Strings

CSE 1310 – Introduction to Computers and Programming
Vassilis Athitsos
University of Texas at Arlington

String Comparisons

```
>>> my_strings = ["Welcome", "to", "the", "city", "of", "New",
    "York"]
>>> my_strings
['Welcome', 'to', 'the', 'city', 'of', 'New', 'York']
>>> my_strings.sort()
>>> my_strings
['New', 'Welcome', 'York', 'city', 'of', 'the', 'to']
```

- Python uses a string order of its own.
  - Follows alphabetical order, with the exception that capital letters **<u>are always before</u>** lower case letters.

String Comparisons

- It is easy to verify the order that Python uses, by trying out different pairs of strings.

```
>>> "hello" < "goodbye"
False
>>> "Hello" < "goodbye"
True


>>> "ab" > "abc"
False
```

String Comparisons

```
>>> "123" < "abc"
True
>>> "123" < "ABC"
True
```

- Numbers come before letters.

- Guideline: do not memorize these rules, just remember that Python does NOT use exact alphabetical order.

Strings Cannot Change

```
>>> a = "Munday"
>>> a[1] = 'o'

Traceback (most recent call last):
  File "<pyshell#297>", line 1, in <module>
    a[1] = 'o'
TypeError: 'str' object does not support item assignment
```

If You Must Change a String…

- You cannot, but you can make your variable equal to another string that is what you want.
- Example:

>>> my_string = "Munday"

  - my_string contains a value that we want to correct.

>>> my_string = "Monday"

  - We just assign to variable my_string a new string value, that is what we want.

For More Subtle String Changes…

- Suppose that we want a program that:
  - Gets a string from the user.
  - Replaces the third letter of that string with the letter A.
  - Prints out the modified string. We just assign to variable my_string a new string value, that is what we want.

For More Subtle String Changes…

- Strategy:
  - convert string to list of characters
  - do any manipulations we want to the list (since lists can change)
  - convert list of characters back to a string

An Example

- Write a program that:
  - Gets a string from the user.
  - Modifies that string so that position 3 is an A.
  - Prints the modified string.

An Example

```
my_string = raw_input("please enter a string: ")

if (len(my_string) >= 3):
   # convert string to list, make the desired change (change third letter to "A")
   my_list = list(my_string)
   my_list[2] = "A";

   # create a string out of the characters in the list
   new_string = ""
   for character in my_list:
      new_string = new_string + character

   my_string = new_string

print "the modified string is", my_string
```

A Variation

```
my_string = raw_input("please enter a string: ")

my_string = my_string[0:2] + "A" + my_string[3:]
print "the modified string is", my_string
```

The * Operator on Strings

```
>>> a = "hello"
>>> a*3
'hellohellohello'
```

- The **`string*integer`** expression repeats a string as many times as the integer specifies.

The **in** Operator

```
>>> a = [1, 2, 3]
>>> 2 in a
True
>>> 5 in a
False

>>> vowels = 'aeiou'
>>> "a" in vowels
True
>>> "k" in vowels
False
```

- The **in** operator works for lists and strings.
- Syntax:
  - **element in container**
- Returns **true** if the element appears in the container, false otherwise.

upper and **lower**

```
>>> vowels = 'aeiou'
>>> b = vowels.upper()
>>> vowels
'aeiou'
>>> b
'AEIOU'

>>> a = 'New York City'
>>> b = a.lower()
>>> b
'new york city'
```

- The string.**upper()** method returns a new string where all letters are upper case.
- The string.**lower()** method returns a new string where all letters are lower case.
- Note: upper() and lower() **do not modify the original string**, they just create a new string.
  - Should be obvious, because **strings cannot be modified.**

The **len** Function

```
>>> len('hello')
5
```

- Similar as in lists, **len** returns the number of letters in a string.

Reversing a String

```
>>> a = "hello"
>>> b = a[::-1]
>>> b
'olleh'
>>> a[3:0:-1]
'lle'
```

- Slicing with step -1 can be used to reverse parts, or all of the string.

index and **find**

```
>>> a = [10, 11, 12, 10, 11]
>>> a.index(10)
0
>>> a.index(11)
1


>>> b = "this is crazy"
>>> b.find('is')
2
>>> b.find('q')
-1
```

- The **my_list.index(X)** method returns the first position where X occurs.
  - Gives an error if X is not in my_list.
- The **my_string.find(X)** method returns the first position where X occurs.
  - X can be a single letter or more letters.
  - Returns -1 if X is not found.

Converting Other Types to Strings

```
>>> a = 2012
>>> b = str(a)
>>> b
'2012'
```

- The **str** function converts objects of other types into strings.

```
>>> a = ['h', 'e', 'l', 'l', 'o']
>>> b = str(a)
>>> b
"['h', 'e', 'l', 'l', 'o']"
```

- Note: **str** does **NOT** concatenate a list of characters (or strings). See example on left.

Converting Strings Into Ints/Floats

```
>>> a = '2012'
>>> b = int(a)
>>> b
2012
>>> float(a)
2012.0

>>> a = "57 bus"
>>> int(a)

<error message>
```

- The **int, float** functions converts strings to integers and floats.
  - Will give error message if the string does not represent an integer or float.

Converting Strings Into Lists

```
>>> a = "hello"
>>> list(a)
['h', 'e', 'l', 'l', 'o']
```

- The **list** function can convert a string to a list.
  - Always works.
  - **Very handy** if we want to manipulate a string's contents and create new strings based on them.