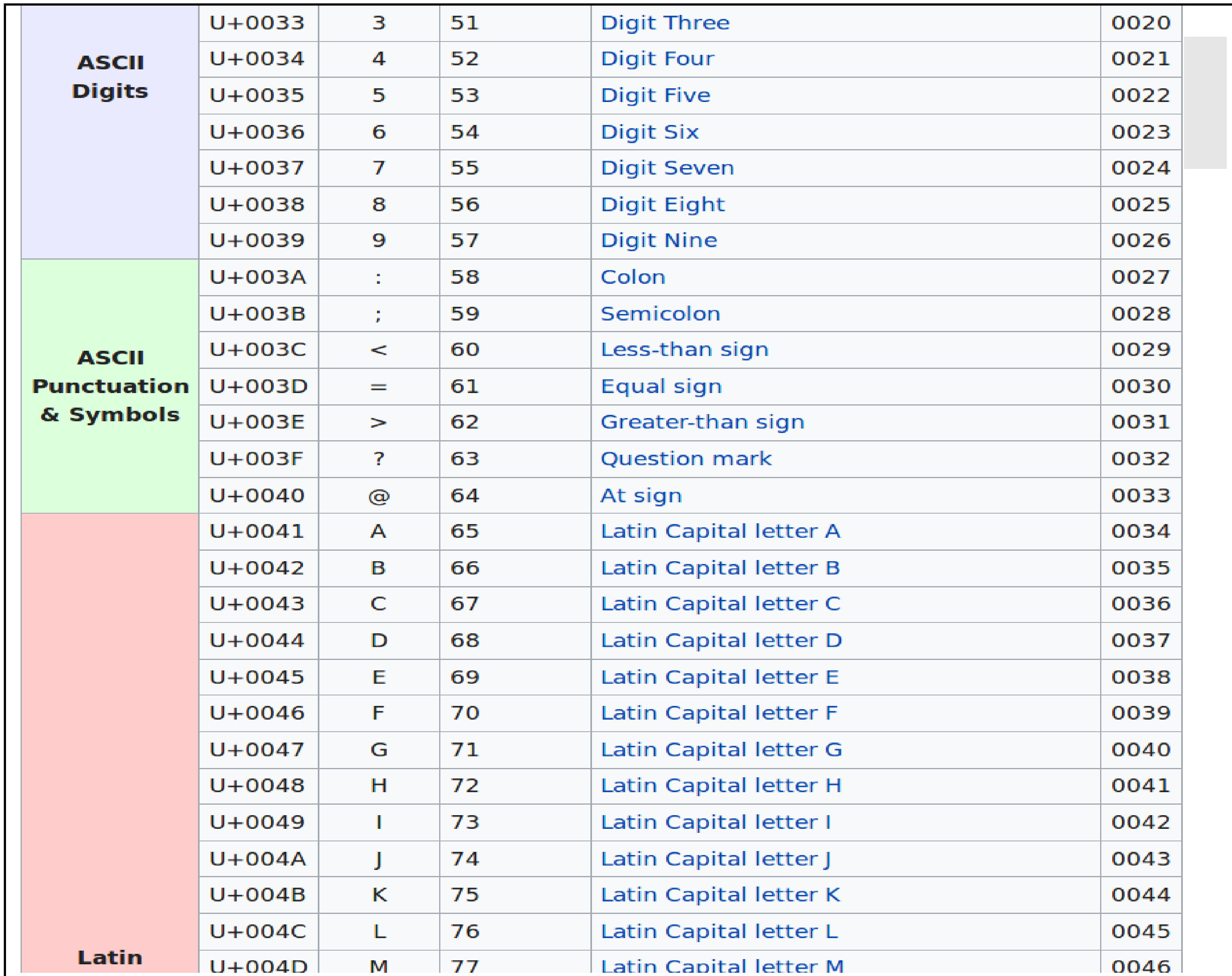
Unicode



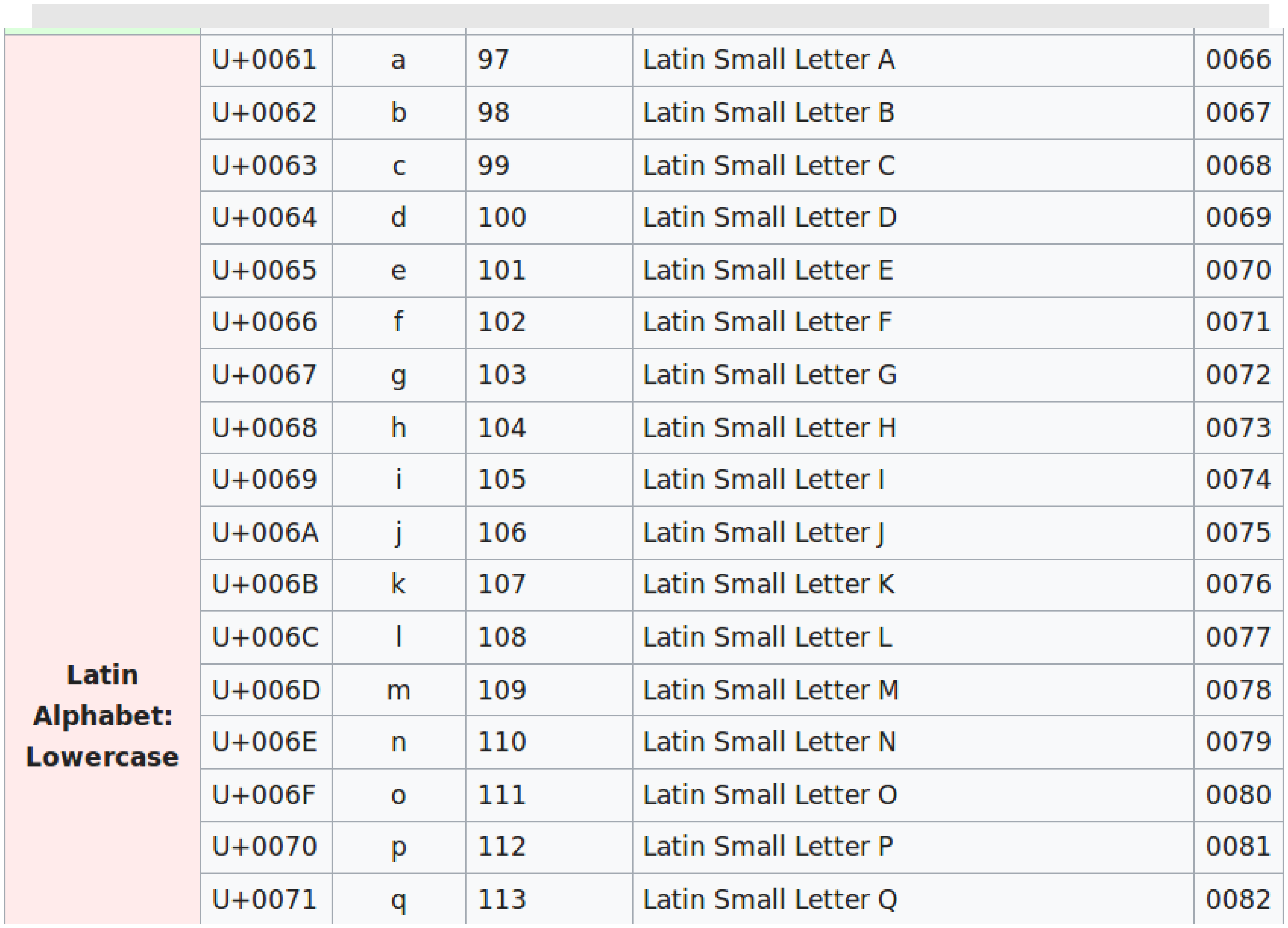
[hide]	Code	Decimal	Description	Abbreviation
C0	U+0000	0	Null character	NUL
	U+0001	1	Start of Heading	SOH
	U+0002	2	Start of Text	STX
	U+0003	3	End-of-text character	ETX
	U+0004	4	End-of-transmission character	EOT
	U+0005	5	Enquiry character	ENQ
	U+0006	6	Acknowledge character	ACK
	U+0007	7	Bell character	BEL
	U+0008	8	Backspace	BS
	U+0009	9	Horizontal tab	HT
	U+000A	10	Line feed	LF
	U+000B	11	Vertical tab	VT
	U+000C	12	Form feed	FF
	U+000D	13	Carriage return	CR
	U+000E	14	Shift Out	SO
	U+000F	15	Shift In	SI
	U+0010	16	Data Link Escape	DLE
	U+0011	17	Device Control 1	DC1
	U+0012	18	Device Control 2	DC2
	U+0013	19	Device Control 3	DC3
	U+0014	20	Device Control 4	DC4



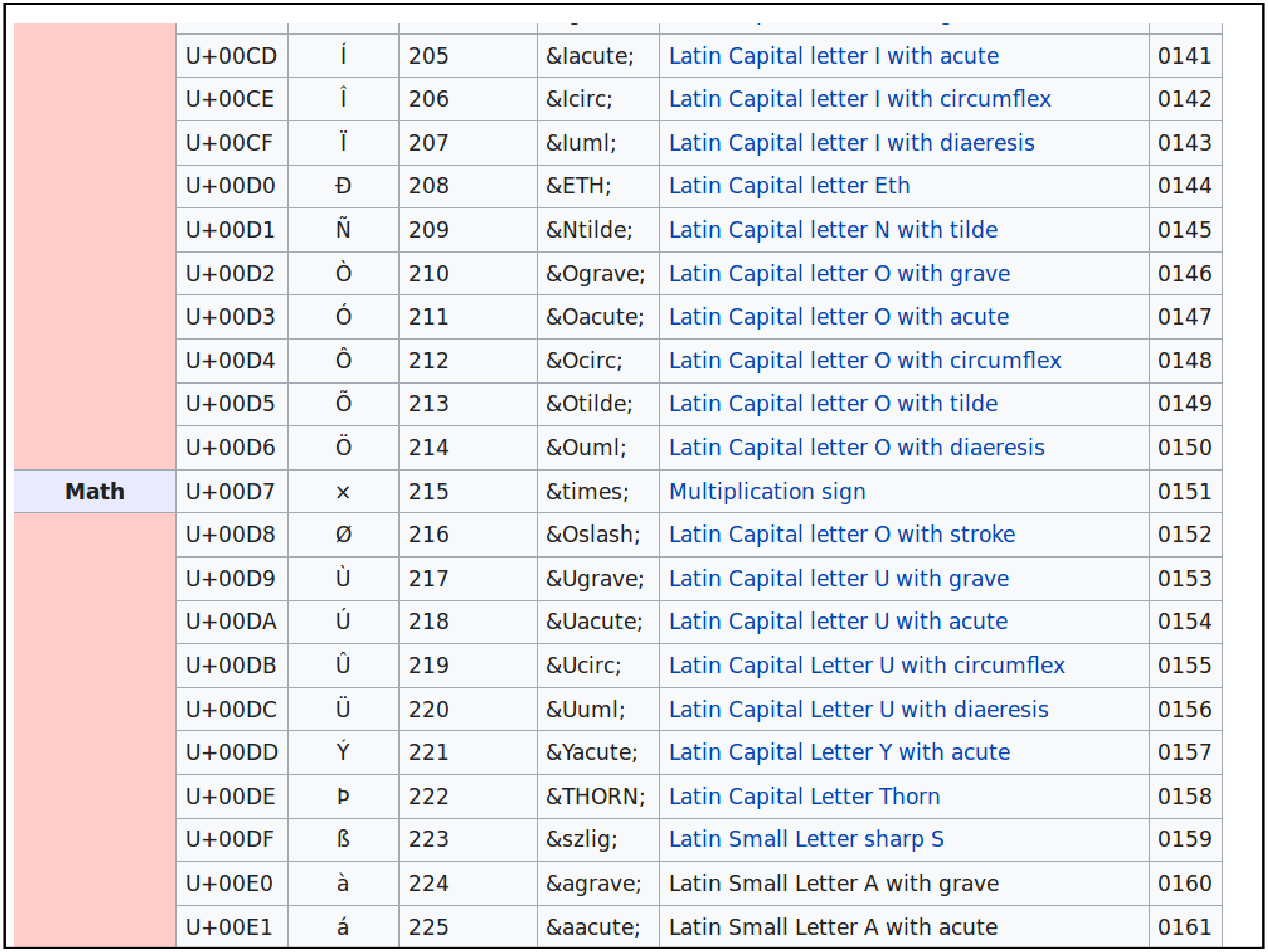
[hide]	Code	Glyph	Decimal	Description	#
ASCII Punctuation & Symbols	U+0020		32	Space	0001
	U+0021	!	33	Exclamation mark	0002
	U+0022	"	34	Quotation mark	0003
	U+0023	#	35	Number sign, Hashtag, Octothorpe, Sharp	0004
	U+0024	$	36	Dollar sign	0005
	U+0025	%	37	Percent sign	0006
	U+0026	&	38	Ampersand	0007
	U+0027	'	39	Apostrophe	0008
	U+0028	(	40	Left parenthesis	0009
	U+0029	)	41	Right parenthesis	0010
	U+002A	*	42	Asterisk	0011
	U+002B	+	43	Plus sign	0012
	U+002C	,	44	Comma	0013
	U+002D	-	45	Hyphen-minus	0014
	U+002E	.	46	Full stop	0015
	U+002F	/	47	Slash (Solidus)	0016
	U+0030	0	48	Digit Zero	0017
	U+0031	1	49	Digit One	0018
	U+0032	2	50	Digit Two	0019



ASCII Digits	U+0033	3	51	Digit Three	0020
	U+0034	4	52	Digit Four	0021
	U+0035	5	53	Digit Five	0022
	U+0036	6	54	Digit Six	0023
	U+0037	7	55	Digit Seven	0024
	U+0038	8	56	Digit Eight	0025
	U+0039	9	57	Digit Nine	0026
ASCII Punctuation & Symbols	U+003A	:	58	Colon	0027
	U+003B	;	59	Semicolon	0028
	U+003C	<	60	Less-than sign	0029
	U+003D	=	61	Equal sign	0030
	U+003E	>	62	Greater-than sign	0031
	U+003F	?	63	Question mark	0032
	U+0040	@	64	At sign	0033
Latin	U+0041	A	65	Latin Capital letter A	0034
	U+0042	B	66	Latin Capital letter B	0035
	U+0043	C	67	Latin Capital letter C	0036
	U+0044	D	68	Latin Capital letter D	0037
	U+0045	E	69	Latin Capital letter E	0038
	U+0046	F	70	Latin Capital letter F	0039
	U+0047	G	71	Latin Capital letter G	0040
	U+0048	H	72	Latin Capital letter H	0041
	U+0049	I	73	Latin Capital letter I	0042
	U+004A	J	74	Latin Capital letter J	0043
	U+004B	K	75	Latin Capital letter K	0044
	U+004C	L	76	Latin Capital letter L	0045
	U+004D	M	77	Latin Capital letter M	0046



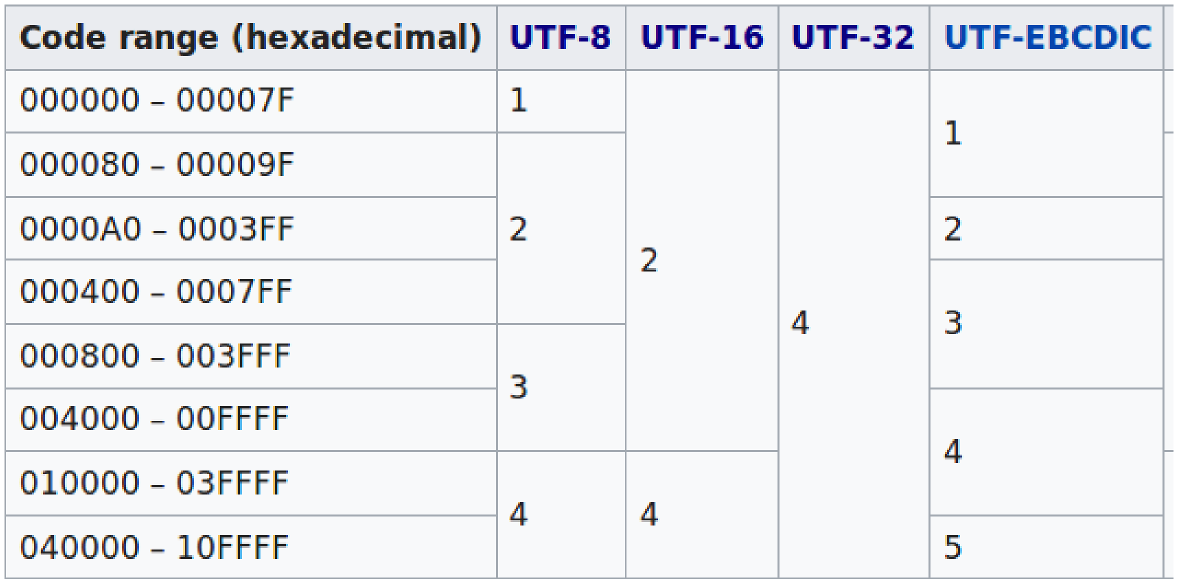
Latin Alphabet: Lowercase	U+0061	a	97	Latin Small Letter A	0066
	U+0062	b	98	Latin Small Letter B	0067
	U+0063	c	99	Latin Small Letter C	0068
	U+0064	d	100	Latin Small Letter D	0069
	U+0065	e	101	Latin Small Letter E	0070
	U+0066	f	102	Latin Small Letter F	0071
	U+0067	g	103	Latin Small Letter G	0072
	U+0068	h	104	Latin Small Letter H	0073
	U+0069	i	105	Latin Small Letter I	0074
	U+006A	j	106	Latin Small Letter J	0075
	U+006B	k	107	Latin Small Letter K	0076
	U+006C	l	108	Latin Small Letter L	0077
	U+006D	m	109	Latin Small Letter M	0078
	U+006E	n	110	Latin Small Letter N	0079
	U+006F	o	111	Latin Small Letter O	0080
	U+0070	p	112	Latin Small Letter P	0081
	U+0071	q	113	Latin Small Letter Q	0082



	U+00CD	Í	205	&Iacute;	Latin Capital letter I with acute	0141
	U+00CE	Î	206	&Icirc;	Latin Capital letter I with circumflex	0142
	U+00CF	Ï	207	&Iuml;	Latin Capital letter I with diaeresis	0143
	U+00D0	Ð	208	&ETH;	Latin Capital letter Eth	0144
	U+00D1	Ñ	209	&Ntilde;	Latin Capital letter N with tilde	0145
	U+00D2	Ò	210	&Ograve;	Latin Capital letter O with grave	0146
	U+00D3	Ó	211	&Oacute;	Latin Capital letter O with acute	0147
	U+00D4	Ô	212	&Ocirc;	Latin Capital letter O with circumflex	0148
	U+00D5	Õ	213	&Otilde;	Latin Capital letter O with tilde	0149
	U+00D6	Ö	214	&Ouml;	Latin Capital letter O with diaeresis	0150
Math	U+00D7	×	215	&times;	Multiplication sign	0151
	U+00D8	Ø	216	&Oslash;	Latin Capital letter O with stroke	0152
	U+00D9	Ù	217	&Ugrave;	Latin Capital letter U with grave	0153
	U+00DA	Ú	218	&Uacute;	Latin Capital letter U with acute	0154
	U+00DB	Û	219	&Ucirc;	Latin Capital Letter U with circumflex	0155
	U+00DC	Ü	220	&Uuml;	Latin Capital Letter U with diaeresis	0156
	U+00DD	Ý	221	&Yacute;	Latin Capital Letter Y with acute	0157
	U+00DE	Þ	222	&THORN;	Latin Capital Letter Thorn	0158
	U+00DF	ß	223	&szlig;	Latin Small Letter sharp S	0159
	U+00E0	à	224	&agrave;	Latin Small Letter A with grave	0160
	U+00E1	á	225	&aacute;	Latin Small Letter A with acute	0161

Unicode

A 0041 1



Code range (hexadecimal)	UTF-8	UTF-16	UTF-32	UTF-EBCDIC
000000 – 00007F	1	2	4	1
000080 – 00009F	2			
0000A0 – 0003FF				2
000400 – 0007FF				3
000800 – 003FFF	3			
004000 – 00FFFF				4
010000 – 03FFFF	4	4		
040000 – 10FFFF				5

Õ 00D5 2

[illegible] 0756 ~~3~~ 2

[illegible] 0759 2

[illegible] 1035 3

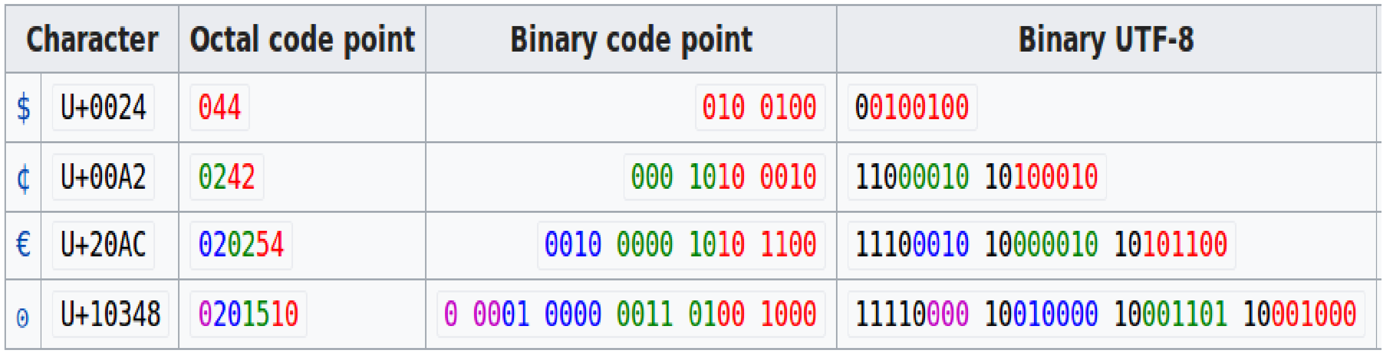
[illegible] 10D6 3

Unicode

[illegible]

Number of bytes	Bits for code point	First code point	Last code point	Byte 1	Byte 2	Byte 3	Byte 4
1	7	U+0000	U+007F	0xxxxxxx			
2	11	U+0080	U+07FF	110xxxxx	10xxxxxx		
3	16	U+0800	U+FFFF	1110xxxx	10xxxxxx	10xxxxxx	
4	21	U+10000	U+10FFFF	11110xxx	10xxxxxx	10xxxxxx	10xxxxxx

Unicode



Character		Octal code point	Binary code point	Binary UTF-8
$	U+0024	044	010 0100	00100100
¢	U+00A2	0242	000 1010 0010	11000010 10100010
€	U+20AC	020254	0010 0000 1010 1100	11100010 10000010 10101100
𐍈	U+10348	0201510	0 0001 0000 0011 0100 1000	11110000 10010000 10001101 10001000

Conversion

- To summarize the previous section: a Unicode string is a sequence of code points, which are numbers from 0 through 0x10FFFF.
- This sequence needs to be represented as a set of bytes (meaning, values from 0 through 255) in memory. The rules for translating a Unicode string into a sequence of bytes are called an encoding.
- More procedurally, this translation back and forth between bytes and strings is defined by two terms:
  - Encoding is the process of translating a string of characters into its raw bytes form, according to a desired encoding name.
  - Decoding is the process of translating a raw bytes into its character string form, according to its encoding name.

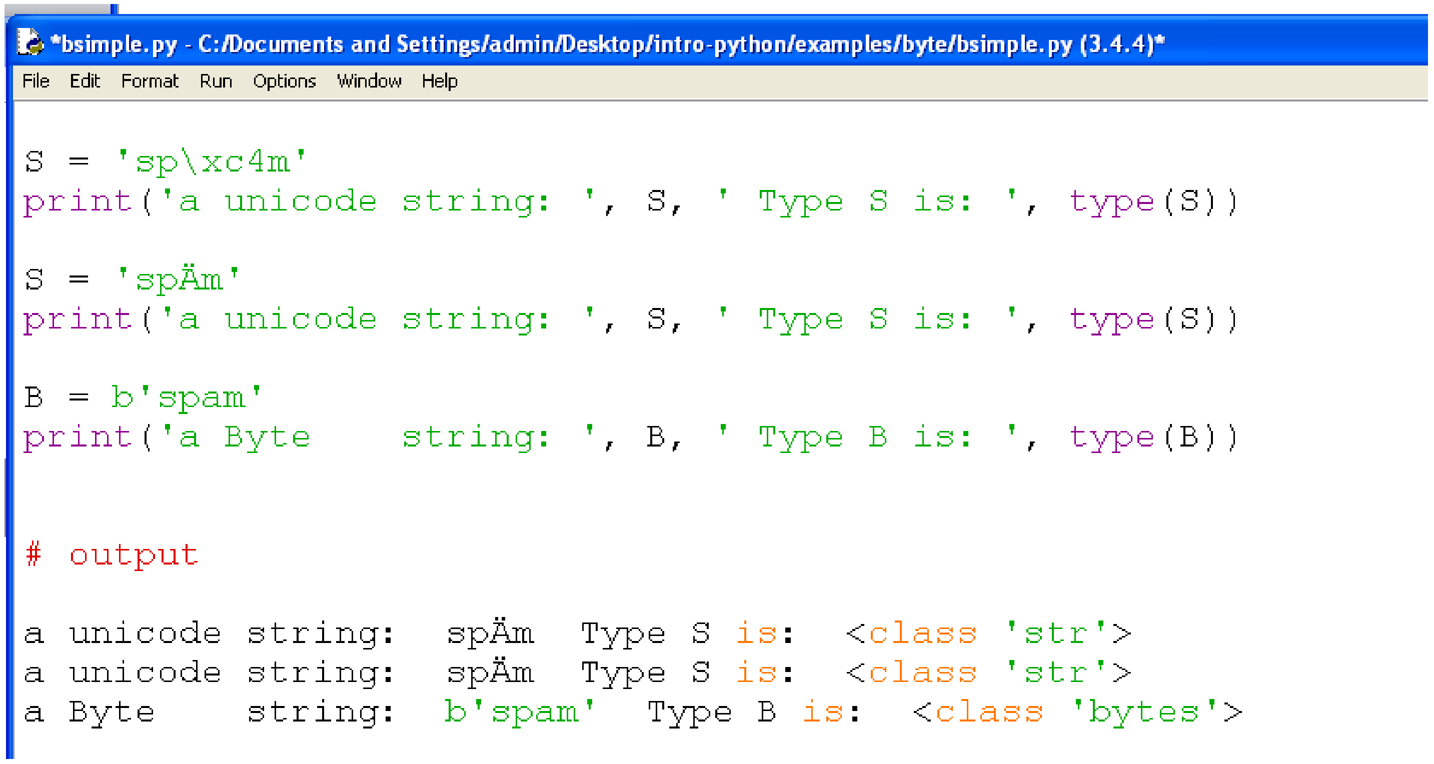
Byte and Bytearray

- Python comes with three string object types one for textual data and two for binary data:
  - str for representing decoded Unicode text (including ASCII)
  - bytes for representing binary data (including encoded text)
  - bytearray, a mutable flavor of the bytes type

Byte and Bytearray

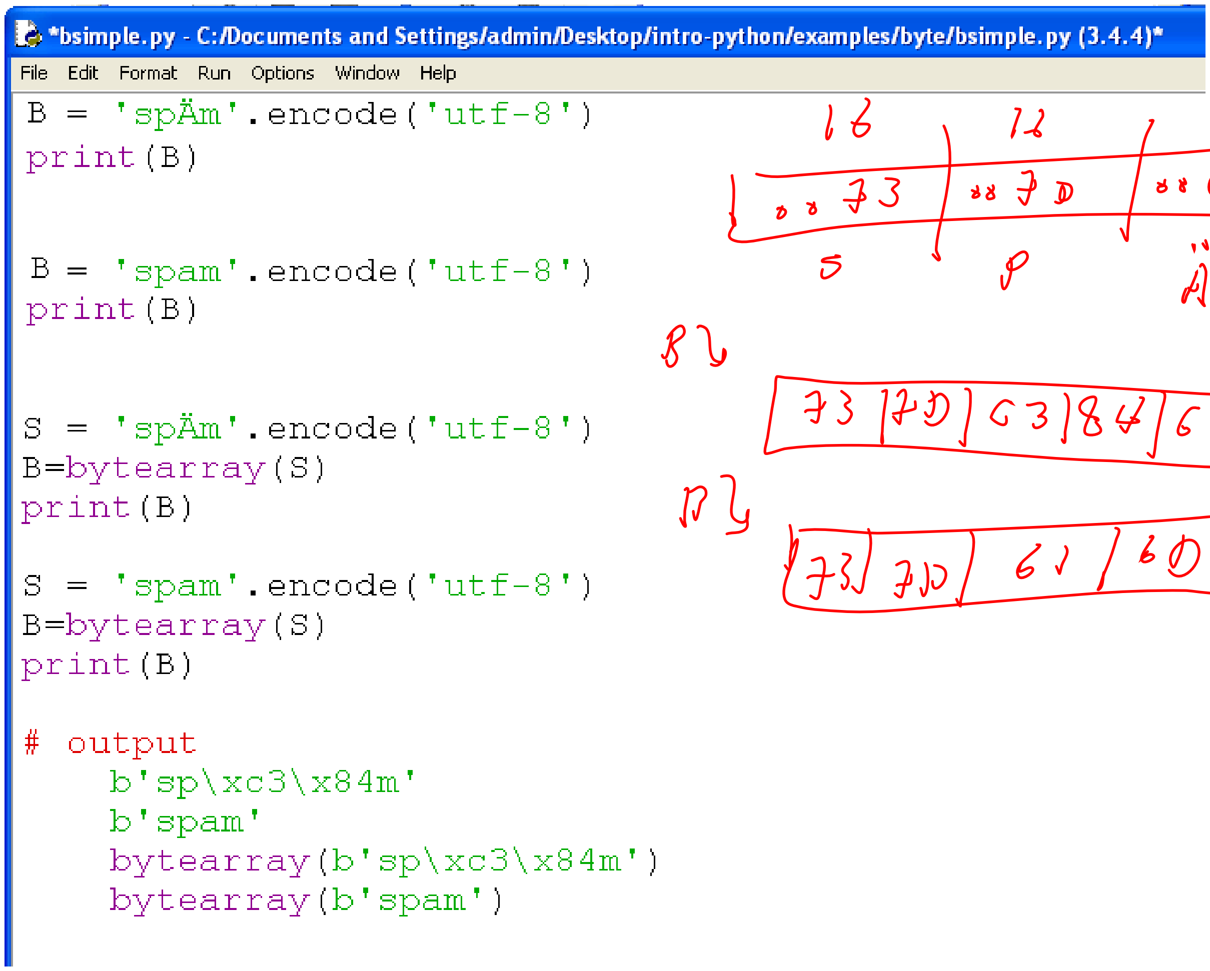
- A bytes object stores a mutable sequence of integers that are in the range 0 to 255. Unlike string objects, indexing a bytes object returns an integer.
- A byte type is immutable
- A bytearray type is a mutable version of byte type

Creation


bsimple.py - C:/Documents and Settings/admin/Desktop/intro-python/examples/byte/bsimple.py (3.4.4)
File Edit Format Run Options Window Help

```
S = 'sp\xc4m'
print('a unicode string: ', S, ' Type S is: ', type(S))

S = 'spÄm'
print('a unicode string: ', S, ' Type S is: ', type(S))

B = b'spam'
print('a Byte    string: ', B, ' Type B is: ', type(B))


# output

a unicode string:  spÄm  Type S is:  <class 'str'>
a unicode string:  spÄm  Type S is:  <class 'str'>
a Byte    string:  b'spam'  Type B is:  <class 'bytes'>
```

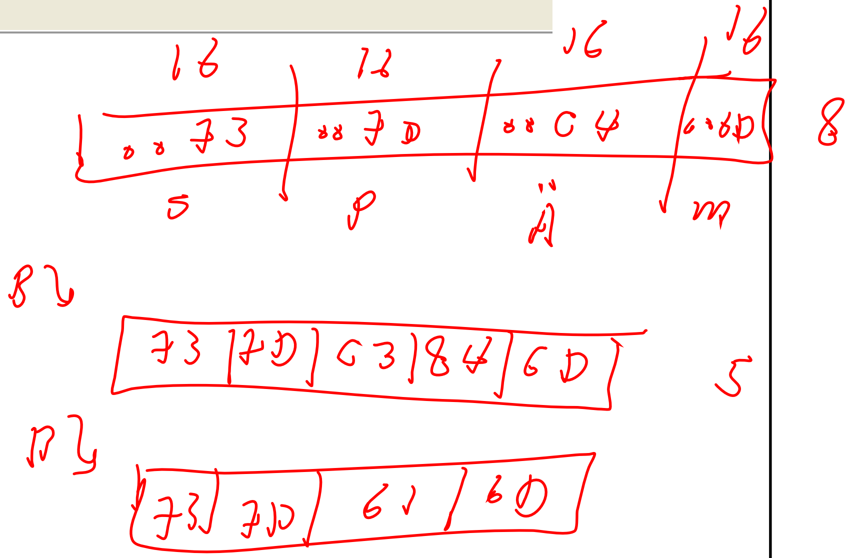
Creation



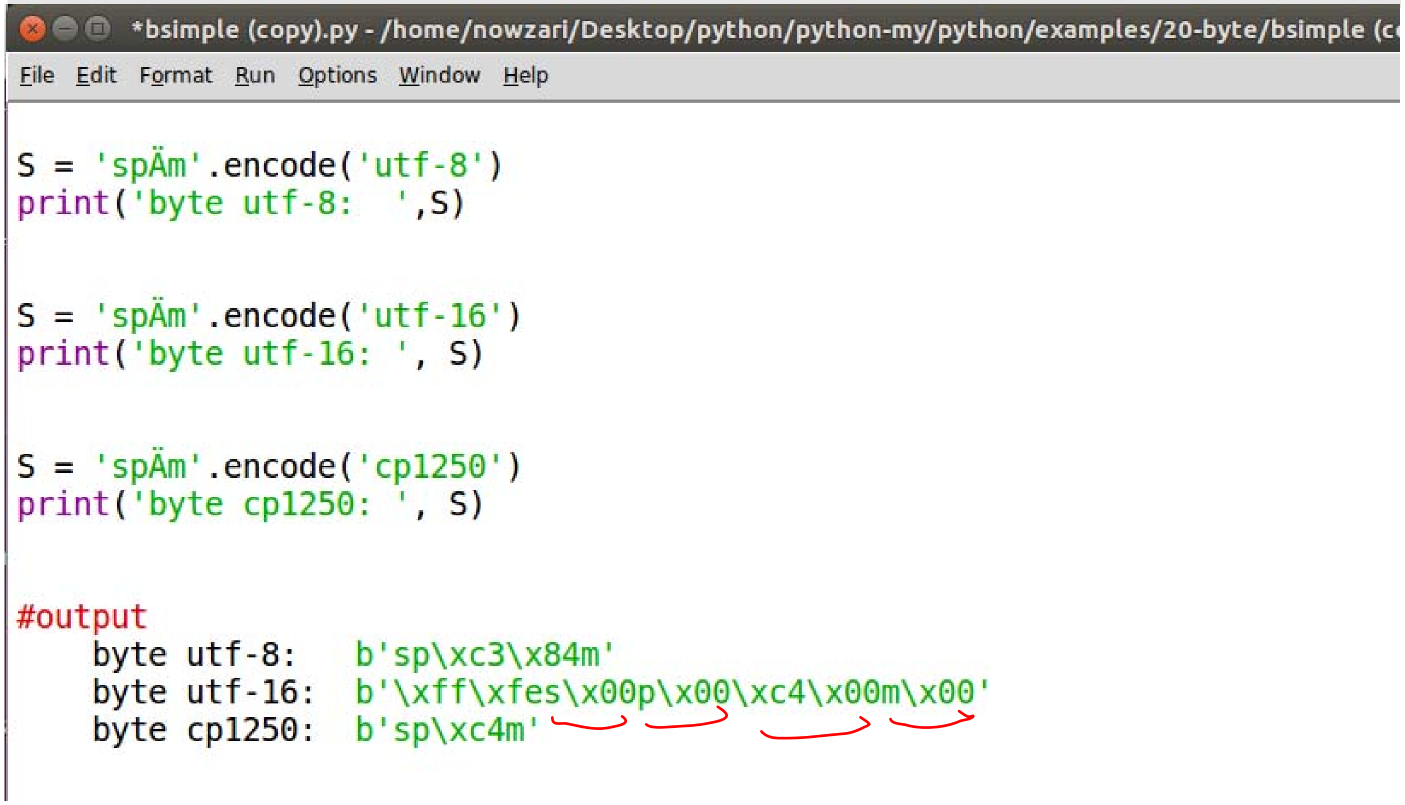
bsimple.py - C:/Documents and Settings/admin/Desktop/intro-python/examples/byte/bsimple.py (3.4.4)

File Edit Format Run Options Window Help

```
B = 'spÄm'.encode('utf-8')
print(B)


B = 'spam'.encode('utf-8')
print(B)


S = 'spÄm'.encode('utf-8')
B=bytearray(S)
print(B)

S = 'spam'.encode('utf-8')
B=bytearray(S)
print(B)

# output
    b'sp\xc3\x84m'
    b'spam'
    bytearray(b'sp\xc3\x84m')
    bytearray(b'spam')
```



Creation


*bsimple (copy).py - /home/nowzari/Desktop/python/python-my/python/examples/20-byte/bsimple (c
File Edit Format Run Options Window Help

```
S = 'spÄm'.encode('utf-8')
print('byte utf-8:   ',S)


S = 'spÄm'.encode('utf-16')
print('byte utf-16: ', S)


S = 'spÄm'.encode('cp1250')
print('byte cp1250: ', S)


#output
    byte utf-8:    b'sp\xc3\x84m'
    byte utf-16:   b'\xff\xfes\x00p\x00\xc4\x00m\x00'
    byte cp1250:   b'sp\xc4m'
```

Creation

```
S = 'spam'.encode('utf-8')
print('byte utf-8: ',S)

S = 'spam'.encode('utf-16')
print('byte utf-16: ', S)

S = 'spam'.encode('cp1250')
print('byte cp1250: ', S)


#output

    byte utf-8:   b'spam'
    byte utf-16:  b'\xff\xfes\x00p\x00a\x00m\x00'
    byte cp1250:  b'spam'
```

Creation

```
S = 'spÄm'.encode('utf-8')
B=bytearray(S)
print('byte array utf-8:  ', B)

S = 'spÄm'.encode('utf-16')
B=bytearray(S)
print('byte array utf-16: ', B)

S = 'spÄm'.encode('cp1250')
B=bytearray(S)
print('byte array cp1250: ', B)

#output

    byte array utf-8:   bytearray(b'sp\xc3\x84m')
    byte array utf-16:  bytearray(b'\xff\xfes\x00p\x00\xc4\x00m\x00')
    byte array cp1250:  bytearray(b'sp\xc4m')
```

Creation

```
S = 'spÄm'.encode('utf-8')
B=bytes(S)
print('byte utf-8:  ',B)

S = 'spÄm'
B=bytes(S, 'utf-8')
print('byte utf-8:  ',B)


S = 'spÄm'.encode('utf-8')
B=bytearray(S)
print('byte array utf-8:  ', B)

S = 'spÄm'
B=bytearray(S,'utf-8')
print('byte array utf-8:  ', B)

#output
    byte utf-8:   b'sp\xc3\x84m'
    byte utf-8:   b'sp\xc3\x84m'
    byte array utf-8:   bytearray(b'sp\xc3\x84m')
    byte array utf-8:   bytearray(b'sp\xc3\x84m')
```

Traverse

Python program that uses byte literals

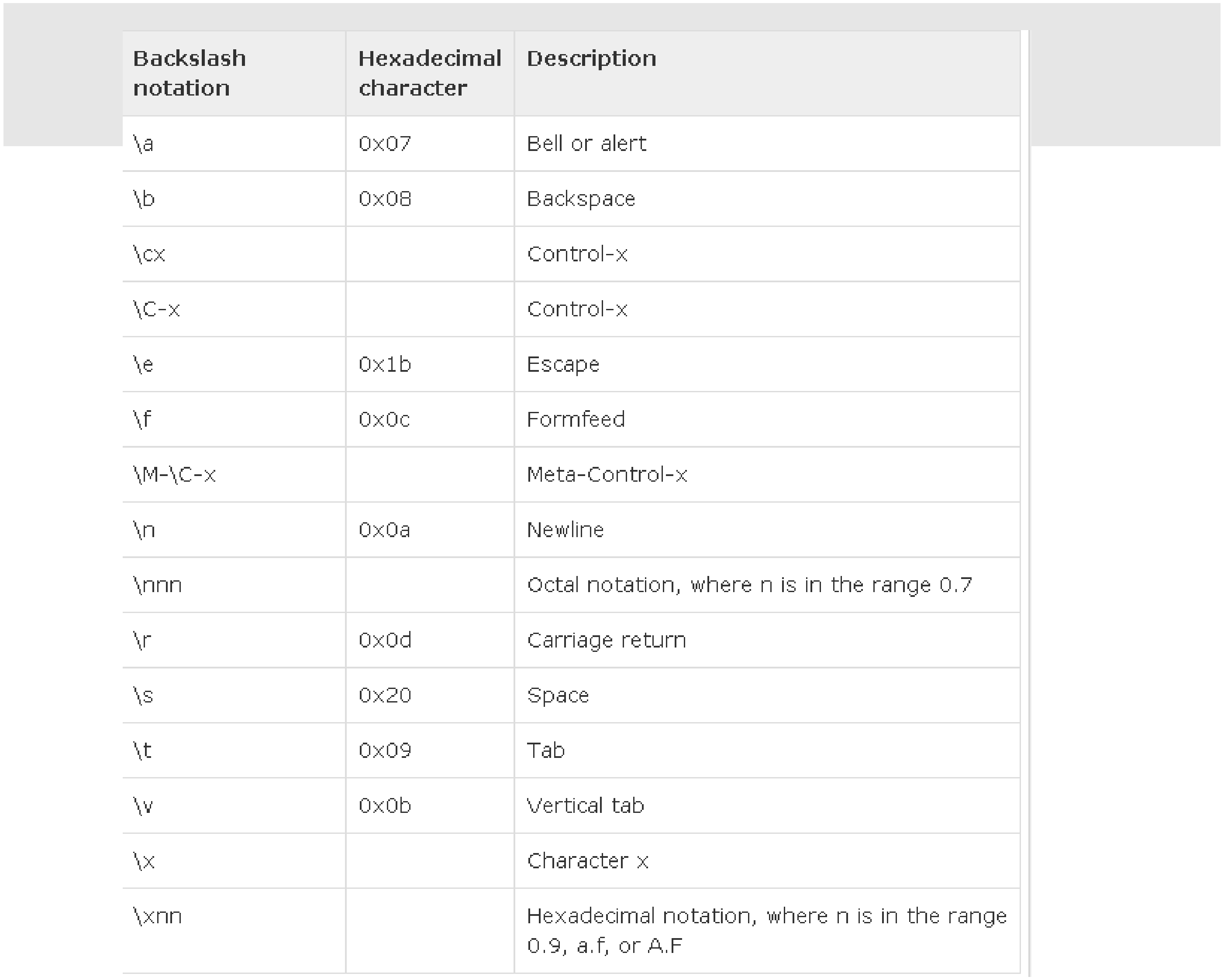
```
# Create bytes object from byte literal.
data = bytes( "abc', 'utf-8')
for value in data:
    print(value)

print()

# Create bytearray from byte literal.
arr = bytearray( "abc", 'utf-8')
for value in arr:
    print(value)
```

Output

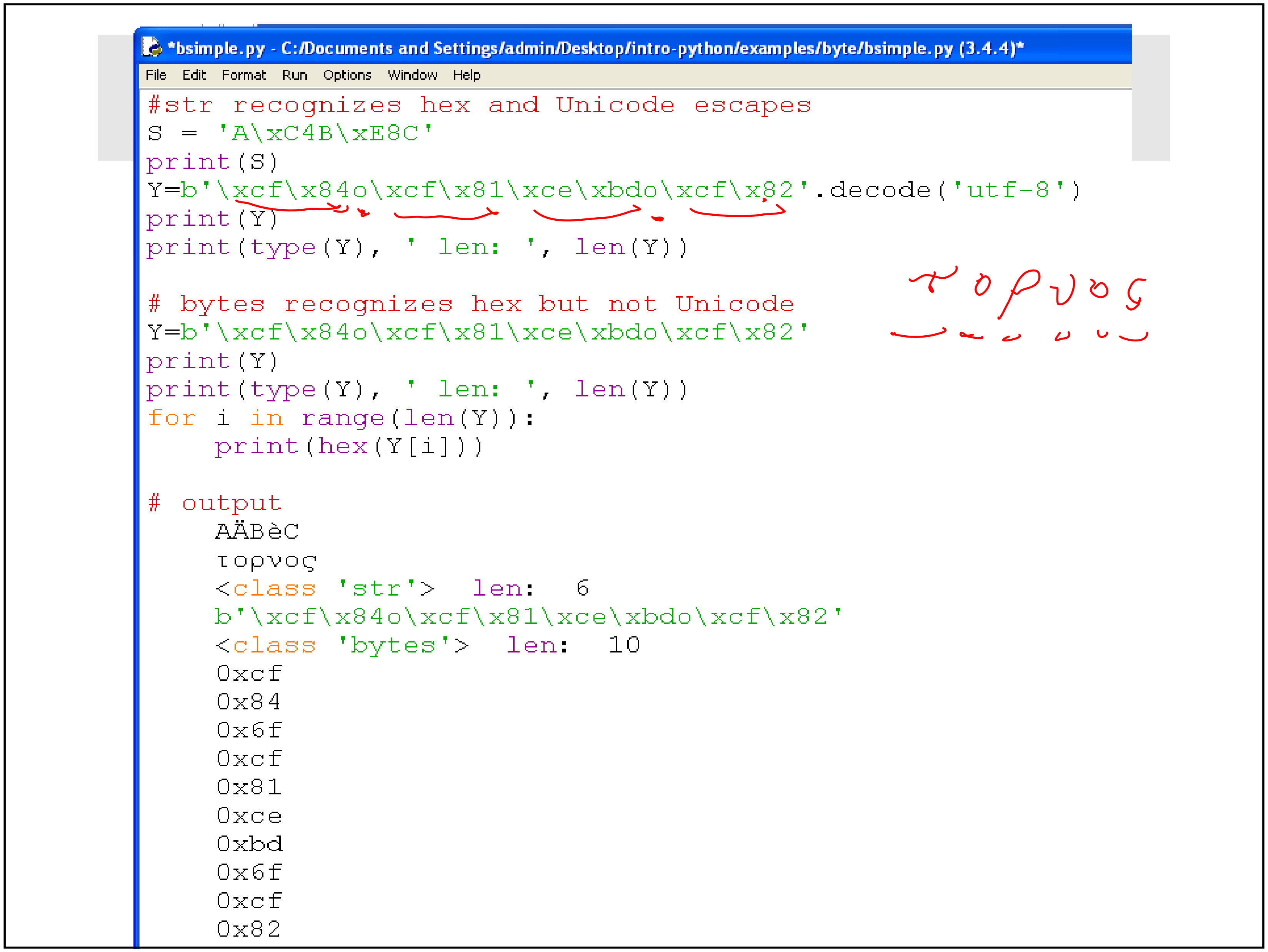
```
97
98
99

97
98
99
```

Encoding and escape

- Following table is a list of escape or non-printable characters that can be represented with backslash notation.
- An escape character gets interpreted; in a single quoted as well as double quoted strings.



Backslash notation	Hexadecimal character	Description
\a	0x07	Bell or alert
\b	0x08	Backspace
\cx		Control-x
\C-x		Control-x
\e	0x1b	Escape
\f	0x0c	Formfeed
\M-\C-x		Meta-Control-x
\n	0x0a	Newline
\nnn		Octal notation, where n is in the range 0.7
\r	0x0d	Carriage return
\s	0x20	Space
\t	0x09	Tab
\v	0x0b	Vertical tab
\x		Character x
\xnn		Hexadecimal notation, where n is in the range 0.9, a.f, or A.F


bsimple.py - C:/Documents and Settings/admin/Desktop/intro-python/examples/byte/bsimple.py (3.4.4)
File Edit Format Run Options Window Help
τορνος

```
#str recognizes hex and Unicode escapes
S = 'A\xC4B\xE8C'
print(S)
Y=b'\xcf\x84o\xcf\x81\xce\xbdo\xcf\x82'.decode('utf-8')
print(Y)
print(type(Y), ' len: ', len(Y))

# bytes recognizes hex but not Unicode
Y=b'\xcf\x84o\xcf\x81\xce\xbdo\xcf\x82'
print(Y)
print(type(Y), ' len: ', len(Y))
for i in range(len(Y)):
    print(hex(Y[i]))

# output
    AÄBèC
    τορνος
    <class 'str'>  len:  6
    b'\xcf\x84o\xcf\x81\xce\xbdo\xcf\x82'
    <class 'bytes'>  len:  10
    0xcf
    0x84
    0x6f
    0xcf
    0x81
    0xce
    0xbd
    0x6f
    0xcf
    0x82
```

Modifying Bytearray

Python program that creates bytearray from list

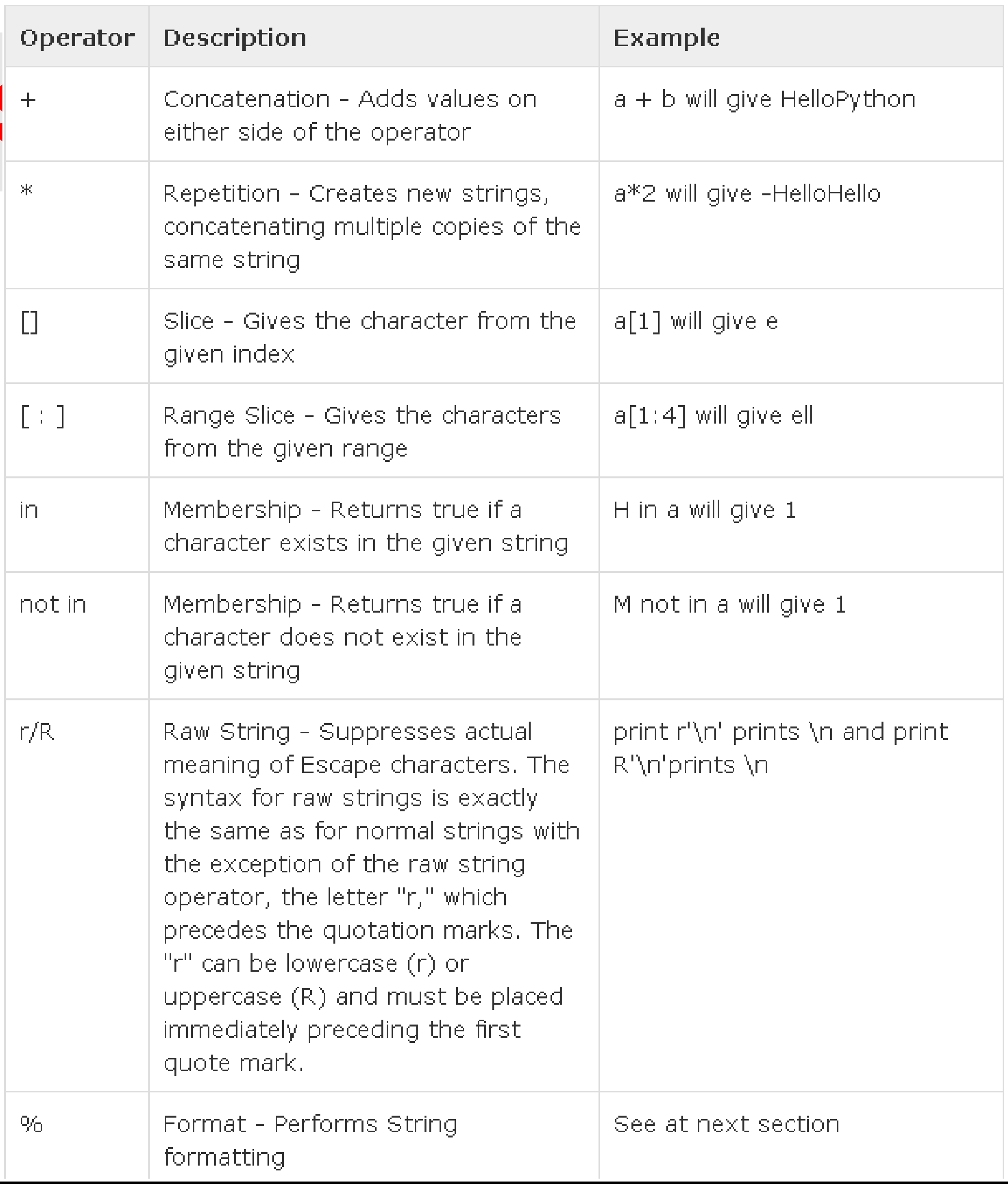
```
elements = [0, 200, 50, 25, 10, 255]

# Create bytearray from list of integers.
values = bytearray(elements)

# Modify elements in the bytearray.
values[0] = 5
values[1] = 0

# Display bytes.
for value in values:
    print(value)
```

Output

```
5
0
50
25
10
255
```

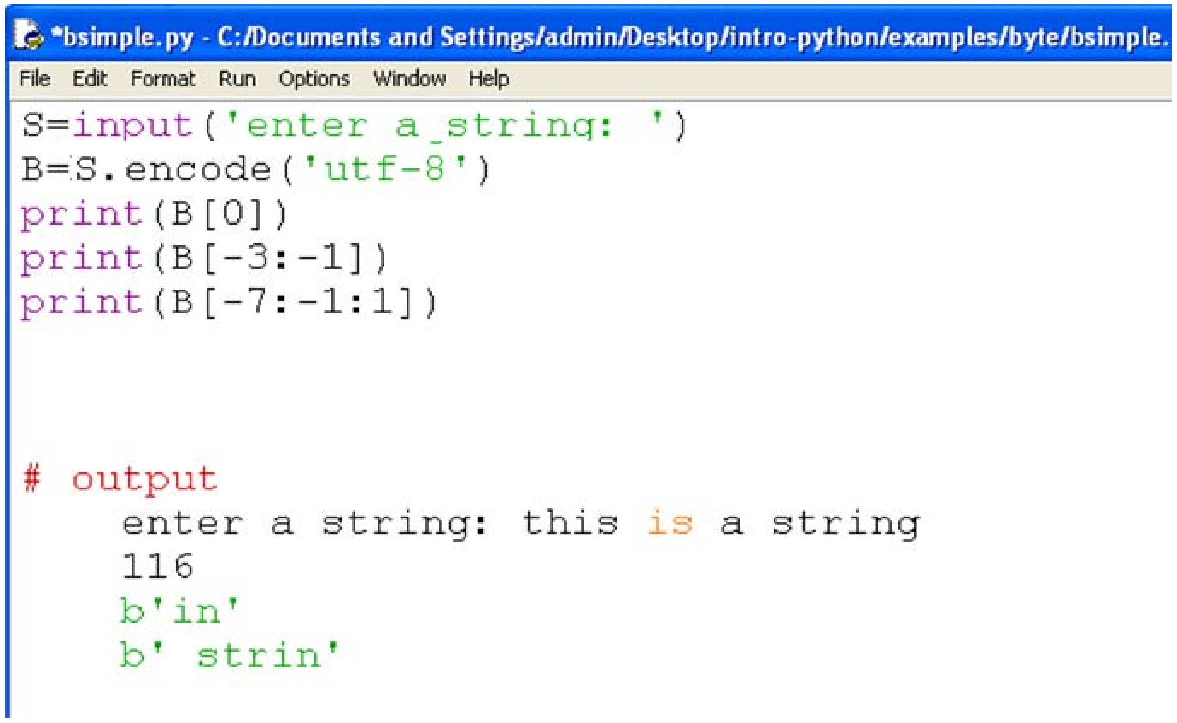
Bytes Special Operators

- All string special operator can work on bytes and bytearray
- Assume string variable a holds 'Hello' and variable b holds 'Python', then −



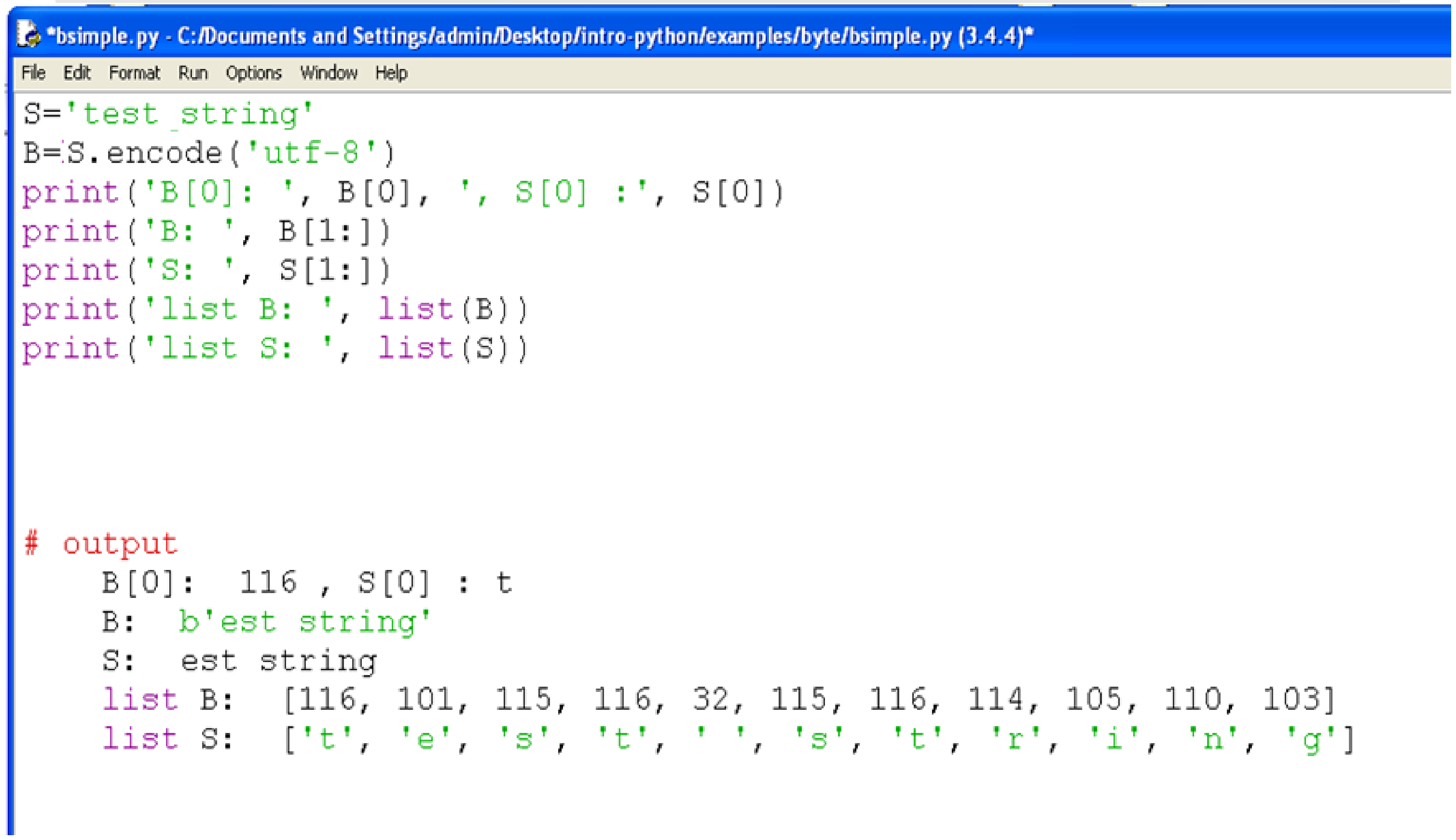
Operator	Description	Example
+	Concatenation - Adds values on either side of the operator	a + b will give HelloPython
*	Repetition - Creates new strings, concatenating multiple copies of the same string	a*2 will give -HelloHello
[]	Slice - Gives the character from the given index	a[1] will give e
[:]	Range Slice - Gives the characters from the given range	a[1:4] will give ell
in	Membership - Returns true if a character exists in the given string	H in a will give 1
not in	Membership - Returns true if a character does not exist in the given string	M not in a will give 1
r/R	Raw String - Suppresses actual meaning of Escape characters. The syntax for raw strings is exactly the same as for normal strings with the exception of the raw string operator, the letter "r," which precedes the quotation marks. The "r" can be lowercase (r) or uppercase (R) and must be placed immediately preceding the first quote mark.	print r'\n' prints \n and print R'\n'prints \n
%	Format - Performs String formatting	See at next section

Indexing


*bsimple.py - C:/Documents and Settings/admin/Desktop/intro-python/examples/byte/bsimple.
File Edit Format Run Options Window Help

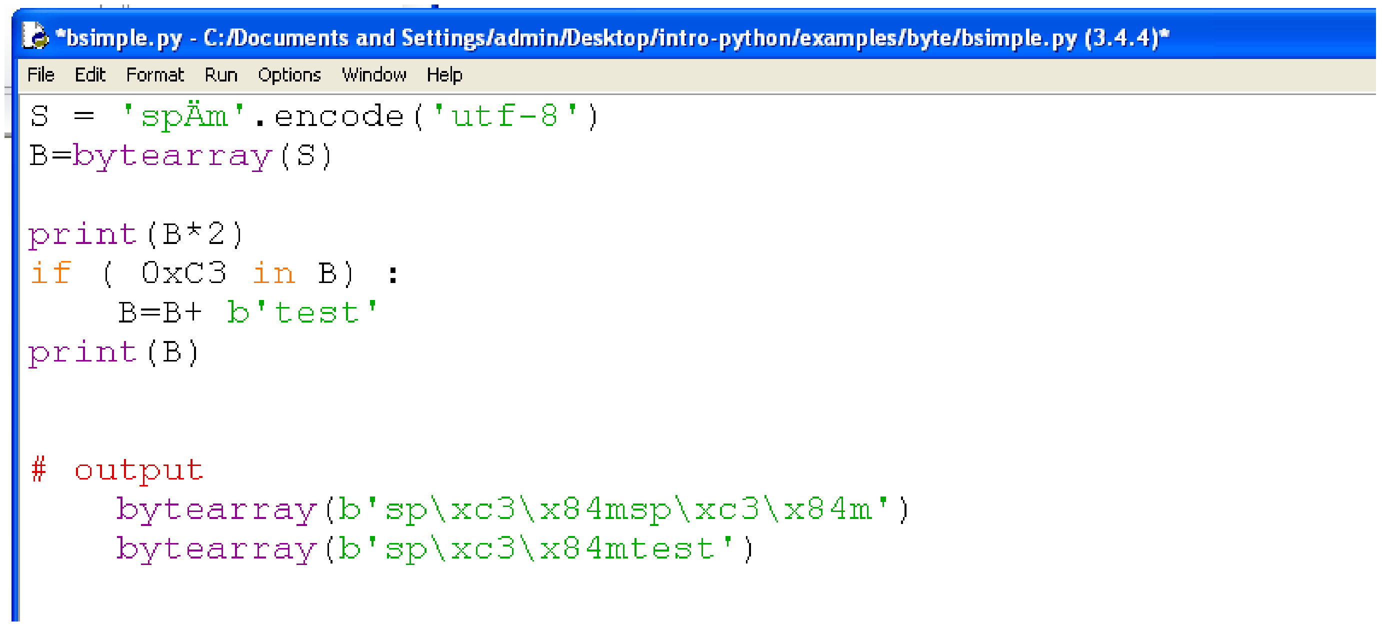
```
S=input('enter a string: ')
B=S.encode('utf-8')
print(B[0])
print(B[-3:-1])
print(B[-7:-1:1])



# output
    enter a string: this is a string
    116
    b'in'
    b' strin'
```

Indexing


bsimple.py - C:/Documents and Settings/admin/Desktop/intro-python/examples/byte/bsimple.py (3.4.4)
File Edit Format Run Options Window Help

```
S='test string'
B=S.encode('utf-8')
print('B[0]: ', B[0], ', S[0] :', S[0])
print('B: ', B[1:])
print('S: ', S[1:])
print('list B: ', list(B))
print('list S: ', list(S))




# output
    B[0]:  116 , S[0] : t
    B:  b'est string'
    S:  est string
    list B:  [116, 101, 115, 116, 32, 115, 116, 114, 105, 110, 103]
    list S:  ['t', 'e', 's', 't', ' ', 's', 't', 'r', 'i', 'n', 'g']
```

Add and multiply

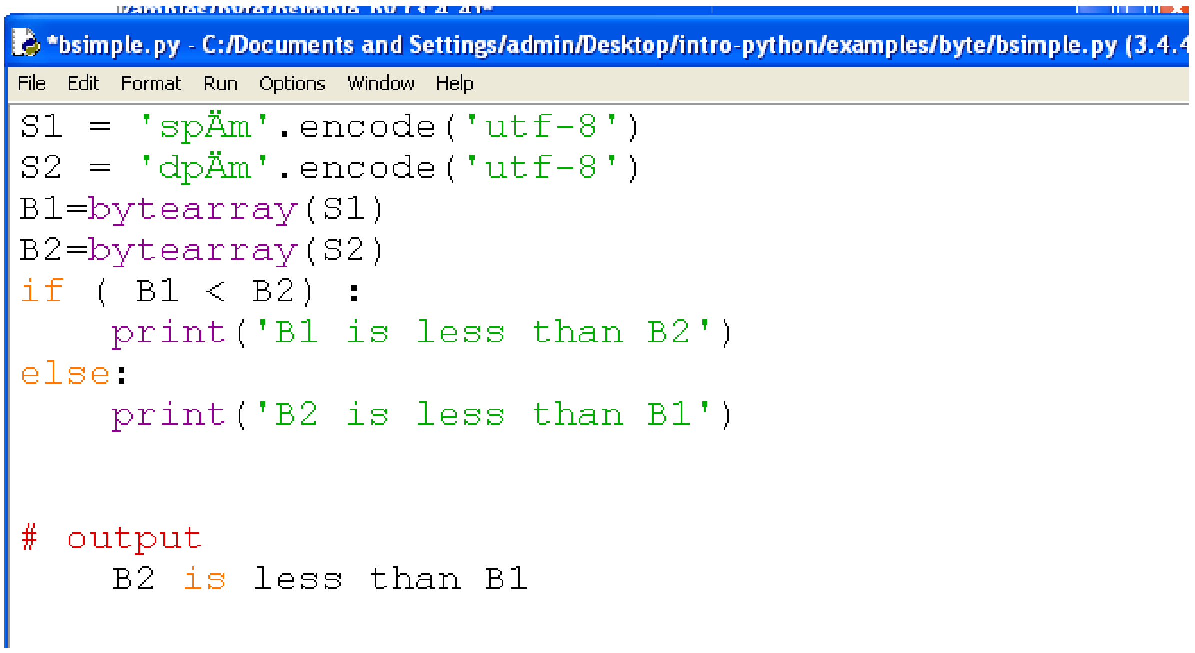

bsimple.py - C:/Documents and Settings/admin/Desktop/intro-python/examples/byte/bsimple.py (3.4.4)
File Edit Format Run Options Window Help

```
S = 'spÄm'.encode('utf-8')
B=bytearray(S)

print(B*2)
if ( 0xC3 in B) :
    B=B+ b'test'
print(B)


# output
    bytearray(b'sp\xc3\x84msp\xc3\x84m')
    bytearray(b'sp\xc3\x84mtest')
```

Bytes comparing Operator

- The standard comparisons (<, <=, >, >=, ==, !=) can apply to bytes strings. These comparisons use the standard byte-by-byte comparison rules.

Bytes comparing Operator

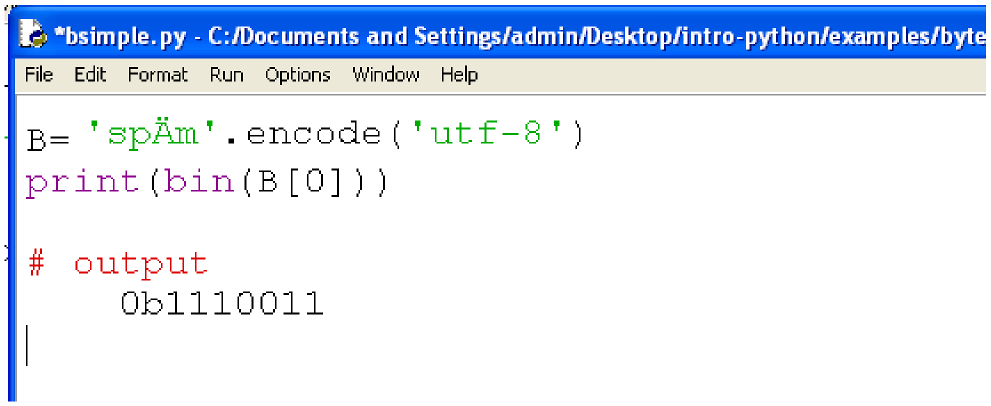

*bsimple.py - C:/Documents and Settings/admin/Desktop/intro-python/examples/byte/bsimple.py (3.4.4
File Edit Format Run Options Window Help

```
S1 = 'spÄm'.encode('utf-8')
S2 = 'dpÄm'.encode('utf-8')
B1=bytearray(S1)
B2=bytearray(S2)
if ( B1 < B2) :
    print('B1 is less than B2')
else:
    print('B2 is less than B1')


# output
    B2 is less than B1
```

built-in functions for Bytes

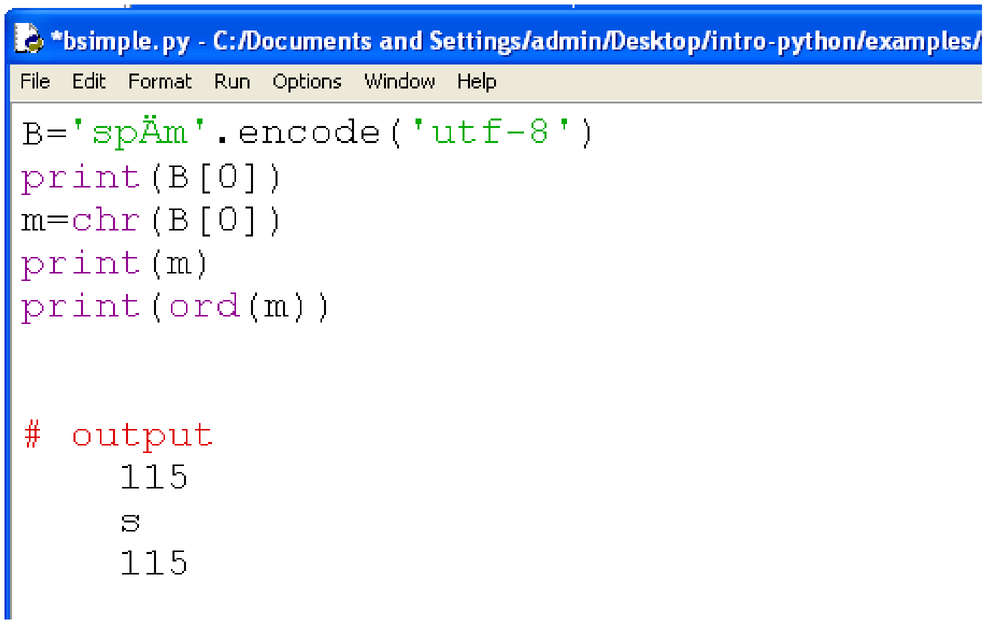
- bin(*x*)
- hex(*x*)
- oct(*x*)
- chr(*i*)
- ord(*c*)
- len(*s*)
- type(s)
- id(s)

built-in functions for Bytes


*bsimple.py - C:/Documents and Settings/admin/Desktop/intro-python/examples/byte
File Edit Format Run Options Window Help

```
B= 'spÄm'.encode('utf-8')
print(bin(B[0]))

# output
    0b1110011
```

built-in functions for Bytes

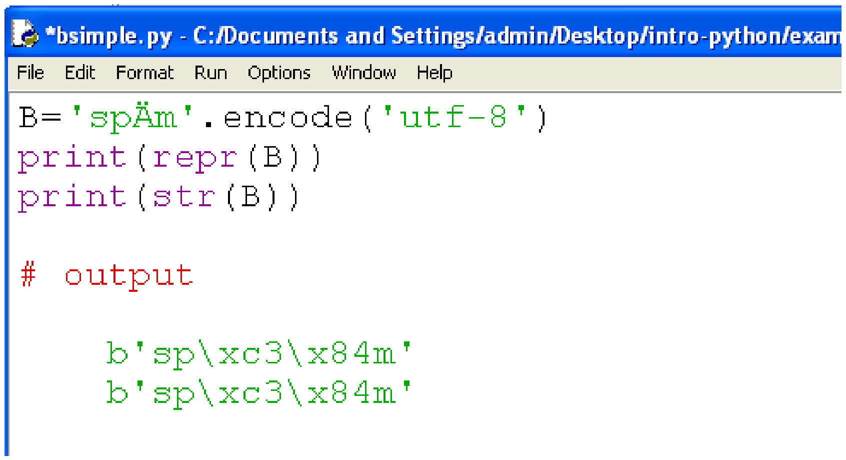

*bsimple.py - C:/Documents and Settings/admin/Desktop/intro-python/examples/
File Edit Format Run Options Window Help

```
B='spÄm'.encode('utf-8')
print(B[0])
m=chr(B[0])
print(m)
print(ord(m))


# output
    115
    s
    115
```

Type Conversion built-in functions

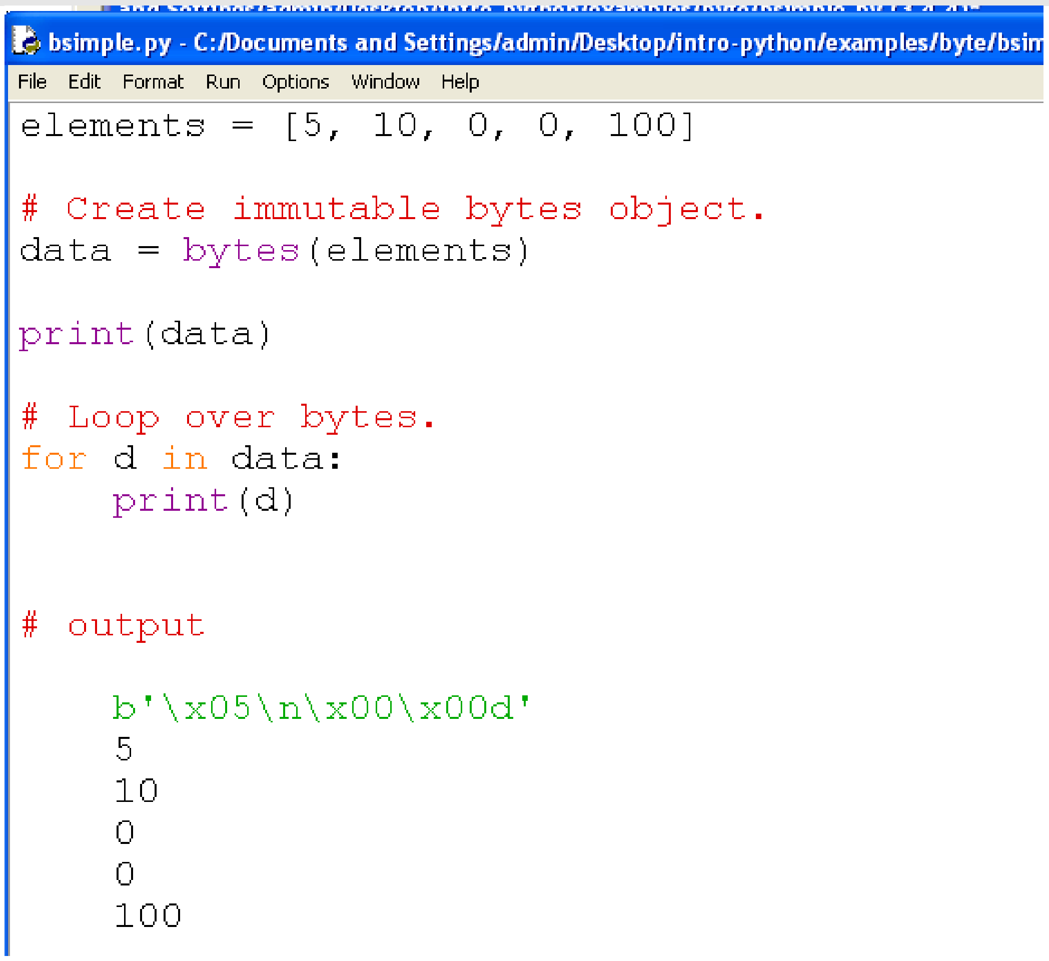
- repr(*object*)
- str(object)
- bytes
- bytearray

Type conversion


*bsimple.py - C:/Documents and Settings/admin/Desktop/intro-python/exam
File Edit Format Run Options Window Help

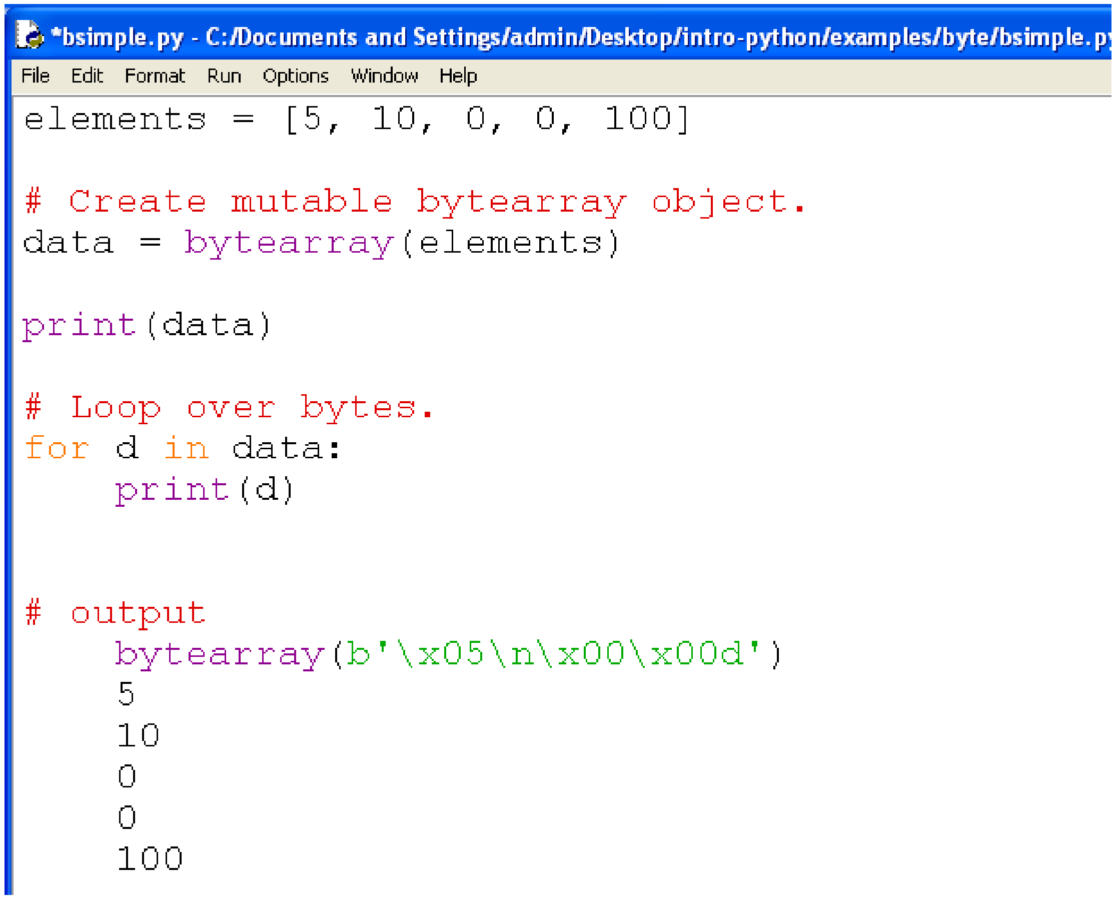
```
B='spÄm'.encode('utf-8')
print(repr(B))
print(str(B))

#  output

    b'sp\xc3\x84m'
    b'sp\xc3\x84m'
```

Type conversion


bsimple.py - C:/Documents and Settings/admin/Desktop/intro-python/examples/byte/bsim
File Edit Format Run Options Window Help

```
elements = [5, 10, 0, 0, 100]

# Create immutable bytes object.
data = bytes(elements)

print(data)

# Loop over bytes.
for d in data:
    print(d)


# output

    b'\x05\n\x00\x00d'
    5
    10
    0
    0
    100
```

Type conversion


*bsimple.py - C:/Documents and Settings/admin/Desktop/intro-python/examples/byte/bsimple.py
File Edit Format Run Options Window Help

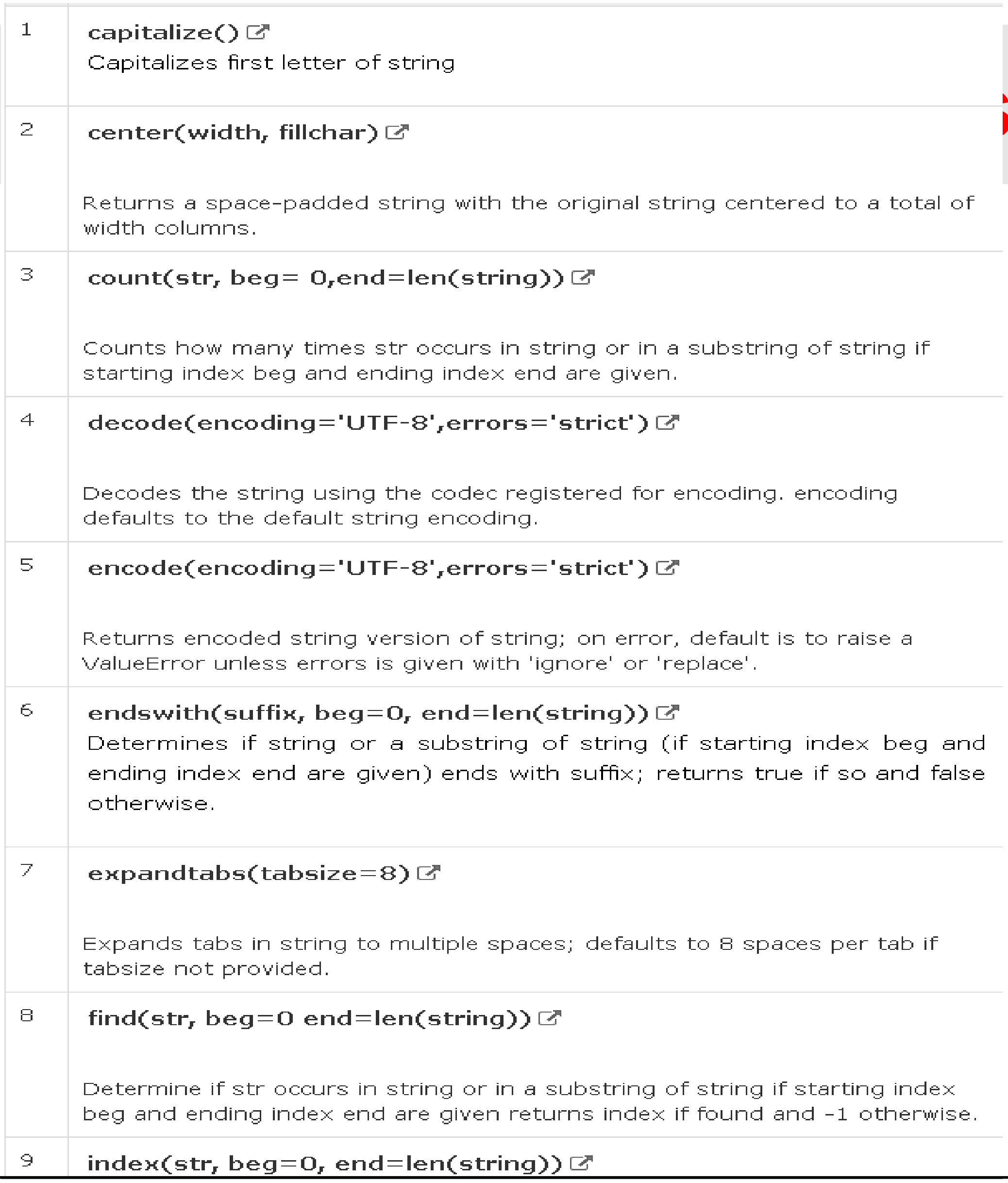
```
elements = [5, 10, 0, 0, 100]

# Create mutable bytearray object.
data = bytearray(elements)

print(data)

# Loop over bytes.
for d in data:
    print(d)


# output
    bytearray(b'\x05\n\x00\x00d')
    5
    10
    0
    0
    100
```

Type specific Methods

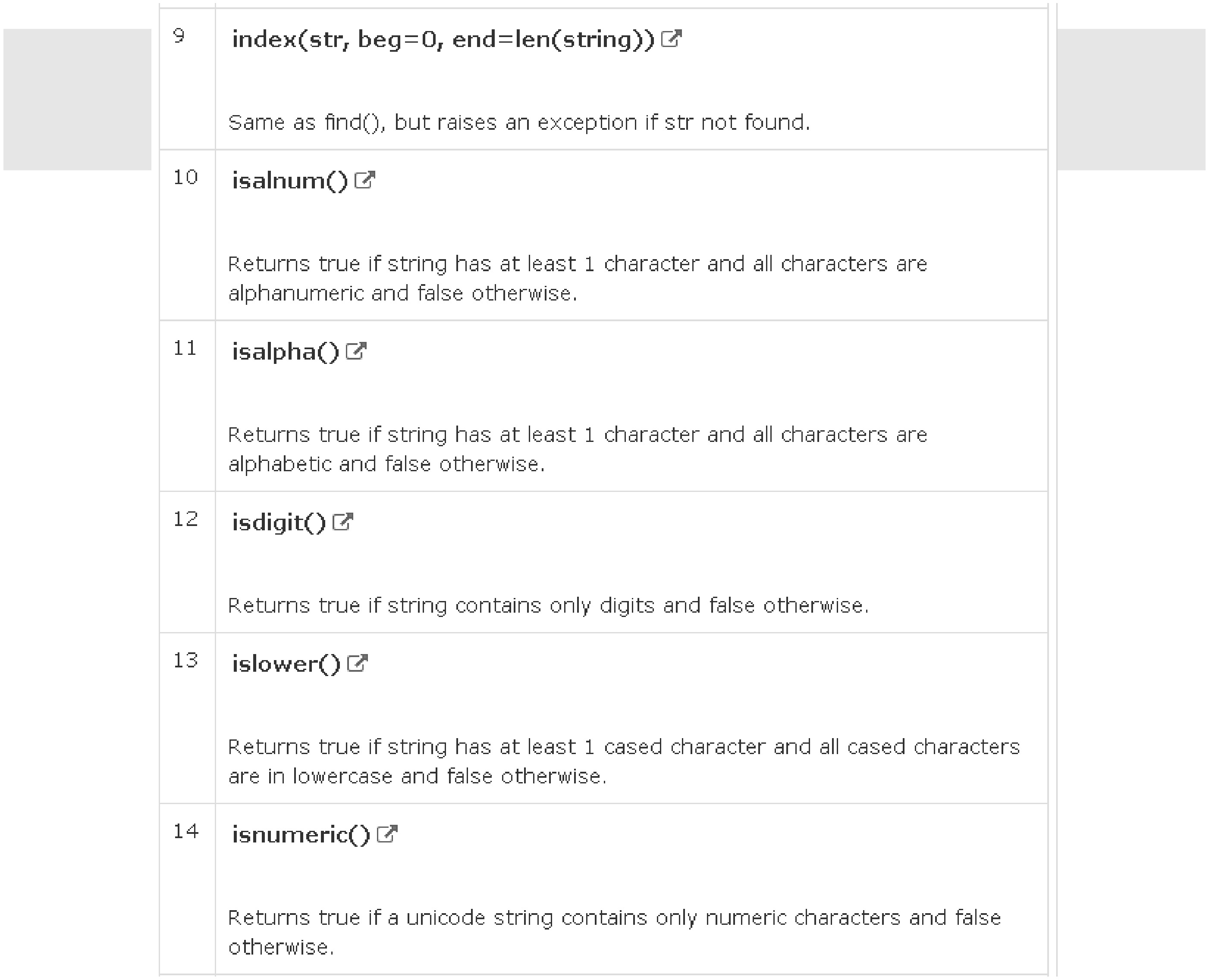
- Python includes some Type specific methods to manipulate strings
- These methods work on bytes and bytearray too.

Type specific Methods

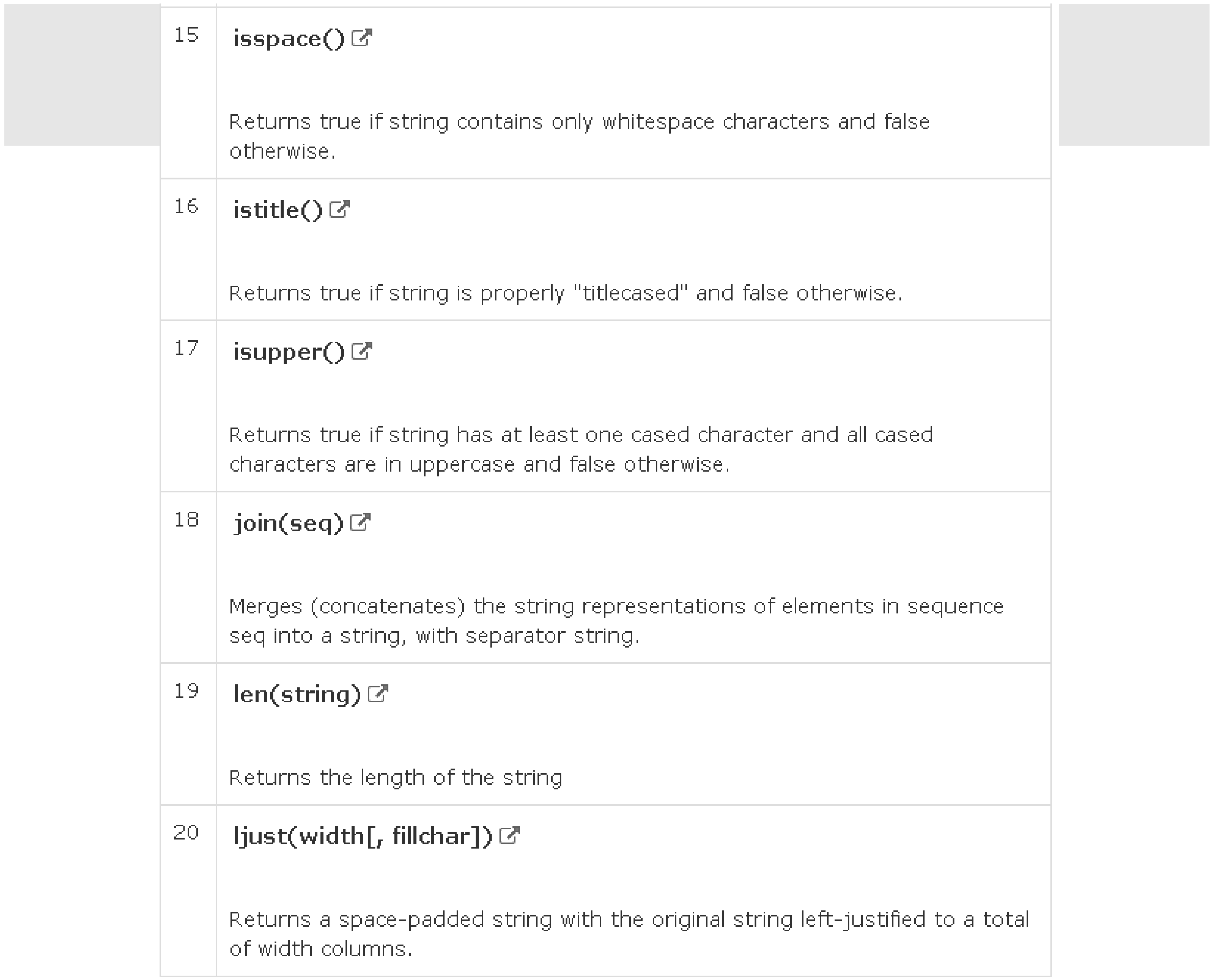
- capitalize
- Count
- Find
- Index
- Decode and Encode
- Append, insert, and del
- Replace
- Startwith and endwith
- Split
- join



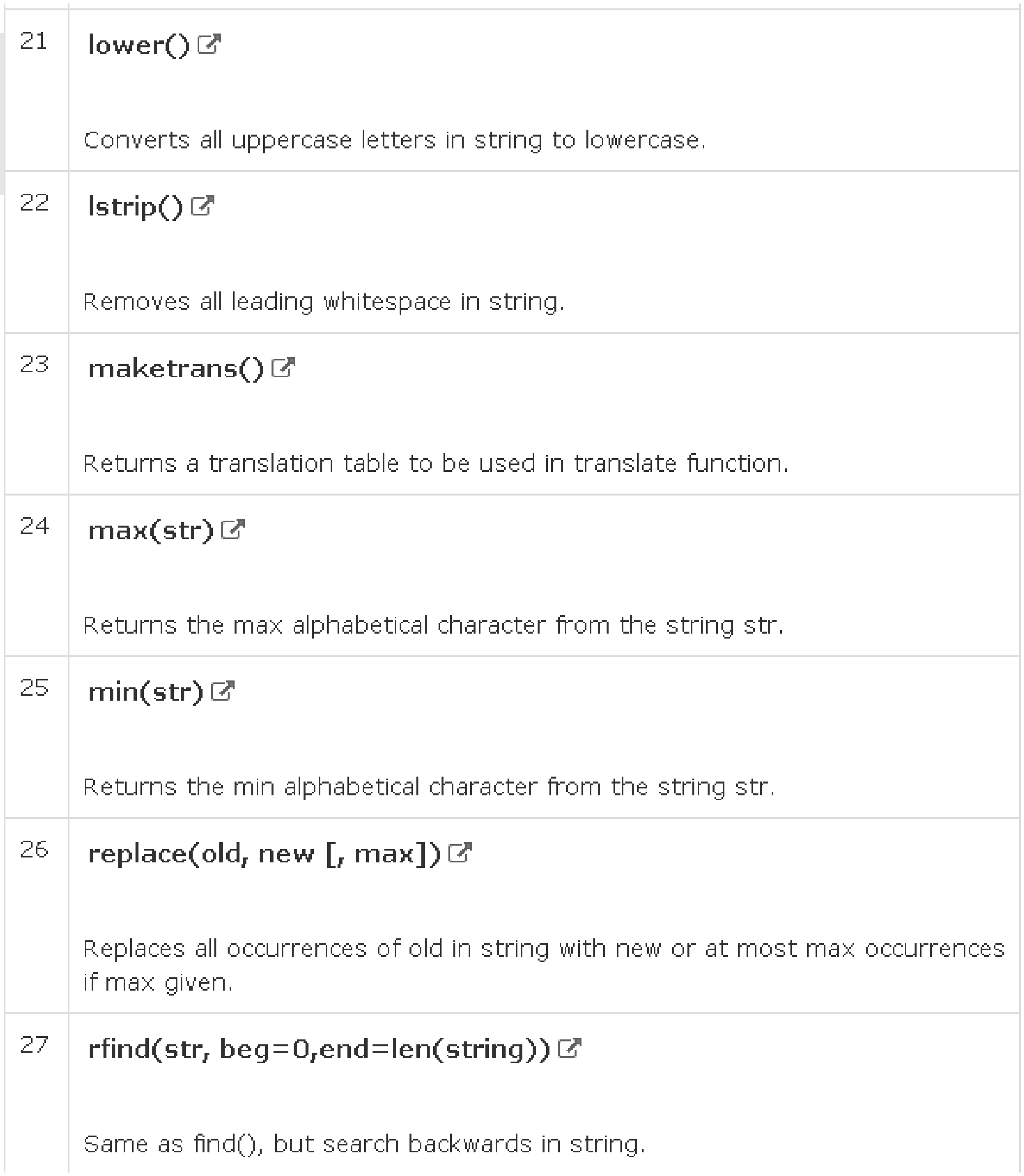
1	**capitalize()** Capitalizes first letter of string
2	**center(width, fillchar)** Returns a space-padded string with the original string centered to a total of width columns.
3	**count(str, beg= 0,end=len(string))** Counts how many times str occurs in string or in a substring of string if starting index beg and ending index end are given.
4	**decode(encoding='UTF-8',errors='strict')** Decodes the string using the codec registered for encoding. encoding defaults to the default string encoding.
5	**encode(encoding='UTF-8',errors='strict')** Returns encoded string version of string; on error, default is to raise a ValueError unless errors is given with 'ignore' or 'replace'.
6	**endswith(suffix, beg=0, end=len(string))** Determines if string or a substring of string (if starting index beg and ending index end are given) ends with suffix; returns true if so and false otherwise.
7	**expandtabs(tabsize=8)** Expands tabs in string to multiple spaces; defaults to 8 spaces per tab if tabsize not provided.
8	**find(str, beg=0 end=len(string))** Determine if str occurs in string or in a substring of string if starting index beg and ending index end are given returns index if found and -1 otherwise.
9	**index(str, beg=0, end=len(string))**



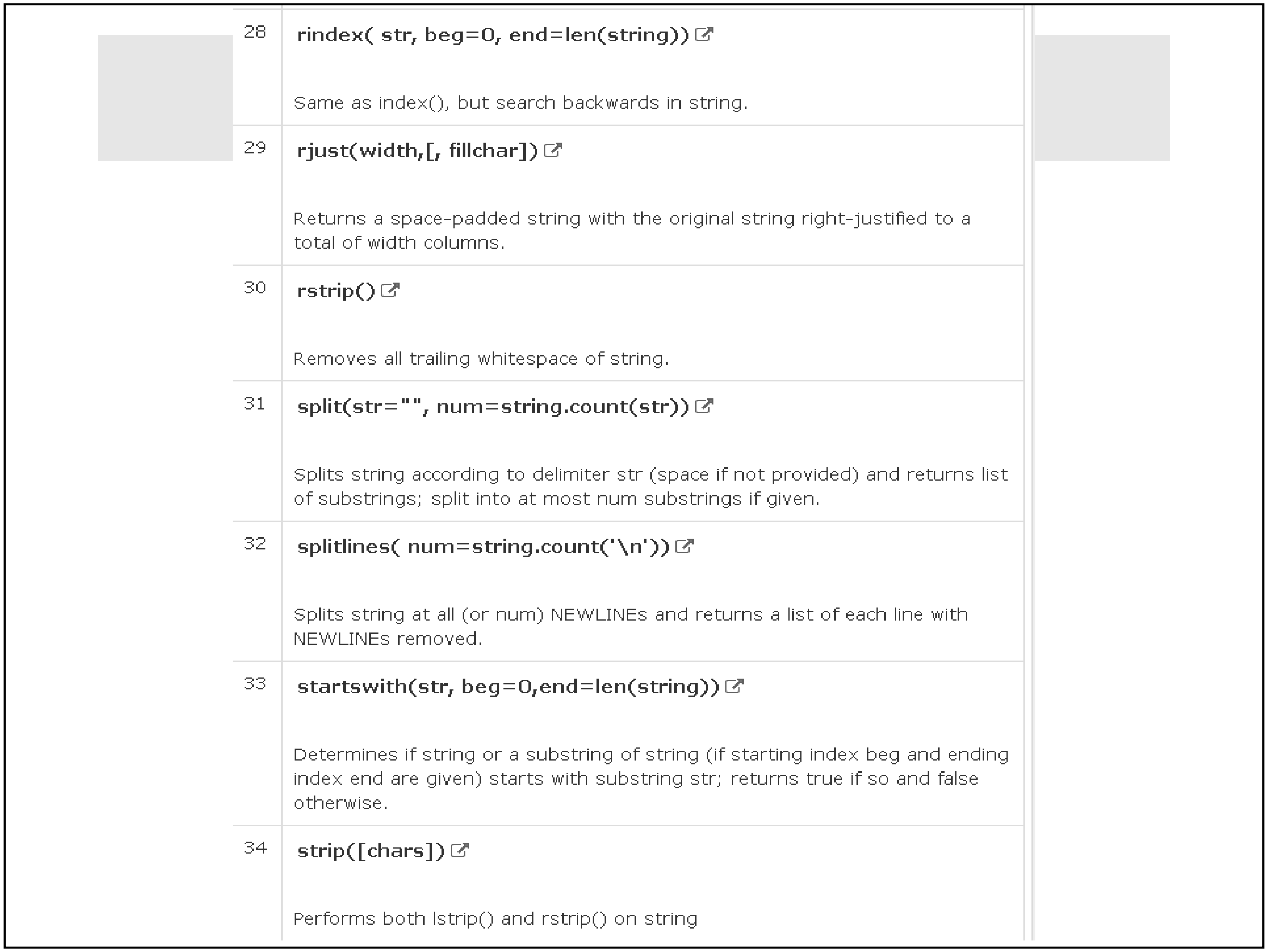
9	**index(str, beg=0, end=len(string))** Same as find(), but raises an exception if str not found.
10	**isalnum()** Returns true if string has at least 1 character and all characters are alphanumeric and false otherwise.
11	**isalpha()** Returns true if string has at least 1 character and all characters are alphabetic and false otherwise.
12	**isdigit()** Returns true if string contains only digits and false otherwise.
13	**islower()** Returns true if string has at least 1 cased character and all cased characters are in lowercase and false otherwise.
14	**isnumeric()** Returns true if a unicode string contains only numeric characters and false otherwise.



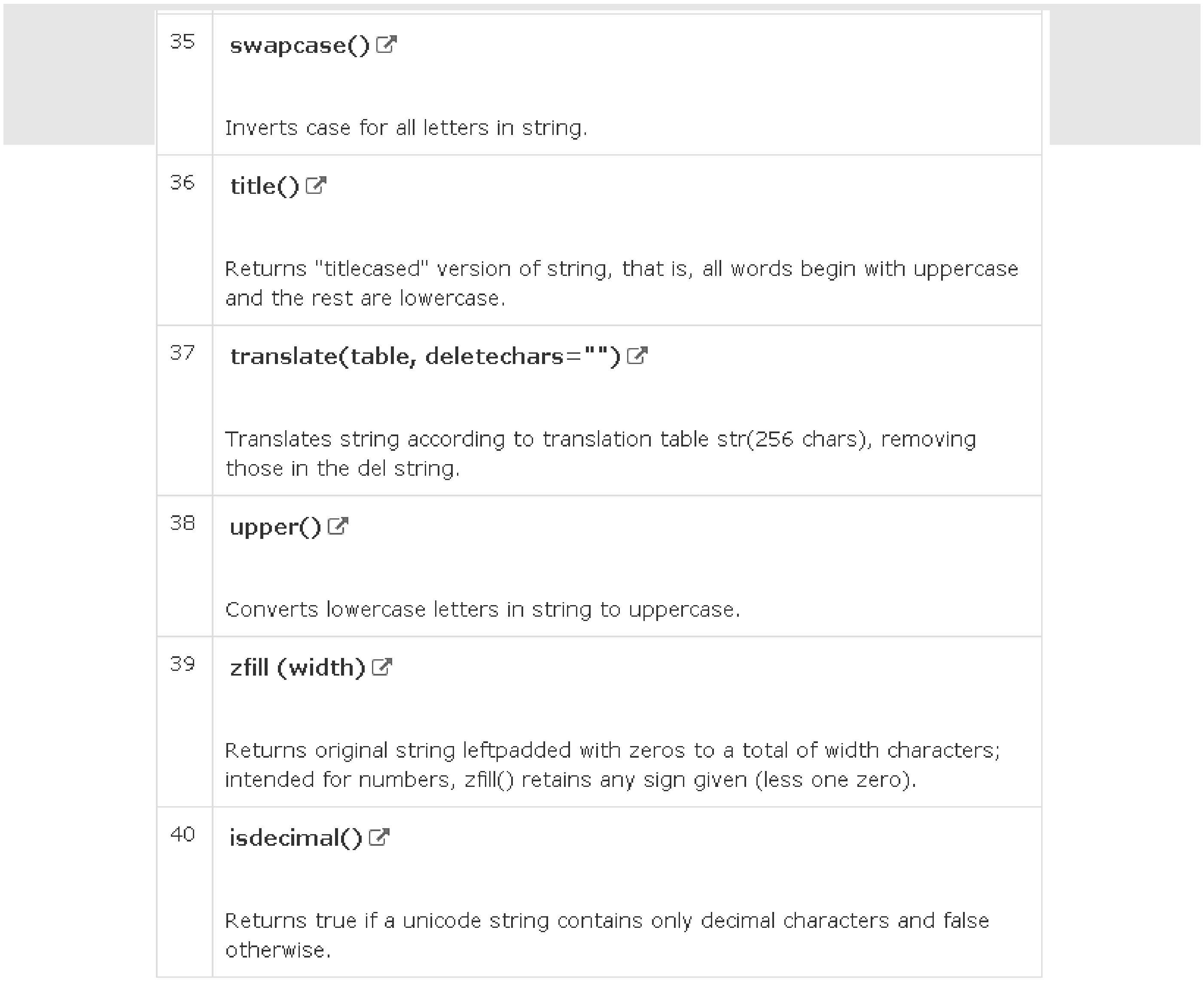
15	**isspace()** Returns true if string contains only whitespace characters and false otherwise.
16	**istitle()** Returns true if string is properly "titlecased" and false otherwise.
17	**isupper()** Returns true if string has at least one cased character and all cased characters are in uppercase and false otherwise.
18	**join(seq)** Merges (concatenates) the string representations of elements in sequence seq into a string, with separator string.
19	**len(string)** Returns the length of the string
20	**ljust(width[, fillchar])** Returns a space-padded string with the original string left-justified to a total of width columns.



21	**lower()** Converts all uppercase letters in string to lowercase.
22	**lstrip()** Removes all leading whitespace in string.
23	**maketrans()** Returns a translation table to be used in translate function.
24	**max(str)** Returns the max alphabetical character from the string str.
25	**min(str)** Returns the min alphabetical character from the string str.
26	**replace(old, new [, max])** Replaces all occurrences of old in string with new or at most max occurrences if max given.
27	**rfind(str, beg=0,end=len(string))** Same as find(), but search backwards in string.



28	**rindex(str, beg=0, end=len(string))** Same as index(), but search backwards in string.
29	**rjust(width,[, fillchar])** Returns a space-padded string with the original string right-justified to a total of width columns.
30	**rstrip()** Removes all trailing whitespace of string.
31	**split(str="", num=string.count(str))** Splits string according to delimiter str (space if not provided) and returns list of substrings; split into at most num substrings if given.
32	**splitlines(num=string.count('\n'))** Splits string at all (or num) NEWLINEs and returns a list of each line with NEWLINEs removed.
33	**startswith(str, beg=0,end=len(string))** Determines if string or a substring of string (if starting index beg and ending index end are given) starts with substring str; returns true if so and false otherwise.
34	**strip([chars])** Performs both lstrip() and rstrip() on string

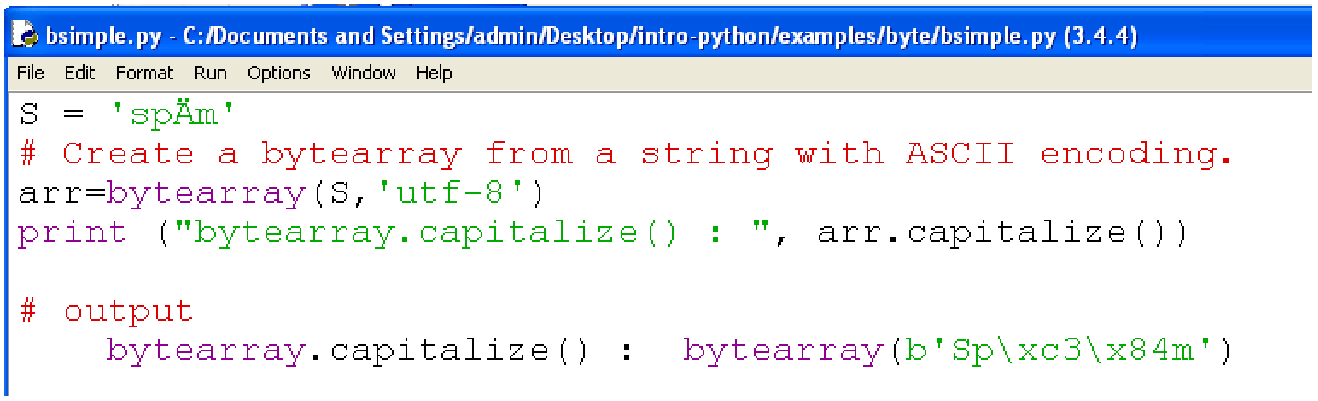


35	**swapcase()** Inverts case for all letters in string.
36	**title()** Returns "titlecased" version of string, that is, all words begin with uppercase and the rest are lowercase.
37	**translate(table, deletechars="")** Translates string according to translation table str(256 chars), removing those in the del string.
38	**upper()** Converts lowercase letters in string to uppercase.
39	**zfill (width)** Returns original string leftpadded with zeros to a total of width characters; intended for numbers, zfill() retains any sign given (less one zero).
40	**isdecimal()** Returns true if a unicode string contains only decimal characters and false otherwise.

Type specific Methods

- capitalize
- Count
- Find
- Index
- Decode and Encode
- Append, insert, and del
- Replace
- Startwith and endwith
- Split
- join

Type specific Methods


bsimple.py - C:/Documents and Settings/admin/Desktop/intro-python/examples/byte/bsimple.py (3.4.4)
File Edit Format Run Options Window Help

```
S = 'spÄm'
# Create a bytearray from a string with ASCII encoding.
arr=bytearray(S,'utf-8')
print ("bytearray.capitalize() : ", arr.capitalize())

# output
    bytearray.capitalize() :  bytearray(b'Sp\xc3\x84m')
```

Type specific Methods

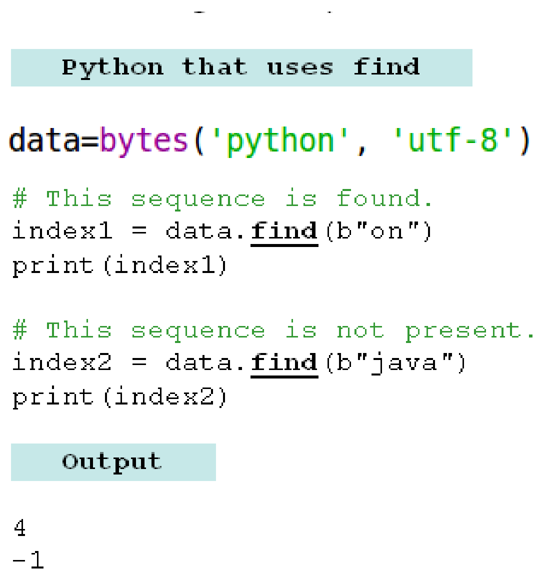
Python that uses count, buffer interface

```
# Create a bytes object and a bytearray.
data=bytes('aabbcccc', 'utf-8')
arr=bytearray('aabbcccc', 'utf-8')

# The count method (from the buffer interface) works on both.
print(data.count(b"c"))
print(arr.count(b"c"))
```

Output

```
4
4
```

Type specific Methods



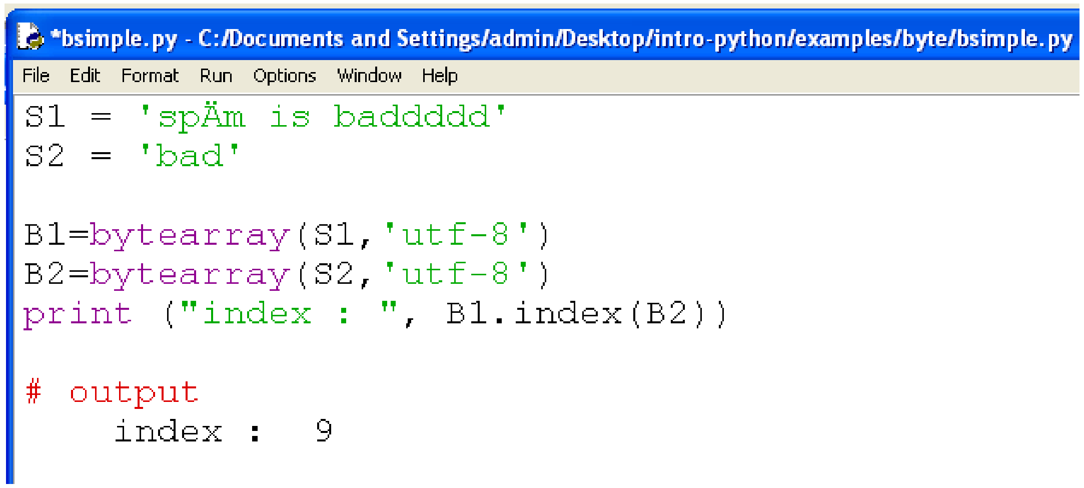
Python that uses find

```
data=bytes('python', 'utf-8')

# This sequence is found.
index1 = data.find(b"on")
print(index1)

# This sequence is not present.
index2 = data.find(b"java")
print(index2)
```

Output

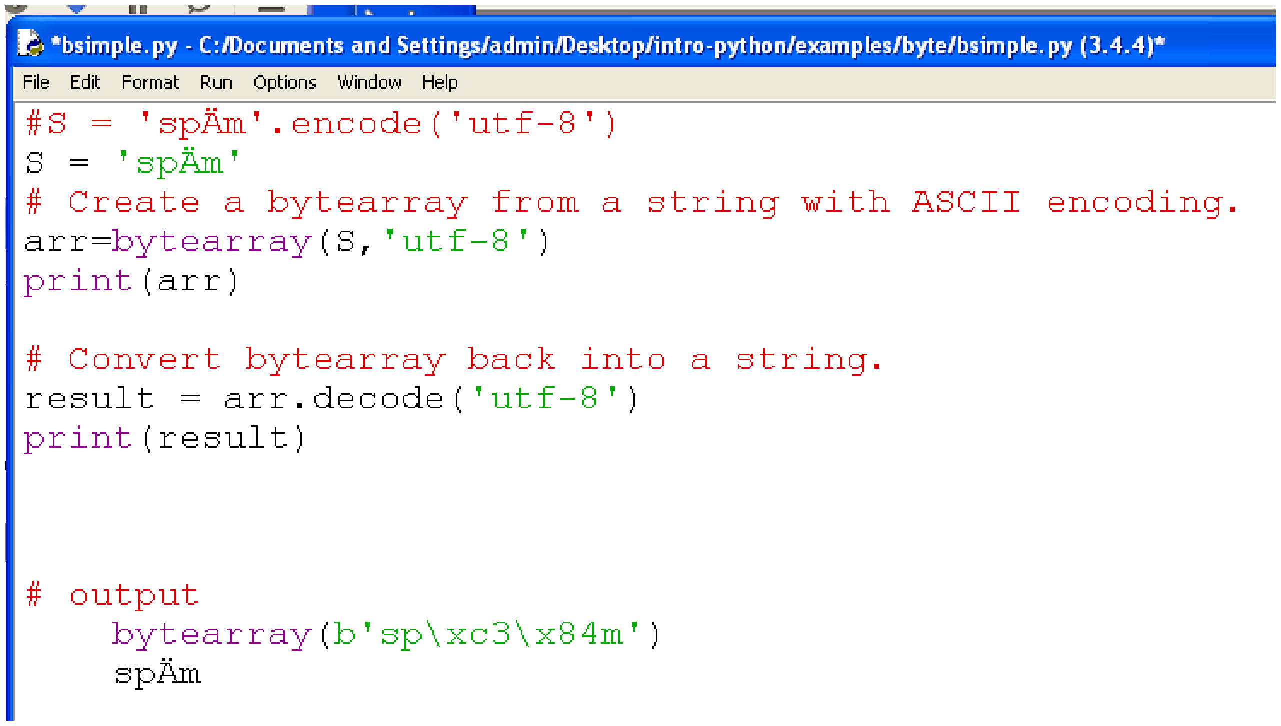
```
4
-1
```

Type specific Methods


*bsimple.py - C:/Documents and Settings/admin/Desktop/intro-python/examples/byte/bsimple.py
File Edit Format Run Options Window Help

```
S1 = 'spÄm is badddddd'
S2 = 'bad'

B1=bytearray(S1,'utf-8')
B2=bytearray(S2,'utf-8')
print ("index : ", B1.index(B2))

# output
    index :   9
```

Type specific Methods


bsimple.py - C:/Documents and Settings/admin/Desktop/intro-python/examples/byte/bsimple.py (3.4.4)
File Edit Format Run Options Window Help

```
#S = 'spÄm'.encode('utf-8')
S = 'spÄm'
# Create a bytearray from a string with ASCII encoding.
arr=bytearray(S,'utf-8')
print(arr)

# Convert bytearray back into a string.
result = arr.decode('utf-8')
print(result)



# output
    bytearray(b'sp\xc3\x84m')
    spÄm
```

Type specific Methods

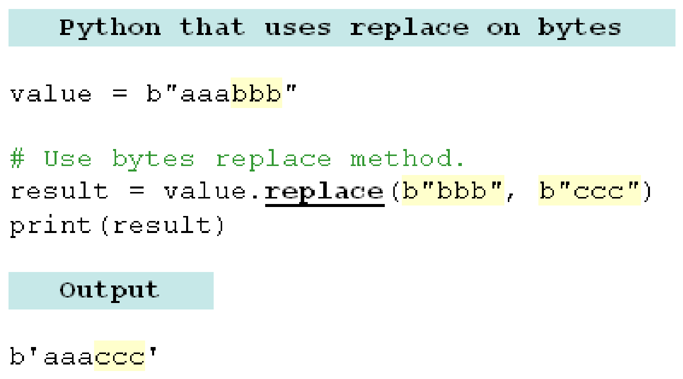
Python that uses append, del, insert

```
# Create bytearray and append integers as bytes.
values = bytearray()
values.append(0)
values.append(1)
values.append(2)
print(values)

# Delete the first element.
del values[0:1]
print(values)

# Insert at index 1 the value 3.
values.insert(1, 3)
print(values)
```

Output

```
bytearray(b'\x00\x01\x02')
bytearray(b'\x01\x02')
bytearray(b'\x01\x03\x02')
```

Type specific Methods



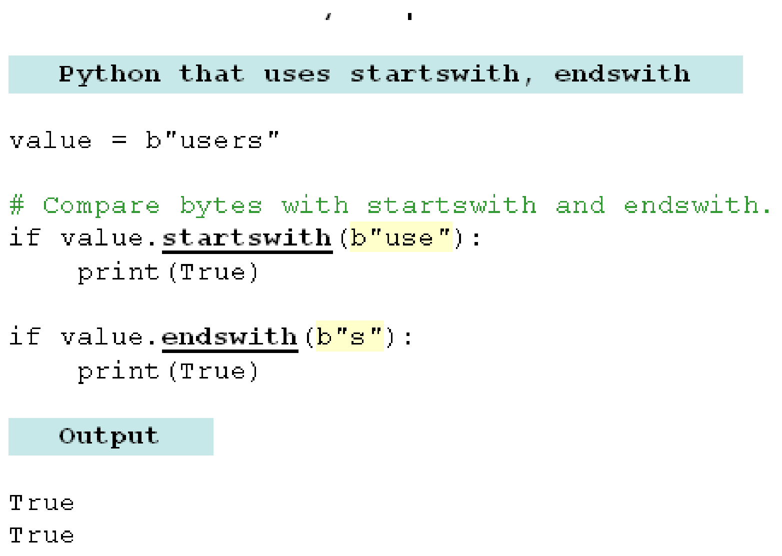
Python that uses replace on bytes

```python
value = b"aaabbb"

# Use bytes replace method.
result = value.replace(b"bbb", b"ccc")
print(result)
```

Output

```
b'aaaccc'
```

Type specific Methods



Python that uses startswith, endswith

```python
value = b"users"

# Compare bytes with startswith and endswith.
if value.startswith(b"use"):
    print(True)

if value.endswith(b"s"):
    print(True)
```

Output

```
True
True
```

Type specific Methods

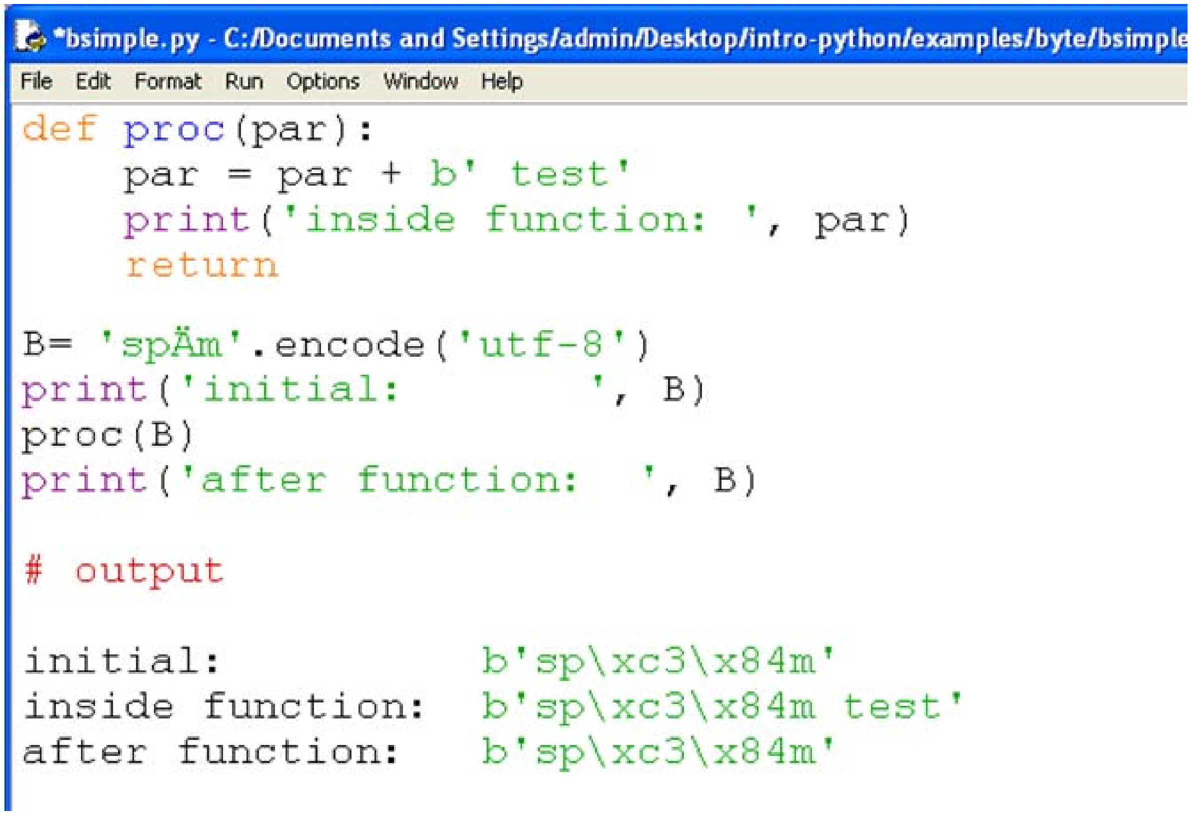
Python that uses split, join

```
# A bytes object with comma-separate values.
data = b"cat,dog,fish,bird,true"

# Split on comma-byte.
elements = data.split(b",")

# Print length and list contents.
print(len(elements))
print(elements)

# Combine bytes objects into a single bytes object.
result = b",".join(elements)
print(result)
```

Output

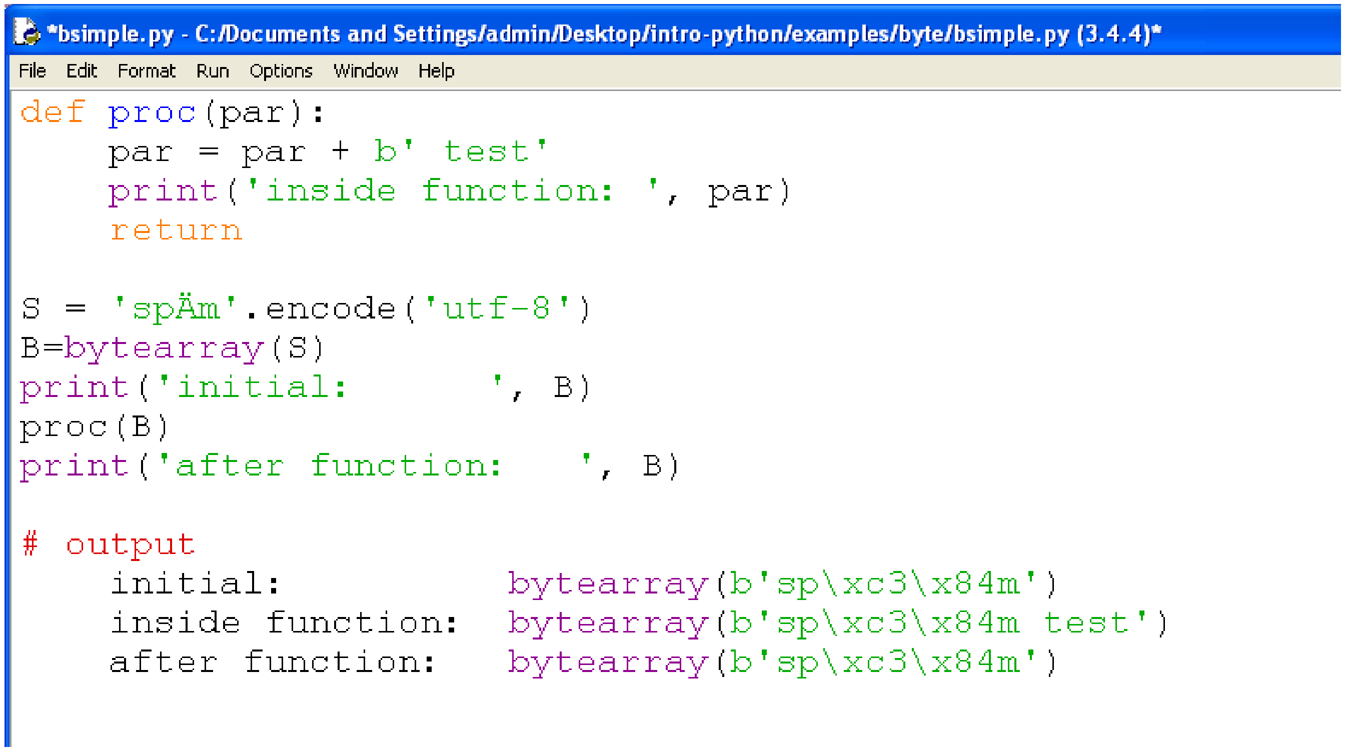
```
5
[b'cat', b'dog', b'fish', b'bird', b'true']
b'cat,dog,fish,bird,true'
```

Byte as a parameter


*bsimple.py - C:/Documents and Settings/admin/Desktop/intro-python/examples/byte/bsimple
File Edit Format Run Options Window Help

```
def proc(par):
    par = par + b' test'
    print('inside function: ', par)
    return

B= 'spÄm'.encode('utf-8')
print('initial:          ', B)
proc(B)
print('after function:   ', B)

# output

initial:           b'sp\xc3\x84m'
inside function:   b'sp\xc3\x84m test'
after function:    b'sp\xc3\x84m'
```

Bytearray as a parameter


bsimple.py - C:/Documents and Settings/admin/Desktop/intro-python/examples/byte/bsimple.py (3.4.4)
File Edit Format Run Options Window Help

```
def proc(par):
    par = par + b' test'
    print('inside function: ', par)
    return

S = 'spÄm'.encode('utf-8')
B=bytearray(S)
print('initial:       ', B)
proc(B)
print('after function:    ', B)

# output
    initial:             bytearray(b'sp\xc3\x84m')
    inside function:     bytearray(b'sp\xc3\x84m test')
    after function:      bytearray(b'sp\xc3\x84m')
```

Bytearray as a parameter

```
def proc(par):
    par[0]=112
    print('inside function: ', par)
    return

S = 'spÄm'.encode('utf-8')
B=bytearray(S)
print('initial:        ', B)
proc(B)
print('after function:  ', B)


# output
    initial:          bytearray(b'sp\xc3\x84m')
    inside function:  bytearray(b'pp\xc3\x84m')
    after function:   bytearray(b'pp\xc3\x84m')
```

End