Strings (str) in Python

Chapter 5
String (str) in python

• String can have one or more character
• (a character can be any character from the keyboard including punctuations).
• String should be enclosed in a single or double quotations as follow:
  name = ‘Steve Jobs’
  line = ‘NYU Rocks!’
  letter = ‘c’
  emptystring=‘’
  tab=‘\t’
  Comma = ‘,’
  space = ‘ ‘
ASCII and Unicode systems to represent characters

ASCII stands for American Standard Code for Information Interchange Old and limited to 256 characters (western alphabet)
Characters (any character from the keyboard)
Characters are represented using the following two systems:

- **ASCII**: 8 bit system and therefore can only represent 256 characters for Western based letters
- **Unicode**: 16 bit based systems and can represent thousands of characters supporting multilingual computing (Chinese, Greek, etc..)

Each character (letters, numbers, punctuations) is represented in ASCII or in the Unicode system in two ways:
1) The letter as it appears on the keyboard such as “A”
2) Or the decimal value representations of the specific value.
For “A”, the unique equivalent decimal ASCII value is 65

- The caps range for A-Z is 65 - 90
- The lower range for a-z is 97- 122
- For examples a space ASCII decimal value is 32.

You can find the full ASCII table at http://www.asciitable.com/
String Built-in Functions useful to use with Strings
String Built-in Functions

ord(character)

The function \texttt{ord(character)} returns the ASCII decimal value of a character:

```python
>>> ord("A")
65

>>> ord("a")
97

>>> ord("Z")
90
>>> ord("z")
122

>>> ord(" ")
32
```
**chr(ASCII decimal value):**

- To find a character from an ASCII decimal value:
  use `chr(ASCII decimal value)`:  
- `>>> chr(67)`  
- ‘C’
len(str)

• len(str) returns the size of the string

```python
>>> name = "cat"
>>> len(name)
3

>>> line = "Hi Class!"
>>> len(line)
9
```
Keeping track of Characters in the String

**Positive String Index**

• Every character in the string (any character including space, comma, etc…) is tracked internally using a **positive and a negative index**.

• Strings are indexed; that means that every

  **Positive index:** The index starts a zero referencing the first character in the string. The index increases by one as we move access the next letter to the right of the current letter.

  The last character in the string is referenced by index value (len(string)-1) which is the (size of the string -1)

<table>
<thead>
<tr>
<th></th>
<th>C</th>
<th>a</th>
<th>t</th>
</tr>
</thead>
<tbody>
<tr>
<td>Positive index</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Negative index</td>
<td>-3</td>
<td>-2</td>
<td>-1</td>
</tr>
</tbody>
</table>
We can use a negative index to access individual characters in a string:

**Negative indexing:** Starts with -1 index referencing last character in the string and then the index decreases by one as we move to the left.

<table>
<thead>
<tr>
<th></th>
<th>C</th>
<th>a</th>
<th>t</th>
</tr>
</thead>
<tbody>
<tr>
<td>Positive index</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Negative index</td>
<td>-3</td>
<td>-2</td>
<td>-1</td>
</tr>
</tbody>
</table>
• Examples of string indexing:
• sent = “NYU ROCKS!”
• >>> sent[0]
  • ‘N’
• >>> sent[-1]
  • ‘!’
• >>> sent[7]
  • ‘K’
• >>> sent[-8]
  • ‘U’
Using a “for” loop to read a string:

“for” loops especially to work with string:

```python
word = input('Type a word: ')
for letter in word:
    print(letter)
```
Let’s use the same example with functions

Def main():
    word = input('Type a word: ')
    for letter in word:
        print(letter)

main()
Slicing Strings (Substrings)

- Substrings are one or more consecutive characters (subset) within a string.
- The first location is always zero and the last position is one less than the length.
- Slicing follows this pattern: `word[begin:end]`

```python
>>> word = 'cat'
>>> w[0:2]
'ca'
>>> w[0:3]
'cat'
```
More string functions (methods)

- `str.capitalize()`
  - Return a copy of the string with only its first character capitalized
  ```python
  >>> word = 'hi'
  >>> word.capitalize()
  'Hi'.
  ```

- `str.lower()`
  - Return a copy of the string converted to lowercase.
  ```python
  >>> word = 'hi'
  >>> word.lower()
  'hi'
  ```
More string functions (str.find)

- **str.find(sub[, start[, end]])¶**
  - Return the lowest index in the string where substring sub is found, such that sub is contained in the range [start, end]. Optional arguments start and end are interpreted as in slice notation.
  - Return -1 if sub is not found.

```python
>>> word = 'hi'
>>> word.find('i')
1
>>> word.find('k')
-1
>>> word.find('h', 1, 2)
-1
```
More string functions (str.count())

- `str.count(sub[, start[, end]])`¶
  - Return the number of occurrences of substring sub in the range [start, end]. Optional arguments start and end are interpreted as in slice notation.
  - Word = ‘i’
- `word.count('i')`
- 1
  - Word = ‘iii’
- `word.count('i')`
- 3
More string functions (str.count())

- str.islower()¶
  Return true if all cased characters in the string are lowercase and there is at least one cased character, false otherwise.

```python
>>> word = 'hi'
>>> word.islower()
True
```
```python
>>> word = 'HI'
>>> word.islower()
False
```
Objectives

After completing this chapter, you will be able to

• Access individual characters in a string
• Retrieve a substring from a string
• Search for a substring in a string
• Convert a string representation of a number from one base to another base
Objectives (continued)

- Use string methods to manipulate strings
- Open a text file for output and write strings or numbers to the file
- Open a text file for input and read strings or numbers from the file
- Use library functions to access and navigate a file system
Accessing Characters and Substrings in Strings

• In this section, we examine the internal structure of a string more closely
• You will learn how to extract portions of a string called substrings
The Structure of Strings

• An integer can’t be factored into more primitive parts
• A string is a **immutable data structure**
  – Data structure: Consists of smaller pieces of data
  – String’s length: Number of characters it contains (0+)

```python
>>> len("Hi there!")
9
>>> len(" ")
0
```

![Figure 4.1](image) Characters and their positions in a string
The Subscript Operator

• The form of the **subscript operator** is:

\[ \text{<a string>}[\text{<an integer expression>}] \]

• Examples: index is usually in range \([0, \text{len})\); can be negative

```python
>>> name = "Alan Turing"
>>> name[0]    # Examine the first character
'A'
>>> name[3]    # Examine the fourth character
'n'
>>> name[len(name)]     # Oops! An index error!
```

Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: string index out of range

```python
>>> name[len(name) - 1]    # Examine the last character
'g'
>>> name[-1]              # Shorthand for the last one
'g'
```
The Subscript Operator (continued)

- Subscript operator is useful when you want to use the positions as well as the characters in a string
  - Use a count-controlled loop

```python
>>> data = "Hi there!"
>>> for index in xrange(len(data)):
    print index, data[index]
0 H
1 i
2
3 t
4 h
5 e
6 r
7 e
8 !
```
Slicing for Substrings

• Python’s subscript operator can be used to obtain a substring through a process called **slicing**
  – Place a colon (:) in the subscript; an integer value can appear on either side of the colon

```python
>>> name = "myfile.txt"
>>> name[0:]          # The entire string
"myfile.txt"
>>> name[0:1]         # The first character
"m"
>>> name[0:2]         # The first two characters
"my"
>>> name[:len(name)]  # The entire string
"myfile.txt"
>>> name[-3:]         # The last three characters
"txt"
```
Testing for a Substring with the \texttt{in} Operator

- When used with strings, the left operand of \texttt{in} is a target substring and the right operand is the string to be searched
  - Returns \texttt{True} if target string is somewhere in search string, or \texttt{False} otherwise

```python
>>> fileList = ["myfile.txt", "myprogram.exe", "yourfile.txt"]
>>> for fileName in fileList:
...     if ".txt" in fileName:
...         print fileName

myfile.txt
yourfile.txt
>>> 
```
String Methods

• Python includes a set of string operations called methods that make tasks like counting the words in a single sentence easy.

```python
>>> sentence = raw_input("Enter a sentence: ")
Enter a sentence: This sentence has no long words.
>>> listOfWords = sentence.split()
>>> print "There are", len(listOfWords), "words."
There are 6 words.
>>> sum = 0
>>> for word in listOfWords:
    sum += len(word)

>>> print "The average word length is", sum / len(listOfWords)
The average word length is 4
```
String Methods (continued)

- A method behaves like a function, but has a slightly different syntax
  - A method is always called with a given data value called an **object**

\[<\text{an object}.<\text{method name}>(<\text{argument-1}, \ldots, \text{argument-n}>)\]

- Methods can expect arguments and return values
- A method knows about the internal state of the object with which it is called
- In Python, all data values are in fact objects
String Methods (continued)

<table>
<thead>
<tr>
<th>STRING METHOD</th>
<th>WHAT IT DOES</th>
</tr>
</thead>
<tbody>
<tr>
<td>s.center(width)</td>
<td>Returns a copy of s centered within the given number of columns.</td>
</tr>
<tr>
<td>s.count(sub [, start [, end]])</td>
<td>Returns the number of non-overlapping occurrences of substring sub in s. Optional arguments start and end are interpreted as in slice notation.</td>
</tr>
<tr>
<td>s.endswith(sub)</td>
<td>Returns True if s ends with sub or False otherwise.</td>
</tr>
<tr>
<td>s.find(sub [, start [, end]])</td>
<td>Returns the lowest index in s where substring sub is found. Optional arguments start and end are interpreted as in slice notation.</td>
</tr>
<tr>
<td>s.isalpha()</td>
<td>Returns True if s contains only letters or False otherwise.</td>
</tr>
<tr>
<td>s.isdigit()</td>
<td>Returns True if s contains only digits or False otherwise.</td>
</tr>
</tbody>
</table>

*TABLE 4.2* Some useful string methods, with the code letter s used to refer to any string
String Methods (continued)

<table>
<thead>
<tr>
<th>STRING METHOD</th>
<th>WHAT IT DOES</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>s.join(sequence)</code></td>
<td>Returns a string that is the concatenation of the strings in the sequence. The separator between elements is <code>s</code>.</td>
</tr>
<tr>
<td><code>s.lower()</code></td>
<td>Returns a copy of <code>s</code> converted to lowercase.</td>
</tr>
<tr>
<td><code>s.replace(old, new [, count])</code></td>
<td>Returns a copy of <code>s</code> with all occurrences of substring <code>old</code> replaced by <code>new</code>. If the optional argument <code>count</code> is given, only the first <code>count</code> occurrences are replaced.</td>
</tr>
<tr>
<td><code>s.split([sep])</code></td>
<td>Returns a list of the words in <code>s</code>, using <code>sep</code> as the delimiter string. If <code>sep</code> is not specified, any whitespace string is a separator.</td>
</tr>
<tr>
<td><code>s.startswith(sub)</code></td>
<td>Returns <code>True</code> if <code>s</code> starts with <code>sub</code> or <code>False</code> otherwise.</td>
</tr>
<tr>
<td><code>s.strip([aString])</code></td>
<td>Returns a copy of <code>s</code> with leading and trailing whitespace (tabs, spaces, newlines) removed. If <code>aString</code> is given, remove characters in <code>aString</code> instead.</td>
</tr>
<tr>
<td><code>s.upper()</code></td>
<td>Returns a copy of <code>s</code> converted to uppercase.</td>
</tr>
</tbody>
</table>

**TABLE 4.2** Some useful string methods, with the code letter `s` used to refer to any string.
String Methods (continued)

```python
>>> s = "Hi there!"
>>> len(s)
9
>>> s.center(11)
'  Hi there!  '
>>> s.count('e')
2
>>> s.endswith("there!")
True
>>> s.startswith("Hi")
True
>>> s.find('the')
3
>>> s.isalpha()
False
>>> 'abc'.isalpha()
True
>>> "326".isdigit()
True
>>> words = s.split()
>>> words
['Hi', 'there!']
>>> " ".join(words)
'Hi there!'
>>> " ".join(words)
'Hi there!'
>>> s.lower()
'hi there!'  
>>> s.upper()
'HI THERE!'
>>> s.replace('i', 'o')
'Ho there!'
>>> " Hi there! ".strip()
'Hi there!'
```
String Methods (continued)

• Example: extracting a filename’s extension

```python
>>> "myfile.txt".split(".")
['myfile', 'txt']
>>> "myfile.py".split(".")
['myfile', 'py']
>>> "myfile.html".split(".")
['myfile', 'html']
>>> 
```

• The subscript `[-1]` extracts the last element
  – Can be used to write a general expression for obtaining any filename’s extension, as follows:

```python
filename.split(".")[-1]
```