CSc 110, Spring 2017

Lecture 10: Strings

Adapted from slides by Marty Stepp and Stuart Reges
# This program enhances our Receipt program using a cumulative sum.
def main():
    subtotal = meals()
    results(subtotal)

# Prompts for number of people and returns total meal subtotal.
def meals():
    people = float(input("How many people ate? "))
    subtotal = 0.0;
    # cumulative sum

    for i in range(1, people + 1):
        person_cost = float(input("Person #" + str(i) + 
            ": How much did your dinner cost? "))
        subtotal = subtotal + person_cost;
        # add to sum

    return subtotal
...
# Calculates total owed, assuming 8% tax and 15% tip

def results(subtotal):
    tax = subtotal * .08
    tip = subtotal * .15
    total = subtotal + tax + tip

    print("Subtotal: ", subtotal)
    print("Tax: ", tax)
    print("Tip: ", tip)
    print("Total: ", total)
Strings

• **string**: a sequence of characters

  name = "text"
  name = expression

• Examples:

  name = "Daffy Duck"
  x = 3
  y = 5
  point = "(" + str(x) + ", " + str(y) + ")"
Indexes

• Characters of a string are numbered with 0-based *indexes*:

\[
\text{name} = \text{"Ultimate"}
\]

<table>
<thead>
<tr>
<th>index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>-8</td>
<td>-7</td>
<td>-6</td>
<td>-5</td>
<td>-4</td>
<td>-3</td>
<td>-2</td>
<td>-1</td>
</tr>
<tr>
<td>character</td>
<td>U</td>
<td>l</td>
<td>t</td>
<td>i</td>
<td>m</td>
<td>a</td>
<td>t</td>
<td>e</td>
</tr>
</tbody>
</table>

• First character's index : 0
• Last character's index : 1 less than the string's length
Indexes

- You can access a character with `string[index]`:
  
  ```python
ame = "Merlin"
print(name[0])
```

  **Output:**  M

  `name[0]` produces a string of length 1
Subscripting

• Syntax:

```python
part = string[start : stop]
```

produces a substring of `string`, including `start`, excluding `stop`

• Example:

```python
s = "Merlin"
mid = [1:3]  # characters from position 1 (included)
            # to position 3 (excluded)
```

• `mid` is the following string:

"er"
Subscripting

• If you want to start at the beginning you can leave off the start position
  
  ```python
  s = "Merlin"
  mid = [:3]  # characters from the beginning to position 2
  ```
  
  Produces the string
  
  "Mer"

• If you want to stop at the end you can leave off the stop position
  
  ```python
  mid = [1:]
  ```
  # characters from position 1 to the end
  
  Produces the string
  
  "erlin"
Built-in String function - length

• Syntax:
  
  \[
  \text{length} = \text{len}(\text{string})
  \]

• Example:
  
  \[
  \text{s} = "\text{Merlin}"
  \]
  
  \[
  \text{count} = \text{len}(\text{s}) \quad \# \quad 6
  \]

Note: Not all "functions" defined for strings are built-in
String methods

• Some functions are associated with a specific data type, but are not part of the built-in set of functions

• Such functions are called methods

• Special syntax is used to call methods (dot notation)

  ```python
  s = "Merlin"
  s.lower()  # merlin
  ```
### String methods

<table>
<thead>
<tr>
<th>Method name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>find(str)</code></td>
<td>index where the start of the given string appears in this string (-1 if not found)</td>
</tr>
<tr>
<td><code>substring(index1, index2)</code> or <code>substring(index1)</code></td>
<td>the characters in this string from <code>index1</code> (inclusive) to <code>index2</code> (exclusive); if <code>index2</code> is omitted, grabs till end of string</td>
</tr>
<tr>
<td><code>lower()</code></td>
<td>a new string with all lowercase letters</td>
</tr>
<tr>
<td><code>upper()</code></td>
<td>a new string with all uppercase letters</td>
</tr>
</tbody>
</table>

- These methods are called using the dot notation shown below:

```python
starz = "Biles & Manuel"
print(starz.lower())  # biles & manuel
print(starz.find("Manuel"))  # 8
```
String method examples

```python
# index     012345678901
s1 = "Oliver Twist"
s2 = "Merlin The Cat"
print(s1.find("r"))       # 5
print(s2.lower())         # "merlin the cat"
```

• Given the following string:

```python
# index     012345678901234567890123
book = "Building Python Programs"
```

• How would you extract the word "Python"?
Modifying strings

• String methods like `lowercase` build and return a new string, rather than modifying the current string.

```python
s = "Aceyalone"
s.upper()
print(s)  # Aceyalone
```

• To modify a variable's value, you must reassign it:

```python
s = "Aceyalone"
s = s.upper()
print(s)  # ACEYALONE
```
Looping through a string

• You can use a `for` loop and indexes to print each character in a string:

```python
major = "CSc";
for letter in range(0, len(major)):
    print(major[letter:letter + 1])
```

• You can also use a `for` loop to print or examine each character without range:

```python
major = "CSc";
for letter in major:
    print(letter)
```

Output:

C
S
c
Name border

• Prompt the user for full name

• Draw out the pattern to the left

• This should be resizable. Size 1 is shown and size 2 would have the first name twice followed by last name twice
Write a program that reads two people's first names and suggests a name for their child.

Example Output:

Parent 1 first name? Danielle
Parent 2 first name? John
Child Gender? f
Suggested baby name: JODANI

Parent 1 first name? Danielle
Parent 2 first name? John
Child Gender? Male
Suggested baby name: DANIJO
String methods that produce True or False

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>startswith(str)</td>
<td>whether one contains other's characters at start</td>
</tr>
<tr>
<td>endswith(str)</td>
<td>whether one contains other's characters at end</td>
</tr>
</tbody>
</table>

name = "Voldemort"
if (name.startswith("Vol")):
    print("He who must not be named")

• The in operator can be used to test if a string contains another string.

example: "er" in name    # true
Strings and ints

• All characters are assigned numbers internally by the computer, called ASCII values.

  • Examples:
    
    'A' is 65,   'B' is 66,   ' ' is 32
    'a' is 97,   'b' is 98,   '*' is 42

  • We can get the ASCII value of a String of length 1 using \texttt{ord(str)}
    \[
    \text{ord('a')} \quad \text{is 97}
    \]

  • The function \texttt{chr(n)} returns the character represented by the ASCII value \(n\)
    \[
    \text{chr(66)} \quad \text{is 'B'}
    \]

  • This is useful because you can do the following:
    \[
    \text{chr(\text{ord('a' + 2))}} \quad \text{is 'c'}
    \]
String question

• A *Caesar cipher* is a simple encryption where a message is encoded by shifting each letter by a given amount.
  • e.g. with a shift of 3,  A → D,  H → K,  X → A,  and Z → C

• Write a program that reads a message from the user and performs a Caesar cipher on its letters:

  Your secret message:  Brad thinks Angelina is cute
  Your secret key:  3
  The encoded message:  eudg wklqnv dqjholqd lv fxwh
# This program reads a message and a secret key from the user and
# encrypts the message using a Caesar cipher, shifting each letter.
def main():
    message = input("Your secret message: ")
    message = message.lower()
    key = int(input("Your secret key: "))
    encode(message, key)

# This method encodes the given text string using a Caesar
# cipher, shifting each letter by the given number of places.
def encode(text, shift):
    print("The encoded message: ")
    for letter in text:
        # shift only letters (leave other characters alone)
        if (letter >= 'a' and letter <= 'z'):
            letter = chr(ord(letter) + shift)
            # may need to wrap around
        elif (letter > 'z'):
            letter = chr(ord(letter) - 26)
        elif (letter < 'a'):
            letter = chr(ord(letter) + 26)
        print(letter, end='')
    print()