Python Programming: An Introduction to Computer Science

Chapter 4
Computing with Strings
Objectives

- To understand the string data type and how strings are represented in the computer.
- To be familiar with various operations that can be performed on strings through built-in functions and the string library.
Objectives (cont.)

- To understand the basic idea of sequences and indexing as they apply to Python strings and lists.
The String Data Type

- The most common use of personal computers is word processing.
- Text is represented in programs by the string data type.
- A string is a sequence of characters enclosed within quotation marks (""') or apostrophes ('').
The String Data Type

```python
>>> str1="Hello"
>>> str2='spam'
>>> print str1, str2
Hello spam
>>> type(str1)
<type 'str'>
>>> type(str2)
<type 'str'>
```
The String Data Type

>>> firstName = input("Please enter your name: ")
Please enter your name: John

Traceback (most recent call last):
  File "<pyshell#12>", line 1, in -toplevel-
    firstName = input("Please enter your name: ")
  File "<string>", line 0, in -toplevel-
NameError: name 'John' is not defined

- What happened?
The String Data Type

- The input statement is a delayed expression.
- When you enter a name, it’s doing the same thing as:
  \[ \text{firstName} = \text{John} \]
- The way Python evaluates expressions is to look up the value of the variable John and store it in firstName.
- Since John didn’t have a value, we get a NameError.
The String Data Type

- One way to fix this is to enter your string input with quotes around it:
  ```python
  >>> firstName = input("Please enter your name: ")
  Please enter your name: "John"
  >>> print "Hello", firstName
  Hello John
  ```

- Even though this works, this is cumbersome!
The String Data Type

- There is a better way to handle text – the `raw_input` function.
- `raw_input` is like `input`, but it doesn’t evaluate the expression that the user enters.

```python
>>> firstName = raw_input("Please enter your name: ")
Please enter your name: John
>>> print "Hello", firstName
Hello John
```
The String Data Type

- We can access the individual characters in a string through indexing.
- The positions in a string are numbered from the left, starting with 0.
- The general form is `<string>[<expr>]`, where the value of expr determines which character is selected from the string.
The String Data Type

>>> greet = "Hello Bob"

>>> greet[0]
'H'

>>> print greet[0], greet[2], greet[4]
H l o

>>> x = 8

>>> print greet[x - 2]
B
The String Data Type

- In a string of n characters, the last character is at position n-1 since we start counting with 0.
- We can index from the right side using negative indexes.

```python
>>> greet[-1]
'b'
>>> greet[-3]
'B'
```
The String Data Type

- Indexing returns a string containing a single character from a larger string.
- We can also access a contiguous sequence of characters, called a substring, through a process called slicing.
The String Data Type

- **Slicing:**
  \[
  \text{<string>[:<start>:<end>]} \\
  \]
- start and end should both be ints
- The slice contains the substring beginning at position start and runs up to but doesn’t include the position end.
The String Data Type

>>> greet[0:3]
'Hel'
>>> greet[6:9]
' Bob'
>>> greet[:6]
'Hello'
>>> greet[6:]
' Bob'
>>> greet[:]
'Hello Bob'
The String Data Type

- If either expression is missing, then the start or the end of the string are used.
- Can we put two strings together into a longer string?
- Concatenation “glues” two strings together (+)
- Repetition builds up a string by multiple concatenations of a string with itself (*).
The String Data Type

- The function `len` will return the length of a string.

  >>> "spam" + "eggs"
  'spameggs'

  >>> "Spam" + "And" + "Eggs"
  'SpamAndEggs'

  >>> 3 * "spam"
  'spamspamspam'

  >>> "spam" * 5
  'spamspamspamspamspam'

  >>> (3 * "spam") + ("eggs" * 5)
  'spamspamspameggseggseggseggseggsegs'
The String Data Type

```python
>>> len("spam")
4
>>> for ch in "Spam!":
    print ch,

S p a m!
```
## The String Data Type

<table>
<thead>
<tr>
<th>Operator</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>Concatenation</td>
</tr>
<tr>
<td>*</td>
<td>Repetition</td>
</tr>
<tr>
<td><code>&lt;string&gt;[:]</code></td>
<td>Indexing</td>
</tr>
<tr>
<td><code>&lt;string&gt;[:</code></td>
<td>Slicing</td>
</tr>
<tr>
<td><code>len(&lt;string&gt;)</code></td>
<td>Length</td>
</tr>
<tr>
<td>For &lt;var&gt; in &lt;string&gt;</td>
<td>Iteration through characters</td>
</tr>
</tbody>
</table>
Simple String Processing

- Usernames on a computer system
  - First initial, first seven characters of last name

```python
# get user’s first and last names
first = raw_input(“Please enter your first name (all lowercase): ”)
last = raw_input(“Please enter your last name (all lowercase): ”)

# concatenate first initial with 7 chars of last name
uname = first[0] + last[:7]
```
Simple String Processing

>>> 
Please enter your first name (all lowercase): john
Please enter your last name (all lowercase): doe
uname = jdoe

>>> 
Please enter your first name (all lowercase): donna
Please enter your last name (all lowercase): rostenkowski
uname = drostenk
Simple String Processing

- Another use – converting an int that stands for the month into the three letter abbreviation for that month.

- Store all the names in one big string: “JanFebMarAprMayJunJulAugSepOctNovDec”

- Use the month number as an index for slicing this string:
  monthAbbrev = months[pos:pos+3]
Simple String Processing

<table>
<thead>
<tr>
<th>Month</th>
<th>Number</th>
<th>Position</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jan</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Feb</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>Mar</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>Apr</td>
<td>4</td>
<td>9</td>
</tr>
</tbody>
</table>

- To get the correct position, subtract one from the month number and multiply by three
Simple String Processing

# month.py
# A program to print the abbreviation of a month, given its number

def main():

    # months is used as a lookup table
    months = "JanFebMarAprMayJunJulAugSepOctNovDec"

    n = input("Enter a month number (1-12): ")

    # compute starting position of month n in months
    pos = (n-1) * 3

    # Grab the appropriate slice from months
    monthAbbrev = months[pos:pos+3]

    # print the result
    print "The month abbreviation is", monthAbbrev + "."
Simple String Processing

```python
>>> main()
Enter a month number (1-12): 1
The month abbreviation is Jan.
>>> main()
Enter a month number (1-12): 12
The month abbreviation is Dec.
```

- One weakness – this method only works where the potential outputs all have the same length.
- How could you handle spelling out the months?