CSE 1310: Introduction to Computers and Programming

Strings

Dr. Gian-Luca Mariottini
Dept. of Computer Science and Engineering

University of Texas at Arlington, TX, USA
THE STRING TYPE
Sequence of Characters

• In programming, a string is a sequence of characters.
• A string is indicated between ‘ ‘ or “ “
• The exact kind of quotes must be preserved on the left and right hand side
Strings in Python

Can use single or double quotes:
• $S = \text{“spam”}$
• $s = \text{‘spam’}$

Just don’t mix them!
• $\text{myStr = ‘hi mom” ⇒ ERROR}$

Inserting an apostrophe:
• $A = \text{“knight’s”  \# mix up the quotes}$
• $B = \text{‘knight’s’  \# escape single quote}$
Triple quotes

• Triple quotes preserve both the vertical and horizontal formatting of the string
• Allow you to type tables, paragraphs, initial comments to code, ...whatever and preserve the formatting

"""this is

a test
today"""
Indexing string elements

• Because the elements of a string are a sequence, we can associate each element with an index, a location in the sequence:
  
  — positive indices count up from the left, beginning with index 0
  
  — negative indices count down from the right, starting with -1

<table>
<thead>
<tr>
<th>characters</th>
<th>H</th>
<th>e</th>
<th>l</th>
<th>l</th>
<th>o</th>
<th>W</th>
<th>o</th>
<th>r</th>
<th>l</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>index</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>...</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>-2</td>
</tr>
</tbody>
</table>
Accessing an element

• A particular element of the string is accessed by the index of the element surrounded by square brackets `[ ]`

```python
helloStr = 'Hello World'
print helloStr[1]  ⇒ Prints ‘e’
print helloStr[-1] ⇒ Prints ‘d’
print helloStr[11] ⇒ ERROR
```
Slicing: the Rules

- Slicing consists of selecting a subsequence of the overall string
- uses the syntax \([\text{start:finish}]\), where:
  - \text{start} is the index of where we start the subsequence
  - \text{finish} is the index of \textbf{one after} where we end the subsequence
- if either \text{start} or \text{finish} are not provided, it \textbf{defaults} to the \textit{beginning of the sequence} for \text{start} and the \textit{end of the sequence} for \text{finish}

helloString[6:10]
```plaintext
helloString[6:]

<table>
<thead>
<tr>
<th>characters</th>
<th>H</th>
<th>e</th>
<th>l</th>
<th>l</th>
<th>o</th>
<th>W</th>
<th>o</th>
<th>r</th>
<th>l</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>index</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
</tbody>
</table>

first

last

helloString[5]

<table>
<thead>
<tr>
<th>characters</th>
<th>H</th>
<th>e</th>
<th>l</th>
<th>l</th>
<th>l</th>
<th>o</th>
<th>W</th>
<th>o</th>
<th>r</th>
<th>l</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>index</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
</tr>
</tbody>
</table>

first

last
```
helloString[3:-2]

FIGURE 4.5 Another slice example.

See string_ex1.py
Extended Slicing

- Takes **three arguments**:
  - `[start:finish:countBy]`
- **Default arguments** are:
  - `start` is beginning, `finish` is end, `countBy` is 1
- Example:
  
  ```
  myStr = 'hello world'
  myStr[0:11:2] ⇒ 'hlowrd'
  ```

```python
helloString[::2]
```

<table>
<thead>
<tr>
<th>characters</th>
<th>H</th>
<th>e</th>
<th>l</th>
<th>l</th>
<th>o</th>
<th>W</th>
<th>o</th>
<th>r</th>
<th>l</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>index</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
</tbody>
</table>
```
Common Python “Idioms”

• Idioms are python **short commands** ("phrases") that are used for common tasks that might be less obvious to non-python folks.

• How to make a **copy of a string**:

```python
myStr = 'hi mom'
newStr = myStr[:]
```

• How to **reverse a string**:

```python
myStr = 'madam I’m adam'
reverseStr = myStr[::-1]
```
STRING OPERATIONS
Basic String Operations

\[
\begin{align*}
\text{s} &= \text{‘spam’} \\
\text{• length operator } \text{len()} \\
\text{len(s)} &\Rightarrow 4 \\
\text{• Concatenation: } \text{myString + myString} \\
\text{newStr} &= \text{‘spam’} + \text{‘-’} + \text{‘spam-’} \\
\text{print newStr} &\Rightarrow \text{spam-spam-} \\
\text{• Repetition: } \text{myString} \ast \text{Int} \\
\text{newStr} \ast 3 &\Rightarrow \text{spam-spam-spam-spam-spam-spam-}
\end{align*}
\]
String Operations: Details

• Both + and * on strings make a new string, but do not modify the arguments.

• Order of operations:
  – important for concatenation
  – irrelevant for repetition.

• The types required are specific.
  – For concatenation you need two strings; for repetition, a string and an integer.
What kind of operation (sum or concatenation) does the above represent?

It depends on the types!
- two strings, concatenation
- two integers addition

The operator + is overloaded.
- the result of a + depends on the types it is working on
String Comparison: single-char chase

• There are two systems for representing characters: ASCII and Unicode

• ASCII takes the English letters, numbers and punctuation marks and associates them with an integer number

• Single character comparisons are based on that number
<table>
<thead>
<tr>
<th>Dec</th>
<th>Hx</th>
<th>Oct</th>
<th>Char</th>
<th>Dec</th>
<th>Hx</th>
<th>Oct</th>
<th>Htmlm</th>
<th>Chr</th>
<th>Dec</th>
<th>Hx</th>
<th>Oct</th>
<th>Htmlm</th>
<th>Chr</th>
<th>Dec</th>
<th>Hx</th>
<th>Oct</th>
<th>Htmlm</th>
<th>Chr</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>00</td>
<td>0</td>
<td>NUL (null)</td>
<td>32</td>
<td>20</td>
<td>040</td>
<td>×32</td>
<td>Space</td>
<td>64</td>
<td>40</td>
<td>100</td>
<td>×64</td>
<td>B</td>
<td>96</td>
<td>60</td>
<td>140</td>
<td>×96</td>
<td>`</td>
</tr>
<tr>
<td>1</td>
<td>01</td>
<td>1</td>
<td>SOH (start of heading)</td>
<td>33</td>
<td>21</td>
<td>041</td>
<td>×33</td>
<td>!</td>
<td>65</td>
<td>41</td>
<td>101</td>
<td>×65</td>
<td>A</td>
<td>97</td>
<td>61</td>
<td>141</td>
<td>×97</td>
<td>a</td>
</tr>
<tr>
<td>2</td>
<td>02</td>
<td>2</td>
<td>STX (start of text)</td>
<td>34</td>
<td>22</td>
<td>042</td>
<td>×34</td>
<td>'</td>
<td>66</td>
<td>42</td>
<td>102</td>
<td>×66</td>
<td>B</td>
<td>98</td>
<td>62</td>
<td>142</td>
<td>×98</td>
<td>b</td>
</tr>
<tr>
<td>3</td>
<td>03</td>
<td>3</td>
<td>ETX (end of text)</td>
<td>35</td>
<td>23</td>
<td>043</td>
<td>×35</td>
<td>#</td>
<td>67</td>
<td>43</td>
<td>103</td>
<td>×67</td>
<td>C</td>
<td>99</td>
<td>63</td>
<td>143</td>
<td>×99</td>
<td>c</td>
</tr>
<tr>
<td>4</td>
<td>04</td>
<td>4</td>
<td>EOT (end of transmission)</td>
<td>36</td>
<td>24</td>
<td>044</td>
<td>×36</td>
<td>$</td>
<td>68</td>
<td>44</td>
<td>104</td>
<td>×68</td>
<td>D</td>
<td>100</td>
<td>64</td>
<td>144</td>
<td>×100</td>
<td>d</td>
</tr>
<tr>
<td>5</td>
<td>05</td>
<td>5</td>
<td>ENQ (enquiry)</td>
<td>37</td>
<td>25</td>
<td>045</td>
<td>×37</td>
<td>%</td>
<td>69</td>
<td>45</td>
<td>105</td>
<td>×69</td>
<td>E</td>
<td>101</td>
<td>65</td>
<td>145</td>
<td>×101</td>
<td>e</td>
</tr>
<tr>
<td>6</td>
<td>06</td>
<td>6</td>
<td>ACK (acknowledge)</td>
<td>38</td>
<td>26</td>
<td>046</td>
<td>×38</td>
<td>&amp;</td>
<td>70</td>
<td>46</td>
<td>106</td>
<td>×70</td>
<td>F</td>
<td>102</td>
<td>66</td>
<td>146</td>
<td>×102</td>
<td>f</td>
</tr>
<tr>
<td>7</td>
<td>07</td>
<td>7</td>
<td>BEL (bell)</td>
<td>39</td>
<td>27</td>
<td>047</td>
<td>×39</td>
<td>'</td>
<td>71</td>
<td>47</td>
<td>107</td>
<td>×71</td>
<td>G</td>
<td>103</td>
<td>67</td>
<td>147</td>
<td>×103</td>
<td>g</td>
</tr>
<tr>
<td>8</td>
<td>08</td>
<td>8</td>
<td>BS (backspace)</td>
<td>40</td>
<td>28</td>
<td>050</td>
<td>×40</td>
<td>(</td>
<td>72</td>
<td>48</td>
<td>110</td>
<td>×72</td>
<td>H</td>
<td>104</td>
<td>68</td>
<td>150</td>
<td>×104</td>
<td>h</td>
</tr>
<tr>
<td>9</td>
<td>09</td>
<td>9</td>
<td>HT (horizontal tab)</td>
<td>41</td>
<td>29</td>
<td>051</td>
<td>×41</td>
<td>)</td>
<td>73</td>
<td>49</td>
<td>111</td>
<td>×73</td>
<td>I</td>
<td>105</td>
<td>69</td>
<td>151</td>
<td>×105</td>
<td>i</td>
</tr>
<tr>
<td>10</td>
<td>0A</td>
<td>A</td>
<td>LF (NL line feed, new line)</td>
<td>42</td>
<td>2A</td>
<td>052</td>
<td>×42</td>
<td>`</td>
<td>74</td>
<td>4A</td>
<td>112</td>
<td>×74</td>
<td>J</td>
<td>106</td>
<td>6A</td>
<td>152</td>
<td>×106</td>
<td>j</td>
</tr>
<tr>
<td>11</td>
<td>0B</td>
<td>B</td>
<td>VT (vertical tab)</td>
<td>43</td>
<td>2B</td>
<td>053</td>
<td>×43</td>
<td>+</td>
<td>75</td>
<td>4B</td>
<td>113</td>
<td>×75</td>
<td>K</td>
<td>107</td>
<td>6B</td>
<td>153</td>
<td>×107</td>
<td>k</td>
</tr>
<tr>
<td>12</td>
<td>0C</td>
<td>C</td>
<td>FF (NP form feed, new page)</td>
<td>44</td>
<td>2C</td>
<td>054</td>
<td>×44</td>
<td>,</td>
<td>76</td>
<td>4C</td>
<td>114</td>
<td>×76</td>
<td>L</td>
<td>108</td>
<td>6C</td>
<td>154</td>
<td>×108</td>
<td>l</td>
</tr>
<tr>
<td>13</td>
<td>0D</td>
<td>D</td>
<td>CR (carriage return)</td>
<td>45</td>
<td>2D</td>
<td>055</td>
<td>×45</td>
<td>-</td>
<td>77</td>
<td>4D</td>
<td>115</td>
<td>×77</td>
<td>M</td>
<td>109</td>
<td>6D</td>
<td>155</td>
<td>×109</td>
<td>m</td>
</tr>
<tr>
<td>14</td>
<td>0E</td>
<td>E</td>
<td>SO (shift out)</td>
<td>46</td>
<td>2E</td>
<td>056</td>
<td>×46</td>
<td></td>
<td>78</td>
<td>4E</td>
<td>116</td>
<td>×78</td>
<td>N</td>
<td>110</td>
<td>6E</td>
<td>156</td>
<td>×110</td>
<td>n</td>
</tr>
<tr>
<td>15</td>
<td>0F</td>
<td>F</td>
<td>SI (shift in)</td>
<td>47</td>
<td>2F</td>
<td>057</td>
<td>×47</td>
<td>/</td>
<td>79</td>
<td>4F</td>
<td>117</td>
<td>×79</td>
<td>O</td>
<td>111</td>
<td>6F</td>
<td>157</td>
<td>×111</td>
<td>o</td>
</tr>
<tr>
<td>16</td>
<td>10</td>
<td>10</td>
<td>DLE (data link escape)</td>
<td>48</td>
<td>30</td>
<td>060</td>
<td>×48</td>
<td>0</td>
<td>80</td>
<td>50</td>
<td>120</td>
<td>×80</td>
<td>P</td>
<td>112</td>
<td>70</td>
<td>160</td>
<td>×112</td>
<td>p</td>
</tr>
<tr>
<td>17</td>
<td>11</td>
<td>11</td>
<td>DC1 (device control 1)</td>
<td>49</td>
<td>31</td>
<td>061</td>
<td>×49</td>
<td>1</td>
<td>81</td>
<td>51</td>
<td>121</td>
<td>×81</td>
<td>Q</td>
<td>113</td>
<td>71</td>
<td>161</td>
<td>×113</td>
<td>q</td>
</tr>
<tr>
<td>18</td>
<td>12</td>
<td>12</td>
<td>DC2 (device control 2)</td>
<td>50</td>
<td>32</td>
<td>062</td>
<td>×50</td>
<td>2</td>
<td>82</td>
<td>52</td>
<td>122</td>
<td>×82</td>
<td>R</td>
<td>114</td>
<td>72</td>
<td>162</td>
<td>×114</td>
<td>r</td>
</tr>
<tr>
<td>19</td>
<td>13</td>
<td>13</td>
<td>DC3 (device control 3)</td>
<td>51</td>
<td>33</td>
<td>063</td>
<td>×51</td>
<td>3</td>
<td>83</td>
<td>53</td>
<td>123</td>
<td>×83</td>
<td>S</td>
<td>115</td>
<td>73</td>
<td>163</td>
<td>×115</td>
<td>s</td>
</tr>
<tr>
<td>20</td>
<td>14</td>
<td>14</td>
<td>DC4 (device control 4)</td>
<td>52</td>
<td>34</td>
<td>064</td>
<td>×52</td>
<td>4</td>
<td>84</td>
<td>54</td>
<td>124</td>
<td>×84</td>
<td>T</td>
<td>116</td>
<td>74</td>
<td>164</td>
<td>×116</td>
<td>t</td>
</tr>
<tr>
<td>21</td>
<td>15</td>
<td>15</td>
<td>NAK (negative acknowledge)</td>
<td>53</td>
<td>35</td>
<td>065</td>
<td>×53</td>
<td>5</td>
<td>85</td>
<td>55</td>
<td>125</td>
<td>×85</td>
<td>U</td>
<td>117</td>
<td>75</td>
<td>165</td>
<td>×117</td>
<td>u</td>
</tr>
<tr>
<td>22</td>
<td>16</td>
<td>16</td>
<td>SYN (synchronous idle)</td>
<td>54</td>
<td>36</td>
<td>066</td>
<td>×54</td>
<td>6</td>
<td>86</td>
<td>56</td>
<td>126</td>
<td>×86</td>
<td>V</td>
<td>118</td>
<td>76</td>
<td>166</td>
<td>×118</td>
<td>v</td>
</tr>
<tr>
<td>23</td>
<td>17</td>
<td>17</td>
<td>ETB (end of trans. block)</td>
<td>55</td>
<td>37</td>
<td>067</td>
<td>×55</td>
<td>7</td>
<td>87</td>
<td>57</td>
<td>127</td>
<td>×87</td>
<td>W</td>
<td>119</td>
<td>77</td>
<td>167</td>
<td>×119</td>
<td>w</td>
</tr>
<tr>
<td>24</td>
<td>18</td>
<td>18</td>
<td>CAN (cancel)</td>
<td>56</td>
<td>38</td>
<td>070</td>
<td>×56</td>
<td>8</td>
<td>88</td>
<td>58</td>
<td>130</td>
<td>×88</td>
<td>X</td>
<td>120</td>
<td>78</td>
<td>170</td>
<td>×120</td>
<td>x</td>
</tr>
<tr>
<td>25</td>
<td>19</td>
<td>19</td>
<td>EM (end of medium)</td>
<td>57</td>
<td>39</td>
<td>071</td>
<td>×57</td>
<td>9</td>
<td>89</td>
<td>59</td>
<td>131</td>
<td>×89</td>
<td>Y</td>
<td>121</td>
<td>79</td>
<td>171</td>
<td>×121</td>
<td>y</td>
</tr>
<tr>
<td>26</td>
<td>1A</td>
<td>20</td>
<td>SUB (substitute)</td>
<td>58</td>
<td>3A</td>
<td>072</td>
<td>×58</td>
<td>=</td>
<td>90</td>
<td>5A</td>
<td>132</td>
<td>×90</td>
<td>Z</td>
<td>122</td>
<td>7A</td>
<td>172</td>
<td>×122</td>
<td>z</td>
</tr>
<tr>
<td>27</td>
<td>1B</td>
<td>21</td>
<td>ESC (escape)</td>
<td>59</td>
<td>3B</td>
<td>073</td>
<td>×59</td>
<td>;</td>
<td>91</td>
<td>5B</td>
<td>133</td>
<td>×91</td>
<td>[</td>
<td>123</td>
<td>7B</td>
<td>173</td>
<td>×123</td>
<td>{</td>
</tr>
<tr>
<td>28</td>
<td>1C</td>
<td>22</td>
<td>FS (file separator)</td>
<td>60</td>
<td>3C</td>
<td>074</td>
<td>×60</td>
<td>&lt;</td>
<td>92</td>
<td>5C</td>
<td>134</td>
<td>×92</td>
<td>\</td>
<td>124</td>
<td>7C</td>
<td>174</td>
<td>×124</td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>1D</td>
<td>23</td>
<td>GS (group separator)</td>
<td>61</td>
<td>3D</td>
<td>075</td>
<td>×61</td>
<td>&gt;</td>
<td>93</td>
<td>5D</td>
<td>135</td>
<td>×93</td>
<td>]</td>
<td>125</td>
<td>7D</td>
<td>175</td>
<td>×125</td>
<td>}</td>
</tr>
<tr>
<td>30</td>
<td>1E</td>
<td>24</td>
<td>RS (record separator)</td>
<td>62</td>
<td>3E</td>
<td>076</td>
<td>×62</td>
<td>^</td>
<td>94</td>
<td>5E</td>
<td>136</td>
<td>×94</td>
<td>^</td>
<td>126</td>
<td>7E</td>
<td>176</td>
<td>×126</td>
<td>^</td>
</tr>
<tr>
<td>31</td>
<td>1F</td>
<td>25</td>
<td>US (unit separator)</td>
<td>63</td>
<td>3F</td>
<td>077</td>
<td>×63</td>
<td>?</td>
<td>95</td>
<td>5F</td>
<td>137</td>
<td>×95</td>
<td>_</td>
<td>127</td>
<td>7F</td>
<td>177</td>
<td>×127</td>
<td>DEL</td>
</tr>
</tbody>
</table>

Source: www.asciiatable.com
Comparison within sequence

• It makes sense to compare within a sequence (lower case with lower case, etc.).
  – ‘a’ < ‘b’ => True (because ord(‘a’) < ord(‘b’))
  – ‘A’ < ‘B’ => True
  – ‘1’ < ‘9’ => True

• Can get wrong results outside of the sequence:
  – ‘a’ < ‘A’ => False
  – ‘a’ < ‘0’ => False

• The inverse of ord is char
Whole-String Comparison

• Compare the **first element of each string**:
  – if they are equal, move on to the next character in each
  – if they are not equal, the relationship between those to characters are the relationship between the string
  – if one ends up being shorter (but equal), the shorter is smaller
Example

- ‘a’ < ‘b’ => True
- ‘aaab’ < ‘aaac’
  - First difference is at the last char. ‘b’<‘c’ so ‘aaab’ is less than ‘aaac’. => True.
- ‘aa’ < ‘aaz’
  - The first string is the same but shorter. Thus it is “smaller”. => True.

See string_ex2.py
Membership: the “in” operator

- **in**: Used to see if a substring exists in the string. Returns True or False

```python
myStr = 'aabbccdd'
'a' in myStr ⇒ True
'abb' in myStr ⇒ True
'x' in myStr ⇒ False
'bd' in myStr ⇒ False
```
Strings are immutable

• Strings are immutable, that is you cannot change an existing one:
  – `aStr = 'spam'`
  – `aStr[1] = 'l' ⇒ ERROR`

• However, you can use it to create a “modified” copy:
  – `newStr = aStr[:1] + 'l' + aStr[2:]`
  – `aStr ⇒ 'spam'`
  – `newStr ⇒ 'slam'`
STRING METHODS AND FUNCTIONS
Functions, first cut

• A function is a program, used within another program, that performs some operation.
• Its details are hidden (encapsulated), only its interface provided.
• A function takes some number of inputs (arguments) and returns an output based on the arguments and the function’s operation.
An example: the function `len`

- The `len` function takes as an argument a string and returns the length of a string (type integer).

```python
myStr = 'Hello World'
len(myStr) => 11  # space counts
```
Methods

• A **method** is a variation on a function
  – like a function, it represents a program
  – like a function, it has input arguments and an output

• Unlike a function, it **belongs** (and it is then applied) **only to a particular object**.
  – Think of it as a “skill” for that object

• This is indicated by the ‘dot notation’ invocation
An example: the method upper

- upper is the name of a method.
- The method upper is a "skill" of strings.
- It generates a new string that has all upper case characters of the string it was called with.
  
  ```python
  myStr = 'Python Rules!' 
  myStr.upper() ⇒ 'PYTHON RULES!' 
  ```
- The string myStr called the upper() method, indicated by the dot between them.
The method `find`

```python
myStr = 'hello'
myStr.find('l')  # find index of 'l' in myStr
⇒ 2
```

- The method `find` operates on the string object `myStr` and the two are associated by using the "dot" notation: `myStr.find('l')`.

- Terminology: the thing(s) in parenthesis, i.e. the 'l' in this case, is called an **argument**.
Chaining

Methods can be chained together.

• Perform first operation, yielding an object
• Use the yielded object for the next method

```python
myStr = 'Python Rules!
myStr.upper() ⇒ 'PYTHON RULES!'
myStr.upper().find('O') ⇒ 4
```

• Chaining reads from left to right!
Optional Arguments

Some methods have optional arguments:

• if the user doesn’t provide one of these, a **default** is assumed

• **find** has a second argument indicating where the search begins (spaces count!) (0 default)

```python
aStr = 'He had the bat'
aStr.find('t') ⇒ 7 # 1st 't', start @ 0
aStr.find('t', 8) ⇒ 13 # 2nd 't'
```
Nesting methods

• You can “nest” methods, that is, the result of one method as an argument to another.

• Remember that parenthetical expressions are executed “inside out”:
  – do the inner parenthetical expression first, then the next, using the result as an argument.
  – Example:
    • `aStr.find('t', aStr.find('t')+1)`

• Translation: find the second ‘t’.

See string_ex3.py
List of available methods

• You can use **IDLE** to find available methods for any type. You enter a variable of the type, followed by the ‘.’ (dot) and then a tab.

• Remember, **methods match with a type**. Different types have different methods.

• If you type a method name, IDLE will remind you of the needed and optional arguments.
In a Python shell, you can manipulate strings using various methods. Here, the user can see methods like `capitalize`, `center`, `count`, `decode`, `encode`, `endswith`, `expandtabs`, `find`, `index`, and `isalnum`. The example shows the usage of `myString` and `myString.`, which suggests the user might be working with string attributes or methods.
**FIGURE 4.9** IDLE pop-up provides help with function arguments and return types.
More Methods

(Even more exist: http://docs.python.org/lib/string-methods.html)

- s.capitalize
- s.center(width)
- s.count(sub[,start [,end]]])
- s.ljust(width)
- s.lower()
- s.upper()
- s.lstrip()
- s.rfind(sub, [,start [,end]]])
- s.splitlines([keepends])
- s.strip()
- s.translate(table [, delchars])
STRING FORMATTING
String formatting, better printing

• So far, we have just used the defaults of the print function.
• We can do many more complicated things to make that output “prettier” and more pleasing.
• We will apply it to our “display” function.
String formatting

• To understand string formatting... a small example:

```python
>>> print "Sorry, is this the %d minute %s?" % (5, 'ARGUMENT')
Sorry, is this the 5 minute ARGUMENT?
```
Many Descriptors

• %s (string)
• %d (decimal)
• %e (floating point exponent)
• %f (floating point decimal)
• %u (unsigned integer)
• and others
Matching objects

- Objects are matched in order with their format descriptors. The substitution is made and resulting string printed

```python
print """s is %d years old" % ("Bill", 25)
```

prints Bill is 25 years old
Format Descriptors

• The format string contains a set of format descriptors that describe how an object is to be printed.

• Overall:

  %[name][flags][width][.precision]code

where [ ] are optional
print "%10s is %10d years old." % ("Bill", 25)

String 10 spaces wide including the object right justified.

Decimal 10 spaces wide including the object "-" means left justified.

OUTPUT:

Bill is 25 years old.

10 spaces 10 spaces

**FIGURE 4.11** String formatting with width descriptors.
• print `math.pi`
  – `3.14159265359`
• print "%.4f" % math.pi
  – `3.1416` (4 decimal points of precision, with rounding)
• print "%10.2f" % math.pi
  – `3.14` (10 spaces total including the number and the decimal point)
CONTROL FOR STRINGS
The for statement

We use the for statement to execute a code suite, a number of times.

The cycle picks each element belonging to a list, one element at a time:

```
for element in sequence:
    suite
```
myStr='abc'
for myVar in myStr:
    print myVar

Meaning
• first time through, myVar='a' (myStr[0])
• second time through, myVar='b' (myStr[1])
• third time through, myVar='c' (myStr[2])
• no more sequence left, we quit
Find a letter

river = 'Mississippi'
target = raw_input('Input character to find: ')
for index in range(len(river)):  # for each index
    if river[index] == target:  # check
        print "Letter found at index: ", index
        break  # stop searching
else:
    print 'Letter',target,'not found in',river
The enumerate function

• The `enumerate` function prints out two values: the index of an element and the element itself

• Can use it to iterate through both the index and element simultaneously, doing dual assignment
# print first occurrence
river = 'Mississippi'
target = raw_input('Input character to find: ')
for index, letter in enumerate(river):
    if letter == target:
        # check
        print "Letter found at index: ", index
        break  # stop searching
else:
    print 'Letter', target, 'not found in', river
# print all occurrences
river = 'Mississippi'
target = raw_input('Input character to find: ')
for index, letter in enumerate(river):
    if letter == target:
        print "Letter found at index: ", index
    # break
    # stop
else:
    print 'Letter', target, 'not found in', river
Split function

• Takes a string and breaks it into multiple new string parts depending on the argument.
• By default, if no argument is provided, split is on any whitespace character (tab, blank, etc.)
• Otherwise the character indicates the character where to split the string at
Reorder a name

origName = ‘John Marwood Cleese’
first, mid, last = origName.split()
name = last + ‘, ‘ + first + ‘ ‘ + mid
print name