Strings (Ch. 8)

CSC110: INTRO TO COMPUTER PROGRAMMING WITH PYTHON

INSTRUCTOR: BILL BARRY

1

R
o
a
d
m
a
p

☞ Strings

File I/O, Exceptions

Lists

List Algorithms

Dictionaries & Sets

Final Exam

2

This Week: Strings

Bill's Notes

Why a Chapter on Strings?

String Basics & Checking Membership

Testing Values, Modification, Search & Replace

Splitting

String Exercise

3

Bill's Notes

Navigator update: Friday sessions also via MSLC Zoom. Peter's sessions T/F 12pm–1pm

Exam/Project
- Clarify when to format, when to round, when to do neither
- Exam stats and common selection issues (slides follow)

College calendar
- Thursday, November 11, Veteran's Day; College closed
- Friday, November 19, 8am – 4:30pm: last day to change audit/credit status with instructor permission, last day to withdraw

Office hours today (10:50, same Zoom)
- Project 5 debrief (for those who have final feedback posted)

New approach to main()
- A good way to write the call to main, esp. if you're doing unit testing

```
if __name__ == "__main__":
    main()
```

4

Common Selection Issues

```
def percentToLetterGrade(percentGrade):

    if (percentGrade > 110):
        return "ERROR"
    if (percentGrade <= 110 and percentGrade >= 90):
        return "A"
    if (percentGrade < 90 and percentGrade >= 80):
        return "B"
    if (percentGrade < 80 and percentGrade >= 70):
        return "C"
    if (percentGrade < 70 and percentGrade >= 60):
        return "D"
    if (percentGrade < 60 and percentGrade >= 0):
        return "F"
    if (percentGrade < 0):
        return "ERROR"
```

```
def percentToLetterGrade(percentGrade):

    if percentGrade < 0 or percentGrade > 110:
        letter = "ERROR"
    elif percentGrade >= 90:
        letter = "A"
    elif percentGrade >= 80:
        letter = "B"
    elif percentGrade >= 70:
        letter = "C"
    elif percentGrade >= 60:
        letter = "D"
    else:
        letter = "F"
    return letter
```

5

Why a Whole Chapter on Strings?

WHY SPEND A WEEK ON THIS TOPIC?

6

Strings

Strings may seem simple and straightforward; we’ve used them since the start

But they are more interesting than they might appear:

- They act as **collections** of characters
- They are the topic of many **interview** questions, esp. for interns
- They are easier to **manipulate** in Python than in most other languages, making Python the go-to language for string tasks
  - Example: we have a file containing data in one format but want to create a new file with rearranged/changed data
- They give us a glimpse into the world of **objects**, a world where we call **methods**, asking the string itself to accomplish tasks. Here’s one **function** call and one **method** call:

```
myName = "Bill"
print(len(myName))              # ask len function to act on myName
print(myName.lower())           # ask myName to act on itself
```

7

String Basics & Checking Membership

8

String Basics #1

Iterating

- It’s easy to loop through each character in a string. Example:

-
  ```
  for char in userName:       # prints 1 character/line
       print(char)
  ```
- Changing the loop variable char has no effect on the original string:

-
  ```
  myName = 'Bill'
  for oneChar in myName:
       oneChar = 'x'
  print(myName)               # still 'Bill'
  ```

9

String Basics #2

Indexing

- We can retrieve individual characters by *index*
  - The first character is position 0, not 1; the last character is one *less than* len
  - Indexing using negatives counts from the *end* of the string, where -1 is the last character

```
userName = 'Chris'
firstCh = userName[1]                         # 'h'
lastCh = userName[len(userName) - 1]          # 's'
lastCh = userName[-1]                         # 's'
```

- Notes:
  - You’ll get an IndexError exception if you index past the end of the string
  - You can’t replace using indexing. This throws an error: `userName[2] = 'x'`

10

String Basics #3

Concatenation

- Create new strings by concatenating others with +
- Example: `wholeName = firstName + ' ' + lastName`
- Remember that concatenation doesn't automatically add spaces like `print` does

Immutability

- Strings are immutable, meaning you can't change a string in place; you can generate a *new* string and reassign it
- Example: `name = 'Mr. ' + name`
- This is important as we learn string methods; they *never* change the original string

11

String Basics: Slicing

It's not unusual to want to grab pieces (more than single characters) out of existing strings. Python provides a cool method to do this: string slicing

Syntax is *string*[*startIndex* : *endIndex : step*] (where start, end, step are optional)

Examples with userName = 'Chris Jones'

Expression	**Result**	**Comment**
`userName[:]`	`'Chris Jones'`	Whole string (no start or end specified)
`userName[0:5]`	`'Chris'`	Substring; up to *but not including* index 5
`userName[6:]`	`'Jones'`	Rest of string starting at index 6
`userName[-5:]`	`'Jones'`	Rest of string starting 5 *back* from end
`userName[:5]`	`'Chris'`	Beginning of string through *just before* index 5
`userName[0::2]`	`'CrsJns'`	Every other character (step value of 2)

Slicing is forgiving:

- if startIndex < 0, Python uses 0
- if endIndex > beyond end, Python uses the string's length
- if startIndex > endIndex, Python returns an empty string

12

Testing for Membership: `in` & `not in`

It’s easy to find out whether a specified string is in another string: use `in` or `not in`

Examples:

```
if 'E' in userName.upper():
    print('Name contains an E!')

if 'E' not in userName.upper():
    print('Name contains no E!')

print('el' in 'Hello')     # True
```

13

Pause & Practice

Write a function called containsVowels that accepts one parameter, a string, and returns a Boolean indicating whether the string contains *any* vowels (which, for this purpose, are only A, E, I, O, and U)

Example: if passed the string “syzygy” the function would return False; if passed “giraffe”, True

Can you think of different ways to write this? If so, which do you like better and why?

14

Pause & Practice: Code (2 versions)

```
def containsVowels1(word):
    word = word.upper()
    if      "A" in word or "E" in word or \
            "I" in word or "O" in word or \
            "U" in word:
        return True
    else:
        return False

def containsVowels2(word):
    word = word.upper()
    for vowel in "AEIOU":
        if vowel in word:
            return True
    return False
```

15

Testing, Modification, Search & Replace

16

String Methods: Testing Values

These are string **methods**; they take the form *string.method*()

.is___ methods test a string and return a Boolean result. Example:

```
hasSpace = False             # assume no space
for char in userName:
    if char.isspace():
        hasSpace = True      # we found one!
if hasSpace:
    print("Username has at least one space in it")
```

Methods in this category include…

- `isalnum` (alphanumeric), `isalpha` (alphabetic), `isdigit` (numeric)
- `islower`, `isupper`
- `isspace` (finds white space, not just spaces)
- *Note: these return false if the length of the string is zero*

Note: these also work on strings, not just single characters

17

Modification Methods

Methods in this category include Stripping functions that remove “white space” from the start and/or end of strings:

lstrip, lstrip(char), rstrip, rstrip(char), strip, strip(char)

These methods return an altered string. Example:

```
userName = '  John Doe   \t \n'
userName = userName.strip()
print('-', userName, '-', sep='')      #  -John Doe-
```

They can also remove specified characters, e.g.,

```
print('!Hello!'.strip('!'))            #  Hello
```

Remember *these only work at the start and/or end*; middle characters require other approaches

18

Modification Methods, cont.

Other methods in this category help with casing:
lower(), upper(), capitalize(), swapcase()

Examples:

```
groceryItem = 'Green Beans'
print(groceryItem.lower())              # green beans
print(groceryItem.upper())              # GREEN BEANS
print(groceryItem.capitalize())         # Green beans
print(groceryItem.swapcase())           # gREEN bEANS
```

Note: all string methods also work on literals, e.g.,

```
print('Mashed Taters'.swapcase())   # mASHED tATERS
```

19

Pause & Practice

Use what you've learned to write code that will turn this string:

```
'    \t+hello+\n   '
```

…into this string:

```
'HELLO'
```

If you get that easily enough, can you do it in one Python statement? Remember that each method returns a new string; you can continue working with the returned string by immediately calling another method

20

Pause & Practice

Write code that will turn the first string into the second:

```
'    \t+hello+\n   '      →     'HELLO'
```

Option #1

```
origString = '     \t+hello+\n   '
newString = origString.strip()        # '+hello+'
newString = newString.strip('+’)      # 'hello'
newString = newString.upper()         # 'HELLO'
print(newString)
```

Option #2

```
origString = '     \t+hello+\n   '
print(origString.strip().strip('+').upper())
```

21

Search and Replace Methods

These methods find characters or strings within other strings, or do replacements, for example:

```
  userName = 'John?Doe?'
  position = userName.find("Doe")          # returns 5 (5th position)
  if position > -1:                        # -1 indicates not found
    print("Username contains Doe")

  userName = username.replace("?"," ")     # change all ?s to spaces
  print(userName)                          # displays 'John Doe '
  print(userName.replace("?", "")          # displays 'JohnDoe'
```

Methods in this category include:
- endswith(*substring*), startswith(*substring*)
- find(*substring*)
- replace(*oldSubstring*, *newSubstring*)

22

Pause & Practice

Write a function called `cleanFillerWords` that cleans up filler words “um,” “uh,” and “like” from voicemail transcriptions. Accept one parameter, the original string. Clean it up and return the new version. Assume the words will never occur at the start or end of the phrase. Don’t remove them from real words like “dislike” or “plum,”, etc. Example:

Original: “I uh went to like um Parker’s house but like got kicked out because Parker’s uh parents didn’t like um want too many kids around”

Cleaned up: “I went to Parker’s house but got kicked out because Parker’s parents didn’t want too many kids around”

23

Pause & Practice: Code

```
def cleanFillerWords(phrase):
    cleaned = phrase.replace(" like ", " ")
    cleaned = cleaned.replace(" um ", " ")
    cleaned = cleaned.replace(" uh ", " ")
    return cleaned
…or…
def cleanFillerWords(phrase):
    return phrase.replace(' like ', ' ') \
                 .replace(' um ',   ' ') \
                 .replace(' uh ',   ' ')
```

24

String Splitting

25

String Splitting

Python is often used to process “record” data. Sometimes a whole line of data comes in at once, with a specific delimiter between each “field” (different attributes), e.g.,

$$\underbrace{\texttt{Dana Wong}}_{\text{name}}\texttt{\%}\underbrace{\texttt{1234}}_{\text{acct\#}}\texttt{\%}\underbrace{\texttt{438.57}}_{\text{balance}}\texttt{\%}\underbrace{\texttt{206-555-3901}}_{\text{phone}}$$

In this example, the delimiter is '%'

Python makes it *super* easy to break up a string at a given delimiter. It returns a **list of strings** with each list element containing one split-out portion of data:

```
fieldList = inputLine.split('%')          # separator defaults to ' '
print(fieldList[1])                       # displays 1234
```

26

Pause & Practice

Together, let's write a function called displayStudentRecord that accepts a delimited student record string and displays each "field" of data on a separate line. The format of the student record is Student ID, Student Name, Student GPA, delimited by a tilde (~)

If passed this string…

```
12345~Pat Patterson~3
```

…the function's output would be:

```
Id:     12345
Name:   Pat Patterson
GPA:    3.0
```

27

Pause & Practice: Code

```
def displayStudentRecord(studentString):
    studentData = studentString.split('~')
    print('ID   : ' + studentData[0])
    print('Name : ' + studentData[1])
    gpa = float(studentData[2])
    print('GPA  : ' + format(gpa, '.1f'))

displayStudentRecord('12345~Pat Patterson~3')
```

28

Pause & Practice

Continuing with displayStudentRecord, what if the data had undesirable leading and/or trailing white space in it? Update your code to handle that

If passed this string…

```
\t 12345 \n~ Pat Patterson \t\t ~ 3\n
```

…the function's output would still be:

```
Id:     12345
Name:   Pat Patterson
GPA:    3.0
```

29

Pause & Practice: Code

```
def displayStudentRecord(studentString):
    studentData = studentString.split('~')
    print('ID   : ' + studentData[0].strip())
    print('Name : ' + studentData[1].strip())
    gpa = float(studentData[2].strip())
    print('GPA  : ' + format(gpa, '.1f’))

displayStudentRecord('\t 12345 \n~ Pat Patterson \t\t ~ 3\n ')
```

30

Looping Through Lists

While we'll study lists in more depth soon, it shouldn't be much of a surprise that you can loop through lists; they are a *collection*. We've done this with strings and can do the same with lists

So, starting with this variable assignment, how can we print our grocery list, one item per line?

```
groceryList = 'bananas, apples, oranges, kombucha'
```

Bonus: if we want the items numbered, how can we do that, too?

31

Pause & Practice: Code

```
def displayGroceryList(groceryString):
    groceryItems = groceryString.split(', ')
    itemNum = 1
    for item in groceryItems:
        print(itemNum, item, sep='\t')
        itemNum += 1

displayGroceryList('bananas, apples, oranges, kombucha')
```

```
1       bananas
2       apples
3       oranges
4       kombucha
```

32

String Exercise

33

Lab Exercise #2

Write a function called wordStats that takes one parameter, a string, containing a sentence with no punctuation. It should calculate and return the number of words, the average word length, the minimum word length, and the maximum word length

For example, if given this sentence: See the boy play ball
the results would be 5, 3.4, 3, 4

If given: The antiestablishment candidate handily swept the election
the result would be 7, 7.4, 3, 17

Hints:

- Here again, pseudocode will be helpful; if you don't have a plan, coding will be much harder
- I recommend tackling the number of words and average word length first, then min/max
- Lists are *collections*; you can loop through them with a for loop just like we did with letters in strings. And you can find out how many items are in the collection using `len()`, just like we do with strings

34

Lab Exercise #2: Solution

```
def wordStats(sentence):
    words = sentence.split()
    totalLen = 0
    minLen = 9999
    maxLen = -9999
    for word in words:
        totalLen += len(word)
        if len(word) < minLen:
            minLen = len(word)
        if len(word) > maxLen:
            maxLen = len(word)
    wordCount = len(words)
    avgLen = totalLen / wordCount
    return wordCount, avgLen, minLen, maxLen
```

35

The End

36