1 Strings: type `str`

1.1 Introduction

A string is a sequence of characters. We have seen some strings in our print statements:

```
print "hello"
```

We've seen a string variable:

```
name = raw_input("What's your name? ")
```

1.2 Basics

We define strings using either single or double quotes:

`'csc108'` or `"csc108"`

If the string needs to span more than one line and then we use ***triple*** quotes:

```
'''This is a very, very long sentence that seems to run on and on and on
and on.'''
```

More string basics:

```
dept = 'csc'
num = 108
dept + 'num'
'cscnum'
dept + str(num)
'csc108'
int('108')
108
int('csc')
Traceback (most recent call last):
  File "<string>", line 1, in <fragment>
ValueError: invalid literal for int() with base 10: 'csc'
int('34.2')
Traceback (most recent call last):
  File "<string>", line 1, in <fragment>
ValueError: invalid literal for int() with base 10: '34.2'
float('45.9')
45.9
float('45')
45.0
"Bwa" + "ha" * 3
'Bwahahaha'
'ha' in 'Bwahahaha'
True
'HA' in 'Bwahahaha'
False
# 3 operators: +, *, in
len('hello')
5
len('hello this world')
16
```

2 Loops

Sometimes we need *walk* or *iterate* through each character in a string. This is called ***looping*** over the string.

ex:
```
word = 'computer'
for char in word:
 print char
```

```
c
o
m
p
u
t
e
r
```

The form for looping over a string:

```
for char in s:
      <do something involving char>
```

`for, in` are Python keywords.
`char` is a *variable* that refers to the value of the current character.
`s` is a string or a variable of type **`str`**

Example: [**`count_e.py`**]

```
if __name__ == '__main__':

    sentence = raw_input('Enter a sentence: ')

    e_count = 0
    # This loop executes len(sentence) times.
    for char in sentence:
        if char == 'e':
            e_count += 1
            # e_count = e_count + 1

    # print "There were " + str(e_count) + ' e characters. '
    print "There were", str(e_count), 'e characters. '
```

Q. How many times does the loop execute?

A.

2.1 Practise working with strings

Complete each of the following functions according to their docstring descriptions:

```
def num_vowels(s):
    '''(str) -> int
    Return the number of vowels in s.
    Do not treat the letter "y" as a vowel.'''

    count_vowels = 0
    for char in s:
        if char in 'aeiouAEIOU':
            # if char == 'a' or char == 'e' or char == 'i' or char == 'o' or char == 'u'
            # we can also use if and elif!
            count_vowels += 1
    return count_vowels
```

```
def reverse(s):
    '''(str) -> str
    Return a new string that is s in reverse.'''

    rev = ''
    for char in s:
      rev = char + rev

    return rev
```

```
def remove_spaces(s):
   '''(str) -> str
   Return a new string that is the same as s but with any blanks
   removed.'''
   new_str = '' #accumulator

     for char in s:
       if char != ' ':
          new_str += char   #new_str = new_str + char
     return new_str
```

```
def num_matches(s1, s2):
   '''(str, str) -> int
   Return the number of characters in s1 that appear in s2.'''
    count = 0
    for char in s1:
      if char in s2:
         count += 1
    return count

  if __name__ == '__main__':
    remove_spaces("Hi there")
```

2.2 Comparing strings

Strings are ***comparable*** in that lowercase letters increase in alphabetic order, as do uppercase letters which makes it easy to compare words. For example:

```
'a' > 'b' ; 'ant' <= 'apple' ; 'First' < 'Second'
```

Q. What about `'a'<'A'` or `',','<','!'` or `'!'<'a'`?

A. false, false, true

3 Indexing and Slicing

Strings are sequences of characters and the first character is at index 0 (i.e., at position 0). Each subsequent character has an index one greater. We can access characters or substrings using indices and slicing. Example:

```
s = "sliceofspam"
```

```
s l i c e o f s p a m
0_1_2_3_4_5_6_7_8_9_10
```

Index/Slice	Output	Explanation
`s[0]`	's'	
`s[3]`	'c'	
`s[-2]`	'a'	
`s[2:5]`	'ice'	
`s[3:]`	'ceofspam'	
`s[:8]`	'sliceofs'	
`s[:]`	'sliceofspam'	
`s[3:-2]`	'ceofsp'	
`s[-5:]`	'fspam'	
`s[5:2]`	'' # nothing output	
s [9: -2]	'' # nothing output	

3.1 Strings Are Immutable

[Strings slides 2-5]

`strings` are ***immutable*** (their state cannot be changed once they are defined). This is just like **`int`**s.

We can't do the following:

```
s[1] = "x"
s[2:5] = "zzz"
```

Q. What can we do instead?

```
s = 'sliceofspam'
s = 'hello'
id(s)
34252608
s = s[2:]
s
'llo'
id(s)
29907904
```

Indexing and slicing ALWAYS produce a brand-new string.

Exercise: what is the output?

```
s = "hello"
s2 = s
s = s[2:]
print s
print s2
```

Exercise: what is the output?

```
s = "abcde"
c = s[2]
c = "Z"
print c
print s
```

4 Methods

[Strings slides 6-11]

```
# String method descriptions
help(str)
```

Some examples:

```
help(str.lower)
Help on method_descriptor:

lower(...)
    S.lower() -> string

    Return a copy of the string S converted to lowercase.
s = "What a WONDERFUL morning"
s.lower()
'what a wonderful morning'
s
'What a WONDERFUL morning'
s.lower() # lower is a str method
'what a wonderful morning'
lower(s) # Error. lower is a method, not a function
Traceback (most recent call last):
  File "<string>", line 1, in <fragment>
NameError: name 'lower' is not defined
```

```
help(str.find)
Help on method_descriptor:

find(...)
    S.find(sub [,start [,end]]) -> int

    Return the lowest index in S where substring sub is found,
    such that sub is contained within s[start:end].  Optional
    arguments start and end are interpreted as in slice notation.

    Return -1 on failure.
```

```
"This is CSC???".replace('???', '108')
'This is CSC108'
'This is CSC108'.count('i')
2
help(str.count)
Help on method_descriptor:

count(...)
    S.count(sub[, start[, end]]) -> int

    Return the number of non-overlapping occurrences of substring sub in
    string S[start:end].  Optional arguments start and end are interpreted
    as in slice notation.

s = 'yabababababa'
s.count('aba')
3
s.count('aba', 0, 5)
1
```

```
help(str.strip)
Help on method_descriptor:

strip(...)
    S.strip([chars]) -> string or unicode

    Return a copy of the string S with leading and trailing
    whitespace removed.
    If chars is given and not None, remove characters in chars instead.
    If chars is unicode, S will be converted to unicode before stripping
```

```
"This is CSC108".find("is")
2
"This is CSC108".rfind("is")
5
"This is CSC108".find("is", 3)
5
"This is CSC108".find("is", 5)
5
"This is CSC108".find("is", 6)
-1
```

The first position that be found from left; while the lower method is from the right side!!!

-1 is always failure!!

4.1 Escape Sequences

Several characters have special meaning to Python (e.g., quotes, newline). To use them in our strings, we need to do something special:

```
\n
\'   \"
\t
\\
```

Exercise. Write a **`print`** statement to generate the following output:

```
Jen's CSC108 lecture sections:
L0101    L5101
```

Solution:

```
print 'Jen\'s CSC108 lecture sections:\nL0101\tL5101'
```

```
Jen's CSC108 lecture sections:
L0101 L5101
```

```
print
# Escape sequences
'''This is a string
that spans
multiples lines.'''
'This is a string\nthat spans\nmultiples lines.'
'This is a string\nthat spans\nmultiples lines.'
'This is a string\nthat spans\nmultiples lines.'
# n\ is newline
# \ is the escape character
'Jen\'s'
"Jen's"
'Jen\Campbell'
'Jen\\Campbell'
print 'Jen\\Campbell'
Jen\Campbell
```

5 Conversion Specifiers

Consider this code:

```
a1 = 89
a2 = 76
a3 = 91
print "The maximum of the three grades (", a1, ",", a2, "and", a3, ") \
is", max(a1, a2, a3)
```

The output:

```
 The maximum of the three grades ( 89 , 76 and 91 ) is 91
```

To get rid of the spaces inside the brackets, we need to concatenate (some of the time):

```
print "The maximum of the three grades (" + a1, ",", a2, "and", a3 + ") \
is", max(a1, a2, a3)
```

Q. Why doesn't this work?

A. cannot concatenate 'str' and 'int' objects!!!

Q. How can we fix it?

A. change a1 to str(a1), respectively, a2 to str(a2), and a3 to str(a3)

Another solution is to use conversion specifiers :

```
print "The maximum of the three grades (%d, %d and %d) is %d % (a1, a2, a3, max(a1, a2, a3)
The maximum of the three grades (89, 76 and 91) is 91
```

- Each “%d” is a placeholder for a decimal (i.e., base 10) integer.
- After the string (with its placeholders in it), you type “%” and then the list of values to substitute in, in parentheses. You need the parentheses even if there is only one value in the list.
- We call these placeholders “***conversion specifiers***”. They indicate the ***format*** (*e.g.,* integer, float, string) we want the values printed in.

```
print '%d' % (342)
 342
# %d: decimal integer (base 10 int)
print '%f' % (45.89)
45.890000
import math
print '%f' % (math.pi)
3.141593
print '%.2f' % (math.pi)
3.14
```

Q. The “%” is overloaded. What else is “%” used for?

A.

Q. What other two overloaded operators have we seen?

A.

Other conversion specifiers include:

%f	
%.2f	
%s	

Examples.