SAMS
Programming A/B

Week 3 Lecture – Strings
July 17, 2017

Mark Stehlik

Strings

- We have already seen strings – they are sequences of characters delimited by ' and ' or " "
- Let's take a closer look…

String literals

- A string literal is anything in quotes
- But everything in the computer is stored in binary, so each character is stored as a number
- Examples:

```
ord("a")  ->  97
chr(97)   ->   a
ord("b")  ->  98
ord("A") ->  65
"a" < "A"  ->  False
```

ASCII values

ASCII TABLE

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[ENG OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

How might we use this?

```
def toUpperCaseLetter(character):
        if ("a" <= character <= "z"):
                return chr(ord(character) – 32)
        return character
```

Escape sequences

- Escape sequences:
  - single quote \'
  - double quote \"
  - backslash \\
  - newline \n
  - tab \t

String operators

- Operators:
  - Concatenation +
  - Multiple concatenation *
  - Length len (a function)
  - Indexing [valid values are -len(s) to len(s) -1]
    - <string>[n]
      - gives you the character at position n (starting from 0)
    - <string>[-n]
      - gives you the character at position len(<string>)-n
    - examples…

String Indexing

s = "Professor Mark"

len(s) -> 14 (so valid indices are -14 .. 13)

s[0] -> P

s[len(s)-1] -> k

s[-1] -> k

s[-14] -> P

s[42] -> error

More string operators

- Slicing
  - <string>[start:end:step]
    - gives you the substring beginning at start up to but not including end
  - Examples

    s = "Professor Mark"

    s[10:12] -> Ma

    s[10:] -> Mark

    s[:10] -> Professor (with the space)

More string operators

- Contains
  - in
    - "ark" in "Mark" -> True
    - "Mark" in "Professor Mark" -> True
    - "Mark" in "Professor" -> False
  - not in (this is OK in Python, as opposed to not (c in s))
    - not "Mark" in "Professor" -> True
    - "Mark" not in "Professor" -> True

Strings are immutable

- A string, once created, cannot be modified

```
s = "abcd"
s[0] = "d"  # error!
```

- But s can hold a different, new string…

```
s += "efg"
print(s)   # prints "abcdefg"  Why?
```

Suppose I wanted to reverse the contents of a string variable? How could I do that?

Strings and loops

- Iterating over a string with a for loop
  - likely to use len( )
  - an example

```
for i in range(len(<string-variable>)):
        print(i, s[i])
```

  - a different way to iterate over a string (if position is not needed):

```
for c in <string-variable>:
        print(c)
```

  - examples: let's write isInteger() and isPalindrome()

String constants

- String constants (must do what to use these?):
  - string.ascii_letters 'a..zA..Z'
  - string.ascii_lowercase 'a..z'
  - string.ascii_uppercase 'A..Z'
  - string.digits '0123456789'
  - string.punctuation lots of things ☺
  - string.whitespace space, tab, return
  - string.printable letters + digits + punc + whitesp

String methods (v. functions, constants)

- String functions and methods
  - Functions take a string as a parameter, e.g.,
    - len( ) – takes what as a parameter? returns what?
    - input( ) – takes what as a parameter? returns what?
  - Methods operate *on a particular* string, e.g.,
    - <str>.find( ) [and <str>.replace ( ), <str>.count( )]
    - <str>.isdigit( ) [.isalpha(), .islower(), .isupper(), .isspace()]
    - <str>.lower( ) [and <str>.upper( ), <str>.capitalize( )]
    - <str>.split( ) [and <str>.strip ( )]
- https://docs.python.org/3/library/stdtypes.html?highlight=strip#string-methods

String Formatting

- How to create a *formatting* string, which looks like:
  - "format_string" % (values)
  - The format_string contains conversion specifiers
    - %s – string
    - %d – integer
    - %f – floating point
    - %c – single character
    - %% – the character '%'
  - Specifiers can be preceded by optional width.precision

String Formatting

- If there is more than one conversion specifier, (values) must have the same number of items, and is called a *tuple*
- For each conversion specifier (%x) in the format string, there must be a corresponding value in the tuple of values
- Examples:
  - s = last + ", " + first
  - s2 = "%s, %s" % (last, first)
  - s3 = "%s\t%s" % (first, last)
  - s4 = "|%s|%s|%s|" % ("x", "o", "x")
  - s5 = "Pi is approximately %0.2f" % (22/7)