CSC 221: Introduction to Programming

Fall 2011

Text processing

- Python strings
- indexing & slicing, len function
- functions vs. methods
- string methods: capitalize, lower, upper, find, remove
- string traversal: in, indexing
- examples: palindromes, encryption, pig latin

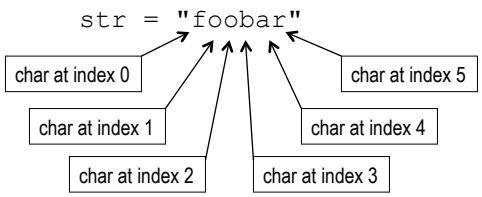
Python strings

so far, we have used Python strings to represent simple pieces of text

- displayed messages with a print statement
- passed in a name to a function
- the + operator concatenates strings, appends them end-to-end

strings are composite values, made of a sequence of individual characters

- can access individual characters using an index in brackets
- first character is at index 0; can specify negative index to count from end
- the built-in `len` function will return the length (number of chars) in a string




```
>>> str = "foobar"
>>> str[0]
'f'
>>> str[1]
'o'
>>> str[5]
'r'
>>> str[-1]
'r'
>>> len(str)
6
>>> str[len(str)-1]
'r'
```

Example: glitch

consider the following function

```
def glitch(str):
    return str[0] +"-" + str[0] + "-" + str[0] + "-" + str
```

what would be returned by `glitch("hello")` ?

note: you can multiply strings by an integer, appends that number of copies

```
def glitch(str):
    return (str[0]+"-")*3 + str
```

3

String slicing

can also slice off a substring by specifying two indices

`str[low:high]` evaluates to the substring, starting at index `low` and ending at `high-1`

if you omit either number, it assumes the appropriate end

`str[low:high:step]` can specify a step distance to skip characters

```
>>> str = "foobar"
>>> str[0:3]
'foo'
>>> str[3:6]
'bar'
>>> str[1:]
'oobar'
>>> str[::2]
'foa'
>>> str[::-1]
'raboof'
```

4

Example: rotating a string

consider the following function for rotating the characters in a string

```
def rotateLeft(str):
    if str == "":
        return str
    else:
        firstChar = str[0]
        restString = str[1:]
        return restString + firstChar
```

- why do we need to test if `str == ""`

an error occurs if you index a character out of bounds (but slicing ignores out-of-bounds indices)

```
>>> str = "foobar"
>>> str[6]

Traceback (most recent call last):
  File "<pyshell#14>", line 1, in <module>
    str[6]
IndexError: string index out of range
>>> str[1:100]
'oobar'
```

EXERCISE: define a `rotateRight` function that rotates the string in the opposite direction

5

Strings vs. primitives

although they behave similarly to primitive types (int, float, Boolean), strings are different in nature

- strings are *objects* with their own *state* (sequence of characters) and *methods* (functions that manipulate that state)
- unlike standard functions, methods are applied to a particular object and are dependent upon its internal state (e.g., the character sequence in a string object)

```
OBJECT.METHOD(PARAMETERS)
```

- e.g.,

```
>>> str = "foobar"
>>> str.capitalize()
'Foobar'
>>> str.upper()
'FOOBAR'
>>> str.lower()
'foobar'
>>> str.find("oo")
1
>>> str.find("z")
-1
>>> str.replace("oo", "u")
'fubar'
```

note: each of these methods returns a value based on the string's state – the string is NOT altered

6

Common string methods

capitalize( )	Return a copy of the string with only its first character capitalized
lower( )	Return a copy of the string converted to lowercase.
upper( )	Return a copy of the string converted to uppercase.
center(*width*)	Return centered in a string of length *width.*
rjust(*width*)	Return the string right justified in a string of length *width*.
strip( )	Return a copy of the string with the leading and trailing characters removed.
count(*sub*)	Return the number of occurrences of substring *sub* in string. *Can provide two additional inputs, low & high, to limit to [low:high] slice.*
find(*sub*)	Return the lowest index in the string where substring *sub* is found; -1 if not found. *Can similarly provide low & high inputs to limit the range.*
replace(*old, new*)	Return a copy of the string with all occurrences of substring *old* replaced by *new*. *Can similarly provide low & high inputs to limit the range.*

7

Example: censoring words

what does the following function do?

```
def censor(str):
    str = str.replace("a", "*")
    str = str.replace("e", "*")
    str = str.replace("i", "*")
    str = str.replace("o", "*")
    str = str.replace("u", "*")
    return str
```

what if we wanted to censor capital vowels as well?

```
def censor(str):
    str = str.replace("a", "*")
    str = str.replace("e", "*")
    str = str.replace("i", "*")
    str = str.replace("o", "*")
    str = str.replace("u", "*")
    str = str.replace("A", "*")
    str = str.replace("E", "*")
    str = str.replace("E", "*")
    str = str.replace("O", "*")
    str = str.replace("U", "*")
    return str
```

this is getting tedious!

8

Looping through a string

behind the scenes of a for loop

- a for loop works because the built-in `range` function returns a range of numbers
  e.g., `range(5)` → `[0,1,2,3,4]`
- the variable `i` steps through each number in that range – one loop iteration per number

```
>>> for i in range(5):
        print "Howdy"


Howdy
Howdy
Howdy
Howdy
Howdy
```

```
>>> for i in range(5):
        print i


0
1
2
3
4
```

- if desired, can use that loop value in the code
  (MORE USEFUL EXAMPLES LATER)

similarly, can use a for loop to step through the characters in a string

```
>>> for ch in "foobar":
        print ch


f
o
o
b
a
r
```

9

Example: censor revisited

using a for loop, can greatly simplify our `censor` function

```
def censor(str):
    for ch in "aeiouAEIOU":
        str = str.replace(ch, "*")
    return str
```

EXERCISE: generalize the `censor` function so that the letters to be censored are provided as inputs

- note: lowercase and uppercase occurrences of the letters should be censored

```
>>> censor("aeiou", "foobar")
'f**b*r'
>>> censor("aeiou", "Oof dah Abba goo")
'**f d*h *bb* g**'
>>> censor("abcrst", "Frak")
'F**k'
```

10

Palindrome

suppose we want to define a method to test whether a word or phrase is a palindrome (i.e., same forwards and backwards, ignoring non-letters)

```
"bob"

"madam"

"Madam, I'm Adam."

"Able was I ere I saw Elba."

"A man, a plan, a canal: Panama."
```

if we ignore the non-letters issue, it's fairly straightforward

```
def reverse(str):
    return str[::-1]

def isPalindrome(str):
    lowStr = str.lower()
    return lowStr == reverse(lowStr)
```

11

Building up a string

to strip non-letters from a string, could try to

- call remove to remove every possible non-letter character
  way too many possibilities, most of which won't appear in the string

- traverse the string, character by character
- for each non-letter encountered, call remove to remove that letter
  could work, but inefficient (remove has to search for the char all over again)

better solution: build up a copy of the string, omitting non-letter characters

```
def strCopy(str):
    copy = ""
    for ch in str:
        copy += ch
    return copy
```

this simple function copies `str`, char-by char:

```
str = "foot"
copy = ""
     = "f"
     = "fo"
     = "foo"
     = "foot"
```

12

Example: stripNonLetters

can extend `strCopy` to make the character concatenation conditional

- check each char to see if it is a letter (using `isalpha`)
- if it is, then concatenate it onto the copy; otherwise, ignore it

- can then use this in the final version of `isPalindrome`

```
def strCopy(str):
    copy = ""
    for ch in str:
        copy += ch
    return copy
```

```
def stripNonLetters(str):
    copy = ""
    for ch in str:
        if ch.isalpha():
            copy += ch
    return copy

def isPalindrome(str):
    lowStr = str.lower();
    stripStr = stripNonLetters(lowStr)
    return stripStr == reverse(stripStr)
```

13

Example: Caesar cipher

one of the earliest examples of encryption (secret codes for protecting messages) was the Caesar cipher

- used by Julius Caesar in 50-60 B.C. to encrypt military messages
- worked by shifting each letter three spots in the alphabet

  e.g., ET TU BRUTE → HW WX EUXWH

for each letter in the message:

- need to be able to find its position in the alphabet
- then find the character three spots later (wrapping around to the front for "xyz")

- there are numerous ways of doing this
  simplest: construct a string made up of all the letters in the alphabet,
  use the find method to find the index of a char in that string,
  use indexing to find the char at (index+3)

14

Example: Caesar cipher

for simplicity, we'll assume the message is made of lowercase letters only

```
ALPHABET = "abcdefghijklmnopqrstuvwxyz"

def caesar(word):
    copy = ""
    for ch in word:
        index = ALPHABET.find(ch)
        nextIndex = (index + 3) % 26
        copy += ALPHABET[nextIndex]
    return copy
```

wrap-around is handled using the remainder operator

for the letter "z", index = 25
nextIndex = (25+3)%26 = 28%26 = 2
so, "z" → "c"

15

Exercise: generalized rotation cipher

generalize the Caesar cipher so that it can be used to perform any rotation

- rotate(3, str) → encode using Caesar cipher
- rotate(-3, str) → decode using Caesar cipher

- rotate(13, str) → encode/decode using rot13 (used in many online forums)

```
ALPHABET = "abcdefghijklmnopqrstuvwxyz"

def caesar(word):
    copy = ""
    for ch in word:
        index = ALPHABET.find(ch)
        nextIndex = (index + 3) % 26
        copy += ALPHABET[nextIndex]
    return copy

def rotate(numChars, word):
    ???
```

16

Pig Latin

suppose we want to translate a word into Pig Latin

- examples

  pig → igpay
  shoe → oeshay
  apple → appleway
  latin → atinlay
  chronic → onicchray
  nth → nthway

- describe an algorithm that covers all cases

1. if there are no vowels in the word OR the word starts with a vowel

   → add "way" to the end of the word

2. otherwise, find the first vowel in the word

   → concatenate the suffix starting at the first vowel +
   the prefix up to the first vowel +
   "ay"

17

Finding a vowel (v. 1)

need to find the first occurrence of a vowel in a word

- seems like the `find` method should be useful

how about the following?

```
def findVowel(str):
    return min(str.find("a"), str.find("e"), str.find("i"), \
               str.find("o"), str.find("u"))
```

doesn't work because `find` returns -1 if the specified substring is not found

- if not all vowels appear in the word, `findVowel` will return -1

can this be (easily) fixed?

18

Finding a vowel (v. 2)

what if we traverse the word char-by-char,

- test each char to see if it is a vowel
- if so, return its index

```
def findVowel(word):
    index = 0
    for ch in word:
        if ch in "aeiouAEIOU":
            return index
        else:
            index += 1
    return -1
```

when "in" is used in a Boolean test of the form "X in Y", the test evaluates to True if X appears anywhere in Y

KLUDGY! requires keeping track of the index as you traverse the string

19

Finding a vowel (v. 3)

instead, could loop through the range of indices

- use bracket to test each character
- return the index when a vowel is found

```
def findVowel(word):
    for i in range(len(word)):
        if word[i] in "aeiouAEIOU":
            return i
    return -1
```

`i` ranges from `0` to `len(word)-1`

each time through the loop, `word[i]` is tested

EXERCISE: generalize so that any collection of letters can be searched for

```
def findOneOf(letters, word):
    ???
```

20

Finally, Pig Latin

```
def findOneOf(letters, word):
    for i in range(len(word)):
        if word[i] in letters:
            return i
    return -1

def pigLatin(word):
    index = findOneOf("aeiouAEIOU", word)
    if index < 1:
        return word + "way"
    else:
        return word[index:] + word[:index] + "ay"
```

21

Testing code

when you design and write code, how do you know if it works?

- run it a few times and assume it's OK?

to be convinced that code runs correctly in all cases, you must analyze the code and identify special cases that are handled

- then, define a test data set (inputs & corresponding outputs) that covers those cases
- e.g., for Pig Latin,
  words that start with single consonant: "foo"→"oofay" "banana"→"ananabay"
  words that start with multiple consonants: "thrill"→"illthray" "cheese"→"eesechay"
  words that start with vowel: "apple"→"appleway" "oops"→"oopsway"
  words with no vowels: "nth"→"nthway"

  what about capitalization? punctuation? digits?

22

Practice exercises

codingbat.com is a Web site developed by Nick Parlante at Stanford

- it contains numerous practice problems in Java & Python
- also has short tutorials and example code

- for each problem, you complete the definition of the specified function
  - you receive immediate feedback as to whether it worked in some/all cases
  - problems in the Warmup sections have solutions that can be studied

- go to codingbat.com and check out the String-1 and String-2 sections
  String-1 problems can be solved without loops (using indexing & slicing)
  String-2 problems may require loops to traverse or build up strings

  ✓ solve all of the String-1 problems
  ✓ solve first three String-2 problems

- if you create a (free) account, your solutions will be remembered between visits

23