Strings, Formatting, and Files

Rose-Hulman Institute of Technology

Computer Science and Software Engineering

Check out **09-StringsFormattingAndFiles** from SVN

ROSE-HULMAN INSTITUTE OF TECHNOLOGY

Strings (sequences of characters)

- String literals (constants):
- `"One\nTwo\nThree"`
- `"Can't Buy Me Love"`
- `'I say, "Yes."  You say, "No." '`
- `"'A double quote looks like this \",' he said."`
- `"""I don't know why you say, "Goodbye,"
  I say "Hello." """`

Q1,2

ROSE-HULMAN INSTITUTE OF TECHNOLOGY

Operating on Strings

Reference

Operations/Methods	What does each of these operation/method do?
s1 + s2	Concatenates two strings e.g. "xyz" + "abc" ⇒ "xyzabc"
s * <int>	Replicates string s <int> times e.g. "xyz" * 4 ⇒ "xyzxyzxyzxyz"
s.capitalize()	Copy of s with only 1st letter capitalized
s.lower()	Copy of s with all lower case characters
s.reverse()	Copy of s will all characters reversed
s.split()	List of the words in s (split on spaces by default)
s.center(<int>, [<str>])	Copy of s padded on either end to be <int> characters long (padded with spaces by default, or else optional <str>)

Some more string methods

Reference

Methods	What does each of these operation/method do?
s.count(sub)	Returns the number of occurrences of sub in s
s.find(sub)	Returns the first position (index, 0-based) where sub occurs in s
s.title()	Copy of s with first character of each word capitalized
s.replace(old, new)	Copy of s where all occurrences of old in s have been replaced with new
s.lstrip()	Copy of s with leading white space removed
s.strip()	Copy of s with leading and trailing white space removed
s.join(list)	Concatenate list into a string, using s as the separator between items in the list

Practice with string operations

- **+** is used for String *concatenation*: **"xyz" + "abc"**
- * is used for String *duplication*: **"xyz " * 4**
  - **>>> franklinQuote = 'Who is rich? He who is content. ' + 'Who is content? Nobody.'**
  - **>>> franklinQuote.lower()**
  - **'who is rich? he who is content. who is content? nobody.'**
  - **>>> franklinQuote.replace('He', 'She')**
  - **'Who is rich? She who is content. Who is content? Nobody.'**
    >>> franklinQuote.find('rich')

Q3,4

ROSE-HULMAN INSTITUTE OF TECHNOLOGY

Review: Strings are immutable

- Lists are mutable:
  - >>> **colors = ["red", "white", "blue"]**
  - >>> **colors[1] = "grey"**
  - >>> **colors.append("cyan")**

- A string is an immutable sequence of characters
  - >>> **building = "Taj Mahal"**
  - >>> **building[2]**
  - >>> **building[1:4]**
  - >>> **building[4] = "B"** **# Error!**

Q5,6

ROSE-HULMAN INSTITUTE OF TECHNOLOGY

Strings and Lists

- A String method: **split** breaks up a string into separate words
  - >>> **franklinQuote = 'Who is rich? He who is content. ' + 'Who is content? Nobody.'**
  - >>> **myList = franklinQuote.split(' ')**
    ['Who', 'is', 'rich?', 'He', 'who', 'is', 'content.', 'Who', 'is', 'content?', 'Nobody.']

- A string method: join creates a string from a list
  - >>> **'#'.join(myList)**
  - >>> **'Who#is#rich?#He#who#is#content.#Who#is#content?#Nobody.'**

- What is the value of **myList[0][2]**? **myList[2][0]**?

Lists of Strings

```
>>> beatles = ['John', 'Paul']
>>> beatles.append('George')
>>> beatles
['John', 'Paul', 'George']
>>> beatles + ['Ringo']
['John', 'Paul', 'George', 'Ringo']
>>> beatles
['John', 'Paul', 'George']
```

```
>>> beatles = beatles + ['Ringo']
>>> beatles
['John', 'Paul', 'George', 'Ringo']
>>> beatles[1]
'Paul'
>>> beatles[1][2]
'u'
```

Getting a string from the user

```
>>> name = input('Enter your name:')
Enter your name:John
>>> name
'John'
>>>
```

You'll need input, string methods, lists, and print!

Q8

ROSE-HULMAN INSTITUTE OF TECHNOLOGY

String Representation

- Computer stores 0s and 1s
  - Numbers are stored as 0s and 1s
  - What about text?
- Text also stored as 0s and 1s
  - Each character has a code number
  - Strings are sequences of characters
  - Strings are stored as sequences of code numbers
  - Does it matter what code numbers we use?
- Translating: **ord(<char>)** and **chr(<int>)**

Q9,10

Consistent String Encodings

- Examples:
  - ASCII—American Standard Code for Info. Interchange
    - “Ask-ee”
    - Standard US keyboard characters plus “control codes”
    - 7 bits per character
  - Extended ASCII encodings (8 bits)
    - Add various international characters
  - Unicode (16+ bits)
    - Tens of thousands of characters
    - Nearly every written language known

String Formatting

- Use **format()** for complex text output
  - Uses a *template* string with *slots*
    - `t = "All {0} wants for {2} is his {1} front teeth"`
  - Values to format are plugged into each slot
  - The string method **format** does the plugging
    - **<template-string>.format(<values>)**
    - `t.format("Curt", 2, "Christmas")`

Returns a string, rather than printing it.

ROSE-HULMAN INSTITUTE OF TECHNOLOGY

String Formatting

- What does each slot look like?
  - **{<index>:<format-specifier>}**
  - **<index>** tells which of the parameters is inserted in slot
  - Optional **<format-specifier>** lets us do fancier formatting

Format Specifiers

- Syntax:
  - **<width>.<precision><typeChar>**
- **<width>**: total spaces to use
  - 0 (or omitted) means as many as needed
  - 0*n* means pad with leading 0s to *n* total spaces
  - <*n* means “left justify” in the *n* spaces
  - ^n means “center justify” in the n spaces
- **<precision>** gives digits after decimal point, rounding if needed.
- **<typeChar>** is:
  - **f** for float, **s** for string, or **d** for decimal (i.e., int)

Q12,13

funcDisplay.py

Live Coding

From HW9:

Prompt the user for a number of data points, *num*, to list. Write that many points to the screen, as described below. Each line should have the data:

n *200 + 200cos(nπ/180)* where *n* ranges from 0 to *num* - 1. Make sure the output is neatly formatted so that numbers and decimal points line up. Below is a sample of the expected output. Your program's output should match this format exactly.

```
 98   172.165
 99   168.713
100   165.270
```

ROSE-HULMAN INSTITUTE OF TECHNOLOGY

File Processing

- Manipulating data stored on disk
- Key steps:
  - *Open* file for reading or writing
    - Associates file on disk with a *file variable* in program
  - *Manipulate* the file with operations on file variable
    - Read or write information
  - *Close* file
    - Causes final “bookkeeping” to happen

File Writing in Python

- Open file:
  - Syntax: **<filevar> = open(<name>, <mode>)**
  - E.g., **outFile = open(‘coord.txt', 'w')**
- Write to file:
  - Syntax: **<filevar>.write(<string>)**
  - E.g., **outFile.write(“lat={0} long={1}”.format(39.4, -87.4)”))**
- Close file:
  - Syntax: **<filevar>.close()**
  - E.g., **outFile.close()**

This mode writes file *replacing* contents

File Reading in Python

- Open file: **inFile = open('grades.txt', 'r')**
- Read file:
  - **<filevar>.read()** Returns one BIG string
  - **<filevar>.readline()** Returns next line, including \n
  - **<filevar>.readlines()** Returns BIG list of strings, 1 per line
  - **for <id> in <filevar>** Iterates over lines efficiently
- Close file: **inFile.close()**

A “Big” Difference

readlines()

```
inFile = open ('grades.txt', 'r‘)
for line in inFile.readlines():
        # process line
inFile.close()
```

Looping on file directly

```
inFile = open ('grades.txt', 'r‘)
for line in inFile:
        # process line
inFile.close()
```

Which uses the most memory?

Q15,16

ROSE-HULMAN INSTITUTE OF TECHNOLOGY

Pair programming, but with new partners

New Pairing Partners

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY