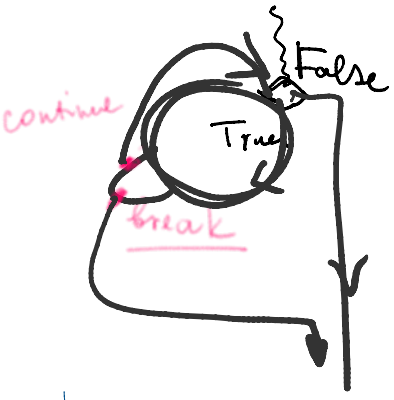
Class 7-8 Strings

Tuesday, February 20, 2024 9:17 AM

- Please, put cellphones away from your desk
- Feel free to interrupt with a question
- Raise hand to respond to a question

Practice problems due Fri, Handout 4, 1-7.



word = 'March'

word = word.upper()

word 'March'

'MARCH'

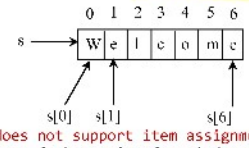
Handout 4

Strings and string methods.

- A string (type – str) is a sequence of characters, each character is also of string type.
- *String* literals can be created using matching *single quotes* (') or *double quotes* (").
  e.g. `"Good morning", 'A', '34', "56.87"`
- Some special characters:
  \n – newline \t – tab \\ - denotes \ \'– denotes ' \" - denotes "
- Python strings are **immutable**, i.e. methods and operations do not change the string (instead, they create new ones as a result)

BASIC PYTHON FUNCTIONS FOR STRINGS AND OTHER SEQUENCES

```
>>> s = "Welcome"
>>> len(s)
7
>>> s[0]
W
>>> s[0] = "w"
TypeError: 'str' object does not support item assignment
>>> s[3 : 6]  #slicing-part of the string from index 3 to index 6
'com'
>>> 'Wel' in s
True
>>> 'X' in s
False
>>> s1 = s + " to Bentley."
>>> s1
>>> "Welcome to Bentley."
>>> s2 = 2 * s
>>> s2
'WelcomeWelcome'
>>> s[-2] #negative index. Count positions from the end: len(s)-2
'm'
>>> s[-3 : -1]
'om'
```



[from : to : step]

CHARACTERS AND NUMBERS - CONVERSION FUNCTIONS

- Python does not have a data type for characters. A single-character string represents a character. Some other languages denote a single character with a single char, hence the book follows the same convention.
- Python characters use *Unicode*, a 16-bit encoding scheme in which each symbol is numbered. That number is the *code* number of the symbol.
  Unicode is an encoding scheme for representing international characters.
  ASCII is a small subset of Unicode.

it's code / number

symbol

ASCII Table

Dec	Hex	Oct	Chr	Dec	Hex	Oct	HTML	Chr	Dec	Hex	Oct	HTML	Chr	Dec	Hex	Oct	HTML	Chr
0	0	000	NULL	32	20	040	&#032;	Space	64	40	100	&#064;	@	96	60	140	&#096;	`
1	1	001	Start of Header	33	21	041	&#033;	!	65	41	101	&#065;	A	97	61	141	&#097;	a
2	2	002	Start of Text	34	22	042	&#034;	"	66	42	102	&#066;	B	98	62	142	&#098;	b
3	3	003	End of Text	35	23	043	&#035;	#	67	43	103	&#067;	C	99	63	143	&#099;	c
4	4	004	End of Transmission	36	24	044	&#036;	$	68	44	104	&#068;	D	100	64	144	&#100;	d
5	5	005	Enquiry	37	25	045	&#037;	%	69	45	105	&#069;	E	101	65	145	&#101;	e
6	6	006	Acknowledgment	38	26	046	&#038;	&	70	46	106	&#070;	F	102	66	146	&#102;	f
7	7	007	Bell	39	27	047	&#039;	'	71	47	107	&#071;	G	103	67	147	&#103;	g
8	8	010	Backspace	40	28	050	&#040;	(	72	48	110	&#072;	H	104	68	150	&#104;	h
9	9	011	Horizontal Tab	41	29	051	&#041;	)	73	49	111	&#073;	I	105	69	151	&#105;	i
10	A	012	Line feed	42	2A	052	&#042;	*	74	4A	112	&#074;	J	106	6A	152	&#106;	j
11	B	013	Vertical Tab	43	2B	053	&#043;	+	75	4B	113	&#075;	K	107	6B	153	&#107;	k
12	C	014	Form feed	44	2C	054	&#044;	,	76	4C	114	&#076;	L	108	6C	154	&#108;	l
13	D	015	Carriage return	45	2D	055	&#045;	-	77	4D	115	&#077;	M	109	6D	155	&#109;	m
14	E	016	Shift Out	46	2E	056	&#046;	.	78	4E	116	&#078;	N	110	6E	156	&#110;	n
15	F	017	Shift In	47	2F	057	&#047;	/	79	4F	117	&#079;	O	111	6F	157	&#111;	o
16	10	020	Data Link Escape	48	30	060	&#048;	0	80	50	120	&#080;	P	112	70	160	&#112;	p
17	11	021	Device Control 1	49	31	061	&#049;	1	81	51	121	&#081;	Q	113	71	161	&#113;	q
18	12	022	Device Control 2	50	32	062	&#050;	2	82	52	122	&#082;	R	114	72	162	&#114;	r
19	13	023	Device Control 3	51	33	063	&#051;	3	83	53	123	&#083;	S	115	73	163	&#115;	s
20	14	024	Device Control 4	52	34	064	&#052;	4	84	54	124	&#084;	T	116	74	164	&#116;	t
21	15	025	Negative Ack.	53	35	065	&#053;	5	85	55	125	&#085;	U	117	75	165	&#117;	u
22	16	026	Synchronous idle	54	36	066	&#054;	6	86	56	126	&#086;	V	118	76	166	&#118;	v
23	17	027	End of Trans. Block	55	37	067	&#055;	7	87	57	127	&#087;	W	119	77	167	&#119;	w
24	18	030	Cancel	56	38	070	&#056;	8	88	58	130	&#088;	X	120	78	170	&#120;	x
25	19	031	End of Medium	57	39	071	&#057;	9	89	59	131	&#089;	Y	121	79	171	&#121;	y
26	1A	032	Substitute	58	3A	072	&#058;	:	90	5A	132	&#090;	Z	122	7A	172	&#122;	z
27	1B	033	Escape	59	3B	073	&#059;	;	91	5B	133	&#091;	[	123	7B	173	&#123;	{
28	1C	034	File Separator	60	3C	074	&#060;	<	92	5C	134	&#092;	\	124	7C	174	&#124;	|
29	1D	035	Group Separator	61	3D	075	&#061;	=	93	5D	135	&#093;	]	125	7D	175	&#125;	}
30	1E	036	Record Separator	62	3E	076	&#062;	>	94	5E	136	&#094;	^	126	7E	176	&#126;	~
31	1F	037	Unit Separator	63	3F	077	&#063;	?	95	5F	137	&#095;	_	127	7F	177	&#127;	Del

asciicharstable.com

'Apple' < 'apple'
True

table < tablular
True

for establishing/comparing alphabetically
str 1 str 2
Must have same lettercase

>= > < == !=

ord(ch) returns a number corresponding to a symbol (*Unicode/ASCII* table code)
chr(num) returns a string with the character corresponding to the num code
str(num) produces a string version of num

```
>>> ch = 'a'
>>> ord(ch)
 97
>>> chr(98)
 'b'
>>> s = str(3.4) # Convert a float to string
>>> s
  '3.4'
>>> s = str(3) # Convert an integer to string
>>> s
   '3'
```

PRACTICE PROBLEMS

1. Assume **Bentley course section id** is a combination of two letters, designating a subject, followed by a 3 digit course number, then section number after a dash. Given a string defining a course section, output the components separated as shown:
   Input: cs230-006
   Output: Subject: CS Course Number: 230 Section: 6

2. Repeat the operation in 1, until user enters -1. Output how many CS courses above 100 level were entered. For example,
   ```
   Enter course section: CS213-001
   Enter course section: CS350-002
   Enter course section: CS180-001
   Enter course section: CS230-004
   Enter course section: -1
   There were 3 courses above 100 level.
   ```

3. Repeat reading course section ids as described in 1. Output a string consisting of CS courses above 100 level from the entered. For example,
   ```
   Enter course section: CS213-001
   Enter course section: CS350-002
   Enter course section: CS180-001
   Enter course section: CS230-004
   Enter course section: -1
   CS213 CS350 CS230
   ```

4. Generate a 8-char long password that includes 3 uppercase, 3 lowercase letters and two digits.
   Hint: Produce a random character using random integer generator. To generate a random integer **num** in the range from number 5 to number 10, inclusively, include the following code:
   ```
   import random #put in the top of the program
   num = random.randint(5, 11)
   ```

STRING METHODS

Method – a function that is **called by an object.** In the following, s is the calling object, where upper is the method

```
>>> s = "Welcome"
>>> s.upper()
'WELCOME'
```

ON the other hand, len() is a function, not a method, because it does not require a calling object.

- Recall, that Python strings are **immutable**, i.e. methods and operations do not change the string (instead, they create new ones as a result).

What is the value of s after the above segment has been executed?

Modifying and formatting - the titles of the methods are pretty self-explanatory

`capitalize()`	`lstrip()`
`lower()`	`rstrip()`
`upper()`	`strip()`
`title()`	`center(width)`
`swapcase()`	`ljust(width)`
`replace(old, new)`	`rjust(width)`
	`format(items)`

Testing characters in a function – return a boolean value: True or False

Assuming s is an object of type str,
method returns True iff s **has at least one character and**

- `s.isalnum()` all characters in s are **alphanumeric**
- `s.isalpha()` all characters in s are **alphabetic**
- `s.isdigit()` s contains only **number** characters.
- `s.islower()` s contains only **lowercase** letters.
- `s.isupper()` s contains only **uppercase** letters.
- `s.isspace()` s contains only **whitespace** characters (newlines, spaces, tabs, etc).

PRACTICE PROBLEMS

5. Check if a word entered by a user defines a phone number, e.g.

```
213-001-2345
is a phone number
```

or

```
CIS-210001
Is not a phone number
```

6. User will enter a string. Print how many letters, how many digits, spaces and other symbols are in it. Hint: examine it character-by-character using a loop
7. User will enter a string. Replace all punctuation in it.
   a. Simpler version: Assume punctuation symbols are `,.:;!?.` Are you using a loop to do it ?

b. Extra challenge: Assume a punctuation symbol is anything that is not a letter, a digit, or a space. *Hint*: first, go through the string, examining every character and collecting a string of punctuation symbols. After that, in a loop, replace every punctuation symbol.

Searching for Substrings.

Assuming s and s1 are strings. [ ] mean parameter is optional

`s.endswith(s1)`	returns True if s **ends** with s1
`s.startswith(s1)`	returns True if the string **starts** with s1
`s.find(s1 [,start[,end]])`	Returns the **lowest** index where s1 starts in this string, between positions **`start`** and **`end`**, or -1 if s1 is not found in this string.
`s.rfind(s1 [,start[,end]])`	Returns the **highest** index where s1 starts in this string between positions **`start`** and **`end`**, or -1 if s1 is not found in this string.
`s.count(s1)`	Returns the **number of non-overlapping occurrences** of s1

PRACTICE PROBLEMS

8. User will be entering email addresses one per line, followed by word STOP. For each entry, separate the username from the domain. Print "Bentley" if the email ends with bentely.edu
9. User will enter string defining distance in either inches or feet until -1 is entered. Produce the total length in inches, e.g.

```
Enter next value: 3 feet
Enter next value: 0.5 Feet
Enter next value: 5 inches
Enter next value: 2 FEET
Enter next value: -1
Total 71 inches
```

Extra challenges:

a) assume there may be any number of spaces separating the numeric value from the string

b) allow both feet and inches to be entered on one line, e.g.

```
3 feet 2 inches
```

or, even

```
3 inches   7 feet
```

Extra -- Method split()

```
s.split(sep=None, maxsplit=-1)
      sep – optional parameter, separator between the words
      maxsplit – optional parameter – number of seps considered
```

Return a list of words (***word*** *is a sequence of characters not equal to* ***sep***) in string s, separated by sep.

If separator is not specified, **all runs of consecutive whitespace are regarded as a single separator.**

If **`maxsplit`** is given, at most maxsplit splits are done (i.e. consider only the first **`maxsplit`** separators, thus, the list will have at most maxsplit+1 elements). If **`maxsplit`** is not specified or -1, then there is no limit on the number of splits (all possible splits are made).

```
>>>words = "Welcome  to    the  US\n".split()
>>>words
       ['Welcome', 'to', 'the', 'US']
>>>words[0]
      'Welcome'
>>>words[1]
      'to'
>>>words[-1]
      'US'
>>>"34-13-foo-45".split("-")  # - used as a separator
      ['34', '13', 'foo', '45']
```