Modifying Strings

• Sometimes you want to create a new string from an existing string:

Examples:
- For creating a string to print
- Encoding or decoding a message
- Correcting grammar or spelling
Modifying Strings

String Concatenation: \texttt{“abc” + “def”} \# \texttt{“abcdef”}

String Replication: \texttt{“abc” * 3} \# \texttt{“abcabcabc”}

String functions:

\begin{verbatim}
str.upper(“abc”) \# “ABC”
str.lower(“ABC”) \# “abc”
\end{verbatim}
Strings Are Immutable

• Strings are immutable
  – Once they are created, they cannot be changed
    • Concatenation doesn’t actually change the existing string, but rather creates a new string and assigns the new string to the previously used variable
  – Cannot use an expression of the form
    \[ \text{string}[\text{index}] = \text{new_character} \]
    • Statement of this type will raise an exception
String Slicing

• **Slice**: span of items taken from a sequence, known as *substring*
  – Slicing format: `string[start : end]`
    • Expression will return a string containing a copy of the characters from `start` up to, but not including, `end`
    • If `start` not specified, 0 is used for start index
    • If `end` not specified, `len(string)` is used for end index
  • Out-of-bounds does not generate an exception!
String Slicing Examples

name = “Alice Waters”

name[6 : 12]

name[6 : len(name)]

name[ : 5]

name[ 0 : len(name)]

name[ : ]

name[1:3]
String Slicing Examples

name = “Alice Waters”

name[6 : 12]  # “Waters”
name[6 : len(name)]
name[ : 5]
name[ 0 : len(name)]
name[ : ]
name[1:3]
String Slicing Examples

name = “Alice Waters”

name[6 : 12] # “Waters”
name[6 : len(name)] # “Waters”
name[ : 5]
name[0 : len(name)]
name[ : ]
name[1:3]
String Slicing Examples

name = “Alice Waters”

name[6 : 12]      # “Waters”

name[6 : len(name)]  # “Waters”

name[ : 5]       # “Alice”

name[ 0 : len(name)]

name[ : ]

name[1:3]
String Slicing Examples

name = “Alice Waters”

name[6 : 12]       # “Waters”
name[6 : len(name)] # “Waters”
name[ : 5]          # “Alice”
name[ 0 : len(name)] # “Alice Waters”
name[ : ]
name[1:3]
String Slicing Examples

name = “Alice Waters”

name[6 : 12]   # “Waters”
name[6 : len(name)] # “Waters”
name[ : 5]      # “Alice”
name[ 0 : len(name)] # “Alice Waters”
name[ : ]       # “Alice Waters”
name[1:3]
String Slicing Examples

name = “Alice Waters”

name[6 : 12]  # “Waters”
name[6 : len(name)]  # “Waters”
name[ : 5]  # “Alice”
name[ 0 : len(name)]  # “Alice Waters”
name[ : ]  # “Alice Waters”
name[1:3]  # “li”
String Functions/Methods

We have already used some String functions, which are functions belong to the type “str”:

```python
str.upper(“cat”)  # “CAT”
str.lower(“DOG”)  # “dog”
str.alpha(“ABC”)  # True
```

There are dozens!!
String Methods

myname = “Mary”
str.upper(name)       # “MARY”

Can also be written as a *method* of the variable myname:

myname.upper()       # “MARY”
Object-Oriented Programming

• An *object* is data together with methods (like functions) associated with it.

  – Example: A string is an object that has methods `upper`, `lower`, `isalpha`, `islower`, `isdigit`, …
String “Testing” methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>isalnum()</td>
<td>Returns true if the string contains only alphabetic letters or digits and is at least one character in length. Returns false otherwise.</td>
</tr>
<tr>
<td>isalpha()</td>
<td>Returns true if the string contains only alphabetic letters, and is at least one character in length. Returns false otherwise.</td>
</tr>
<tr>
<td>isdigit()</td>
<td>Returns true if the string contains only numeric digits and is at least one character in length. Returns false otherwise.</td>
</tr>
<tr>
<td>islower()</td>
<td>Returns true if all of the alphabetic letters in the string are lowercase, and the string contains at least one alphabetic letter. Returns false otherwise.</td>
</tr>
<tr>
<td>isspace()</td>
<td>Returns true if the string contains only whitespace characters, and is at least one character in length. Returns false otherwise. (Whitespace characters are spaces, newlines (\n), and tabs (\t).</td>
</tr>
<tr>
<td>isupper()</td>
<td>Returns true if all of the alphabetic letters in the string are uppercase, and the string contains at least one alphabetic letter. Returns false otherwise.</td>
</tr>
</tbody>
</table>
# String Modification Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>lower()</code></td>
<td>Returns a copy of the string with all alphabetic letters converted to lowercase. Any character that is already lowercase, or is not an alphabetic letter, is unchanged.</td>
</tr>
<tr>
<td><code>lstrip()</code></td>
<td>Returns a copy of the string with all leading whitespace characters removed. Leading whitespace characters are spaces, newlines (\n), and tabs (\t) that appear at the beginning of the string.</td>
</tr>
<tr>
<td><code>lstrip(char)</code></td>
<td>The <code>char</code> argument is a string containing a character. Returns a copy of the string with all instances of <code>char</code> that appear at the beginning of the string removed.</td>
</tr>
<tr>
<td><code>rstrip()</code></td>
<td>Returns a copy of the string with all trailing whitespace characters removed. Trailing whitespace characters are spaces, newlines (\n), and tabs (\t) that appear at the end of the string.</td>
</tr>
<tr>
<td><code>rstrip(char)</code></td>
<td>The <code>char</code> argument is a string containing a character. The method returns a copy of the string with all instances of <code>char</code> that appear at the end of the string removed.</td>
</tr>
<tr>
<td><code>strip()</code></td>
<td>Returns a copy of the string with all leading and trailing whitespace characters removed.</td>
</tr>
<tr>
<td><code>strip(char)</code></td>
<td>Returns a copy of the string with all instances of <code>char</code> that appear at the beginning and the end of the string removed.</td>
</tr>
<tr>
<td><code>upper()</code></td>
<td>Returns a copy of the string with all alphabetic letters converted to uppercase. Any character that is already uppercase, or is not an alphabetic letter, is unchanged.</td>
</tr>
</tbody>
</table>
## String Search and Replace Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>endswith(substring)</code></td>
<td>The <code>substring</code> argument is a string. The method returns true if the string ends with <code>substring</code>.</td>
</tr>
<tr>
<td><code>find(substring)</code></td>
<td>The <code>substring</code> argument is a string. The method returns the lowest index in the string where <code>substring</code> is found. If <code>substring</code> is not found, the method returns −1.</td>
</tr>
<tr>
<td><code>replace(old, new)</code></td>
<td>The <code>old</code> and <code>new</code> arguments are both strings. The method returns a copy of the string with all instances of <code>old</code> replaced by <code>new</code>.</td>
</tr>
<tr>
<td><code>startswith(substring)</code></td>
<td>The <code>substring</code> argument is a string. The method returns true if the string starts with <code>substring</code>.</td>
</tr>
</tbody>
</table>