Lecture 4

Strings
Announcements For This Lecture

Readings

• Chapter 8
  ▪ 8.1, 8.2, 8.4, 8.5
  ▪ Avoid for-loop sections

Next Lab

• More expression tables
• Testing functions

Assignment 1

• Will post it on Monday
  ▪ Need one more lecture
• Due Thu, Sep. 21st
  ▪ Lab 4 gives time to work
  ▪ Revise until correct
• Can work in pairs
  ▪ Submit one for both

9/8/17 Strings
Purpose of Today’s Lecture

• Return to the string (str) type
  ▪ Saw it the first day of class
  ▪ Learn all of the things we can do with it

• See more examples of functions
  ▪ Particularly functions with strings

• Learn the difference between…
  ▪ Procedures and fruitful functions
  ▪ print and return statements
String: Text as a Value

- **String are quoted characters**
  - 'abc d' (Python prefers)
  - "abc d" (most languages)

- **How to write quotes in quotes?**
  - Delineate with “other quote”
  - **Example**: " ' " or ' " '
  - What if need both " and '?

- **Solution**: escape characters
  - Format: \ + letter
  - Special or invisible chars

<table>
<thead>
<tr>
<th>Char</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>\</td>
<td>single quote</td>
</tr>
<tr>
<td>&quot;</td>
<td>double quote</td>
</tr>
<tr>
<td>\n</td>
<td>new line</td>
</tr>
<tr>
<td>\t</td>
<td>tab</td>
</tr>
<tr>
<td>\</td>
<td>backslash</td>
</tr>
</tbody>
</table>

Type: str
String are Indexed

- $s = \text{'abc d'}$
  
  \[
  \begin{array}{cccc}
  0 & 1 & 2 & 3 & 4 \\
  a & b & c & d \\
  \end{array}
  \]

- Access characters with []
  - $s[0]$ is 'a'
  - $s[4]$ is 'd'
  - $s[5]$ causes an error
  - $s[0:2]$ is 'ab' (excludes c)
  - $s[2:]$ is 'c d'

- Called “string slicing”

- $s = \text{'Hello all'}$
  
  \[
  \begin{array}{ccccccccccc}
  0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\
  H & e & l & l & o & a & l & l \\
  \end{array}
  \]

- What is $s[3:6]$?
  
  A: 'lo a'
  B: 'lo'
  C: 'lo '
  D: 'o '
  E: I do not know

9/8/17
String are Indexed

- \( s = 'abc d' \)
  
<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
<td></td>
</tr>
</tbody>
</table>

- Access characters with [ ]
  - \( s[0] \) is 'a'
  - \( s[4] \) is 'd'
  - \( s[5] \) causes an error
  - \( s[0:2] \) is 'ab' (excludes c)
  - \( s[2:] \) is 'c d'

- Called “string slicing”

- \( s = 'Hello all' \)
  
<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>H</td>
<td>e</td>
<td>l</td>
<td>l</td>
<td>o</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>a</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>l</td>
</tr>
</tbody>
</table>

- What is \( s[3:6] \)?
  - A: 'lo a'
  - B: 'lo'
  - C: 'lo'  CORRECT
  - D: 'o '
  - E: I do not know
String are Indexed

- $s = 'abc d'$
  - Access characters with []
    - $s[0]$ is 'a'
    - $s[4]$ is 'd'
    - $s[5]$ causes an error
    - $s[0:2]$ is 'ab' (excludes c)
    - $s[2:]$ is 'c d'
- Called “string slicing”

- $s = 'Hello all'$
  - What is $s[:4]$?
    - A: 'o all'
    - B: 'Hello'
    - C: 'Hell'
    - D: Error!
    - E: I do not know
String are Indexed

• \( s = '\text{abc d}' \)

\[
\begin{array}{cccc}
0 & 1 & 2 & 3 & 4 \\
\text{a} & \text{b} & \text{c} & \text{d} \\
\end{array}
\]

• Access characters with []
  - \( s[0] \) is 'a'
  - \( s[4] \) is 'd'
  - \( s[5] \) causes an error
  - \( s[0:2] \) is 'ab' (excludes c)
  - \( s[2:] \) is 'c d'

• Called “string slicing”

• \( s = '\text{Hello all}' \)

\[
\begin{array}{cccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\
\text{H} & \text{e} & \text{l} & \text{l} & \text{o} & \text{a} & \text{l} & \text{l} \\
\end{array}
\]

• What is \( s[:4] \)?
  - A: 'o all'
  - B: 'Hello'
  - C: 'Hell' CORRECT
  - D: Error!
  - E: I do not know
Other Things We Can Do With Strings

- **Operation** `in`: $s_1 \text{ in } s_2$
  - Tests if $s_1$ “a part of” $s_2$
  - Say $s_1$ a *substring* of $s_2$
  - Evaluates to a bool

- **Examples**: 
  - $s = 'abracadabra'$
  - 'a' in s == True
  - 'cad' in s == True
  - 'foo' in s == False

- **Function** `len`: `len(s)`
  - Value is # of chars in s
  - Evaluates to an int

- **Examples**: 
  - $s = 'abracadabra'$
  - len(s) == 11
  - len(s[1:5]) == 4
  - s[1:len(s)-1] == 'bracadabr'
Defining a String Function

• Start w/ string variable
  ▪ Holds string to work on
  ▪ Make it the parameter

• Body is all assignments
  ▪ Make variables as needed
  ▪ But last line is a return

• Try to work in reverse
  ▪ Start with the return
  ▪ Figure ops you need
  ▪ Make a variable if unsure
  ▪ Assign on previous line

```
def middle(text):
    """Returns: middle 3rd of text
    Param text: a string"
    # Get length of text
    # Start of middle third
    # End of middle third
    # Get the text
    # Return the result
    return result
```
Defining a String Function

• Start w/ string variable
  - Holds string to work on
  - Make it the parameter
• Body is all assignments
  - Make variables as needed
  - But last line is a return
• Try to work in reverse
  - Start with the return
  - Figure ops you need
  - Make a variable if unsure
  - Assign on previous line

```python
def middle(text):
    """Returns: middle 3rd of text
    Param text: a string""
    # Get length of text
    # Start of middle third
    # End of middle third
    # Get the text
    result = text[start:end]
    # Return the result
    return result
```
Defining a String Function

• Start w/ string variable
  ▪ Holds string to work on
  ▪ Make it the parameter

• Body is all assignments
  ▪ Make variables as needed
  ▪ But last line is a return

• Try to work in reverse
  ▪ Start with the return
  ▪ Figure ops you need
  ▪ Make a variable if unsure
  ▪ Assign on previous line

```python
def middle(text):
    """Returns: middle 3rd of text
    Param text: a string""

    # Get length of text
    # Start of middle third
    end = 2*size//3
    # End of middle third
    # Get the text
    result = text[start:end]
    # Return the result
    return result
```
Defining a String Function

• Start w/ string variable
  ▪ Holds string to work on
  ▪ Make it the parameter

• Body is all assignments
  ▪ Make variables as needed
  ▪ But last line is a return

• Try to work in **reverse**
  ▪ Start with the return
  ▪ Figure ops you need
  ▪ Make a variable if unsure
  ▪ Assign on previous line

```python
def middle(text):
    """Returns: middle 3\textsuperscript{rd} of text
    Param text: a string""

    # Get length of text
    size = len(text)

    # Start of middle third
    start = size // 3

    # End of middle third
    end = 2 * size // 3

    # Get the text
    result = text[start:end]

    # Return the result
    return result
```
Defining a String Function

- Start w/ string variable
  - Holds string to work on
  - Make it the parameter
- Body is all assignments
  - Make variables as needed
  - But last line is a return
- Try to work in reverse
  - Start with the return
  - Figure ops you need
  - Make a variable if unsure
  - Assign on previous line

```python
def middle(text):
    """Returns: middle 3rd of text
    Param text: a string""
    # Get length of text
    size = len(text)
    # Start of middle third
    start = size//3
    # End of middle third
    end = 2*size//3
    # Get the text
    result = text[start:end]
    # Return the result
    return result
```
```
def middle(text):
    """Returns: middle 3\textsuperscript{rd} of text
    Param text: a string"""
    # Get length of text
    size = len(text)
    # Start of middle third
    start = size//3
    # End of middle third
    end = 2*size//3
    # Get the text
    result = text[start:end]
    # Return the result
    return result

>>> middle('abc')
'b'

>>> middle('aabbcc')
'bb'

>>> middle('aaabbbccc')
'bbb'
```
def greet(n):
    """Prints a greeting to the name n

    Parameter n: name to greet
    Precondition: n is a string""

    print('Hello '+n+'!')
    print('How are you?')

    No assignments or return
    The call frame is EMPTY
## Procedures vs. Fruitful Functions

<table>
<thead>
<tr>
<th>Procedures</th>
<th>Fruitful Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Functions that <strong>do</strong> something</td>
<td>• Functions that give a <strong>value</strong></td>
</tr>
<tr>
<td>• Call them as a <strong>statement</strong></td>
<td>• Call them in an <strong>expression</strong></td>
</tr>
<tr>
<td>• Example: <code>greet('Walker')</code></td>
<td>• Example: <code>x = round(2.56,1)</code></td>
</tr>
</tbody>
</table>

### Historical Aside

- Historically “function” = “fruitful function”
- But now we use “function” to refer to both
# Print vs. Return

<table>
<thead>
<tr>
<th><strong>Print</strong></th>
<th><strong>Return</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>• Displays a value on screen</td>
<td>• Defines a function’s value</td>
</tr>
<tr>
<td>▪ Used primarily for <strong>testing</strong></td>
<td>▪ Important for <strong>calculations</strong></td>
</tr>
<tr>
<td>▪ Not useful for calculations</td>
<td>▪ But does not display anything</td>
</tr>
</tbody>
</table>

```python
def print_plus(n):
    print(n+1)

>>> x = print_plus(2)
3
```

```python
def return_plus(n):
    return n+1

>>> x = return_plus(2)
>>> 9
```
## Print vs. Return

<table>
<thead>
<tr>
<th>Print</th>
<th>Return</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Displays a value on screen</td>
<td></td>
</tr>
<tr>
<td>▪ Used primarily for <strong>testing</strong></td>
<td></td>
</tr>
<tr>
<td>▪ Not useful for calculations</td>
<td></td>
</tr>
</tbody>
</table>

```python
def print_plus(n):
    print(n+1)

>>> x = print_plus(2)
3

>>> Nothing here!
```

```python
def return_plus(n):
    return n+1

>>> x = return_plus(2)
>>> x
3
```

• Defines a function’s value  
  ▪ Important for **calculations**  
  ▪ But does not display anything
Advanced String Features: Method Calls

- Methods calls are unique (right now) to strings
- Like a function call with a “string in front”
  - Usage: `string.method(x,y...)`
  - The string is an *implicit argument*
- Example: `upper()`
  - `s = 'Hello World'
  - `s.upper() == 'HELLO WORLD'
  - `s[1:5].upper() == 'ELLO'
  - `'abc'.upper() == 'ABC'

Will see why we do it this way later in course
Examples of String Methods

- $s_1$.index($s_2$)
  - Position of the first instance of $s_2$ in $s_1$

- $s_1$.count($s_2$)
  - Number of times $s_2$ appears inside of $s_1$

- $s$.strip()
  - A copy of $s$ with white-space removed at ends

- $s = 'abracadabra'$
  - $s$.index('a') == 0
  - $s$.index('rac') == 2
  - $s$.count('a') == 5
  - $s$.count('b') == 2
  - $s$.count('x') == 2
  - ' a b '.strip() == 'a b'

See Python Docs for more
def firstparens(text):
    
    """Returns: substring in ()
    Uses the first set of parens
    Param text: a string with ()""

    # Find the open parenthesis
    start = s.index('(')
    # Store part AFTER paren
    tail = s[start+1:]
    # Find the close parenthesis
    end = tail.index(')')
    # Return the result
    return tail[:end]

>>> s = 'Prof (Walker) White'
>>> firstparens(s)
'Walker'

>>> t = '(A) B (C) D'
>>> firstparens(t)
'A'
def second(thelist):
    """Returns: second elt in thelist
    The list is a sequence of words separated by commas, spaces.
    Ex: second('A, B, C') => 'B'
    Param thelist: a list of words""
    start = thelist.index(',,')
    tail = thelist[start+1:]
    end = tail.index(',,')
    result = tail[:end]
    return result

>>> second('cat, dog, mouse, lion')
'dog'
>>> second('apple, pear, banana')
'pear'
def second(thelist):
    """Returns: second elt in thelist
    The list is a sequence of words separated by commas, spaces.
    Ex: second('A, B, C') => 'B'
    Param thelist: a list of words""

    start = thelist.index(',',$)
    tail = thelist[start+1:]
    end = tail.index(',',')
    result = tail[:end]
    return result

>>> second('cat, dog, mouse, lion')
'dog'

>>> second('apple, pear, banana')
'pear'

Where is the error?

A: Line 1
B: Line 2
C: Line 3
D: Line 4
E: There is no error
**String Extraction Puzzle**

```python
def second(thelist):
    '''Returns: second elt in thelist
    The list is a sequence of words separated by commas, spaces.
    Ex: second('A, B, C') => 'B'
    Param thelist: a list of words'''

    start = thelist.index(',
    tail = thelist[start+1:]
    end = tail.index(',
    result = tail[:end]
    return result
```

>>> second('cat, dog, mouse, lion')
'dog'

>>> second('apple, pear, banana')
'pear'

9/8/17  Strings 25