Strings

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

Strings Store Text

- In the same way that **int** and **float** are designed to store numerical values, the **string** type is designed to store text.
- Strings can be enclosed in: single quotes, double quotes, or triple double quotes.
- Examples:

```
name = 'George'

phone_number = "310-123-987"

message = """Please go shopping. We need milk,
cereal, bread, cheese, and apples. Also, put gas in
the car."""
```

A Simple Program Using Strings

```
text = input("please enter a day: ")

weekdays = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday']
weekend = ['Saturday', 'Sunday']

if (text in weekdays):
    print('This is a weekday')
elif (text in weekend):
    print('This is a weekend day')
else:
    print('This is not a valid day')
```

Accessing the Elements of a String

- Accessing elements of a string is done as in lists, using the [] operator.

```
>>> a = 'hello'
>>> a[0]
'h'
>>> a[2]
'l'
>>> a[1:3]
'el'
>>> a[-1]
'o'
>>> a[-2]
'l'
```

Accessing the Elements of a String

- Accessing elements of a string is done as in lists, using the [] operator.

```
>>> 'goodbye'[3]
'd'
>>> 'goodbye'[4:1:-1]
'bdo'
>>> 'goodbye'[::2]
'gobe'
>>> 'goodbye'[:4]
'good'
>>> 'goodbye'[4:]
'bye'
```

Concatenation Using The + Operator

- The **string1+string2** expression produces the concatenation of **string1** and **string2**.

```
>>> a = "hello"
>>> b = a + " " + "world"
>>> print(b)
hello world
```

Concatenation Using The += Operator

- The **string1 += string2** statement assigns to **string1** the concatenation of **string1** and **string2**.

```
>>> c = "Arlington"
>>> c += ", TX"
>>> print(c)
Arlington, TX
```

The * Operator on Strings

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

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

For Loops with Strings

- Print out all letters in a string.

```
text = "hello world"

for letter in text:
    print('found letter', letter)
```

OUTPUT:

```
found letter h
found letter e
found letter l
found letter l
found letter o
found letter
found letter w
found letter o
found letter r
found letter l
found letter d
```

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.

String Comparisons

```
>>> "123" == 123
```

- What will this line produce?

String Comparisons

```
>>> "123" == 123
False
```

- What will this line produce?
  - False, because a string cannot be equal to a number.

String Comparisons

```
>>> "123" < 150
```

- What will this line produce?

String Comparisons

```
>>> "123" < 150
Traceback (most recent call last):
  File "<pyshell#195>", line 1, in <module>
    "123" < 123
TypeError: unorderable types: str() < int()
```

- What will this line produce?
  - An error message, because comparisons between strings and numbers are illegal in Python.

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 = 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 = input("please enter a string: ")

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

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.

The **index** method

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

>>> b = "this is crazy"
>>> b.index('i')
2
>>> b.index('cr')
8
```

- 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.index(X)** behaves the same way, but:
  - X can be a single letter or more letters.

The **find** method

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

- The **my_string.find(X)** method, like **index**, returns the first position where X occurs.
  - X can be a single letter or more letters.
  - Difference from **index**: **my_string.find(X)** returns -1 if X is not found.

The **isspace** method

```
>>> b = "\t\n   \t"
>>> b.isspace()
True


>>> "hello".isspace()
False
```

- The **my_string.isspace()** method, returns **True** if the string only contains white space (space, tab, newline).
  - X can be a single letter or more letters.
  - " " is the space character.
  - "\t" is the tab character.
  - "\n" is the newline character.

The **strip** method

```
>>> a = "  hello   world   "
>>> b = a.strip()

>>> b
'hello   world'
```

- The **my_string.strip()** method, returns a string that is equal to my_string, except that white space (space, tab, newline) has been removed from the beginning and the end of my_string.
  - **White space in the middle of the string (between non-white-space characters) is not removed.**

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 convert strings to integers and floats.
  - Will give error message if the string does not represent an integer or float.

ASCII Codes

- Each letter corresponds to an integer, that is called the **ASCII code** for that letter.
- The **ord** function can be used to get the ASCII code of a letter.

ASCII Codes

- Each letter corresponds to an integer, that is called the **ASCII code** for that letter.
- The **ord** function can be used to get the ASCII code of a letter.

```
for i in 'hello world':
      print(i, ord(i))
```

OUTPUT:

```
h 104
e 101
l 108
l 108
o 111
  32
w 119
o 111
r 114
l 108
d 100
```

From ASCII Code to Character

- The **chr** function can be used to get the letter corresponding to an ASCII code.

From ASCII Code to Character

- The **chr** function can be used to get the letter corresponding to an ASCII code.

```
list1 = [104, 101, 108, 108, 111]
text = ""

for item in list1:
    text = text + chr(item)

print("text =", text)
```

OUTPUT:

```
text = hello
```

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.

Converting a List to a String

- To convert a list to a string, **do not** use the **str** function.

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

Converting a List to a String

- To convert a list to a string, use a **for** loop.

```
a = ['h', 'e', 'l', 'l', 'o']
b = ""

for letter in a:
    b = b+letter

>>> b
'hello'
```

Example: Strings to Lists and Back

```
# sort all the letters in a string.

text = input('Enter some text: ')
text_list = list(text)
text_list.sort()

new_text = ""
for letter in text_list:
    new_text = new_text + letter

print("The sorted text is:", new_text)
```

OUTPUT:

```
Enter some text: hello world
The sorted text is:  dehlllloorw
```