Chapter 10:
Creating and Modifying Text
Lists
Modules
Text

- Text is manipulated as strings
- A string is a sequence of characters, stored in memory as an array

<table>
<thead>
<tr>
<th></th>
<th>H</th>
<th>e</th>
<th>l</th>
<th>l</th>
<th>o</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>
Strings

- Strings are defined with quote marks.
- Python supports three kinds of quotes:
  - Single quotes: ` ' `  
  - Double quotes: ` " `  
  - Triple quotes: `""" """`  
- You must start and end a string with the same type of quote
- Example
  >>> print 'this is a string'
  this is a string
  >>> print "this is a string"
  this is a string
  >>> print """this is a string""
  this is a string
Nesting quotes

- Quotes can be nested
  - It makes it easy to put quotes inside strings
  - Use the right one that allows you to embed quote marks you want

Example:
```python
>>> aSingleQuote = "   '    
>>> print aSingleQuote
   
>>> print 'Muhammad Ali once said “Don't count the days, make the days count”'
```
What is a triple quote, anyway?

- ```
  ```

- A triple quote allows us to embed new lines, spaces, tabs, etc. in our strings.
- In general, it is more useful when used in the program area
- Example: `tripleQ()`
For loops and Strings

- Since a string is an array of characters, we can use the for loop to get access to each individual character

- Example:

  ```python
  for eachCharacter in myString :
    print eachCharacter
  ```

  - In this case `eachCharacter` serves as the index for each element (character) in the array

- Example: `theCharacters()`
Encodings for strings

- Strings are just arrays of characters
- In most cases, characters are represented with a single byte.
  - The ASCII encoding standard maps between single byte values and the corresponding characters
  - Example: `theCode()`
- More recently, characters are represented with two bytes.
  - Unicode uses two bytes per character so that there are encodings for glyphs (characters) of other languages
  - Java uses Unicode. The version of Python we are using is based in Java, so our strings are actually using Unicode.
There are more characters than we can type

- Our keyboards don’t have all the characters available to us, and it’s hard to type others into strings. For example:
  - Backspace
  - Return
  - \u
- We use escape sequences or backslash escapes to get other characters in to strings
  - We put together a backslash followed by a character to represent these other symbols
Backslash escapes

- \b is backspace
- \n is a newline (to represent pressing the Enter key)
- \t is a tab
- \uXXXX is a Unicode character, where XXXX is a code:
  - Each X can be a digit (0 - 9) or a letter between A – F.
  - [http://www.unicode.org/charts/](http://www.unicode.org/charts/)
  - Must precede the string with “u” for Unicode to work
- Example: escape()
Remember the \textit{r}

- Remember the \textit{r} we were using in front of the path to a file?

\begin{verbatim}
pic = makePicture (r"C:\bio101\tree.jpg")
\end{verbatim}

- The \textit{r} in this case tells Python not to use a escape sequence whenever it finds a backslash followed by a character inside a string --- instead it should use the string in raw mode.
  - In this example note that we have \texttt{\backslash b} and \texttt{\backslash t} inside the string but we don’t mean backspace or tab in this case.
  - \textbf{NOTE}: this is for Windows machines only
Manipulating strings

- We can get the length of a string (how many characters are in the string) with the `len()` function.

  ```python
  >>> text = "hello there"
  >>> print len(text)
  >>> 11
  ```

- The `+` operator when used with strings performs a **concatenation** operation: appends the second string at the end of the first

  ```python
  >>> s = "cat"
  >>> t = "fish"
  >>> print s + t
  catfish
  ```
Manipulating strings

- The * operator when used with strings performs a repeat operation, where the resulting string has as many copies of the original string as indicated by the integer.
- Example:

  >>> s = "\$
  >>> t = s * 5
  >>> print t
  $$$$$
The `str()` and `int()` functions

- **The `str()` function** converts a number into a string

  **Example:**
  ```python
  num = 99
  num2 = 10.55
  strNum = str(num)  # strNum is ‘99’
  strNum2 = str(num2)  # strNum2 is “10.55”
  ```

- **The `int()` function** can be used to convert a string into an integer number

  **Example:**
  ```python
  ticket = “101”
  number = int(ticket)  # number = 101
  ```
The **float()** function

- The **float()** function can be used to convert a string into an floating-point number.

  **Example:**
  ```python
  average = "10.56"
  number = float(average)  #number = 10.56
  ```

- The **float()** function can also be used to force an integer value into a float.

  **Example:**
  ```python
  n = float(101)  #n becomes 101.0
  ```
Getting parts of strings

- We use the subscript operator \([\]\) to get parts of strings.
- \texttt{string[n]} gives you the \(n^{th}\) character in the string
- \texttt{string[n:m]} gives you the \(n^{th}\) up to (but not including) the \(m^{th}\) character.
- \texttt{string[:m]} \(n\) is assumed to be zero.
- \texttt{string[n:]} \(m\) is assumed to be the end of the string
- Negative numbers can be used at either end to trim off that much from that side.
- \textbf{Example:} \texttt{subscript()}

---

---

---

---

---

---
Very important to remember

- The index of the first character is always zero
- The index of the last character is always the length minus 1

<table>
<thead>
<tr>
<th>H</th>
<th>e</th>
<th>l</th>
<th>l</th>
<th>o</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

- Here size is 5
- First character H at index 0
- Last character o at index 4
Dot notation

- All data in Python are actually objects.
- Objects combine data and methods that act on the object.
- Methods are special functions that only objects of the same type understand.
  - Methods are functions known only to certain objects.
- To execute a method, you use dot notation:
  
  ```python
  Object.method()
  ```
- Examples:
  - color = pixel.getColor()
  - len = mySound.getLength()
Useful string methods

- **capitalize()**: returns a capitalized string → the first character of the string is made into an uppercase letter. It does not alter the original string
- **startswith(prefix)**: returns true if the string starts with the given prefix
- **endswith(suffix)**: returns true if the string ends with the given suffix
- **find(myStr)** and **find(myStr,start)** and **find(myStr,start,end)** finds `myStr` in the string and returns the index number where `myStr` starts. You can tell it what index number to start the search from, and even where to stop looking. It returns -1 if it fails.
Useful string methods

- `rfind(myStr)`, `rfind(myStr,start)` and `rfind(myStr,start,end)` similar to `find` but searches from the back of the string toward the front. Returns the index number where `myStr` starts. You can tell it what index number to start the search from, and even where to stop looking. It returns -1 if it fails.

- `upper()` returns a string where all the characters are upper case. It does not alter the original string.

- `lower()` returns a string where all the characters are lower case. It does not alter the original string.
Useful string methods

- **swapcase()** returns a copy of the original string where all uppercase letters have been swapped to lowercase and vice versa. It does not alter the original string.
- **title()** returns a string where just the first character is uppercase and the rest lowercase. It does not alter the original string.
- **isalpha()** returns true if the string is not empty and all letters. Otherwise it returns false.
- **isdigit()** returns true if the string is not empty and all numbers. Otherwise it returns false.
- **replace(search, replace)** searches for the search string and replaces it with the replace string. It returns the result but does not change the original string.
Lists

• We’ve seen lists before—that’s what `range()` returns.
• Lists contains a collection of values
  • Lists can contain strings, numbers, even other lists.
  • List can contain a mix of types
  • They work very much like strings
    • You get elements out with `[]`
    • You can add lists together (using the `+` sign)
    • You can use `for` loops on them
  • We can use them to process a variety of types of data.
• To define a list, use the `[]` and separate the elements with commas
Lists and access to elements

- We use the subscript operator `[]` to access elements in a list
  
  ```python
  >>> print (list[2])  #this will access the element at index 2
  ```

- `[m:n]` will return all the elements between m and n, not including the element at n
  
  ```python
  >>> print (list[2:5])
  ```

- `[m:]` will return all the elements from m until the end of the list
  
  ```python
  >>> print (list[2:])
  ```

- `[:n]` will return all the elements from the beginning until n, not including the element at n
  
  ```python
  >>> print (list[:5])
  ```

- Negative numbers can be used at either end to trim off that much from that side.
  
  ```python
  >>> print list[1:-1]
  ```
More on lists

- **Examples:**
  ```python
  >>> names = ["Ana", "Bob", "Claire"]
  >>> grades = [100, 98, 88, 75]
  >>> mixedList = ["Jane", "Bennet", 25, 45000.50]
  >>> phoneList = ["Ana", ["home", 1234567], ["mobile", 9876543], ["work", 2348765, ["extension", 3456]]]
  ```

- **Remember that the first element of a list is at index zero.**
  ```python
  >>> student = names[0]  # that would be "Ana"
  ```

- **The + sign allows us to append one list at the end of another**
  ```python
  >>> newList = names + grades
  >>> print newList
  >>> ["Ana, "Bob", "Claire", 100, 98, 88, 75]
  ```

- **Using the subscript operator**
  ```python
  >>> print grades[1:3]  # will print 98 88
  ```
The `in` operator

- You can use the `in` operator to determine whether an item is contained in a list.

- Format: `item in list`

- This expression returns `true` if item is found in the list, or `false` otherwise.

- Example:
  ```python
  fruits = ['banana', 'cherry', 'pear', 'apple']
  search = 'pear'
  if search in fruits :
    print (search, " was found in fruits")
  else :
    print (search, " was not found in fruits")
  ```
Methods to use with lists

- **append(element)** adds `element` at the end of the list.
- **remove(element)** removes `element` from the list, if it’s there.
- **sort()** puts the list in alphabetical/ascending order.
- **reverse()** reverses the list.
- **count(element)** tells you the number of times that `element` appears in the list. It returns an integer.
- **split(delimiter)** converts a string into a list of sub-strings, using `delimiter` to separate the strings. It returns a list of strings.
  - **Note**: This is a string method that returns a list.
Some useful list functions

- `max()` takes a list as input and returns the maximum value in the list.
- `min()` takes a list as input and returns minimum value in the list.
- `len()` takes a list as an input and returns the number of elements in that list (the size of the list) as an integer.
Adding new capabilities: Modules

- In every programming language there is a way of extending the basic capabilities of the language by adding new functionality.
- In Python we do this by importing external modules.
- A module is a Python file with a bunch of additional functions and objects defined within it.
- By importing the module, we make the module’s capabilities available to our program.
Python’s Standard Library

- Python has an extensive *library* of modules that come with it.
- The Python standard library includes modules that allow us to:
  - access the Internet,
  - deal with time,
  - generate random numbers,
  - access files in a directory,
  - and lots, lots more.
Accessing features of a module

- We access the additional capabilities of a module using dot notation, after we import the module:
  ```python
  import moduleName  #this line imports a module
  moduleName.feature  #use a feature of moduleName
  ```

- How do you know what modules and features are there?
  - Check the documentation. Python comes with a Library Guide.
    - [http://docs.python.org/py3k/library/](http://docs.python.org/py3k/library/)
  - There are books like *Python Standard Library* by Fredrik Lundh, that describe the modules and provide examples.
Module documentation

- *Python Standard Library* by Fredrik Lundh
The random module

- This module has functions for generating random numbers and making random choices.
- Here are three of the functions in this module:
  - `random.random()` generates random numbers between 0 and 1. It returns a floating point number.
  - `random.randint(low, high)` generates random integer numbers between low and high (inclusive)
  - `random.choice(stringList)` picks random words from a list. It takes a list of strings as input
Many other Python Standard Libraries

- *datetime* and *calendar* know how to manipulate dates, times, and calendars.
- *math* knows lots of important mathematical functions
- *zipfile* knows how to make and read .zip files
- *email* lets you build your own spam program, or filter spam, or build an email tool for yourself.
- *SimpleHTTPServer* is a complete working Web server.